

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

185,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Using Ontologies in Autonomous Robots Engineering

Esther Aguado and Ricardo Sanz

Abstract

The construction and operation of autonomous robots is heavily based of systemic conceptualizations of the reality constituted by the robot, its controller and the environment where it performs. In this chapter we address the role that computer ontologies play in the whole life cycle—engineering and operation—of autonomous robots: from its conception and construction by human engineering teams to deployment and autonomous operation in dynamic and uncertain environments. This chapter summarizes the state of the art, gives some examples and establishes a roadmap for future activity in this domain to produce shareable ontologies that could streamline autonomous robot development and exploitation.

Keywords: Robotics, Engineering, Ontology, Autonomy, Adaptation

1. Introduction

Technical systems are designed and built to perform a variety of operations in pursue of user needs. In many cases we want these systems to operate without human intervention for performance, cost or safety reasons. We want many technical systems to be autonomous. From fridge thermostats to the country-wide electrical utilities we expect 100% availability even in changing circumstances.

Autonomy requires the ability to perform the assigned task without external help. The autonomous car shall be able to negotiate an intersection and the autonomous space probe shall be capable of reorienting itself. But autonomy without robustness is a no go. Real autonomy also requires capabilities for enduring disturbances during operation. Conventional control systems are built to overcome some forms of disturbance—up to a limit. Autonomy shall be robust to operate in a wider range of circumstances.

In many cases, these systems need to be endowed with autonomy features to be able to operate in unstructured and hazardous environments. Many real-world situations show high levels of disturbance and uncertainty that displace the system from the normal operating region for which it was designed. In other cases, the disturbances come from inside the system. Electrical interference or device faults can lead the system to mission-level failure.

In all these situations—when the system is pushed out of the designed region of operation—the system requires certain adaptation capabilities to provide the levels of robustness and resilience necessary to overcome the adverse situation and ensure mission fulfillment.

As said, these disturbances may come from outside the system, such as changes in the environmental conditions, *e.g.* amount of obstacles, terrain characteristics,

etc., or from the system itself, such as failures in individual components used to perform a specific task or providing communication mechanisms. To overcome localized failures, autonomous systems require, when deployed, a certain level of redundancy to enable the use of fault-handling techniques to overcome these disturbances. These redundancies can be structural—e.g, when having spare components or using triple modular redundancy—or functional—*e.g.* when having different ways of reducing car speed.

In any case, all these mechanisms require ways of representing the knowledge about them both at design time—by engineers building the system—and at run-time—by the autonomous system itself. Engineers capture this knowledge in the engineering models and autonomous systems may use it in knowledge-driven perception-decision-action loops.

In this chapter, we address the use of ontologies as substratal assets for these knowledge based processes. Ontologies can be used to decouple those conceptual elements for system adaptation from the particular implementation used in a concrete deployed system. Decoupling design knowledge and realization promotes reusability, modularity, and scalability. All of them, critical properties of sound engineering processes. Ontologies can provide a shared understanding of all stages of the system life cycle—from conceptualization to decommission—to both ease the task of the engineer and improve system run-time operation. In this chapter, we specifically focus on the benefits of using ontologies in autonomous systems, especially in autonomous robots, and present an implementation case with adaptive mobile robots.

The chapter is organized as follows: Section 2 defines what is an ontology and collects well-known ontologies for autonomous robots. Section 3 defines what is autonomy and other the key concepts we aim to reach in this type of systems. Section 4 presents the scope of ontologies within the life cycle of autonomous systems. Section 5 addresses a concrete proof-of-concept of ontologies for augmenting the autonomy level in a mobile robot. Section 6 discusses the general implications at system level when using ontologies. Section 7 presents a roadmap for the use of ontologies to streamline autonomous robot development and exploitation. Lastly, Section 8 presents the conclusions of the chapter.

2. Ontologies for autonomous robots

Ontology—with upper case—is the branch of philosophy dedicated to the study of being. From this perspective of analyzing what exists, derives the use of ontologies—with lower case—in computer and information science. Computer ontologies are specifications of conceptualizations [1]. They formally document the types of entities that exist in a domain, their properties, and the relationships between them¹. A conceptualization is an abstract, simplified perspective in some area of interest. The conceptualization includes the objects, concepts, and other entities and the relationships among them. Ontologies are used in information systems to guarantee the conformance of a knowledge base with a certain conceptual specification.

Ontologies are built on top of terms that are used to capture the concepts. They provide a formal naming and definition of categories, properties, and relations between concepts, data, and entities. In practice, ontologies are just computer-readable files in a specific computer language that reify a conceptualization of elements of a specific domain.

¹ To be precise, they formally document *the information about* the types of entities, their properties, and the relationships between them.

In general, ontologies define a common vocabulary for a domain, allowing the reuse of domain-specific knowledge. Moreover, it provides a common understanding about a field for both people and artificial agents. It is this shareable knowledge among humans and software entities that makes ontologies a valuable asset in autonomous robot engineering. In [2] the authors remark the importance of formally represented knowledge and its foundation in *conceptualizations* that are shared among people. In our case these conceptualizations will be shared among humans—engineers—and intelligent machines—robots. These concepts are widely used in computational operations nowadays. For instance, in [3], the authors use an ontology-based method to assign tasks to a satellite cluster; in [4] they use ontology-aided reconfiguration to manage an IT infrastructure; and in robotics, works like the one presented in [5], make use of ontologies to define an information model to integrate the knowledge of heterogeneous fleets of robots in underwater operations.

2.1 Ontologies in robotics

The IEEE 1872-2015 standard [6] was published in early 2015. This is a standard for the robotics and automation domain that defines a set of ontologies for robots. The most popular of these ontologies is the core ontology for robotics and automation (CORA) which specifies the most general concepts in the robotics domain [7].

This standard is based on the Suggested Upper Merged Ontology (SUMO) [8]. SUMO—a top-level ontology—is used to provide differentiation among terms that refer to physical and abstract entities and serves as the basis of the robotics ontology. In SUMO, classes such as *agent*, *process* or *proposition* are defined. This knowledge is then specialized by CORA with classes such as *Robot* or *Robotic System*. Other ontologies defined in this standard are the position (POS) ontology [9] that captures the main concepts and relations regarding the position, orientation, and pose which are key elements for robot navigation and manipulation, and the RPARTS ontology that provides a set of specific types of roles that specialize the general role of robot parts.

The IEEE 1872 ontologies are explicit and formal, however, they are maybe too general for practical use. For this reason, there are a variety of more specific ontologies with different scopes in the robotics domain. Olszewska [10] presents an autonomous robot architecture ontology (ROA). ROA defines the main concepts and relations for defining a robot architecture. This ontology has been tested in driving a human-robot interaction scenario. A human operator specifies a task and this task is divided and associated with a robot according to its capabilities. All these types of entities are captured in the ontology and used to create information elements that enable human-robot communication. In [11] an extension of CORA is presented with concepts of design, environment, and interaction on artificial systems.

Most of these ontologies reuse knowledge from other ontologies. Sometimes ontologies are identified with knowledge bases, especially after the use of standardized languages to capture both generic and specific knowledge. In order to facilitate reuse, ontologies are designed following the principle of modularisation. Modularity provides a series of benefits in contrast to the problems of using big, complex, monolithic ontologies [12]:

- Scalability for querying data and reasoning. Small-scale ontologies allow easier concept assertion and reasoning than when handling a large number of entities.
- Scalability for update and maintenance. Ontologies, as any other artifact requires maintenance and may need to be updated and enlarged with new knowledge. This task is easier if the knowledge is structured in modules.

- Complexity management. The design process is more straightforward when working with small modules—*i.e.* a reduced set of concepts—integrated into the final ontology.
- Understandability. It is easier to understand an ontology in small portions than a huge ontology, either in a textual or visual form.
- Context-awareness. The use of modular ontologies simplifies the contextualization in the creation of knowledge. Each module can focus on any aspect regarding one context.
- Reusability. The split of an ontology into modules provides the reuse of specific parts in other ontologies.

Nevertheless, modular ontologies may lead to some problems during the process of creation and use. For example, the integration of other concepts by importing existing ontologies may lead to unexpected consequences such as inconsistencies related to reused vocabulary—*e.g.* conflicting definitions for homonyms. Keeping safety and correctness in a modular ontology is a key element when extracting or importing knowledge among ontologies and modules.

2.2 Standard ontologies beyond CORA

As said, CORA is a quite general standard that is not very effective for concrete applications. The above-mentioned ontologies by Olszewska or Fiorini try to provide more specific ontologies directly usable in concrete applications.

This fact is recognized by the Standards Association of the IEEE who is currently developing a collection of CORA-based standards to address different aspects of the robotics domain. These standards address different aspects of relevance like task specification, autonomy, ethics, agility, or verification of autonomous behavior.

In particular, the Robotics and Automation Society Standing Committee on standards working group 1872.2² is elaborating a CORA-based standard ontology for autonomous robotics [13]—the Autonomous Robotics (AuR) Ontology.

The AuR standard under development shall extend the CORA ontology by defining additional ontologies for the autonomous robots domain. These ontologies will address different aspects of relevance: (i) general concepts for autonomous robots; (ii) core design patterns specific to autonomous robot systems; and (iii) general use cases and/or case studies.

2.3 Ontologies and model-driven engineering

Currently, there are no generally accepted method or framework for the design of complex robotic systems [14]. However, this task of building complex robotic systems can easily leverage extant systems and software development methods. In complex system developments, the design is focused at a different level of abstractions, and modularity is used to both organize the design and implement the system [15]. Examples of this modular approach are the developments based on object-oriented methods, middleware, and component-based design.

The structural, modular organization of design knowledge and the exploitation of formally captured system knowledge is the basement of the large collection of

² https://standards.ieee.org/project/1872_2.html

model-based approaches in model-driven engineering (MDE). For example, OMG's Model-driven Architecture (MDA) focuses on design models a level of abstraction up of objects and components to reach modular reusable abstractions that can be later particularised for specific uses (Platform-Independent Model (PIM) → Platform-Specific Models (PSM)). In the system's domain—closer to the robotics domain—languages like Systems Modeling Language (SysML) are gaining momentum due to their universality as a vehicle for augmented design formalization. However, MDE methods often suffer from a lack of semantics and truly formal knowledge representations that can be effectively exercised [16].

To overcome these bottlenecks, a formal ontology can be included as part of the model definition. One example of a model that makes use of ontologies to specify the system behavior and architecture is the Teleological and Ontological Model for Autonomous Systems (TOMASys) framework [17]. TOMASys is a domain-independent metamodel that allows the construction of models to define architectural alternatives in component-based systems. This metamodel is teleological because it incorporates core concepts in the engineering conceptualization as are the concepts of system intention and the purpose of the designers when creating a specific subsystem. And it is ontological because it defines a formal vocabulary for systems structure and behavior.

The ontological approach followed in this chapter and presented in the proof-of-concept in Section 5 is built upon the TOMASys framework that was designed following the ideas of model-based systems engineering and particularized for the autonomous system engineering domain.

3. Autonomy and relate

The work described in this article addresses the use of ontologies for the augmentation of autonomy in robots [18]. As ontologies foster the use of formality in conceptualizations, it seems natural to try to provide a definition of the adjective *autonomous*.

The term “autonomous” is a buzzword these days and has received different meanings in different contexts³. In the analysis of the use of the term “autonomous” in automatic control and robotics, there are two major generalized uses of the term “autonomous”:

- A robot is said to be autonomous if it has the capability of *moving* by its own resources and under self-control.
- A robot is said to be autonomous if it has the capability of performing certain tasks without human—or external—help. A task-generalization of the former.

In our position as autonomous *systems* engineers, it is the second interpretation that we focus on. In systems engineering the task to be performed by the system is always something of value to the final user. An *useful* mobile robot shall not just wander around but perform some task of value during this wandering (find an object, move an object, detect intruders, *etc.*). When we say that a robot is autonomous we mean that it is capable of performing its assigned activities—*e.g.* generate a map—without the need of external intervention [19]. This also applies to the

³ It has indeed a long tradition of use in the domains of healthcare and political science.

capability of movement—including the whole robot navigation infrastructure—and to all the other functions that the robot may perform subsidiarily to the main task [20].

3.1 Autonomy and disruption

A second aspect concerning task execution that is of maximal importance is the distinction between (i) being able to perform certain tasks alone (e.g. moving to a pose or building a map); (ii) doing so while handling some degree of disturbance; and (iii) being able to perform these tasks *alone* in the presence of *severe disturbances*⁴. In the first case, a simple automaton can do the job. In the second case, a feedback control system can do the job. In the third case, a perception-thought-action loop is necessary to provide both feedback, adaptation, and anticipation. Some people use the term “automatic” for the first or second cases, keeping “autonomous” for the third. In the automatic control domain, some authors may use “open loop” and “closed-loop” to make this distinction, but for us, the second case also includes closed-loop controllers for operational set-points.

A more thorough distinction could be done concerning the nature of the disturbances, especially when severe. In the case of anticipated, well known severe disturbances, the system could be built in accordance to them to be able to respond adequately and predictably. If the disturbances are not predictable—or don’t want to bother about their anticipation—the system can be built to respond reactively to them. In the design of the system, we shall define, however, a set of bounds of the system operational environment to be able to design the system to behave robustly in this region.

In the work described in this chapter, we address situations where the system finds itself outside the boundaries set for its operation at design time—its normal operational profile. In these circumstances, the only possibility for keeping the mission going is for the robot to adapt to the new situation: it shall change its very design/realization to be able to still achieve mission objectives in this new situation.

3.2 Autonomy and trustworthiness

Trustworthiness is a necessary but not sufficient condition to carry out tasks in open environments [23]. In real operation, autonomous systems are deployed in complex environments plagued with uncertainty. This affects the system capability to complete the mission assigned to it by the user. For a user to confidently rely on an autonomous system, the system shall be trustworthy.

Trust and trustworthiness may seem similar but they must be distinguished; especially in an autonomous system, where behavior assurance is quite more complex. Trust is a human-system relational property; *i.e.* something that the human user perceives or *feels about* the robot. On the contrary, trustworthiness is a property of the system itself, *i.e.* that the system is robust and resilient in relation to its mission and hence, it deserves trust by the human user [24]. This implies that a human user may not trust a trustworthy system [25] because user perception is

⁴ A severe disturbance is a disturbance that violates the system design assumptions for normal operational conditions. An example of severe external disturbance is a slippery floor for an unmanned ground vehicle (UGV) when designed to operate on a non-slippery floor. An example of severe internal disturbance is the failure of a laser range sensor used in robot navigation. See [21] for a discussion of types of system change under the Klir general systems framework [22].

affected by limited knowledge, observation capability, and biased by previous experiences.

The problem we are addressing here is achieving trustworthiness, specifically dependability and mission assurance. The framework discussed here provides engineering tools in terms of system and mission concepts and relationships to define system design alternatives to deal with abnormal scenarios and unpredictable environments.

The underlying idea is to break the design/operation barrier. Using ontologies we can make available the engineering design knowledge at run-time to allow system self-reconfiguration using self-knowledge. With this approach, the scope of the ontologies covers from the system conceptualization until the system deployment. The use of ontologies at run-time provides an information-driven adaptation capability to enhance system autonomy [18].

4. Ontologies in the life cycle of autonomous systems

In systems engineering, the life cycle of an artifact usually includes eight stages: (i) identify the needs, (ii) define the system concept, (iii) specify system requirements, (iv) design the system, (v) implement the system, (vi) verify the system, (vii) deploy the system and (viii) operate it⁵. In the first six stages, the work is typically iterative until the deployment phase, when requirements and design decisions are frozen and remain implicit in the final artifact.

In fault-tolerant systems, a set of methods and algorithms intervene at run-time to keep the functional activity of the system, *i.e.* to maintain the operation as it was designed. The fault-tolerance mechanics is predefined, blind, and triggered by certain events. There is no system knowledge to reason about but its reification in rigid adaptation mechanisms. The idea we pursue in this work is the usage of ontologies to include the knowledge of engineering as part of the run-time system to endow the system with *flexible reconfiguration capability based on system knowledge*. With this approach, the design phase and the deployed phase maintain an explicit link through the system knowledge because the system ontology provides a metamodel that spans the whole system life cycle. This link can be exploited to combine other subsystems and create new designs at run-time more suitable for addressing certain contingencies.

Ideally, the system knowledge base should encode all include all the system concepts developed in early phases of the system life cycle, for example, *user needs* as the artifact is produced to satisfy the needs defined in the first stage. With this information, the system could be able to ensure the mission and reason about it at any stage.

In adaptive systems, with component or functional redundancy, the early stages of the life cycle are not addressed. The reconfiguration in this case aims to comply with the initial system design or a few designs for possible known contingencies.

However, by providing the system with capabilities to trace until the needs that justify its existence as well as the requirements that justify that design, the system can augment its autonomy in search of trustworthiness. If a requirement is imposed by a component that is not functioning and is going to be substituted with another element, that requirement is no longer applicable to the system. Therefore, besides the component in use, other adjustments can be made in the system for better performance.

⁵ A final decommissioning stage is also of importance, esp. in terms of sustainability, for real-world systems. We do not address this stage here.

An example of this case is the use of different navigation sensors in a mobile robot. Suppose we have an autonomous robot with laser and ultrasound sensors to navigate. An initial objective may be to reach a point as fast as possible. According to the final design of the robot, that requirement would be specified with a specification of a targeted velocity value.

The laser is a device with a high refresh rate so the robot can navigate safely at higher velocities. If the robot enters a room with glass walls, the laser is not reliable. If the robot detects through the reasoning that the environmental conditions are not suitable for the laser and triggers a reconfiguration to use the ultrasound sensor, the robot can keep its operation to fulfill the mission. However, as the design has changed, the requirements that can be fulfilled are not the same. In this case, as the ultrasound sensor has a shorter range, the maximum velocity of the robot shall be significantly less to keep a safe operational profile. Once the robot has traversed that glass room, the laser can be re-activated so the requirements must change again to achieve the maximum performance available.

This is a naive example of how a system engineering knowledge base can improve a navigation task. However, real-world missions are composed of complex-orchestrated tasks, for instance, the operation of a waiter-robot which must serve a drink, or a miner-robot that must obtain a certain mineral. In this case, that knowledge can be further exploited with deep reasoning to perform adaptation at different tasks and several stages of the system life cycle.

5. Fault-tolerant mobile robot proof-of-concept

Following the naive example above, an ontology-driven reconfiguration capability for mobile robots has been explored in the Metacontrol for Robot Operating System (MROS) project⁶. In this implementation, the robot's mission is to move to a certain point. During the mission, several contingencies may occur. In this case, we contemplate the internal contingency cases of (i) laser rangefinder failure and (ii) low battery. Additionally, the mission has some operational requirements associated in terms of performance, safety, and energy consumption that the robot must ensure during the navigation.

The assurance of the operational requirements along with the contingency handling is governed by a knowledge base structured on description logics and exerted by a reasoner. The key of this approach is the usage of the general modeling framework presented in Section 2.3, TOMASys. The TOMASys ontology is particularized with two sets of individuals: the navigation-domain ontology and an application-specific ontology. We use a modular approach in the construction of the ontology to be capable of reusing a part of the non-specific knowledge in any navigation application in mobile robotics. The ontologies are instantiated for run-time use as a knowledge base composed of three OWL 2 [26] files.

The TOMASys metamodel is used to depict structure and behavior with an explicit representation of the objectives of the system as well as the components required to realize them. The system concepts provided by this metamodel are divided into two main groups:

- The static knowledge is stored in Functions and Function Designs. The Function element allows the definition of abstract Objectives for the system to complete the mission. The Function Design element stores all the design

⁶ <https://robmosys.eu/mros/>

alternatives the system engineer has thought as possible to fulfill a certain Function.

- The instantaneous state is captured with Objectives, which define a set of operational requirements pursued at run-time when executing a Function; Function Groundings, that are used at run-time to specify which Function Design is in use; and Components, used to describe the structural modules at that instant. Lastly, Quality Attributes affect both static and run-time knowledge. They are used to make explicit the operational requirements of the mission.
- Each Objective has a Quality Attribute associated to meet operational requirements such as safety, performance, and energy consumption. Likewise, each Function Design has a Quality Attribute value estimation to select the best design alternative to meet the mission requirements. Additionally, the Function Grounding measures the real Quality Attribute value to monitor if those requirements are being fulfilled.

As it was previously mentioned, the knowledge base is completed with two sets of individuals. The navigation-domain file contains instances of widely-used navigation sensors such as ultrasound, laser, RGBD cameras, etc., and other important elements in autonomous robots such as the battery. These elements are instances of the TOMASys class Component. Besides, popular Quality Attributes are defined such as energy, safety, and performance.

The application-specific knowledge base is made of all the Function Designs, these are the design alternatives to perform navigation. Other elements are the instance of an Objective, the instance of a Function Grounding, this is the Function Design in use, and the Quality Attributes relative to them. Each Function Design has a Quality Attribute estimation in safety and energy, which is calculated for the Function Grounding. This calculated Quality Attribute value is compared with the non-functional requirements (NFR) defined for the Objective. The NFRs are the Quality Attributes required for the specific mission.

5.1 Run-time reconfiguration for fault-tolerance

To use the knowledge base at run-time, it is written in a machine-readable format using the Web Ontology Language (OWL). A descriptive logic (DL) reasoner uses it during the system operation to evaluate the robot's functioning. Once an Objective is defined, and it is linked to the Function that solves it, a Function Grounding is selected according to the mission requirements and the Component availability. In the MROS proof-of-concept, two possible classes of contingencies are addressed: component fault and mission requirements non-fulfillment.

Each Component has a *required by* relationship with the Function Design that makes use of it. If a Component is malfunctioning, those Function Designs that use it becomes unavailable. **Figure 1** depicts the main relationships contained in the knowledge base. The two components considered, laser and battery, are required for all Function Designs except one. In case of laser failure, the Function Design degraded mode should be selected. Likewise, in the case of a low battery, the Function Design energy saving mode should be selected. This is implicitly shown in the figure, as there are no links between those Function Design individuals and the corresponding Component individual.

The ontology also includes some rules using the Semantic Web Rule Language (SWRL) to perform functional diagnosis. This is done by asserting the information

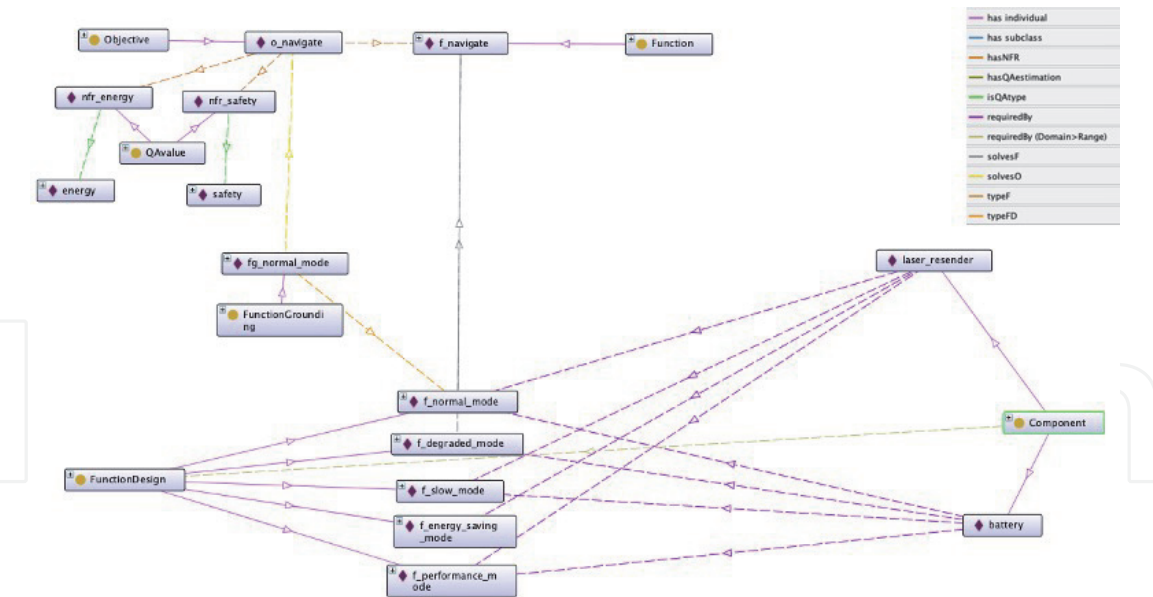


Figure 1. Main individuals and relationships of the proof-of-concept knowledge base. The Objective *o_navigate* is fulfilled by the Function Grounding *fg_normal_mode*, this Function Grounding is a realization of the Function Design *f_normal_mode* which solves the Function *f_navigate*. This Function is required as is the one that solves the Objective. Component required for this Function Design are laser and battery. Among all the possible Function Design, the one that does not require the laser if it becomes unavailable is *f_degraded_mode*. Additionally, Quality Attributes values relative to the non-functional requirements (NFRs) of the Objective are depicted for the Quality Attributes of safety and energy.

about the status of the Components that compose the system, the design in use to solve a function (Function Grounding), and the status of the Objective.

There are three sets of rules that: (i) set the Objective in error if the objective requirements, NFR Quality Attributes, are not met; (ii) set an Function Grounding in error if a Component in use is in error; and (iii) propagate Function Grounding error to the Objective. Lastly, there are some additional rules regarding the storage in a log file of the Function Grounding that have been in error and the status of a Function Design realisability depending on the status of the Components. For instance, if the laser is in error, the only Function Design with realisability with a true value will be *f_degraded_mode* according to **Figure 1**.

Table 1 shows three example SWRL rules used in this proof-of-concept.

Rule no.1 <i>tomasys:Component(?c) tomasys:c_status(?c, false) mros:requiredBy(?c, ?fd) tomasys:typeFD(?fg, ?fd) tomasys:FunctionGrounding(?fg) → tomasys:fg_status(?fg, INTERNAL_ERROR)</i>
If a Component has a status in false (in error), and that component is <i>required by</i> a Function Design with the same <i>type</i> as the Function Design in use, Function Grounding, then that Function Grounding status is set as <i>INTERNAL ERROR</i> .
Rule no.2 <i>tomasys:FunctionGrounding(?fg) tomasys:fg_status(?fg, INTERNAL_ERROR) tomasys:solvesO(?fg, ?o) tomasys:Objective(?o) → tomasys:o_status(?o, INTERNAL_ERROR)</i>
If a Function Grounding has a status in <i>INTERNAL ERROR</i> , and that Function Grounding <i>solves</i> an Objective, then that Objective status is set as <i>INTERNAL ERROR</i> .
Rule no.3 <i>tomasys:Component(?c) tomasys:c_status(?c, false) mros:requiredBy(?c, ?fd) → tomasys:fd_realisability(?fd, false)</i>
If a Component has a status in false (in error), and that Component is <i>required by</i> a Function Design then that <i>realisability</i> is set to <i>false</i> .

Table 1. SWRL rules for proof-of-concept implementation. The first one sets the Function Grounding in error if it uses a faulty Component, the second one sets the Objective in error if the Function Grounding is in error and the third one marks as unreachable the Function Design that require unavailable Components.

When the system needs adaptation because the mission (Objective) is in error according to rule no. 2, a selection of a design alternative is required. In this case, rule no. 3 is applied to determine the solution available in terms of components. An equivalent rule in terms of Quality Attributes and NFRs is used to select the design compliant with the mission requirements. If there are several Function Designs available, the module in charge of the reconfiguration, called Metacontroller, selects the Function Design with higher estimated performance.

In this case, each Function Design represent a system mode. For instance, the normal mode uses the laser to navigate at maximum velocity levels in environments with few obstacles but not crowded. The degraded mode, uses an RGBD camera instead of a laser to navigate, as the refresh rate of this device is considerably less than the laser, the velocity is reduced to keep navigation with safety. By contrast, the energy-saving mode is a very safe and slow implementation to reduce at maximum the battery consumption, impacting the duration of the mission, and therefore, the performance.

The ontology implementation has been evaluated in a complete robotic application, a patrolling corridors mission. While the robot performs the patrol, contingencies such as the laser error described previously. The robot used is a TurtleBot2 composed of a Kobuki platform RPLidar A2 laser and an Orbec Astra RGBD camera. The experiments consisted of simulating a laser error at a random instant. These data were corrupted by generating realistic data as if something was blocking the laser, or if there was a misalignment, maybe due to a hit or a fall of the robot. This was done by publishing scan messages with erroneous data (a vector of 0's) in the gazebo plugin topic. **Figure 2** depicts the simulation used to develop the reasoner to implement the ontology-driven reconfiguration. The output from the reasoner once the laser is malfunctioning and the robot with the navigation mode to degraded is shown in **Figure 3**.

The main experiment carried measures the recovery time for a laser failure using ontology-driven reconfiguration. After 50 iterations of the experiment, the time required is 1.995 s with a standard deviation of 0.478. Without it, the estimated recovery time for this failure is about 300 s (indeed tied to system maintenance).

Furthermore, another testbed has been used to prove the ontology reusability along with the reconfiguration performance. In this case, we have used a simulation of an unmanned underwater vehicle performing exploration in a flooded mine [27].

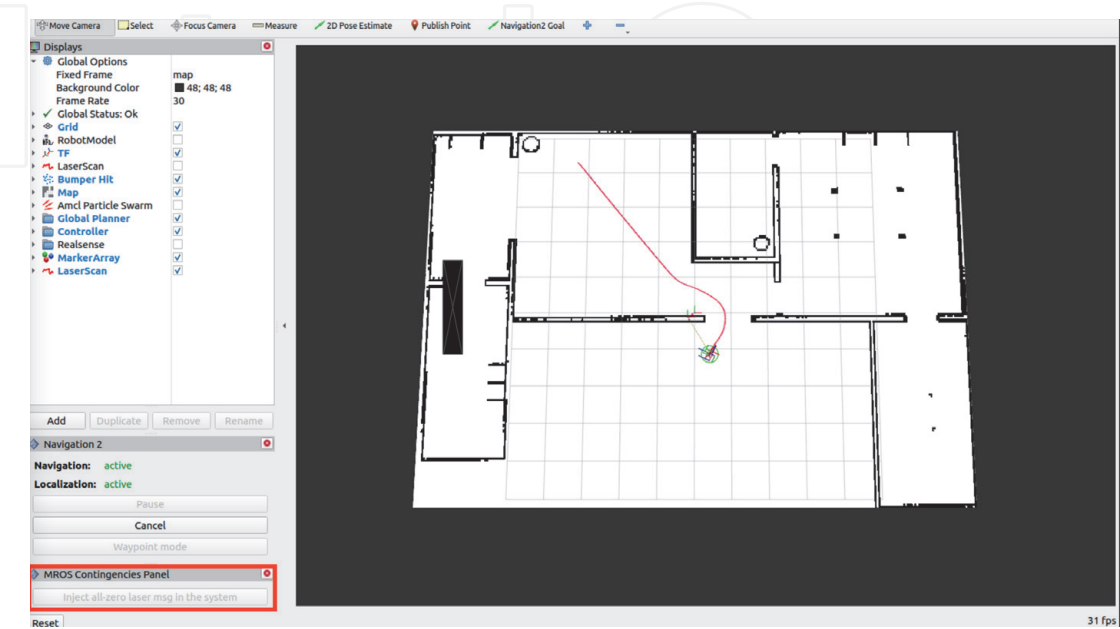


Figure 2.
Simulation of robot patrolling with navigation and localization system; in red, the button to inject laser failure.

```

esther@esther-OMEN: ~/mros_ws 100x27
[mros2_reasoner_node-1] [INFO] [1616066185.155250511] [mros2_reasoner_node]: >> Started MAPE-K **
EXECUTION **
[mros2_reasoner_node-1] [WARN] [1616066185.155491741] [mros2_reasoner_node]: New Configuration requ
ted: f_degraded_mode
[mros2_reasoner_node-1] [INFO] [1616066185.158567864] [mros2_reasoner_node]: Got Reconfiguration res
lt True
[mros2_reasoner_node-1] [INFO] [1616066185.159710372] [mros2_reasoner_node]: Exited timer_cb after s
ccessful reconfiguration - Obj set to None
[mros2_reasoner_node-1] [INFO] [1616066185.260128879] [mros2_reasoner_node]: QA value received! TYPE
energy
VALUE: 0.091024
[mros2_reasoner_node-1] WARNING:root: >>> Ontology Status <<<
[mros2_reasoner_node-1] WARNING:root:
Component Status: [['laser_resender', 'FALSE']]
[mros2_reasoner_node-1] WARNING:root:
FG: fg f_degraded_mode Status: None Solves: obj_navigate_161606614
FD: f_degraded_mode QValues: [['energy', 0.091024]]
[mros2_reasoner_node-1] WARNING:root:
OBJECTIVE: obj_navigate_161606614 Status: None NFRs: [['energy'
0.5), ('safety', 0.5)]
[mros2_reasoner_node-1] WARNING:root: >>>>>>>>>> <<<<<<<<<<<<
[mros2_reasoner_node-1] [INFO] [1616066186.568968799] [mros2_reasoner_node]: >> Started MAPE-K **
analysis (ontological reasoning) **
[mros2_reasoner_node-1] [INFO] [1616066187.219712102] [mros2_reasoner_node]: >> No Objectives in E
ROR: no adaptation is needed
[mros2_reasoner_node-1] [INFO] [1616066187.260064658] [mros2_reasoner_node]: QA value received! TYPE
energy
VALUE: 0.073761

```

Figure 3.

Console from the reasoner ROS2 node to implement the ontology-driven reconfiguration; in red the component status of the laser as 'FALSE', malfunctioning and the navigation mode required, the Function Grounding 'DEGRADED MODE'.

With this proof-of-concept, we have implemented a fault-adaptive subsystem in general terms to ensure mission requirements and face component faults. This approach provides a general and reusable asset for systems in which there are design alternatives in terms of behavior and/or components. This approach aptly realizes the vision of using ontologies for building knowledge bases to decouple the mission-oriented system operation core from the reconfiguration needed to overcome disturbances or failures. Moreover, with this experimental setting, we have shown evidence of the advantages of automatic reconfiguration through ontological architectures for reducing the recovery time for laser contingencies.

6. Systemic implications of ontologies

As stated in Section 2, ontologies provide a variety of tools to define a system in terms of concepts and the relationships among them. Besides, given their formal nature, they can be included in the system as an explicit source of knowledge to improve its run-time operation.

Ontologies can be treated as a sub-system itself, that may be designed following the systems engineering principles of modularity, scalability, and reusability. The proof-of-concept presented here (Section 5), is an example of the use of ontologies to augment the autonomy level of a robot, increasing its dependability by improving mission assurance. In this proof-of-concept system ontologies are components of the deployed system.

However, the use of ontologies as part of autonomous systems engineering processes goes well beyond this [18], because they can have a strong impact on the many processes of the systems life cycle [28]. In this section, we analyze three classes of impacts: (i) impacts on complexity; (ii) impacts on collaboration; and (iii) impacts on risk management.

6.1 Implications on complexity

Section 2 summarised some of the benefits of modularity in ontologies. The key feature of the modular approach is the reduction obtained in the ontology

complexity when working on small modules integrated into a whole ontology. This conceptual decomposition and complexity reduction can be extrapolated to the conceptualization that underlies all system engineering processes⁷, especially in systems-of-systems contexts, [29].

The use of a knowledge base that makes explicit the requirements and the design decisions on the system, provides systems engineering knowledge that can be leveraged at the whole life cycle. The explicitness of mission-oriented concepts makes it easier for the developer at the verification phase to understand possible fault sources and integration problems. The trend in systems engineering towards model-based approaches is rooted in the formal verification capability that models provide at all stages (esp. at early stages where the costs of re-engineering are much lower). At the early stages of the model, complexity is lower and formal analyses may be more exhaustive and effective.

In the deployment phase—as shown in our proof-of-concept—ontologies may be used as fault-tolerance assets. The use of ontologies to reason about the state of the mission and the architectural components in use in general terms, provide a common framework to decouple reconfiguration actions from the particular implementation. This separation of concerns allows for a strong reduction in the complexity of the fault-tolerant mechanisms by both (i) the localized nature of the fault-tolerant mechanisms; and (ii) the possibility of reusing general tested assets such as TOMASys and some of the more general knowledge bases (as the ones presented here⁸).

6.2 Implications on collaborative systems

The use of ontologies in the construction of formal knowledge bases provides a common understanding within a domain. The encoding of ontologies in machine-readable formats such as OWL allows a truthful integration when sharing information between different agents and/or tools. This integration obviously includes the possibility of collaboration between different types of systems in a group mission. This collaboration is not limited to the activities in the systems life cycle as described earlier but spans all classes of multi-agent collaboration in fielded systems.

An example of this is the enabling of collaborative work between fleets of robots, especially when they are heterogeneous. For example, the shared information between an unmanned aerial vehicle (UAV) and an unmanned ground vehicle (UGV) may encounter incompatibilities just in the coordinate systems they use, as the UGV does not take into account the vertical axis. Moreover, robots may have different capabilities, a UAV may be able to map the environment whilst the UGV may have an arm to interact with it.

The same occurs when the system includes a human as an external operator or supervisor, the system must exchange information to ensure collaborative work. Usually, the combination of data is done implicitly by the system designers. However, using ontologies for formal integration of all the different perspectives of a system of systems with an explicit *conceptualization* provides a robust and trustworthy method for sharing information that affects the way humans interact and collaborate with machines.

6.3 Implications on risk management

As said before, a formal definition of system concepts can be used for better formal analysis along the whole life cycle. This upstream analysis implies a

⁷ See for example <http://www.sebok.org>.

⁸ Available at: https://github.com/MROS-RobMoSys-ITP/mros_ontology

reduction of the probability of faults during system operation but it also includes the possibility of better diagnostic features. Having explicit knowledge on relationships among system entities allows the system to trace the source of faults and take action in the implicated parts.

Besides, the system may adapt itself as the case presented in Section 5 to reduce damage or can even take preventive action based on deep reasoning. When the system is able to adapt to overcome problems derived from environmental changes or malfunctioning components, the system becomes more autonomous and trustworthy.

Design decisions have always associated risks. Usually, the final design is commonly the solution with less risk in the context where the system is deployed. The use of ontologies provides a tool for risk management as design decisions may be justified in the knowledge base. Furthermore, this information can be used at run-time to apply the most suitable solutions if the operational environment changes.

In the proof-of-concept presented here, quality attributes are used to select among design alternatives. When risk augments because of not meeting the security standards of the mission, other designs can be used. This may affect the performance of the mission but ensures its fulfillment. The system engineer must coordinate adequately those quality attributes to ensure a trade-off between performance and security. In this context, risk ceases to be a collateral effect of system design and operation to become a first-level citizen in the explicit design of the system.

7. Towards a full life cycle ontology

Ontologies are commonly used to store information within a domain. For this reason, they are a valuable asset to model the shared understanding of the system and its concepts at different stages of its life cycle. Here, we propose to take a further step and use that information model at run-time to provide the system with knowledge-driven self-adaptation.

The proof-of-concept presented here addresses reconfiguration to ensure the mission fulfillment within a set of predefined operational requirements. The main limitation is imposed by the set of alternative designs predefined for the system and its possible contingencies. Here, ontologies have been used to orchestrate the deployment of different design solutions at run-time.

To increase the autonomy levels of systems, particularly in the case of robots, we propose to take a further step and include all early phases of system life cycles, from the need identification to fulfill a mission to the verification of the system (see **Figure 4**).

When the system is deployed, each time a contingency is detected and it requires reconfiguration, the system should return to the needs evaluation and analyze which tools it has available to satisfy the need. According to that, it can adopt some requirements and come across with a design by itself. That design may be tested in simulation with a digital twin or just deployed if it has been evaluated in previous operations.

To reach such an elevated autonomy level, a massive effort in ontology standardization in systems engineering is required⁹. Moreover, the design engineers must encode all the information they have about the system in terms of those

⁹ See for example the efforts of the IEEE 1872.1 working group on robot task standardization or the IOF group on systems engineering.

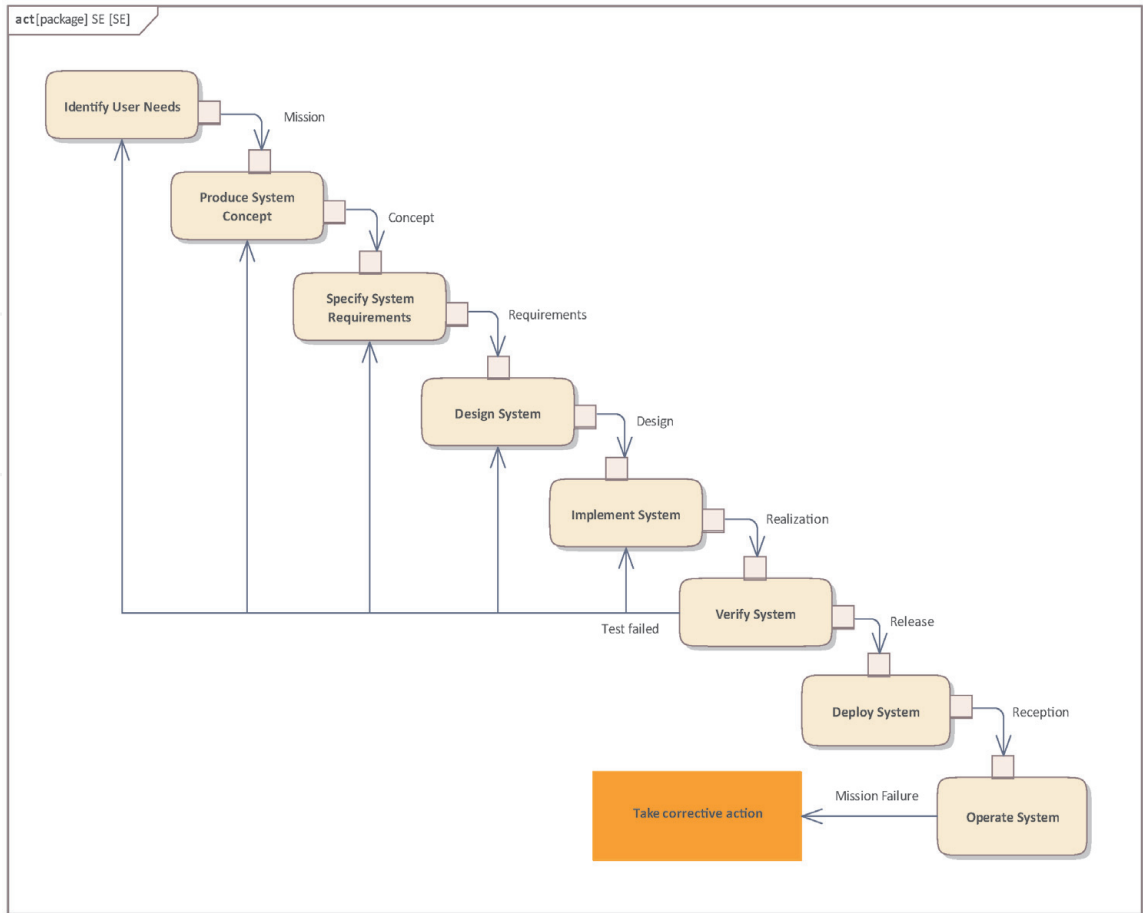


Figure 4.
Going back in the system life cycle from an operational failure.

standards. That information also includes discarded ideas and the reason why they are not included in the final design, as can become part of a contingency solution.

To select the best possible design available in case of contingency, knowledge bases should include metrics regarding the cost (in terms of time, energy consumption, reliability) of the resulting system after applying reconfiguration. This selection must preserve the mission fulfillment with optimal features.

8. Conclusions

Ontologies provide a baseline for shared information between systems, subsystems, and external agents. But that information can also be a tool for augmenting the autonomy levels of systems. Ontologies provide a formal conceptualization of entities and their relationships. The knowledge stored can be used along architectural models in system engineering to provide a general framework for autonomous systems.

A concrete realization of this general framework has been tested with a mobile robot navigating to a point in a cluttered environment. The proof-of-concept address two contingency types, (i) a component-level failure, as the case when the laser becomes unavailable, or (ii) not reaching the mission requirements in terms of safety or energy consumption.

The contingencies are solved by reasoning about the status of the components in use and the status of the objective and its quality attributes associated. Once a contingency is encountered, the reasoner provides the most suitable design

alternative to overcome it according to the component availability and the estimated quality values in terms of energy consumption, safety and performance.

However, this proof-of-concept focus on the selection of alternatives. To provide a complete module of self-adaptation and reconfiguration in the whole life cycle of systems, we need to take a larger perspective. To reach complete autonomy, the system needs to have access to knowledge from the needs for which it is designed, besides the design alternatives and the restrictions the engineers have faced. With this information, in case of failure, the system can have a wider picture of its context and take corrective action at different levels of its architecture to adapt to run-time situations.

To achieve this vision a *full system life cycle ontology for autonomous systems* is needed. Current efforts in ontology development for systems engineering, robots, and autonomous systems are quite valuable but they shall be (i) based on a general systems foundation; (ii) harmonized, and (iii) built with a modular ontology approach.

Acknowledgements

This work was supported by RobMoSys-ITP-MROS (Grant Agreement No. 732410) and ROBOMINERS (Grant Agreement No. 820971) projects with funding from the European Union's Horizon 2020 research and innovation programme.


Author details

Esther Aguado* and Ricardo Sanz

Autonomous Systems Laboratory, Centre for Automation and Robotics UPM-CSIC, Universidad Politécnica de Madrid, Spain

*Address all correspondence to: e.aguado@upm.es

IntechOpen

© 2021 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] Gruber TR. A translation approach to portable ontologies. *Knowledge Acquisition*. 1993;5(2):199–220.
- [2] Guarino N, Giaretta P. Ontologies and knowledge bases: Towards a terminological clarification. In: *Towards very Large Knowledge bases: Knowledge Building and Knowledge sharing*. IOS Press; 1995. p. 25–32.
- [3] Zhao X, Wang Z, Cui Y, Zheng G. Novel Ontology-Based Method for Generating Satellite Cluster's Task Configuration. *Journal of Aerospace Information Systems*. 2020;17(2):86–96.
- [4] Poggi F, Rossi D, Ciancarini P. Integrating Semantic Run-Time Models for Adaptive Software Systems. In: *Journal of Web Engineering*. vol. 18; 2019. p. 1–42.
- [5] Zhai Z, Martínez Ortega JF, Lucas Martínez N, Castillejo P. A Rule-Based Reasoner for Underwater Robots Using OWL and SWRL. *Sensors (Basel, Switzerland)*. 2018 10;18(10):3481.
- [6] IEEE Standard Ontologies for Robotics and Automation. *IEEE Std 1872-2015*. 2015:1–60.
- [7] Prestes E, Carbonera JL, Rama Fiorini S, M Jorge VA, Abel M, Madhavan R, et al. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*. 2013; 61(11):1193 – 1204. *Ubiquitous Robotics*.
- [8] Niles I, Pease A. Towards a Standard Upper Ontology. In: *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*. FOIS '01. New York, NY, USA: Association for Computing Machinery; 2001. p. 2–9.
- [9] Carbonera JL, Fiorini SR, Prestes E, Jorge VAM, Abel M, Madhavan R, et al. Defining positioning in a core ontology for robotics. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*; 2013. p. 1867–1872.
- [10] Olszewska JI, Barreto M, Bermejo-Alonso J, Carbonera J, Chibani A, Fiorini S, et al. Ontology for autonomous robotics. In: *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*; 2017. p. 189–194.
- [11] Fiorini SR, Carbonera JL, Gonçalves P, Jorge VAM, Rey VF, Haidegger T, et al. Extensions to the core ontology for robotics and automation. *Robotics and Computer-Integrated Manufacturing*. 2015;33: 3 – 11. Special Issue on Knowledge Driven Robotics and Manufacturing.
- [12] Parent C, Spaccapietra S. In: *Stuckenschmidt H, Parent C, Spaccapietra S, editors. An Overview of Modularity*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009. p. 5–23.
- [13] Olivares-Alarcos A, Beßler D, Khamis A, Gonçalves P, Habib MK, Bermejo J, et al. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*. 2019;34.
- [14] Ragavan SV, Ganapathy V. A General Telematics Framework for Autonomous Service Robots. In: *2007 IEEE International Conference on Automation Science and Engineering*; 2007. p. 609–614.
- [15] Bayat B, Bermejo J, Carbonera J, Facchinetti T, Fiorini S, Gonçalves P, et al. Requirements for building an ontology for autonomous robots. *Industrial Robot: An International Journal*. 2016 08;43.
- [16] Kleppe AG, Warmer J, Bast W. *MDA Explained: The Model Driven Architecture: Practice and Promise*.

USA: Addison-Wesley Longman Publishing Co., Inc.; 2003.

[17] Hernández C, Bermejo-Alonso J, Sanz R. A self-adaptation framework based on functional knowledge for augmented autonomy in robots. *Integrated Computer-Aided Engineering*. 2018;25:157–172.

[18] Sanz R, Bermejo J, Morago J, Hernández C. Ontologies as Backbone of Cognitive Systems Engineering. In: Bryson J, Vos MD, Padget J, editors. *Proceedings of AISB CAOS 2017: Cognition And Ontologies*. Bath, UK; 2017. p. 218–223.

[19] Sanz R, Matía F, Galán S. Fridges, Elephants and the Meaning of Autonomy and Intelligence. In: Groumpos PP, Koussoulas NT, Polycarpou M, editors. *IEEE International Symposium on Intelligent Control, ISIC'2000*. Patras, Greece; 2000. p. 217 – 222.

[20] López I, Sanz R, Hernández C, Hernando A. General Autonomous Systems: The Principle of Minimal Structure. In: Grzech A, editor. *Proceedings of the 16th International Conference on Systems Science*. vol. 1; 2007. p. 198–203.

[21] López I. A Framework for Perception in Autonomous Systems [Ph.D. thesis]. Departamento de Automática, ETS de Ingenieros Industriales, Universidad Politécnica de Madrid; 2007.

[22] Klir GC. *An Approach to General Systems Theory*. Van Nostrand Reinhold; 1969.

[23] Abbass HA, Scholz J, Reid DJ. In: Abbass HA, Scholz J, Reid DJ, editors. *Foundations of Trusted Autonomy: An Introduction*. Cham: Springer International Publishing; 2018. p. 1–12.

[24] Amaral G, Guizzardi G, Guizzardi R, Mylopoulos J. *Ontology-*

based Modeling and Analysis of Trustworthiness Requirements: Preliminary Results. In: Dobbie G, Frank U, Kappel G, Liddle SW, Mayr HC, editors. *International Conference on Conceptual Modeling (ER 2020)*. Vienna, Austria; 2020. p. 342–352.

[25] Kok B, Soh H. Trust in Robots: Challenges and Opportunities. *Current Robotics Reports*. 2020 12;1:1–13.

[26] W3C. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. World Wide Web Consortium; 2012. REC-owl2-syntax-20121211.

[27] Aguado E, Milosevic Z, Hernández C, Sanz R, Garzon M, Bozhinoski D, et al. Functional Self-Awareness and Metacontrol for Underwater Robot Autonomy. *Sensors*. 2021;21(4). Available from: <https://www.mdpi.com/1424-8220/21/4/1210>.

[28] ISO/IEC/IEEE. *ISO/IEC/IEEE 15288-2015 Systems and software engineering – System life cycle processes*. International Standards Organisation; 2015.

[29] Guariniello C, Raz AK, Fang Z, DeLaurentis D. System-of-systems tools and techniques for the analysis of cyber-physical systems. *Systems Engineering*. 2020;23(4):480–491.