



aslab

Systems that know what they are doing

A Model-Based Formal Specification
for Robust Autonomy

ASLAB-R-2024-007 / July 2024

Esther Aguado

Autonomous Systems Laboratory
aslab.upm.es

The Autonomous Systems Laboratory is a research organisation hosted by the Universidad Politécnica de Madrid, Spain. Requests for permission to reproduce this document or to prepare derivative works of this document should be addressed to the ASLab Licensing Agent.

Copyright © 2004-2024 by ASLab and Universidad Politécnica de Madrid.

NO WARRANTY — Please read the fine print:

THIS ASLAB MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. UNIVERSIDAD POLITECNICA DE MADRID NOR ANY OF THE INDIVIDUAL ASLAB RESEARCHERS MAKE NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, ACCURACY, COMPLETENESS, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. UNIVERSIDAD POLITECNICA DE MADRID DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

For information about other ASLab monographies, reports or articles, please visit the publications portion of our Web site:

<http://www.aslab.upm.es/public/>



The ASLab **ASys Project** is a long-term research framework focused in the development of universal technology for the construction of autonomous systems. What makes ASys different from other research projects in this field is the extreme ambitious objective of addressing all the broad domain of autonomy, not just mobile robots. We capture this purpose in the motto “**Engineering any-x autonomous systems**”. Get more info at:

<http://www.aslab.upm.es/projects/ASYS>

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Industriales



Systems that know what they are doing:
A Model-Based Formal Specification for Robust Autonomy

DOCTORAL THESIS

Submitted for the degree of Doctor by:

Esther Aguado González

Master in Industrial Engineering and
Master in Automation and Robotics

Madrid, 2024



UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Industriales

Doctoral Degree in Automatic Control and Robotics

**Systems that know what they are doing:
A Model-Based Formal Specification for Robust Autonomy**

DOCTORAL THESIS

Submitted for the degree of Doctor by:

Esther Aguado González

Master in Industrial Engineering and
Master in Automation and Robotics

Under the supervision of:

Dr. Ricardo Sanz (Supervisor)

Dr. Claudio Rossi (Co-supervisor)

Madrid, 2024

Title: Systems that know what they are doing: A Model-Based Formal Specification for Robust Autonomy

Author: Esther Aguado González

Doctoral Programme: Automatic Control and Robotics

Thesis Supervision:

Dr. Ricardo Sanz, Profesor Titular de Universidad, Universidad Politécnica de Madrid
(Supervisor)

Dr. Claudio Rossi, Catedrático de Universidad, Universidad Politécnica de Madrid
(Co-supervisor)

External Reviewers:

Thesis Defense Committee:

Thesis Defense Date:

This thesis has been partially supported by “UPM Programa Propio” grant for doctoral studies with international mention.

A mis abuelos, Carmen y Fidel

Agradecimientos

Han pasado más de cuatro años desde que comencé mis estudios de doctorado. Durante este tiempo, amigos y compañeros han contribuido, cada uno a su manera, en lo que soy y pienso; acompañándome de algún modo y poniendo su granito de arena en este viaje.

En primer lugar, me gustaría agradecer a Ricardo, mi tutor y director de tesis, por brindarme la oportunidad de embarcarme en este camino. Recuerdo claramente el momento en que me habló sobre el emocionante proyecto que estaba comenzando, en el que se abría un inmenso número de posibilidades. Debo confesar que al principio no estaba segura, pero su honestidad y empatía me hizo decidirme a perseguir esa curiosidad por la investigación que he tenido desde que tengo uso de razón. Ricardo siempre ha dado mucho más de lo que podría esperar de un tutor, más allá de la orientación y guía para desarrollar esta tesis, me ha dado nuevas perspectivas para introducirme en el mundo de la ciencia y la investigación. Sus sugerencias instándome a leer sobre distintos ámbitos ha perfilado mi manera de ver el mundo, haciéndome evolucionar no solo como investigadora sino como persona. Además, tengo que agradecer que, desde el primer día, me permitiera viajar, asistir a congresos y participar en proyectos; animándome también a seguir cuando el tema parecía demasiado complejo o fuera de nuestro dominio. Su generosidad es inconmensurable.

También me gustaría expresar mi gratitud a mi co-director, Claudio, por brindarme la primera oportunidad de participar en un proyecto europeo. Su firme orientación hacia la sencillez y la claridad ha sido de gran ayuda, animándome constantemente a fundamentar conceptos abstractos en aplicaciones prácticas. Las ideas innovadoras de Claudio han sido fundamentales para el desarrollo de esta tesis.

Debo agradecerle a Carlos Hernández que me haya permitido realizar una estancia internacional. Con él he tenido la oportunidad de colaborar en dos proyectos. Su contribución a esta tesis es innegable, al permitirme abordar su investigación desde otra perspectiva. Sin sus consejos y orientación, esta tesis no habría sido posible.

Mis compañeros de laboratorio, Andrés y Virgilio han desempeñado un papel fundamental durante estos años, compartiendo enfoques, alternativas e innumerables charlas de café. Asimismo, Williams quien nos aportaba experiencia y perspectiva. Quisiera agradecer también a los demás miembros del laboratorio, especialmente a Rosa, Esther y Carlos, que siempre están dispuestos a ayudar y facilitar cualquier aspecto que esté en su mano.

También quiero expresar mi gratitud hacia mis compañeros en los proyectos europeos MROS, ROBOMINERS y CORESENSE. En estos proyectos, he disfrutado de valiosas discusiones técnicas y perspectivas que han enriquecido enormemente mi investigación.

Por último, me gustaría agradecer a Radu Calinescu y Antonio Chella por revisar amablemente este trabajo y formar parte del tribunal. Sus comentarios han sido increíblemente útiles para su mejora. También quiero expresar mi profundo agradecimiento a Vicente Matellán, Paloma

de la Puente, Manuel Rodríguez, Francisco Martín Rico y Francisco Javier Rodríguez Lera por aceptar formar parte del comité de defensa.

Fuera del ámbito universitario, quiero expresar mi más profundo agradecimiento a todos mis amigos por su paciencia y comprensión. A pesar de mis ausencias repentinas y mis interminables conversaciones sobre el mismo tema, siempre han estado ahí para apoyarme. Quiero hacer una mención especial a Jeny, Eli, Loreto, Patri y Dani; mis amigos de siempre. También agradezco enormemente a Elena, Fernando, Alberto y Félix, con quienes empecé la carrera hace casi diez años. Con todos ellos he compartido innumerables aventuras, risas y momentos emocionantes. Gracias por hacerme sentir siempre tan acompañada y respaldada, esta tesis también tiene un trocito de vosotros.

Por último y más importante, quiero expresar mi más profundo agradecimiento a mi familia. Todo lo que he logrado se lo debo a mis padres, quienes siempre han dado todo lo que estaba en sus manos y me han enseñado los verdaderos valores de la bondad y el cariño. Quiero hacer una mención especial a mi abuela, a quien le gustaría saber que ha sido y siempre será mi guía espiritual; y a mi abuelo, quien día tras día demuestra su inquebrantable determinación y me ha inculcado el valor del trabajo y la sinceridad. También quiero reconocer la parte fundamental que ha tenido Daniel en mi vida. Él ha sido mi gran apoyo durante todos estos años, siendo el ancla que me ha proporcionado serenidad y calma cuando más lo necesitaba. Daniel, eres un verdadero ejemplo de comprensión y dedicación.

Acknowledgements

It has been more than four years since I started my doctoral studies. During this time, friends and colleagues have contributed, each in their own way, to who I am and what I think; they have helped me in some way and are doing their part in this journey.

First of all, I would like to thank Ricardo, my tutor and thesis director, for giving me the opportunity to embark on this journey. I clearly remember the moment when he told me about the exciting project I was starting, in which an immense number of possibilities were opening up. I must confess that at first I was unsure, but his honesty and empathy made me decide to pursue that curiosity for research that I have had for as long as I can remember. Ricardo has always given much more than what I could expect from a tutor, beyond the orientation and guidance to develop this thesis, he has given me new perspectives to introduce me to the world of science and research. His suggestions urging me to read about different fields have shaped my way of seeing the world, making me evolve not only as a researcher but also as a person. In addition, I have to thank him for allowing me to travel, attend conferences, and participate in projects from day one, encouraging me to continue when the topic seemed too complex or out of our domain. His generosity is invaluable.

I would also like to express my gratitude to my co-director, Claudio, for giving me my first opportunity to participate in a European project. His strong focus on simplicity and clarity has been a great help, constantly encouraging me to ground abstract concepts in practical applications. Claudio's innovative ideas have been fundamental to the development of this thesis.

I must thank Carlos Hernandez for allowing me to do an international stay. I have had the opportunity to collaborate with him on two projects. His contribution to this thesis is undeniable, allowing me to approach his research from another perspective. Without his advice and guidance, this thesis would not have been possible.

My lab colleagues, Andres and Virgilio, have played a fundamental role during these years, sharing approaches, alternatives and countless coffee chats. Also, to Williams who provided us with experience and perspective. I would also like to thank the other members of the laboratory, especially Rosa, Esther and Carlos, who are always willing to help and facilitate any aspect that is in their hands.

I would also like to express my gratitude to my colleagues in the European projects MROS, ROBOMINERS and CORESENSE. In these projects, I have enjoyed valuable technical discussions and perspectives that have greatly enriched my research.

Finally, I would like to thank Radu Calinescu and Antonio Chella for kindly reviewing this work and being part of the tribunal. Their comments have been extremely helpful in improving it. I would also like to express my deepest gratitude to Vicente Matellán, Paloma de la Puente, Manuel Rodríguez, Francisco Martín Rico and Francisco Javier Rodríguez Lera

for agreeing to be part of the defense committee.

Outside of the university environment, I would like to express my deepest gratitude to all of my friends for their patience and understanding. Despite my sudden absences and endless conversations on the same topic, they have always been there to support me. I want to make a special mention to Jeny, Eli, Loreto, Patri and Dani; my longtime friends. I am also very grateful to Elena, Fernando, Alberto and Felix, with whom I started my career almost ten years ago. With all of them I have shared countless adventures, laughter and exciting moments. Thank you for always making me feel so accompanied and supported; this thesis also has a little piece of you.

Lastly and most importantly, I want to express my deepest gratitude to my family. Everything I have achieved I owe to my parents, who have always given everything in their hands and have taught me the true values of kindness and affection. I would like to make a special mention to my grandmother, who I would like to let you know has been and always will be my spiritual guide; and to my grandfather, who day after day demonstrates his unwavering determination and has instilled in me the value of work and sincerity. I also want to acknowledge the fundamental role that Daniel has played in my life. He has been my great support during all these years, being the anchor that has provided me with serenity and calm when I needed it the most. Daniel, you are a true example of understanding and dedication.

Resumen

Los sistemas técnicos se enfrentan a exigencias cada vez mayores, con tareas que crecen en complejidad e incertidumbre. Cada vez se buscan más sistemas autónomos por razones de rendimiento, coste o seguridad. Sin embargo, sus capacidades a menudo se quedan cortas, exigiendo la intervención humana o limitando su aplicabilidad. Esta investigación explora el conocimiento explícito como medio de gestionar las perturbaciones durante el funcionamiento.

Para ello, integramos elementos de tres ámbitos aparentemente no relacionados. En primer lugar, la ingeniería de sistemas ofrece un enfoque holístico, con la ingeniería basada en modelos (MBSE), que aboga por la representación del sistema a lo largo de todo el ciclo de vida. En robótica, los modelos formales desempeñan un papel crucial en el diseño de robots. Sirven de base para lograr propiedades holísticas, como la fiabilidad funcional o la resiliencia adaptativa, y facilitan la producción automatizada de módulos. Esta investigación propone ampliar las conceptualizaciones formales más allá de la fase de ingeniería, utilizando modelos en tiempo de ejecución.

El segundo ámbito, Representación del Conocimiento y Razonamiento (KR&R), se refiere a la descripción de la información en formatos computacionales que permitan a los agentes utilizarla para deducir hechos implícitos. De este modo, tendemos un puente entre la ingeniería y la adaptación en tiempo de ejecución, mediante el uso de estas técnicas para explotar la información disponible desde el diseño hasta el funcionamiento del robot.

El tercer pilar es la Teoría de Categorías (CT), una teoría general de estructuras matemáticas que sirve como herramienta para razonar en diferentes niveles de abstracción. Proponemos utilizar estos formalismos para representar aspectos MBSE del sistema a través de la composicionalidad.

Aplicando estos principios, hemos desarrollado un *metamodelo* que sirve como lenguaje para definir aspectos de diseño relevantes para la adaptación del sistema. El modelo se basa en cuatro conceptos fundamentales: capacidad, componente, objetivo y valor, para garantizar que los sistemas aporten los beneficios necesarios a su propietario. El uso de CT para este lenguaje facilita el razonamiento a niveles superiores de abstracción. Por ejemplo, proporciona diferentes nociones de equivalencia dependiendo de la perspectiva o diversas granularidades en las relaciones conceptuales.

El *metamodelo* utiliza los resultados del lema de Yoneda—por el cual dos objetos son iguales si están relacionados con las mismas entidades—para identificar el candidato de adaptación más adecuado y con menor impacto en el funcionamiento. Además, utiliza el concepto de “*pushout*” para mejorar la explicabilidad. Al propagar información sobre cómo han cambiado los objetos y las relaciones, se puede informar a las partes interesadas sobre los cambios previstos en las métricas de desempeño. Por ejemplo, el cambio de un componente por otro puede aumentar la seguridad o reducir la precisión prevista. Estas métricas son relevantes

para el operador, que recibirá notificaciones sobre estos ajustes.

Este *metamodelo* se hace operativo a través de un *metacontrolador*. Este instrumento busca completar los objetivos de la misión de forma similar a como los controladores clásicos persiguen mantener la señal de referencia. El *metacontrolador* se integra en la ejecución del robot razonando sobre su estado actual y las capacidades disponibles. Por tanto, cuando surge una emergencia, no hay formas preestablecidas de adaptar el sistema; se seleccionan en función de sus necesidades y disponibilidad.

La amplia aplicabilidad y versatilidad de este enfoque se demuestran a través de dos plataformas de pruebas—un robot minero y un robot móvil—en cinco escenarios. Al abordar retos como los fallos de los sensores, las capacidades insuficientes y la pérdida de comunicación, nuestra solución muestra robustez y posibilidad de reutilización. En conclusión, nuestra investigación, que formaliza diseños y recuperaciones, presenta un enfoque prometedor para mejorar la autonomía de los robots. El *metacontrolador* y su *metamodelo* sirven de marco universal y neutral para representar formalmente y razonar sobre sistemas autónomos. La solución avanza en la fiabilidad de los robots mediante el auto-conocimiento de los sistemas robóticos complejos.

Abstract

Technical systems face increasing demands, with tasks that grow in complexity and uncertainty. Autonomous systems are increasingly sought after for performance, cost, and safety reasons. However, their capabilities often fall short, requiring human intervention or limiting their applicability. This research explores *understanding* as a means of handling disturbances during operation.

For this purpose, we integrate insights from three apparently unrelated domains. First, Systems Engineering offers a holistic approach, with Model-Based Systems Engineering (MBSE) advocating system representation throughout the life cycle. In robotics, formal models play a crucial role in robot design. They serve as the basis for achieving holistic properties, such as functional reliability or adaptive resilience, and facilitate the automated production of modules. We propose to extend formal conceptualizations beyond the engineering phase in the system life-cycle, providing accurate models for runtime operation.

The second domain, Knowledge Representation and Reasoning (KR&R), concerns capturing knowledge in computational formats so that agents can use it to derive implicit facts. We bridge the gap between engineering and runtime adaptation, using these techniques to exploit information available from design to the robot operation.

The third pillar is Category Theory (CT), a general theory of mathematical structures that serves as a tool for reasoning on different levels of abstraction. We propose using this formalism to represent MBSE aspects of the system through compositionality.

Applying these approaches, we have developed a *metamodel* which serves as a language to define relevant design aspects for system adaptation. The model is rooted in four core concepts—capability, component, goal, and value—to ensure that the systems provide the required benefits to its acquirer. The use of CT for this language facilitates reasoning at higher levels of abstraction. For example, it provides different notions of sameness depending on the perspective or various granularities in conceptual relationships.

The *metamodel* uses the results of the Yoneda lemma—by which two objects are the same if they are related to the same entities—to identify the most suitable adaptation candidate with less impact on operation. This renders a formalization of mission-oriented functional equivalence of systems. Additionally, it uses the concept of “*pushout*” to enhance explainability. By propagating information on how objects and relationships have changed, stakeholders can be informed of expected changes in performance metrics. For example, the substitution of one component for another may result in increased expected security or decreased accuracy. These metrics hold major relevance for the human operator, who will receive notifications about these adjustments.

This *metamodel* is operationalized through a *metacontroller*; an application-agnostic instrument to pursue mission goals similar to classical controllers pursue set-point references. The

metacontroller is integrated into robot execution by reasoning about its current state and available capabilities. Therefore, when a contingency arises, there are no pre-established ways to adapt the system; they are selected based on its necessities and availability.

The broad applicability and versatility of this approach is demonstrated through two testbeds—mining robots and mobile robotics—across five scenarios. Addressing challenges such as sensor failures, insufficient capabilities, and loss of communication, our proposed solution exhibits robustness and reusability. In conclusion, our research formalizing designs and recoveries presents a promising approach to enhancing robot autonomy and the engineering processes to attain it. The *metacontroller* and its *metamodel* serve as a neutral universal framework for formally representing and reasoning about systems. The solution advances in the topic of robot dependability through the self-awareness of complex robotic systems.

Contents

Agradecimientos	v
Acknowledgements	vii
Resumen	x
Abstract	xii
Contents	xiii
I Context: Engineering Autonomous Adaptive Systems	1
1 Introduction and Objectives	3
1.1 Problem Statement	3
1.2 Research Scope	4
1.3 Objectives	5
1.4 Methodology	5
1.5 Outline	7
1.6 Notation	8
2 Context and State of the Art	9
2.1 General Systems Theory	9
2.2 Autonomous Systems	11
2.3 Autonomous Robots	17
2.4 Fault Tolerance	23
2.5 Self-Adapting Systems	27
2.6 Formal Methods in System Development	34
II Foundations: Four Pillars for a Rigorous Technology	37
3 Systems Engineering	39
3.1 Definitions of Systems Engineering	39
3.2 System Concepts	44
3.3 Model-Based Systems Engineering	49
4 Knowledge Representation and Reasoning	53
4.1 Rational Agents	54
4.2 Symbolic Representation	55
4.3 Standard Forms of Logic	57
4.4 Representational Paradigms in Robotics	62
5 Ontologies for Robotics	69
5.1 Fundamental and Domain Ontologies	69

5.2	A Survey on Applications Using Ontologies for Robot Autonomy	72
5.3	Research Directions and Conclusions	113
6	Category Theory	115
6.1	Category Theory, a Natural Fit for Model-Based Systems Engineering	115
6.2	Category Theory Concepts for Engineering	116
6.3	Related Work in Applied Category Theory	126
III Framework: SysSelf - Systems That Know What They Are Doing		129
7	A Formal Metamodel for Autonomous Robots	131
7.1	Antecedents: TOMASys Metamodel	132
7.2	Implementation and Evaluation of TOMASys	135
7.3	Requirements of the SysSelf Metamodel	141
7.4	SysSelf: A Metamodel for Systems that Know What They are Do(ing)	142
8	An Operational Metamodel: Metacontrol	153
8.1	Metamodel Operationalization: The SysSelf Ontology	154
8.2	Using Ontological Reasoners	159
8.3	An Implementation Using ROS 2 Nodes	164
8.4	Using SysSelf: Step-by-Step Process	166
9	Evaluation and Results	169
9.1	Miner Robot	169
9.2	Mobile Robot	189
9.3	Analysis	194
10	Conclusions and Future Work	199
10.1	Conclusions and Main Contributions	199
10.2	Limitations and Future Work	201
IV Appendices		203
A	Scientific Dissemination	205
A.1	Journals	205
A.2	Conferences	206
A.3	Book Chapter	206
A.4	Patent	207
B	Implementation for an Underwater Mine Explorer Robot	209
B.1	Experimental Results	211
	Bibliography	219
	List of Figures	237
	List of Tables	242
	Acronyms	245

Glossary

249

Part I

Context: Engineering Autonomous
Adaptive Systems

Chapter 1

Introduction and Objectives

Technical systems are designed and built to perform a variety of activities in pursuit of user needs. In many cases, we want these systems to operate without human intervention for performance, cost or safety reasons. *Autonomy* requires the ability to accomplish the assigned task without external help. The autonomous car shall be able to negotiate an intersection and the autonomous space probe shall be able to reorient itself. However, autonomy alone is insufficient without *resilience*. A truly effective autonomous system must possess the resilience to withstand various disturbances that may occur during operation. Traditional control systems are designed to handle only limited forms of disturbance, but autonomy should handle a broader range of circumstances.

1.1 Problem Statement

The growth of computational capabilities, as well as the increase in data availability, has shown enormous potential to improve system performance. In principle, greater access to information and processing resources should contribute significantly to the development of autonomy, but it also poses strong risks. The recent wave of artificial intelligence (AI) based on machine learning has serious issues that shall be addressed to build dependable autonomous systems: (i) lack of quantity and/or quality of data, (ii) lack of transparency, (iii) lack of reproductability, (iv) lack of explainability, (v) ethical concerns, etc.

To overcome these issues and achieve AI-driven robot efficiency, robustness, safety, adaptability, and dependability, we shall endow the robot with a better *understanding* about the situation it faces, what capabilities the robot has, and the tools to act and react properly to uncertain situations. However, a mere increase in the resources used is not sufficient to gain true awareness of their actions. As stated by Brachman [1], transitioning from the age of information to the age of cognition requires adopting new perspectives.

We can't afford merely to increase the speed and capacity of our computer systems. We can't just do better software engineering. We need to change our perspective on computational systems and get off the current trajectory [1].

Our vision supports Brachman's statement in the sense that just improving existing methods is not enough. However, Software Engineering, or more generally, Systems Engineering provides valuable tools and practices to handle complexity. These tools help avoid omissions, mitigate the dependence on invalid assumptions, and facilitate effective change management throughout the engineering process [2].

Autonomy can greatly benefit from a novel perspective when applying Systems Engineering. Using these approaches, one can effectively tackle the variability that systems face throughout their entire life cycle. Such insights can potentially be used during operation to effectively manage and mitigate disruptions. This research aims to bridge the gap between system design and its operation as a means to reach robust autonomy.

1.2 Research Scope

This research is framed on the use of *declarative knowledge* to represent and reason about the cross-cutting elements of the robot, such as its mission, its design, and its operation in open environments. By exploiting this information at runtime, robots can make informed decisions to keep pursuing their goals, even in unexpected situations.

The concept of *self-awareness* is used to provide the system with the ability to observe itself in order to meet the requirements of autonomous systems in terms of robustness and reliability. Self-awareness makes it possible for deployed systems to be able to decide how to act; adapting if necessary. These features contribute to a fundamental motivation of the research: to deploy systems capable of controlling themselves, shifting the responsibility of adaptation from engineers at design time to the control system at runtime.

Models are the enablers of self-awareness. *Model-Based Systems Engineering* is a promising methodology that uses system representation throughout the entire life cycle, which has gained traction in robotics engineering due to the critical role that models play in robot design, enabling the achievement of holistic properties such as functional reliability and adaptive resilience while facilitating automated module production. However, to support the deployment of autonomous robots in real-world scenarios, a deeper understanding is needed beyond the engineering phase. We propose the use of formal conceptualizations using *Category Theory* as a mathematical framework to describe abstractions and produce accurate robot models.

The last fundamental aspect of this research is *generality*. This thesis aims to generate reusable assets and metamodels for managing system self-knowledge with broad applicability. While other control approaches leverage system knowledge to enhance robustness, such as fault-tolerant control or adaptive control, they are often suitable for specific faults or a particular operational region, relying on ad-hoc algorithms or predefined configurations. In contrast, our approach may be less focused on optimization for specific components or tasks, but is directed towards enhancing the overall operation of a diverse range of systems and domains.

1.3 Objectives

Our research focuses on using robust models at runtime as a means to attain a better understanding of complex situations and respond effectively in unstructured environments. This goal can be encapsulated in the following research question:

“How can we enhance autonomous robots self-awareness from a systemic perspective to make them more robust?”

This question can be translated into specific objectives to create a software asset that can be used at runtime.

1. Analyze existing approaches in technical systems and explore available methods for improving their dependability.
2. Explore the domain of system modeling and knowledge-driven robots to extract functional value for designing reconfigurable and adaptive systems.
3. Examine abstract mathematical domain for describing general structures that can serve as formal ground to systems conceptualizations.
4. Conceptualize elements and functions to drive system adaptation. This conceptualization shall establish a common basis for self-aware systems, addressing key autonomous system capabilities such as perception, reasoning, planning, actuation, explanation, learning, and reconfiguration.

Definitions shall adhere to formal standards, using concepts and relationships validated in the literature and abstract mathematical models.

5. Build reusable software assets that exploit robot system models at runtime. The solution shall be of general applicability to autonomous systems, independently of the domain, and have different possibilities depending on the extent to which the system can be reconfigurable.
6. Demonstrate the validity of the solution by developing testbed applications. Evaluate different aspects of the architecture using diverse testbeds, including both simulations and real robots.

1.4 Methodology

The methodology best suited for this thesis, given its multidisciplinary nature and the ambition of a general solution without losing sight of the engineering application, is iterative. We have followed Boehm’s spiral model of software processes [3], in which the project is represented as a spiral rather than a linear sequence of activities.

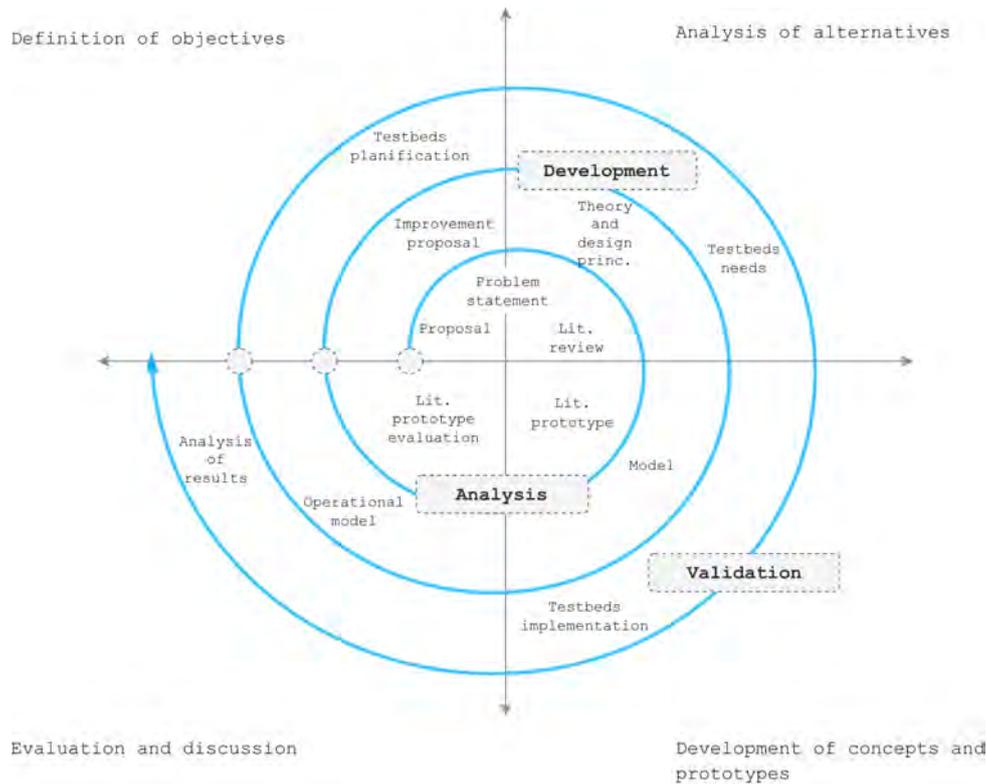


Figure 1.1: Methodology phases of the iterative process followed in this thesis, main milestones to achieve.

In this model, each loop of the spiral represents a phase. Figure 1.1 shows the specific stages considered in this research. The inner loop focuses on *analysis*, the next loop on *development*, and the external loop on *validation*. This structure constitutes a hybrid methodology that integrates elements of both the scientific method and the engineering process. Each loop in the spiral consists of four sectors: (i) definition of objectives, which determines specific goals, constraints, and plans for the process; (ii) analysis of alternatives and identification of needs; (iii) development of concepts and prototypes based on the previous stage; and (iv) evaluation and discussion of these concepts. Note that each sector affects each spiral loop—analysis, development, and validation—resulting in an iterative process. The main milestones to achieve are as follows:

1. Analysis

- Proposal of the problem to be solved.
- Definition of problem statement and research scope.
- Literature review based on the problem statement.
- Prototype of the literature-based model to determine the gaps in current approaches.
- Evaluation of the literature-based prototype.

2. Development

- Proposal for improvement based on lessons learned from the previous stage.

- Theoretical framework and design principles to address the founded gaps.
- Conceptualization of a model for dependable autonomy.
- Operationalization of the model and consistency evaluation.

3. Validation

- Planning of evaluation processes.
- Identification of testbeds needs.
- Implementation of testbeds to determine the viability of the proposed solution.
- Discussion and analysis of results.

1.5 Outline

The research work developed follows a spiral process, as introduced above. However, for enhanced readability, we have structured the discussion by unwinding the spiral. This dissertation is organized into four parts and ten chapters, outlined as follows:

- *Part I* consists of two chapters that contextualize the research problem.
 - *Chapter 1* introduces the motivation and objectives of the thesis, along with the methodology employed.
 - *Chapter 2* delineates the framework in which the problem is located, i.e., autonomous systems and various techniques to enhance their dependability, including fault tolerance, self-adaptation, and formal methods.
- *Part II* explains the core topics in the research and provides an in-depth review of projects that employ similar approaches. This part is subdivided into four chapters.
 - *Chapter 3* presents the domain of Systems Engineering to address complex engineering challenges and demonstrates its value in robotics. We also analyze the semi-formalized methodology of Model-Based Systems Engineering, which furnishes foundational concepts for the models developed in this research.
 - *Chapter 4* introduces Knowledge Representation and Reasoning as a means for deliberative action selection. This chapter describes symbolic representation, logic reasoning, and the most popular approaches for knowledge-based robotics.
 - *Chapter 5* delivers a systematic review of the use of ontologies in robotics and how conceptualizations of its functional competencies support action selection.
 - *Chapter 6* describes Category Theory, a mathematical domain for describing general structures used as baseline for the models developed in this research.
- *Part III* offers the core contribution of this thesis: the model-based framework for self-aware and self-adaptive autonomous systems.

- *Chapter 7* presents the concepts necessary for a formal language to determine the most appropriate adaptation action using explicit engineering knowledge.
 - *Chapter 8* details how to encode the aforementioned information so that the system can reason on it during robot execution.
 - *Chapter 9* describes the evaluation of this approach through deployment in two testbeds across five scenarios, discussing the results in terms of reusability, applicability and performance.
 - *Chapter 10* concludes this thesis, highlighting the main contributions achieved and suggesting future directions for this research.
- *Part IV* includes reference materials such as the bibliography, the scientific dissemination results of this research, and implementation details when the conceptual framework that precedes this thesis was evaluated.

1.6 Notation

This dissertation uses various terms in different domains. To enhance the clarity of the discourse, two text styles are used to disambiguate the context.

- *Concepts in the theoretical framework.*
- Specific elements for the model and metamodel.

Additionally, when there is a clarification on a concept or framework, the following separator is used.

► CLARIFICATION

Lastly, to aid in the explanation of abstract concepts and generalizations, examples are provided throughout the text. The following text style is used in a gray box.

Example: This is an example text.

Chapter 2

Context and State of the Art

Technical systems have become an integral part of our everyday lives, spanning from smart watches that monitor our daily activities to the electrical grids that power our homes, along with critical infrastructures in hospitals, manufacturing and transportation industries. As we increasingly rely on technology, ensuring dependability and trust becomes of utmost importance; especially when human intervention is limited.

This research project aims to address the challenge of enhancing the robustness of autonomous operation in these technical systems. This chapter serves as an introduction to systems, in particular autonomous systems, and highlights crucial aspects for improving their dependability, including fault tolerance, self-adaptation, and formal methods to guide its development. By examining these core elements, we aim to establish the context for our research.

2.1 General Systems Theory

The concept of *system* dates back to early Western philosophy and later to science [4]. Ludwig von Bertalanffy defined *system* as a “whole” consisting of interacting “parts” [5]. The configuration of these parts leads to attributes or behaviors that cannot be achieved by parts in isolation. As a result, certain holistic properties emerge from (i) the individual elements of the system and their properties, and (ii) the relationships and interactions among the parts, other systems, and the environment.

Bertalanffy aimed to bring together models, principles, and laws applicable to generalized systems, regardless of their specific nature. This perspective aimed to build a theory of universal principles applying to systems in general [5]. Boulding envisioned General Systems Theory (*GST*) as a theoretical model-building between highly generalized constructions of mathematics and the particular theories of specialized disciplines [6]. Klir highlighted the significance of a theory with the optimal degree of generality, a balance between content and its practical applications [7]. For him, such theory should not be exclusively limited to mathematicians, but rather serve as a tool for practitioners across various domains. The integration of theory and practice, grounded in a mathematical framework, is a fundamental

element of this research.

Klir provides two requirements for applying the term *system*. It must consist of a set of things of some kind, and they must be connected in some recognizable manner [8]. This broad definition is often bounded with an associative noun (e.g., medical system, robotic system, electrical system) or a context-dependent synonym (e.g., cardiac pacemaker, robot, power plant). As stated by Rosen [9], certain properties are subsumed under the modifier attribute, and others under the system conception, while still others may depend on both.

The definition of *system* differentiates a specific element of interest from anything external to it, known as the *environment*. Note that the boundaries between system and environment can sometimes be ambiguous within certain classifications. This idea is reinforced by the fact that systems generally cannot be studied in all its complexity; we observe or measure values of certain quantities relevant for the given purpose.

The system may refer to a unique object or may include some other relevant parts. Figure 2.1 illustrates an *engineered system*, which the International Council on Systems Engineering (*INCOSE*) defines as “a system designed or adapted to interact with an anticipated operational environment to achieve one or more intended purposes while complying with applicable constraints” [4]. The figure represents a system composed of (i) a device with software and hardware and (ii) an external sensor. Note that the choice of system definition is based on the observer and their interests. Moreover, the system may be viewed as a whole in one context and as a component subsystem in another.

In conclusion, systems offer diverse perspectives that can be used effectively in problem-solving scenarios. Most system challenges typically align with at least one of the following categories:

- *Analysis*: The process of dissecting a system with identifiable parts and established relationships to understand their behavior, as often occurs in sciences such as biology.
- *Synthesis*: In systems whose behavior and requirements are understood, the objective becomes uncovering the optimal structure to achieve the desired outcome, as in engineering domains.
- *The Black-Box Problem*: Addressing this challenge involves dealing with systems whose behaviors and structures are partially unknown, yet external factors provide observable insights into their properties, allowing for the determination of behavior, and inferring a hypothetical structure. This challenge is not limited to one domain but applies wherever information is incomplete.

The concept of system sets a common ground to tackle a wide array of challenges, providing tools and patterns for problem-solving and innovation from different perspectives.

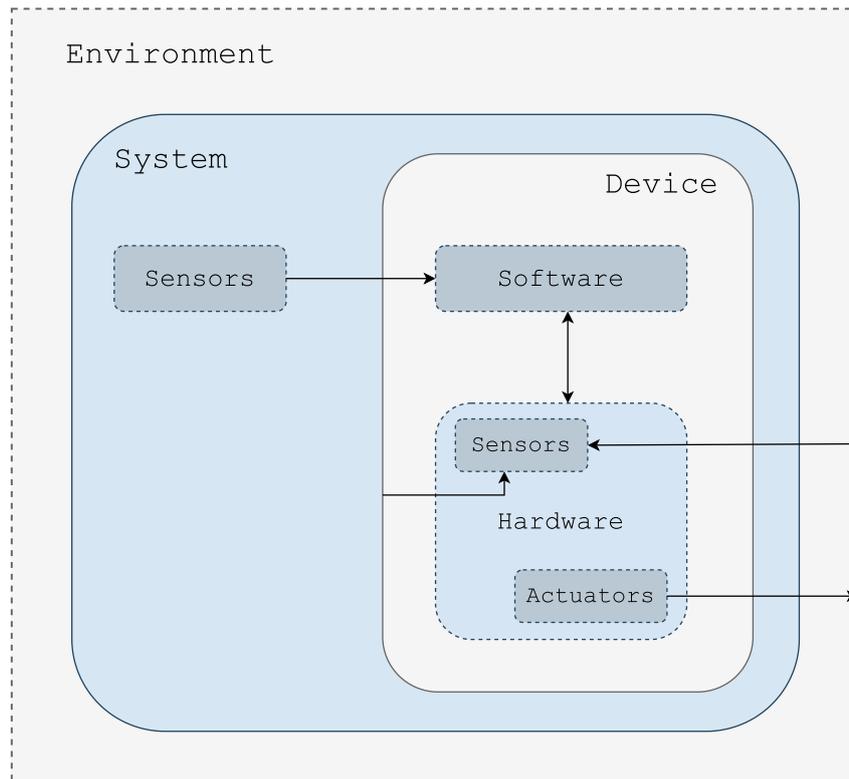


Figure 2.1: An engineered system and its environment. The system comprises a device and some external sensors in the environment. The device is further decomposed into software and hardware; the hardware includes actuators and sensors for both the device and its surroundings. The external sensors are used by the device’s software.

2.2 Autonomous Systems

Autonomy means, etymologically, being governed by the laws of itself, rather than conforming to external directives. The concept of system autonomy has garnered multiple definitions. Here we present some viewpoints:

- Bekey characterizes autonomy as the capacity of systems to function in the real world without any form of external control for extended periods of time [10].
- Franklin defines an autonomous agent as a system located within a part of an environment that senses and acts on it over time, in pursuit of its own agenda and to effect what it senses in the future [11].
- Beer conceptualizes autonomy as the extent to which a robot can *sense* its environment, *plan* based on that environment and *act* on that environment with the intention of reaching some task-specific *goal* (either given or created by the robot) without external control [12].
- Sanz declares that a system is autonomous in relation with a task and a context, if it can fulfill this task in this context without external help [13].

The core idea underlying these perspectives is that autonomy implies the capability to operate without external governance. A related, systems-oriented perspective pursued in our lab is to consider autonomy as a relation between system, task, and context. From an engineering standpoint, our focus is on achieving a level of autonomy that allows the system to pursue objectives without requiring human intervention. These objectives can either be commanded or fixed by the system itself to provide further value.

2.2.1 Levels of Autonomy

An autonomous system exhibits the inherent capacity to make choices. It is expected to select a course of action and subsequently adhere to it; opposed to automatic systems in which the choices are pre-given and it must only focus on following them. When evaluating autonomy, the viewpoint is often on how well a system performs tasks or, even better, how well the system development of tasks provides the expected benefits to the user.

Although stakeholders such as engineers, users, and acquirers may not always seek full autonomy due to practical reasons, they typically expect systems to function as independently as possible for specific activities in certain environments. This interplay between the system, task, and environment defines the challenge of engineering autonomous systems [13]. A scale of autonomy can provide a common language to describe requirements and standard metrics for assessment. However, the complexity arises from the multifaceted nature of autonomy.

Numerous attempts have been made to evaluate how autonomous the operation of different systems can be. One of the efforts to quantify autonomy was made by the Air Force Research Laboratory (*AFRL*). They focused on creating a set of measures that could cover all sorts of autonomous systems, especially Unmanned Aerial Vehicles (*UAV*). These measures needed to be detailed enough to show how technology investments were paying off, but simple enough for upper management to understand. This led to the creation of a tool called the Autonomous Control Levels (*ACL*) scale [14].

The *ACL* scale has ten levels, ranging from 0 (remotely piloted vehicle) to 10 (fully autonomous). In between, there is level 3, which means the system can handle small faults or events in real-time, and level 7, where there is a coordination between multiple systems. The first attempt evaluated three things: how well the system understands its surroundings (perception and situational awareness), how it makes decisions (analysis and decision making), and how it communicates with others (cooperation and communication). However, they realized that the metrics were not broad enough to cover the type of systems managed in the *AFRL* group. In particular, they found out that these metrics only measured tactical thinking, but did not quantify the achievement of strategic results. They also noticed that the last metric mixed “what” the system does (cooperation) with “how” it does it (communication). Consequently, they decided to adopt the Observe, Orient, Decide and Act (*OODA*) loop [15] as a framework to evaluate autonomous systems. This model, originally devised to understand how to get into human enemies decision loops—given the *AFRL* military background—was applied to understand unmanned systems decision loops. Therefore, the revised *ACL* scale is structured around four distinct aspects:

1. *Observe*: How well the system can understand what is happening around it, i.e., perception and situational awareness. It measures models, data and sensor fusion in the system.
2. *Orient*: How the system decides what is happening and how to work with others, i.e., analysis and coordination. It measures the capability to make inferences and diagnosis once a goal is provided.
3. *Decide*: How good is the system at selecting next actions, i.e., decision making. It evaluates if it is capable of avoiding obstacles; formulate plans, either independently or collaboratively; optimize them or replan.
4. *Act*: How well the system can actually execute actions to achieve its goals and the level of human intervention required on those actions.

Another approach to quantify systems self-govern was made by the Autonomy Levels for Unmanned Systems (*ALFUS*) Working Group. Their motivation was to establish a shared set of precise definitions which facilitated comparisons between diverse systems and their capabilities. This framework was also designed to determine the degree of autonomy required for each system development. The *ALFUS* model adopts three key points of view as pivotal parameters to evaluate autonomy levels. These viewpoints include mission complexity, environmental challenges, and the extent of human intervention [16]. Each of these points of view corresponds to a different axis within the framework metrics (Figure 2.2).

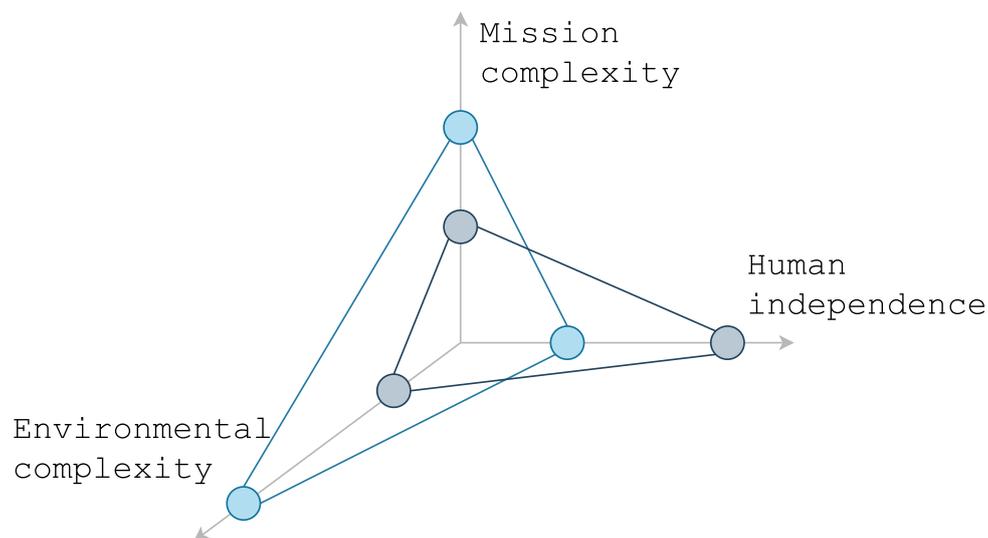


Figure 2.2: *ALFUS* model in which each axis corresponds to a metric value for mission complexity, environmental complexity, and human intervention (inverse).

To derive the different metric values for each axis, mathematical models or established standards were currently unavailable. Consequently, the framework proposed the following process [17]:

1. *Establish metric sets* for each axis, such as collaboration and performance for mission complexity; decisions per time ratio and trust per skill for human independence; and for environmental difficulty, factors per type of terrain and density of dynamic object.

2. *Decompose missions* to generate a task structure that can be broken down to the lowest skill levels.
3. *Assign relative weights*:
 - a) To each sub-task based on their significance to parent tasks or the overall mission.
 - b) To each of the metrics for each axis based on specific requirements or critical objectives.
4. *Evaluate and Score*:
 - a) Task and skills are evaluated against each metric.
 - b) Composite scores for sub-tasks are determined using metric weights.
 - c) Scores are aggregated with task weights to produce upper-level task scores. This process can be iterated until the entire mission is evaluated.

This methodology generates a rank from 0 to 10, quantifying autonomy levels based on factors such as human intervention, mission complexity, and environmental challenges. However, while ALFUS is theoretically applicable to all types of unmanned systems, it primarily centers around ground vehicles. Nevertheless, there are limited examples of the framework's implementation, and those available are partial. For example, planning metrics are proposed for mission complexity and workload metrics for human independence, but a metric for environmental complexity is absent [16].

While these methods tackle the complex challenge of quantifying autonomy within the system-task-environment triad, certain limitations are apparent. Both the ACT and ALFUS methods lack a standardized subjective evaluation criteria for the proposed metrics. Additionally, they do not provide a standard method for task decomposition and do not account for interdependencies among metrics.

These frameworks aim to evaluate autonomy in context, i.e., depending on the mission and the environment in which it is deployed. However, Durst et al. [18] propose to focus solely on the system. They present the Non-Contextual Autonomy Potential (*NCAP*) model, which introduces an explicit differentiation between autonomy and autonomous performance. This separation of concerns strives for a simplified approach, with the aim of a less subjective scale of autonomy.

NCAP introduces a four-tier categorization for assessing autonomy. Level 0 is attained once the physical sensors are tested and capable of perceive. Level 1 is reached with modeling software such as mapping, localization, target detection, etc. Level 2 is attained if the system is capable of direct its actions towards a goal, which depends on the availability of functions such as path planning, behavior generation, etc. Lastly, level 3 is reached when the system is capable to execute the previous capabilities, i.e., after testing the platform and its controls, the human interface, its mobility, etc.

However, these levels only quantify the capabilities of the platform itself. To determine autonomous performance, it is imperative to evaluate the mission outcomes and the environment in which it is deployed. A context-independent method can only provide some

potential autonomy. NCAP is limited to individual component performance such as bench testing cameras or SLAM algorithms. These scores are subsequently combined into one single performance metric that corresponds to the potential autonomy.

Therefore, the NCAP model has limited application in comparing between systems deployed in different environments or performing in a variety of systems. It is a valuable solution to compare similar platforms intended for specific contexts.

The three assessment methods—ACT, ALFUS, and NCAP—were developed in the context of military autonomous systems. As such, their conceptualization of a “mission” often equates to the system’s ability to change location. However, the focal point of this thesis is on reaching autonomy for complex missions and environments. Our objective is to provide a general formal framework with broad applicability across diverse systems, particularly within complex and open settings. An example mission within this framework involves a robot performing selective extraction of minerals in flooded mines (Section 9.1). Therefore, our proposed approach aims to facilitate resilient adaptations in the face of unforeseen contingencies. Although we abstain from directly quantifying levels of autonomy, the consequences of this thesis can be seen as a tool to enhance the system’s autonomy level.

2.2.2 Concepts Related to Autonomy

Autonomy is a broad term that covers a range of capabilities, from function independently or adaptation to changing conditions until the effective completion of tasks. The following concepts refer to possible requirements for the system operation:

- *Functionality*: The first requirement for an autonomous systems is having the capability of doing what is expected from it. The rest of the requirements refer to this one.
- *Reliability* is the ability of a system or a system element to perform its required functions under specific conditions without failure for a given period of time. These conditions can include the environment, the mission load, and any other factor affecting it which had been explicitly defined. Increasing autonomy could introduce complexities or risks that could affect reliability, as it emphasizes consistent and predictable performance under normal conditions.

The term reliability requires a failure—considered as a departure from the specification—and the appropriate time scale to be specified [19]. It is often measured as a probability.

- *Availability* is the probability of a system or a system element to be operational at a given point in time, under a given set of environmental conditions [19]. Availability refers to the extent to which a system is accessible and able to function, accounting for factors such as downtime and maintenance. A system with greater autonomy might identify problems in component availability and perform corrective actions to continue functioning.

- *Safety* is a property of a system which depends on how it behaves when used and sustained in a specific way in a specific environment [4]. In most cases, an autonomous system shall operate with safety, i.e., without causing unintended harm or damage to other systems, individuals, or the environment. Safety involves both the prevention of incidents and the mitigation of their effects if they do occur.
- *Security* is the ability to protect against intentional subversion or forced failure. It is a composite of four attributes—confidentiality, integrity, availability, and accountability—with the objective of assure usability [19]. An autonomous system shall be equipped with the necessary security measures to protect against threats while allowing them to effectively perform their intended functions.
- *Dependability* is the ability to provide the intended services of the system with a certain level of assurance, including factors such as availability, reliability, safety, and security defined above. Dependability ensures consistent performance that goes beyond reliability, as it also considers factors such as fault tolerance, error recovery, and maintaining service levels. Fisher et al [20] defines *survivability* as an aspect of dependability that focuses on the preservation of the core services of the system when systems are compromised.
- *Resilience* is the ability to maintain capability in the face of adversity, either by avoiding the cause of stress, endure this degradation or recover from it [4]. It includes the capacity to recover to continue functions in the presence of disruptions. Autonomy is reinforced in a resilient system, as it enables decision-making and adaptation.
- *Robustness* is the ability to resist degradation of capabilities under adverse conditions [19]. This property implies that the system can handle a range of conditions, deviations, or disturbances while still functioning adequately. It can be considered a component of resilience; however, it does not include the capacity to recover. A system exhibits higher autonomy when its robustness is enhanced, as the system can proactively address issues and continue its operations even in the presence of uncertainties or disruptions.
- *Maintainability* refers to the probability that a system or system element can be repaired in a defined environment with defined resources within a specified period of time. Increased maintainability implies shorter repair times [19]. A system that has higher maintainability is considered to be more available.

Moreover, other attributes are often associated with highly autonomous systems. These factors are not requirements but characteristics of systems that may become useful during the design phase of such systems.

- An *intelligent system* is capable of producing appropriate behavior with incomplete, non-formal knowledge; this definition was provided by McCarthy [21]. However, certain authors break down the definition to include capabilities such as process and analyze information, adapt to changing circumstances, make informed decisions, learn from experiences, and effectively achieve the system goals.

- A *cognitive system* is based on knowledge. We use the etymological meaning as opposed to the common use in the domain of intelligent systems, which is “imitating human thought processes.” This conceptualization aligns with the notion of rationality, where the system orients decision-making towards obtaining the best expected outcome, even when it faces uncertainty or contingencies.
- The system *value* is the amount of benefit, feature, or capability provided to various stakeholders. It considers the optimal set of requirements to deliver customer satisfaction [19]. Autonomous systems seek to perform tasks on behalf of some user. Its goals are commonly understood as the expected change in the world to complete these tasks. However, artificial systems are designed with a purpose; the real objective of any artificial system is to provide some value to the user [22]. We might think of value as the “ends” while system goals as the “means.”

In this research, our primary focus is on cognitive systems, which, as a result, leads to the development of intelligent systems. Our approach centers around increasing dependability and resilience of these systems, with a specific emphasis on delivering the expected value.

2.3 Autonomous Robots

Although all processes developed during this research are applicable to systems and based on their methodologies, all the evaluation is done within a particular type of system, autonomous robots. *Robots* are physical agents that perform tasks by manipulating the world. All robots are *agents*, which is any entity, physical or virtual, that acts. Robots are equipped with effectors such as legs, wheels, joints, and grippers to interact with the environment. They also have sensors which enable them to perceive their surroundings. Sensors such as cameras, radars, lasers, and microphones can get information about the environment or other systems around them—e.g., other robots, people, technological stations, etc. Robots can also measure its own state with, proprioceptive sensors such as gyroscopes, strain and torque sensors, and accelerometers [23]. In this section, we deepen in the essential capabilities required to achieve autonomy.

Building upon the concepts introduced in Section 2.2.2, rational agents use sense-decide-act loops to select the best possible action. Vernon highlights the interplay between cognition and autonomy [24]. For him, cognition includes six attributes: perception, learning, anticipation, action, adaptation, and, of course, autonomy.

Following this approach, Langley breaks down the main functional competencies that autonomous robots must demonstrate [25]. He describes knowledge as an internal property to achieve the following aspects:

- *Recognition and categorization* to generate abstractions on percepts and past actions.
- *Decision-making and choice* to represent alternatives to select the best possible action considering the situation.

- *Perception and situation assessment* to combine perceptual information from different sources and provide an understanding of the current circumstances.
- *Prediction and monitoring* to evaluate the situation and the possible effects of actions.
- *Problem solving and planning* to specify the desired intermediate states and the actions required to reach them.
- *Reasoning and belief maintenance* to use and update knowledge.
- *Execution and action* to support reactive and deliberative behaviors.
- *Interaction and communication* to share knowledge with other agents.
- *Remembering, reflection, and learning* to store past executions as experiences. Then, they can be used in future operations.

Similarly, Brachman believed that real improvements occur when *systems know what they are doing*, i.e., when the agent can understand the situation: what it is doing, where and why. He establishes practically the same foundations listed above, emphasizing the necessity of coordinated teams and robust infrastructure to improve autonomy [1].

The three authors acknowledge the significance of the mentioned capabilities, each with different levels of granularity. This shared understanding of the processes that enable autonomy provides us with a robust basis to further explore them.

2.3.1 Perception

A percept is the belief produced as a result of a perceptor sensing the environment in an instant. Perception involves five entities: sensor, perceived quality, perceptive environment, perceptor, and the percept itself.

A *sensor* is a device that detects, measures, or captures a property in the environment. Sensors can measure one particular aspect of the physical world, such as thermometers; or capture complex characteristics, such as segmenting cameras.

A *perceived quality* is a feature that allows the perceptor to recognize some part of the environment—or the robot itself. Note that this quality is mapped into the percept as an instantiation, a belief produced to translate physical information to the system model. Perceived qualities are temperature, or visual images of the environment; but also the rotation of a wheel measured by a rotative encoder placed in a robot.

A *perceptive environment* is the part of the environment that the sensor can detect. The perceptive region can be delimited not only by sensor resolution but also to save memory or other resources.

A system that perceives is a *perceptor*. The perceptor constitutes the link between perception and categorization, since it takes the sensor information and processes it to categorize it.

Usually, the perceptor embodies the sensor, as is the case in autonomous robots, but could be decoupled if the system processes information from external sensors. Finally, the *percept* is the inner entity—a belief—that results from the perceptual process.

2.3.2 Categorization

Percepts are instantaneous approximate representations of a particular aspect of the physical world. To provide the autonomous robot with an understanding of the situation in which it is deployed, the perceptor has to abstract the sensor information and recognize objects, events, and experiences.

Categorization is the process of finding patterns and categories to model the situation in the robots' knowledge. It can be done at different levels of granularity. Examples of categorizations appear everywhere: at sensor fusion processes from different types of sensor, with its corresponding uncertainty propagation; at the classification of an entity as a mobile obstacle when it is an uncontrolled object approaching the robot; or, in a more abstract level, when a mobile miner robot recognizes the type of mine ore depending on its geo-chemical properties.

In general, this step corresponds to a combination of information about objects, events, action responses, physical properties, etc. to create a picture of what is happening in the environment and in the robot itself. For this purpose, the robot shall incorporate other processes, such as reasoning and prediction.

2.3.3 Decision-making

An autonomous robot must direct its actions towards a goal. When an action cannot be performed, the robot shall implement mechanisms to make decisions and select among the most suitable alternatives for the runtime situation.

Note the difference between decision-making and planning. Decision-making has a shorter time frame because it focuses on the successful completion of the plan. An example of a decision at this level could be to slightly change the trajectory of a robot to avoid an obstacle and then return to the initial path. Planning, on the other hand, is concerned with achieving a goal; it has a longer time horizon to establish the action sequences to complete a mission. For example, the miner robot presented above has to examine the mine, detect the mineral vein, and dig in that direction.

Decision-making acts upon the different alternatives that the robot can select, for example, in terms of directions and velocity, but also in terms of what component with functional equivalence can substitute for a defective one. The execution of the action—how it is done—changes slightly, but the plan—the action sequence that achieves the goal—remains the same.

2.3.4 Prediction and Monitoring

Once the robot has an internal model, it can supervise the situation. This model can be given a priori or created before stating the task; it could also be learned. The model reflects the understanding of the robot about its own characteristics, its interactions with the environment, and the relationships between its actions and its outcomes. At runtime, the robot can use the model to predict the effect of an action. It can also anticipate future events based on the way the situation is evolving.

This mechanism also allows the robot to monitor processes and compare the result obtained with the expected response. In the event of inconsistencies, it can inform an external operator or use adaptation techniques to solve possible errors. For example, if a robot is stuck, it may change its motion direction to get out; and if this is not possible, alert the user.

The prediction process can also benefit from a learning procedure to improve the models from experience and refine them over time. Furthermore, monitoring can be used during learning, as it detects errors that can help improve the model.

2.3.5 Reasoning

Reasoning is the process of using existing knowledge to draw logical conclusions and extend the scope of that knowledge. It requires solid definitions and relationships between concepts. It uses instances of such concepts to ground them to the robot operation and to be used during the mission.

The reasoning process can be used to infer events based on the current percepts, such as the dynamic object approaching to the robot; or to infer the best possible action to overcome it based on its background knowledge, such as reducing speed and slightly changing the direction. Lastly, as robots operate in dynamic worlds—specifically, they operate by changing it—the knowledge shall evolve over time.

Reasoning includes two processes: (i) infer and maintain beliefs, and (ii) discard beliefs that are no longer valid. Logical systems support assertions and retractions for this purpose; however, they must be handled carefully to maintain ontological consistency. Truth maintenance is a critical capability for cognitive agents situated in dynamic environments.

2.3.6 Planning

Planning is the process of finding a sequence of actions to achieve a goal. To reach a solution, the problem has to be structured and well defined, especially in terms of the starting state, which the robot shall transform into a desired goal state. The system also needs to know the constraints to execute an action and its expected outcome, i.e., preconditions and

postconditions. These conditions are also used to establish the order between actions and the effect that they may have on subsequent actions.

The required information is usually stored in three types of models: environment, robot, and goal models. Most authors only mention the environmental model, which includes the most relevant information about the robot world and its actions, tasks, and goals; however, we prefer to isolate the three models to make explicit the importance of proprioceptive information and performance indicators for a more dependable autonomous robot.

Plans can completely guide the behavior of the robot or suggest a succession of abstract actions that can be expanded in different ways. This can result in branches of possible actions, depending on the result of previous states.

Planning is also closely related to monitoring; the supervision output can conclude the effectiveness of the plan or detect some unreachable planned actions. In this case, the plan may need adjustments, such as changing parameters or replacing some actions. Re-planning can use part of the plan or draw a completely new structure depending on the progress and status of the plan and the available components.

Lastly, successful plans or sub-plans can be stored for reuse. These stored plans can also benefit from learning, especially with regard to the environmental response to changes and action constraints and outputs.

2.3.7 Execution

A key process in robotic deployments is the execution of actions that interact with the environment. The robot model must represent the motor skills that produce such changes. Execution can be purely deliberative or combined with more reactive approaches; for example, a patrolling robot may reduce its speed or stop to ensure safety when close to a human—reactiveness—but it also needs a defined set of waypoints to fully cover an area—deliberation.

Hence, an autonomous robot must facilitate the integration of both reactive and deliberative actions within a goal-oriented hierarchy. A strictly reactive approach would limit its ability to direct the robot actions towards a defined objective; while an exclusively deliberative approach might be excessively computationally intensive and lead to delayed responses to instantaneous changes.

Another aspect of execution is control. Robots use controllers to overcome small deviations from their state. These controllers can operate in open-loop or closed-loop mode. Open-loop controllers apply predetermined actions based on a set of inputs, assuming that the system will respond predictably. Although they lack the ability to correct runtime deviations, they are often simpler and faster. Closed-loop controllers provide a more accurate and precise action based on the inputs and the feedback received. Control grounds the decision-making process by specifying the final target value for the robot effectors. It constitutes the final phase of action execution.

2.3.8 Communication and Coordination

In many applications, robots operate with other agents—humans or robots of a different nature. Communication is a key feature to organize actions and coordinate them towards a shared goal. Moreover, in knowledge-based systems, communication provides an effective way to obtain knowledge from other agents' perspectives.

Shared information provides means to validate perceived elements, fusing it with other sources, and providing access to unperceivable regions of the world. However, this requires a way to exchange information between agents in a neutral, shared conceptualization that is understandable and usable for both. In this research, we exploit ontologies as a common baseline for system knowledge.

Once the communication is established, we shall coordinate the actions of the systems involved. Decision-making and planning processes should take into account the capabilities and availability of agents to direct and sequence its actions toward the most promising solution. For example, in multi-robot patrols, agents shall share its pose and planned path to avoid collisions. Another example could be exploring a difficult access mine; in which we could use a wheeled robot for most of the inspection and a leg robot for the unreachable areas.

2.3.9 Interaction and Design

Interaction and design are often omitted when analyzing autonomous capabilities. Although they are part of the design phase, this engineering knowledge hold considerable influence over the robot's performance and dependability.

Interaction between agents can be handled through coordination; however, embodiment of robots can produce interactions between software and/or hardware components. Robots should be aware of the interaction ports and the possible errors that arise from them. This concern is presented by Brachman [1], as awareness of interaction allows the robot to step back from action execution and understand the sources of failure. This becomes particularly significant during the integration of diverse components and subsystems, where the application of Systems Engineering techniques proves to be highly beneficial.

Hernández et al. [26] argues about the need to exploit functional models to make explicit design decisions and its alternatives at runtime. These models can provide background knowledge about requirements, constraints, and assumptions under which a design is valid. With this knowledge, we can endow robots with more tools for adaptability; providing the capability to overcome deviations or contingencies that may occur. For example, a manipulator robot with several tools may have one optimal tool for a task; but if this component is damaged, it can use an alternative tool to solve the problem in a less than optimal way.

2.3.10 Learning

Most of the processes described above can improve their efficacy through learning: categorization, decision-making, prediction and monitoring, planning, execution, coordination, design, etc. Learning can be divided into three steps: remember, reflect, and generalize.

- *Remember* is the ability to store information from previous executions.
- *Reflect* involves analyzing that remembered information to detect patterns and establish relationships.
- *Generalize* is the process of abstract conclusions derived from reflection and subsequently extended to use them in future experiences.

Our focus in this analysis has been on knowledge-based robots, precisely the category that our work address. In this research, our emphasis is on adaptation, involving fundamentally the capabilities of reasoning, execution, and design. Learning is beyond the scope of this thesis.

2.4 Fault Tolerance

The common strategy to achieve system dependability is straightforward: identify potential disruptions—whether internal or external—and design, build, and deploy system capabilities to manage them. In general, a fault is something that changes the behavior of a system so that it no longer satisfies its purpose [27]. When considering faults, the first thing that usually comes to mind is something that is wrong inside the system itself, such as a wheel suddenly stopping or a problem with the communication subsystem. However, faults can also be triggered by changes in environmental conditions such as an increase in the ambient temperature or an unexpected presence of water. Moreover, faults might stem from design errors, such as unresolved maneuverability issues for specific setups or errors in parameter settings. In all of these cases, the fault eventually leads to considerable degradation in performance or even the loss of a system function.

There are three methodologies for handling system faults [28]:

- *Fault prevention*: This method anticipates faults and aims to prevent their occurrence. It does not require component redundancy since it assumes that all components will consistently operate as designed to ensure proper functioning of the system.
- *Fault repair*: As anticipating all possible faults is impossible, this technique assumes that failures will happen occasionally. In such cases, manual maintenance procedures are considered during the design phase and implemented when necessary during operation.

- *Fault tolerance*: To reduce maintenance-related downtime and increase the reliability of the system beyond anticipation, fault tolerant systems are designed to provide the service even in the presence of faults. A common strategy to achieve this is the use of redundancies.

For complex systems where manual repairs are impractical due to factors such as inaccessibility, high costs, or time constraints, fault tolerance is the optimal choice. This approach is typically complemented with fault prevention, as the components must be reliable. *Fault tolerance* is the engineering domain dedicated to developing methodologies for preventing, minimizing, and recovering from faults. When a malfunction occurs, it is crucial to quickly identify the source and make the appropriate decisions to avoid its propagation and reduce its effects.

In this domain, we shall distinguish three closely related concepts [28]. A system fails when it cannot provide the desired service. Therefore, a *failure* is a deviation in the behavior of the system from its specified requirements. This specification is a crucial process of Systems Engineering (*SE*) which is further elaborated in Chapter 3. An *error* is a property of the system state which may result in a failure. As errors are inherent attributes, they can be observed and evaluated. Conversely, since failures are not directly observable, they are typically inferred by identifying certain errors in the system state. Lastly, a *fault* represents some defect with the potential to produce an error. Faults cause changes in component and system characteristics, which produce undesired modes of operation or performance outcomes [27]. When an error arises, there is a fault involved; however, the mere presence of faults does not ensure the occurrence of an error.

2.4.1 Redundancy

Redundancy is a crucial aspect in fault-tolerant systems. Jalote defines redundancy as those system parts that are not essential for correct system operation, but come into play when fault-tolerant mechanisms are required [28]. Redundancy can manifest itself in various forms: hardware, software, or time [29].

Hardware redundancy, often called *physical redundancy*, involves the inclusion of additional hardware components to ensure system operation in the presence of faults. These components are commonly used in safety-critical systems and complex distributed systems, including spare elements, such as processors, memories, and power supplies. For example, in aircrafts, a well-known approach is Triple Modular Redundancy (*TMR*) where three hardware copies run in parallel, and their outputs are compared for consensus. Another approach is dynamic redundancy, where spare components are available to replace defective ones. In addition, coding plays a crucial role in fault tolerance, particularly in enhancing communication reliability. This approach involves detecting bit errors in transmitted information and, whenever possible, correcting them based on the applied encoding [28].

Software redundancy covers all programs and instructions designed to support fault tolerance. One straightforward method is to execute a repetition of a sequence of instructions to

overcome a fault, at the cost of reduced time performance. However, a widely adopted approach is analytical redundancy, in which an explicit mathematical model is used to diagnose the faulty system. This diagnostic information is then used to adjust the model and redesign the controller to meet the system specifications again [27]. This approach offers a cost-effective, reusable means of enhancing dependability through *fault-tolerant control*. Lastly, *Time redundancy* refers to the additional time needed to complete tasks when employing fault-tolerant techniques. A well-known software fault-tolerance strategy is N-version programming, where a critical algorithm is implemented in different ways to tolerate potential design or programming errors in any of the implementations.

2.4.2 Fault-Tolerant Control

From a *SE* perspective, achieving fault tolerance involves a close integration between the system and its controller. In systems-theory, we commonly refer to the model of the system as the “plant.” In the context of fault-tolerant control, the term “controller” has a broader meaning. It extends beyond the traditional role of feedback or feed-forward control law, providing an additional decision-making process responsible for the selection of the system configuration [27].

The rationale behind this approach is a control layer that continuously monitors the behavior of the system and adjusts the controller to ensure that the system continues to fulfill its objectives. In this case, even though a fault may persist, its impact is effectively masked because the system maintains an acceptable level of performance. This method is referred to as *active fault-tolerant control*.

Active fault tolerant control involves five phases:

1. *Fault detection*: This phase recognizes the presence of a fault based on an observed error in the state of a system component.
2. *Fault diagnosis*: This phase entails the identification of the fault’s nature.
3. *Control redesign*: Following fault detection, the control loop structure is adapted and appropriate algorithms and parameters are selected to accommodate the altered system.
4. *System reconfiguration*: In this phase, the system leverages the adapted controller to ensure it operates properly despite the fault.
5. *Application*: In the final phase, the system returns to regular operation.

However, in classical control theory, controllers are typically designed with the assumption of a faultless plant, although they are often engineered to accommodate faults to some extent. This approach is known as *passive fault-tolerant control*. Classical control techniques for passive fault tolerance include robust and adaptive control strategies.

Robust control provides a fixed controller that tolerates changes in plant dynamics without a change in its parameters. It is designed to explicitly handle uncertainty [30]. However, the scope of faults it can effectively handle is limited [27]. Moreover, its parameters are typically set with a trade-off between performance and robustness, making it less suitable for scenarios where maximizing performance during normal operation is the primary objective.

Some of these limitations can be addressed by *adaptive control*, where the controller parameters are adjusted in real time as the system parameters evolve. This principle is particularly efficient for dealing with slowly changing parameters in linear models [27].

Adaptive control shares a structural similarity to fault-tolerant control. However, faulty behaviors in adaptive control are often induced by non-linearities, as this approach often involve larger variations, and are not uniquely restricted to parameter changes. Therefore, fault-tolerant control becomes necessary for complex scenarios that may require altering the controller type, its algorithm, or the plant model.

In this research, we leverage the foundational concepts of fault-tolerant control to support model-based system adaptation. This approach extends beyond addressing parameter and controller changes and cover other aspects of adaptation, such as the structural reconfiguration of system components or even the change of mission tasks. Although fault-tolerant control seeks to complete task goals, we aim to ensure the delivery of the expected *value*.

2.4.3 Fault Tolerance in Operation

The implementation of fault-tolerant systems is closely tied to their architecture. Fault tolerance is a feature that must be integrated into the system during its design phase.

When applied to fault-tolerant control, as proposed by Blanke [27], the phases of error detection and damage confinement are consolidated into *fault diagnosis*, involving the detection and identification of faults. The stages of fault treatment and continued service are grouped into *control redesign*, which aims to adapt the controller to ensure that the overall system continues its operation. Figure 2.3 illustrates the architecture of a fault-tolerant control system employing this approach. As mentioned above, the separation of concerns between operation and reconfiguration is a fundamental aspect of the research conducted in this thesis.

Fault-tolerant control constitutes an important field for enhancing the reliability of systems. It has the capacity to address more substantial faults and disturbances compared to classical control techniques and is a cost-effective alternative to traditional physical redundancy solutions. However, in practical applications, fault-tolerant control solutions tend to be ad-hoc, tailored to specific anticipated problems. For this reason, the focus of our research is to develop a systematic methodology with an architectural approach that can effectively address faults and other contingencies.

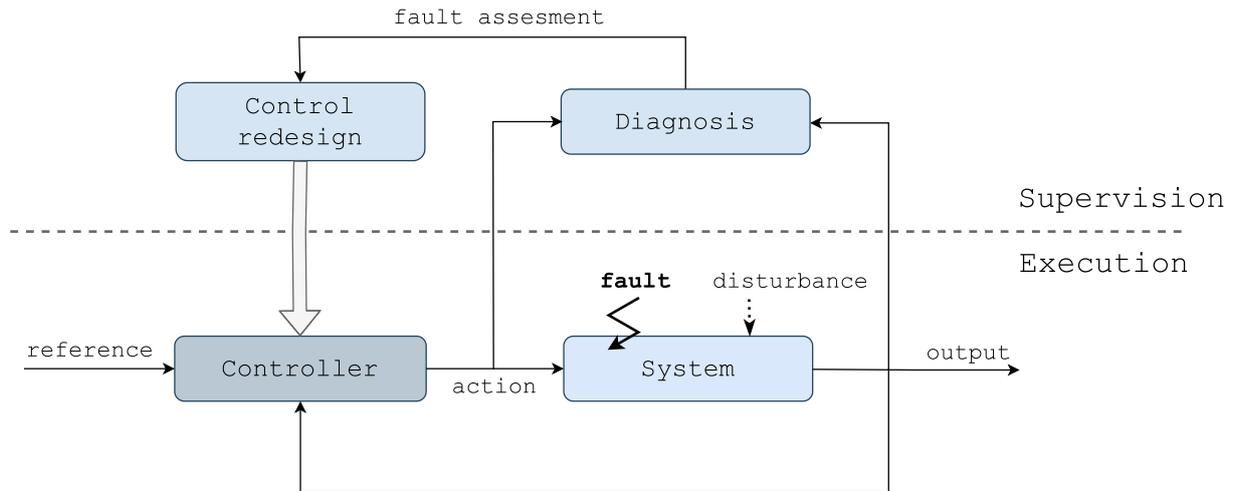


Figure 2.3: Architecture of a fault-tolerant control system, adapted from [27].

2.5 Self-Adapting Systems

Dynamic adaptation and fault tolerance are similar, yet distinct concepts that contribute to ensure system reliability and robustness. While both share the goal of preserving system performance, they address different aspects of system behavior. Self-adaptation focuses on performance optimization in response to changing conditions, while fault tolerance centers on maintaining system operation despite faults and failures. However, fault-tolerant techniques are typically predetermined during the development phase, limiting their ability to address unforeseen contingencies and leverage real-time information for identifying and implementing corrective actions.

Cheng et al. [31] define *self-adaptation* as the capability of a system to dynamically adjust its behavior at runtime in response to its perception of both the environment and its internal state. However, from our perspective, there is an additional triggering factor for adaptation: the recognition that the delivered value does not align with the specified requirements.

Self-adaptive systems rely on feedback loops as a fundamental mechanism to comprehend various aspects of self-adaptation. As described in [31], self-adaptation can be understood through four interrelated dimensions:

- *Goal:* Primary objectives that the system must accomplish.
- *Change:* Triggering event for adaptation. When there are variations on the operational context, the system must decide on how to face them. In this domain, context includes any computationally accessible information on which behavioral variations depend [32]. This involves aspects related to the environment, entities that interact with the system, and the system itself.
- *Mechanism:* Adaptation process that defines the relationship between the triggering change and the system's response to it.
- *Effect:* Impact and consequences of adaptation on the system.

In the domain of systems, the prefix “self” indicates that systems decide autonomously how to adapt or accommodate changes in their context and environment [33]. Nonetheless, practical implementation often involves higher-level objectives, hierarchical policies, or human-on-the-loop verification to facilitate the decision-making process for adaptation.

2.5.1 Architectural Approach for Self-Adaptation

There are some considerations that must be taken into account to effectively implement self-adaptation. System reconfigurability, mainly achieved through redundancy—analytical or physical—is essential to facilitate adjustments. Monitoring changes in the system’s context is crucial to determine when to apply modifications. Moreover, to maintain the expected value, the system should pursue a set of high-level goals, regardless of the runtime conditions. However, non-critical objectives could be relaxed to provide some degree of flexibility, especially when adaptation serves as a fault-tolerant mechanism.

Cheng et al. [31] propose a flexible approach that accommodates some level of incomplete information about the environment and its behavior during the requirement specification phase. They suggest to define requirements within some range that can evolve during system operation, especially those falling under the category of non-functional requirements, such as performance, flexibility, cost, reliability, security, safety, etc. However, this introduces a challenge in managing the system’s life cycle, specifically in dealing with the uncertainty surrounding requirements. While some requirements remain static, others allow for a certain degree of flexibility.

An effective method to ensure the fulfillment of requirements throughout the entire life cycle is the utilization of architectural design models. Garlan et al. introduce a straightforward metamodel for representing the system architecture in the form of a *hierarchical graph*. This metamodel offers a reusable structure embedded with semantic properties. To achieve this, they define a set of types for components, connectors, interfaces, and properties together with a set of rules that govern how elements of these types can be composed [34].

Figure 2.4 presents Garlan’s architecture, which naturally aligns with the generic feedback control loop. In this depiction, the core system is enhanced with additional solutions outlined in the architectural model. An external controller, known as the repair handler, generates the adapted architecture. Then, this new architecture is translated and executed within the system to facilitate runtime adaptation.

The approach taken by Garlan et al. is of particular relevance to the research developed in this work. They extend the use of architecture beyond the design phase, incorporating it as a fundamental element in supporting runtime adaptation through architectural models. These models serve to determine (i) which system properties to monitor, (ii) which constraints to assess, (iii) the course of action to take when constraints are violated, and (iv) how to adapt the system accordingly.

However, this strategy still exhibits some limitations, primarily in its constraints on adaptation operators (actions for altering the architecture) and predefined repair strategies. In contrast,

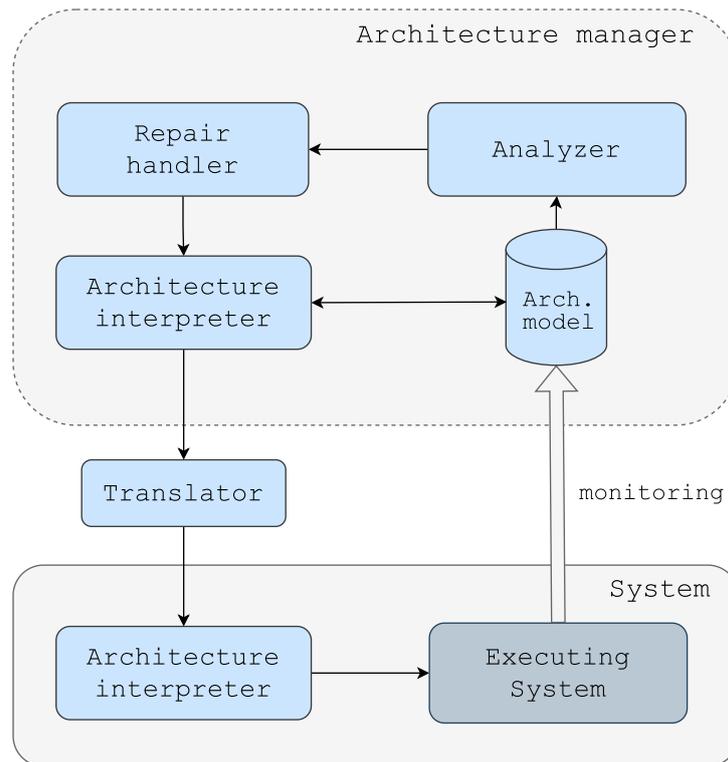


Figure 2.4: Garlan’s et al. architectural adaptation framework, adapted from [34]. The monitor mechanisms on the executing system trigger the architecture manager, which decides whether an adaptation to the system’s execution structure is required.

our research provides a formal framework that enables the computation of alternative adaptation solutions during robot operation, offering a more flexible approach to handling changes in real-time scenarios.

2.5.2 Self-Adaptation and Explicit Control

Self-adaptation can often be hidden by system design, especially in software-intensive systems [31]. We consider an adaptive system when it combines two key operational aspects: they make runtime decisions, at least partially, with regard to the system design; and they reason about the system environment and their own internal state.

To address these challenges, the self-adaptive community use control theory [33], a domain equipped with well-established mathematical models, tools, and techniques for analyzing various aspects of system behavior, including performance, stability, sensitivity, and correctness [35].

De Lemos et al. underscore the utility of control theory’s concepts and abstractions, even for systems too complex for direct implementation [36]. These theoretical aspects provide some design guidance to identify relevant control characteristics, as well as to determine the general steps and specific strategies to control the system.

Dobson et al. [37] introduced the autonomic control loop, which represents an evolution of the sense-plan-act loop originally conceived for the control of autonomous mobile robots. The autonomic control loop comprises four key stages:

1. *Collection*: Gathering relevant information from environmental sensors and the executing system.
2. *Analysis*: Processing and analyzing the collected raw data.
3. *Decision*: Making informed decisions on how to adapt the system to achieve a desirable state.
4. *Act*: Implementing the decision through concrete actions.

Nonetheless, when implementing this approach in the development of a self-adaptive system, various aspects within the engineering process require careful consideration, as outlined by Brun et al. [33]:

1. In the *collection* phase, considerations include aspects such as determining the ideal sampling rate and ensuring the reliability of the raw data.
2. In the *analysis* phase, the focus shifts to understanding the current state of the system, determining which information should be stored and for how long it remains valid.
3. In the *decision* phase, questions emerge regarding how to infer future system states to select the optimal solution while assessing potential risks and establishing priorities among multiple adaptation options.
4. In the *act* phase, decisions must be made concerning which entity performs the action, the timing of its execution, and how the adaptation is harmonized with other concurrent activities within the system.

While the autonomic control loop offers a high-level framework outlining the main activities for a generic feedback loop, it does not describe the essential properties and data flow specifics. These aspects are often partially addressed in particular implementations on the feedback loops.

► AUTONOMIC ELEMENT AND THE MAPE-K LOOP

IBM [38; 39] introduced the concept of an autonomic element, a fundamental building block to achieve self-adaptation through an explicit feedback control loop. This architecture is a well-recognized engineering approach for adaptation. The autonomic element consists of two main entities: the *managed element*, which includes the system with its sensors and effectors, and the *autonomic manager*, responsible for controlling the processes running on the system.

Much of the work in this domain focuses mainly on pure computing systems. IBM's conceptualization of managed elements comprises hardware resources such as CPUs, printers, or storage, as well as software resources such as databases, directory services, or large legacy systems. At the highest level, a managed element can even represent entities such as e-utilities or application services [39].

The architecture of the autonomic element is illustrated in Figure 2.5. The autonomic manager can be further decomposed into four components: a monitor, an analyzer, a planner, and an executor, all of which share a common knowledge base. The information exchange between these components produces a continuous loop known as the monitor-analyze-plan-execute loop, or *MAPE-K* loop. Each component provides a functionality in the manager sub-system [33]:

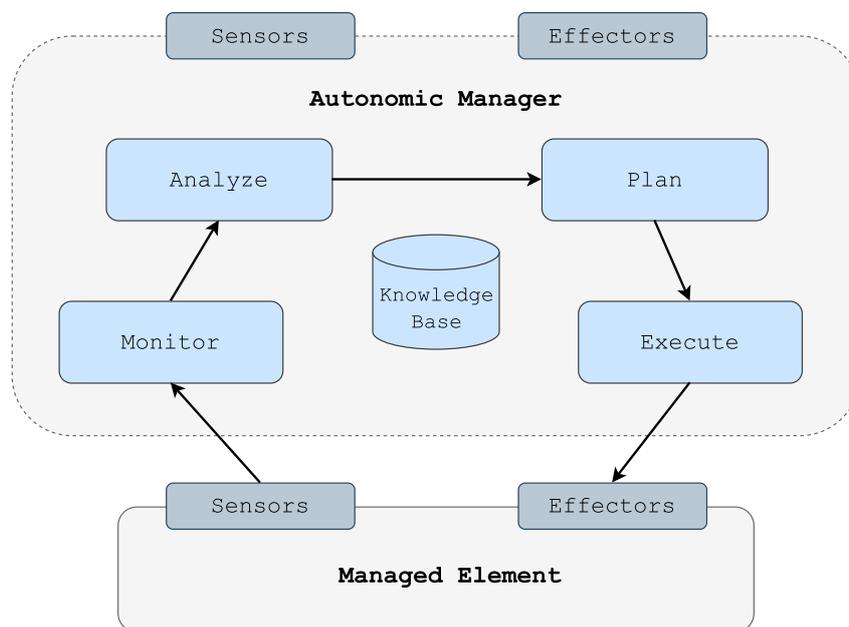


Figure 2.5: Autonomic element composed of a managed element entity and an automatic manager which executes the MAPE-K loop, adapted from [40].

1. *Monitor*: This component tracks the managed process, its environment, and its internal states. It filters sensor data and stores relevant information in the shared knowledge base.
2. *Analyzer*: The analyzer interprets the data from the monitor by comparing it with patterns in the knowledge base to diagnose the system. It also stores symptoms for future reference in the knowledge base.
3. *Planner*: The planner's role is to generate a plan based on the interpreted context.
4. *Executor*: The executor carries out the plan to implement the necessary changes in the system through its effectors.

One significant advantage of this approach is its composability. An autonomic element itself can become a managed element, and interfaces such as sensors and effectors are standardized

across building blocks to facilitate collaboration, data sharing, and control integration among autonomic elements [33].

As a result, autonomic elements can operate effectively at various levels of granularity [39]. They can manage individual components, such as disk drives, or supervise complete computing systems, such as servers. This scalability is also reflected in the complexity exhibited by each element, which includes a wide range of internal behaviors and relationships with other elements, as well as the ability to interact with different sets of elements. This dynamic results in a hierarchy of interrelated, self-governing components.

The autonomic element constitutes a self-adaptive system because it modifies the behavior of an underlying subsystem—the managed element—to achieve its overall objectives. However, conceptually, it operates as a feedback control loop, a concept derived from classic control theory [33; 41]. Nevertheless, it shares some of the limitations with common fault-tolerance approaches, as it often relies on ad-hoc solutions for contingencies defined in advance.

2.5.3 Beyond adaptation: Self-X Properties

Kephart and Chess [39] conceived Autonomic Computing as the primary solution to address the software complexity crisis within the IT industry. Their main motivation was to create systems capable of managing themselves according to the goals of an administrator. They intentionally used the term “autonomic” introduced by Horn [38] to draw an analogy with the autonomic nervous system, which governs fundamental, lower-level functions such as heart rate and body temperature without requiring conscious brain intervention. IBM’s initiative extends beyond self-adaptation and can be broadly described as *self-X* [42]: using system reflection for any task. In the world of pure software systems, this strategy is commonly named *self-** [43].

At the core of autonomic computing is the concept of *self-management* which seeks to free administrators from the details of system operation and maintenance. That is, to make systems more autonomous in their continuous operation. Several key aspects must be considered to achieve self-management [39]:

- *Self-configuration*: This is the capability of a system to autonomously configure itself based on high-level policies that specify desired outcomes without prescribing the exact methods. In scenarios where new components are introduced, the system seamlessly integrates them, making automatic adjustments as needed.
- *Self-optimization*: A self-optimizing system continuously seeks opportunities to enhance its operation by identifying ways to become more efficient in terms of performance or cost. In practice, this often involves the automatic search for updates to improve system execution.
- *Self-healing*: Self-healing systems possess the ability to detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware. They leverage knowledge about the system’s configuration and employ problem-diagnosis

components to recover the system. This aspect is closely aligned with the use of fault-tolerant capabilities.

- *Self-protection*: Self-protecting systems are equipped to defend against malicious attacks or cascading failures that cannot be resolved by self-healing measures. They can also proactively anticipate potential issues based on early monitoring and take actions to either avoid, or mitigate them.

To achieve these self-management capabilities, systems should have awareness of both their context and their internal state. This need is reinforced as the concept of *self-awareness* has gained interest as the complexity of systems has increased [44].

► SELF-AWARENESS

According to Kounev [45], self-aware agents may refer to three aspects. First, there is *awareness of the world*, which relates to the information accessible to the agent to generate or modify its behavior in response to the environment. Secondly, *awareness of the self* involves distinguishing the agent from its external world, which is facilitated by tracking the agent's development through past states and actions stored in its memory. Lastly, the third aspect is *attention* to determine which information the agent shall consider during self-reflection. Attention mechanisms enable the agent to decide which process and information segments are vital for monitoring and adapting to the situation.

In psychology and neuroscience, understanding how a system becomes *aware* of itself and its environment is conceptualized through two terms: “awareness” and “consciousness,” which, according to Francis Crick, refer to the same phenomenon [46]. In the context of machine intelligence, efforts to develop awareness have been significantly influenced by these biological perspectives [47; 48; 49].

In this research, the aim is not to follow a bioinspired approach to self-awareness but rather to establish a system-oriented method that provides functional benefits [50] associated with self-awareness. From this perspective, while there is no consensus on a precise definition, most researchers agree on two fundamental aspects. First, self-awareness refers to how a system distinguishes itself from its surroundings. In other words, having information about its own internal states and being able to differentiate itself from external world information. Secondly, it relates to having sufficient knowledge to perceive how the self-aware subsystem influences other parts of the system. Combining both approaches allows the agent to make decisions on how to act, which is especially useful for overcoming contingencies during operation.

Beyond adaptation, self-X can yield various advantages for systems [51]. For instance, during the design phase, programming can be less constraining, as the system can dynamically allocate resources based on availability at runtime. In collective robotics, self-awareness can serve as a mediator for processing information about other subsystems, predicting how the entire system will behave based on the interactions of individual components. Another benefit of system awareness is self-explanation. By comparing the current situation with the intended outcome, failures can be traced back to a specific component or group of candidates.

In cases requiring human intervention, the system can precisely direct the operation to the source that demands attention. All these aspects contribute to the main objective for which self-X is implemented, more autonomous systems.

2.6 Formal Methods in System Development

As technological systems become more complex, it is suggested to embrace a rigorous approach to ensure the reliability and integrity of these systems throughout their entire life cycle. *Formal methods (FMs)* represent an engineering methodology for specifying, developing and verifying software. This is achieved through a mathematically grounded notation and language. By using a specification language, we can systematically and, in many cases, automatically assess the software model for three critical qualities: consistency (by eliminating ambiguity), completeness, and correctness [19].

The application of FMs can vary in terms of rigor, ranging from occasional mathematical notations within natural language specifications to fully formal specification languages with precise semantics. In mathematical terms, formal specification languages are recognized as *formal systems*, which we further examine in Chapter 4. Beyond specification, they can ensure a system's behavior within specified conditions. This can be achieved through proofs, establishing the truth of a specific characteristic for all possible inputs, or through model checks, which determine if a state machine model satisfies given requirements [52].

FMs offer valuable tools to ensure that systems behave as intended. However, their full potential is often constrained because the available proof tools are still in a relatively primitive state [53]. As a result, they are frequently used for simplified designs or within the most critical components of a system [54]. Nevertheless, formal specification, as proposed by Hall [53], can be applied to enhance any type of system, regardless of its complexity. This approach involves reasoning about the system before constructing it, reducing not only errors but also the costs associated with discovering them late in the life cycle.

The essence of formal specification relies on extending the analysis and design phases while subsequently reducing the implementation, integration, and testing phases. This strategy aims to identify and rectify various types of error by establishing a common understanding of the intended behavior of the system. The main method to accomplish this is through abstraction on three different aspects:

- *What over How*: It specifies what needs to be done rather than prescribing how it should be done, providing flexibility in the implementation process.
- *Flexibility*: It adapts the level of granularity in the description to the stakeholder or the stage of development, facilitating communication processes and shared understanding of the system.
- *Data Types*: It utilizes data types such as sets and relations, rather than computer-oriented types such as arrays. This approach encourages a representation-oriented

method, making it accessible even without a strong mathematical or programming background.

A shared and unambiguous specification provides benefits far beyond the early phases of system development. It helps clients understand their purchase and offers a foundation to support other ways of expressing the specification. Furthermore, it enhances maintainability because it clarifies the role of each part and which functions need to be preserved during system adaptation. However, there is still a challenge to bridge the gap between real-world requirements and mathematical formalism, as well as to determine how best to represent them.

In this research, we focus on formal specification as a means to enhance system dependability. Sommerville divides specification styles into two main approaches: model-based and algebraic [55]. Model-based specification uses mathematics entities such as sets, relations, and sequences to define abstract machines, whereas the algebraic style relies on abstract data types and its operations to form axioms.

Additionally, there exist semi-formal notations such as UML and SysML that provide a set of symbols to represent specific roles in system descriptions, but these notations often lack precisely defined semantics [52]. Chapter 3 deepens into these semi-formal methodologies, while Chapter 6 introduces an abstract mathematical domain, Category Theory. Within this framework, we develop a novel approach to formal system specification that can be exploited at runtime to drive system adaptation.

Part II

Foundations: Four Pillars for a Rigorous Technology

Chapter 3

Systems Engineering

This research is grounded in three fundamental domains: Systems Engineering, Knowledge Representation and Reasoning, and Category Theory. This chapter focuses on the first domain, while Chapter 4 and Chapter 6 examine the other two.

In this chapter, we introduce the domain of Systems Engineering and explore relevant concepts for this research. Additionally, we deepen in Model-Based Systems Engineering, a semi-formalized methodology that provides the foundational concepts for the models developed in this research.

3.1 Definitions of Systems Engineering

Systems Engineering (SE) provides a multidisciplinary and integrative approach to address complex engineering challenges. Its primary objective is to facilitate teams in understanding and effectively managing systems development complexity to deliver successful systems. SE constitutes a discipline essential for the realization of systems that not only achieve their intended objectives but also exhibit resilience in real-world scenarios, while minimizing unintended actions, side effects, and its consequences [56]. SE is no longer an emerging discipline, but a well-established approach to achieving success in systems development with widespread adoption in critical sectors such as defense and aerospace [57].

In Section 2.1, a system is described as an arrangement of parts exhibiting a collective behavior beyond what individual constituents can accomplish. SE ensures that these elemental parts function effectively to achieve the objectives of the whole. Although these principles have been applied to technical artifacts throughout history, SE was not formally established as an engineering discipline until the mid-20th century. The term “systems engineering” emerged at Bell Telephone Laboratories in the early 1940s [4]. Its formal recognition as a discipline came in 2002 with the introduction of the international standard ISO/IEC 15288 [58].

Today, SE is guided by authoritative sources such as ISO/IEC/IEEE 15288 [59], which describes system life cycle processes, activities, and tasks. The International Council on Systems Engineering (*INCOSE*) is a non-profit organization that plays a central role in

developing and disseminating SE knowledge. The INCOSE handbook [4] defines best practices and serves as a reference for the application of SE techniques. Furthermore, the Guide to the Systems Engineering Body of Knowledge (SEBoK) [19] offers a comprehensive manual to key knowledge sources and references, providing in-depth coverage of the latest developments in SE and related topics.

The official definition of SE provided by INCOSE [60] and ISO/IEC/IEEE 15288 [59] is as follows:

***Systems Engineering** is a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods.*

SEBoK, while aligned with the official definition, adds emphasis on what it means to realize a successful system. They define SE as an approach and all means that enable the realization of systems that satisfy the needs of their customers, users, and other stakeholders [19].

Additionally, various authors have presented their own interpretations of SE:

- Chestnut [61] defined SE as a method that recognizes each system as an integrated whole composed of diverse, specialized structures and sub-functions. This approach seeks to optimize overall system functions according to weighted objectives of each sub-system to achieve maximum compatibility between them.
- Ramo [62] defined SE as a discipline that focuses on the design and application of the whole system, as distinct from individual parts. It involves examining a complete problem, taking into account all the facets and variables, including both social and technical aspects. According to Holt [57], Ramo's perspective is considered foundational to modern SE as he proposes to examine the system along with the influence of non-technical facets.
- Eisner [63] conceptualizes SE as an iterative process of top-down synthesis, development, and operation of real-world systems that satisfy a full range of requirements in a near-optimal manner. He introduced the concept of achieving requirements as a process that guides the operation of the engineer.
- The NASA SE Handbook [64] describes SE as a robust approach to design, creation, and operation of systems. It involves identifying and quantifying system goals, creating alternative design concepts, performing design trades, selecting the best design, verifying proper construction and integration, and assessing post-implementation performance. This definition highlights the concepts required in SE further developed in Section 3.2.1.

In summary, these diverse perspectives contribute to a comprehensive analysis of SE, with the purpose of establishing a common understanding of a system throughout its life cycle, from conception and development to production, utilization, and retirement. It provides a systematic methodology for achieving a shared vision of the current state of the system and its

future evolution. This approach transforms stakeholder needs, expectations, and constraints into a viable solution while considering objectives, budget, and schedule constraints.

According to INCOSE definition [60], SE focus on the following aspects:

- *Integration of Stakeholder Goals*: SE integrates the goals of different stakeholders with diverse needs, purposes, and success criteria.
- *Process Approach*: It establishes a process approach to manage complexity, uncertainty, change, and variety throughout the system's life cycle.
- *Concept and Architecture Generation*: SE involves generating and evaluating alternative concepts and architectures to find the most suitable solution.
- *Guidelines for Requirements Modeling*: It provides guidelines for modeling requirements and deriving a solution architecture at each stage of the system life cycle.
- *Design Synthesis and Verification*: SE includes design synthesis, and system verification and validation to ensure the system meets its intended objectives.
- *Identification of Enabling Systems and Services*: It identifies enabling systems and services and defines the roles of parts and their relationships to achieve a satisfactory solution.

In conclusion, SE provides a structured methodology for integrating the different disciplines required to engineer a system that addresses both the business and technical needs of users, acquirers, and other stakeholders. The goal of SE activities is to manage risks and avoid or minimize contingencies throughout the system's life cycle, from development to retirement, including its real-world operation. This systematic approach improves the likelihood of successful engineered systems.

3.1.1 SE as a Tool for Handling Complexity

When addressing the domain of system design, especially within the context of autonomous systems, it becomes necessary to distinguish between two fundamental phases:

- The engineering phase which addresses the development of a system, with the objective of optimizing its operational effectiveness.
- The operational phase, which implies the execution and practical functioning of the system, culminating in the achievement of desired outcomes.

SE provides a methodology for the first facet, with the ultimate goal of enhancing the second. Similarly, this research aims to improve the dependability of system operation by offering a

set of formal tools to system developers. Therefore, a comprehensive understanding of SE foundations is essential for the proper realization of this work.

This dual perspective, covering both the engineering and operational aspects, is also relevant when evaluating systems that face difficulties. These challenges can result in failures when a system cannot fulfill its intended purpose or when project constraints become unmanageable, such as cost or time limitations. Alternatively, they can be termed “disasters” if they produce adverse environmental impacts or harm individuals. Both scenarios require examination from (i) the engineering process and (ii) the system’s operational perspective. However, almost all disasters and failures can ultimately be attributed to what Holt [57] identifies as the *three evils of engineering*: complexity, lack of understanding, and poor communication. Addressing these challenges is the main motivation of SE.

► COMPLEXITY

Complexity in engineering refers to the number of elements and the degree of interconnect- edness between the elements that make up a system. It requires understanding not only each individual element, but also the relationships between them. Moreover, elements themselves can be composed of other elements and relationships, making the process of understanding complexity iterative. Furthermore, the relationships between elements may change based on the states or conditions that the system experiences.

Nowadays, most technological systems have the potential to offer solutions to challenging problems at the cost of being complex. Some of them may even exhibit holistic characteristics such as autonomy or self-organization. However, they often involve not well-known and even counterintuitive dynamics and cause-and-effect relationships that are difficult to manage and predict [4].

Sheard and Mostashari [65] divide complexity into three dimensions:

- *Structural Complexity*: This dimension pertains to the composition of a system, involving its constituent parts and the relationships between them. It includes various ways in which elements can be combined or configured, ultimately influencing the system’s potential to adapt to external needs.
- *Dynamic Complexity*: Dynamic complexity focuses on the behavior of a system as it performs specific tasks within its environment. It focuses on how systems interact over time to produce particular behaviors and their subsequent effects.
- *Socio-Political Complexity*: This dimension considers the influence of human factors on complexity. In SE, it is related to the multiple perspectives and biases of stakeholders in the system context. The interplay of these individual viewpoints can lead to unpredictable emergent behaviors during system conceptualization and design.

SE uses specialized techniques to understand complex system behavior and predict the impli- cations of altering it. Moreover, it addresses the aspect of “perceived” complexity throughout

the engineering process, illustrated by the *brontosaurus analogy* [66]. This analogy suggests that project complexity is low at the beginning, “head of the brontosaurus,” but as requirements and constraints gain clarity, the project enters the “belly of the brontosaurus,” where it is often perceived as exceedingly complicated. Subsequently, as the design progresses and optimization takes hold, the project advances towards the “tail of the brontosaurus,” signifying proximity to a solution. This perception of complexity evolves in tandem with the project’s life cycle, reflecting the growing comprehension of the system under development.

► LACK OF UNDERSTANDING

A lack of understanding can occur at any stage of the project. It often occurs during the initial phases when needs and requirements are being gathered, leading to ambiguity on system’s functionalities.

Similarly, it may emerge during the development phase, particularly when there is insufficient domain knowledge. In such cases, it may result in flawed assumptions about critical aspects such as production constraints, operation parameters, or characteristics of the deployment environment.

Lastly, during the system’s operational phase, incorrect product usage, such as the disregard of safety procedures or deploying the system in an unsuitable conditions, can lead to failures or disasters. Regardless of the stage, a deficiency in system understanding can give rise to more significant issues later in the system’s life cycle.

► POOR COMMUNICATIONS

Communication problems can arise at any stage during the engineering process. These issues may occur from personal difficulties in understanding due to language barriers, differences in technical background, or personalities clashes between team members. Furthermore, during system operation, effective communication is crucial, including both system-to-system and system-to-human interactions. This extends to communication within the system, human operators, and other supporting systems.

This aspect forms a cyclical relationship, often referred to as the “three evils triangle,” where each element aggravates the others. For example, unmanaged complexity can lead to a lack of understanding and communication problems. Conversely, communication problems can make it difficult to identify and manage complexity, while a lack of understanding can further worsen communication problems and increase overall system complexity [66].

It is important to note that while these risks cannot be completely eliminated, SE provides a framework for addressing them individually and from different perspectives to increase the opportunities to manage and mitigate them.

3.2 System Concepts

Most technological systems are Systems-of-Systems (*SoS*). ISO/IEC/IEEE 15288 [59] defines this concept as a set of systems or system elements that interact to provide a unique capability that none of the constituent systems can accomplish on its own. Essentially, a SoS is a composition of interacting systems—that, in turn, is comprised of system elements. A System-of-Interest (*SoI*) is defined a system whose life cycle is under consideration. Up to this point, we have explored the domain of systems and their inherent complexity. However, to fully leverage SE we shall establish some foundational concepts.

► HIERARCHY

One of the challenges of SE is to determine the level of detail required to represent its related entities. The concept of *hierarchy* involves defining the composing elements of a system only in relation to the whole system—or SoI—but suppress all interaction and interrelations with other parts. Figure 3.1 shows the hierarchical decomposition of a SOI; each element within this structure can be considered atomic or, in some cases, as a system itself [4].

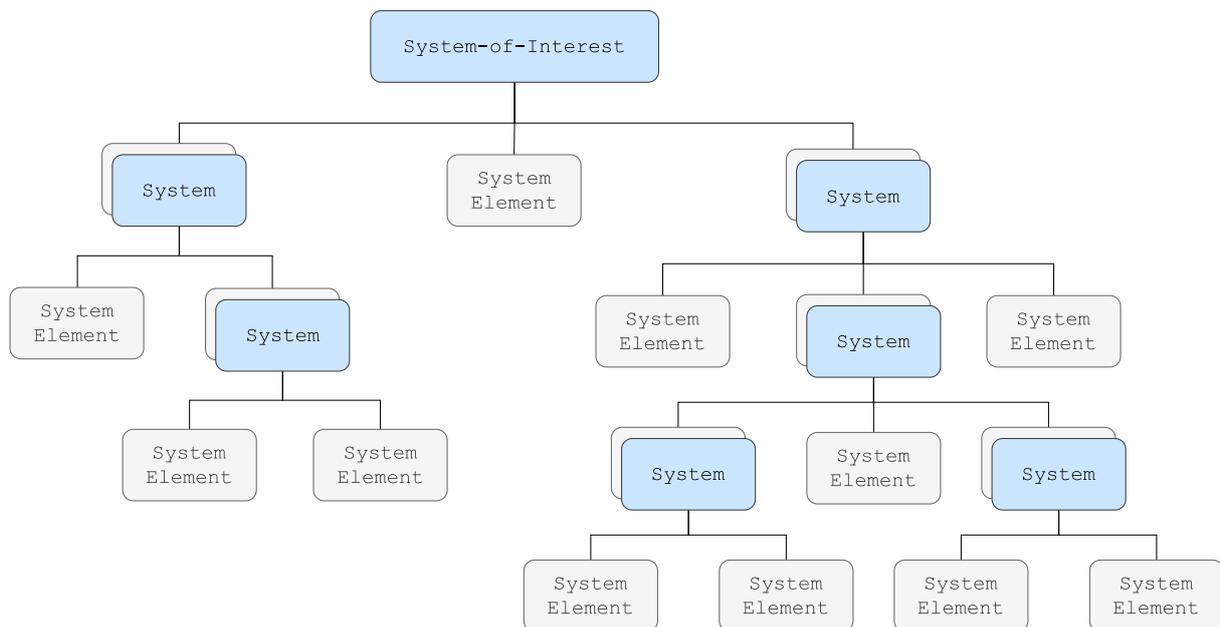


Figure 3.1: Hierarchical decomposition of a System-of-Interest composed of several systems and atomic elements, adapted from [59].

Furthermore, at a specific level, we can depict the interrelationships between system elements in a horizontal view. This perspective typically includes aspects related to requirements, integration, verification, validation, external systems, and activities. This structure also captures how these horizontal elements are derived from or lead to other elements in the lower and higher levels of the hierarchy, which constitutes the vertical view of the system.

► BOUNDARY

Systems typically require interactions with external elements. These elements constitute the environment of the system, which may involve users, operators, enabling systems, and other surrounding elements. A *boundary* delimitates the “line of demarcation” separating the system under consideration, *SoI*, and its environment [4]. This delimitation helps to distinguish elements belonging to the system from those that are contextual, enabling engineers to account for them effectively.

► EMERGENCE

As systems consist of an integrated combination of elements, their behavior derives not only from the interactions of individual elements but also from the interrelationships among system elements. *Emergence* is the concept that describes the manifestation of properties attributed to the whole system that individual elements, in isolation, do not possess [4]. This phenomenon is observed in properties such as those described in Section 2.2.2, including system safety and resilience.

► TRACEABILITY

In complex systems, it becomes essential to manage various horizontal aspects of systems, such as needs, requirements, architectural elements, and verification artifacts, which influence different parts of the system. *Traceability* is the capability to establish these associations between concepts and elements [4]. It ensures the consistency of information across various stages of the system life cycle processes, such as requirements definition, architecture definition, design, analysis, or verification and validation. Traceability is instrumental in establishing relationships between these processes, allowing us to monitor, for example, the influence of stakeholder requirements on the conceptualization of the physical product and its functionalities; so when a requirement changes, we can assess how it will impact its associated elements and their configurations.

► REQUIREMENT

A requirement is a statement that describes a need with its associated constraints and conditions [67]. It represents an abstract concept that may exist in various forms, even if only tacitly on the thoughts of some individuals.

SE offers some guidelines for formalizing requirements, allowing them to be captured in a structured manner for traceability and reasoning. Requirements capture is seen as an essential process and many methods have been proposed to this end. For example, Holt et al. [68] provide some recommendations on how to formalize requirements:

- Requirements must be uniquely identifiable to facilitate traceability.

- Requirements must be defined clearly and unambiguously. Using verb constructs such as “shall,” “will,” and “should” can provide a precise meaning for the requirement.
- Each requirement must have an owner, ensuring that stakeholders take responsibility for meeting it.
- Requirements must specify their origin, making it explicit who or what request them.
- Requirements must be verifiable, which means that they can be demonstrated to function correctly.
- Requirements must be possible to validate, i.e., it shall be demonstrable that they have been satisfied.
- Requirements shall be prioritized to establish an order for their fulfillment.

The taxonomy of requirements can vary depending on the project, but it commonly includes three main types. First, there are *business requirements*, which focus on economic and value-oriented considerations. They are shared between several projects within the organization. Second, *functional requirements* deal with observable outcomes related to actions, operations, or constructions. Finally, *non-functional requirements* encompass various constraints, including performance, legislative compliance, and other intangible attributes that specify how the system should perform and behave—such as reliability, maintainability, safety, and security.

► ARCHITECTURE

A system architecture is a framework to establish the fundamental concepts or properties of a system in its environment. It establishes the governing principles for the realization and evolution of the system and its related life cycle processes [59]. The purpose of its conceptualization is to determine suitable solutions that address stakeholder concerns [69] and system requirements [59]. This process relies on the use of different views and models, ensuring that the system architecture remains consistent across various perspectives and abstraction levels.

The definition of a system architecture characterizes fundamental concepts and properties of the system and its constituent elements. Its primary objective is to outline the high-level structure of the system and its components while specifying the desired attributes and characteristics. As a result, this process requires active participation from a diverse set of stakeholders, including engineers, SE practitioners, and domain-specific specialists. In general, its definition is an iterative process, subject to refinement throughout the system’s life cycle and across multiple levels within the system hierarchy to ensure that the system design continues to satisfy stakeholder needs and requirements during the whole system progression [70].

3.2.1 Life Cycle Concepts

A system, from its initial conception to its eventual retirement, evolves through various stages known as the system *life cycle*. It can be valuable to have a model that represents this progression and guides the system's evolution. This model is referred to as the *life cycle model*, which constitutes a framework of processes and activities associated with the whole system development [59]. The life cycle model serves as a common reference for communication and understanding throughout the entire system span.

The life cycle model facilitates effective planning and provisioning for the system's progression. It is organized into *stages* which help define and organize milestone achievements. ISO/IEC/IEEE 15288 [59] defines six common stages for systems: concept, development, production, utilization, support, and retirement.

The transition from one stage to another depends on the completion of specific tasks and the demonstration of successful outcomes, as outlined in ISO/IEC/IEEE 15288 [59] or ISO/IEC/IEEE 12207 [71]. This progression is not strictly linear in terms of time or extension.

Figure 3.2 depicts a possible life cycle progression where the concept stage leads to development and production. After utilization, there is an iteration back to the concept stage. Support activities occur in parallel with both the production and utilization stages. Life cycle stages can be iterative, concurrent, or overlap.

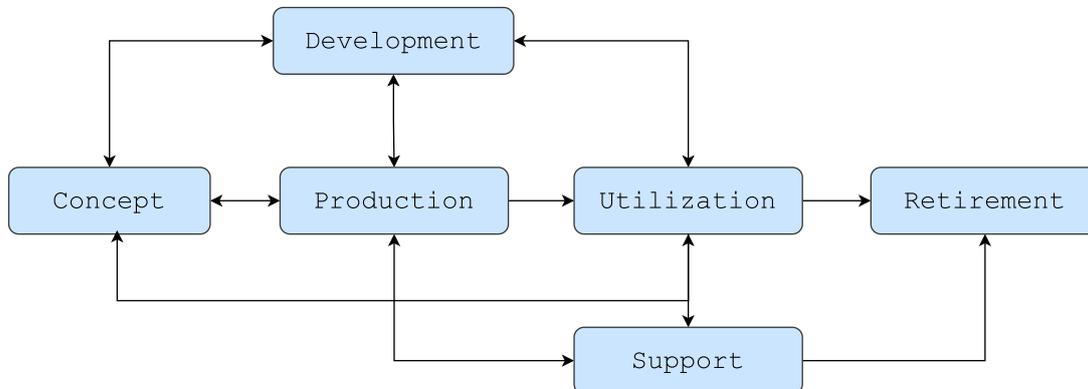


Figure 3.2: Example of a life cycle progression, adapted from [72].

The duration and processes involved on each phase be tailored to align with the system's scope, magnitude, and complexity to benefit from the model outcomes, such as high-level visibility and control of project and technical processes. The following describes the six common stages in any system's life cycle.

► CONCEPT

The concept stage serves the purpose of identifying stakeholder needs and proposing one or more viable solutions. This stage involves defining a problem space and exploring potential solutions, which entails conducting surveys and performing trade-off analyses.

This stage produces preliminary concept artifacts, the most important of which is the Operational Concept (*OpsCon*); a user-oriented document that describes characteristics of the system to be delivered [19]. Typically, it includes preliminary architectural solutions and stakeholder requirements. Additionally, it may involve defining essential initial system requirements—such as capabilities or overall performance—and establishing acquisition and risk management strategies.

Note that the decisions made during this stage will significantly influence, with increasing difficulty of changing, the possibilities for all subsequent stages [4].

► DEVELOPMENT

The development stage aims to refine system requirements to create an engineering baseline for developing a feasible system that meets stakeholder needs and requirements. This baseline addresses architecture, design, and documentation of the system for subsequent stages. The results from this stage may include prototypes and plans for integration, verification, and validation.

The description of the solution and its assets is typically incremental, allowing for iteration throughout the system realization process, which is especially beneficial for complex systems.

► PRODUCTION

The production stage involves translating the baselines from the development stage into an actual system. This includes the production, inspection, and testing of each part of the system to ensure its qualification for use. Additionally, this stage involves the provision of the necessary documentation for the utilization, support, and retirement stages.

► UTILIZATION

The utilization stage involves operating the system to fulfill the users' requirements. During this stage, potential product modifications may be identified and the system can be iterated based on both operational experience and stakeholder feedback. To support these changes, it is essential to maintain documentation from previous stages.

► SUPPORT

The support stage has the purpose of maintaining the system's capabilities during its utilization. This stage is responsible for addressing deficiencies and failures and taking mitigation actions to overcome them. It also involves documenting evolutionary modifications for future implementations. Modifications can be related to operational or support issues, as well as cost reduction or extending the system's operational life.

The support stage ends when a decision is made that the system has reached the end of its

useful life or should no longer be supported [4].

► RETIREMENT

The retirement stage includes the storage, archiving, and/or disposal of the system. Activities in this stage focus mainly on ensuring that the disposal requirements defined in the concept and development phases are satisfied.

A proper retirement process is essential to avoid unwanted consequences. Nowadays, most countries have laws and regulations for the proper end-of-life disposal of systems.

3.3 Model-Based Systems Engineering

Model-Based Systems Engineering (*MBSE*) is a semi-formalized methodology that exploits system representations to manage system structure and behavior along the whole life cycle. MBSE provides support for SE activities such as requirement definition, design, analysis, verification, and validation throughout the entire life cycle of complex systems.

MBSE is often compared to legacy document-based approaches, where systems engineering captures system design information through multiple independent documents in various non-standardized formats [19]. This methodology proposes using models as primary artifacts to express critical system information in a concise and coherent format. Models serve to integrate all knowledge related to the system, facilitating consistency and traceability along the whole life cycle, and therefore reducing developmental risk.

The primary objective of MBSE approaches is to create a model that abstracts the system [66]. Such models consist of a set of views that can be visualized using diagrams or text. These views are generated from a set of viewpoint templates for a specific purpose. Ontologies play a crucial role in maintaining consistency by offering a domain-specific language to provide meanings for model languages. Beyond acting as a data dictionary, ontologies establish a consistent connection between different models' viewpoints [57].

Several approaches aim to provide a common language for modeling, such as graphical, mathematical, or textual. Regardless of the approach, Holt [68] suggests the following requirements for any modeling language:

- It shall be flexible enough to allow different representations of the same information, meaning that it should accommodate different views of the information produced.
- It shall be able to represent the system at different levels of abstraction while maintaining consistency and traceability between any level.
- It shall be connected to reality, providing models with meaning and justification in the real world.

- It shall allow for the presentation of information about the system from different, yet consistent viewpoints.

3.3.1 Systems Modeling Language

A robust and standardized modeling language is considered a critical enabler for MBSE [70]. The Systems Modeling Language (*SysML*) is a general-purpose modeling language to specify, analyze, design and verify complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation to model requirements, behavior, structure, and parameters of a system [73].

SysML has become the *de facto* standard system architecture modeling language for MBSE. It is designed to accommodate a wide range of systems engineering methods and practices. However, thanks to its extensibility specific procedures may introduce additional constraints on how the language is applied [74].

The language originated from an initiative between the Object Management Group (*OMG*) and the International Council on Systems Engineering (*INCOSE*) in 2003 to adapt the Unified Modeling Language (*UML*) for SE applications. The previous version, SysML v1, was specified as a graphical UML profile. SysML v2, on the other hand, is defined as a metamodel that extends the Kernel Modeling Language (*KerML*). The current version of SysML aims to improve the language's precision, expressiveness, interoperability, consistency, and integration. This enhancement includes specific specifications for textual and graphical syntax and semantics [74].

According to Friedenthal et al. [75], the use of MBSE through formalized SysML syntax and semantics can contribute to:

- Facilitate communication between several stakeholders across the system life cycle.
- Capture and manage knowledge related to system architectures, analyses, designs, and processes.
- Provide a scalable framework for effective problem-solving.
- Furnish rich abstractions to manage size and complexity.
- Explore multiple solutions or ideas at the same time with minimal risk.
- Detect errors and omissions early in the system life cycle.

SysML handles the complete definition of a system by explicitly addressing both its structure and behavior. The structural aspect of a model defines the entities within a system and the relationships that exist among them. In contrast, the behavioral part addresses how the system evolves over time, specifying the sequence of events, the conditions under which they

occur, and the interactions involved. In essence, the structure represents “what” the system consists of and “what” it does, while the behavioral aspect constitutes “how” it operates.

In this research, we rely on models at runtime for a better adaptation to unexpected situations. For this reason, including MBSE concepts can provide solid ground for other specific models. In particular, a critical aspect is the *system architecture*, because it affects and even determines most of the properties of the system. MBSE modeling of system architectures serves as a fundamental building block for constructing and deploying reliable systems.

SysML facilitates the definition, communication and maintenance of system models. However, when systems are deployed in unpredictable environments, it is hard to ensure its behavior at the development time. We propose to exploit the model ontology at runtime to reason about which aspects of the system design can be leveraged to adapt its operation as conditions change. This approach may provide the flexibility needed to address dynamic and unexpected situations.

Chapter 4

Knowledge Representation and Reasoning

There are different strategies to pursue a better grasp of intelligence: neuroscience tries to understand how the brain processes information; mathematics seeks computation and rules to draw valid conclusions using formal logic, or handling uncertainty with probability and statistics; control theory and cybernetics aim to ensure that the system reaches desired goals, etc. [23]. One possible way that we explore in this research is to use symbolic *explicit knowledge* as a means to enhance system intelligence.

Knowledge Representation and Reasoning (KR&R) is a sub-area of *AI* that concerns analyzing, designing, and implementing ways of representing information in computational formats so that agents can use it to derive implicit facts [76]. Reasoning is the process of extracting new information from the implications of existing knowledge.

One of the challenges of getting robots to perform tasks in open environments is that programmers cannot fully predict the state of the world in advance. KR&R provides some background to reason about the runtime situation and act in consequence. In addition to adaptability, these approaches can provide an explanation; knowledge can be queried, so humans or other agents can understand why a robot acts in a certain way. Moreover, KR&R provides reusability. The robot needs information about its capabilities and about the environment in which it is involved; and this information can be shared among different agents, applications, or tasks since a knowledge base (*KB*) can be stored in broadly applicable modular chunks.

In this chapter, we explore rational agents and strategies to enhance the reusability of their programs across various applications. Using declarative approaches, along with symbolic representation as introduced in Section 4.2, can contribute to the achievement of this goal. Section 4.3 introduces the most pertinent approaches to logical reasoning, while Section 4.4 outlines the most widely adopted formalisms for encoding knowledge in robotics.

4.1 Rational Agents

An agent is an entity that acts. According to Ertel [77], an agent is a system that processes information to produce an output based on the input. There is a slight distinction between software agents, handling computational inputs and outputs, and hardware agents, utilizing sensors for inputs and actuators for outputs. Autonomous systems develop their actions in an environment in response to a percept. Basically, any autonomous agent (i) senses its state and the state of the environment, (ii) decides on which action it has to conduct to reach a goal, and (iii) executes the action to change the state accordingly.

Depending on how a system makes decisions, we can distinguish between rational and reflex agents. Reflex agents select actions depending on the current percept, for example, a vehicle that turns left if it encounters an obstacle. When we want to perform complex tasks, this reactive strategy is not adequate. In the previous example, we cannot ensure that the robot completes the patrol of a crowded room with this approach. An effective way is to use *models* to maintain a representation of the previous precepts. Models can be seen as a map from the modeled subject to the understanding the agent has about that subject. We can have a model on how the world evolves during time, e.g. the relationship among the time of the day and the outside temperature. We can also model how an action changes the world, e.g. the displacement of an object based on the applied force. Sensor modeling can also help to understand how the device will respond, for example, how rain affects the sharpness of a camera image. Models are also used at design time. Engineers use mathematical models to select components or to tune controllers. Knowledge-enabled agents can possess such models and use them at runtime to redesign themselves or at least adopt a more prosperous behavior, according to the current situation.

Rational agents are those who select their actions to achieve the best expected result [23]. This approach requires maximizing a measure of performance. To be autonomous, rational agents also need information about their goals and constraints. For example, we may want our robot to patrol a crowded room while maintaining a certain safety level, so there are requirements on maximum velocity and minimum distance from people and obstacles. *Autonomous rational agents* must derive sequences of actions to achieve a goal. If the user provides a task, such as rescuing a trapped worker, the robot first needs to find the worker, which requires it to follow a path to cover the area until it locates the person. Then, it must clear the affected zone by moving several objects, and lastly, inform about the state of the person. In addition to knowing the sequence of actions, it needs to know various parameters, such as key identifiers of a person, maximum weight the robot can carry, or maximum safe velocity for each sub-task.

Usually, knowledge about agents, their actions, and the environment is tailored for a specific application. Moreover, it is hard-coded into the program, limiting transferability to other tasks or agents with different capabilities. This approach is known as procedural, where the intended behavior is explicitly encoded in the agent program. On the contrary, knowledge-based agents use declarative approaches to abstract the control flow. In this approach, a KB stores information that enables the robot to deduce how and when to execute actions in the environment.

Unlike static procedural coding, declarative knowledge can be updated with new facts. Autonomous systems or external users can use queries to derive relevant information for the mission. Successful agents often combine declarative knowledge with procedural programming to create more efficient code [23]. This declarative knowledge is frequently encoded in logic systems to formally capture conceptualizations.

4.2 Symbolic Representation

Knowledge representation languages serve as a means of expressing assertions that are true in a particular world. In essence, they function as a set of symbolic encodings—*syntax*—to depict relevant aspects of that system. The meaning is defined as a mapping from the symbol to the world entity. This aspect is known as *semantics*. For example, consider the symbol , from it we automatically derive the obligation to halt.

A *KB* consists of *sentences* expressed with a particular syntax and semantics. In this context, a *model* represents a possible world in which sentences can be true or false. Models, as defined in previous chapters, correspond to abstract representations with reduced detail. The model of a sentence α , $M(\alpha)$ corresponds to all possible sets of assignments to variables that satisfy sentence α , i.e., all possible worlds where α is true. Note that this definition of “model” is the one used in logic. It is not necessarily translatable to other uses of “model”.

This structure provides the foundation for logical reasoning. *Reasoning* is the process of manipulating symbols to deduce new sentences from existing ones that hold in a given world. *Entailment* can be applied to derive conclusions, i.e., to carry out logical inference by checking if a sentence is true deriving it from previous ones.

Entailment captures the relationships between sentences, while *inference* is the mechanism responsible for generating new sentences by identifying and tracing these relationships. Therefore, inference is a form of reasoning.

An inference algorithm that derives only entailed sentences is sound or truth-preserving. This ensures that if the initial sentences hold in the real world, then all generated expressions will also be true. Completeness, on the other hand, denotes the property that an inference mechanism can derive any sentence that is entailed, i.e., it generates all expressions that hold in the KB.

$$\alpha \models \beta \text{ if and only if } M(\alpha) \subseteq M(\beta) \quad (4.1)$$

The entailment process is conventionally expressed as in Equation 4.1, where α represents previous sentences and β the derived ones. This equation also anchors entailment to the model, i.e., to the world these sentences represent. Logical reasoning must guarantee that the new assertions on the logical sentences mirror aspects of the world that logically follow from the configurations of the world. Figure 4.1 illustrates this relationship between representation

and world dimensions. The concept of *grounding* serves as the bridge connecting logical reasoning with the real environment in which the agent exists.

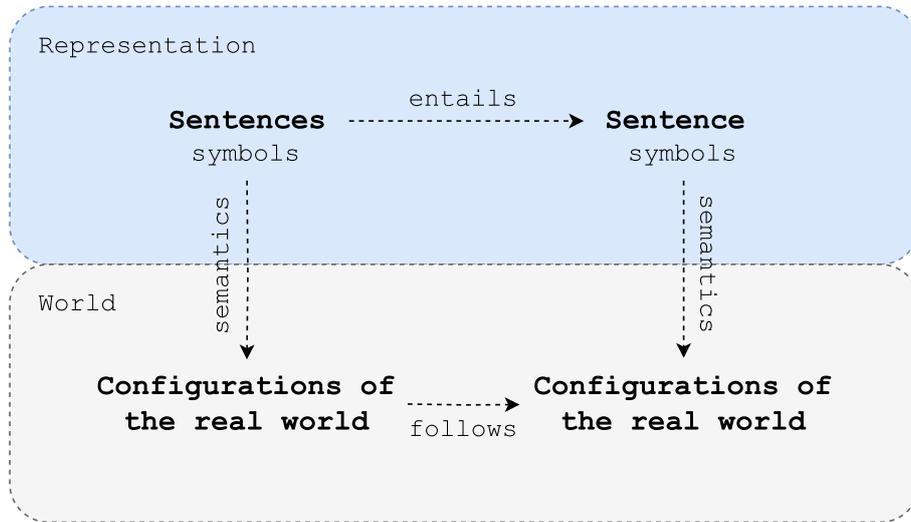


Figure 4.1: Relationship between the representation and world dimensions. Sentences represent some aspects of the world which can be derived to extract new facts through entailment. This entailment should actually follow from the aspects in the real world.

Figure 4.2 depicts a possible application of symbolic representation within a robot environment. The initial KB may contain information tailored to a specific robot task—such as the initial pose, safe distances from walls, and the next waypoint for the robot. Leveraging this information, the reasoning system can use predefined rules to deduce a recommended speed. In the robot world, a controller can use this knowledge to move the robot base.

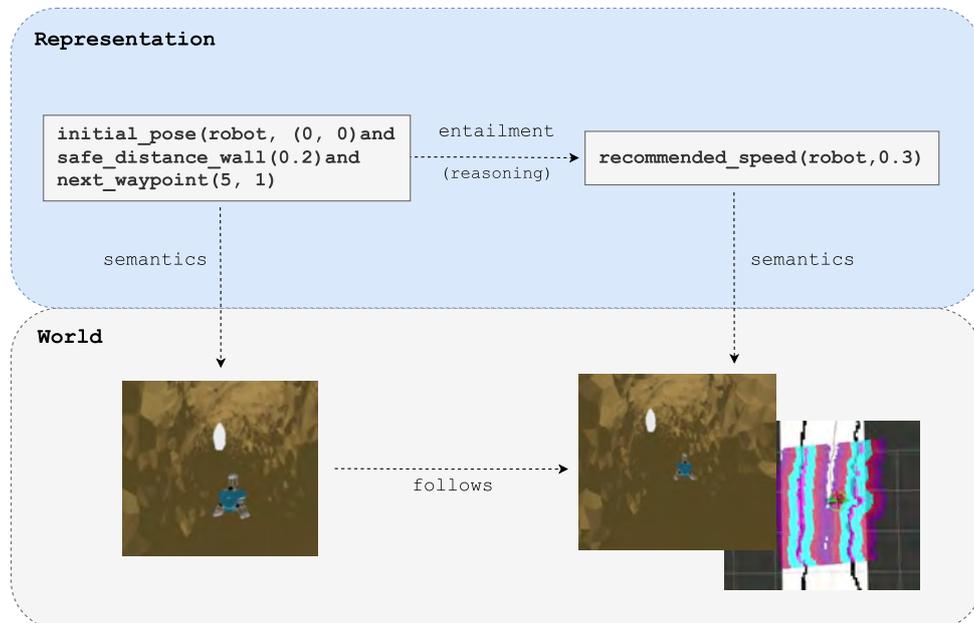


Figure 4.2: Example of a symbolic representation for a robotic task. The provided information includes the robot initial pose, the considered safe distance from a wall, and the designated next waypoint for the robot. These details are derived using predefined rules to compute the recommended speed. In the robot world, a controller can leverage this knowledge to move the robot base.

► SYMBOLIC VS. SUBSYMBOLIC REPRESENTATION

Symbolic representation involves the use of symbols to depict concepts or objects. This approach is explicit as it directly signifies objects with a clear and interpretable meaning. Additionally, it maintains transparency, wherein symbols are easily understandable and their manipulation adheres to predefined rules or algorithms.

Example: An equation is a statement in mathematics that asserts the equality of two expressions. As an illustration, the expression for position involves the double integration of acceleration, which is accomplished by substituting Equation 4.2 into Equation 4.3. This formulation breaks down the velocity $v(t)$ into the integral of acceleration $a(t)$ with respect to time, while the position $x(t)$ is defined as the integral of velocity $v(t)$ with respect to time. The constants v_0 and x_0 hold the values of the initial velocity and position, respectively.

$$v(t) = \int a(t) dt + v_0 \quad (4.2)$$

$$x(t) = \int v(t) dt + x_0 \quad (4.3)$$

However, subsymbolic representation involves encoding information without explicit symbols or language-like structures. This approach often employs distributed or connectionist models, such as neural networks, where structures map input data to output data without explicitly using symbolic representations for variables.

Example: A neural network equivalent to the equations mentioned earlier would associate acceleration values with positions. The dynamics of the system would be hidden in the weights and activation values within the network.

Subsymbolic representation is often more suitable for learning from data, especially when dealing with complex patterns that are not easily expressible through predefined rules. However, its internal representation remains inaccessible, and the resulting connections are often tailored for a specific set of inputs and outputs, making them less flexible.

4.3 Standard Forms of Logic

Logic is the discipline of formal reasoning, whereas *logics* define the different realization forms of logic. This includes standard logics, where every sentence is either true or false in each possible world, as well as “alternative” logics such as probabilistic logic or fuzzy logic, where sentences are assigned probabilities or degrees of truth, respectively. This section introduces two standard logics, propositional logic and first-order logic, establishing the groundwork for the formalisms used in this work.

4.3.1 Propositional Logic

Propositional logic (*PL*) is perhaps the most simple formalism [23]. It defines atomic sentences—propositions—and combines it with logical connectives. These operators include not (\neg), and (\wedge), or (\vee), implies (\Rightarrow), and if and only if (\Leftrightarrow).

Semantics in propositional logic establishes rules to determine the truth or falsehood of a formula. In this logic, a model assigns a truth value—either true or false—to every proposition symbol [23].

Example: Consider the proposition symbol:

`robot_A_in_corridor`

This represents an atomic sentence conveying the presence of robot A in the corridor, with its truth value being either true or false.

Sentences can become more complex. For example, if the robot is supposed to be in the kitchen instead of the corridor, two propositions and logical connectives come into play:

`robot_A_in_kitchen \wedge \neg robot_A_in_corridor`

An additional connective can be introduced to determine whether the robot has accomplished its goal:

`robot_A_in_kitchen \wedge \neg robot_A_in_corridor \Rightarrow robot_A_goal_accomplished`

The semantics of the implication symbol dictate that if the antecedent is true, in this example, both `robot_A_in_kitchen` and `\neg robot_A_in_corridor`, then the conclusion, `robot_A_goal_accomplished`, will also be true. Consequently, the notion of goal accomplishment becomes a newly derived expression within the KB.

To compute the truth of sentences (P , Q) there are five fundamental rules in any model m [23]:

- $\neg P$ is true if and only if P is false in m .
- $P \wedge Q$ is true if and only if both P and Q are true in m .
- $P \vee Q$ is true if and only if either P or Q is true in m .
- $P \Rightarrow Q$ is true unless P is true and Q is false in m .
- $P \Leftrightarrow Q$ is true if and only if P and Q are both true or both false in m .

These rules can also be expressed in truth tables, an example of which is shown in Table 4.1. Truth tables serve as a basic inference procedure, allowing computation of formulas that are

satisfiable (true for some model), valid (true for all interpretations), or unsatisfiable (not true for any interpretation). However, this method is computationally expensive, with a time complexity of $\mathcal{O}(2^n)$. Alternatively, syntactic manipulation of formulas using rules such as the modus ponens or the resolution rule is possible, but these methods are still costly.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Table 4.1: Truth tables for the five logical connectives.

Formal reasoning and proof systems usually use Horn clauses. These clauses are a subset of predicate logic with properties that permit faster derivation techniques. In particular, formulas with at most one positive literal and definite clauses are Horn clauses that contain exactly one positive literal. By adhering to these constraints, the “Selection rule driven Linear resolution for Definite clauses (*SLD*)” for derivation can be employed, allowing backward chaining and deriving only the necessary formulas.

Although *PL* is simple and intuitive, it has limitations in expressiveness and generality. The formalism discussed below takes a further step to overcome these issues by incorporating more powerful relationships among objects.

4.3.2 First-Order Logic

First-order logic (*FOL*) serves as an extension of *PL* to enhance expressiveness. While *PL* provides a factored representation, *FOL* introduces more structure by incorporating elements from natural language. Fundamental syntactical elements include symbols for [23]:

- **Object:** Represents nouns in natural language and constitutes constant symbols to represent entities in the world.
- **Relation:** Represents properties or connections between objects. These often correspond to verbs, adverbs, and adjectives in natural languages. A relation is a set of tuples that relate objects.
- **Function:** Certain types of relationship are related to exactly one object. Functions are relations in which there is only one value for a given input.

With these elements, more complex assertions can be expressed. For example, the sentence $3 + 4 = 7$ has as objects the numbers 3, 4 and 5, a relation represented by the equal symbol $=$, and a function denoted by the plus symbol $+$. Applying the $+$ function to the objects 3 and 4 produces a new object, $3 + 4$, which is equivalent to 7. Another example is the

sentence “John eats with a fork,” where “John” and “fork” represent objects, and “eats with” is a relation.

FOL constructs formulas through terms, where a *term* is a logical expression that refers to an object, variable, or constant. Terms can also be defined by functions, as in the example above. The expressiveness of FOL arises from predicates that represent relationships between terms. These predicates can be used with logical connectors, such as those used in PL, as well as quantifiers such as universal quantifier (\forall), existential quantifier (\exists), and equality ($=$). FOL quantifiers are used to formulate *axioms*, which provide factual information from which valuable conclusions can be derived.

Example: The statement “all horses are fast” can be represented with the universal quantifier as:

$$\forall x \text{ horse}(x) \Rightarrow \text{fast}(x)$$

This can be read as “for all elements X, if X is a horse, then it is fast.”

On the other hand, using the existential quantifier, the statement would be “there exists at least one horse that is fast,” represented as:

$$\exists x \text{ horse}(x) \Rightarrow \text{fast}(x)$$

This sentence can be read as “there exists some element X such that if X is a horse, then it is fast.”

Beyond expressiveness, an important distinction between PL and FOL lies in their ontological commitments—the assumptions about reality in each logic. PL assumes that predicates are facts that either hold or not in that world, while in FOL, it is the relationships between objects that hold or do not hold.

Moving on to reasoning, the use of quantifiers and variables in FOL makes the rules of PL not applicable in general. FOL reasoning involves variable *substitution* which consists of defining the reasoning process as a search in the space of possible substitutions of variables, corresponding to the Herbrand universe¹. *Unification* is a specific case of substitution, occurring when there is a substitution for all variables that makes the literals equal. *Propositionalisation* is another possibility of reasoning, involving the elimination of variables by substitution to generate an expression in PL. However, the most efficient and automated calculus in FOL is *resolution* [78], where the knowledge base *KB* is combined with the negated query $\neg Q$ in an attempt to reach a contradiction. In other words, the resolution aims to prove that $KB \wedge \neg Q \vdash \emptyset$, where \vdash represents the derivation, and \emptyset denotes the empty clause. For more details on FOL reasoning, refer to Sections 3.4 and 3.5 in Ertel’s book [77].

Lastly, PL is decidable, meaning that all true entailments can be found and all false entailments can be refuted using truth tables. However, FOL is semi-decidable, implying that there exists

¹Herbrand’s theorem has played a vital role in the development of automated reasoning [23]. This theorem defines that if a set of clauses, *S*, is unsatisfiable, then there exists a particular set of ground instances of the clauses of *S* such that this set is also unsatisfiable. The Herbrand universe of *S*, is the set of all ground terms constructible from (i) the function symbols in *S*, if any; (ii) the constant symbols in *S*, if any; if none, then a default constant symbol, *S*.

an algorithm to find true entailments, but it might not halt while searching for a refutation for a false entailment. For this reason, formalisms are often defined using *decidable fragments of FOL* such as most Description Logics.

4.3.3 Description Logics

Description Logics (*DL*) constitute a family of formal knowledge representation languages, positioned as a solution between PL and FOL. While PL is decidable, it lacks the expressiveness to handle unbounded environments. On the other hand, FOL is more descriptive but semi-decidable according to Gödel’s completeness theorem—which demonstrates that there are true arithmetic sentences that cannot be proved [23]. DL strikes a balance between expressivity and scalability by employing decidable fragments of FOL, allowing for reasoning with more expressiveness than PL. Note that DL specifically addresses challenges related to the search space problem, decidability, and incompleteness, distinct from logics dealing with uncertainty (e.g., fuzzy logic, probabilistic logic) or different necessities and possibilities (e.g., modal logic).

All DLs are based on three kinds of entities [79]:

- **Individuals:** Similar to FOL constant objects, these are interpreted as specific elements in the real world. For example, a robot *A* in an experiment with a sensor camera001.
- **Concepts:** According to the unary predicates of FOL, these encapsulate generalized entities such as the concept of a *Robot* or *RobotPart*.
- **Role Names:** Corresponding to binary predicates in FOL, these include roles such as *hasComponent*, which indicates the parts that constitute the robot—connecting instances of *Robot* and instances of *RobotPart*.

This knowledge is typically organized into three sets within each *KB*. The *RBox* stores the interdependencies between roles, the *TBox* refers to the terminological knowledge from concepts and the *ABox* denotes all the assertional knowledge about individuals or entities.

The simplest form of DL is denoted as \mathcal{ALC} , representing *Attributive (Concept) Language with Complements*. However, our implementation specifically focuses on \mathcal{SROJQ} , as conceptualized by Horrocks et al.[80]. According to Rudolph [79], the \mathcal{SROJQ} acronym can be decomposed as follows:

- \mathcal{SR} : Denotes an extension of \mathcal{ALC} with all types of *RBox* axioms and self-concept.
- \mathcal{O} : Specifies the support for nominal concepts.
- \mathcal{J} : Stands for the inverse of the allowed role.
- \mathcal{Q} : Indicates the support for arbitrary qualified number restrictions.

\mathcal{SROIQ} serves as the logical foundation for OWL 2 DL, one of the most widely used languages for the Semantic Web, further detailed in Section 4.4.3. This language is used for making machine-readable the models conceptualized in Chapter 7.

4.4 Representational Paradigms in Robotics

Robots require machine-understandable formalisms to effectively handle KBs. Systems should be capable of extracting, reasoning about, and updating their knowledge. In the context of intelligent robots, the following section introduces the most prevalent paradigms for organizing and representing knowledge.

4.4.1 Prolog

Prolog, an abbreviation for “Programming in Logic,” stands as one of the most widely used declarative programming languages. Many knowledge-based systems have been developed in various domains, such as legal, medical, and financial, using this language [23]. Its origins can be traced to the 1970s when pioneers such as Kowalski and Colmerauer conceptualized its procedural interpretation which culminated in the Prolog language [81]. This paradigm defines problems and solutions as logical axioms.

Prolog programs consist of sets of definite clauses based on decidable fragments of FOL, although with some changes in notation conventions. Notably, Prolog reverses the convention of FOL, using uppercase letters for variables and lowercase letters for constants; clauses are written in backwards. For example, the FOL expression $A \wedge B \Rightarrow C$ is represented in Prolog as $C :- A, B$.

Its implementations incorporate specific features to make them suitable for computational systems. In defining sentences, it adheres to the *closed-world assumption*, meaning that predicates explicitly defined as true are considered true. Moreover, Prolog operates on the *unique name assumption*, considering that distinct names always refer to different entities in the world. The execution of Prolog programs uses depth-first backward chaining, implying that clauses are attempted in the order they appear in the KB.

Several tools enhance Prolog implementations, such as GNU Prolog², a compiler and classical interactive interpreter; or SWI Prolog³, which offers a comprehensive application environment and scalable multithreading support for embedded implementations. Moreover, the Robot Operating System (ROS) has a dedicated package named *rospolog*⁴, which provides a bidirectional interface between SWI Prolog and ROS. This integration facilitates the deployment of Prolog in robotic infrastructures.

²<http://www.gprolog.org/>

³<https://www.swi-prolog.org/>

⁴<https://github.com/KnowRob/rospolog>

Overall, Prolog continues to be a popular tool within the domain of *AI* due to its ability to allow transparent and concise program development with a compromise between declarativeness and efficiency. However, it is important to consider that Prolog lacks procedural constructs typically found in other languages. For instance, there are no checks for infinite recursion, potentially causing certain programs that seem logically valid to fail in implementation.

4.4.2 Planning Domain Definition Language

Planning Domain Definition Language (*PDDL*) constitutes a family of languages designed to support the declarative representation of planning [82]. Planning is essential for autonomous agents to determine a sequence of actions that lead to achievement of the goal. Although classical approaches often rely on ad-hoc heuristics and explicit codifications, *PDDL* offers a structured framework to effectively represent problems and complex action schemas.

Although *PDDL* may not be considered a traditional *KR&R* language in the strict sense, it does collect and structure explicit knowledge about entities, actions, and relationships in the universe of discourse. This information is organized in two description files: the *domain description*, which stores object types, feasible actions with their preconditions and effects, and any domain-specific predicates; and the *problem description*, which stores specific instances of a planning problem, detailing the initial state, goal state, and a set of objects in the world. The result is a plan—a sequence of actions driving the transition from the initial state to the goal.

To address specific limitations, various extensions of *PDDL* have emerged over time, enhancing the language's syntax and expanding its capabilities. These extensions accommodate features such as time, resources, percepts, contingent plans, and hierarchical plans [23]. Notable extensions include:

- Fully Observable Non-Deterministic Planning (*FOND*): This extension offers a more flexible representation of action effects, generating decision trees rather than sequences of actions to handle multiple potential outcomes [83].
- Probabilistic *PDDL* (*PPDDL*): This introduces actions that can lead to several states with associated probabilities, addressing uncertain outcomes through probabilistic effects [84].
- Planning with Partial Observability and Sensing (*PPOS*): Designed to handle uncertainties in both initial states and action outcomes, *PPOS* enables planners to consider all potential states an agent might encounter [85].
- Multi-Agent Planning: This extension merges the traditional single agent *PDDL* approach with the dynamics of multi-agent systems, providing a unified planning solution [86].

In general, planning initiatives such as the European project AIPlan4EU⁵ play a crucial role in operationalizing the use of explicit knowledge in robot applications to find optimal solutions. PDDL, along with its extensions, is seamlessly integrated into various robotic software tools for *ROS*, including PlanSys2 [87], ROSplan [88], Merlin2 [89], and SkiROS2 [90]. However, it remains a declarative language with a few procedural constructs and limitations in expressivity, as described above.

4.4.3 Ontologies and the Ontology Web Language

The construction of KBs can benefit from the examination of the categories of objects present in a domain. The explicit specification of a conceptualization is referred to as an *ontology* [91]. A conceptualization is an abstract, simplified perspective of a specific area of interest, describing a shared vocabulary at both textual and conceptual—formal—level.

Ontologies play a crucial role in technical domains as they facilitate the sharing of common concepts among heterogeneous agents, including various types of robots and human users. This shared understanding helps enhance communication without ambiguity, contributing to explainability in the decision-making processes of artificial agents, or enabling the sharing of a particular perspective on a domain.

Entities in ontologies revolve around the categorization of objects into *classes*, which can be seen as categories within the domain of discourse. While interaction with the world involves specific objects, referred to as *individuals* or instances, much reasoning occurs at the class level.

Classes can possess *properties* that characterize the collection of objects they represent or be *related* to other classes. Additionally, they are organized through inheritance, expressed by the *subclass* relation.

Ontologies have gained importance with the Semantic Web. According to the original vision, the availability of machine-readable metadata would enable automated agents to retrieve meanings from available information on the web [92]. However, their potential goes beyond this. For technological systems, ontologies separate the domain knowledge from the procedural and problem-solving knowledge, allowing all this knowledge to be reused in other programs. Moreover, it enables the solution of concrete problems through generic techniques and algorithms that can be shared among different setups. To achieve this, ontologies propose a syntax and semantics that are intuitive to human users and compatible with existing (Web) standards while providing enough expressive power to define relevant concepts in sufficient detail but without being overly expressive to make reasoning infeasible. This is where the Ontology Web Language comes into play.

⁵<https://www.aiplan4eu-project.eu/>

Example:

Consider the assertion of the class *Food* with the property *edible*. According to this assertion, every instance of *Food*, such as the specific pie *pie_01*, is considered *edible*.

In this hierarchy, *Fruit* is a subclass of *Food*, and *Apple* is also a subclass of *Food*. Given this hierarchy, we can deduce that every instance of *Fruit* and *Apple* inherits the properties of their superclass *Food*.

Since *Apple* is a subclass of *Food*, every individual apple, such as *red_apple_01*, inherits the *edible* property. Therefore, we can conclude that *red_apple_01* is also considered *edible* based on its classification within the ontology hierarchy. Figure 4.3 represents this ontology.

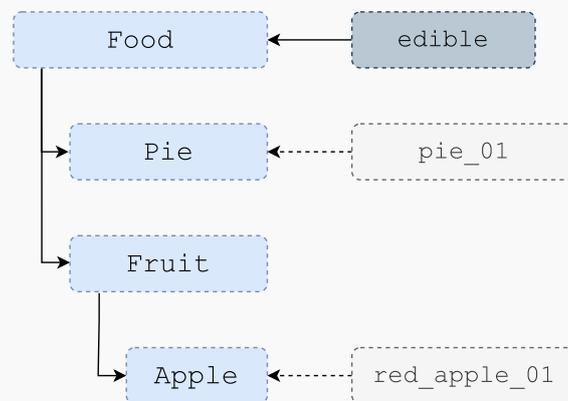


Figure 4.3: Example of the Apple ontology.

► ONTOLOGY WEB LANGUAGE

The Ontology Web Language (*OWL*) is not a programming language like Prolog but rather a family of languages based on *Description Logics (DL)*. It is designed for applications that need to process information rather than simply present information to humans. Therefore, OWL facilitates greater machine interpretability and supports various coding formats such as XML, RDF, and RDF Schema (RDF-S). The World Wide Web Consortium [93] has specified it as the cornerstone of the semantic web.

OWL not only provides a language to describe concepts and structure them in a hierarchy; it also allows the use of logical reasoners to check consistency and classify them to build new assertions. This reasoning derives implied relations and hierarchies, testing non-contradictory concepts to ensure ontology consistency. In contrast to Prolog, OWL adopts the *open-world assumption*, meaning that a statement can be considered true whether it is known or not—only explicitly false predicates are considered false.

This ontology language seamlessly integrates with the Semantic Web Rule Language (*SWRL*), based on OWL DL and OWL Lite sublanguages of OWL. SWRL implements Horn-like rules to express complex relationships and conditions based on the information in the KB. These rules enable more advanced reasoning and inferencing capabilities for ontological applications.

Regarding accessibility, OWL can be encoded by hand using text editors or using graphical editors such as Protégé [94] for a user-friendly environment. OWL can be integrated into robotic software through Application Programming Interfaces (APIs) such as Jena Ontology API⁶ or OWLAPI⁷ implemented in Java or OWLREADY [95] implemented in Python. Additionally, there are reasoners such as FaCT++⁸, Pellet⁹, or Hermit¹⁰ that can be invoked directly from code or ontology editors.

4.4.4 Graph-Based Representation

The concept of a *knowledge graph* applied to computation can be traced back to Richard H. Richens in 1956, where it was initially conceived as an "interlingua" for machine translation of natural languages [96]. However, it has gained popularity since 2012 [97], especially with its implementation in major industries such as Google Knowledge Graph [98] and other platforms such as Microsoft [99], Facebook [100], and IBM [101], among others [102].

Although there is no universally agreed upon definition, Hogan et al. [103] provides the following:

Knowledge graph can be defined as a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities.

A knowledge graph essentially represents data as a graph, using a set of nodes connected by edges to capture relations. Because information is not restricted to a rigid schema, the data can evolve with more flexibility [104]. There are five basic types of graphs used to represent the data according to Hogan et al. [103]:

- **Directed Edge-labeled Graphs:** This is the fundamental type of knowledge graph, defined as a set of nodes with directed labeled edges between them. This approach offers flexibility, particularly when integrating new sources of data, since incomplete information can be represented by simply omitting unknown edges. An example of this type is the Resource Description Framework (*RDF*) [105]. Note that RDF is a core model of the Semantic Web, which is often used with additional restrictions imposed by *OWL* to represent ontologies.
- **Heterogeneous Graphs:** In these graphs, each node and edge have an assigned type rather than specifying types as a special node-edge relation.
- **Property Graphs:** These graphs allow the association of property-value pairs with nodes and edges. This leads to a more concise and intuitive representation. However, this

⁶<https://jena.apache.org/documentation/ontology>

⁷<https://github.com/owlcs/owlapi/wiki>

⁸<http://owl.cs.manchester.ac.uk/tools/fact>

⁹<https://github.com/stardog-union/pellet>

¹⁰<http://www.hermit-reasoner.com/>

model requires more complex query languages and formal representation. Property graphs are popular in graph databases, such as Neo4j¹¹.

- **Graph Dataset:** This consists of a set of pairs of IDs and graphs. It is especially useful for managing and querying data from multiple sources.
- **Other Graph Data Models:** This approach provides greater flexibility, as nodes may contain individual edges or even nested graphs—hypernodes—and edges may connect sets of nodes rather than pairs.

Queries in knowledge graphs often involve traversing relationships and patterns within the interconnected data, especially on a vast and heterogeneous dataset. There are various languages for querying graphs, such as the SPARQL query language for RDF graphs [106]; Cypher [107], Gremlin [108], and G-CORE [109], which are used for querying property graphs. This approach contrasts with querying ontologies, where the focus is on class hierarchies, property restrictions using logical rules, and constraints.

Although knowledge graphs offer an effective means of capturing knowledge, particularly in data-intensive contexts, they may lack the structure and formal traits found in the methods described in the previous sections. Formal implementations aim to be precise and provide traceable reasoning. On the other hand, knowledge graphs foster a more dynamic representation of information. They excel in representing relationships and connections within diverse datasets, providing a flexible and interconnected framework.

However, ontological representations such as those based on OWL can be seen as knowledge graphs with well-defined meaning [110]. Moreover, ontologies and rules can be used to provide semantics to the terms in the graph. In this research, we focus on formalizing engineering concepts and using them as a means to increase dependability of autonomous systems. For this reason, we select OWL ontologies as the logical framework under which we develop our solution.

¹¹<http://neo4j.org>

Chapter 5

Ontologies for Robotics

This chapter presents an analysis of both popular and recent works that use ontologies to support robot autonomy, with a focus on identifying the most extended capabilities expressed in these ontologies and understanding how they are used at runtime. Section 5.1 introduces foundational ontologies, along with the most widely used domain ontologies in the field of robotics. Section 5.2 provides a systematic review of application ontologies in robotics, comparing how the functional competencies of autonomous robots are represented and how this information is used for action selection.

5.1 Fundamental and Domain Ontologies

Ontological systems can be classified according to several criteria. We distinguish three levels of abstraction based on Guarino's hierarchy [111]: upper level, domain, and application. *Upper-level* or foundational ontologies conceptualize general terms such as object, property, event, state, and relations such as parthood, constitution, participation, etc. *Domain* ontologies provide a formal representation of a specific field that defines contractual agreements on the meaning of terms within a discipline [112]; these ontologies specify the highly reusable vocabulary of an area; the concepts, objects, activities, and theories that govern it. *Application* ontologies contain the definitions required to model knowledge for a particular application; information about a robot, in a specific environment, describing a particular task. Note that the environment or task knowledge could be a subdomain ontology, depending on the reusability it allows. In fact, the progress from upper-level to application ontologies is a continuous spectrum of concepts' sub-classing [113], with somewhat arbitrary divisions into levels of abstraction reified as ontologies.

Perhaps the most extended upper-level ontology is the Suggested Upper Merged Ontology (*SUMO*) [114], [115]¹. *SUMO* is the largest open-source ontology that has expressive formal definitions of its concepts. There are domain ontologies that are part of *SUMO* for Medicine, Economics, Engineering, and many other topics. This formalism uses Standard Upper Ontology Knowledge Interchange Format (*SUO-KIF*), a logical language to express

¹<https://www.ontologyportal.org>, <https://github.com/ontologyportal>

concepts with higher-order logic (a logic beyond the expressiveness of first-order logic) [116].

Another relevant foundational ontologies for this research is the Descriptive Ontology for Linguistic and Cognitive Engineering (*DOLCE*), described as a “ontology of universals” [117]; which means that it has classes but not relations. It aims to capture the ontological categories that underlie natural language and human common sense [118]. The taxonomy of the most basic categories of particulars assumed in *DOLCE* includes, for example, abstract quality, abstract region, agentive physical object, amount of matter, temporal quality, etc. Although the original version of the few dozen terms in *DOLCE* was defined in FOL, it has since been implemented in OWL; most extensions of *DOLCE* are also in OWL.

The *DOLCE+DnS Ultralite ontology*² (*DUL*) simplifies some parts of the *DOLCE* library, such as the names of classes and relations with simpler constructs. The most relevant aspect of *DUL* is perhaps the design of the ontology architecture based on patterns.

Other important foundational ontologies are the Basic Formal Ontology (BFO) [119], the Bunge-Wand-Weber Ontology (BWW) [120; 121] and the Cyc Ontology [122]. BFO focuses on continuant entities involved in a three-dimensional reality and occurring entities, which also include the time dimension. BWW is an ontology based on Bunge’s philosophical system that is widely used for conceptual modeling [123]. Cyc is a long-term project in artificial intelligence that aims to use an ontology to understand how the world works by trying to represent implicit knowledge and perform human-like reasoning.

5.1.1 Robotic Domain Ontologies

Ontologies have gained popularity in robotics with the growing complexity of actions that systems are expected to perform. A well-defined standard for knowledge representation is recognized as a tool to facilitate human-robot collaboration in challenging tasks [124]. The IEEE Standard Association of Robotics and Automation Society (*RAS*) created the Ontologies for Robotics and Automation (*IEEE ORA*) working group to address this need.

They first published the Core Ontology for Robotics and Automation (*CORA*) [125]. This standard specifies the most general concepts, relations, and axioms for the robotics and automation domain. *CORA* is based on *SUMO* and defines what a robot is and how it relates to other concepts. For this, it defines four main entities: robot part, robot, complex robot, and robotic system. *CORA* is an upper-level ontology currently extended in the IEEE standard 1872-2015 [126] with other sub-ontologies as *CORAX*, *RPART*, and *POS*.

CORAX is a sub-ontology created to bridge the gap between *SUMO* and *CORA*. It included high-level concepts the authors claimed to not be explicitly defined in *SUMO* and particularized in *CORA*; in particular those associated with design, interaction, and environment. *RPARTS* provides notions related to specific kinds of robot parts and the roles it can perform, such as grippers, sensors, or actuators. *POS* presents general concepts associated with spatial

²http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite

knowledge such as position and orientation—represented as points, regions, and coordinate systems.

However, CORA and its extensions are intended to cover a broad community, so their definitions of ambiguous terms are based purely on necessary conditions without specifying sufficient conditions [124]. For this reason, concepts in CORA require to be specialized according to the needs of specific subdomains or robotics applications.

CORA, like most of the other application ontologies considered here, is defined in a language of very limited expressiveness, mostly expressible in OWL-Lite, and therefore limited to simple classification queries. Although it is based upon upper-level terms from SUMO, it recreated many terms that could have been used directly from SUMO. Moreover, given its choice of representation language, it did not use the first- and higher-order logic formulas from SUMO, limiting its reuse to just the taxonomy.

The IEEE ORA group spawned the Robot Task Representation sub-group to produce a *middle-level ontology* with a comprehensive decomposition of tasks, from goal to sub-goals that enables humans or robot to accomplish its expected outcome at a specific instance in time [127]. It includes a definition of tasks and its properties, and terms related with which performance capabilities are required to accomplish them. Moreover, it covers a catalog of tasks demanded by the community, especially in industrial processes [128].

This working group also created three additional subgroups for more specific domain knowledge. In concrete, for Autonomous Robots Ontology, Industrial Ontology, and Medical Robot Ontology; of which only the first one is active. The Autonomous Robot sub-group (*AuR*) specializes CORA and its associated ontologies in the domain of autonomous robots, including, but not limited to, aerial, ground, surface, underwater, and space robots.

They provided the IEEE Standard for Autonomous Robotics Ontology [129] with an unambiguous identification of the basic hardware and software components necessary to provide a robot or a group of robots with autonomy. It was conceived to serve for different purposes, such as to describe the design patterns of AuR systems; to represent AuR system architectures in a unified way; or as a guideline to build autonomous systems consisting of robots operating in various environments.

In addition to the developments of the IEEE ORA working group, there are other relevant domain ontologies such as OASys and SOMA. The Ontology for Autonomous Systems (*OASys*) [130] captures and exploits concepts to support the description of any autonomous system with an emphasis on the associated engineering processes. It provides two levels of abstraction systems in general and autonomous systems in particular. This ontology connects concepts such as architecture, components, goals, and functions with the engineering processes required to achieve them.

On the other hand, the Socio-physical Model of Activities (*SOMA*) for Autonomous Robotic Agents represents the physical and social context of everyday activities to facilitate accomplish tasks that are trivial for humans [131]. It is based on DUL, extending their concept to different event types such as action, process, and state; the objects that participated in the activities, and the execution concept. It is worth to mention that SOMA was intended to be used in

runtime along with the concept of narratively-enabled episodic memories (*NEEMs*), which are comprehensive logs of raw sensor data, actuator control histories, and perception events, all semantically annotated with information about what the robot is doing and why using the terminology provided by SOMA.

The relation between upper-level ontologies and domain ontologies is a relation of progressive domain focalization [113]. The frameworks described below are mostly specializations of these general robotic ontologies and other foundations.

5.2 A Survey on Applications Using Ontologies for Robot Autonomy

In this section, we provide an analysis and comparison of projects that exploit ontologies to enhance robot autonomy. Specifically, we examine the representation of processes introduced in Section 2.3 that are involved in autonomous operation, including perception, categorization, decision-making, prediction and monitoring, communication, etc. We focus on works that use this knowledge for action selection. As represented in Figure 5.1, we analyze how knowledge representation is included in an autonomous agent when interacting with the environment. The knowledge captured by the ontology may be related to any entity, but is intended to support the decision-making process within the agent. This section first introduces the review process and related surveys to subsequently provide comparison criteria between the works. The review is divided in four domains: manipulation, navigation, social and industrial. Lastly, we include a discussion on relevant findings.

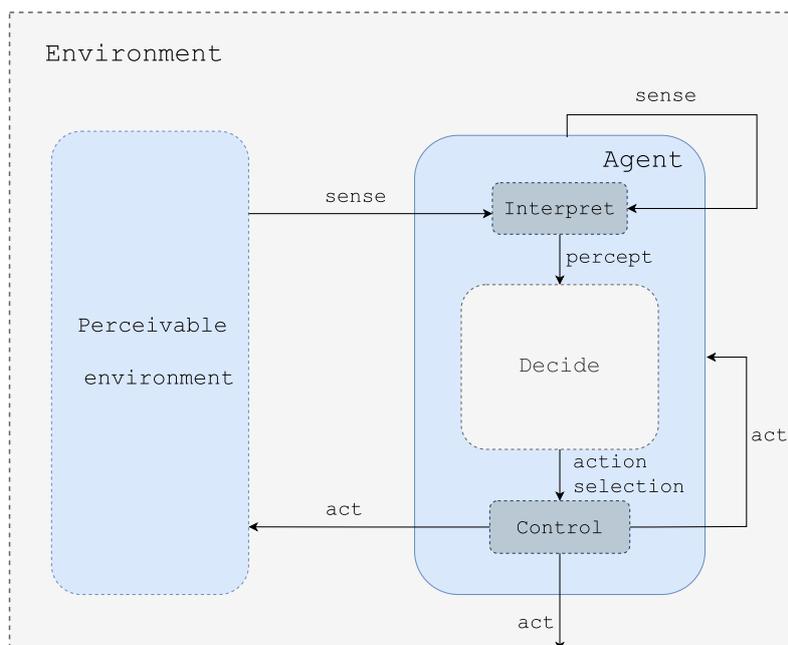


Figure 5.1: Conceptualization of an autonomous agent interacting with the environment and upon itself.

5.2.1 Review Process

In this chapter, one of our main objectives is to conduct a systematic analysis of relevant and recent projects that use ontologies for autonomous robots. The first step in our review process is to search for relevant keywords in scientific databases. Specifically, we have used the most extended literature browsers, *Scopus*³ and *Web of Science*⁴. These databases provide a vast array of peer-reviewed literature, including scientific journals, books, and conference proceedings. It also has a user-friendly interface for storing, analyzing, and displaying articles. The search was done in terms of title, abstract, or keywords containing the terms robot, ontology, and plan, behavior, adapt, autonomy, or fault. In practice, we have used the following search string.

```
TITLE-ABS-KEY (robot AND ontology AND (plan*
OR adapt* OR behavior* OR autonom* OR fault*))
```

The required terms are “ontology” and “robot” as they are the foundations of our research. Using this restriction may seem somewhat limiting, because there could be knowledge-based approaches to cognitive robots that were not based on ontologies. However, in this review, we are specifically interested in the use of ontologies for this purpose, hence the strong requirement regarding the “ontology”. We also target at least one keyword related to (i) selection and arrangement of actions—autonomous, planning, behavior—or (ii) overcoming contingencies—fault, adapt. Note that we use asterisks to be as flexible as possible with the notation.

This search returned 695 articles after removing duplicates; but as we point to the most relevant and/or recent articles, we apply inclusion criteria based on the publication date and the number of citations. We only include articles widely cited, with 20 or more citations, before 2010; articles between 2010 and 2015 with 5 or more citations; and articles between 2015 and 2018 with 2 or more citations. All articles from 2018 onward were included. With these criteria, we have reduced the list to 351 articles. By applying these criteria, the selection process aims to identify articles that are not only recent, but have also demonstrated impact and influence within their respective publication periods.

Then we analyze the content of the articles. We include works in which the main point of the article is the ontology. In particular, (i) proposes or extends the ontology, and (ii) uses the ontology to select, adapt, or plan actions. We recall a number of articles on how to use ontologies to encode simulations, on-line generators of ontologies, or ontologies only used for conversation, perception, or collaboration without impact in robot action. The application of these criteria produces 26 relevant articles.

Finally, we complement our search with articles from related surveys described in Section 5.2.2. This ensures that we include all relevant and historical articles in the field with a snowball process. In this step, we obtain 22 more articles; some of them were already included in the previous list, and others did not meet our inclusion criteria. This results in

³<https://www.scopus.com>

⁴<https://www.webofscience.com/wos/allldb/basic-search>

the inclusion of 6 more papers; resulting in a total of 32 articles in deep review. The whole process is depicted in Figure 5.2 with a Preferred Reporting Items for Systematic reviews and Meta-Analyses (PRISMA) flow diagram.

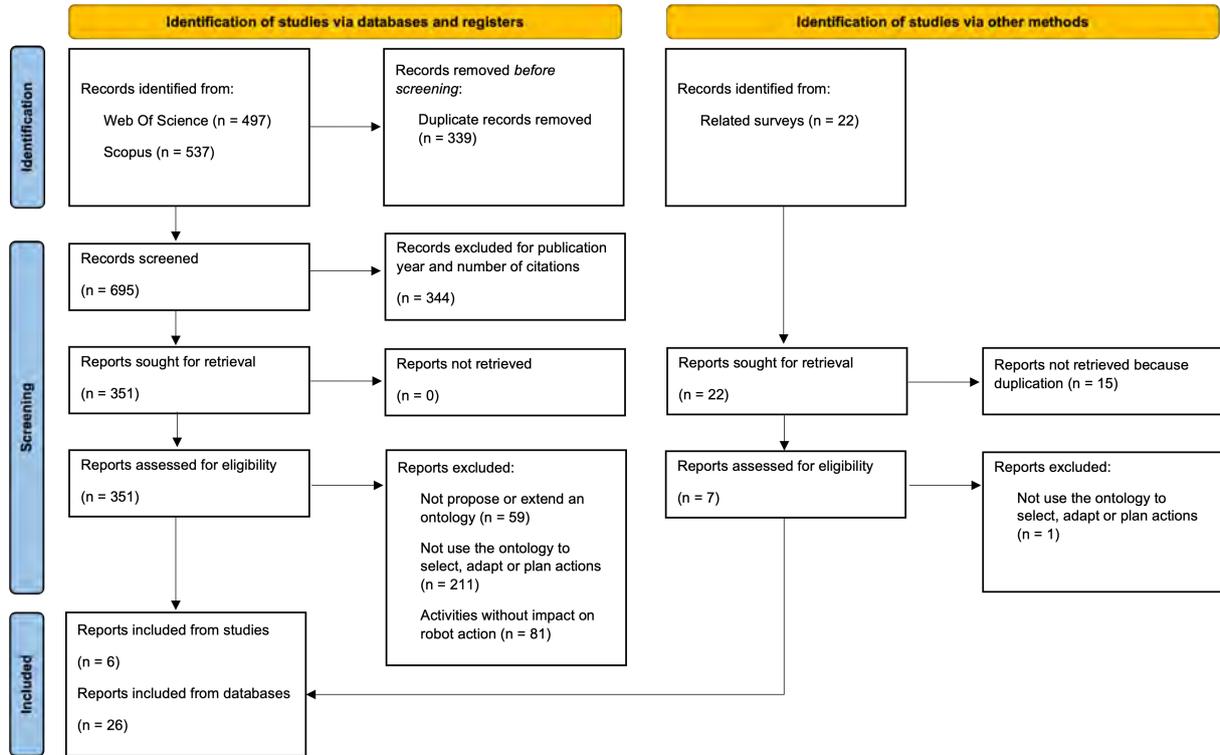


Figure 5.2: PRISMA flow diagram, adapted from [132].

5.2.2 Related Surveys

This section evaluates a variety of comparative studies and surveys on the application of KR&R to several robotic domains. These works address robot architecture, as well as specific topics such as path planning, task decomposition, and context comprehension. Nevertheless, our approach transcends these specific domains, aiming to offer a more comprehensive view of the role of ontologies in enhancing robot understanding and autonomy.

Gayathri et al. [133] compares languages and planners for robotic path planning from the KR&R perspective. They discuss ontologies for spatial, semantic, and temporal representations and their corresponding reasoning. In [134], the same authors expand their review by focusing on DL for robot task planning.

Closer to our perspective are articles that compare and analyze different approaches for ontology-based robotic systems; specifically, those articles that focus on what to model in an ontology. Cornejo-Lupa et al. [135] compares and classifies ontologies for Simultaneous Localization and Mapping (SLAM) in autonomous robots. The authors compare domain and application ontologies in terms of (i) robot information such as kinematic, sensor, pose, and trajectory information; (ii) environment mapping such as geographical, landmark, and uncertainty information; (iii) time-related information and mobile objects; and (iv)

workspace information such as domain and map dimensions. They focus on one important but specific part of robot operation, autonomous navigation; and, in a particular type of robot, mobile robots.

Manzoor et al. [136] compares projects based on application, ideas, tools, architecture, concepts, and limitations. However, this article does not go deeply into how architecture in the compared frameworks supports autonomy. Their review focuses on objects, environment map, and task representation from an ontology perspective. It does not compare the different approaches regarding the robot's self-model. It also does not tackle the mechanisms and consequences of such knowledge to enhance the robot's reliability.

Perhaps the most relevant review from our perspective is the one by Olivares-Alarcos et al. [137]. They analyze five of the main projects that use KBs to support robot autonomy. They ground its analysis in (i) ontological terms, (ii) the capabilities that support robot autonomy, and (iii) its application domain. Moreover, they provide a discussion of robot and environment modeling. This work is closely related to the development of the recent IEEE Standard 1872.2 [129].

Our approach differentiates itself from previous reviews because we focus on modeling not only robot actions and its environment but also engineering design knowledge. This type of knowledge is not often considered but provides a deeper understanding of the robot's components and its interaction, design requirements, and possible alternatives to reach a mission. We also base our comparison on explicit knowledge of the mission and how we can ensure that it satisfies the user-expected performance. For this reason, we focus on works that select, adapt, or plan actions. Our approach is more flexible on inclusion criteria to analyze a variety of articles, even if they are partial or its ontologies are not publicly available. We have adopted this wide perspective to draw a general picture of different approaches that build the most important capabilities for dependable autonomous robots.

5.2.3 Comparison Criteria

This section introduces the classification criteria used to review application ontologies to improve robot autonomy. We compare how the functional competencies of autonomous robots are represented in ontologies to later use them for action selection. These capabilities are listed below, with detailed descriptions available in Section 2.3. Note that we have analyzed some categories together, as the boundaries between them partially overlap.

- Perception and Categorization
- Decision-making and Planning
- Prediction and Monitoring
- Reasoning
- Execution

- Communication and Coordination
- Interaction and Design
- Learning

The classification of these competencies serves to reveal their incorporation into KBs, identify potential underrepresented elements, and explore the contributions of knowledge structures to the decision-making process. The following sections review and compare specific projects to find individual contributions to robot ontologies application field.

We have organized the analysis of the projects under review based on the main areas addressed—manipulation, navigation, social, and industrial—in which they have been deployed. However, it is important to note that many of these works have broader implications and can be applied across different domains. For instance, manipulation and navigation skills are often required in both social and industrial robot applications. This overlap reflects the multifaceted nature of robotic competencies.

While each reviewed work is categorized by its primary domain of interest, most of them possess a broader scope and applicability. The analysis of each project includes a description and a discussion of the above criteria. Additionally, a comparative table is provided for each domain, highlighting the most relevant aspects of each capability.

5.2.4 Manipulation Domain

In the domain of robotics, manipulation refers to the control and coordination of robotic arms, grippers, and other mechanical systems to interact with objects in the physical world. This includes a wide range of applications, from industrial automation to tasks in unstructured environments, such as household chores or healthcare assistance. In this section, most of the works under analysis focus on domestic applications for manipulation.

► KNOWROB AND KNOWROB-BASED APPROACHES

KnowRob⁵ is a framework that provides a KB and a processing system to perform complex manipulation tasks [138], [139]. KnowRob2⁶ [140], represents the second generation of the framework and serves as a bridge between vague instructions and the specific motion commands required for task execution.

KnowRob2's primary objective is to determine the appropriate action parameterization based on the required motion and identify the physical effects that need to be achieved or avoided. For example, if the robot receives a query on how to pick up a cup to pour out its contents, the KB retrieves the necessary action of pouring, which includes a sub-task to grasp the

⁵<http://KnowRob.org>

⁶<https://github.com/KnowRob/KnowRob>

source container. Subsequently, the framework queries for pre-grasp and grasp poses, along with grasp force, to establish the required motion parameters.

KnowRob ontology includes a spectrum of concepts related to robots, including information about their body parts, connections, sensing and action capabilities, tasks, actions, and behavior. Objects are represented with their parts, functionalities, and configuration, while context and environment are also taken into account. Additionally, the ontology incorporates temporal predicates based on event logic and time-dependent relations.

KnowRob's initial ontology was originally rooted in Cyc, an initiative aimed at understanding how the world works by representing implicit knowledge and performing human-like reasoning [122]. However, with the closure of the public initiative OpenCyc, KnowRob underwent a transition to adopt the DUL ontology. DUL was chosen for its compatibility with concepts for autonomous robots. KnowRob uses Prolog as a query and assert interface, but all perception, navigation, and manipulation actions are encoded in plans rather than Prolog queries or rules.

One of the main expansions of KnowRob is RoboEarth, a worldwide open-source platform that allows any robot with a network connection to generate, share, and reuse data [141]. It uses the principle of linked data connections through web and cloud services to speed up robot learning and adaptation in complex tasks.

RoboEarth uses an ontology to encode concepts and relations in maps and objects, and an SLAM map that provides the geometry of the scene and the locations of objects with respect to the robot [142]. Each robot has a self-model that describes its kinematic structure and a semantic model to provide meaning to robot parts, such as a set of certain joints that form a gripper. This model also includes actions, their parameters, and software components, such as object recognition systems [143].

Another example of a KnowRob-based application is [144]. In this case, the focus is on using semantic concepts to annotate an SLAM map with additional conceptualizations. This application diverges somewhat from KnowRob's initial emphasis on robot manipulators. The work models the environment by utilizing semantic concepts but specifically captures the relations between rooms, objects, object interactions, and the utility of objects.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization:* KnowRob and RoboEarth incorporate inference mechanisms to abstract sensing information, particularly in the context of object recognition [140], [144]. As highlighted in [139], inference processes the information perceived from the external environment, abstracting it to the most appropriate level while retaining the connection to the original percepts. These ontologies serve as shared conceptualizations that accommodate various data types and support various forms of reasoning; effectively handling uncertainties arising from sensor noise, limited observability, hallucinated object detection, incomplete knowledge, and unreliable or inaccurate actions [138].
- *Decision-making and planning:* KnowRob places a strong emphasis on determining

action parameterizations for successful manipulation, using hybrid reasoning with the goal of *reasoning with eyes and hands* [140].

This approach equips KnowRob with the ability to reason about specific physical effects that can be achieved or avoided through its motion capabilities. Although the KnowRob *KB* might exhibit redundancy or inconsistency, the reasoning engine computes multiple hypotheses, subjecting them to plausibility and consistency checks, ultimately selecting the most promising parameterization. In the case of KnowRob2, its planning component is tailored for motion planning and solving inverse kinematics problems. Tasks are dynamically assembled based on the robot's situation.

- *Prediction and monitoring*: KnowRob leverages its ontology to represent the evolution of the state, facilitating the retrieval of semantic information and reasoning [140]. Through these heterogeneous processes, the framework can predict the most appropriate parameters for a given situation. However, the monitoring capabilities within this framework are limited to objects in the environment. Unsuccessful experiences are labeled and stored in the robot's memory, contributing to the selection of action parameters in subsequent scenarios. The framework introduces narrative-enabled episodic memories (*NEEMs*), allowing queries about the robot's actions, their timing, execution details, success outcomes, the robot's observations and beliefs during each action. This knowledge is used primarily during the learning process.
- *Reasoning*: KnowRob2 incorporates a hybrid reasoning kernel comprising four *KBs* with their corresponding reasoning engines [140]:
 - Inner World *KB*: Contains *CAD* and mesh models of objects positioned with accurate 6D poses, enhanced with a physics simulation.
 - Virtual *KB*: Computed on demand from the data structures of the control system.
 - Logic *KB*: Comprises abstracted symbolic sensor and action data, enriched with logical axioms and inference mechanisms. This type of reasoning is the focus of our discussion in this article.
 - Episodic Memories *KB*: Stores experiences of the robotic agent.
- *Execution*: KnowRob execution is driven by answering queries about the competence of the robot for a particular task to bridge the gap between undetermined instructions and action. The framework incorporates the cognitive robot abstract machine (*CRAM*), where one of its key functionalities is the execution of the plan. They provide a plan language for articulating flexible, reliable, and sensor-guided robot behavior. The executor then updates the *KB* with information about perception and action results, facilitating the inference of new data to make real-time control decisions [145].

RoboEarth and earlier versions of KnowRob rely on action recipes for execution. Before executing an action recipe, the system verifies the availability of the skills necessary for the task; and orders each action to satisfy the constraints. In cases where the robot encounters difficulties in executing a recipe, it downloads additional information to enhance its capabilities [143]. Once the plan is established, the system links robot perceptions to the abstract task description given by the action recipe. RoboEarth ensures the execution of reliable actions by actively monitoring the link between robot perceptions and actions [141].

- *Communication and coordination*: RoboEarth [143] uses a communication module to facilitate the exchange of information with the Web. This involves making web requests to upload and download data, allowing the construction and updating of the KB.
- *Learning*: KnowRob includes learning as part of its framework. It focuses on acquiring generalized models that capture how the physical effects of actions vary depending on motion parameterization [140]. The learning process involves abstracting action models from data, either by identifying a class structure among a set of entities or by grouping the observed manipulation instances based on a specific property [138]. KnowRob2 extends its capabilities with the Open-EASE knowledge service [146], offering a platform to upload, access, and analyze NEEMs of robots involved in manipulation tasks. NEEMs use descriptions of events, objects, time, and low-level control data to establish correlations between various executions, facilitating the learning-from-experience process.

► PERCEPTION AND MANIPULATION KNOWLEDGE

The Perception and Manipulation Knowledge (*PMK*)⁷ framework is designed for autonomous robots, with a specific focus on complex manipulation tasks that provide semantic information about objects, types, geometrical constraints, and functionalities. In concrete, this framework uses knowledge to support Task and Motion Planning (*TAMP*) capabilities in the manipulation domain [147].

PMK ontology is grounded in the IEEE standard 1872.2 [129] for knowledge representation in the robotic domain, extending it to the manipulation domain by incorporating sensor-related knowledge. This extension facilitates the link between low-level perception data and high-level knowledge for comprehensive situation analysis in planning tasks. The ontological structure of PMK comprises a meta-ontology for representing generic information, an ontology schema for domain-specific knowledge, and an ontology instance to store information about objects. These layers are organized in seven classes: *feature*, *workspace*, *object*, *actor*, *sensor*, *context reasoning*, and *actions*. This structure is inspired by OUR-K [148] further described in Section 5.2.6).

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: The PMK ontology incorporates RFID sensors and 2D cameras to facilitate object localization, explicitly grouping sensors to define equivalent sensing strategies. This design enables the system to dynamically select the most appropriate sensor based on the current situation. PMK establishes relationships between classes such as *feature*, *sensor*, and *action*. For example, it stores poses, colors, and IDs of objects obtained from images.
- *Decision-making and planning*: PMK augments TAMP parameters with data from its KB. The KB contains information on action feasibility considering factors such as

⁷<https://github.com/MohammedDiabl/PMK>

object features, robot capabilities, and object states. The TAMP module utilizes this information, combining a fast-forward task planner with physics-based motion planning to determine a feasible sequence of actions. It helps in deciding where the robot should place an object and the associated constraints.

- *Reasoning*: PMK reasoning targets potential manipulation actions employing description logic's inference for real-time information, such as object positions through spatial reasoning and relationships between entities in different classes. Inference mechanisms assess robot capabilities, action constraints, feasibility, and manipulation behaviors, facilitating the integration of TAMP with the perception module. This process yields information about constraints such as, including interaction parameters (e.g., friction, slip, maximum force), spatial relations (e.g., inside, right, left), feasibility of actions (e.g., arm reachability, collisions) and action constraints related to object interactions (e.g., graspable from the handle, pushable from the body).
- *Interaction and design*: PMK represents interaction as manipulation constraints, specifying, for example, which part of an object is interactable, such as a handle. It also considers interaction parameters such as friction coefficient, slip, or maximum force.

► FAILURE INTERPRETATION AND RECOVERY IN PLANNING AND EXECUTION

Failure Interpretation and Recovery in Planning and Execution (*FailRecOnt*)⁸ is an ontology-based framework featuring contingency-based task and motion planning. This system empowers robots to handle uncertainty, recover from failures, and engage in effective human-robot interactions. Grounded in the DUL ontology, it addresses failures and recovery strategies, but it also takes some concepts from *CORA* and *SUMO* for robotics and ontological foundations, respectively.

The framework identifies failures through the *non-realized situation* concept and proposes corresponding *recovery strategies* for actions. To improve the understanding of failure, the ontology models terms such as *causal mechanism*, *location*, *time*, and *functional considerations*, which facilitates reasoning-based repair plan [149], [150].

The failure ontology requires a knowledge model of the system in terms of *tasks*, *roles*, and *object concepts*. Lastly, FailRecOnt hold some similarities with KnowRob, both target manipulation tasks, are at least partially based on DUL and share some of the authors. Moreover, they propose using PMK as a model of the system [149].

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: Perception in FailRecOnt is limited to action detection to detect abnormal events. For geometric information and environment categorization, the framework leverages PMK to abstract perceptual information related to the environment.

⁸<https://github.com/ease-crc/failrecont>

- *Decision-making and planning:* FailRecOnt is structured into three layers: planning and execution, knowledge, and an assistant low-level layer. The planning and execution layer provides task planning and a task manager module. The assistant layer manages perception and action execution for the specific robot, determining how to sense an action, and checking if a configuration is collision-free. The framework has been evaluated in a task that involves storing an object in a given tray according to its color, handling situations such as facing a closed or flipped box to continue the plan [149].
- *Prediction and monitoring:* Monitoring is a crucial aspect of the FailRecOnt framework. It continuously monitors executed actions and signals a failure to the recovery module if an error occurs. The reasoning component interprets potential failures and, if possible, triggers a recovery action to repair the plan.
- *Reasoning:* FailRecOnt ontology describes how the perception of actions should be formalized. Reasoning selects an appropriate recovery strategy depending on the kind of failure, why it happened, if other activities are affected, etc.
- *Execution:* FailRecOnt relies on planning for execution. The task planner generates a sequence of symbolic-level actions without geometric considerations. Geometric reasoning comes into play to establish a feasible path. During action execution, the framework monitors each manipulation action for possible failures by sensing. Reasoning is then applied to interpret potential failures, identify causes, and determine recovery strategies.
- *Communication and coordination:* FailRecOnt incorporates reasoning for communication failure in [150] to address failures in scenarios where multiple agents exchange information.

► PROBABILISTIC LOGIC MODULE

Probabilistic Logic Module (*PLM*) offers a framework that integrates semantic and geometric reasoning for robotic grasping [151]. Specific details about the KB such as the source files are not publicly available, so the information provided here is derived from their articles.

The primary focus of this work is on an ontology that generalizes similar object parts to semantically reason about the most probable part of an object to grasp, considering object properties and task constraints. This information is used to reduce the search space for possible final gripper poses. This acquired knowledge can also be transferred to objects within the same category.

The object ontology comprises specific objects such as *cup* or *hammer*, along with super-categories based on functionality, such as *kitchen container* or *tool*. The task ontology encodes grasping tasks with objects, such as pick and place right or pouring in. Additionally, a third ontology conceptualizes object-task affordances, considering the manipulation capabilities of a two-finger gripper and the associated probability of success.

In the use of the ontology, high-level knowledge is combined with low-level learning based on visual shape features to enhance object categorization. Subsequently, high-level knowledge

utilizes probabilistic logic to translate low-level visual perception into a more promising grasp planning strategy.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: This approach is based on visual perception, employing vision to identify the most suitable part of an object for grasping. The framework utilizes a low-level perception module to label visual data with semantic object parts, such as detecting the top, middle, and bottom areas of a cup and its handle. The probabilistic logic module combines this information with the affordances model.
- *Decision-making and planning*: PLM leverages ontologies to support grasp planning. It integrates ontological knowledge with probabilistic logic to translate low-level visual perception into an effective grasp planning strategy. Once an object is categorized and its affordances are inferred, the task ontology determines the most likely object part to be grasped, thereby reducing the search space for possible final gripper poses. Subsequently, a low-level, shape-based planner generates a trajectory for the end effector.
- *Prediction and monitoring*: While this framework does not specifically predict or monitor robot actions, it uses task prediction to select among alternative tasks based on their probability of success, although it does not actively monitor them.
- *Reasoning*: PLM uses semantic reasoning to grasp. It selects the best grasping task based on object affordances, addressing the uncertainty of visual perception through a probabilistic approach.
- *Learning*: Learning techniques are used to identify the visual characteristics of the shape, which are then categorized. The acquired knowledge is utilized by the robot for grasp planning.

Table 5.1 provides a concise comparison among the four frameworks analyzed. The primary focus of these frameworks is on complex manipulation tasks, such as grasping, picking and placing objects, and executing manipulation actions effectively. However, their specific scope and emphasis differ. For instance, KnowRob focuses on task parameterization and action execution, PMK centers around task and motion planning, while FailRecOnt emphasizes failure interpretation and recovery.

Each framework integrates perception and action components to enable robots to interact with and manipulate their environment effectively, albeit using different approaches to perception and categorization. These range from semantic reasoning to probabilistic logic, incorporating various sensor modalities. Prediction and monitoring are addressed by KnowRob for objects, FailRecOnt for failure detection, and PLM for tasks. Regarding execution and communication, KnowRob enables the web exchange of information, whereas FailRecOnt handles communication failures between multiple agents through ontology.

On the topic of interaction, only PMK addresses interactions between objects and their constraints. Lastly, only KnowRob and PLM utilize learning to store object descriptions.

KnowRob stands out as the most complete framework in terms of ontological representation leveraging information from previous executions such as descriptions of events, objects, time, and low-level control data for subsequent use.

5.2.5 Navigation Domain

The navigation domain describes the challenges and techniques involved in enabling robots to autonomously move around their environment. This involves the processes of Guidance, Navigation, and Control (*GNC*), incorporating elements such as computer vision and sensor fusion for perception and localization; as well as control systems and artificial intelligence for mapping, path planning, or obstacle avoidance.

► TELEOLOGICAL AND ONTOLOGICAL MODEL FOR AUTONOMOUS SYSTEMS

Teleological and Ontological Model for Autonomous Systems (*TOMASys*)⁹ is a metamodel designed to consider the functional knowledge of autonomous systems, incorporating both teleological and ontological dimensions. The teleological aspect involves the inclusion of engineering knowledge, representing the intentions and purposes of the system designers. On the other hand, the ontological dimension categorizes the structure and behavior of the system. This framework constitutes the predecessor to our model as further detailed in Section 7.1.

TOMASys serves as a metamodel to ensure robust operation, by focusing on mission-level resilience [26]. This metamodel relies on a functional ontology derived from the Ontology for Autonomous Systems (*OASys*) [130] establishing connections between the robot's architecture and its mission. In *TOMASys*, the core concepts include functions, objectives, components and configurations. However, it operates as a metamodel and intentionally avoids representing specific features of the operational environment, such as objects, maps, etc.

At the core of the *TOMASys* framework is the metacontroller. While a conventional controller closes a loop to maintain a system component's output close to a set point, the metacontroller closes a control loop on top of a system's functionality. This metacontroller triggers reconfiguration when the system deviates from the functional reference, allowing the robot to adapt and maintain desired behavior in the presence of failures. Explicit knowledge of mission requirements is leveraged for reconfiguration, by using the system's functional specifications captured in the ontology.

In practice, *TOMASys* has been applied to various robots and environments, particularly for navigation tasks. Examples include its application to an underwater mine explorer robot [152] and a mobile robot patrolling a university campus [153].

The detailed analysis of comparison criteria is as follows:

⁹https://github.com/meta-control/mc_mdL_tomasys

	KnowRob [138], [140], [139], [141], [143], [144], [142]	PMK [147]	FailRecOnt [149], [150]	PLM [151]
P/C	Integrate data-types, object recognition.	Dynamically select the most appropriate sensor.	Action sensing to detect abnormal events.	Identify the most suitable part of an object for grasping.
DM/P	Action parameterization for successful manipulation.	Task and motion planning feasibility.	How to sense an action and checking if a configuration is collision-free.	Translate low-level visual perception into an effective grasp planning.
P/M	Predict the most appropriate parameters. Monitoring objects in the environment.	-	Failure detection and identification of failed recovery attempts.	Task prediction to select among alternatives.
R	Hybrid, symbolic reasoning for sensor and action data.	Integration of task and motion planning with the perception module.	How the perception of actions should be formalized.	Select the best grasping task based on object affordances.
E	Match undetermined instructions and action.	-	Symbolic-level actions without geometric considerations.	-
C/C	Web exchange of information.	-	Communication failures between multiple agents.	-
I/D	-	Information about which part of an object is interactable, constraints.	-	-
L	Use previous descriptions of events, objects, time, and low-level control data.	-	-	Categorize visual shape features.

Table 5.1: Use of ontologies in the manipulation domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).

- *Decision-making and planning:* In TOMASys, the metacontrol system manages decision-making by adjusting parameters and configurations to address contingencies and mission deviations. It assumes the presence of a nominal controller responsible for standard decisions. In case of failure detection or unmet mission requirements, the metacontroller selects an appropriate configuration. The planning process is integrated into the metacontrol subsystem, where reconfiguration decisions can impact the overall system plan, potentially altering parameters, components, functionalities, or even relaxing mission objectives to ensure task accomplishment.
- *Prediction and monitoring:* Monitoring is a critical aspect of TOMASys, providing failure models to detect contingencies or faulty components. Reconfiguration is triggered not only in the event of failure but also when mission objectives are not satisfactorily achieved. Observer modules are used to monitor reconfigurable components of the system.
- *Reasoning:* TOMASys uses a DL reasoner for runtime system diagnosis. It propagates component failures to the system level, identifying affected functionalities and available alternatives. This reasoning process helps to select the most promising alternative to fulfill mission objectives.
- *Execution:* The execution in TOMASys follows the *MAPE-K* loop [40]. It evaluates the mission and system state through monitoring observers, uses ontological reasoners for assessing mission objectives and propagating component failures, decides reconfigurations based on engineering and runtime knowledge, and executes the selected adaptations.
- *Communication and coordination:* TOMASys employs its hierarchical structure to coordinate components working towards a common goal. Components utilize roles that specify parameters for specific functions, and bindings facilitate communication by connecting component roles with function specifications during execution. Bindings are crucial for detecting component failures or errors. In cases where the metacontroller cannot handle errors, a function design log informs the user.
- *Interaction and design:* TOMASys provides a metamodel that leverages engineering models from design time to runtime. This approach aims to bridge the gap between design and operation, relying on functional and component modeling to map mission requirements to the engineering structure. The explicit dependencies between components, roles, and functions, along with specifications of required component types based on functionality, support the system's adaptability at runtime.

► ONTOLOGY-BASED MULTI-LAYERED ROBOT KNOWLEDGE FRAMEWORK

Ontology-based Multi-layered Robot Knowledge Framework (*OMRKF*) aims to integrate high-level knowledge with perception data to enhance a robot's intelligence in its environment [154]. Specific details about the KB such as the source files, are not publicly available, so the information provided here is derived from their articles.

The framework is organized into knowledge boards, each representing four knowledge classes: perception, activity, model, and context. These classes are divided into three

knowledge levels (high, middle, and low). Perception knowledge involves visual concepts, visual features, and numerical descriptions. Similarly, activity knowledge is classified into service, task, and behavior, while model knowledge includes space, object, and its features. The context class is organized into high-level context, temporal context, and spatial context.

At each knowledge level, OMRKF employs three ontology layers: (a) a metaontology for generic knowledge, (b) an ontology schema for domain-specific knowledge, and (c) an ontology instance to ground concepts with application-specific data. The framework uses rules to define relationships among ontology layers, knowledge levels, and knowledge classes.

OMRKF facilitates the execution of sequenced behaviors by allowing the specification of high-level services and guiding the robot on recognizing objects even with incomplete knowledge. This capability enables robust object recognition, successful navigation, and inference of localization-related knowledge. Additionally, the framework provides a querying-asking interface through Prolog, enhancing the robot's interaction capabilities.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: OMRKF includes perception as one of its knowledge classes, specifically addressing the numerical descriptor class in the lower-level layer. Examples of these numerical descriptors are generated by robot sensors and image processing algorithms such as Gabor Filter or scale-invariant feature transform (SIFT) [154].
- *Decision-making and planning*: OMRKF uses an event calculus planner to define the sequence for executing a requested service. The framework relies on query-based reasoning to determine how to achieve a goal. In cases where there is insufficient knowledge, the goal is recursively subdivided into sub-goals, breaking down the task into atomic functions such as *go to*, *turn*, and *extract feature*. Once the calculus planner generates an output, the robot follows a sequence to complete a task, such as the steps involved in a delivery mission.
- *Reasoning*: OMRKF employs axioms, such as the inverse relation of *left* and *right* or *on* and *under*, to infer useful facts using the ontology. The framework uses Horn rules to express concepts and relations, enhancing its automated reasoning capabilities.

► SMART AND NETWORKED UNDERWATER ROBOTS IN COOPERATION MESHES ONTOLOGY

The Smart and Networked Underwater Robots in Cooperation Meshes (*SWARMs*) ontology addresses information heterogeneity and facilitates a shared understanding among robots in the context of maritime or underwater missions [155]. Specific details about the KB such as the source files are not publicly available, so the information provided here is derived from their articles.

SWARMs leverages the probabilistic ontology PR-OWL¹⁰ to annotate the uncertainty of the context based on the Multi-Entity Bayesian Network (*MEBN*) theory [156]. This allows

¹⁰<https://www.pr-owl.org>

SWARMS to perform hybrid reasoning on (i) the information exchanged between robots and (ii) environmental uncertainty.

SWARMS establishes a core ontology to interrelate several domain-specific ontologies. The core ontology manages entities, objects, and infrastructures. These ontologies include:

- *Mission and Planning Ontology*: Provides a general representation of the entire mission and the associated planning procedures.
- *Robotic Vehicle Ontology*: Captures information on underwater or surface vehicles and robots.
- *Environment Ontology*: Characterizes the environment through recognition and sensing.
- *Communication and Networking Ontology*: Describes the communication links available in SWARMS to interconnect different agents involved in the mission, enabling communication between the underwater segment and the surface.
- *Application Ontology*: Provides information on scenarios and their requirements. PROWL is included in this layer to handle uncertainty.

Li et al. [155] presents an example using SWARMS for monitoring chemical pollution based on a probability distribution. SWARMS incorporates this model into the ontology and uses MEBN to deduce the emergency level of a polluted sea region.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: SWARMS ontology provides a shared framework to represent the underwater environment. For this reason, it contains classes on sensors and the main concepts of the environment to understand it through its properties, such as water salinity, conductivity, temperature, and currents. Uncertainty reasoning is critical for the categorization of sensor information, especially in the harsh maritime and underwater environment.
- *Decision-making and planning*: SWARMS uses two levels of abstraction. High-level planning allows the user to describe different tasks related to operations without specifying the exact actions that each robotic vehicle must perform. Low-level planning is performed on each robot to generate waypoints, actions, and other similar low-level tasks.
- *Reasoning*: SWARMS uses a hybrid context reasoner, as it combines ontological rule-based reasoning with MEBN for probabilistic annotations.
- *Communication and coordination*: A main concern in SWARMS is cooperation; robots share tasks, operations, and actions. The ontology provides transparent information sharing to support the heterogeneity of the data. It also provides an abstraction for communication and networking, describing the communication links available from command control stations to vehicles and backward.

► ROBOT TASK PLANNING ONTOLOGY

The Robot Task Planning Ontology (*RTPO*) [157] serves as an effective knowledge representation framework for robot task planning. Specific details about the KB such as the source files, are not publicly available, so the information provided here is derived from their articles.

Designed to accommodate temporal, spatial and continuous and discrete information, RTPO prioritizes scalability and responsiveness to ensure practicality in task planning. The ontology comprises three main components: robot, environment, and task.

- *Robot Ontology*: Comprise hardware and software details, location information, dynamic data, and more. Sensors are explicitly modeled as a subclass of hardware, specifying the measurable aspects of the environment. In an experimental context, the robot's perception is limited to obstacles.
- *Environment Ontology*: Focuses on location and recognition of humans and objects, the environment map, and information gathered from other robots.
- *Task Ontology*: Aims to understand how to decompose high-level tasks into atomic actions and adapt plans when the environment undergoes changes.

This multi-ontology approach allows RTPO to capture the details of robot task planning by representing both the robot's internal state and its interactions with the environment.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: The robot ontology in RTPO considers sensors as a subclass of hardware, providing a specification of measurable aspects of the environment. In the experiment discussed, the robot's perception is limited to detecting obstacles.
- *Decision-making and planning*: Task planning is realized using a domain and a problem file, as in PDDL. The algorithm described in [157] generates and utilizes these files. The planner recursively searches sub-tasks that meet the preconditions of the initial state. When encountering an atomic task, it adds it to the execution and employs post-conditions to select the subsequent action.
- *Reasoning*: The reasoning process in RTPO involves the updating and storage of knowledge within the ontology. The scalability of robot knowledge aims to enhance the efficiency of reasoning. Experimental scenarios involving the addition of elements to the indoor environment and corresponding KB instances demonstrate changes in consumed time, particularly affecting knowledge query speed. The authors show real-time performance in an application that involves 52,000 instances, although the impact on the planning process is not explicitly detailed.
- *Communication and coordination*: This process is not explicitly explained in [157]. However, a scenario with three robots and two humans is described. Communication among the three robots is highlighted, emphasizing knowledge sharing. The relationships between these entities can be defined by users or developers based on their

requirements. In situations with various robots mapping different rooms and using various sensors, the ontology facilitates linking and adding knowledge to constraints to maintain coherence.

- *Learning*: RTPO incorporates learning by updating and adding the plans generated by the task planning algorithm back into the ontology. This iterative process aims to enhance the efficiency of future task planning, especially when the same task is encountered again.

► GUIDANCE, NAVIGATION, AND CONTROL FOR PLANETARY ROVERS

Burroughes and Gao [158] present an architectural solution to address limitations in autonomous software and GNC structures designed for extraterrestrial planetary exploration rovers. Specific details about the KB such as the source files are not publicly available, so the information provided here is derived from their articles.

This framework uses an ontology to facilitate autonomous reconfiguration of mission goals, software architecture, software components, and the control of hardware components during runtime. To manage complexity, the self-reconfiguration ontology is organized into modules. The base ontology serves as an upper ontology, including modules that delineate logic, numeric aspects, temporality, fuzziness, confidences, processes, and block diagrams. Additional modules describe the functions of software, hardware, and the environment.

The primary focus of this framework is reconfiguration. Once the ontology manager detects an undesired state change through the monitoring process, it activates a safety mode. In this safety mode, the system can execute either a reactive plan or create a new plan based on inferences drawn from the new situation.

Following the re-planning process, new mission goals are established, allowing the robot to exit safety mode and resume normal operation. The framework specifically aims to minimize odometry error and ensure safety during travels. To achieve this objective, the connective architecture and processes such as navigation, localization, control, mappings, as well as sensors and the locomotion system, can be fine-tuned and optimized according to the environment and faulty hardware conditions. For example, certain methods may prioritize tolerance to sensor noise over absolute accuracy in localization.

The detailed analysis of comparison criteria is as follows:

- *Decision-making and planning*: Burroughes and Gao [158] integrates a reactive approach with a deliberative layer for quick responses. Pre-calculated responses are prepared for likely changes, but in the absence of options, the rational agent resorts to deliberative techniques. Ontologies and PDDL are used for knowledge representation and planning, respectively. Actions correspond to specific configurations of services, with plans defined using the initial state of the world, the goal state, the resources of the system, the safety criteria, and the rules. The approach employs a pruning algorithm to reduce possible actions and planning space.

- *Prediction and monitoring:* The focus is on re-planning, requiring monitoring to trigger the process. Inspectors, such as generic ones for network, resource, and state checks, and specific ones for tasks like checking camera performance, monitor, and update the ontology on the current state of the world. The system can decide the depth of monitoring for each subsystem, balancing computational resources and self-protection.
- *Reasoning:* Reasoning is used to configure the software elements of the rover, updating the state of the world, and deciding whether the system should re-plan to adapt to changes or use pre-established reactive responses. The ontology checks for knowledge incoherence when adding new information.
- *Execution:* The MAPE-K loop is used for self-reconfiguration. When the monitor detects a change, the ontology provides knowledge to select how the system should evolve. The configuration of the components is established through planning or reactivity. Navigation and operation components are organized into modular services with self-contained functionality to allow reconfiguration.
- *Communication and coordination:* Communication requirements, such as publication rate, conditions, and effects, are used to establish appropriate communication links between modules, treating it as a reconfigurable service within the framework.
- *Interaction and design:* While not explicitly detailing the design or requirements, the system reasons in terms of service capacities, considering measures such as accuracy or suitability for sensor noise in sensor processing algorithms. It also incorporates safety criteria to select system changes, providing design and engineering knowledge to select alternative designs based on runtime situations.

► COLLABORATIVE CONTEXT AWARENESS IN SEARCH AND RESCUE MISSIONS

Chandra and Rocha [159] present a framework for collaborative context awareness using an ontology that comprises high-level aspects of Urban Search and Rescue (*USAR*) missions. Specific details about the KB such as the source files, are not publicly available, so the information provided here is derived from their articles.

The framework serves as an efficient knowledge-sharing platform to represent and correlating various mission concepts, including those related to agents and their capabilities, scenarios, and teams. The ontology facilitates the sharing of homogeneous knowledge among all robots, and supports human-robot collaboration by reasoning based on rules provided by humans.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization:* The system employs perceived and pre-processed information gathered from its own sensors, as well as from other agents such as humans and robots, to continually update its KB. Specifically, it focuses on entities within context classes and their relationships, including information about smoke levels, visibility, location, and temperature.

- *Decision-making and planning*: The framework offers an efficient strategy to share knowledge that improves decision-making processes. For example, it facilitates the management of a global map with shared events and the real-time tracking of agent locations.
- *Reasoning*: In this framework, reasoning plays a crucial role in storing and disseminating information between agents. The KB retains essential details such as localization, temperature, visibility, battery status, tasks, and detection of victims or fires. Furthermore, reasoning contributes to resilience against communication failures by storing information for future sharing. This process supports coordination, including the incorporation of new team members or the temporary removal and subsequent reintegration of teammates.
- *Communication and coordination*: Building on a foundation of multi-robot cooperation, [159] structures its framework to represent the team and mission within an ontology. This enables the creation of a global map and the selection of suitable candidates for various tasks. Examples of coordination include telepresence and compensating for a teammate's immobility. In addition, the framework ensures resilience in the face of communication failures. For example, all events detected by a robot are logged into the local KB and shared across the team to synchronize knowledge. In situations of communication loss or isolation from teammates, a list of unattended events is maintained and shared after communication is restored.

► AUTONOMOUS VEHICLE SITUATION ASSESSMENT AND DECISION-MAKING

Huang et al. [160] presents the use of ontologies for situation assessment and decision-making in the context of autonomous vehicles navigating urban environments. Specific details about the KB such as the source files are not publicly available, so the information provided here is derived from their articles.

The KB is used to integrate vehicle information, storing details of the permissible *directions* it can take. Furthermore, it includes representations of both static and dynamic obstacles, as well as relevant information about the characteristics of roads, distinguishing between *highways* and *urban roads*. The ontology incorporates specific scenarios that the autonomous vehicle can encounter on the road, such as proximity to an *intersection*, traversing a *bridge*, or executing a *U-turn*. The autonomous driving system leverages this KB to assess the current driving scenario, facilitating well-informed decisions on whether to maintain its current trajectory or initiate a lane change.

The detailed analysis of comparison criteria is as follows:

- *Decision-making and planning*: The decision-making process is tied to situation assessment, employing a methodology that evaluates the safety of the surroundings of the vehicle. This involves characterizing the regions around the car and assigning a binary safety value based on the presence of obstacles.

If a region is considered safe, the vehicle continues in its current lane; otherwise, a lane change is initiated. This decision-making step incorporates statistical indicators that

account for velocity. Moreover, the model considers legitimacy by adhering to traffic rules during lane changes and reasonableness by querying the global planning path to identify the next road segment or lane, particularly when approaching intersections. This strategic approach prevents the system from optimizing locally at the expense of compromising the overarching global plan.

- *Reasoning*: The reasoner is used to generate behavioral decisions. It employs rules derived from traffic regulations and driving experiences, taking into account various factors influencing the road, such as speed limits, traffic lights, and the presence of surrounding obstacles.
- *Interaction and design*: While the framework does not explicitly include concepts as design decisions, it addresses requirements such as legality by incorporating traffic rules and reasonableness. This ensures that the behavior of the lane change aligns with the main goal of global planning, avoiding changes to local optimization that may compromise the broader strategic plan.

► SEARCH AND RESCUE SCENARIO

Sun et al. [161] introduces an ontology tailored for Search and Rescue (*SAR*), enhancing the decision-making capabilities of robots in complex and unpredictable scenarios. Specific details about the KB such as the source files, are not publicly available, so the information provided here is derived from their articles. The SAR ontology comprises three interlinked sub-ontologies: the *entity* ontology, *environment* ontology, and *task* ontology.

- The *entity* ontology includes various types of robots, including ground, underwater and air robots. It delineates their constituent parts and specifications on hardware and software aspects.
- The *environment* ontology extends the scope to store most of the elements present in SAR scenarios; which includes environment map and objects that shall be recognized. Updated knowledge from the environment can be shared among other robots, although the specific mechanisms are not detailed in the presented use case.
- The *task* ontology encapsulates task-related knowledge vital for informed decision-making. This involves task decomposition and allocation facilitated by a hierarchical structure. The ontology defines four typical tasks: *charge*, *search*, *rescue*, and *recognize*. Additionally, atomic actions are articulated by their effects on the state of the environment. The framework proposes a task planning algorithm that aligns the pre-conditions of execution with the effects on the environment, utilizing the SAR ontology as a foundational framework.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: The SAR framework focuses on *SLAM* and autonomous navigation. They use a semantic map, which facilitates the recognition and localization

of objects. The acquired information dynamically updates the environment ontology, establishing connections between the SLAM map and the semantic details. Bayesian Reasoning enhances the precision of victim positioning, while QR code scanning streamlines the acquisition of semantic information, such as vital signs (e.g., heart rate and blood pressure), in scenarios involving disaster rescue.

- *Decision-making and planning:* Decision-making in this framework is based on structured queries. It initiates by defining tasks and identifying requisite actions, followed by a comprehensive examination of action properties, including time constraints, pre-conditions, and post-conditions, to achieve the desired state. The planner then specifies a sequenced set of atomic actions. The program has the flexibility to adopt specific search algorithms for planning or reuse previously established plans.
- *Reasoning:* Before planning, the robot conducts a preliminary assessment to determine the suitability of the task for the current state. If considered appropriate, the planning phase starts, identifying the tasks the robot should execute under the given circumstances.
- *Execution:* The robot systematically executes the sequence of atomic actions. When faced with a previously planned task, the robot can query the task definition, retrieving the corresponding atomic action sequence.

Tables 5.2 and 5.3 summarize the studies in the navigation domain; which vary in their focus and specific application, such as underwater exploration, search and rescue missions, or urban navigation.

In terms of perception and categorization, frameworks exhibit different degrees of sophistication. OMRKE, SWARMS, RTPO and the SAR Scenario integrate perception knowledge into their ontologies to represent environmental properties and sensor data. However, the USAR Scenario primarily relies on pre-processed sensor information for perception tasks.

Regarding decision-making and planning, TOMASys, the Planetary Rover Scenario, and the SAR Scenario address adaptive decision-making and reconfiguration, crucial for mission resilience in dynamic environments. OMRKE, SWARMS, and RTPO employ planners or algorithms to generate task sequences or plans, while the USAR Scenario and the Autonomous Vehicle Scenario prioritize safe and efficient route planning in urban and search and rescue contexts.

Prediction and monitoring are only addressed by two of them: TOMASys, which employs failure models to detect contingencies and monitor mission objectives, and the Planetary Rover Scenario utilizes monitoring to trigger re-planning in response to changes in system state or environment.

Reasoning capabilities are present across all frameworks, TOMASys, OMRKE, and SWARMS use advanced reasoning techniques such as rule-based reasoning, probabilistic annotations, and ontology-based reasoning. The USAR Scenario and the Autonomous Vehicle Scenario employ reasoning to evaluate the safety of surroundings and apply traffic regulations effectively, while the Planetary Rover Scenario and the SAR Scenario leverage reasoning to ensure the suitability of tasks in their respective domains.

Execution strategies vary among the frameworks, with TOMASys employing the MAPE-K loop for reconfiguration, while others like the Planetary Rover Scenario focus on configuring software elements and updating world states. RTPO emphasizes real-time performance in reasoning, which is crucial for efficient task execution.

Communication and coordination are critical aspects addressed by TOMASys, SWARMS, and RTPO, facilitating transparent information sharing and multi-robot cooperation. The USAR Scenario and the Autonomous Vehicle Scenario emphasize communication requirements and coordination strategies to ensure effective collaboration among agents.

Interaction and design considerations are explicit in TOMASys, the Planetary Rover Scenario, and the Autonomous Vehicle Scenario, where frameworks bridge the gap between design and operation through meta-models or by incorporating engineering knowledge. RTPO also focuses on design elements, ensuring scalability and responsiveness in task planning.

Learning aspects are only addressed by RTPO, which incorporates learning by updating the plans generated during task planning back into the ontology for future use.

5.2.6 Social Domain

Social robots usually refer to the interaction between robots and humans in a variety of contexts, such as homes, healthcare, education, entertainment, and public spaces. The goal is to create robots that are not only technically capable but also socially aware and able to interact with humans in a manner that is natural, intuitive, and socially acceptable.

► ONTOLOGY-BASED UNIFIED ROBOT KNOWLEDGE

The Ontology-based Unified Robot Knowledge (*OUR-K*) framework for service robots [148] consists of five knowledge classes: features, objects, spaces, contexts, and actions. It takes from its predecessor, *OMRKF*, the concept of layer division. Although *OMRKF* was originally evaluated in navigation domains, *OUR-K* extends its application to social robots operating within domestic environments. Specific details about the KB such as the source files are not publicly available, so the information provided here is derived from their articles.

A notable feature of *OUR-K* is its ability to perform tasks even when provided with incomplete information. The framework incorporates a knowledge description of both the robot and its environment, employing algorithms for knowledge association. This involves logic, Bayesian inference, and heuristics; however, logical inference in *OUR-K* is specifically limited to performing associations between classes and ontological levels, with no indication of alternative inference mechanisms in the literature.

OUR-K includes mechanisms for object recognition, context modeling, task planning, space representation, and navigation. Regarding action representation, its descriptions are simpler and do not contemplate processes as its predecessor, *OMRKF*.

	TOMASys [26], [153], [152]	OMRKF [154]	SWARMS [155]	RTPO [157]
P/C	-	Lower-level perception features such as numerical descriptors.	Represent underwater environment properties and its sensors.	Specification of measurable aspects of the environment.
DM/P	Adjust parameters and configurations to address mission contingencies.	Calculus planner to accomplish goals and sub-goals.	Task-level planning and low-level planning to generate waypoints, actions, etc.	Generate and use domain and problem file to generate plans.
P/M	Failure models to detect faulty components and goals not achieved.	-	-	-
R	Propagate failures to system level, identify affected functionalities and available alternatives.	Geometrical relationships between objects.	Hybrid context reasoner: rule-based reasoning and probabilistic annotations.	Evaluation of real-time performance in reasoning with 52,000 elements of indoor environments.
E	MAPE-K loop to evaluate mission status.	-	-	-
C/C	Component coordination.	-	Robots share tasks, operations, and actions.	Knowledge sharing between three robots and two humans.
I/D	Use of metamodels to bridge the gap between design and operation.	-	-	-
L	-	-	-	Update and add plans generated by the task planning algorithm back into the ontology.

Table 5.2: Part 1: Use of ontologies in the navigation domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).

	Planetary Rover Scenario [158]	USAR Scenario [159]	Autonomous Vehicle Scenario [160]	SAR Scenario [161]
P/C	-	Pre-processed information from sensors, and other intelligent agents.	-	Semantic map for recognize and localize objects.
DM/P	Pre-calculated solutions for likely changes, combined with deliberative techniques.	Management of a global map with shared events and the real-time tracking of agent locations.	Evaluates the safety of the vehicle's surroundings with statistical indicators.	Definition of tasks and identification of requisite actions and its properties.
P/M	Monitor resources and state checks to trigger re-planning.	-	-	-
R	Configure SW elements, update world state, and decide re-planning.	Store and distribute information among agents.	Apply rules derived from traffic regulations and driving experiences.	Evaluate suitability of task for the current state.
E	MAPE-K loop for reconfiguration.	-	-	Query task definition to retrieve the corresponding atomic action sequence.
C/C	Communication requirements between modules.	Multi-robot cooperation, global map, and selection of suitable candidates for each task.	-	-
I/D	Explicit service capacities.	-	Addresses requirements such as legality by incorporating traffic rules and reasonableness.	-
L	-	-	-	-

Table 5.3: Part 2: Use of ontologies in the navigation domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).

Diab et al. [162] extend OUR-K with a physics-based manipulation ontology to address the challenges that a motion planner might encounter. An actor class is introduced within the knowledge classes to describe the working constraints of the robot, enhancing the planner's capability to handle interaction dynamics. In addition, the authors propose a prediction mechanism for the entire OUR-K framework. Instead of relying on inferences, a semantic map is generated for categorizing and assigning manipulation constraints, using reasoning based on logical axioms. This approach is evaluated in the context of a specific manipulation task, where a robot serves a cup of liquid contained in a can.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: OUR-K follows the same approach as OMRKF. Perception is abstracted and stored in the feature, space, and object classes. The feature class defines the same knowledge level as the OMRKF. The space class defines a topological map in the middle level and a semantic map in the higher one. The object class middle level includes the object name and function, whereas the top layer defines generic information and relationships among objects; for example, a cup is a type of container.
- *Decision-making and planning*: OMRKF successor, OUR-K [148] uses the same structure based on the abductive event calculus planner to reach a hierarchical abstraction of space elements and behaviors. High-level tasks, such as *delivery*, are decomposed into mid-level sub-tasks such as *go to goal space*, *find object*, or *generate context*. These sub-tasks are further planned as sequences of primitive behaviors such as *go to* or *recognize object*. The approach in [162] based on OUR-K also integrates the three levels of the action class for planning. The planner consults the topological map to determine which object (or robot) occupies which space, and then the semantic map is used to extract object and robot constraints concerning the action. This results in a sequence of actions that may consider contextual information, particularly at the temporal level.
- *Prediction and monitoring*: OUR-K includes rules for navigation monitoring and missing object recognition tasks. However, monitoring is not treated as a distinct process in this framework; instead, it is represented as rules that use several knowledge classes.
- *Reasoning*: Similar to OMRKF, OUR-K relies on logical inference to associate classes and ontological levels. The bidirectional links between low-level data and high-level knowledge enable both frameworks to fill-in missing information, contributing to mission accomplishment.
- *Execution*: OUR-K execution is plan-based, where the robot follows the plan, executing a sequence of actions. However, action knowledge is coupled with all other concepts to represent world environments using features such as sensory-motor coordination, object action complex, and affordances. This approach allows robots to perform actions without explicit planning, potentially enabling reactive behavior when necessary [148].
- *Interaction and design*: The OUR-K extension presented in [162] introduces dynamic interaction, focusing on how robotic controllers should adapt to runtime situations. This extension provides information on how a motion planner should handle dynamic forces when manipulating objects, enhancing the framework's capability in dealing with real-time interactions.

► OPENROBOTS COMMON SENSE ONTOLOGY

The OpenRobots Common Sense Ontology (ORO) [163], establishes a knowledge processing framework and a common sense ontology tailored for facilitating semantic-rich human-robot interaction environments¹¹. This approach focuses on conceptualization, but offers flexibility for implementing other cognitive functions simultaneously, such as object recognition, task planning, or reasoning. Cooperation is a crucial aspect in ORO, as it targets human-robot interactions.

ORO ontology is based on OpenCyc. The authors specify in the project Wiki¹² that it shares most of its concepts with the first version of the KnowRob ontology. It includes categories such as *spatial thing*, *action*, and more concrete concepts such as *event* or *book*. They evaluated their approach by showing robot objects and asking humans about its properties.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: They use several algorithms, such as common ancestors or the calculation of the best discriminant to categorize perceptions. Discrimination is an important element in human-robot collaboration. For example, if a user asks to bring the bottle and there are two available, the robot can use its differences to select the best option.
- *Reasoning*: ORO provides ontological reasoning, as well as external modules that trigger when an event occurs. External modules are used to provide reactive responses. For example, when a human asks the robot to bring an object, the robot creates an instance of this desire. Statements are stored in different memory profiles, such as long-term and short-term memory. Each profile is characterized by a lifetime that is assigned to the stored facts; when a lifetime ends, the ontology removes the fact.
- *Execution*: The framework integrates CRAM [145] to automatically update the ORO server when an object enters or exits the field of view. Execution is query-based, involving combinations of patterns like *is the bottle on the table?* or filters such as *weight < 150*.
- *Communication and coordination*: ORO is designed as an intelligent blackboard, allowing various modules to push or pull knowledge to and from a central repository. This facilitates knowledge sharing among agents. Examples of heterogeneity with shared information include event registration, categorization capabilities, and the existence of different memory profiles.

Each agent has an alternative cognitive model for other agents with whom it has interacted. When ORO identifies a new agent, it automatically creates a new separate model that can be shared with others. This feature enables the storage and reasoning of different, and potentially globally inconsistent, models of the world.

¹¹<http://kb.openrobots.org/>

¹²<https://www.openrobots.org/wiki/oro-ontology>

► ONTOLOGY FOR COLLABORATIVE ROBOTICS AND ADAPTATION

Ontology for Collaborative Robotics and Adaptation (OCRA)¹³, as described in [164], is a specialized ontology designed to represent relevant knowledge in collaborative scenarios, facilitating plan adaptation. Based on KnowRob [138], [140], and its upper ontology, DUL, OCRA shares the support of its predecessor for the temporal history of the KB through episodic memories.

This work primarily employs FOL definitions to establish a foundation for reliable collaborative robots, with the aim of enhancing interoperability and reusability of terminology within this domain. The ontology serves as a tool for the robot to address questions about the competence of the robot, such as identifying ongoing collaborations, understanding current plans and their goals, determining the agents involved, and assessing plans before and after adaptation.

This ontology has been qualitatively validated for human-robot cooperation, sharing the task of filling the compartments of a tray. The main asset of OCRA, from our perspective, is its explicit representation of collaboration requirements, including safety considerations and a measure of risks, with the objective of effective plan adaptation.

The detailed analysis of comparison criteria is as follows:

- *Decision-making and planning*: OCRA uses ontological knowledge for dynamic plan adaptation during runtime. For example, if the robot had the plan to fill a certain compartment and it is not empty, it adapts to fill another empty compartment. The ontology also stores what triggers the adaptation.
- *Reasoning*: As stated above, this work focuses on answering queries related to agents involved in collaboration, their plans, goals, and the adaptation of plans when required.
- *Communication and coordination*: OCRA focuses on collaboration with humans, providing a mechanism to explain its plan adaptation through the ontology. This collaboration is enabled through coordination in terms of the plan to solve the task. Specifically, the robot changes its plan according to variations in the environment, such as an already filled compartment.
- *Learning*: While not explicitly implemented in the current work, OCRA mentions future plans to incorporate episodic memories for learning tasks. For example, modeling the preferences of different users or learning the structure of tasks to generalize to new ones.

► INTELLIGENT SERVICE ROBOT ONTOLOGY

Intelligent service robot ontology (ISRO) [165] introduces an ontology-based model tailored for human-robot interaction. The practical application of this approach is demonstrated through the implementation of a social robot that functions as a medical receptionist in

¹³https://github.com/alberto0A/know_cra

a hospital setting. Specific details about the KB, such as the source files, are not publicly available, so the information provided here is derived from their articles.

ISRO serves as an abstract knowledge management system designed to comprehend information about agents, encompassing both users and robots, as well as their environment. The ontology establishes connections between user knowledge and the robot's actions and behaviors, incorporating considerations for the spatial and temporal environment. This integration is facilitated through the Artificial Robot Brain Intelligence (ARBI) framework, which includes a task planner, a context reasoner, and a knowledge manager.

ISRO is not limited to a specific domain because it provides a high-level scheme for dynamic generation and management of information. The ontology is divided into four sub-models:

- *User ontology* to store profile information.
- *Robot ontology* defining the robot type, its components and capabilities.
- *Perception and environment* to define objects and its attributes, maps, places, temporal events, and relations such as before, after, etc.
- *Action ontology* to define required actions to perform a task and the expected events in such situations.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: The framework uses sensor information to recognize the user involved, its face, gender, and age; and recognize its state, i.e., the user's expression in its application. It also senses the robot pose to guide the user to the medical department. Additionally, the framework is also prepared to handle information related to spaces and objects, but is not defined in the experiment.
- *Decision-making and planning*: ARBI framework uses path planning to move the robot, specifically, to guide patients to their corresponding medical department. It is based on JAM architecture [166] which defines goals and plans using the belief, desire, and intention (BDI) agent [165]. The path is produced using semantic spatial knowledge, defining the spaces the robot shall traverse and relationships between spaces such as *connected to*. The semantic path is translated into topological waypoints using knowledge about objects and a map. This path is stored in the KB and shared between robots.
- *Reasoning*: The knowledge manager infers spatial and temporal information, such as the relations of currently recognized objects and the time intervals between events. It also characterizes users. It uses Prolog to recognize dynamically generated information. However, only information considered important or critical is stored in the KB as static information.

► SERVICE ROBOT SCENARIO

Ji et al. [167] provide a flexible framework for service robots using the automatic planner Stanford Research Institute Problem Solver (*STRIPS*). Specific details about the KB such as the source files are not publicly available, so the information provided here is derived from their articles.

The ontology represents two main types of information, environmental description, and robot primitive actions to handle spatial uncertainties of particular objects and the primitive actions that the robot can execute. Actions shall include four main attributes to comply with STRIPS: pre-condition, post-condition, input, and result. The planning process is optimized using a recursive back-trace searching method and knowledge information to limit the search space. The framework is applied to the Care-O-Bot agent in a scenario in which it shall get a milk box.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: Symbol grounding is crucial for [167], as it bridges the gap between abstract planning and actual robot sensing and actuation. For example, a task of moving a table involves moving the robot near the table. The symbol grounding specifies exactly where *is near the table*.
- *Decision-making and planning*: This framework uses recursive backtracing search for action planning in dynamic environments. The use of semantic maps can improve the search for an object by limiting the search space using semantic inference.
- *Reasoning*: Reasoning in [167] is used to retrieve information about spatial objects, such as the location of the table where milk is stored. Specific actions, such as *workspace of*, are defined to support reasoning, which enables the retrieval of spatial information about an object. For example, a milk box could provide a result of *above the table* or *in the refrigerator*, as it is a perishable product. This framework also provides a likelihood estimation of possible locations of objects.
- *Execution*: Execution uses a central controller to propagate the action to the task planner, the user interface, and the low-level robot modules. Each action is represented as a state machine that is executed in real time. After the action is finished, it updates the models based on the result and the action post-condition.

Table 5.4 shows the main aspects for each capability in the social applications reviewed. In terms of perception and categorization, OUR-K and ORO differ in their methodologies. OUR-K follows a similar approach to OMRKF, abstracting perception into features, objects, and spaces. On the other hand, ORO employs algorithms like common ancestors or discriminants for categorization, enhancing discrimination in human-robot collaboration scenarios.

Decision-making and planning strategies vary across frameworks. OUR-K utilizes an abductive event calculus planner, extending the hierarchical abstraction approach from OMRKF. ORO emphasizes dynamic plan adaptation during runtime, adapting to changing circumstances. Similarly, OCRA focuses on plan adaptation and collaboration queries, while the Service Robot Scenario employs recursive back-trace searching for abstract planning.

Prediction and monitoring capabilities are evident in OUR-K through navigation monitoring and missing object recognition tasks. In terms of reasoning, OUR-K and ORO leverage ontological reasoning combined with external modules triggered by events. OCRA focuses on answering queries related to collaboration and plan adaptation, while ISRO emphasizes inferential reasoning for user characterization and spatial understanding.

Execution strategies differ, with each framework employing unique approaches. OUR-K utilizes plan-based execution, ORO relies on query-based execution, and the Service Robot Scenario employs a central controller for action propagation and execution.

For communication and coordination, ORO stands out with its intelligent blackboard architecture, facilitating knowledge sharing among agents. OCRA emphasizes collaboration with humans and explainability.

In terms of interaction and design just ORO explicitly uses the ontology. It focuses on dynamic interaction through its blackboard architecture. Learning is not addressed by none of the frameworks even though it envisioned as future work for the Service Robot Scenario, suggesting the incorporation of learning mechanisms for task adaptation and generalization.

5.2.7 Industrial Domain

Robots in industrial domains aim to increase efficiency and precision in a variety of tasks. These are often related to manufacturing, production, and other industrial processes. Ontologies in this domain aims to increase flexibility, reduce maintenance, or enhance control and inspection processes.

► ROBOT CONTROL FOR SKILLED EXECUTION OF TASKS

Robot control for Skilled Execution of Tasks (*ROSETTA*) provides an ontology for robots performing manufacturing tasks¹⁴. Its origins can be traced back to the European projects SIARAS, RoSta, and ROSETTA [168]. The core of the ontology is mostly focused on robot devices and its skills. It relies on a component called the Knowledge Integration Framework (KIF) to provide services for robotic ontologies and data repositories [168]. Note that this should not be confused with the Knowledge Interchange Format, which is a syntax for FOL. KIF acts as an interface to users. They can specify the task partially ordering the sub-goals in an assembly tree; the framework then establishes the action planning and its schedule when limited resources. KIF also provides an execution structure that generates state machines from skill descriptions and its constraints.

Hoebert et al. [169] use ROSETTA ontology to control the assembly process of a variety of electronic devices. This work focuses on a world model that represents robotic devices and skills of ROSETTA. It also relies on the boundary representation (*BREP*) ontology [170] to semantically encode geometric entities. This work uses the ontology to conceptualize

¹⁴https://github.com/jacekmaliec/Rosetta_ontology

	OUR-K [148], [162]	ORO [163]	OCRA [164]	ISRO [165]	Service Robot Scenario [167]
P/C	Same approach as OMRKF.	Use common ancestors or the best discriminant to categorize.	-	Sensor information to recognize the user involved.	Abstract planning information.
DM/P	Abductive event calculus planner for hierarchical abstraction.	-	Dynamic plan adaptation during runtime.	Path planning, semantic maps.	Recursive back-trace searching.
P/M	Navigation monitoring and missing object recognition tasks.	-	-	-	-
R	Bidirectional links between low- and high-level to fill in missing information.	Ontological reasoning combined with external modules triggered when an event occurs.	Answering queries for collaboration, plans, goals, and its adaptation.	Infer spatial and temporal information, user characterization.	Information about spatial objects.
E	Action knowledge is coupled with all other concepts of the knowledge of models and features.	Integrated CRAM to automatically update ontology, query based.	-	-	Central controller to propagate the action to planner, user interface, and lower-levels.
C/C	-	Intelligent blackboard for knowledge sharing among agents.	Collaboration with humans, explainability.	-	-
I/D	Dynamic interaction, runtime controllers adaptation.	-	-	-	-
L	-	-	Envisioned as future work.	-	-

Table 5.4: Use of ontologies in the social domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).

objects in the environment and its properties. It defines a product as a hierarchy of sub-assemblies and parts. Requirements are also included to determine the correct automated manufacturing.

Merdan et al. [171] also employ the ROSETTA ontology to control an industrial assembly process, in particular a pallet transport system and the control of an industrial robot. In this case, it uses ROSETTA and BREP ontologies to conceptualize the robotic system (e.g., skills, properties, constraints, etc.), the product model (e.g. parts, geometries, assembly orientation, etc.), and the manufacturing infrastructure (e.g., product, storage, sensors, etc.). Rosetta has been defined in OWL.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization:* ROSETTA application [169] provides an object recognition module which links perception data with geometric features represented in the KB.
- *Decision-making and planning:* The KIF executor of the ROSETTA ontology serves as the planning mechanism. It transforms an assembly graph into a sequence of operations with preconditions and postconditions, subsequently translated into a task state machine. Both [171] and [169] approach decision-making as a plan generator, utilizing PDDL. Hoebert [169] provides a generator that extracts information from the ontology to produce the required PDDL files for planning. Merdan et al. [171] also translate the semantic model, i.e., states and actions, into domain and problem files through templates.
- *Reasoning:* ROSETTA reasoning is enabled by the KIF server [168]. It allows the user to download and upload libraries with object descriptions, task specifications, and skills. The KIF reasoner assists robot programming by retrieving information about tools, sensors, objects, object properties, etc. Hoebert et al. [169] takes advantage of the ROSETTA ontology to select the individual actions and the equipment required to manufacture a product part.
- *Execution:* KIF uses state machines to execute the skills defined in ROSETTA ontology [168]. However, [169] and [171] use the execution of atomic actions through PDDL commands.
- *Communication and coordination:* ROSETTA-based framework [171] aims to communicate between robots and external production environment entities. Additionally, being a component-based approach, it provides infrastructure for intercomponent communication.
- *Interaction and design:* ROSETTA provides an engineering specification of workspace objects, skills, and tasks from libraries [168]. It acts as a data base for all the information present in the object and in the environment. Hoebert et al. [169] use these design models to establish and verify the manufacturing process, tailoring requirements to parameters such as assembly operation type, manufacturing constraints, material used, and piece dimensions.

► INDUSTRIAL ROBOTIC SCENARIO

Bernardo et al. [172] define an ontology-based approach is introduced for an industrial robotic application focused on inserting up to 56 small pins (sealants) into a harness box terminal, specifically tailored for the automotive industry. Specific details about the KB, such as the source files, are not publicly available, so the information provided here is derived from their articles.

The KB is built upon CORA [125], providing specifications for the robot, machine vision system, and the required tasks for seamless operation. Task sequences are used to execute the plan, but the machine vision system also uses them to inspect if the sealants were correctly inserted into the connecting boxes. If not, the system places a priority on applying the sealant in the faulty position(s) in the connecting box.

The detailed analysis of comparison criteria is as follows:

- *Perception and categorization*: It uses vision to inspect robot operations, specifically to verify the correct insertion of sealants in connecting boxes. Image processing is applied for this inspection, although the information is not categorized in the ontology. The framework also utilizes proximity, position, and fiber optic sensors to facilitate robot operation.
- *Decision-making and planning*: The framework incorporates visual inspection to validate the precision of the insertion of sealants into the connecting boxes. Additionally, it features a re-planning process that modifies production orders when an error calculus task detects errors. Ontological knowledge is used to establish a sequence of tasks, and in the event of an error, a priority task is immediately added to determine the position of the sealant that requires attention. Once this high-priority task is completed successfully, the robotic system resumes the original production order. The authors define adding this priority task as a re-plan, but we think it corresponds to a plan repair as it returns to the original plan after.
- *Reasoning*: This framework uses DL reasoning to acquire valuable data for production and maintenance at the factory level. This reasoning includes information about sensors, actuators, and their purpose in various tasks. The knowledge obtained is then utilized to plan and respond to queries, improving the overall decision-making process.

► ADAPTIVE AGENTS FOR MANUFACTURING DOMAINS

Borgo et al. [173] extend the *DOLCE* ontology from a broad perspective. They validate their approach using a real pilot plant, a reconfigurable manufacturing system designed for recycling printed circuit boards (PCBs). Specific details about the KB, such as the source files, are not publicly available, so the information provided here is derived from their articles. The ontological framework aims to store knowledge about fundamental assumptions and identify the current scenario, including details such as the presence and location of objects, executed actions, responsible agents, changes occurring, etc.

The KB created by this approach seeks to integrate various perspectives within the enterprise, covering intelligent agents, engineering activities, and management activities. The framework is structured in a deliberative layer to synthesize the actions needed to achieve a goal, an executive and monitoring layer to verify the actions, and the mechatronic system that determines the capabilities of the agent.

- *Decision-making and planning:* The updated KB uses an abstraction of the device to be controlled, the environment parameters, and the execution constraints to generate a timeline-based planning model. This model provides the atomic operations that a transportation module can perform, depending on its components and the available collaborators. The timeline incorporates temporal flexibility, allowing for relaxed start and end times. The resulting plan represents the potential evolution of the relevant feature. The framework also supports re-planning in case of plan execution failure, generating a new plan based on the current state of the mechatronic system.
- *Prediction and monitoring:* Execution is monitored by comparing action outcomes with respect to the expected state of the system and the environment. The monitor process receives signals about the (either positive or negative) outcome of the execution, for example, from the transportation module and checks whether the actual status of the mechatronic system complies with the plan or not.
- *Reasoning:* Two types of reasoning are used in this approach. Low-level reasoning infers information about the internal and local contexts of transport modules, identifying system components and available collaborating agents. High-level reasoning utilizes low-level inferences to extract knowledge about the transportation machine's functional capabilities, deducing specific internal and local contexts. This knowledge is then used to generate the plan and its control model.
- *Execution:* The execution is based on a Knowledge-Based Control Loop (KBCL) [173]. This loop facilitates monitoring by dynamically representing the robot's capabilities, internal status, and environmental situation to infer the available functionalities. A reconfiguration phase is activated in case of failure or when new capabilities are added, updating the KB, and initiating a new iteration of the overall loop. The executor receives sensor signals and feedback from the transportation module, issuing action commands based on the plan.
- *Communication and coordination:* The framework includes the exchange of information through ontologies, employing commands for sending and receiving interactions with other entities. The concept of *collaborators* represents relationships between the agent (transport module) and any connected entities, such as other transport modules or machines.
- *Interaction and design:* The ontology also models engineering and management activities. They use engineering approaches to identify high-level functions to be executed to reach a given goal. For this, one explores the difference between the actual state and the desired state and isolates the changes to be made. They include information on operand integrity, operand qualities, quality relationships, etc. These concepts make explicit engineering facts that are usually not included in robotic approaches that complement knowledge about robot capacities and its context.

Table 5.5 provides a summary of the three frameworks as applied to industrial scenarios. For perception and categorization, ROSETTA employs object recognition to link perception data with geometric features, facilitating accurate perception. The Industrial Robotic Scenario utilizes vision systems and sensors for object inspection and categorization.

For decision making and planning, ROSETTA transforms assembly graphs into task state machines, enabling efficient task planning and execution. The Industrial Robotic Scenario incorporates a re-planning process based on error detection to modify production orders dynamically; whereas Adaptive Agents for Manufacturing Domains scenario generates timeline-based planning models with atomic operations, facilitating flexible and adaptive planning.

Only the Industrial Robotic Scenario employs prediction and monitoring by comparing action outcomes with respect to the expected status of the system and the environment. However, the three frameworks reason with the ontology. ROSETTA uses reasoning to retrieve information about tools, sensors, objects, and their properties, helping in decision-making processes. The Industrial Robotic Scenario uses it to acquire valuable data for production and maintenance, and the Adaptive Agents for Manufacturing Domains scenario employs low- and high-level reasoning for local context and functional capabilities, improving overall reasoning abilities.

Execution is used through state machines in ROSETTA and just for reconfiguration for dynamic updates and adjustments in the Adaptive Agents for Manufacturing Domains scenario. On communication and coordination, ROSETTA facilitates communication between robots and external entities, enhancing coordination in industrial environments, and the Adaptive Agents for Manufacturing Domains scenario utilizes commands for interaction with other entities, improving communication and coordination capabilities.

Lastly, for interaction and design, ROSETTA provides design models to establish and verify manufacturing processes, ensuring efficient and tailored requirements. Similarly, the Adaptive Agents for Manufacturing Domains scenario models engineering and management activities, enhancing interaction and design capabilities. None of the mentioned frameworks explicitly addresses learning mechanisms in the industrial robotics domain.

5.2.8 Discussion

In this part, we summarize the most relevant results on the reviewed efforts as a whole, independently of the discipline in which they were evaluated, as most of them aim to be generally applicable to any domain. We also discuss other relevant information on the work reviewed, such as the use of temporal information or the encoding language.

As far as *perception and categorization* is concerned, most of the projects focus on situation assessment, providing information about the environment to handle the situation and make decisions accordingly. This part is directly related to reasoning, since frameworks such as KnowRob [138], [140], SWARMs [155], or PLM [151] use probability reasoning to generate knowledge about the environment and its affordances.

	ROSETTA [168], [169], [171]	Industrial Robotic Scenario [172]	Adaptive Agent for Manufacturing Domains [173]
P/C	Object recognition linking perception data with geometric features.	Vision to inspect robot operations and proximity, position, and fiber optic sensors.	-
DM/P	Transform an assembly graph into a sequence of operations with preconditions and postconditions to create a task state machine.	Re-planning process that modifies production orders when an error calculus task detects them.	Generate a timeline-based planning model with atomic operations.
P/M	-	-	Compare action outcomes with respect to the expected status of the system and the environment.
R	Retrieve information about tools, sensors, objects, object properties, etc.	Reasoning to acquire valuable data for production and maintenance at the factory level.	Low-level reasoning for local context and high-level reasoning for the transportation machine's functional capabilities.
E	State machines to execute skills.	-	Reconfiguration phase is activated in case of failure or when new capabilities are added.
C/C	Communication between robot and external production environment entities; and infrastructure for intercomponent communication.	-	Commands for sending and receiving interactions with other entities.
I/D	Design models to establish and verify the manufacturing process, tailoring requirements to parameters.	-	Model engineering and management activities
L	-	-	-

Table 5.5: Use of ontologies in the industrial domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).

All works except ORO [163] and OCRA [164] use explicit knowledge for *decision-making and planning*. These two focus on answering queries related to actions or plan for this in future developments. Geometric and grasp planning are widely used along with task planners to establish a sequence of atomic actions. PDDL is the language most widely used for task planning. Re-planning is also important for some of these works, TOMASys [26], RTPO [157], FailRecOnt [149], industrial application [172], adaptive manufacturing application [173] or the planetary rover case [158]. In these projects, the objective is to maintain the operation in the presence of faults. Lastly, some of them also use ontologies to plan in combination with reactive behaviors, such as OUR-K [148], ORO [163], or the planetary rover application [158].

Prediction and monitoring are less present in the articles reviewed. This activity requires a fully operational KB used during runtime and a deeper understanding of the situation and the autonomous robot. Most of the works monitor only the environment, such as KnowRob [138], [140], OUR-K [148]. Others only assess the robot state, such as FailRecOnt [149], the manipulation application [173] and the planetary rover application [158]. Lastly, TOMASys [26] assesses the robot state and mission status.

Reasoning is addressed in all the works, as all use ontologies to, at least, answer queries and verify ontological consistency. This was, indeed, the inclusion requirement for the review. Ontological information is used at runtime to recover from failure in some frameworks such as FailRecOnt [149], TOMASys [26] or the planetary rover application [158]. Reasoners can download additional required information, such as RoboEarth [143] using Web and cloud services or ROSETTA [168] using the KIF server to download and upload libraries.

SWARMS [155] uses a hybrid context reasoner, combining ontological rule-based reasoning with *MEBN* theory [156] for probabilistic annotations. PLM [151] uses a probabilistic logic module for grasping, which combines semantic reasoning with object affordances. KnowRob2 [140] takes a further step and provides a hybrid reasoning kernel providing physics-based reasoning, flexible data structure reasoning, and a detailed robotic agent experience. However, this use comes with a higher cost, as its computation may yield inconsistencies or inefficient reasoning. For this reason, RTPO [157] targets knowledge scalability and efficient reasoning by conducting a study on the performance of real-time reasoning with 52.000 individuals. However, the authors do not describe how this approach affects the planning process.

Not all frameworks use explicit knowledge during *execution*, some of them only carry out the action sequence defined in the plan. Others drive its execution to answer queries about the competence of the robot for a particular task, such as OCRA [164] or KnowRob [138], [140]. KnowRob additionally uses the *CRAM* [145] executor to update the KB with information about perception and action results, inferring new data to make control decisions at runtime. ORO [163] also uses *CRAM* to update the server when new objects are detected.

Other approaches use state machines, such as ROSETTA [168]; or the MAPE-K loop [40], such as TOMASys [26] or the planetary rover application [158]. Similarly, the manufacturing application from [173] presents a knowledge-based control loop to combine execution with monitoring.

Other important processes for autonomous systems are *communication and coordination*. All ontological systems can be used easily to answer queries from a human operator. However, this is not sufficient for reliable autonomy; the ontology shall provide transparent information for knowledge completion. SWARMS [155] enable an abstraction for communication and networking and information sharing of heterogeneous data. For this, ORO [163] uses an intelligent blackboard that allows other modules to push or pull knowledge to a central repository. RoboEarth [142] and ROSETTA [168] obtain knowledge from other sources. RoboEarth defines a communication module for uploading and downloading information from the Web, and ROSETTA uses the KIF server to download and upload libraries.

Coordination between agents and components of the agent is critical for autonomous operation. ROSETTA provides an infrastructure for inter-component communication. Component coordination is explicitly addressed in TOMASys [26], the planetary rover from [158] and the adaptive manufacturing from [173]. The USAR application from [159] targets multi-robot cooperation not only for information sharing, but also for task coordination with other agents. OCRA [164] also handles task coordination and plan adaptation, but only concerns human-robot collaboration.

Interaction and design are often not explicitly handled during robot operation. However, awareness of interaction allows the robot to step back from action execution and understand the sources of failure. TOMASys [26] provides a metamodel to benefit from engineering models used at design time. ROSETTA also uses the engineering specification of workspace objects, skills, and tasks as part of the KB. The planetary rover [158], does not explicitly include any information on design or requirements; however, it reasons in terms of the capacity of the services onboard, using some sort of operational requirement to select among alternatives. Similarly, the autonomous vehicle framework [160] handles requirements such as legality using traffic rules or reasonableness to support decision-making. Moreover, the adaptive manufacturing application from [173] takes a further step in modeling engineering and management activities. Lastly, with regard to dynamical interactions, the OUR-K extension from [162] includes this knowledge to support the adaptation of robotic controllers. While PMK [147] represents the interaction as manipulation constraints.

Given the availability of general learning methods, all the processes described above can improve their effectiveness through *learning*. However, this process is less present in the works analyzed in the review. Most systems use a form of case-based learning when storing previous successful plans or using episodic memories to recall past situations, such as CORA [164]. Other frameworks such as PMK [147] or PLM [151] use learning as part of situational assessment to categorize perceptual information.

KnowRob provides the most complete learning mechanics. It learns class structures of entities and identifies manipulation places; in addition, it produces generalized models of the physical effects of actions. KnowRob2 [140] makes available the learned information through the Open-EASE knowledge service [146]. This service enables any user to upload, access, and analyze episodic memories of robots performing manipulation tasks.

► OTHER ASPECTS

Table 5.6 depicts other aspects of the frameworks studied, such as languages used, use of temporal conceptualizations and foundational ontologies.

Almost all works use OWL or a combination of OWL and Prolog as the language for the KB. As discussed for each work, most encoding languages were used to facilitate planners. Most of them used geometric planners, such as PLM [151], along with task planners. In some works, task planners are custom made, such as the service robot from [167]. However, in general, task planners are based on ontological queries, such as the SAR scenario [161], the industrial application [172], or the adaptive manufacturing of [173]. The Planetary Rover [158] and ROSETTA [168] uses PDDL directly, whereas RTPO [157] and SWARMS [155] use a custom approach similar to PDDL.

To address planning dynamics, only some work uses temporal conceptualizations. Most of them rely on an ordered sequence of actions. However, KnowRob [138], [140], OCRA [164] and the adaptive manufacturing approach use time intervals to recall a time frame in which the action is executed. OMRKF [154] and OUR-K [148] include time as part of the context ontology, as well as the planetary rover [158] and ISRO [165] that conceptualize time in their KBs.

Framework	Encoding Lang.	Temporal	Upper-level Ont.
KnowRob [138], [140], [139], [141], [143], [144], [142]	OWL, Prolog	Intervals	DUL
PMK [147]	OWL, Prolog	-	IEEE-1872.2 Std.
FailRecOnt [149], [150]	OWL	-	DUL
PLM [151]	OWL	-	-
TOMASys [26], [153], [152]	OWL	-	OASys
OMRKF [154]	Prolog	Context	-
SWARMS [155]	OWL	-	PR-OWL
RTPO [157]	OWL, Prolog	-	-
Planetary Rovers Scenario [158]	PDDL, OWL	Time Concept	-
USAR Scenario [159]	OWL	-	-
Autonomous Vehicles Scenario [160]	Prolog	-	-
SAR Scenario [161]	OWL	-	-
OUR-K [148], [162]	OWL	Context	OMRKF
ORO [163]	OWL	-	-
OCRA [164]	OWL	Intervals	DUL, KnowRob
ISRO [165]	OWL, Prolog	Time Concept	OpenCyc
Service Robot Scenario [167]	OWL, STRIPS	-	-
ROSETTA [168], [169], [171]	PDDL, OWL	-	-
Industrial Robotic Scenario [172]	OWL	-	CORA
Adaptive Agents [173]	OWL	Intervals	DOLCE

Table 5.6: Summary of other aspects of the framework: concrete encoding languages used, incorporation of temporal conceptualizations, and whether the framework is built upon other works or utilizes an upper-level ontology.

5.3 Research Directions and Conclusions

Considering the projects surveyed, we believe that ontologies are a valuable asset to support robot autonomy. The frameworks discussed provide an advance towards mission-level dependability, increasing robot reliability and availability. However, there are both unsolved issues and valuable possibilities for further research in this domain.

An unsolved issue is the limited dissemination and convergence of the different ontological approaches. There is still not enough reuse of existing ontologies, nor interchangeability or interoperability of the different realization frameworks. In this direction, KnowRob [138], [140] is the most documented and impactful project, as it proves its influence on other projects such as ORO [163], FailRecOnt [149], or OCRA [164]. Ontology convergence is always a challenge that is exacerbated by the variety ontologies, both in vertical—the levels of abstraction—and horizontal—the domains of application. However, efforts must be made to ensure this harmonization, given the fact that robots are not isolated entities, but are always part of systems-of-systems that share specific forms of knowledge [174].

We believe that future engineering-grade knowledge-driven autonomous robot software platforms should also be capable of providing three characteristics: explainability, reusability, and scalability. In the articles under review:

- *Explainability* is enabled in most cases—at least in a shallow form—as most frameworks include query/answer interfaces to provide information on robot operation. However, this explainability is tightly bound to the capability of the human user that shall commonly be a robot operator to fully grasp the explanation. More broad-spectrum, human-aligned ontologies are needed to support a wider spectrum of users.
- *Reusability* is intrinsically present, given the explicitness of declarative knowledge and the sharing of common backgrounds, as some of them are built on previous works or upper-layer ontologies. Effort shall be put in the harmonization and integration of conceptualizations to effectively reuse ontologies—esp. in heterogeneous systems.
- *Scalability* and information handling are also partially addressed in some of the realizations, such as reasoning in RTPO [157] and KnowRob [138], [140]. However, the problems of scalability remain. For example, when dealing with complex systems-of-systems, when addressing geographical or temporal extensive missions, or when knowledge-based collaboration is required as is the case of cognitive multirobot systems or human-robot teams.

Therefore, we propose a shared concern in future works in this field to improve the capabilities of robots and contribute to the community in these three aspects. There are already some initiatives such as *CRAM* from KnowRob authors, a software toolbox for the design, implementation and deployment of manipulation activities [145]. This toolbox supports planning, beliefs, and KnowRob reasoners and has been used in other frameworks such as ORO [163]. Furthermore, the same authors provide the package *rosprolog*, a bidirectional interface between SWI-Prolog and ROS to make this logic language accessible to the main

robotics middleware. However, these tools are tailored for KnowRob-based environments and sometimes are not well documented or maintained.

The conclusions we extract from this survey are that most of the work is focused on categorization, decision-making, and planning. However, *monitoring and coordination* are critical processes regarding robot control, especially for robust and reliable operation. These kinds of processes are difficult to assess using reusable implementations because of the enormous variability between applications, context, and intervening agents. We propose the use of *explicit engineering models* to facilitate these processes. They are already partially included in TOMASys [26], ROSETTA [168], the Planetary Rover Scenario [158], and the adaptive manufacturing application from [173]. We believe that the combination of runtime information with design knowledge can be used during decision-making and to drive adaptation to bridge the differences between the expected results of robotic users and the actual robot performance.

This research aims to bridge the gap between engineering and runtime adaptation using *systems engineering*. The adaptation based on predefined and analyzed systems patterns has been widely demonstrated [26] but the need for on-the-fly design processes has yet to be fully addressed. Automating system design has always been a challenge, but the combination of knowledge-intensive approaches based on ontology and exploratory methods may provide new possibilities for reliable robots that fully understand and adapt themselves [175]. Moreover, we explore the use of Category Theory to formalize relationships between terms, offering a mathematical ground to enhance reusability and interoperability.

Chapter 6

Category Theory

Category Theory (CT) is a general theory of mathematical structures. It was developed in the 1940s to unify and synthesize different areas of mathematics [176]. CT serves as a tool for describing structures and maintaining control over which aspects are preserved when performing abstractions. Its mathematical foundation enables powerful communication between apparently unrelated fields.

One of the remarkable features of CT is its application in the reshaping and reformulation of problems within pure mathematics, including topology, homotopy theory, and algebraic geometry [177]. CT offers techniques, tools, and ideas to identify recurring patterns in various disciplines and formalize them. For this reason, we use it as a framework to support system modeling and behavioral analysis for complex *SoS*. The next section details the motivation for using CT in combination with MBSE. The subsequent sections provide an introduction to the discipline and related work in applied CT.

6.1 Category Theory, a Natural Fit for Model-Based Systems Engineering

We propose to use CT tools to represent *MBSE* aspects, including requirements, system behaviors, and system architectures, with formal traceability provided by CT. The application of composition provides precise abstraction and refinement. *SysML* and other languages supporting MBSE require increasing modeling effort to capture different views of the system and maintain them concurrently, as they evolve asynchronously. The tools provided by CT can help to address this challenge.

Unlike domain-specific languages, CT constructs have precise meanings derived from how its concepts are described mathematically [178]. Therefore, it is a natural framework for modeling and analyzing systems regulated by its underlying rigor. *SE* studies how to build systems to fulfill a purpose based on structures and functional behaviors that emerge from the combination elements. Compositionality represents the idea that the meaning of a complex expression is determined by (i) the meanings of its constituent parts and (ii) the rules for

how those parts are combined. This notion was shown in Chapter 3 which describes how *MBSE* can be used to decompose large and complex systems into manageable parts.

CT provides a solid background for abstracting the system model into multiple layers, emphasizing only essential aspects within each layer. This approach enables us to discuss the interrelations between components rather than focusing on their individual attributes. In essence, categorical semantics is focused on perceiving a system through its compositional structure rather than the characteristics of its constituent elements [179].

This discipline is centered around roles and relationships through the combination and transformation of structures. Across multiple levels of abstraction, it allows the analysis of concepts from different perspectives. This entails detailing the connections between components, their coupling, and their structural and behavioral aspects both individually and in conjunction. In addition, it explores the emergence of phenomena within specific contextual combinations. Another advantage of this approach is the consistency between model perspectives, which may pose a challenge when applying classic MBSE techniques [64].

The following sections describe the main aspects of CT for systems engineering and how formal composition can be applied in various applications to provide expressive representation and structural consistency; in other words, how we can reason about situations where we are composing elements together. The ultimate goal of this perspective is to use abstractions to build the best possible system based on the runtime situation.

6.2 Category Theory Concepts for Engineering

This section introduces the main CT elements used to capture engineering views of the system. CT allows us to be completely precise about otherwise informal concepts [180]. We start with some basic elements to represent structures and evolve them into more complex formalisms such as *operads*. Lastly, we introduce the concepts of *pushout* and *pullback*, which provides a method to condense and combine data from pre-existing structures.

Note that this introduction is just a basic description for engineering applications; for further reading, refer to [181], [182], [183] from a mathematical perspective, [184] from a computer science perspective and [176], [185] for a scientific application with a strong mathematical flavor.

6.2.1 Category, Functor, and Natural Transformation

A *category* \mathcal{C} , is a collection of elements with relations between them. It constitutes an aggregation of objects with an imposed structure [178]. To specify a category, we need three constituents:

- A collection of *objects*, $\text{Ob}(\mathcal{C})$.

- A collection of *morphisms* $f : X \rightarrow Y$, which represents arrows between pair of objects $X, Y \in \text{Ob}(\mathcal{C})$. Morphisms are transformations that preserve the relevant properties of objects in a given category.
- A *composition* operation $g \circ f : X \rightarrow Z$ composing morphisms $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ provided that the target of the first matched the source of the second.

Example: Consider the category \mathcal{C} that describes information about passengers and seats in an airline company as shown in Figure 6.1.

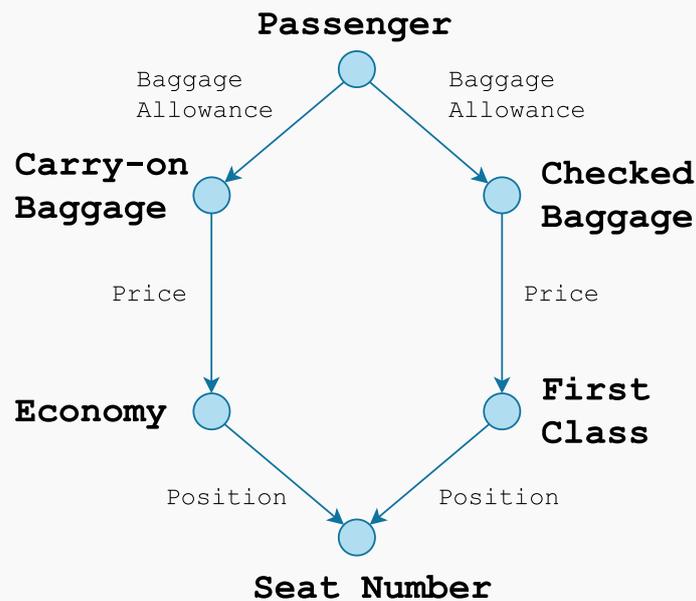


Figure 6.1: Passengers and seats category.

The composing elements of this category are:

- Objects: Specific values representing the passenger, its allowed baggage, class and seat number, i.e., {Passenger Name, Carry-on Baggage, Checked Baggage, Economy, First Class, Seat Number}.
- Morphisms: Arrows connecting the information, i.e., {Baggage Allowance, Price, Position}
- Composition: Paths connecting pieces of information, for example, get passenger class provided passenger and its allowed baggage.

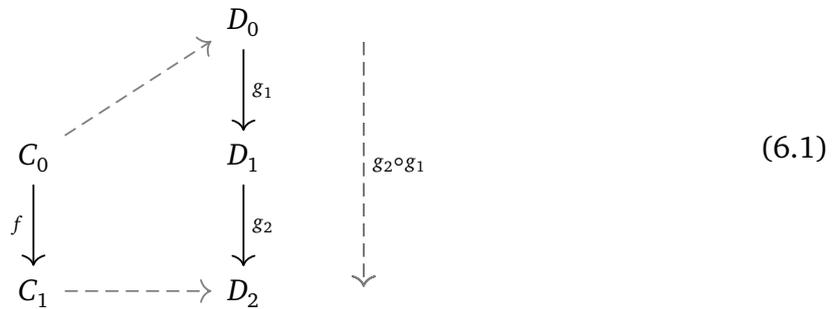
Additionally, categories must satisfy the following conditions:

- *Identity*: for every object $X \in \text{Ob}(\mathcal{C})$, there exists an identity morphism $id_X : X \rightarrow X$ in which the relationship terminates at the source object.
- *Associativity*: for any three morphisms, $f : X \rightarrow Y$, $g : Y \rightarrow Z$, $h : Z \rightarrow W$, the following expressions are equal: $(h \circ g) \circ f = h \circ (g \circ f) = h \circ g \circ f$.

- **Unitality:** for any morphism $f : X \rightarrow Y$, the composition with the identity morphisms at each object does not affect the result, $id_X \circ f = f$ and $f \circ id_Y = f$.

A *functor* F , is a map between two categories \mathcal{C}, \mathcal{D} . It assigns objects to objects and morphisms to morphisms, preserving identities and composition properties. Functors maintain structures when projecting one category inside another.

Diagram 6.1 represents the functor $F : \mathcal{C} \rightarrow \mathcal{D}$, mapping between a category \mathcal{C} composed of $C_0, C_1 \in \text{Ob}(\mathcal{C})$ with morphism $f : C_0 \rightarrow C_1$, and a category \mathcal{D} composed of $D_0, D_1, D_2 \in \text{Ob}(\mathcal{D})$ with morphisms $g_1 : D_0 \rightarrow D_1$ and $g_2 : D_1 \rightarrow D_2$. The functor map corresponds to the dashed gray arrows assigning $F(C_0) = D_0, F(C_1) = D_2$, and $F(f) = g_2 \circ g_1$.



Functors can map between the same types of category, such as **Set** \rightarrow **Set**, or between different categories, such as **Set** \rightarrow **Vect**, between the category of sets and the category of vector spaces.

Example: Following the above example, there is a category \mathcal{D} with roughly the same information as \mathcal{C} . A *Functor* assigns objects from \mathcal{C} to objects in \mathcal{D} , note that the baggage allowance morphism in the source category corresponds to an identity morphism in the destination category as shown in Figure 6.2.

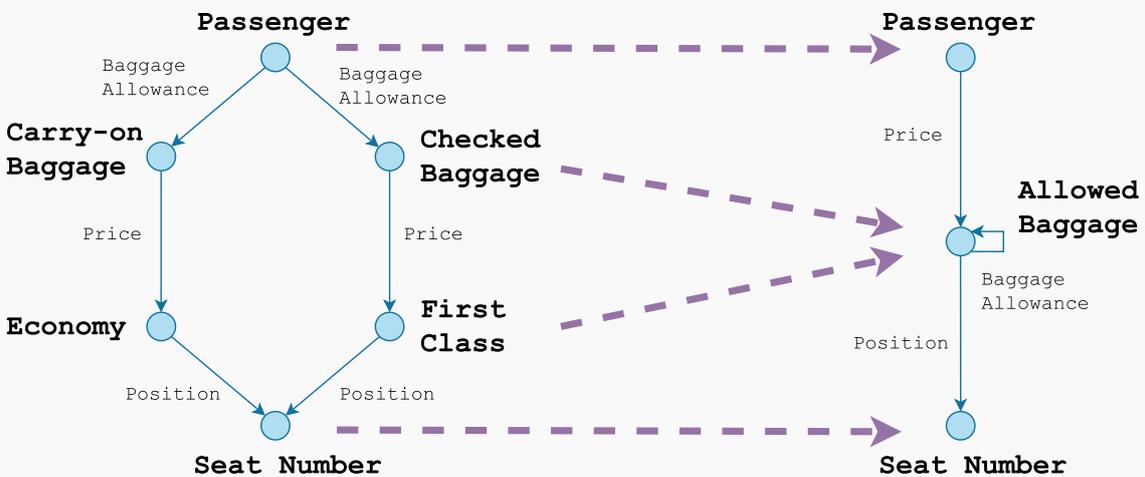


Figure 6.2: Functors for the passengers and seats category.

Example: Now consider a category \mathcal{O} that assigns passengers to seats and the airline category \mathcal{C} discussed before. There is a functor F assigning passengers to the type of baggage that is authorized for that passenger (ie., carry-on or checked-in baggage). This is depicted in Figure 6.3 as a purple dashed arrow. There is also a functor G assigning classes to seat numbers, represented as the green dashed arrow.

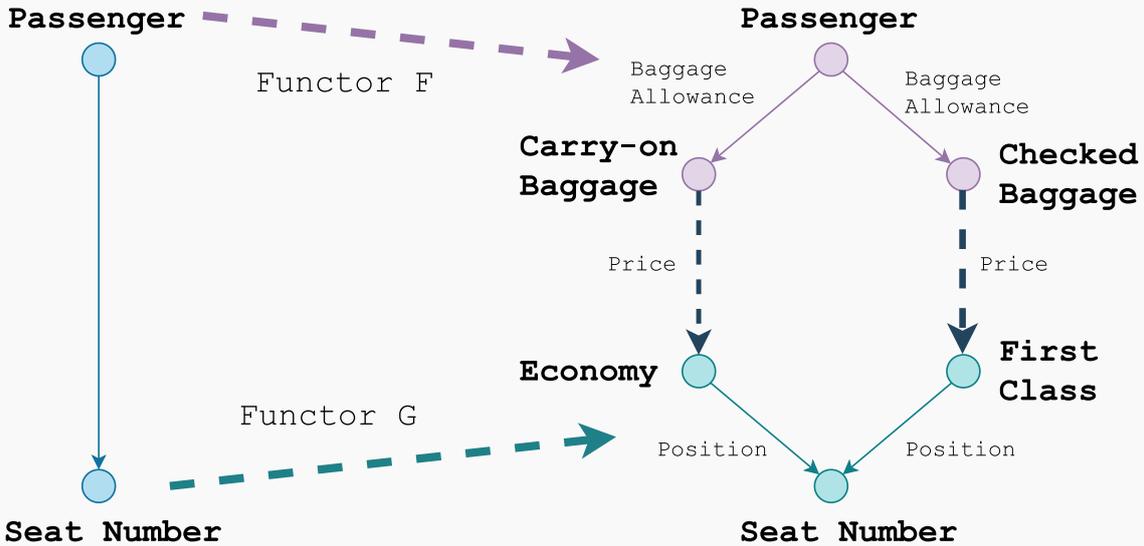


Figure 6.3: Functors and natural transformation for the passengers and seats category.

Both functors depict how the information about a plane ticket is enhanced in the category \mathcal{O} with two perspectives F with assigns the type of baggage permitted and G establishing the seats depending on the class. There is a natural transformation α that combines both perspectives. This structure is depicted as the blue dashed morphism representing the price in category \mathcal{C} that relates baggage and class.

6.2.2 Representing Resources: Wiring Diagrams

Fong and Spivak [185] define *wiring diagrams* as a solution to the question “Can I transform what I have into what I want?” These diagrams offer a categorical formalization of engineering block diagrams. In this framework, boxes symbolize transformations (i.e., morphisms in a category), wires represent resources (i.e., objects in a category), and the composition of morphisms is depicted by placing two boxes in series.

In a broader sense, wiring diagrams represent monoidal categories $(\mathcal{C}, \otimes, \{1\})$, where \mathcal{C} is a category equipped with a tensor product \otimes and a monoidal unit $\{1\}$, allowing the combination of objects in the category. This structure provides a basis for placing boxes in parallel in the diagram. Furthermore, the *symmetric monoidal categories (SMC)* and their corresponding coherence conditions allow the inclusion of feedback wires and swapping in

these diagrams. Consequently, SMC allows the representation of most engineering diagrams.

Figure 6.4 illustrates an example of a wiring diagram that represents a robot in a navigation task. It processes the distance to the objects to determine the velocity to move the robot. The objective is to transform sensor information and available battery into a velocity command to reach a desired point. In particular, this diagram resembles a block definition diagram in *SysML*.

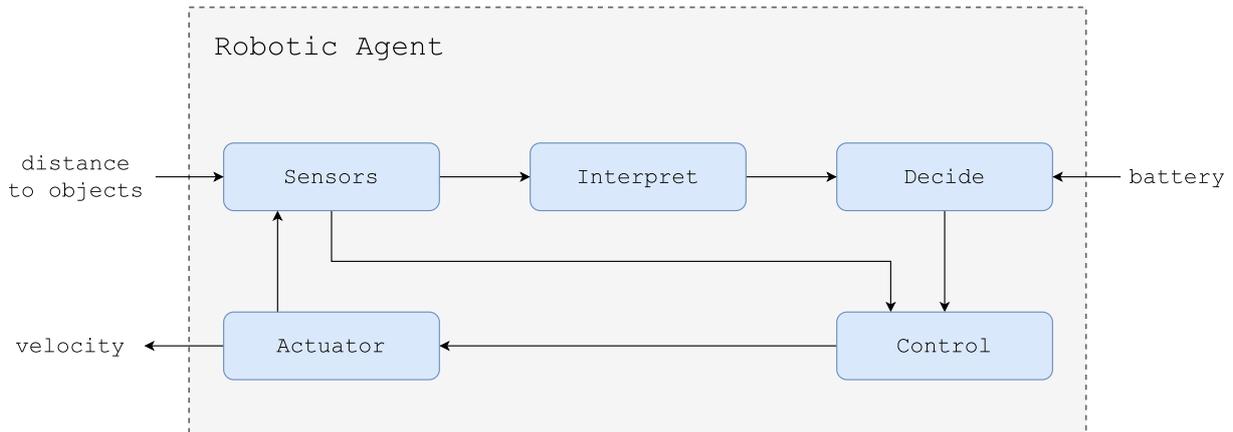


Figure 6.4: Example of a wiring diagram representing a robot in a navigation task. It has distance to objects and available battery as resources and velocity command as product.

In general, wiring diagrams share similarities with input-output models. However, their syntax is more versatile, as the content within the boxes need not be strictly a mathematical function. It can encompass various processes, ranging from concrete descriptions such as automata to more abstract processes or even requirements of a mathematical unknown formula. Similarly, the arrows do not necessarily contain information; they serve to establish relationships between objects [186]. For a formal definition of wiring diagrams, refer to [182] and [187]. This structure is an essential resource in CT, and some of its applications are discussed in more detail in Section 6.3. However, for this research, we prefer alternative representations.

6.2.3 Representing Networks: Operads and Algebras

Categories, functors, and natural transformations are the building blocks of CT. They allow us to express objects, relationships among them, and abstractions while preserving their structure. This approach imposes some directionality since morphisms, functors, and natural transformations are maps with a domain and a co-domain. From an engineering perspective, we could see domains and co-domains as input and output ports, respectively. However, we sometimes prefer to express systems in terms of networks in which ports exit without a direction. CT provides a generalization over categories; operads which models assemblies of structures.

An *operad* is a mathematical object that describes operations with multiple inputs and one output [188]; classically, it only describes one object. We use here what are commonly known

as colored operads, which can hold many objects (i.e., colors). Leinster [189] provides a precise definition of operads; a rough conceptualization is as follows. An operad consists of:

- A set of objects $\text{Ob}(\mathcal{O}) = X_1, \dots, X_n$ each of which can be seen as cells.
- A set of morphisms φ, ψ, \dots with domain and co-domain $\text{Ob}(\mathcal{O})$ that constitute the operations. These morphisms act as an assembly of cell interfaces creating an external interface.
- A composition formula, \circ , or substitution that enables the assembly of morphisms, i.e., the composition of interfaces.

Figure 6.5 depicts an operad with four cells f, g, h, k with, respectively, 5, 4, 4, 1 ports. They form an external cell with 4 ports. The objects are circles with a finite set of ports, the morphisms are the wires that connect the interior cells to the exterior ones, and this external cell can be composed into others [54].

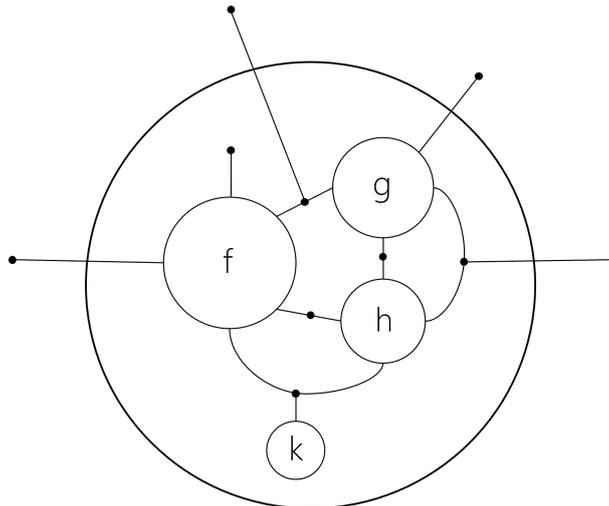


Figure 6.5: Example of an operad composed of a f cell with 5 ports, a g cell with 4 ports, a h cell with 4 ports, a k cell with 1 ports. They form an external cell with 4 ports.

Operads allow us to define abstract operations, but we need to “fill” those cells to ground them. An *algebra* is an operad functor from the operad to the **Set** category, $F : \mathcal{O} \rightarrow \mathbf{Set}$ that produces a concrete realization. It is subject to the operations and composition relations specified by the operad. In practice, an algebra determines:

- For any object $x \in \mathcal{O}$, a set $F(x)$.
- For any morphism in the operad, $f : (x_1, \dots, x_n) \rightarrow y$ and for any element $f_i \in F(x_i)$, a new object $f' = F(f)(f_1, \dots, f_n) \in F(y)$.

Different algebras allow us to “fill” the operad structure from different perspectives and then combine them using a morphism between algebras $\delta : A \rightarrow B$. For example, a *system model* may constitute an operad which can be defined from different perspectives such as, *structure*,

capability or *behavior*. Each viewpoint constitutes an algebra, a way to provide meaning to the abstract structure that can be related to other viewpoints.

Operads constitute higher-order categories, recommended for complex and multi-dimensional perspectives. Some works discussed in Section 6.3 employ this approach. However, they are out of the scope of this research, as the structure provided by categories, functors, and natural transformations is powerful enough for the metamodel defined in Chapter 7.

6.2.4 Combination and Isolation: Pushouts and Pullbacks

So far, we have introduced the machinery to focus on roles and structure via categories and operads. In this work, our aim is to use abstractions to build the best possible system for each situation. Pushouts and pullbacks serve as a valuable tool in this process, as they provide the best approximation of an object in a category that satisfies certain conditions.

Pushouts can be seen as a way to combine two objects in a category with a common object so that all information is preserved. They are usually expressed as Diagram 6.4 in which f', g' are the pushout morphisms that respectively link Y, X with the pushout object $X_{+A}Y$.

If this diagram commutes, i.e., $g' \circ f = f' \circ g$, there exists a unique morphism $u : X_{+A}Y \rightarrow T$ such that Diagram 6.5 also commutes. The dashed arrow represents the unique morphism, and the element T represents the unique pushout object. The arrows l_x, l_y constitute the validity limits of T [190].

$$\begin{array}{ccc}
 A & \xrightarrow{g} & Y \\
 f \downarrow & & \downarrow f' \\
 X & \xrightarrow{g'} & X_{+A}Y
 \end{array} \tag{6.4}$$

$$\begin{array}{ccc}
 A & \xrightarrow{g} & Y \\
 f \downarrow & & \downarrow f' \\
 X & \xrightarrow{g'} & X_{+A}Y \\
 & & \downarrow u \\
 & & T
 \end{array}
 \begin{array}{l}
 \curvearrowright l_y \\
 \curvearrowleft l_x
 \end{array}
 \tag{6.5}$$

Example:

In a category \mathcal{C} with three sets as objects $X, Y, A \in \text{Ob}(\mathcal{C})$. The set A represents three robots, $A = \{\text{robot}_1, \text{robot}_2, \text{robot}_3\}$, set X represents robot grippers, $X = \{\text{finger_gripper}, \text{vacuum_gripper}\}$, and the set Y stores the maximum speed (m/s) in wheeled robots, $Y = \{1.5, 3\}$.

There are two morphisms in this category, $f : A \rightarrow X$ assigning robot_2 to the *finger_gripper* and robot_3 to the *vacuum_gripper*; and morphism $g : A \rightarrow Y$ assigning robot_1 a maximum speed of 1.5 m/s and robot_2 a maximum speed of 3 m/s.

The pushout $X_{+A}Y$ can be seen as the summary of objects and its relationships. In a **Set** category this correspond to the non-disjoint union. In this case, the pushout is a set $X_{+A}Y = \{\text{robot}_1_speed_1.5, \text{robot}_2_speed_3_and_finger_gripper, \text{robot}_3_vacuum_gripper\}$ and its corresponding morphisms to sets X, Y .

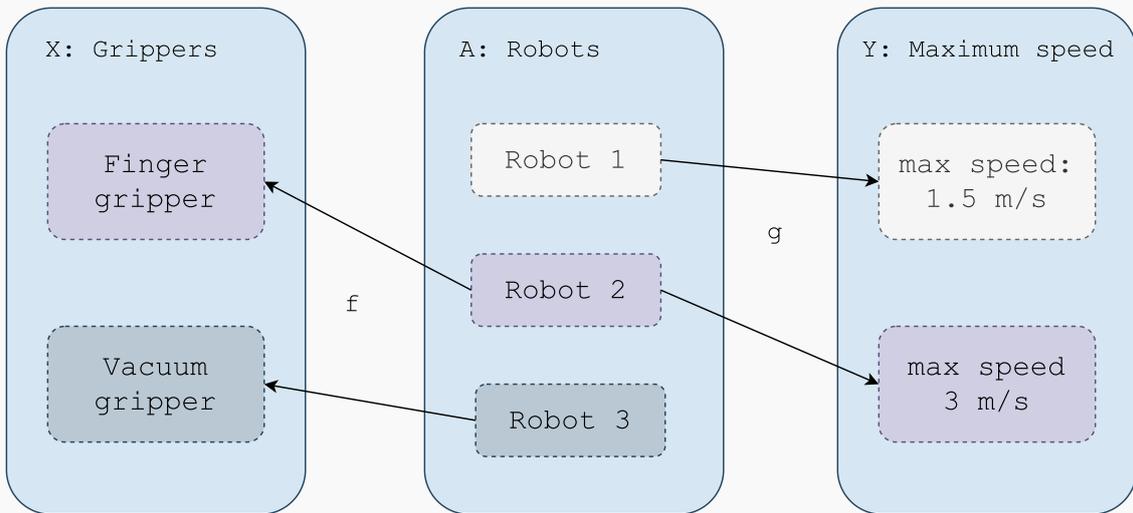


Figure 6.6: Pushout in a Robot category.

The pushout elements of the resulting set are depicted in Figure 6.6 in the same color, as objects connected by morphisms are seen as the same. This notion represents the intuition that the same system is made up of subsystems X, Y, A .

A *pullback* is a construction that allows us to compare two objects in a category by creating a third object that “fits” between them in a particular form. It grants the satisfiability of certain equations or constraints. In a **Set** category, the pullback can be seen as the Cartesian product between common elements, selecting pairs of elements.

Example:

In the previous example, the pullback selects complete information from a robot, i.e., gripper types and maximum wheel speed. In this case, it is the set $X_{\times A}Y = \{\text{robot}_2_speed_3_and_finger_gripper\}$ as it is the only one with information from the three groups. This corresponds to the object in purple in the example above.

Pullbacks are also usually expressed as square diagrams, such as in Diagram 6.6. The pullback is an object $X \times_A Y$ and its morphisms f', g' that project it over Y, X . If this diagram commutes, that is, $g' \circ f = f' \circ g$, there exists a unique morphism $u : T \rightarrow X \times_A Y$ such that the diagram in Diagram 6.7 also commutes. The dashed arrow represents the unique morphism, and the element T represents the unique pullback object. The arrows l_x, l_y constitute the validity limits of T .

$$\begin{array}{ccc}
 X \times_A Y & \xrightarrow{f'} & Y \\
 g' \downarrow & & \downarrow g \\
 X & \xrightarrow{f} & A
 \end{array} \quad (6.6)$$

$$\begin{array}{ccc}
 T & \xrightarrow{l_y} & Y \\
 \downarrow u & & \downarrow g \\
 X \times_A Y & \xrightarrow{f'} & Y \\
 g' \downarrow & & \downarrow g \\
 X & \xrightarrow{f} & A \\
 \downarrow l_x & & \\
 T & &
 \end{array} \quad (6.7)$$

In general, pushouts are used to define categorical quotients, as they combine the common information of a structure $X \leftarrow A \rightarrow Y$; whereas pullbacks are often used to define categorical products, as they find commonalities between a structure $X \rightarrow A \leftarrow Y$.

6.2.5 Equivalence: The Yoneda lemma

There are several concepts in CT that provide different perspectives on representing similarities or relationships between objects in categories. When we say that two entities are the “same,” it means that they are equivalent with respect to some definition or assumptions while ignoring other aspects [191].

The first notion of sameness can be found in *morphisms*. They are the basic building blocks that represent relationships between objects in a category. They constitute transformations between objects that preserve the relevant properties between them. Isomorphisms are arrows that are invertible; i.e., for the morphism $f : A \rightarrow B$ there exists another morphism $g : B \rightarrow A$. For example, if there is a category composed of two elements X, Y representing the sets $X : \{a, b, c\}$ and $Y : \{1, 2, 3\}$; isomorphisms relating the two sets hold the intuition that both sets are the same (in an abstract sense).

Functors map between categories, preserving structures and relationships between objects. They are arrows between two categories. This notion of *sameness* is weaker than above, but represents some equivalence between the two structures.

Lastly, natural transformations describe relationships between functors, associating different mappings between categories and how they relate to each other. This *sameness* is even weaker but provides the possibility of handling several perspectives—objects and morphisms and the structures between them—at the same time.

The most powerful result in comparing two objects in a category is *Yoneda lemma*. According to it, two objects A , B in a category can be considered equivalent if and only if all the relationships that A holds with others in the category are the same as those of B [191]. For example, if two system components are deemed the same based on certain perspectives, such as compatibility in terms of capabilities and interfaces, they can be interchanged when one of them fails, allowing the system to continue its operation seamlessly. Note that this definition of the Yoneda lemma is informal, as it is intended to be used for conceptual application. For a mathematical definition and the proof of the lemma, refer to [183].

6.3 Related Work in Applied Category Theory

The application of CT as a tool for abstract reasoning about mathematical structures has proven to be effective. However, despite its success, CT is not as widely adopted in applied scenarios as other mathematical disciplines such as linear algebra or calculus. This limited adoption may be attributed to its specialized and abstract nature, which comes with a steep learning curve. However, despite these challenges, CT finds applications in abstraction-based domains, like computer science and software engineering, particularly to recognize commonalities and patterns across diverse domains. A notable example is its use in the design and implementation of the C++ Standard Template Library [192].

In recent years, there has been a noticeable increase in the number of fields that implement CT-based approaches. In knowledge representation, Spivak et al. [193] propose the ontology log, *olog*, a CT formalism similar to relational database schemas, providing a friendly definition language and supporting safe data migration [194]. Patterson [195] introduces a relational ontology aligned with this approach, with the aim of bringing ologs closer to *Description Logics (DL)*. These works rely on categories, functors, and natural transformations, utilizing pushouts to combine information and generate new concepts from existing ones.

In the robotics domain, Aguinaldo et al. [196] propose the use of set categories and pushouts to handle the required knowledge for classical planning, capturing implicit preconditions and effects in complex planning scenarios. They leverage an ontology to encapsulate the planning problem, utilizing CT as a formal semantics to model world states and updates within the plan.

Among early adopters of CT in the engineering domain is Censi [197], who introduced the theory of co-design to optimize multi-objective systems. This approach establishes a mathematical foundation for hierarchical design processes [185]. Zardini et al. [198] apply interconnected co-design problems to a self-driving car, utilizing wiring diagrams to optimize resource allocation in the design phase.

Perhaps Lloyd’s vision aligns more closely with our work. Lloyd [178] advocates for the use of CT language and theory to study systems and gives reflections on its implications. His domain of application is biological sciences. In [190], he presents a CT-centered approach to model and simulate the emergent properties of intercellular interactions.

In the *SE* domain, Bakirtzis et al. [179] investigate the unification of requirements, behaviors, and architectures for complex systems. They proposed the use of wiring diagrams as an alternative to engineering modeling languages such as *SysML* and illustrate their approach with a case study involving an unmanned aerial vehicle. Their model relies on “contracted behaviors,” which compose behaviors and requirements through an operad algebra. However, a notable limitation of this approach is the directionality imposed by wiring diagrams.

The generalization provided by operads seems to offer a more suitable structure for modeling complex systems. Schweiker et al. [54] propose the application of the wiring diagrams operad to model safety-critical systems. Specifically, they establish viewpoints with a solid mathematical framework that allows the assessment of behaviors and error propagation from different perspectives. This model is applied to evaluate the failure effects of air traffic management communication channels on aircraft control.

An alternative approach to using operads for system modeling is presented by Breiner et al. [199]. They leverage the inherent separation of concerns in operads to analyze a length calibration device, specifically the length-scale interferometer (LSI), used at the US National Institute of Standards and Technology (NIST). In their modeling approach, system interfaces are represented as a port graph operad. In addition, they employ a functor that links the system and the probability of failure to facilitate the failure diagnosis process. Another example of modeling systems using operads can be found in Foley et al. [200]. They take a top-down modeling approach using wiring diagrams operad to analyze the LSI case from Breiner et al. [199], and a down-top perspective using network operad to design a search and rescue architecture and its design mission task plan.

These works highlight that CT is perceived as a tool to enhance the dependability and explainability of systems, addressing major concerns in technology-centered domains. The increasing number of publications underscores the potential practical utility of CT. However, challenges persist in bridging the abstraction provided by CT with its concrete implementation in real-world scenarios.

In this work, our aim is to advance in that direction by defining a model that is applicable both during the design phase and during robot operation. Our approach aligns with knowledge representation applications of CT, using categories, functors, and natural transformations. In addition, we use pushouts to analyze the information as a whole and derive new facts. Furthermore, the application of the Yoneda lemma enables us to identify similarities within our model. These fundamental elements provide ample power to define a metamodel and reason about its implications, as further discussed in Chapter 7.

Part III

Framework: SysSelf - Systems That Know What They Are Doing

Chapter 7

A Formal Metamodel for Autonomous Robots

Autonomous robot engineering constitutes a major challenge in system development because of its complexity in both hardware/software and operational environments involved. Successfully developing such systems requires meticulous and knowledge-intensive engineering processes that often involve iterative development life cycles.

Our approach requires translating the knowledge of systems engineers into machine-readable models that provide the robot with information about the mission, environment, and its own state. These formal models enable the system to perform runtime reasoning. The solution is rooted in the modeling of the functional architecture of the autonomous robot. The reflective reasoning of the robot facilitates both self-diagnosis and reconfiguration during its operation. Figure 7.1 illustrates the relationship between engineering models and the deployed system.

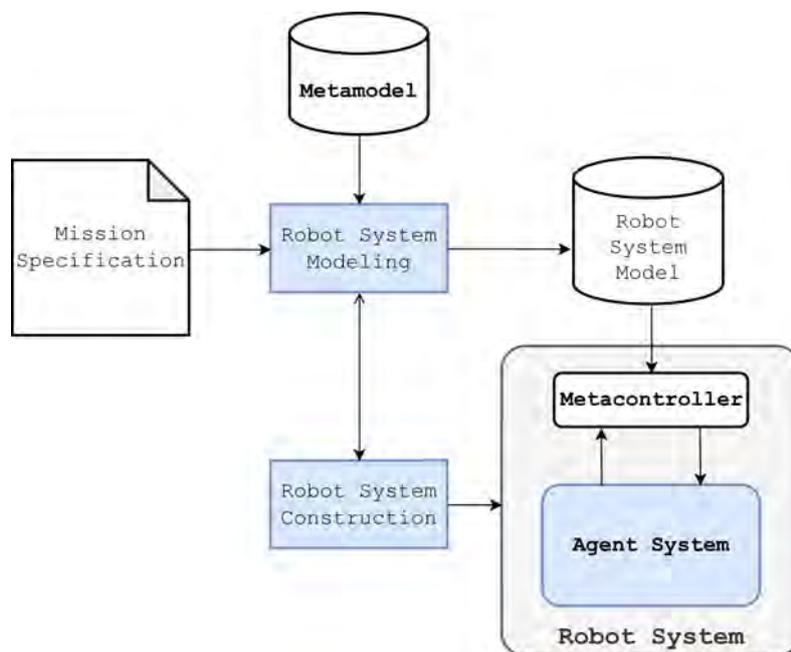


Figure 7.1: Overall metacontrol workflow.

In this chapter, we introduce SysSelf, a formal metamodel that consolidates elements from the preceding chapters with special emphasis on its interconnection with system modeling and construction. The chapter begins with an overview of a pre-existing metamodel from the research group, TOMASys. Section 7.2 describes the implementation details and evaluation conducted to identify the limitations of this approach. Section 7.3 provides a set of requirements for the SysSelf metamodel, aligning with the concepts and foundations discussed in previous chapters and the lessons learned from the TOMASys metamodel implementation. Finally, Section 7.4 describes the SysSelf metamodel, outlining its elements and definitions, and highlighting its unique characteristics using representational structures from *CT*.

7.1 Antecedents: TOMASys Metamodel

The formal metamodel developed in this research is an evolution of a pre-existing metamodel within the research group, *TOMASys*, which has been discussed in Section 5.2, particularly in the context of navigation applications using ontologies. TOMASys is a functional model designed to leverage engineering knowledge generated during the design phase and use it at runtime. TOMASys stands for the Teleological and Ontological Model of an Autonomous System. This model embraces a teleological approach, capturing the intentions and purposes of designers within the system model. Simultaneously, it adopts an ontological perspective to outline the structure and behavior of the system [26], [201].

TOMASys leverages two domains, *MBSE* and *KR&R* through ontologies. This metamodel is used during robot operation to overcome mission contingencies by integrating ontological reasoning into a MAPE-K loop [153] to facilitate architecture analysis and adaptation decisions using the aforementioned metamodel.

The metamodel is influenced by component-based software, guiding the definition of the system structure and functionality. The model elements are categorized into two main groups: static and instantaneous knowledge. In static knowledge, information is stored in Functions and Function Designs. The Function element enables the definition of abstract Objectives for the system, while Function Designs represent design alternatives to execute a Function.

In contrast, instantaneous state information is captured through Objectives, defining a hierarchy of system requirements pursued at runtime, and Function Groundings, instantiating the runtime use of a Function Design. Components are also part of the instantaneous state, representing the structural modules used at a specific moment.

Quality Attributes (*QA*) affect both static and runtime knowledge. They are used to measure how well the system aligns with mission fulfillment. Each Objective is associated with specific requirements related to QAs, such as safety, energy consumption, performance, etc. Each Function Design includes estimated QA values, facilitating the selection of the best alternative based on the situation. Finally, each Function Grounding incorporates measured QA values to actualize the estimations from Function Design with runtime perceptions. An overview of the TOMASys metamodel is depicted in Figure 7.2.

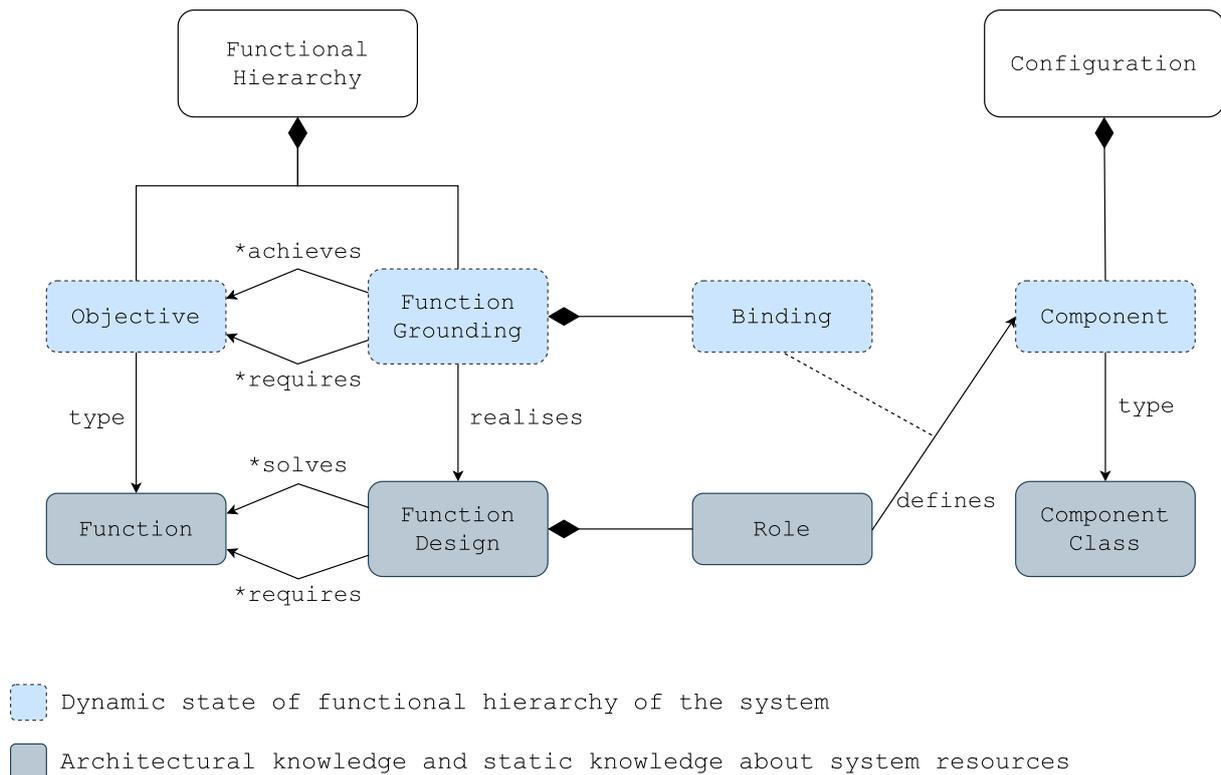


Figure 7.2: Main elements of TOMASys metamodel, the * symbol represents the possible multiplicity in the destination relationship.

The metamodel is implemented as an *OWL-DL* ontology, as introduced in Section 4.4.3, and is publicly available on GitHub¹. In the context of the DL ontology, TOMASys constitutes the TBox (assertion on concepts). The term *TBox* refers to the terminological components of the knowledge base, as opposed to the *ABox* (assertion on individuals), which contains TBox-compliant statements using the defined terminology. Therefore, TOMASys establishes a formal, application-independent vocabulary for developers, promoting reusability across diverse applications, especially in hierarchical, component-based systems.

To utilize the TOMASys metamodel, a system-specific knowledge base is required. The *ABox* defines specific individuals of the application in terms of the TOMASys TBox. It includes information on the system and its mission. The conclusions drawn from the knowledge base are not limited to faults, but also extend to real-time performance and the efficacy of the components involved. This information enables the system to adapt, ensuring continued operation after contingencies, while optimizing its capabilities to achieve the best possible performance.

Therefore, a model ontology comprises two files: the TOMASys TBox and the application-specific *ABox*. These files are then utilized at runtime by a DL reasoner to diagnose the system and compute reconfigurations when necessary.

¹https://github.com/meta-control/mc_mdL_tomasys

Rule no.1 $tomasys:Component(?c) \wedge tomasys:c_status(?c, false) \wedge ux:requiredBy(?c, ?fd) \wedge tomasys:typeFD(?fg, ?fd) \wedge tomasys:FunctionGrounding(?fg) \rightarrow tomasys:fg_status(?fg, INTERNAL_ERROR)$

If a *Component* has a *Component status* in false (in error), and that *Component* is *required* by a *Function Design* with the same *type* as the *Function Grounding* in use, then that *Function Grounding status* is set as *INTERNAL ERROR*.

Rule no.2 $tomasys:FunctionGrounding(?fg) \wedge tomasys:fg_status(?fg, INTERNAL_ERROR) \wedge tomasys:solvesO(?fg, ?o) \wedge tomasys:Objective(?o) \rightarrow tomasys:o_status(?o, INTERNAL_ERROR)$

If a *Function Grounding* has a *Function Grounding status* in *INTERNAL ERROR*, and that *Function Grounding* *solves* an *Objective*, then that *Objective status* is set as *INTERNAL ERROR*.

Rule no.3 $tomasys:Component(?c) \wedge tomasys:c_status(?c, false) \wedge ux:requiredBy(?c, ?fd) \rightarrow tomasys:fd_realisability(?fd, false)$

If a *Component* has a *Component status* in false (in error), and that *Component* is *required* by a *Function Design* then that *Function Design realisability* is set to *false*.

Table 7.1: Semantic Web Rule Language rules for TOMASys 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 Designs* that require unavailable *Components*.

► TOMASYS EXECUTION AT RUNTIME

The incorporation of the metamodel into the agent execution is based in the well-established *Monitor, Analyze, Plan, Execute sharing a Knowledge base (MAPE-K)* loop [40]. The *monitor* stage employs an *observer* to detect *Component* failures or instances where expected QAs are not met.

The subsequent stages of MAPE-K involve reasoning about the model. Upon the creation of an *Objective*, an initial *Function Grounding* is established to define the system configuration. The nominal operation of the executor involves checking if the *Objective* is in an error status, corresponding to the *analysis* phase. This phase uses *SWRL* rules to update the *KB*, with some examples shown in Table 7.1.

Two causes can trigger an *Objective* error: *Component* failure or deficient QA value. When the *observer* notifies either case of malfunction, the *reasoner* updates the knowledge base. In the first scenario, it checks if the disabled *Component* is utilized by the current implementation

(Function Grounding). If the grounded function employs the disabled Component, the Function Grounding status is marked as an error (rule no. 1). Similar rules are applied if the cause is a low QA value, invalidating the grounded Function Design.

In both cases, the error is propagated to the Objective through rule no. 2. Finally, the realisability of Function Designs changes based on the available Components with rule no. 3; a similar rule is applied to invalidate Function Designs with lower expected QAs than required by the mission. Lastly, the *Plan* phase selects the best available Function Design, prioritizing solutions with higher expected QAs. The *executor* grounds this function by carrying out the *reconfiguration*, completing the adaptation process.

7.2 Implementation and Evaluation of TOMASys

We have implemented and evaluated TOMASys in two European projects centered around mobile robotics applications. The first project focused on an underwater mine explorer robot that encountered thruster failures [152]. In the second project, TOMASys was deployed in a mobile robot tasked with patrolling a university campus, addressing challenges related to low battery levels and sensor failures [202], [203], [204]. These applications highlight TOMASys' adaptability and effectiveness in handling real-world scenarios, showcasing its potential to enhance the reliability of robotic systems in diverse operational environments.

7.2.1 An Underwater Mine Explorer Robot: UX-1

The UNEXMIN project² has the purpose of exploring and mapping flooded underground mines. The aim is to provide geological, visual, and spatial data of hazardous areas that currently are expensive and risky to explore. In Europe, there are around 30,000 closed mines with a considerable amount of mineral raw materials. Many of them were abandoned due to low commercial revenue or expensive and dangerous exploration [205]. The UX-1 robot is designed for different applications: open new exploration mines for raw materials, define more informed drilling plans for intricate tunnels and unknown topologies, improve geosciences understanding through new data, exploration of dangerous areas such as nuclear accidents or toxic spills, surveying unstable underwater environments after an earthquake, etc. An image of the UX-1 prototype can be found in Figure 7.3.

To test the TOMASys implementation, only two degrees-of-freedom are taken into account: *surge* and *heave*. Surge is the linear longitudinal (front/back) motion in the terms of the Society of Naval Architects and Marine Engineers (*SNAME*). Similarly, heave is the linear vertical (up/down) motion. The reason to use these directions is because they are the two main movements for traveling during an expedition. Furthermore, each movement uses a different set of thrusters. The reconfiguration action depends to the fault and the affected motion direction. In Figure 7.4, thrusters in blue (T_0 , T_2 , T_4 and T_6) are responsible for the

²<https://www.unexmin.eu/>



Figure 7.3: The UX-1 prototype during operation in Kaatiala Mine (Finland). Image credits: UNEXMIN project.

surge, while the thrusters in green (T_1, T_3, T_4 and T_7) are responsible for the heave. So if T_0 breaks during shaft descent, no reconfiguration is needed until the start of the forward movement, as this thruster is not involved in heave.

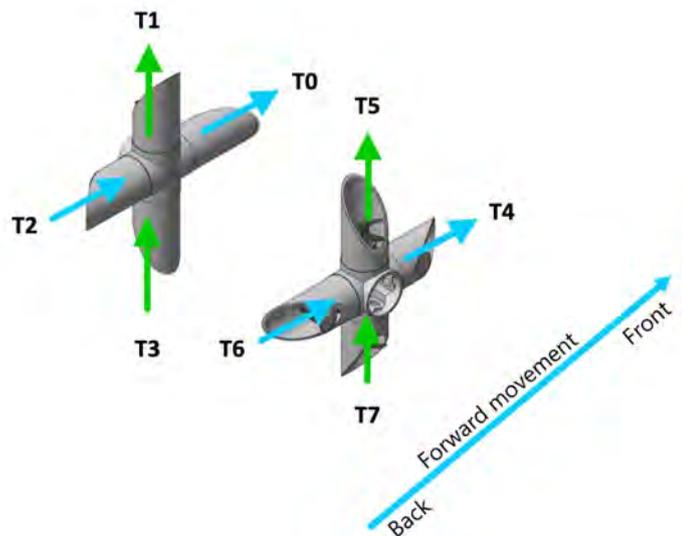


Figure 7.4: Thrusters allocation, the green arrows represent the thrusters implicated on heave movement (T_1, T_3, T_4 and T_7) and the blue ones, the thrusters responsible of forward movement (T_0, T_2, T_4 and T_6).

Table 7.2 collects the thrusters used for each movement direction according to the figure; e.g., if the robot describes a surge motion and T_0 is not available, no reconfiguration action is needed. However, when the robot switches to heave, a reconfiguration action is required to take into account the disabled thruster.

The experiment consists of traversing an L-shaped tunnel, this first Objective is to navigate in the heave direction. When the Objective is created, the selected configuration is the use of the four heaving thrusters for maximum performance. When a thruster malfunctions, for example T_0 , the only available Function Design will heave without T_0 and all surge designs. If the disabled thruster is not used in the current Function Grounding, the Function

Surge	Heave
T_0	T_1
T_2	T_3
T_4	T_5
T_6	T_7

Table 7.2: UX-1 optimal thrusters used according to the movement direction, if one thruster is disabled the system must adapt to use the remaining thrusters in the column.

Design realisability is also set to false, as this result will be used when the Objective changes, e.g., the case while doing a heave movement, when switching to heave without T_0 the adaptation will use the realisability asserted in previous analysis loops.

Further details on how alternative Function Designs were created can be found in Appendix B. The results discussed there show that the use of the modeling approach reduces downtime when a thruster fails. Moreover, it proves the generality of the system engineering process. However, from the point of adaptation response time, it is possible that most fault-handling results for a specific underwater robot in the literature will have better metric values (RMSD and/or latency) than ours, as they are designed specifically for a robot and its concrete operation.

The suitability of using this approach in terms of fault handling depends on the usually soft real-time bounding relation between measured adaptation time and mission requirements. Once the reconfiguration is selected, as the action is the selection of the thruster configuration matrix, its usage is practically immediate. The bottleneck in the use of the reasoner is the time required for asserting the reconfigured system status and choosing the best design alternative depending on the Objective and the contingency.

Nevertheless, the value and strength of the model-based approach are best seen from the SE viewpoint because it provides an application-independent, reusable adaptation engine that can be used to fast-implement a system-tailored module that can be deployed over any extant system with minimal additional instrumentation.

7.2.2 Mobile Robot Laser Contingency

In the EU-funded Metacontrol for Robot Operating System (MROS) project³, we tackle a patrolling corridor mission using commercial robots such as TurtleBot2⁴ and TIAGo⁵.

The experiment is the same in both robots, with the distinction that TurtleBot2 is a simulation, while the TIAGo robot is deployed in a real university setup (Figure 7.5). The experiment

³<https://robmosys.eu/mros/>

⁴<https://www.turtlebot.com/turtlebot2/>

⁵<https://pal-robotics.com/robots/tiago/>

covers two main scenarios involving failure to achieve the required QAs for the mission Objective and a laser contingency that disables the location system.



Figure 7.5: TIAGo robot in a university setup. Image credits: MROS project.

► NOT REACHING REQUIRED QUALITY ATTRIBUTES

The model ontology is designed to assign the mission Objective of navigation along with its associated requirements for energy and safety values. During the execution of the experiment, the MAPE-K loop continuously evaluates the system.

In instances where a measured QA falls below the required value, it starts the analysis process. The reasoner, responsible for selecting Function Designs based on their estimated QA values for energy, safety, and performance, aims to fulfill the Objective requirements. The adaptation candidate is chosen among the Function Designs with the highest performance estimation that also meets the specified requirements.

► LASER NOT AVAILABLE

For the laser unavailability scenario, a simulated laser error is introduced at a random point in time. The corrupted data simulate realistic conditions, as if something was blocking the laser or if the laser was misaligned due to a hit or fall of the robot. Upon receiving a signal of laser unavailability from the observer, the model updates the Component status accordingly. This scenario is similar to the underwater robot case, where a critical component is non-functional.

The reasoner actively seeks alternative Function Designs that do not involve the faulty laser Component. It identifies an RGB-D camera that can provide similar functionalities, although with lower performance. The use of the camera requires to reduce the robot speed to not compromise the integrity of the system and other elements in the surroundings, as the update rate of camera sensory inputs is lower than in the laser sensor. This information is encoded in the expected QAs of that Function Design.

In response to the laser unavailability, the reconfiguration action involves executing the

necessary mappings between camera messages and laser messages, along with adjusting the speed constraints. Although this solution may not meet the required QAs, it ensures the completion of the mission, even in a degraded mode.

► CONSISTENCY OF THE MODELS USING TOMASYS

This project also included an evaluation of the TOMASys metamodel and its OWL implementation, focusing on internal consistency and runtime performance.

To validate the consistency of the internal metamodel, an instance model was created that contains almost all the metamodel elements. Smaller instances were generated to instantiate elements not included in the large instance due to exclusion constraints in the metamodel.

The consistency of the generated model files was then checked using OWL tools to assess KB coherence and consistency, addressing the issue of local consistency checking (model finding). The Debugger Tool in Protégé [94] was used to debug assumptions and laws, and queries were formulated to test expected system behavior based on runtime events.

Specifically, tests were conducted to ensure that property assertions were consistent, focusing on execution of SWRL rules correctly modeled the propagation of laser sensor failure in the ontology. The Debugger Tool was especially valuable in identifying ontology inconsistencies and providing possible repairs.

► RUNTIME USE OF TOMASYS MODELS

In terms of the runtime use of TOMASys models, reconfiguration is driven by ontological reasoning on the KB. The reasoning time performance was tested in relation to KB size, which directly correlates with the possible *Function Designs* defined in the application model. Using the ontological reasoner and an automated individual generator, the performance of SWRL rules was evaluated for models of different sizes.

Performance has been tested by generating *Function Design* individuals in the ontology, with each having three QA estimations (performance, safety, and energy). Initially, the ontology had five individuals, representing three QA types, the *Objective of navigate*, and the required laser and battery *Components*. When a new *Function Design* alternative was added, four more individuals were included. Table 7.3 shows the different model sizes tested, illustrating the increase in the number of individuals and assertions for classes, object properties, and data properties. Additionally, the growth of file size with an increasing number of individuals was captured. The most notable observation was the exponential increase in reasoning time, as depicted in Figure 7.6, where the test data align well with the inferred exponential equation ($y = 681e^{0.0002x}$) adjusted with an R^2 of 0.99.

For the nominal evaluation ontology, with 26 individuals, the estimated reasoning time is 695 ms, corresponding to a frequency of 1.4 Hz. This frequency is considered suitable for runtime adaptation. The measured maintenance time for laser failure using our approach is 1.995 s, with a standard deviation of 0.478 in 50 iterations of the experiment. This is

Number of Function Designs	Individual Count	Class Assertions	Object Property Assertion	Data Property Assertion	File size (KB)	Reasoning Time (ms)
1	9	9	5	3	26	660
10	45	45	50	30	36	704
100	405	405	500	300	134	741
300	1205	1205	1500	900	353	914
600	2405	2405	3000	1800	682	1067
900	3605	3605	4500	2700	1024	1378
1000	4005	4005	5000	3000	1126.4	1617

Table 7.3: TOMASys ontology models of different sizes used to evaluate runtime reasoning.

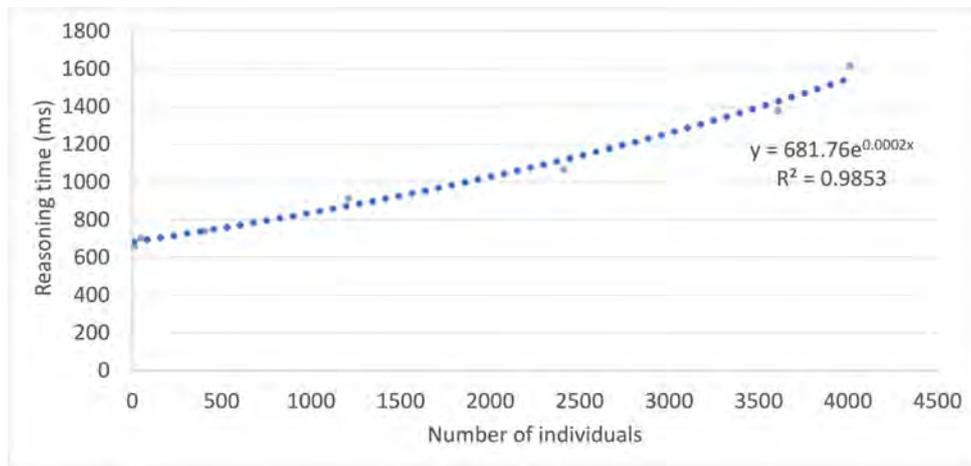


Figure 7.6: Reasoning time in (ms) according to the number of individuals in the ontology. Also shown its corresponding exponential trendline ($y = 681e^{0.0002x}$ with R^2 of 0.99).

compared with the estimated maintenance time without automatic recoveries, which implies restarting the laser module and is expected to take around 300 s [203].

In summary, the TOMASys metamodel provides a baseline with a clear separation of the terminological part and general rules, creating a set of reusable application-specific elements for modeling the functional and physical component architecture of systems. Furthermore, it offers an implementation that supports functional reasoning at the abstract level of needs and capabilities, akin to how an engineer would approach it, rather than focusing on the lower level of concrete realizations. The practical results showcase the effectiveness of the approach in achieving fast adaptation and reducing downtime in the face of faults.

7.2.3 Lessons Learned from TOMASys Implementation

TOMASys uses *OWL* ontologies to deploy its models in applications, employing *OWL* reasoners and *SWRL* rules. However, we have identified some limitations in its application:

- *Restricted reasoning capabilities:* Most reasoners are concerned with classification based on classes and relations, offering only consistency checking. The logic used in OWL enforces monotonic reasoning, which poses challenges when applied to a robot operating in a dynamic environment. The inherent dynamism is not easily handled by this type of logic and OWL reasoners. To address this limitation, we propose incorporating additional formalisms based on CT to track dynamic changes. Alternative options include employing other logics, such as modal logic, which implements dynamic logic according to possible world semantics, or temporal description logic to handle time-dependent processes and subsystems with different life cycles [202].
- *Computationally expensive:* The TOMASys reasoner operates at a frequency of 1.4 Hz [203]. This periodicity is the result of adopting a cyclic execution approach inspired by the MAPE-K adaptation control loop. Although TOMASys benefits from SWRL rules for enhanced expressiveness, they also contribute to increased reasoning time [206].
- *Design constraints:* While the TOMASys metamodel captures information about components, functions, and goals, this information represents only a portion of a system's architecture. The model could be expanded with other *MBSE* concepts to provide a more complete representation.

7.3 Requirements of the SysSelf Metamodel

The aim of the metamodel is to define the minimal set of concepts to reason about mission execution and how to overcome challenges or, at the very least, complete tasks in a degraded mode. For this purpose, the following set of properties and requirements have been elaborated for the SysSelf Metamodel, taking into account the lessons learned from the TOMASys implementation discussed above.

1. *System Organization:* The metamodel shall explicitly capture information about the system's main aspects, especially when redundancies exist in components or capabilities, or when there are multiple approaches to achieve the same outcomes (alternative plans or goals).
2. *Declarative Formal Language:* The metamodel shall produce a language based on well-established engineering concepts using a declarative formal language, making it machine-readable and capable of using reasoning mechanisms to update its content.
3. *Reuse of Existing Definitions:* Whenever possible, the metamodel shall reuse existing expressive definitions rather than recreating them.
4. *Value-Oriented:* The metamodel shall be value-oriented, enabling reasoning to decide the appropriate course of action according to robot goals and user expectations.
5. *Expressive Formal Language:* The metamodel shall be written in a formal language expressive enough to capture domain constraints and draw inferences based on constructs such as numbers, arithmetic, goals, and capabilities.

6. *Runtime Executability*: The metamodel shall be executable at runtime to correctly update the system behavior according to mission requirements.
7. *Applicability to Different Systems*: The metamodel shall be applicable to different systems with various components, tasks, or deployments in a variety of environments.

Requirements number 1 to 4 are met by using concept definitions based on terms and sources from *SE* as outlined in Chapter 3. Requirement 5 is achieved by utilizing the abstract mathematical framework from Chapter 6, specifically *Category Theory (CT)*. The details of Requirement 6 are further elaborated in Chapter 8, focusing on the runtime usage of the metamodel. Finally, Requirement 7 is assessed in Chapter 9, where the model is applied to two robotic systems—a miner robot and a mobile robot—experiencing different contingencies during its operation.

Regarding the limitations identified in TOMASys, as discussed in Section 7.2.3, the SysSelf metamodel aims to broaden its scope to include daptation directed towards capabilities and values. Furthermore, SysSelf can be employed to provide valuable information to the responsible user or concerned stakeholders when system changes occur, thereby enhancing transparency and facilitating decision-making. Lastly, the SysSelf metamodel strives to mitigate the computational cost of its deployment. However, this aspect will be discussed in Section 8.1 as it represents an operational facet of the metamodel.

7.4 SysSelf: A Metamodel for Systems that Know What They are Do(ing)

A metamodel is a model of models [207]; it defines the language for expressing models. This section introduces the SysSelf metamodel as the core language for capturing system engineering knowledge that may be relevant for its adaptation. For each application, this metamodel shall be particularized with instances in each category. The *SysSelf metamodel* is designed to enable the robot to adapt to robustly deliver the expected value, even in the face of unexpected contingencies. As it is intended to leverage engineering knowledge for robot adaptation, it aims to enable systems to provide some knowledge on why they are adapting and how they should do it to comply with user expectations.

This metamodel should be seen as a specification language providing a formal vocabulary. For this purpose, we use definitions from the SE domain extracted from INCOSE [4], ISO/IEC/IEEE 15288:2023 Standard on Systems and Software Engineering [59], and SEBoK Wiki⁶ [19].

Table 7.4 shows the main concepts in the SysSelf model. It includes terms such as value, component, and capability, as well as diagnostic concepts such as component status and goal status or measure of performance and effectiveness. Perhaps the most characteristic aspect

⁶[https://sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_\(SEBoK\)](https://sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK))

Concept	Definition	Related to
Capability	Expected operational outcome (function) of the system that satisfies a need under specified conditions with some (performance) standards [19].	Constraint, Value, Component, MOP
Component	Generic term for the level of decomposition at which system elements are no longer considered complex, and for which specialist design disciplines can be used [19]. One of the parts that make up a system [208].	Constraint, Goal, Capability, Interface, TPM
Constraint	A restriction, limit, or regulation imposed on a product, project, or process [19].	Capability, Component
Goal	A goal is a specific task achievable in a set time [19].	Component, Value
Interface	Boundary used to connect two or more components for the purpose of passing information from one to the other [69]	Component
Measure of Effectiveness (MOE)	The metrics by which an acquirer will measure satisfaction with products produced by the technical effort [209].	Value
Measure of Performance (MOP)	An engineering performance measure that provides design requirements that are necessary to satisfy a MOE (measure of effectiveness) [209].	Capability
Technical Performance Metric (TPM)	Measures of attributes of a system element within the system to determine how well the system or system element is satisfying specified requirements [210].	Component
Stakeholder	Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations [209].	Value
Value	Amount of benefit something, feature or capability provides to various stakeholders. It is an attribute of the problem space and considers the optimum set of requirements to deliver customer satisfaction [19].	Stakeholder, Capability, Goal, MOE

Table 7.4: Main concepts of SysSelf metamodel and relationships between them.

is that it is expressed in terms of *CT* to establish inferences at higher levels of abstraction, which can reveal new insights and facilitate decision-making.

7.4.1 Categories and Functors in SysSelf

One of the design decisions of this model is to be formalized using *CT*. The primary elements of the model are value, component, capability, and goal. These concepts are represented as categories in the SysSelf metamodel. The relationship between the terms can be seen in the case of an artificial system composed of *components* that have *capabilities*, allowing the system to achieve some *goals* that address certain *values* for the robot owner.

As stated in Chapter 6, to define a category, we shall specify objects and morphisms along with the composition operation and the identity morphism. The four categories are defined as follows:

- **Component:** in which the objects are the various parts and subsystems that make up a robotic system (e.g., motors, sensors, controllers, etc.) and the morphisms are the mappings or transformations between different components, i.e., the interfaces between them.
- **Capability:** in which the objects are the capacities to perform a certain function (e.g., sense, move, decide, plan, etc.) and the morphisms the ways in which these capabilities can be composed to perform more complex tasks.
- **Goal:** in which the objects are the desired outcomes commanded to the robot (e.g., extract X quantity of mineral Y or move to $[X, Y]$ position) and the morphisms are mappings from one goal to another.
- **Value:** in which the objects are the advantages that a robotic system provides (e.g., efficiency, safety, precision, etc.) and the morphisms the mappings between values, how they are related.

The remaining elements for composing the category include composition and identity. Composition is illustrated as the chain of relationships between objects. For example, the interfaces between the robot chassis, motors, and wheel to enable the robot's movement. Similarly, the identity morphism represents the reflexive relation with respect to each object, representing how a chassis or the capability to move relates to itself.

A *functor* F is defined as a mapping between categories, comprising two dimensions: a mapping between objects and a mapping between morphisms. The SysSelf metamodel defines the four main functors from Expression 7.1.

$$\begin{aligned}
F_{realizes} &: \text{Component} \rightarrow \text{Capability} \\
F_{enables} &: \text{Component} \rightarrow \text{Goal} \\
F_{contributes\ to} &: \text{Capability} \rightarrow \text{Value} \\
F_{aligned\ with} &: \text{Goal} \rightarrow \text{Value}
\end{aligned}
\tag{7.1}$$

This structure can be represented as the commutative diagram shown in Diagram 7.2. However, commutative diagrams in CT typically depict objects as nodes and morphisms as arrows. In this case, we represent categories as nodes and functors as morphisms, capturing the essence of a higher-order category: the SysSelf category. The commutative condition implies that the Value, destination of the square, can be reached from any of the two paths, that is, using the Components and Goals or Components and Capabilities.

$$\begin{array}{ccc}
\text{Component} & \xrightarrow{\text{enables}} & \text{Goal} \\
\downarrow \text{realizes} & & \downarrow \text{aligned with} \\
\text{Capability} & \xrightarrow{\text{contributes to}} & \text{Value}
\end{array}
\tag{7.2}$$

7.4.2 The SysSelf Category

This section details the characteristics of the SysSelf category built from the categories and functors introduced above. Note that the name *SysSelf* is used to denote two related, yet distinct aspects: a category and a metamodel.

- *SysSelf category*: Corresponds to the abstract mathematical structure composed of objects and morphisms further described in this section. It defines the fundamental aspects of the metamodel and is used for higher-level reasoning, as described in Section 8.2.
- *SysSelf metamodel*: Specification of the abstract syntax, semantics, and constraints of the model. It could be seen as the modeling language. While the fundamental structure is provided by the main category, the metamodel additionally defines other important concepts and relationships relevant for autonomous robot operation. It includes the definitions from Table 7.4 and is used for ontological reasoning.

The SysSelf category is a 2-category. A 2-category is a mathematical structure that extends the concept of a category. While a category is built upon objects, morphisms, and composition of morphisms, a 2-category introduces another structure known as *2-morphisms*, representing relationships between morphisms. This additional layer of abstraction provides a framework for capturing higher-order relationships and interactions. Further details on 2-categories can

be found in [211]. The archetypical 2-category is **Cat**, the 2-category whose (i) objects are categories, (ii) morphisms are functors, and (iii) 2-morphisms are natural transformations.

The SysSelf category constitutes a particular case of the **Cat** category. However, to avoid unnecessary complexity, we do not include natural transformations, i.e., 2-morphisms. In concrete, the SysSelf category is composed of:

- *Objects*: The four categories defined above: Component, Capability, Goal, and Value.
- *1-Morphisms* (also morphisms or 1-arrows): The four functors between the categories above represented in Expression 7.1. This functor must preserve the identity morphisms and composition of morphisms within categories.
- *Composition of 1-morphisms*: Similar to the fundamental categories, composition is accomplished by chaining the functorial relationships. For example, the path to reach the category of Value using Components and Goals, one combines $F_{enables} : Component \rightarrow Goal$ and $F_{aligned\ with} : Goal \rightarrow Value$ functors.
- *Identity 1-Morphisms*: As in the fundamental categories, the identity functor for each category serves as a neutral element with respect to composition—a reflective relation within its own concept.

As this category does not employ the additional 2-morphism structure, it behaves similarly to a standard category. This characteristic facilitates the application of other CT concepts such as pushout and the application of the Yoneda lemma, as described below.

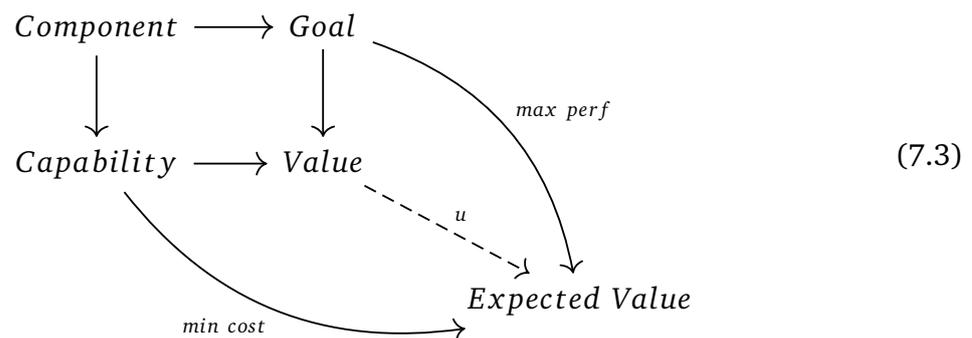
7.4.3 Value and the Concept of Pushout

For the SysSelf metamodel, value takes a central role, similar to *SE*, which focuses on systems delivering some desired benefits. In this *context*, value is derived from the provided capabilities or improvements, not inherent in the system itself. The system serves as a means to attain it.

The concept of *pushout* in the SysSelf category can be employed to reason about value as the holistic properties of the system, including components, capabilities, and goals; and the outcomes of their combinations. In CT, the concept of pushout is frequently utilized to summarize information about its elements, as described in Section 6.2.4.

Given the *commutative diagram* around the concepts of Capability, Goal, Component, and Value, as represented in Diagram 7.2, and the fact that this diagram commutes, Value can be reached from either of the two alternative paths. There exists a unique morphism $u : Value \rightarrow Expected\ Value$ such that Diagram 7.3 also commutes. The dashed arrow represents the unique morphism, and the element Expected Value denotes the unique pushout object. In robotics terminology, this object can be understood as the anticipated behavior and performance the robot should exhibit. Lastly, the minimum cost and maximum performance morphisms constitute the constraints on the Expected Value.

The primary purpose of the SysSelf metamodel is the support of the engineering processes to assure the delivery of value for a system. This model can be used to facilitate the adaptation of the system in the presence of contingencies, ensuring the continued delivery of the *expected value*. The pushout concept can be used to identify possible alternative designs to meet user expectations. This approach involves following the pushout (Diagram 7.3) to determine the optimal combination of Components, Capabilities and Goals which maximizes the Expected Value of the system. In event of a critical incident, reaching the expected value may be unattainable, but this structure uncovers the best possible solution and how this value is affected, such as in terms of performance and cost.



Example:

This case is inspired by the mobile robot laser contingency discussed in Section 7.2.2 and derived from the Marathon 2 experiment by Macenski et al. [212], using Nav2, the widely adopted ROS2 stack for perception, mapping, localization, path planning, and control. This engineering knowledge is represented in Figure 7.7.

In a direct instantiation of the SysSelf category, we may have four objects in the component category: the Adaptive Monte Carlo Localization (AMCL) [213], the A* planner [214], and two sensors for functional redundancy—a LiDAR and an RGB-D camera. The goal is to reach a set of 16 waypoints, and the available capabilities include planning, map updating, control, and navigation.

In this simple scenario, the primary value provided by the system is to traverse the 16 waypoints. In a real-world deployment, other possible provided values could be delivering a specific object or generating a map of a particular area. Other parameters defining the expected value include time efficiency, point accuracy, energy consumption, and safety.

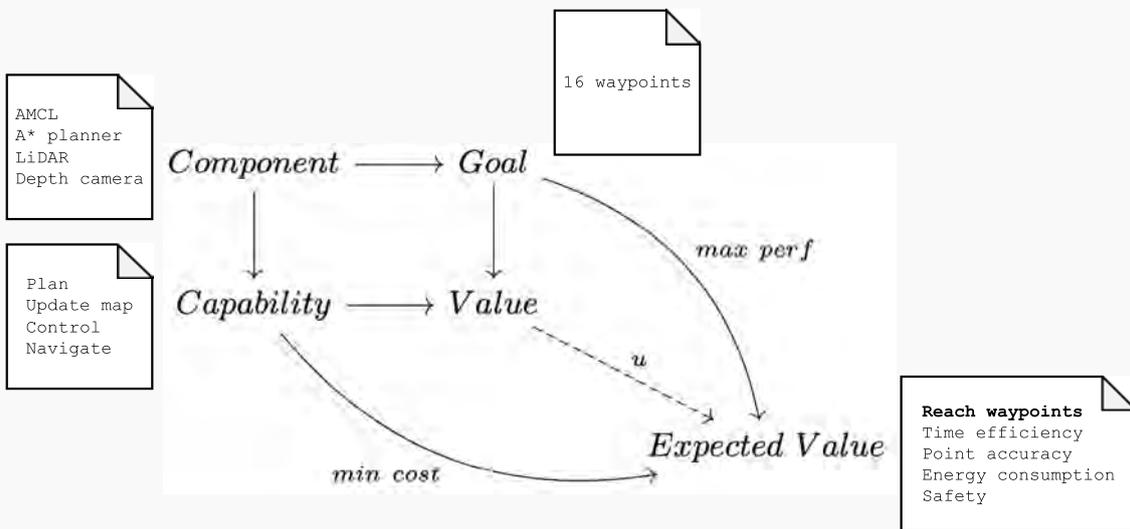


Figure 7.7: Navigation 2 engineering knowledge represented in the SysSelf category.

Different realizations of the system may lead to varying provided values. However, this approach allows us to reason in terms of always staying within an acceptable range of the expected value. Note the relation of this approach with the system architecture evaluation process in SE (and furthermore with verification and validation stages in the system life cycle).

7.4.4 Adaptation and the Yoneda lemma

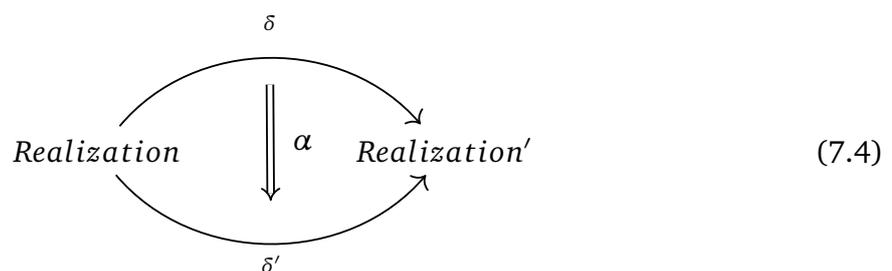
When contingencies arise, systems equipped with functional redundancy can adapt their structure to keep fulfilling their mission. The concept of *functional redundancy* ensures that

the system has several design alternatives when executing a task. These alternatives may include variations in goals, capabilities, or components.

Representing a system as a particular realization corresponds to an instantiation of the SysSelf category within the **Realization** category. After the system adapts, the model evolves into an adapted realization with a corresponding **Realization'** category.

Both categories are connected by two functors (δ , δ') that represent how the system has changed—specifically, how components, goals, and capabilities are affected. Additionally, there is a natural transformation (α) between them, representing the relationship between these changes.

In this context, the term *adapt* refers to the application of a natural transformation between two realizations where objects are considered *the same* from a particular perspective (Diagram 7.4). This concept of sameness aligns with the *Yoneda lemma*. According to the lemma, two objects are considered identical if all their relationships are the same. For example, if two components are compatible in terms of capabilities and interfaces, they can be interchanged, allowing the system to continue its operation. Therefore, the Yoneda lemma is the mathematical concept that enables the identification of adaptation candidates.



Example:

In the given scenario, if the primary sensor (LiDAR) is damaged, a system adaptation could involve switching to a depth camera to fulfill the sensory input requirements for the localization subsystem.

Although both components serve the same functionality, they are not directly interchangeable in terms of interfaces (Figure 7.8). To address this, an additional component for data transformation is necessary. We use the point cloud to laser scan ROS node^a to converse readings from the depth camera (point cloud messages) to simulate LiDAR readings (scan messages).

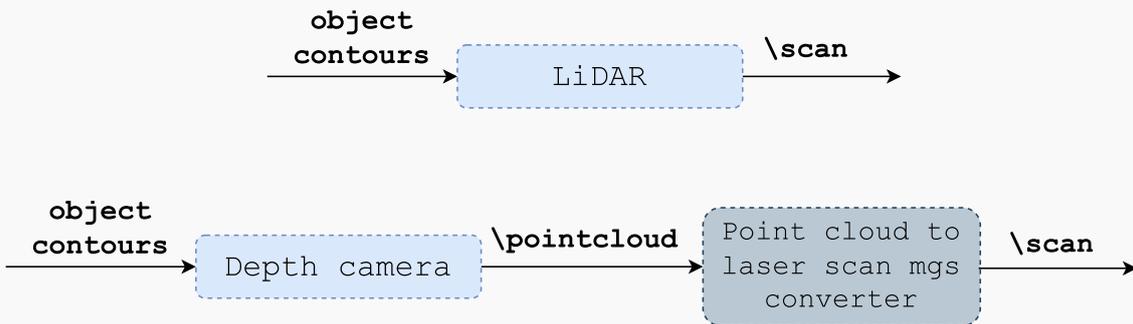


Figure 7.8: Equivalence in terms of component functionality.

In this context, the category **Realization** represents the system utilizing a LiDAR for localization, while the category **Realization'** corresponds to the adapted configuration employing a depth camera with interface transformations.

In the new realization, by replacing the LiDAR with the depth camera, the value is affected by the characteristics of the latter, which has lower resolution and a lower publication rate. These aspects result in reaching the waypoints with less accuracy and lower time efficiency, as the robot speed slows down. However, this change provides less energy consumption and enhances safety. The most significant result is that the task is completed even with the LiDAR out of operation.

^ahttp://wiki.ros.org/pointcloud_to_laserscan

7.4.5 The SysSelf Metamodel

Besides components, capabilities, goals and values, the SysSelf metamodel incorporates other relevant elements from SE. Specifically, various metrics related to these concepts, such as *Measure of Effectiveness (MOE)*, *Measure of Performance (MOP)* and *Technical Performance Measure (TPM)*. These concepts are essential for understanding how the expected value is affected after adaptation.

The metamodel also introduces the concept of *stakeholders*, representing individuals or

organizations with an interest in the system [59]. Identifying stakeholders helps clarify who may be concerned with the result of adaptation, providing explainability based on specific interests. In the case of critical contingencies, it may contribute to identifying where human intervention or maintenance may be required.

Additionally, the metamodel includes concepts related to the usability of the model, such as elements to store values on component and goal statuses. Moreover, the metamodel is enriched with other concepts relevant to specific applications, such as environmental and component constraints, interfaces, or functions that provide more information to select the best adaptation candidates.

The complete form of the SysSelf metamodel is illustrated in Figure 7.9. Note that the central structure is the commutative diagram characteristic of the SysSelf category and how it relates to the rest of the concepts in the metamodel.

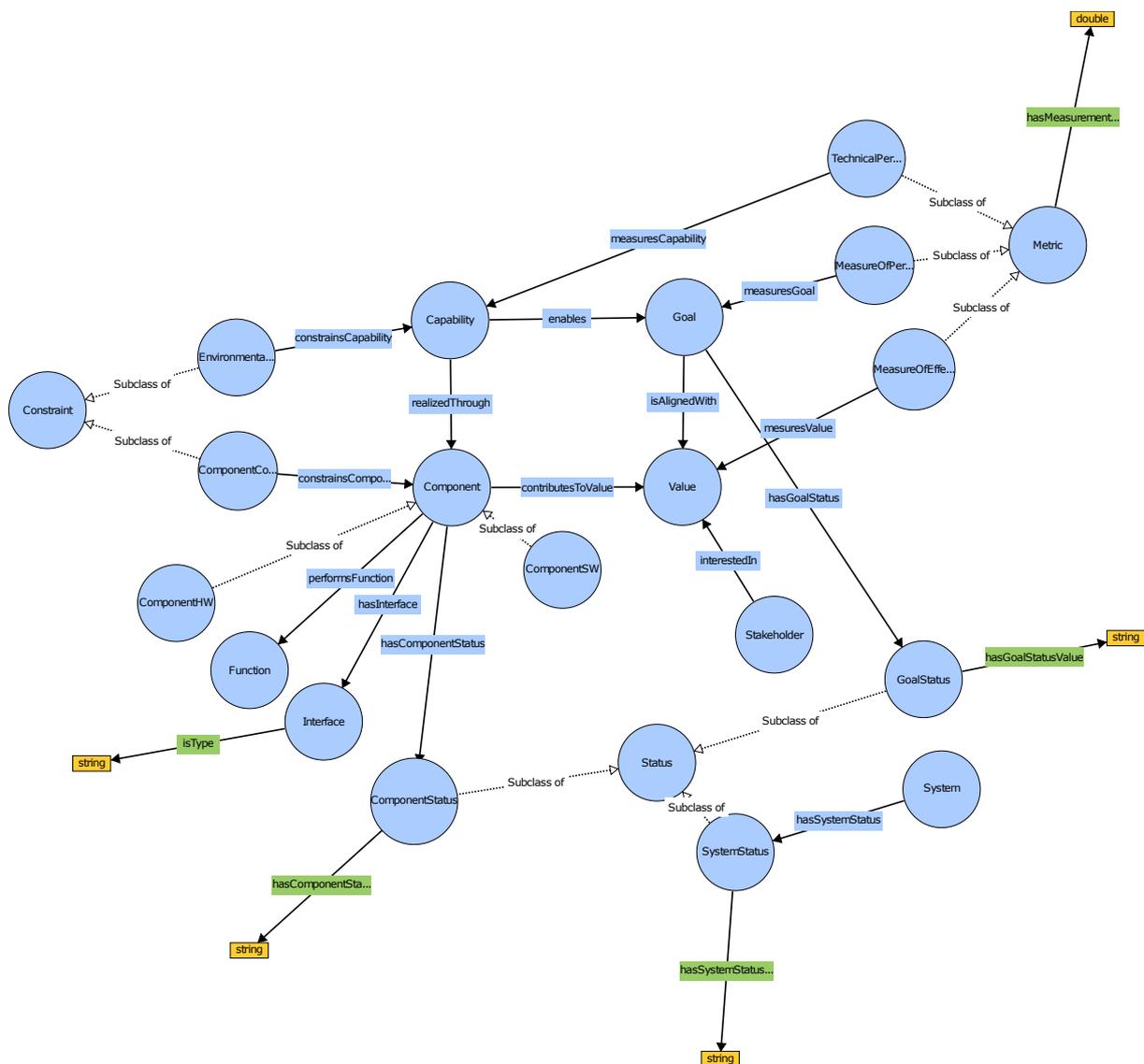


Figure 7.9: SysSelf metamodel, main elements and relationships.

The metamodel provides the infrastructure to incorporate the most relevant aspects from engineering design with an impact on the potential adaptation of systems. Reasoning around these concepts allows the system to respond “as an engineer would” when an unexpected event occurs. The following chapter provides details on how the SysSelf metamodel is utilized at runtime.

Chapter 8

An Operational Metamodel: Metacontrol

Once there is an explicit metamodel serving as a pattern to enhance autonomous systems' self-awareness, this chapter introduces the architecture designed to leverage it, the metacontroller. The concept of a reference architecture to close a loop on top of the system controller was first introduced by Sanz et al. [215; 42] and developed by Hernández et al. [26; 201] to handle the TOMASys metamodel already described in Section 7.1.

The metacontroller serves as the instrument to pursue mission goals in the presence of disturbances. It functions as a modular tool to control the structure and behavior of the robot, to continue providing the expected values. Similar to a traditional controller, which is tasked with maintaining a variable value close to a specified reference set-point, the metacontroller takes the specified functionality of the system as a reference. In the event of deviation from this functional reference, a system reconfiguration becomes necessary. The new and suitable configuration is derived from the KB, which stores specific aspects of the robot and its mission. Figure 8.1 illustrates the relationship between the system, its controller, and the metacontroller. Note that this approach is aligned with the architecture of a fault-tolerant control system (as seen in Figure 2.3) and adapted from [27].

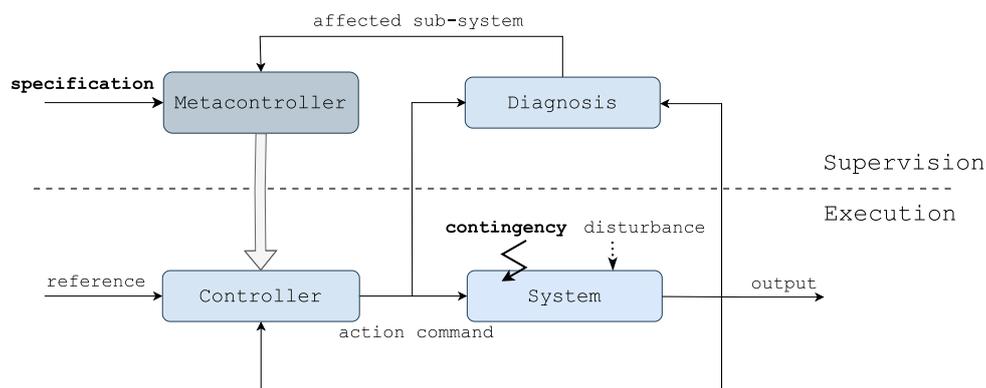


Figure 8.1: Architecture of a system using a metacontroller to face contingencies, inspired from [27].

Therefore, the metacontroller shall be tailored to reason about terms of the SysSelf metamodel in an abstract and multi-applicable way, enabling specific adaptation actions for the system. It is designed under two main requirements: (i) use a declarative approach to decouple application and implementation, and (ii) be reusable and modular.

This chapter is organized as follows. Section 8.1 describes the encoding process of the SysSelf metamodel in an ontology. In Section 8.2, we justify the choice of reasoners used to leverage the model at runtime and explain the execution process of the metacontroller for this purpose. Finally, Section 8.3 outlines the software assets provided to seamlessly integrate this solution with popular robotic frameworks and Section 8.4, the step-by-step process that needs to be carried out to use the framework.

8.1 Metamodel Operationalization: The SysSelf Ontology

The SysSelf metamodel requires transformation into a machine-readable model for effective utilization by the metacontroller. However, applying *Category Theory (CT)* in engineering domains often faces challenges due to the limited tool support. Although there is a developing framework called *Catlab*¹ designed for applied and computational CT, we opt for using *Ontology Web Language (OWL)* due to its better tooling support and ease of integration with other subsystems. This decision is reinforced by the survey conducted in Chapter 5, which revealed that OWL is the most widely used declarative language, especially in the field of robotics.

The decision to use CT fundamentals for the metamodel, rather than relying solely on ontologies, may raise questions. However, ontologies exhibit certain limitations, particularly in the scope of reasoning they support and their associated computational costs. These limitations were previously discussed in Section 7.2.3 as lessons learned from the SysSelf predecessor, TOMASys.

An OWL ontology is made up of *classes*, corresponding to concepts in the domain of discourse and properties. There are two types of relationships: object properties and data properties. *Object properties* establish connections between instances of two classes, while *data properties* are relations between instances of classes and data types, assigning values to variables. The SysSelf ontology contains concepts and relationships derived from the SysSelf metamodel, as defined in Section 7.4 and represented in Figure 7.9.

► FOUR PRIMARY CLASSES AND RELATIONSHIPS

The four primary categories, namely *capability*, *component*, *goal* and *value*, establish the main classes within the ontology. These classes are interrelated by object properties corresponding to the four functors. Specifically, the relationships are as follows: *contributes to goal* links components and goals, *contributes to value* links capabilities and values, *realizes* (with its inverse *is realized by*) links components and capabilities, and *is aligned with* relates goals and values. Furthermore, there is the reflexive relation *composable with capability*, representing the composition of capabilities.

¹<https://github.com/AlgebraicJulia/Catlab.jl>

This structure, depicted in Figure 8.2, represents the key elements involved in the metacontroller's reasoning process to determine the most suitable adaptation mechanism.

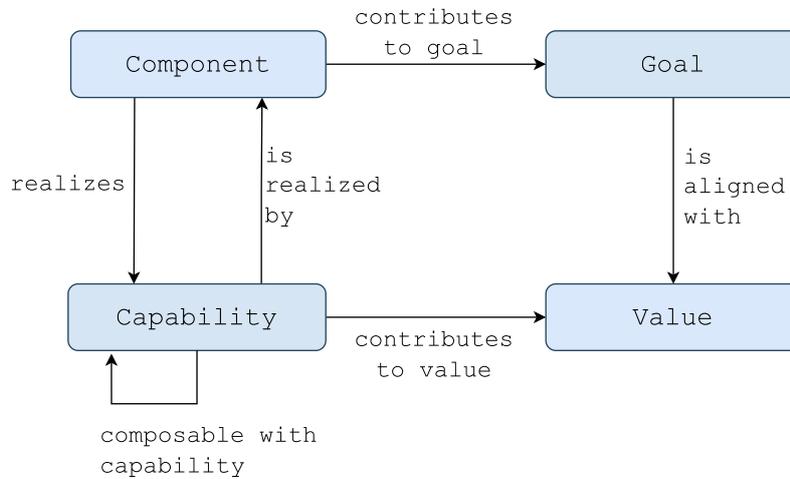


Figure 8.2: Main classes and relationships of the SysSelf ontology.

► PERFORMANCE AND USER-INVOLVED CLASSES AND RELATIONSHIPS

Additionally, there are performance-related elements corresponding to the *metrics* class, further categorized into *MOE*, *MOP* and *TPM*, each addressing specific aspects of the system. The *MOE* class quantifies how the provided value is affected by the efficiency of a system. In contrast, the *MOP* class uses property measures to determine the system performance on achieved capabilities. The *TPM* class focuses on component-level performance measures to assess the functioning of individual elements. Additionally, a reflexive property named *affects metric* describes how distinct metrics influence one another. Finally, the *stakeholder* class is associated with *value* through the *interested in* relationship, allowing the metacontroller to provide relevant information to the parties involved on how the system is expected to change after adaptation. Figure 8.3 represents these classes and relationships.

► SOLUTION-SPECIFIC CLASSES AND RELATIONSHIPS

Moreover, there are supplementary classes designed to capture additional aspects of the system that contribute to the reasoning process but are not considered essential (Figure 8.4). This set includes the *component constraint* (associated with the *constraints component* object property) and *environmental constraint* (linked to the *constraints capability* object property). Additionally, there are the *component status* and *goal status* classes, each with their respective relationships. Furthermore, the *interface* class is connected to the components through the *has interface* relationship. These elements, together with other potential classes and relationships that application engineers may require, comprise the solution-specific entities that can be added without affecting the metacontroller reasoning. These relationships serve as morphisms within the component, goal, capability, or value categories.

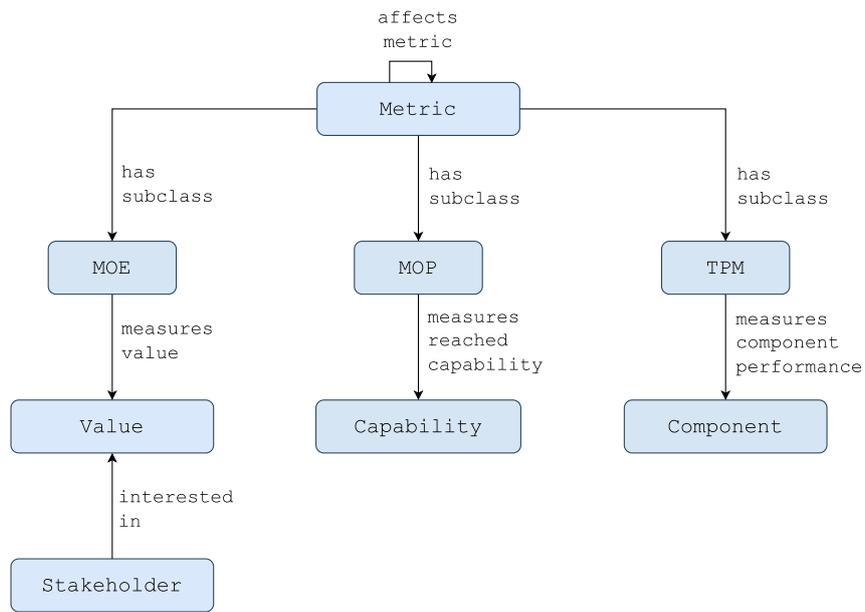


Figure 8.3: Classes and relationships of the SysSelf ontology related with metrics and stakeholders.

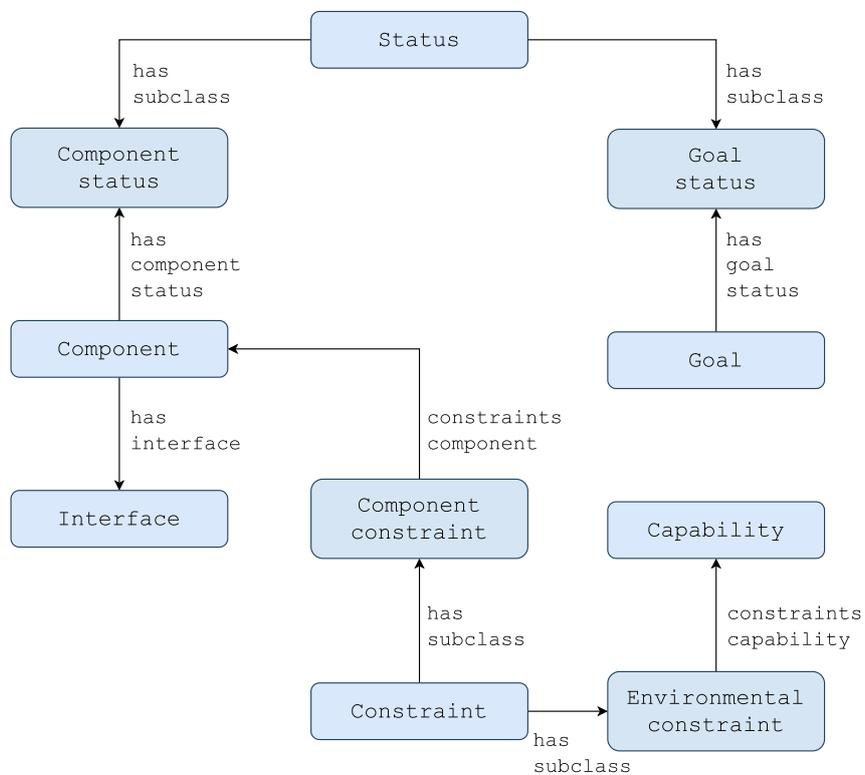


Figure 8.4: Additional classes and relationships of the SysSelf ontology related with constraints, status and interfaces.

► DATA PROPERTIES

Data properties serve to assign values to instances in the model. The most important property of this type is the `has metric value` that serves to assign decimal numbers to MOEs, MOPs and TMPs for quantifying behavioral aspects of values, capabilities, and components, respectively. Furthermore, the `affects in negative` boolean property influences the propagation of metric relationships. Other relevant data properties include `has component status value` and `has goal status value`, which assign statuses to components and goals, respectively; and the `is` type relationship, used to define the type of interface associated with a component.

Lastly, it is important to note that this operationalization of the metamodel does not incorporate any individuals, since it corresponds to the terminological and relational part of the ontology, TBox and RBox. Each application engineer or person responsible for designing applications using the metacontroller is required to create an ABox containing a set of individuals relevant to their specific system.

Example:

Figure 8.5 illustrates the key elements of the ABox ontology, representing the instantiation of the model for our running example: the *laser contingency in a navigation task*. This model features two components—a camera and a LiDAR—that contribute to the goal of navigating a list of waypoints using the nav2 capability. The task aims to provide value by exploring a specific area efficiently while ensuring robot integrity.

The model also assigns expected metrics related to both component and capability accuracy and mission safety, along with relationships that illustrate how these metrics affect each other. Additionally, it includes two stakeholders—*robot operator* and *mine operator*—along with the values in which they are interested in. Dashed arrows depict data value assignments, while bold arrows represent main functors.

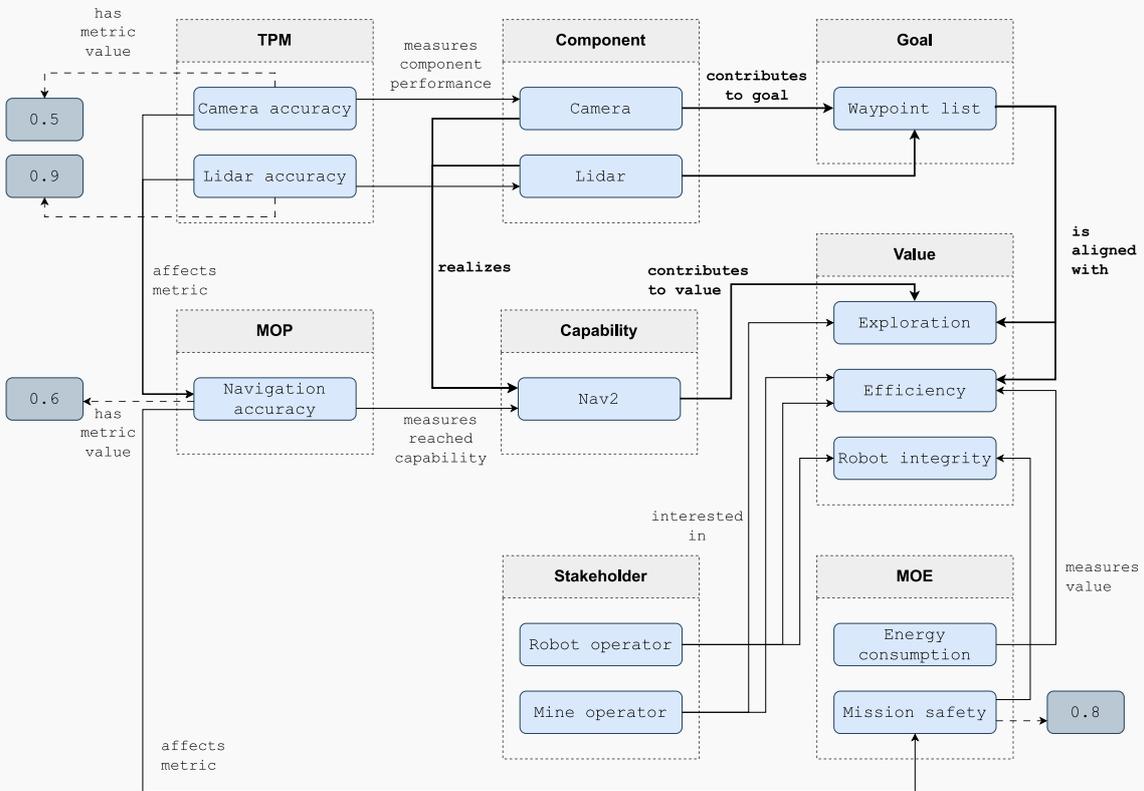


Figure 8.5: ABox example: Application model using the SysSelf metamodel.

8.2 Using Ontological Reasoners

After encoding the metamodel, its deployment in a software application becomes necessary for practical usage. The metacontroller is implemented as a Python class using the Owlready2² API. This API facilitates transparent access to OWL ontologies and offers methods to modify them and execute their reasoners. This section explains the rationale behind the selection of the OWL reasoner and outlines additional methods developed to effectively leverage CT concepts within the metacontroller.

8.2.1 Ontological Reasoning

One of the main changes from TOMASys to the SysSelf metamodel is the elimination of *SWRL* rules. TOMASys used *SWRL* for error propagation and reasoning to determine the best available design alternative, as shown in Table 7.1. However, as discussed in Section 7.2.3, the use of *SWRL* rules contributes to increased reasoning time [206]. To address this issue, the SysSelf solution uses mathematical concepts from CT to enhance expressiveness, eliminating the need for *SWRL* rules for triggering adaptation.

The metacontroller initially employs an ontological reasoner to update the *KB* by classifying individuals and their properties while ensuring consistency. Afterwards, a Python script is employed to perform error propagation and navigate between relationships, uncovering non-explicit connections using CT principles.

► REASONER SELECTION

The ontological reasoning in this project relies on the consistent update of the *KB* using *Pellet*. This reasoner was selected for its compatibility with OWL API libraries. However, this choice is not arbitrary. Three open-source reasoners were considered: FaCT++, HerMiT, and Pellet. All three have a Protégé Plugin, run on all platforms, and support the OWL API. FaCT++ is implemented in C++, while HerMiT and Pellet are Java-based.

Performance comparisons conducted by Dentler et al. [216] on a biomedical ontology test suite, commonly used as a benchmark, revealed varying results. For classifying large ontologies, Pellet outperformed FaCT++ for Gene Ontology (3.41s) and FaCT++ for the National Cancer Institute thesaurus (11.10s). However, FaCT++ outperformed Pellet for the SNOMED CT classification (700.87s). In terms of query times for SNOMED CT, FaCT++ was faster for both named concepts (701.49s) and anonymous concepts (0.06s), while Pellet required more time (2,793.31s and 0.49s, respectively). HerMiT exhibited lower performance in both experiments.

Despite FaCT++ being the faster reasoner in certain contexts, Pellet is chosen for its support, availability, and performance trade-off, particularly excelling in large ontologies such as

²<https://owlready2.readthedocs.io/en/v0.42/>

Gene Ontology. Additionally, Pellet is an updated reasoner, tested with OWL API 2, while FaCT++ has seen less maintenance.

8.2.2 Metacontroller Execution and Application of Category Theory

The metacontroller serves as the primary runtime component for adaptation, with all reasoning directed towards extracting the necessary reconfiguration actions and informing stakeholders about changes in expected values. This process is application- and implementation-agnostic, as it relies on the concepts defined in the SysSelf metamodel. The activity diagram presented in Figure 8.6 outlines the execution flow of the metacontroller. Note that this process is based on interruptions to mitigate the computational cost associated with ontological reasoning.

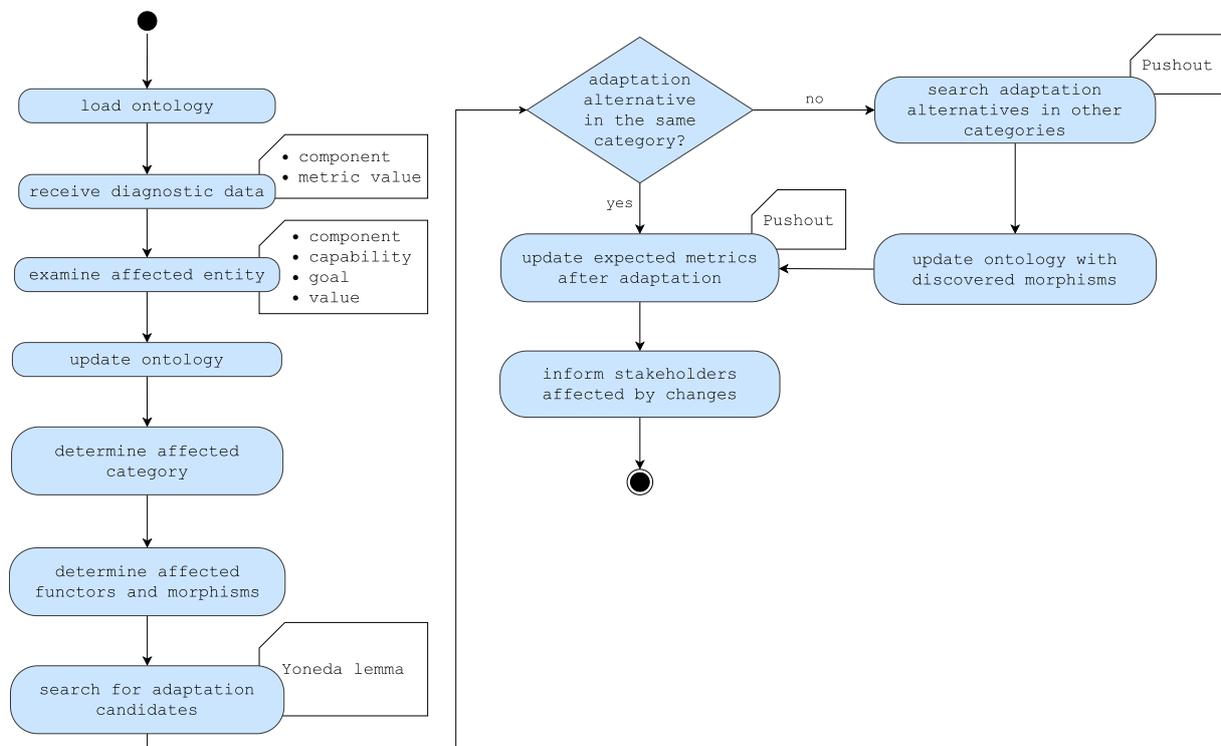


Figure 8.6: Activity diagram for the metacontroller execution.

After loading the ontology, the metacontroller awaits diagnostic data, which could indicate that a component is malfunctioning or a metric value is below the expected threshold. OWL queries are then used to determine the specific entity affected—which can be a capability, value, goal, or component. As a result, the KB is updated accordingly, and the reasoner is executed to ensure consistency.

The subsequent reasoning relies mainly on CT principles to identify an element that can be exchanged. Initially, the metacontroller identifies the main category (corresponding to an ontological base class) along with its associated functors. Additionally, it evaluates the morphisms to uncover a potential equivalent object. The application of the Yoneda lemma

in this context involves establishing the requirements for equivalence. In practical terms, applying the *Yoneda lemma* (Section 7.4.4) means searching for individuals in the same category with identical relationships.

Example:

Figure 8.7 illustrates the individuals related to alternative components in our running example, the *laser contingency in a navigation task*. The camera and LiDAR components are linked to the waypoint list goal and the navigation capability through the functors contributes to goal and realizes, respectively, as indicated by the bold arrows.

Each component is also associated with an interface individual through the has interface relationship, which represents a *component morphism*. This interface defines a specific type with a data property. For example, the camera interface is of type `/rgb_camera/points`, while the LiDAR interface is of type `/scan`. This corresponds to the type of message these components provide.

The application of the Yoneda lemma aims to find an element or relationship that makes both components the same. In this case, the *only difference is the type of interface*, which must be transformed to make both components interchangeable. The interface adaptor component, depicted in purple, along with its in and out interfaces, constitutes the required transformer that connects the camera and the LiDAR. These relationships are represented by dashed bold arrows.

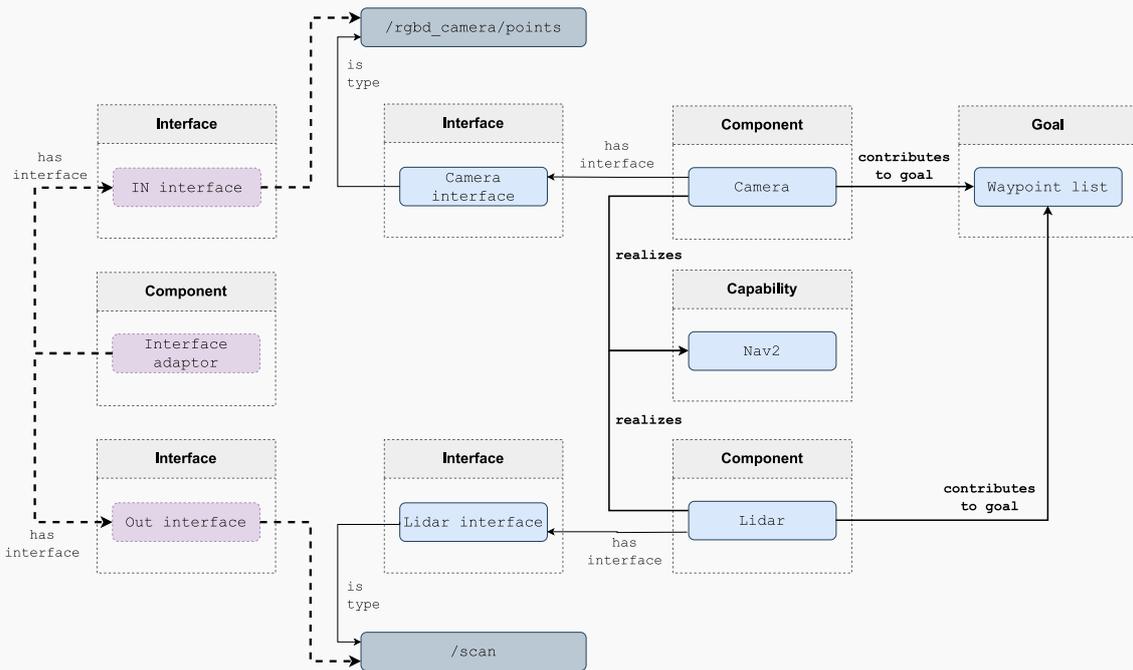


Figure 8.7: Component individuals and morphisms to find equivalences.

Therefore, this structure indicates that the adaptation mechanism, in this case, requires not only stopping the LiDAR execution and launching the camera component but also executing an interface adaptor. The interface adapter ensures that elements using `/scan` messages can continue without further changes.

When a direct equivalence cannot be identified, meaning that there is no equivalent element

in the affected category, the metacontroller retraces the pushout from Diagram 7.3 to explore alternative possibilities in other categories. Throughout this process, implicit morphisms may emerge, as they represent elements that require resources from other categories. These relationships are then integrated into the ontology, which speeds up the process in subsequent uses.

Example:

Consider a simple scenario: a robot composed of a chassis connected to wheels, equipped with the capability to perform navigation tasks. When a gripper arm is added to the robot chassis, a new capability becomes available—manipulation. In this instance, a *new morphism* emerges in the capability category, representing the *composition* between manipulation and navigation. As a result, the robot evolves into a mobile manipulator. Figure 8.8 illustrates how tracking morphisms and functors leads to the emergence of this relationship.

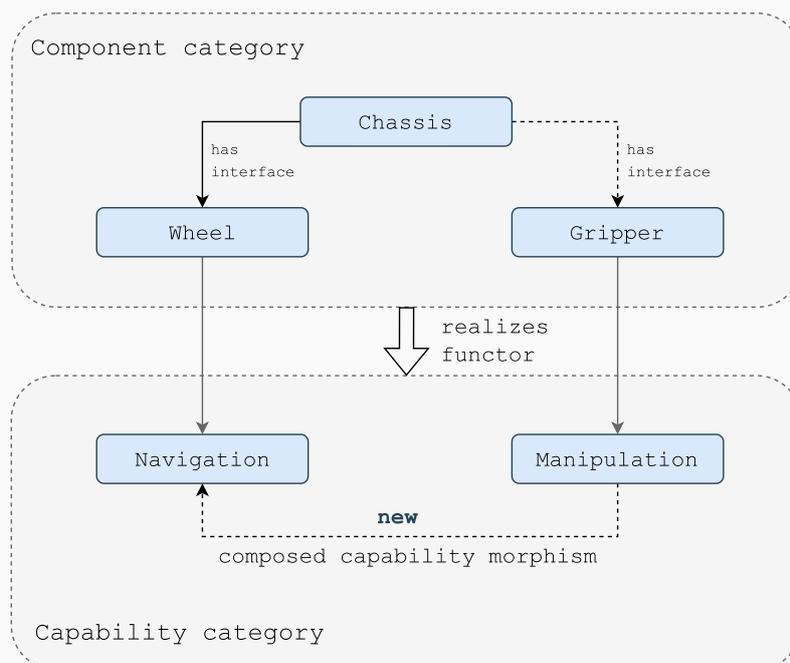


Figure 8.8: Implicit morphism emergence by tracing morphisms and functors between categories.

The final step of this reasoning process involves applying the concept of pushout (Section 6.2.4). Once an adaptation is selected, it propagates the effects of the changes throughout the system, considering relevant metrics such as TPM, MOE, and MOP. By evaluating these metrics, it can determine the expected value resulting from the adapted system. The metacontroller also communicates with stakeholders associated with the values measured by the affected MOEs, informing them of the impending changes.

Example:

Figure 8.9 illustrates some metrics in our ongoing example, the *laser contingency in a navigation task*. Note that the assigned metric values are experimental coefficients provided by application engineers based on their domain expertise. Each redundant component has two TPMs—accuracy and energy consumption—both of which influence the corresponding MOP in the navigation capability. Furthermore, there are three MOEs related to different aspects of the provided value: mission safety and duration, influenced by the accuracy MOP, and mission energy consumption, affected by the corresponding MOP.

When the LiDAR fails, the metacontroller proposes an alternative solution involving the use of the camera, which implies a *reduction* in energy consumption and accuracy, as depicted by the purple values. The propagation of metrics results in a *reduction* in both MOPs, as well as a *decrease* in mission safety and energy consumption MOEs. It is important to highlight that this process is qualitative and that numerical values for MOPs or MOEs are not necessary to determine how the system values are expected to change. The mission duration is expected to be longer due to reduced navigation accuracy, negatively affecting this MOE.

In the final step of this process, the metacontroller informs the robot operator about the increase in safety and energy consumption. Furthermore, both the robot operator and the mission user are notified of the expected increase in mission duration.

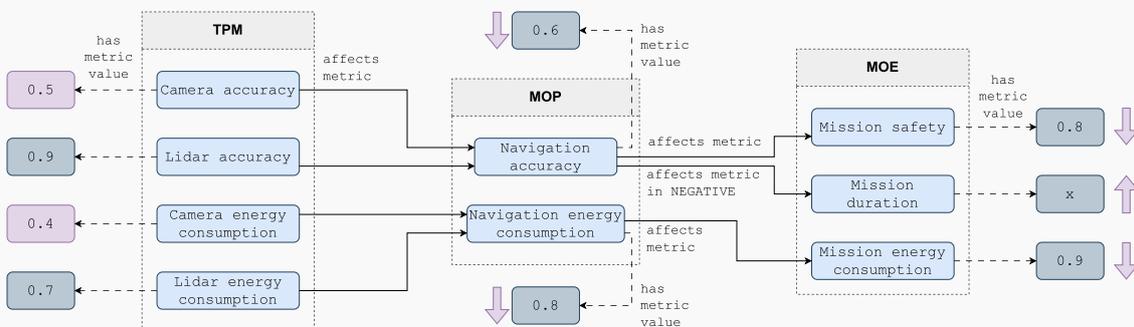


Figure 8.9: Metric propagation to determine how changes in components are expected to affect value MOEs.

8.3 An Implementation Using ROS 2 Nodes

The final step of the metacontroller involves integration with robotic tools. We have focused on the *Robot Operating System (ROS)* due to its widespread popularity, but this structure is easily portable to other frameworks. To facilitate ROS integration, we have developed a metacontroller node wrapper, inheriting from the Python class of the metacontroller

described above. This node is responsible for publishing or executing necessary adaptations based on diagnostic messages. Additionally, there is an observer node that continuously monitors the components, goals, capabilities, values, and performance of the application. It publishes a diagnostic message whenever any of these elements fails or exhibits performance below the expected standards. Figure 8.10 illustrates how the execution of the metacontroller (represented in Figure 8.6) integrates with the two ROS nodes.

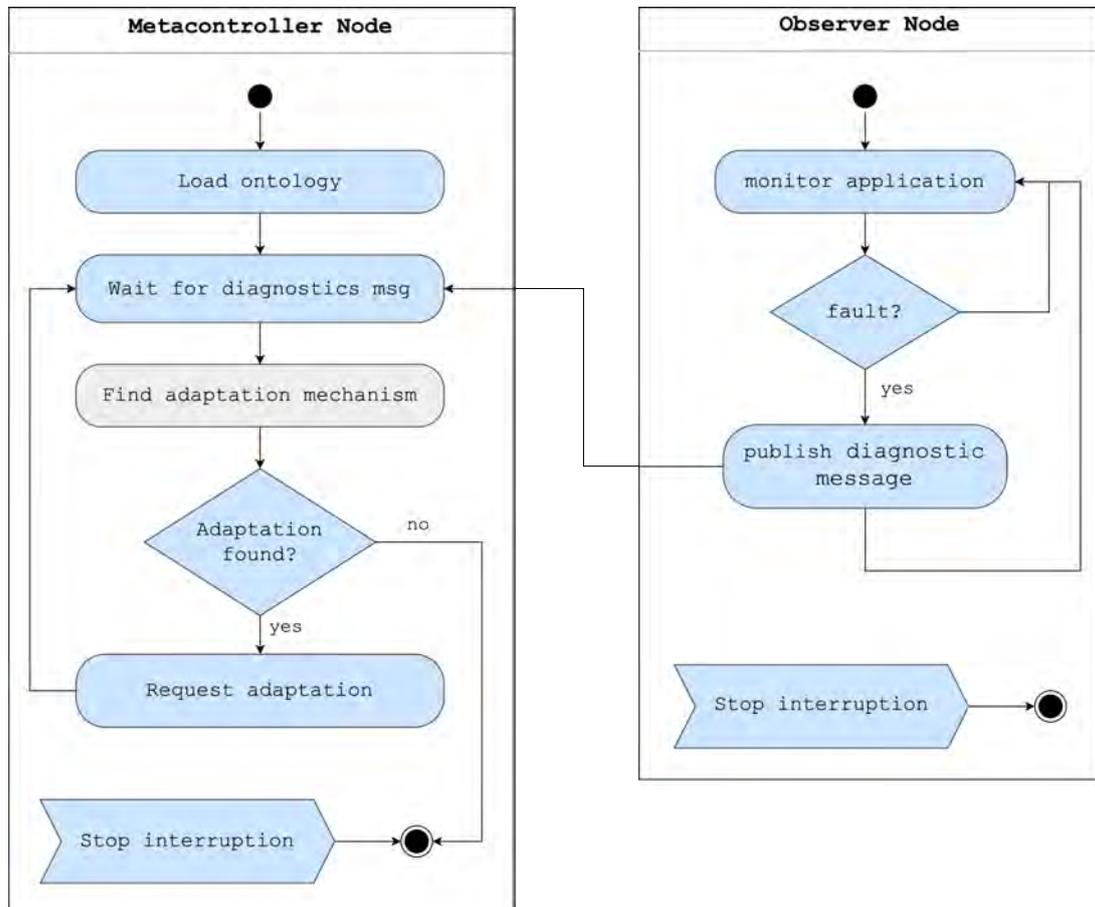


Figure 8.10: Activity diagram of the metacontroller and the observer nodes. Note that ROS 2 nodes are cyclic and they finish with an interruption. The activity “find adaptation mechanism” depicted in gray, corresponds to the main metacontroller execution as explained in Section 8.2.2 and depicted in Figure 8.6.

The execution begins by launching two ROS 2 nodes: the metacontroller and the observer. The metacontroller node first loads the ontology and transitions into a waiting state, until a diagnostic message is received. In our implementation, the observer publishes a diagnostic message following a fault injection process. After this step, the execution detailed in Section 8.2.2 takes place until the metacontroller selects an adaptation mechanism.

For the final step, the metacontroller node requests the requested adaptation. Depending on the nature of the adaptation and any additional requirements, the solution may involve invoking other ROS 2 nodes or publishing to other ROS 2 topics. Figure 8.11 illustrates how the metacontroller infrastructure can be deployed on top of a ROS 2 application.

However, systems using this solution must possess specific features:

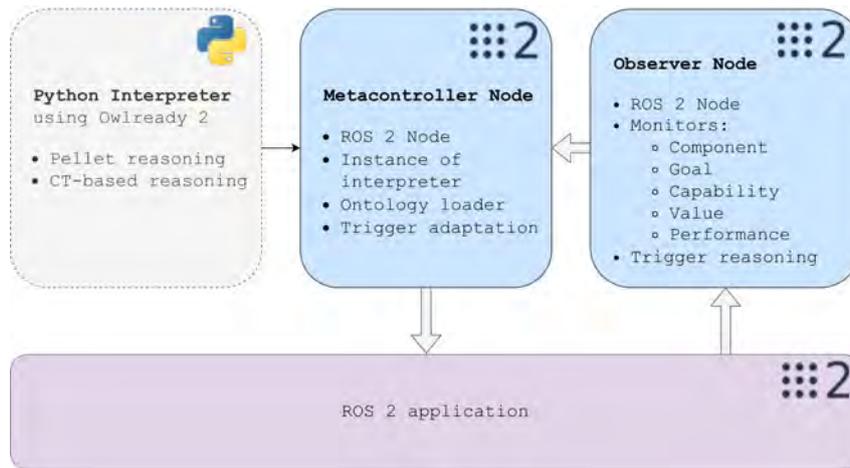


Figure 8.11: ROS 2 wrapper for metacontroller, composed of a Python class and two ROS 2 nodes, an observer and the metacontroller itself.

1. *Redundancy:* Some functional redundancy is necessary to make possible the reconfiguration in the system. This redundancy can manifest itself in diverse aspects such as control laws, algorithms for behavior, structural components, etc.
2. *Monitorization:* The system should allow monitoring, ensuring that processes and elements report realtime information about their operation.
3. *Reconfigurability:* The entire system must be designed to enable reconfiguration. This includes the potential for change, including adjusting parameter values, replacing, eliminating, or launching components.

In conclusion, we have developed reusable assets to provide systems developers with tools that enhance the dependability and autonomy of their applications. Any developer using this approach to make their application self-adaptable should employ the metacontroller and create a specific realization of the metamodel as an application ontology.

Examples of application ontologies used in Chapter 9 can be found in the metacontroller GitHub repository³ in the *ontologies* folder. This ontology must contain information about the system and its components, especially those with functional redundancy, as well as the mission, tasks, and values of the application. All the metacontroller software is also available in that repository.

8.4 Using SysSelf: Step-by-Step Process

This section provides the step-by-step process that needs to be carried out by a project team that wants to use the SysSelf framework to engineer an autonomous system with the self-adaptation capabilities enabled by the framework. Figure 8.12 represents the software assets required to use the SysSelf framework.

³https://github.com/aslab/sys_self_mc

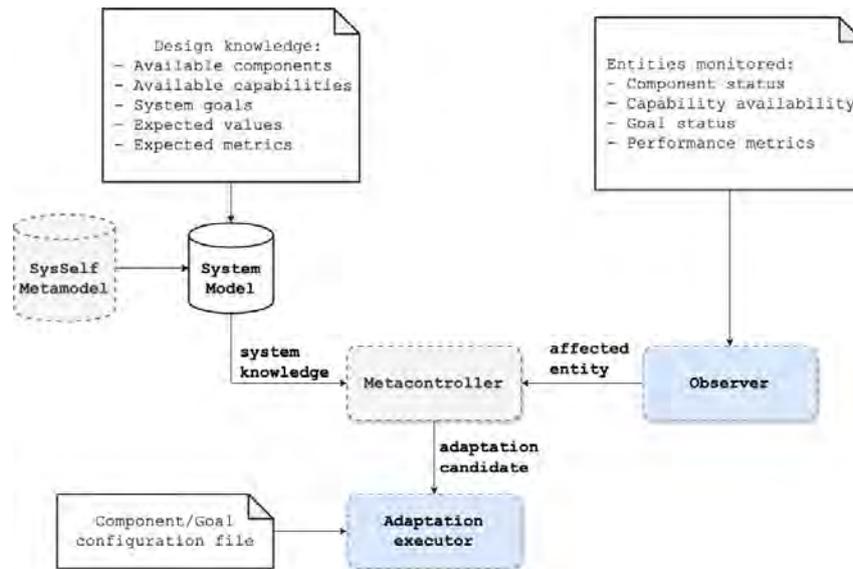


Figure 8.12: Processes required to use the SysSelf framework. The systems engineers shall provide the information necessary to (i) produce a system model, (ii) monitor the system and (iii) execute the pertinent adaptation actions. Dotted lines corresponds to the software provided by the framework. The main constituents—the metamodel and the metacontroller—are highlighted in gray, while those specific to the ROS 2 implementation—the observer and adaptation executor—are highlighted in blue. In scenarios where the ROS 2 infrastructure is not utilized, developers must provide the metacontroller with information regarding the entity affected by a contingency and subsequently execute the reconfiguration action determined by the metacontroller.

First, system designers must generate a system model encoded in an OWL ontology based on the SysSelf metamodel introduced in Section 8.1. This file shall have all the design knowledge of the system. It shall include information about:

- System components and how they are related with optional aspects such as interfaces and constraints.
- The capabilities the system shall exhibit and whether they can be composable.
- The concrete goals of the system and how they are related with optional aspects, such as constraints.
- The expected values provided by the system.
- The relationships between components, capabilities, goals, and values must be explicitly stated defining:
 - Which components contribute to which goal.
 - Which capabilities are realized by which components.
 - Which capabilities contribute to which values.
 - Which goals are aligned with which expected values.
- Metrics in terms of technical performance of the components, capability performance and effectiveness of the expected values. The effectiveness metric can be associated with stakeholders interested in those aspects of the system.

Second, system designers shall decide which elements they want to monitor. They must produce observer software to inform the metacontroller about which system entity has suffered a contingency. This typically corresponds to a component fault, a not available capability, a goal that has not been achieved, or an unacceptable performance drop. When using the ROS2 wrapper introduced in Section 8.3, this means defining or extending the ROS 2 node that sends a diagnostic message with information about (i) the affected entity, (ii) relevant information for the metacontroller if available, especially when there is a performance drop and (iii) the level of severity, i.e., error or warning.

Lastly, the metacontroller requires some methods to execute the reconfiguration action it proposes. In the current ROS 2 implementation, we produce a set of YAML configuration files. These files contain the name of the design solutions that may be employed—usually reused from the ROS workspace—, along with the configuration parameters. These files are used to launch nodes and publish topics, for example, changing the goal of the system or adapting some interfaces when a component is changed.

Chapter 9

Evaluation and Results

In this chapter, we evaluate the metacontroller approach within two distinct testbeds—a miner robot and a commercial mobile robot. The primary objective of this evaluation is to demonstrate the practical application of the theoretical and engineering frameworks introduced in this work, enhancing the autonomy capabilities of the systems through resilient techniques grounded in self-awareness.

Each testbed is subjected to several scenarios, to prove the effectiveness of the proposed solution in diverse contingencies. This evaluation highlights the reusability of the models and the metacontroller across different domains and applications.

This chapter is divided into three main sections. The first two sections introduce the robotic testbeds, providing details of its software and hardware components. In addition, it includes a description of how each scenario is executed and the results of the adaptation. The third section, provides a discussion of the results obtained.

9.1 Miner Robot

The miner robot testbed is part of the ROBOMINERS¹ project, funded by the European Union’s Horizon 2020 Research and Innovation program (grant agreement n°820971). The objective of the project is to enhance EU access to mineral raw materials. Specifically designed to operate in deep, small, and difficult-to-access mineral deposits, the robotic mining system aims to minimize human intervention, especially in scenarios where supervision is severely restricted.

Given the operational environment, the miner robot is required to autonomously navigate through disturbances and overcome challenges, including component failures and evolving mission needs. Resilience, with a focus on reconfiguration at both the hardware and software levels, is a key aspect of our research.

Figure 9.1 illustrates the miner robot during field trials carried out in the open pit, active,

¹<https://robominers.eu/>



Figure 9.1: Full-scale prototype of the miner robot during field tests in the open pit, active, oil shale mine in Kunda, Estonia. Image credits: ROBOMINERS project.

oil shale mine in Kunda, Estonia. These trials validated sensing and structural functions, including mapping lead and zinc ore in various galleries, distances, and conditions, utilizing a full-scale prototype. However, the miner robot we refer to is depicted in Figure 9.2 which corresponds to a scaled-down prototype composed of modular platforms, offering high configurability to form more complex systems for diverse tasks.

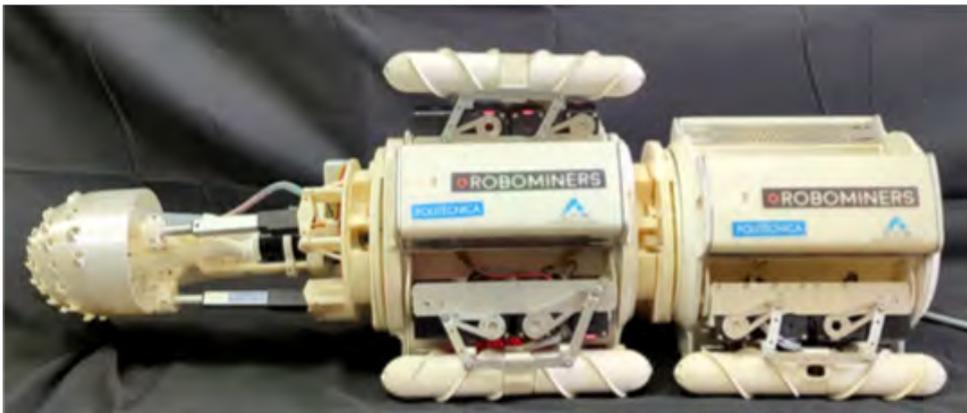


Figure 9.2: Scaled-down prototype of the miner robot composed of highly reconfigurable modular platforms, adapted from [217].

9.1.1 Miner Robot System Architecture

The miner robot scaled-down prototype has been designed to be modular with self-assembly capabilities. This design criterion provides a highly reconfigurable solution [218].

► HARDWARE

The hardware is composed of *modules*, which refers to mobile platforms that can operate

independently. Submodules are considered add-ons that can be attached to the module depending on the requirements of a specific task, such as *end-effectors*, additional *sensors*, or *locomotion devices*. Electronic components such as batteries and other actuation systems are located in the internal compartments of the structure.

Figure 9.3 represents the miner robot structure. Interchangeable end-effectors or additional sensors are assembled in docking ports. Note that the module is equipped with a soft, telescopic, continuum arm integrated in the frontal part of the robot. This arm serves as a mechanical interface to attach other robotic modules or submodules following the principle of a car crane [219].

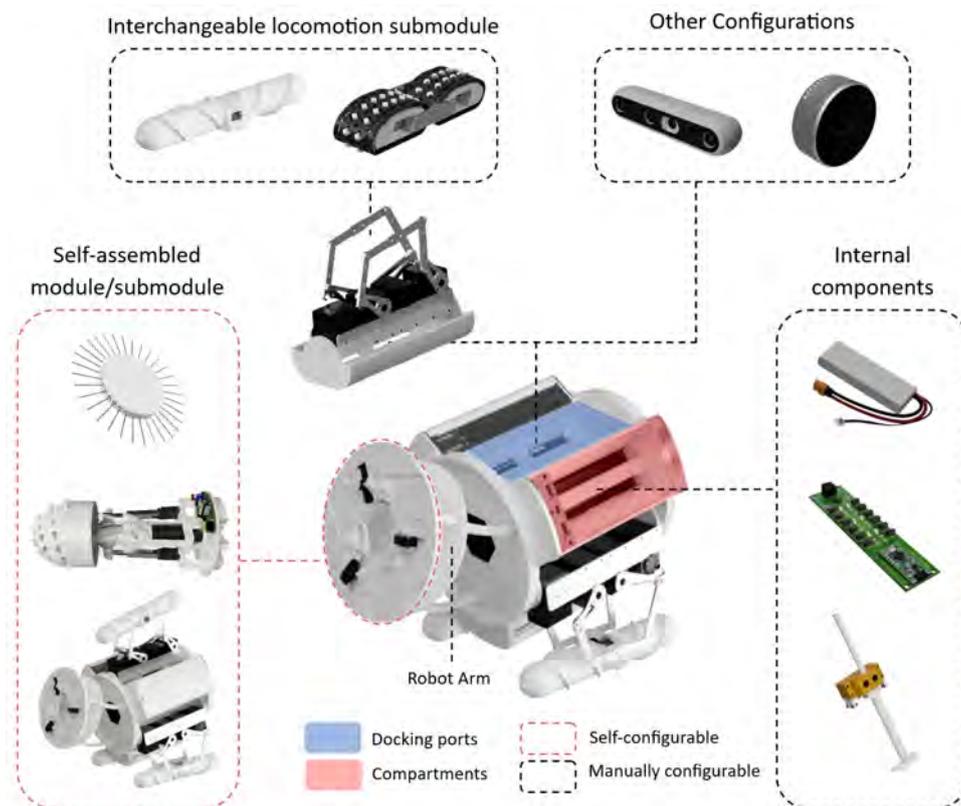


Figure 9.3: Robotic module description with the docking ports to manually attach the locomotion submodules and self-assembly ports to connect additional sensors or modules, adapted from [217].

This approach provides highly adaptable solutions for a variety of tasks, as illustrated in Figure 9.4. For example, the cutting header submodule can be attached for material extraction or detached when the task involves only exploration. In addition, the system can incorporate specialized sensing devices tailored to specific survey requirements. There are two main locomotive modules, which use archimedean screws or continuous track wheels to provide versatility for diverse terrains and challenges. In particular, the important feature of this approach is the autonomous attachment of modules. The soft continuum arm may connect two locomotion modules to create a longer and stronger system or may join a module with a mining head to create a miner robot. For more details on the Robominer design, refer to [217].

Figure 9.5 shows the elements involved in the following scenarios. The connections between components are dynamic: solid lines represent the initial state for each scenario, and

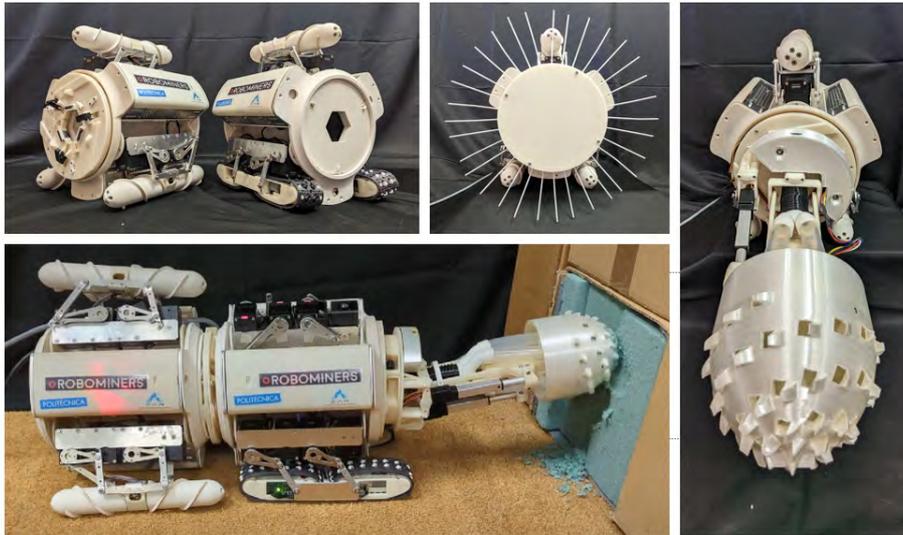


Figure 9.4: Miner robot module configuration examples including different locomotive systems, using archimedean screws or continuous track wheels, and possible configurations combining sensing or cutting headers or other locomotive modules.

dashed lines represent possible new connections. As mentioned above, this prototype allows connections between locomotive modules, headers, or both. The main component in the three cases is an archimedean screw-based locomotive module. The components in blue are those involved in Scenario 1 (Section 9.1.2), while the components in purple are associated with Scenario 2 (Section 9.1.3). Other essential modules of the miner robot concept are depicted in gray for completeness. Since Scenario 3 (Section 9.1.4) handles changes in the mission, there are no specific physical components involved.

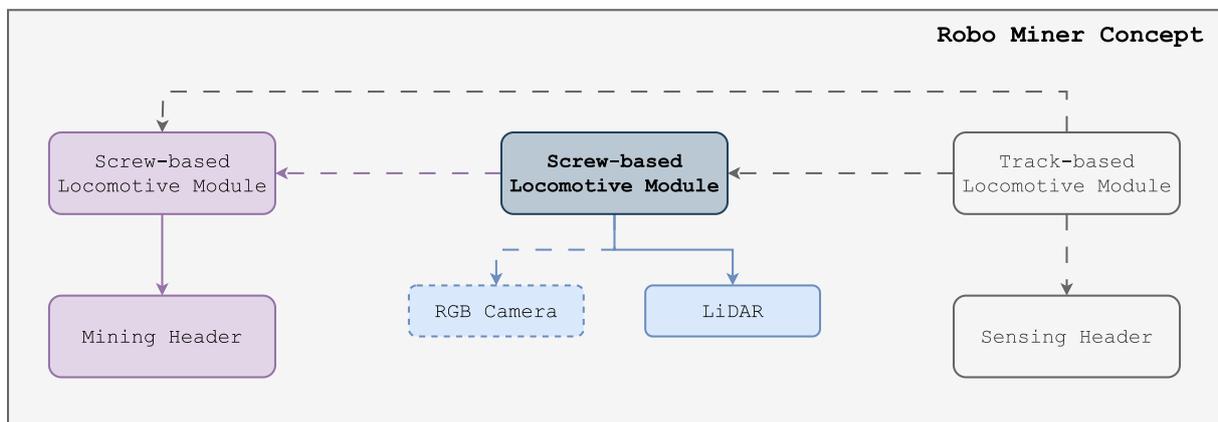


Figure 9.5: Component decomposition of the miner robot concept. Components in blue are involved in Scenario 1, while components in purple are associated with Scenario 2. Additional components of the miner robot concept are shown in gray. Solid lines represent structural connections on the robot from the beginning of the robot’s task, while dashed lines represent dynamic links that may be created during robot operation.

► SOFTWARE

The miner robot software uses existing ROS 2 components to perform its tasks. The main

software used is *Nav2*, the most extended solution for guidance, navigation, and control of mobile and surface robotics. This tool supports different navigation tasks, ranging from traversing designated points to object following and complete coverage navigation, with the added benefit of professional support [212].

The main subsystems that constitute *Nav2* include:

- *Localization*: Global positioning is achieved with the Adaptive Monte Carlo Localization (*AMCL*) [213] based on a particle filter for localization on a static map. Local positioning is based on odometry, which fused with the global pose to update state estimation.

Nav2 also provides the *Simultaneous Localization and Mapping (SLAM)* toolbox to determine position and generate a static map based on *Gmapping* [220] that allows the robot to create a grid map using laser range data and a particle filter.

- *Planner*: This component computes a path to complete some objective, such as determining the shortest path to a goal pose or complete coverage of all free space. The default algorithm used is A* path planning algorithm that takes into account non-holonomic constraints of the robot [214] but other solutions are available.
- *Controller*: The controller provides motion commands to complete a task depending on the local situation. These controllers can include following a path, docking in a charging station, boarding an elevator, interfacing with a tool, etc. In these tests, the default DWB controller is used based on the Dynamic Window Approach².
- *Environmental representation*: The robot perceptive information is encoded in *costmaps*. These correspond to a 2D grid of cells assigning a cost value to each cell depending if it is unknown, free, occupied, or inflated.

Maps often use a layer structure to detect and track obstacles in the scene for collision avoidance. These layers can algorithmically change the underlying costmap based on some rule or heuristic.

- *Point cloud to laser scan*: This ROS 2 component³ is a complementary service from *Nav2* that transforms the 3D point cloud into a 2D laser scan reading. This tool is useful for making devices such as depth cameras behave like a laser scanner for 2D-based algorithms such as SLAM.

The main elements of the software architecture of *Nav2* are represented in Figure 9.6. The framework takes as input a map, a goal pose, and the current pose. The output of this component is velocity commands to drive the robot. The navigation execution relies on a behavior tree (*BT*) [221] to provide flexibility and adaptability to perform the task.

The following sections describe the three scenarios in which we have applied the metacontroller. The first scenario handles a critical component that is experiencing a failure. The second scenario manages a capability that is not meeting its expected performance level. The last scenario focuses on situations where an unsolvable error occurs or the mission becomes

²https://github.com/locusrobotics/robot_navigation/tree/master/dwb_local_planner

³http://wiki.ros.org/pointcloud_to_laserscan

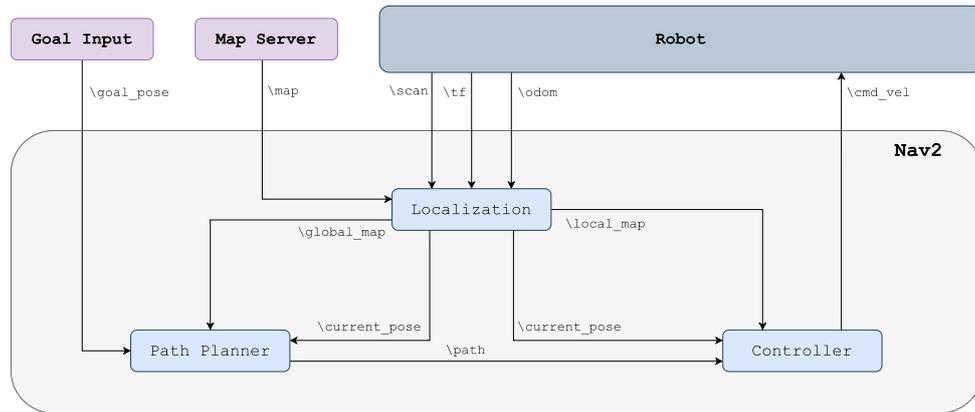


Figure 9.6: Navigation 2 software architecture. ROS topics start with a “\” symbol. The input elements are depicted in purple: goal input and map server. The output is the robot system depicted in dark blue. The main element is the Nav2 subsystem, depicted in gray, which is further decomposed into the light blue boxes: localization, path planner, and controller.

unreachable. In all cases, the metacontroller is responsible for taking the necessary measures to mitigate the impact of these contingencies.

Note that the evaluation was conducted using simulations developed in Gazebo Citadel⁴ and ROS 2 Foxy⁵. The entire source code is available on GitHub^{6,7}.

9.1.2 Scenario 1: Failure in Critical Sensor

In this scenario, there is a robotic module that explores a mine. Figure 9.7a provides a visual representation of the setup and Figure 9.7b represents the navigation software used. During mission execution, the observer detects that the main localization sensor, a LiDAR, is not publishing any topic. This problem can be associated with a malfunctioning sensor or simply a disconnect.

► SCENARIO EXECUTION

The ontological model of this scenario with the elements used is represented in Figure 9.8. The main component is the LiDAR sensor, characterized by its performance metrics and interfaces. Furthermore, the model illustrates the relationship that shows how this component contributes to the goal of traversing a list of waypoints, and realizes the capability of navigation; ensuring the values of exploration, efficiency, and robot integrity.

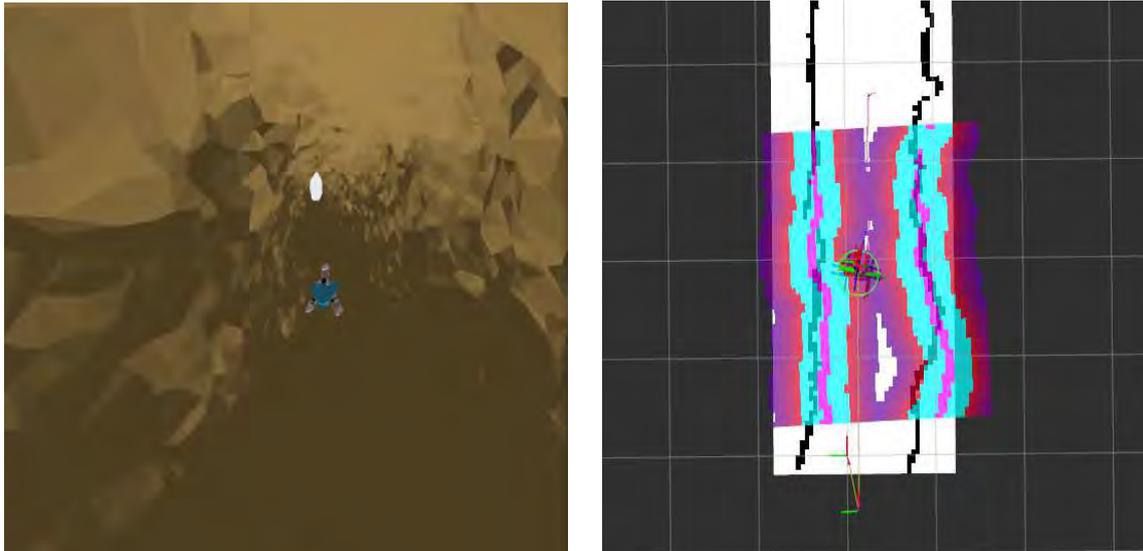
When the fault is injected by substituting the LiDAR messages for zero readings, the observer

⁴<https://gazebosim.org/docs/citadel>

⁵<https://docs.ros.org/en/foxy/>

⁶https://github.com/robominers-eu/rm2_simulation

⁷https://github.com/robominers-eu/rm2_attachable



(a) Simulation of the miner robot main module performing an exploration task in a mine.

(b) Representation of the localization, map and planner output in the navigation software used for exploration. Contours of the mine are delineated with distinct colors within the inflated map layer. These color differentiation affects the controller output, dynamically adjusting the vehicle's speed in response to proximity to walls and obstacles.

Figure 9.7: Setup for the failure in critical sensor scenario for the miner robot.

detects several null readings in the sensor and publishes a diagnostic message⁸. This message issues an error in the LiDAR component with the following structure:

- Level: Possible levels of operations in this case: ERROR.
- Name: A description of the test/component reporting, in this case: lidar.
- Message: A description of the status, in this case: ComponentStatus.

The error level, name, and message is used by the metacontroller to find the affected model entity. When the metacontroller receives that a component is no longer available, it starts looking for another component with similar characteristics. In this case, it was found that the depth information from the camera can provide the same data if it is transformed from point cloud to laser scan. In particular, the metacontroller finds that it can be adapted with an RGB-D camera that requires a point cloud to laser scan interface. This is the case of the Example from Figure 8.7 which is repeated in Figure 9.9 for completeness.

However, the camera is less accurate and the data sent is less frequent. Figure 9.10 compares a map created with a camera and a LiDAR. On the other hand, it consumes less energy. This is translated to a less expected safety and more mission duration MOEs, but also an increased efficiency.

⁸https://docs.ros2.org/foxy/api/diagnostic_msgs/msg/DiagnosticArray.html

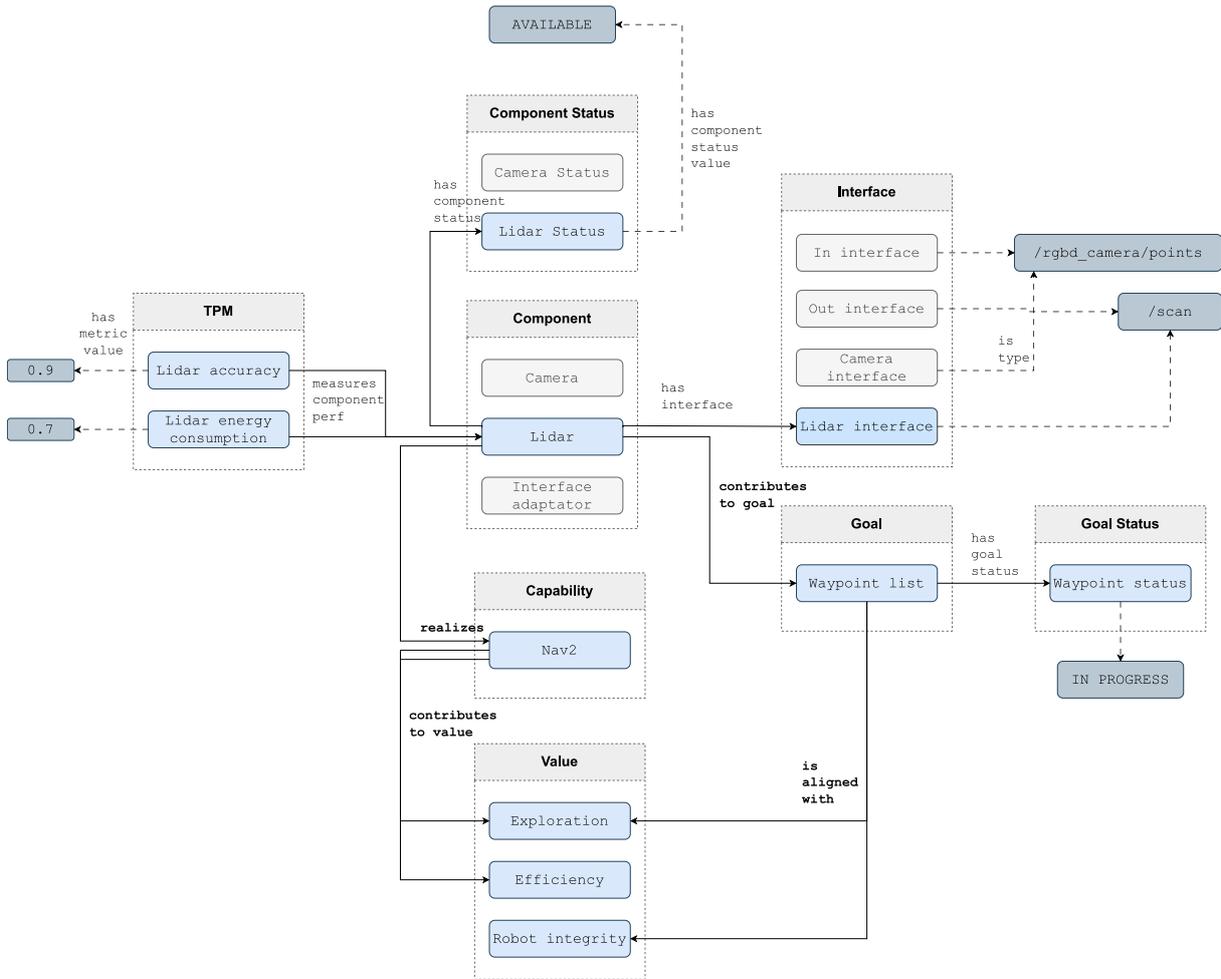


Figure 9.8: Nominal ontological model providing a specification of the LiDAR component, incorporating performance metrics, interfaces, as well as involved capabilities, values, and goals. Other components such as camera and interface adaptor are not used at this moment. Note that not all relationships are visualized for improved readability. Active elements are denoted in blue, whereas inactive ones are presented in gray; in bold, main relationships.

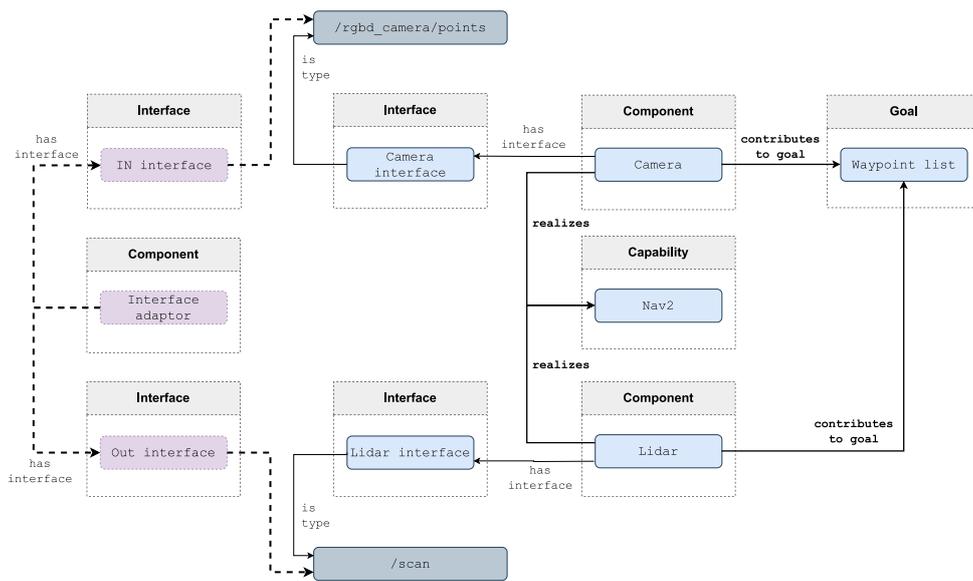
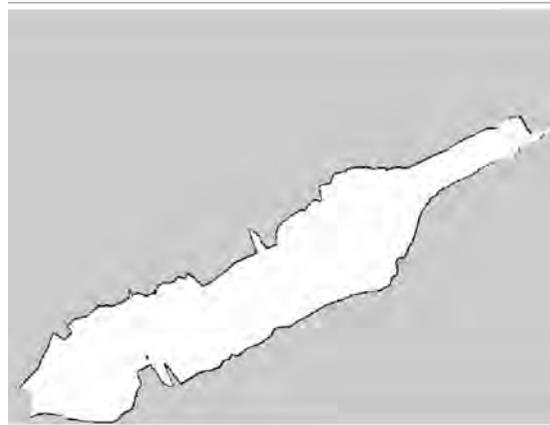


Figure 9.9: Equivalence between LiDAR and camera sensor through point cloud to laser scan interface adaptor.



(a) Mapping results using the depth information from an RGB-D camera which has less accuracy and resolution.



(b) Mapping results using the depth information from a LiDAR, generally more reliable for environmental mapping.

Figure 9.10: Comparison of a map generated with different sensors.

The reconfiguration action calls the point cloud to a laser scan ROS node to start using the camera. Figure 9.11 displays the information on the terminal, which includes details about the metacontroller node and the reconfiguration action.

```
Initialization OK
Component lidar_status updated to value UNAVAILABLE
Component app_loc.camera AVAILABLE
REQUIRES app_loc.pointcloud_to_laserscan to be equivalent
Value value_robot_integrity DECREASED after adaption because
change in MOE mission_safety
Main stakeholder affected: robotic_worker

Value value_efficiency DECREASED after adaption because
change in MOE mission_duration
Main stakeholder affected: mine_worker

Value value_efficiency INCREASED after adaption because
change in MOE mission_energy_consumption
Main stakeholder affected: mine_worker

New Configuration requested: [app_loc.camera,
app_loc.pointcloud_to_laserscan] of type ROS 2 NODE
-----Reconfiguration successful-----
```

Figure 9.11: Information displayed in the terminal of the metacontroller node for the failure in critical sensor scenario, colored text represent the specific application elements on the model from the SysSelf metamodel.

After this process, the robot returns to its exploration case, using the alternative sensor for localization. The resulting model is depicted in Figure 9.12. It shows how the camera and the interface adaptator are used instead of the LiDAR to maintain the system operation. Furthermore, the figure highlights the impact of this alteration on its performance metrics. Lastly, stakeholders interested on each value are represented.

In addition, we provide information on the software execution. Figure 9.13 corresponds to a graph of the initial state of ROS topics and nodes involved in the adaptation, the scan topic coming from the LiDAR is monitored by the observer that publishes a diagnostic topic to the metacontroller when it detects continued null failures.

Figure 9.14 represents the nodes and topics in the system affected after the adaptation, in this case the scan messages that come from the camera after being processed in the point cloud to the laser scan node. Despite the modification, the scan topic continues to be monitored by the observer component.

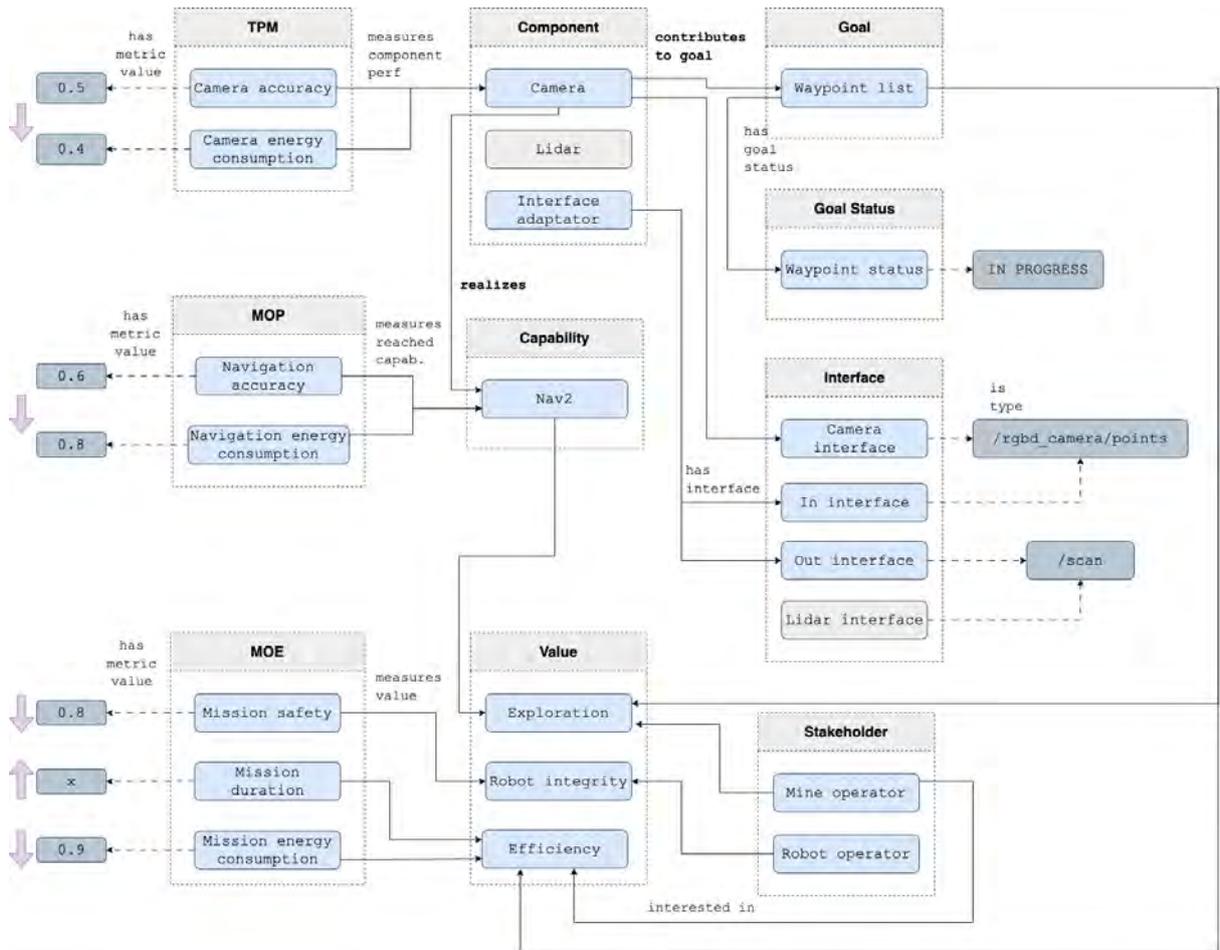
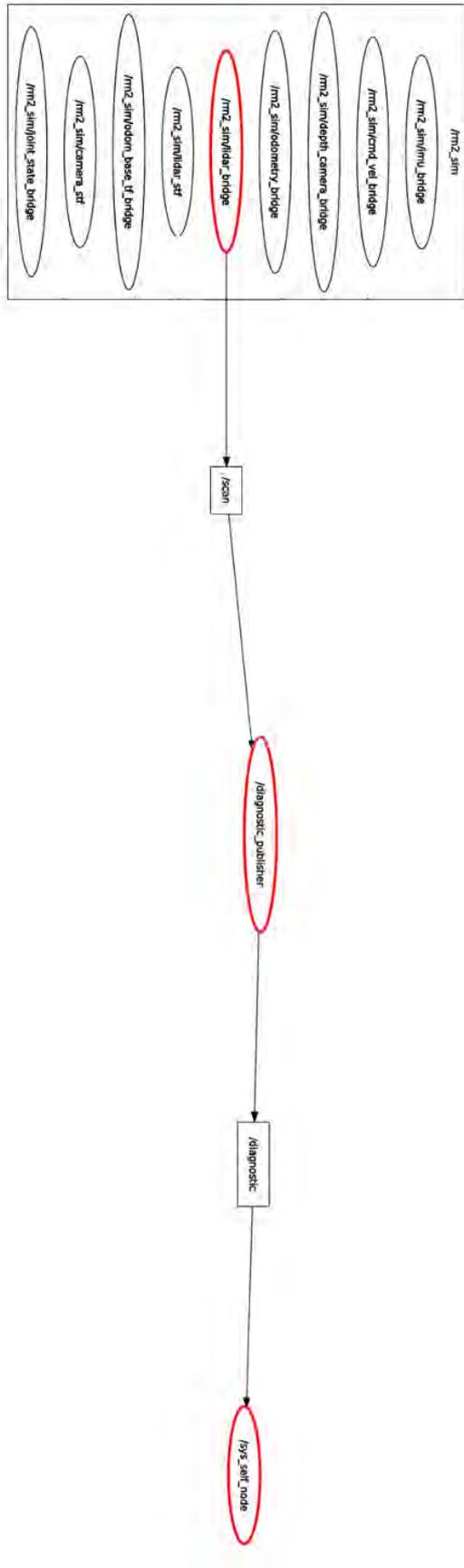


Figure 9.12: Ontological model post-adaptation, showcasing the substitution of the LiDAR with the camera and interface adaptor. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue, while inactive ones are shown in gray; affected metrics are represented by purple arrows; in bold, main relationships.

Figure 9.13: Graphic visualization of relevant nodes and topics for the adaptation running on the system. In red, highlighted the the scan topic coming from the LiDAR and the observer and metaccontroller nodes.



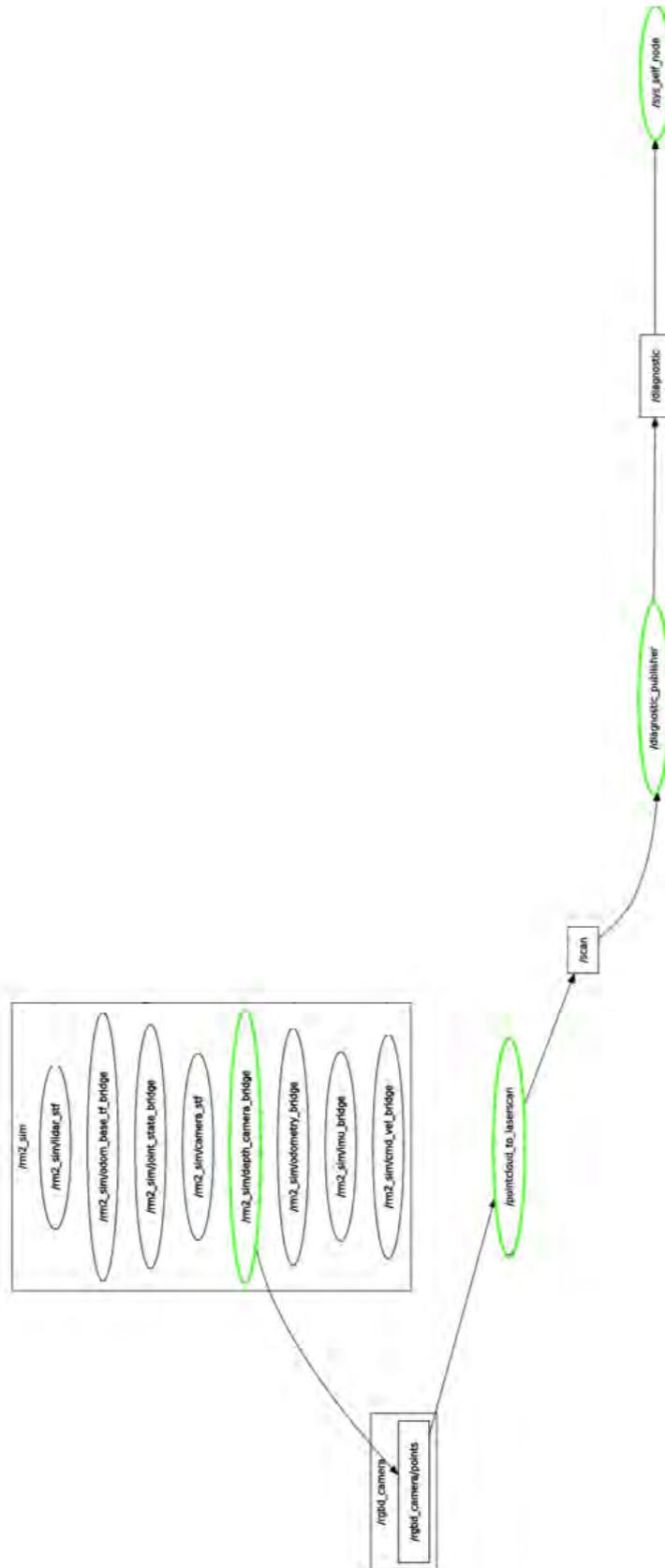


Figure 9.14: Graphic visualization of nodes and topics running on the system involved after adaption. In green, highlighted the nodes and topics affected. Now the scan message comes from the camera after a processing in the point cloud to laser scan. The scan topic is still monitored by the observer.

9.1.3 Scenario 2: Underachieved Capability

In this scenario, there is a robotic module equipped with a cutting head that is used for selective mining. Additionally, a spare robotic module is present. Figure 9.15 provides a visual representation of the setup. During the mining operation, the mission monitor, i.e., the observer, detects that the actual mineral extraction rate is lower than the expected value. This discrepancy is attributed to the robot lacking sufficient forward force to extract the material effectively.



Figure 9.15: Setup for the underachieved capability scenario. On the left, a robotic module equipped with a cutting head that is used for selective mining and, on the right, a spare robotic module.

► SCENARIO EXECUTION

The ontological model of this scenario with the elements used is represented in Figure 9.16. The central component is the robominer mining module, which provides the extract mineral capability, contributing to the respective goal and value. The metrics associated with this scenario include the force produced by the module and the extraction rate and processing time offered by the capability.

The mineral extraction metric rate measures how well the extraction capacity performs. When a fault is injected sending an estimated mineral extraction rate of 0.2, which is significantly lower than the expected value, 0.6. As a result, the observer publishes a diagnostic message, issuing a warning in this metric. This message has the same structure as introduced above with the following values:

- Level: WARN status.
- Name: mineral extraction metric rate.

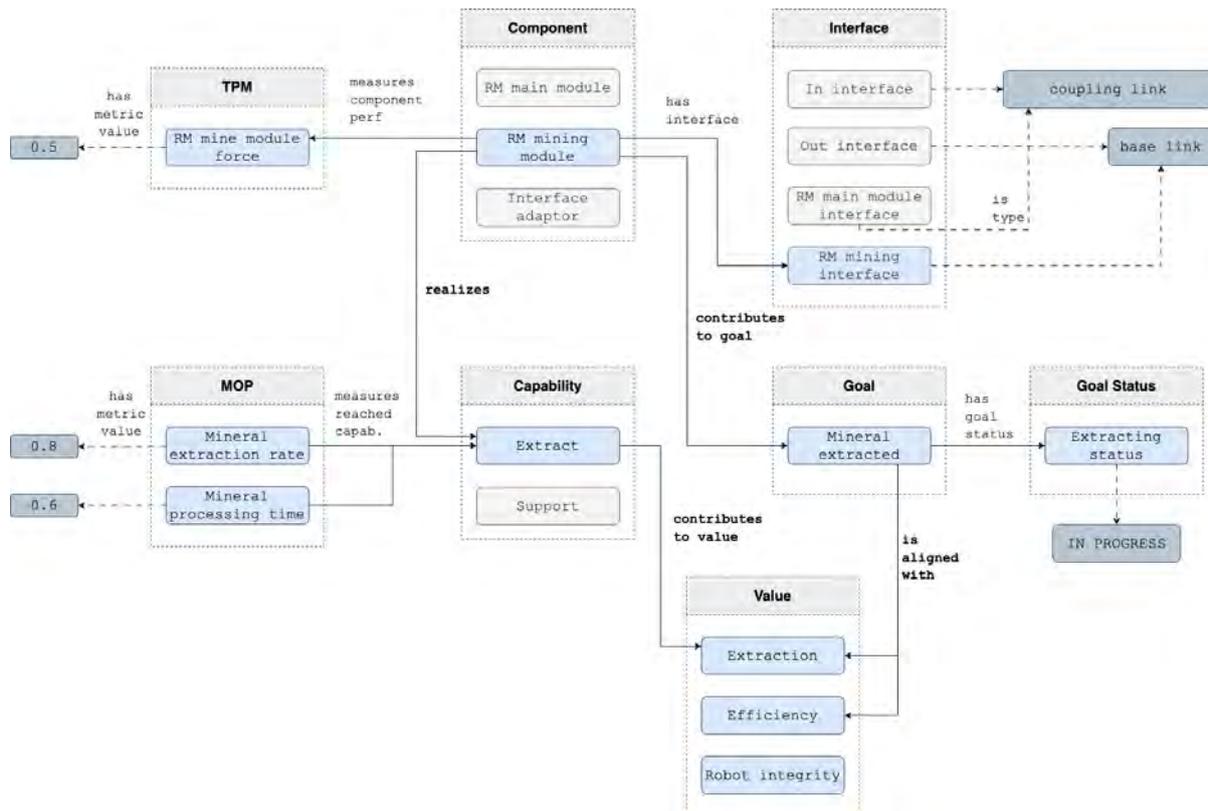


Figure 9.16: Nominal ontological model providing a specification for the robominer mining module and the extract capability it realizes, along with the corresponding goal and value. Note that not all relationships are visualized for improved readability. Active elements are denoted in blue, whereas inactive elements are presented in gray; in bold, main relationships.

- Message: Value in the measured rate, in this case 0.2.

The metacontroller uses the error level and the measured value to find the affected metric. This information is used to determine the base category impacted; in this case the capability.

The metacontroller looks for alternative capabilities but does not find any. However, in the application model, there is a spare robotic module called RM main module that has support capability and can be connected to the mining robotic module. This implicit relationship can be found by tracking morphisms and functors in the SysSelf category, as introduced in the example in Figure 8.8. Figure 9.17 represents how the adaptation action is found by some physical interfaces to connect the spare robot module with the mining one. When both modules are connected, a new *morphism* relationship composable capability emerges between the support and extraction capabilities.

This adaptation leads to an increase in the amount of mineral extracted, improving the mineral productivity metric, but it may reduce the mission efficiency; the application model is updated with this information after the reasoning process. Figure 9.18 represents the model after adaptation by connecting an additional robotic module to increase system support and how metrics are affected. Note that these metric changes are represented qualitatively.

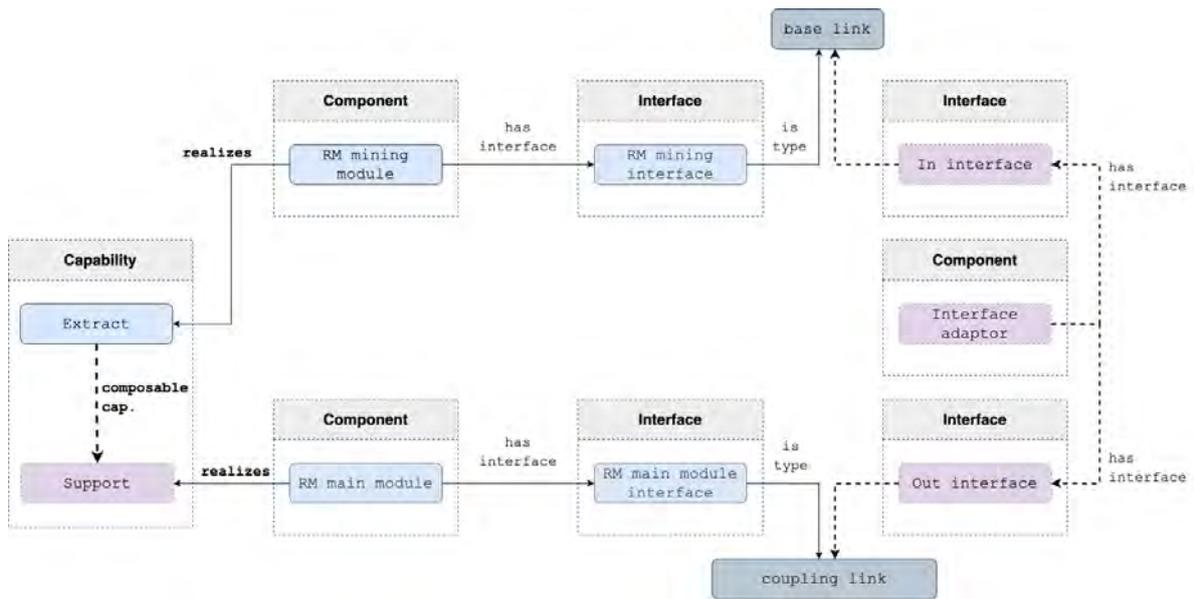


Figure 9.17: Support capability required to solve the lack of force contingency, requires to connect a robominer module through an interface adaptor.

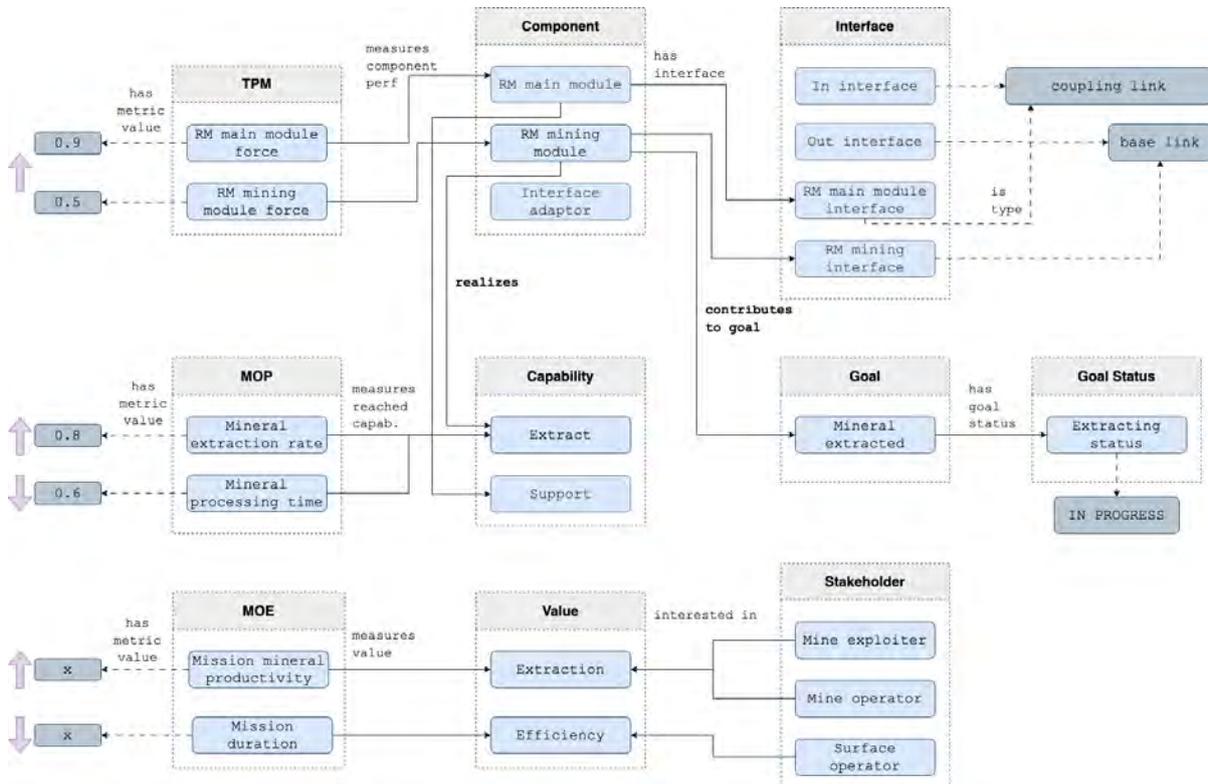


Figure 9.18: Ontological model post-adaptation, showcasing the use of an additional robotic module to increase the system support. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue; affected metrics are represented by purple arrows; in bold, main relationships.

Lastly, the metacontroller commands the adaptation, which in this case involves attaching the spare module to the mining module, connecting their bodies. After the adaptation, the mission resumes. Figure 9.19 represents the information displayed in the terminal, including details about the metacontroller node and the reconfiguration action. Figure 9.20 depicts the resulting outcome of the adaptation, both modules connected to continue the mining task.

```

Initialization OK
New WARN status received, checking metrics
Capability app_attach.capability_extract underachieved,
searching for alternatives
No alternative object found in Capability category
Searching for alternatives in other categories
Component app_attach.rm_main_module AVAILABLE
REQUIRES app_attach.interface_adaptor to be equivalent
Creating new morphism from relation in other category
-----Reasoner executed-----
Value value_extraction INCREASED after adaption because
change in MOE mission_mineral_productivity
Main stakeholder affected: mine_exploiter

Value value_extraction INCREASED after adaption because
change in MOE mission_mineral_productivity
Main stakeholder affected: surface_operator

Value value_efficiency DECREASED after adaption because
change in MOE mission_duration
Main stakeholder affected: mine_operator

New Configuration requested: [app_attach.rm_main_module,
app_attach.interface_adaptor] of type ROS 2 NODE
-----Reconfiguration successful-----
Executing attach action
Action succeeded!
-----RESUMING TO MINING TASK-----

```

Figure 9.19: Information displayed in the terminal of the metacontroller node for the lack of force scenario, colored text represent the specific application elements on the model from the SysSelf metamodel.

9.1.4 Scenario 3: Unachievable Mission

In this scenario, a robotic module equipped with a cutting head designed for selective mining encounters a critical contingency. The robot realizes that accomplishing the mission of extracting a specified amount of mineral from the current deposit is unattainable. The only viable solution involves moving the robot to a new deposit location. Figure 9.21 illustrates the configuration of this setup.



Figure 9.20: Setup after adaptation. The robotic module equipped with a cutting head has connected a spare robotic module to have more force. Now it can continue the mining task.



Figure 9.21: Setup for the mission unachievable scenario. There is a robotic module performing selective mining when the robot realizes that it cannot extract the desired amount of mineral from that deposit.

► SCENARIO EXECUTION

The ontological model in this case uses a mining module to extract mineral. However, there is another capability not used at the moment, the navigation capability, as the main value provided by this setup is the efficient extraction of mineral. Figure 9.22 represents the main elements of the model ontology.

In this case, the ontological model revolves around using a mining module specifically designed for mineral extraction. Although there is an additional capability for navigation, it is not in use at the moment. This decision is intentional, given that the primary focus of this setup is on efficiently extracting mineral. See Figure 9.22 for a visual breakdown of the main elements of the model ontology.

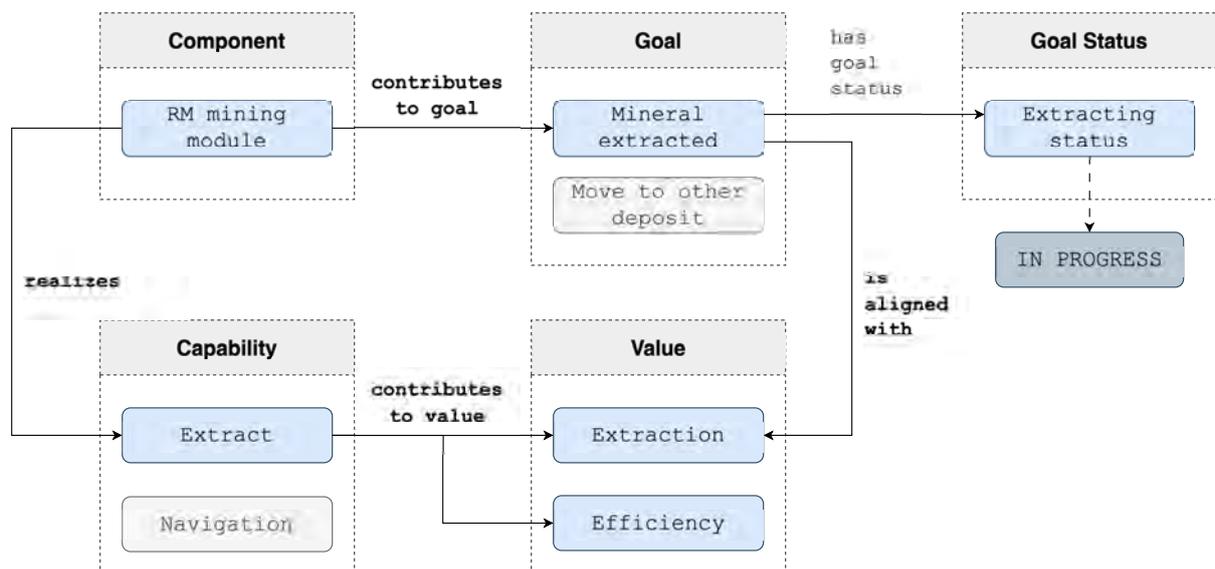


Figure 9.22: Nominal ontological model in the case of the extracting mineral goal scenario. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue; inactive, in gray; in bold, main relationships.

If we introduce an error in the mission extraction status through a ROS topic, the diagnostics node will publish a message with the following structure:

- Level: Error status.
- Name: Mineral extraction goal.
- Message: Not available.

The metacontroller will receive the error status from the diagnostic system. In this case, there is no failure within the system itself, but rather a limitation in the mine deposit, making it impossible to extract the intended material. This state is directly related to the main goal, so the unique possibility is to use an alternative goal of navigating to another deposit location. Figure 9.23 represents the system after adaptation in which the extraction goal and the corresponding capability that enables are no longer active. The new goal is to move to a

new deposit, i.e., a new pose to which the robot shall navigate. These changes are reflected in an increase in the expected MOEs, both in productivity and mission duration. Note that these metric changes are represented qualitatively.

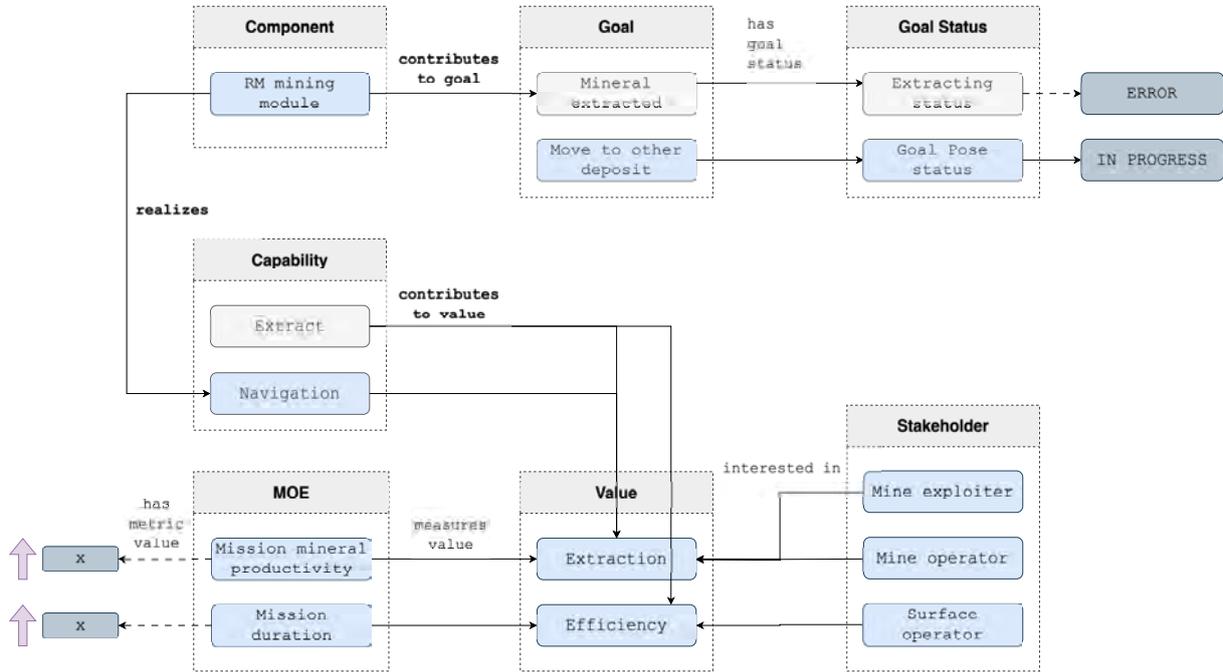


Figure 9.23: Ontological model post-adaptation, it shows the change of goal to continue the operation. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue, inactive in gray; affected metrics are represented by purple arrows; in bold, main relationships.

As a consequence of this adaptation, the robot informs interested stakeholders about changes in their correspondent MOEs. The value productivity is expected to increase as the robot continues to extract the desired material from the new deposit. However, the value efficiency is expected to decrease as the duration of the mission also increases. Figure 9.24 illustrates a terminal output from the observer and metacontroller nodes. Note that in this scenario, the adaptation action does not involve changes to the nodes and topics within the system. Instead, it revolves around updating the destination location for the robot.

```

Initialization OK
New WARN status received
Goal app_mine.extract_quantity_mineral unreachable,
searching for alternatives
Goal app_mine.move_to_deposit AVAILABLE

Value value_extraction INCREASED after adaption because
change in MOE mission_mineral_productivity
Main stakeholder affected: mine_operator

Value value_extraction INCREASED after adaption because
change in MOE mission_mineral_productivity
Main stakeholder affected: mine_exploiter

Value value_efficiency DECREASED after adaption because
change in MOE mission_duration
Main stakeholder affected robotic_operator

New Configuration requested: [app_mine.move_to_deposit] of type ROS 2 TOPIC
-----Reconfiguration successful-----

```

Figure 9.24: Information displayed in the terminal from the observer and metacontroller nodes for the unachievable quantity of mineral extraction, colored text represent the specific application elements on the model from the SysSelf metamodel.

9.2 Mobile Robot

The real experimental trials used a Pioneer 2-AT8 mobile robot, a standardized platform designed by ActivMedia Robotics for research purposes. Figure 9.25 shows the mobile robot used in these experiments.

9.2.1 Mobile Robot System Architecture

This robotic platform has been used in our research group for more than a decade for autonomous navigation tasks in office-like environments. However, to align with contemporary requirements, some modifications have been made. Figure 9.26 depicts the structural decomposition of the robot, showing exclusively the elements relevant for the tests.

The main elements of the system architecture include the following:

- *Differential wheel drive subsystem:* It consists of a four-wheel configuration with an encoder and motor controller per side. Each controller is equipped with an embed-

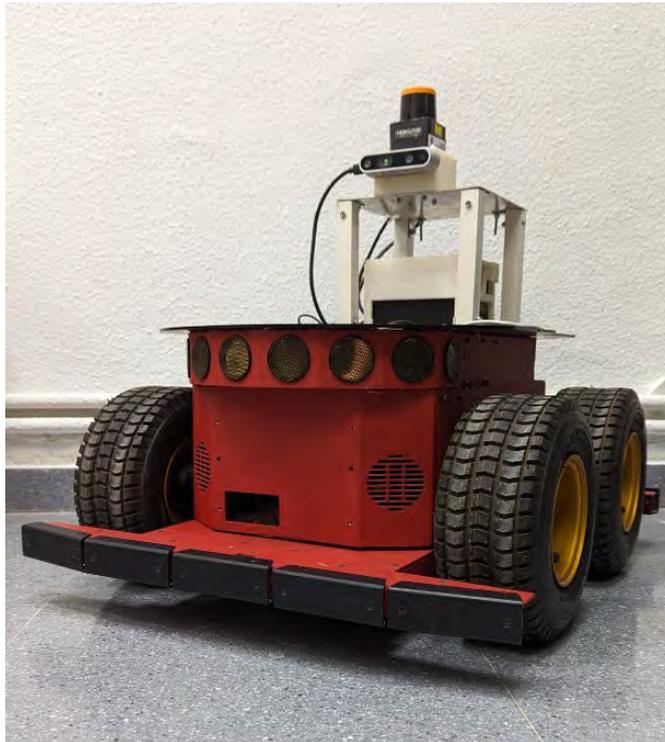


Figure 9.25: Mobile robot testbed Pioneer 2-AT8 with its main sensors, LiDAR and RGB-D camera, mounted.

ded PID control loop, accepting set-points for both linear and angular velocities as commands.

- *LiDAR Hokuyo UST 10LX*: A compact 2D laser sensor offering readings within a 270° angle range and a detection span ranging from 0.06 to 10 m. This device, known for its small size, precision, and high speed, is used for obstacle detection and localization.
- *RealSense D435 camera*: Serving as the redundant localization sensor, this RGB-D camera captures images with resolutions up to 1920×1080 pixels at 30 FPS and a vertical and horizontal range of 85° , as well as stereo depth information. In this case, only the depth information is used when the LiDAR is not available as the information provided is much more imprecise and noisy.
- *NVIDIA Jetson AGX Orin*: The embedded computer in the robot, it uses a 2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores, operating at a maximum GPU frequency of 1.3 GHz. The computer can be configured at 15, 30, or 50 W; for these tests, the minimal potency was sufficient. While its capabilities are tailored for AI applications, which are not utilized in this case, future works aim to explore its potential in conjunction with the approach presented in this dissertation.

For the software architecture, it maintains the core elements outlined in Section 9.1.1, using *Nav2* ROS 2 components and the necessary drivers for the robot platform and sensors available in the research group's GitHub repository⁹.

⁹<https://github.com/aslab/HiggsModel>

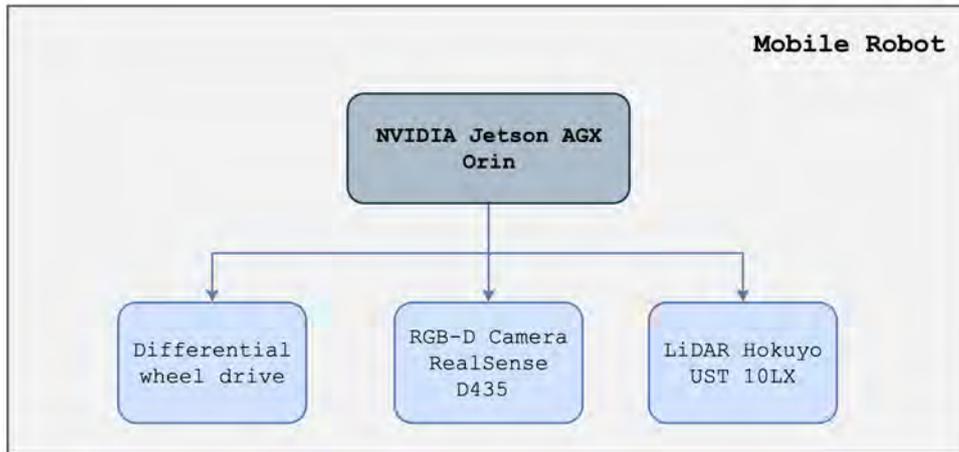
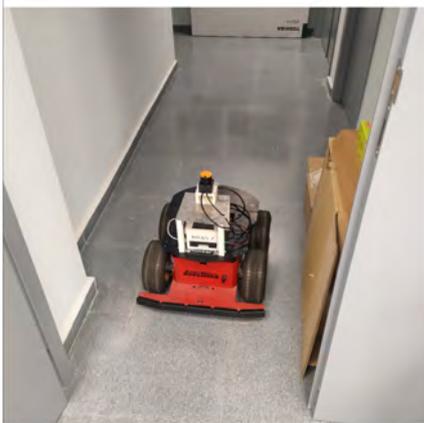


Figure 9.26: Breakdown of the main components involved in the mobile robot tests. The main computer is represented in dark blue, while the remaining components are illustrated in light blue.

9.2.2 Scenario 4: Failure in Critical Sensor

The contingency addressed in this failure is equivalent to the one described in Section 9.1.2 the only change is the robot used and the setup in which the scenario develops. The motivation in this case is to compare reusability and performance in similar settings for simulation and real deployment. Figure 9.27 represents the robot during the navigation task.



(a) Mobile robot in an office-like environment performing an exploration task.



(b) Representation of the localization, map and planner output in the navigation software used for exploration. Contours of the office are delineated with distinct colors within the inflated map layer. These color differentiation affects the controller output, dynamically adjusting the vehicle's speed in response to proximity to walls and obstacles.

Figure 9.27: Setup for the failure in critical sensor scenario for the mobile robot.

Before adaptation, the scan information can be determined from the LiDAR node, denoted as `/urg_node` in Figure 9.28 but after adaptation this information is received from the `/pointcloud_to_laserscan` node transformed from the camera.

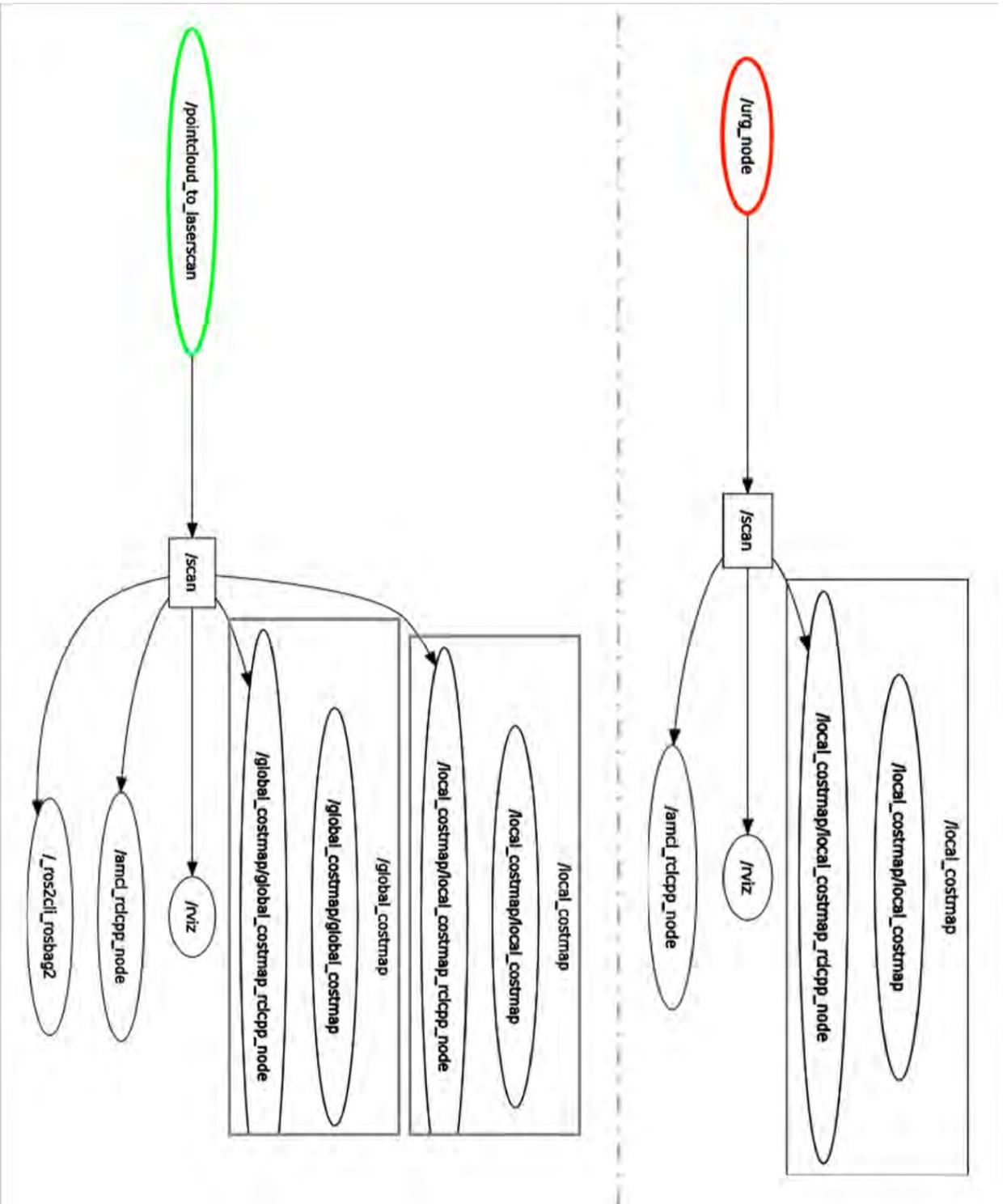


Figure 9.28: Graphic visualization of nodes and topics running on the system involved before and after adaption. In red, the laser node that was substituted, in green the point cloud to laser scan transformer that continues providing scan messages.

9.2.3 Scenario 5: Unsolvable Capability Error

This scenario has the same setup as above, an office-like environment in which the robot is moving around. However, an unsolvable failure in the communication module makes this capability unreachable.

► SCENARIO EXECUTION

The ontological model of this scenario with the elements used is represented in Figure 9.29. The mobile robot has the capability to navigate. The current goal is to traverse a list of waypoints; however, there is the safe waypoint goal that may be used in the event of unsolvable emergencies. Values and stakeholders are also depicted for completeness.

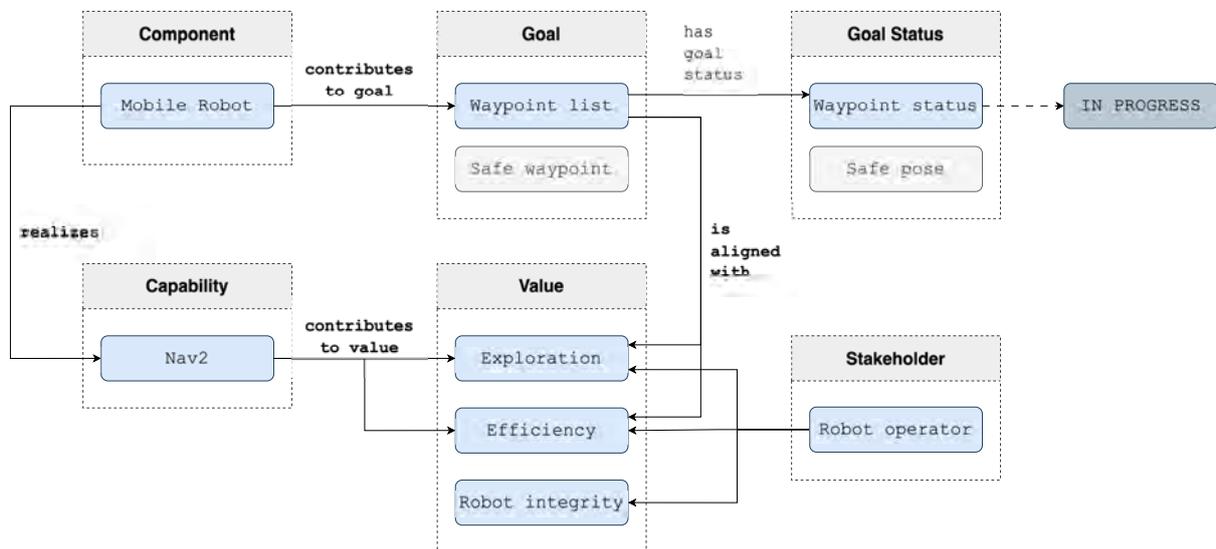


Figure 9.29: Nominal ontological model for the navigation task in the unsolvable capability error scenario. Note that not all relationships are visualized for improved readability. Active elements are denoted in blue, whereas inactive ones are presented in gray; in bold, main relationships.

When the metacontroller receives an error status from the diagnostic system. The primary entity affected by this failure is the robot's capability to communicate with the operator. The message received has the following structure:

- Level: ERROR status.
- Name: Operator communication.
- Message: Not available.

In this case, the failure affects a capability, but there are no viable alternatives within the system to resolve this issue, leaving the only viable solution as a change in the mission goal to go to a safe location and wait for maintenance.

Therefore, to adapt to this situation, the reconfiguration action involves publishing a ROS 2 topic to command the robot to relocate to a safe location. This strategic move aims to facilitate the task of the robot operator, given the inability to communicate directly with the robot. Figure 9.30 shows the terminal output of the observer and metacontroller nodes.

```
Initialization OK
New ERROR status received
Capability app_goal.operator_communication_unreachable, searching for alternatives
Searching for alternatives in other categories
Goal app_goal.safe_wp AVAILABLE
New Configuration requested: [app_goal.safe_wp] of type ROS 2 TOPIC
-----Reconfiguration successful-----
```

Figure 9.30: Information displayed in the terminal from the observer and metacontroller nodes for the communication failure, colored text represent the specific application elements on the model from the SysSelf metamodel.

In cases where the mission changes due to unforeseen circumstances, such as this communication failure, information about how the metrics change may not be readily available or relevant. As a result, such metrics are not included in the metacontroller logger, as the robot is expected to get some maintenance before resuming to its task.

9.3 Analysis

The scenarios outlined above were intentionally designed to address a diverse range of cases, with the aim of evaluating the framework from multiple perspectives. To achieve this objective, two systems were employed—one in simulation and one with a real robot—performing similar and comparable tasks, although with some inherent differences. The following sections examine the scope, performance, and reusability of the SysSelf framework derived from the tests conducted.

9.3.1 Scope

The framework developed in this thesis aims at broad applicability, being independent of specific domains or applications. The solutions offered are not pre-defined or hard-coded; instead, they are dynamically selected at runtime based on the specific requirements of the situation.

The experiments carried out highlight the effectiveness of the model-based metacontrol approach for adaptation. In contrast, other approaches often incorporate ad-hoc solutions tailored to specific robots and their tasks. For example, in the *Navigation 2* framework, predefined recovery actions are triggered using *BTs* based on the failing component:

- *Planner*: When the system cannot generate a plan, it clears the global map used by the path planner.
- *Controller*: When this component fails to determine a suitable velocity, it clears the corresponding area in the environmental model (i.e., local map).
- *Navigation*: When the whole task fails, it first clears the entire environmental model, then spins the robot in place to reorient local obstacles, and ultimately waits for a timeout.

According to Macenski et al. [212], there were two primary scenarios that triggered recovery behaviors in the robot. The first scenario involved crowded spaces where the robot was unable to compute a clear path to its destination. If the *clear map recovery* approach failed and the path was obstructed by people, the robot used *wait recovery*, which stopped navigation until the path was clear. The second scenario occurred when localization confidence was low, particularly in long corridors with repetitive features and many people. In these cases, the *spin recovery* approach—in which the robot spins in place—was used to improve the confidence in position before resuming its task.

Although these ad-hoc recovery approaches did utilize the knowledge and experience of engineers in dealing with contingencies, they lacked the ability to provide a clear rationale for the robot's actions. As a result, the reasoning behind these approaches remained solely in the minds of the engineers. This lack of transparency can raise concerns about the reliability of the system. A particular recovery approach might not be suitable for a given situation or could adversely impact a properly functioning subsystem. For example, the planner recovery approach, which clears the global map, can affect the local map used by a faultless controller.

On the contrary, our solution leverages system models for adaptation. The explicit formalisms used facilitate the traceability of the contingencies and the resulting decisions, providing engineers and the robot itself with updated, consistent, and transparent models throughout the entire operation.

9.3.2 Runtime Performance

To evaluate the performance of the framework, we have measured the downtime when the metacontrol recovers the system for each scenario. Each scenario has been tested under 50 iterations to measure the maintenance time which includes the latency (time to detect the failure), reasoning time, and reconfiguration time. The simulated experiments were performed on a 64-bit Ubuntu 20.04 PC with an Intel i7 13700HX processor and 16 GB of RAM memory and using ROS 2 Foxy as middleware. For the real experiments, a 2048-core NVIDIA Ampere was used using a 64-bit Ubuntu 20.04 NVIDIA Jetson AGX Orin and ROS 2 Foxy. Table 9.1 shows the average maintenance time after 50 iterations and its standard deviation; when possible the time was measured executing only metacontrol and both metacontrol and navigation in the same processor to evaluate the impact of other tasks.

			AMT (s)	Std dev
Simulation	Sc1: Failure in critical sensor	MC	0.977	0.039
		MC and Nav	1.607	0.193
	Sc2: Underachieved Capability	MC	1.041	0.128
		Sc3: Unachievable Mission	MC	0.994
	MC and Nav		1.852	0.048
Real	Sc4: Failure in critical sensor	MC	2.026	0.247
		MC and Nav	5.243	0.774
	Sc5: Unsolvable Capability Error	MC	0.927	0.295
		MC and Nav	3.135	0.848

Table 9.1: Average Maintenance Time (AMT) in the evaluation scenarios after 50 iterations and standard deviation. Scenarios have been evaluated that execute (i) metacontrol (MC) and (ii) metacontrol and navigation (MC and Nav) in the same processor to measure the impact of other tasks.

In the simulation, the average maintenance time is approximately 1 s when solely executing the metacontroller and less than 2 s when running it alongside the navigation framework. All three scenarios exhibited similar AMTs, regardless of the contingency that occurred. However, in the real robot, adapting to the laser contingency (Scenario 4) takes around 2 s when only using metacontrol and approximately 5 s when deploying it in conjunction with navigation. Conversely, the communication failure (Scenario 5) requires less than a second for recovery using metacontrol alone, and about 3 s when executing both subsystems.

The discrepancy between these cases is attributed to the latency of the point cloud laser scan, which takes some time to configure and start sending messages to restart the navigation. Additionally, in both cases, the onboard computer operates in a low-performance mode (15 W) to reduce energy consumption, justifying the differences between using only metacontrol or both metacontrol and navigation. These results underscore the benefits of the SysSelf framework, as they should be compared with the expected maintenance time, estimated at 300 s for a human operator to reach the robot and restart the system.

9.3.3 Reusability

The SysSelf ontology provides general concepts applicable to any system. It constitutes the terminological layer that can be expanded with domain-specific knowledge to reuse and tailor certain concepts in a specific field. For example, sensors or actuators components can be further specified with elements such as range sensors or wheels and gripper actuators, common components in robotics. These ontologies are loaded as libraries, with the final layer being the application-dependent ontology, storing all engineering information about the robot and its mission. This layered approach enables the creation of domain- or application-specific rules and relations to refine the applicability of the framework.

In this research, we have prioritized generality, using the structure outlined in Chapter 7 for the application model for each scenario. Therefore, the reasoning executed by the metacontroller was focused on the SysSelf concepts. For example, the models in the sensor failure contingency in Scenarios 1 and 4 are identical, with only minor changes in names for

clarity purposes, yet they share 29 individuals. The underachieved capability of Scenario 2 is reused in 73% for Scenarios 3 and 5, where the models are less complex, comprising 16 individuals. These findings demonstrate that, while some adjustments are necessary to incorporate application particulars, they are relatively straightforward to implement.

However, implementing this approach in the application code requires certain changes. First, adding a new element—be it a component, goal, capability or metric—requires a new observer responsible for sending diagnostic messages to the metacontroller. This observer monitors the behavior of the system to detect anomalies or malfunctions, notifying the control system accordingly. Additionally, the current version of the metacontroller requires a YAML file containing the name of the application—ROS workspace—that the SysSelf framework is adapting, along with configuration parameters for available design solutions that may be employed, such as parameters for the point cloud to laser scan node or the specific location the robot should move to ensure safety during a fatal contingency.

Through this approach, we have implemented a fault-adaptive subsystem in general terms to address faults at different levels. This method offers a reusable asset for systems with design alternatives to complete their mission in the presence of contingencies without human intervention. Moreover, with these experimental settings we have provided evidence of the benefits of automatic reconfiguration via ontological architectures in reducing recovery time.

Chapter 10

Conclusions and Future Work

This chapter provides a summary of the contributions to knowledge and the main conclusions derived from this research. The second part discusses the limitations of the current approach and explores potential means of overcoming them as proposals for future work.

10.1 Conclusions and Main Contributions

There is a demand for increased dependability in autonomous systems. As proposed by Brachman [1], achieving this goal may require looking beyond existing methods and re-considering systems from a new perspective, addressing their inherent complexities. Our approach suggests stepping back from implementation and ad-hoc solutions, seeking general techniques to address problems that may arise during robot operation. This research aims to equip systems with the tools to respond appropriately to uncertain situations, reflecting the decision-making capabilities of a human operator who supervises the robot.

SE provides an integrative approach to handle systems during their whole life cycle, from design to operation, and even its decommissioning. We use this domain to precisely define terms of the *SysSelf metamodel*. This field not only provides well-established fundamental concepts but also offers a perspective for understanding systems. SE forces us to consider the system's evolution, its architecture, specific designs, alternative approaches, and the diverse interests of all stakeholders involved. Our solution aims to use the information available during the system design phase to the robot operation to ensure it is aware of its capabilities and it can adapt to the situation.

We conceptualize models in alignment with *General Systems Theory (GST)*, considering them as theoretical constructions bridging highly generalized mathematical concepts with specific theories of specialized disciplines. With this principle, we have built the *SysSelf metamodel*. As a formal foundation, we employ CT, a language for mathematical abstractions that introduces a novel perspective in the robotics domain.

The SysSelf metamodel is integrated into the robot execution with the *metacontroller*, a tool that manages real-time knowledge about its current state and attainable capabilities to

adapt the system if a contingency arises. This solution provides flexibility; as there are no pre-established ways to adapt the system, they are determined during the robot operation based on its necessities and availability. This strategy overcomes limitations often associated with typical fault-tolerant and self-adapting approaches, which tend to be ad-hoc and tailored to specific anticipated problems.

The outcome of this research is a collection of reusable software assets—the metamodel and its executor, the metacontroller—that can be integrated with autonomous systems as long as they have some design alternatives or redundancies.

The main contributions of this thesis are summarized below:

1. *Novel Perspective*: This research introduces a novel perspective that effectively bridges the gap between abstract mathematical representations and the domain of technological systems as described in Chapter 6.

This perspective is applied with an in-depth analysis of autonomous system concepts and methods to increase their dependability, including fault tolerance, self-adaptation, and formal methods; as well as a study on other apparently unrelated fields such as SE, KR&R and CT.

2. *Formalization of System-Level Concepts*: The thesis put system-level concepts at the service of robot operation, using reasoners at different levels of abstraction. This conceptualization results in a reusable metamodel that defines the language for expressing engineering knowledge of specific applications.

Our metamodel uses ontological reasoners and CT-methods to overcome the limitations identified in the literature. This solution is built upon the lessons learned from a broad survey conducted on robotic frameworks using ontologies and our evaluation of Hernández et al. [26] metamodel, TOMASys.

3. *Metamodel Reification*: The research uses a metacontroller to exploit system models with an application-independent tool designed to determine optimal solutions in a variety of cases. The metacontroller is technology-agnostic, making it adaptable to systems developed using different frameworks. Nevertheless, a ROS 2 wrapper is provided for easy integration within the robotic domain.

This work provides a complete solution for endowing systems with self-aware capabilities, particularly those featuring functional redundancy or the potential for change. The applicability of the solution is demonstrated across various systems and tasks, as proved in Chapter 9, where the metacontroller managed contingencies at the component, goal or capability levels in two testbeds across five scenarios.

The conducted tests demonstrated the real-time usability, with adaptation downtime ranging from 0.9 to 5.2 s, depending on the ongoing processes and the platform used. Note that the adaptation time values correspond specifically to the autonomous robots and applications evaluated in Chapter 9, and these times may vary in other applications. In contrast, the estimated time required for a human operator to reach the robot and restart the system is approximately 300 s. However, this duration could notably escalate in complex environments such as difficult-to-reach deposits, as those simulated in the initial three scenarios.

10.2 Limitations and Future Work

The question of *enhancing autonomy from a systemical perspective* involves a wide spectrum of topics. Therefore, addressing all aspects in depth was beyond the scope of this research. We focus on three pillars: systems, mathematical abstractions, and declarative knowledge. The main limitations of the proposed framework are as follows:

1. *Limited representation*: While the framework provides representation for general concepts following the CT principles, it is based on conceptual relationships between components, capabilities, goals, and values. However, there are other relevant relationships between systemical aspects across the whole system life cycle that are not explicitly represented in this framework, such as those concerning risks and future states depending on system configurations.

The use of more detailed structures has the potential to enhance reasoning capabilities, extending its scope from the current state of the system to potential challenges and resolutions. Moreover, the representation of the system environment and how it may evolve could yield more substantive insights. However, relying solely on declarative techniques to represent these aspects often proves impracticable. Therefore, integrating the framework with other capabilities, such as learning, holds promise to significantly improve its versatility and efficacy.

2. *Extend evaluation*: Although SysSelf is a versatile framework with potential applicability across a wide range of engineered systems, its evaluation so far has been limited to just two mobile robotic platforms. The solution shall be applied to other complex systems, such as social robots, or multi-agent structures, and diverse technological domains such as manufacturing or chemical plants. Furthermore, additional metrics need to be considered, such as the configurability level, system observability, and a comparative analysis of engineering efforts required to address contingencies with and without SysSelf.
3. *Steep learning curve*: The inherent complexity of the framework may pose challenges in terms of usability for developers. This work provides an introduction on aligning system knowledge with CT principles; the absence of engineering-grade tool support for reasoning in these terms has been mitigated by employing ontologies. However, this reliance on ontologies could potentially discourage developers due to its entry barrier. To overcome this obstacle, providing a tool that facilitates translation between popular developer tools and this framework could be highly beneficial.

To address these limitations, several research paths are open to complement and extend this work. Perhaps the natural continuation of the framework would be to align with Brun et al. [33], addressing its requirements for self-adaptive systems (Section 2.5). In this research, the first two requirements, in which we define the ideal structure and concepts to handle the current state of the system, are covered. However, the solution would be considerably improved by addressing the decision-and-act phase. CT provides reasoning at different levels of abstraction, but *what-if reasoning* or *reasoning based on past events* may considerably improve the capabilities of the framework.

Additionally, we envision significant potential in *alternative forms of abstract reasoning* and their integration with other types of models, such as neural networks or differential equations. This research line is a focal point within the CORESENSE¹ project, where we intend to make progress in this direction. The integration of subsymbolic approaches with the SysSelf framework may contribute to enhance component generalization so that different types of components may be identified automatically or provide a more complex diagnostics tool to detect a wider variety of contingencies.

In terms of *usability* of the provided assets, future work may be directed toward the development of additional tools to simplify the integration process. For example, by creating mechanisms to automatically detect configuration files and generate the required system models. A preliminary approach to the SysSelf antecedent, *TOMASys*, was explored in [203].

Furthermore, a promising approach to facilitate the creation of these models would be the transformation of concepts from MBSE languages such as UML or SysML, allowing translation between engineering models of different aspects. Such models may even have various levels of abstraction, from equations to component hierarchies. The use of CT formalizations to bridge the gap between various granularities of models, levels of abstraction, and viewpoints may help provide a consistent *model-to-model transformation*.

In conclusion, the steps outlined in this research toward formalizing designs and recoveries present a promising approach to enhancing robot autonomy. The metacontroller and its metamodel serve as a neutral and universal framework for formally representing and reasoning about the structure and behavior of complex systems with numerous interconnected components. This formalization has the potential to enhance robot dependability and provide a better understanding of complex robotic systems.

¹<https://coresense.eu/>

Part IV

Appendices

Appendix **A**

Scientific Dissemination

This appendix presents the list of scientific contributions produced during the development of this doctoral work.

A.1 Journals

E. Aguado, Z. Milosevic, C. Hernandez, R. Sanz, M. Garzon, D. Bozhinoski, and C. Rossi, *Functional Self-awareness and Metacontrol for Underwater Robot Autonomy*, *Sensors*, vol. 21, no. 4, p. 1210, 2021. <https://doi.org/10.3390/s21041210>.

V. Gomez, M. Hernando, **E. Aguado**, D. Bajo, and C. Rossi, Design and kinematic modeling of a soft continuum telescopic arm for the self-assembly mechanism of a modular robot, *Soft Robotics*, 2023. <https://doi.org/10.1089/soro.2023.0020>.

V. Gomez, M. Hernando, **E. Aguado**, R. Sanz, and C. Rossi, Robominer: Development of a highly configurable and modular scaled-down prototype of a mining robot, *Machines*, vol. 11, no. 8, p. 809, 2023. <https://doi.org/10.3390/machines11080809>.

R. Sanz, J. Bermejo, M. Rodriguez, and **E. Aguado**, *The Role of Knowledge in Cyber-physical Systems of Systems*, *TASK Quarterly: scientific bulletin of Academic Computer Centre in Gdansk*, vol. 25, 2021. <https://doi.org/10.34808/tq2021/25.3/e>.

R. Sanz and **E. Aguado**, *Understanding and Machine Consciousness*, *Journal of Artificial Intelligence and Consciousness*, vol. 7, no. 02, pp. 231–244, 2020. <https://doi.org/10.1142/S2705078520500137>.

E. Aguado, C. Rossi, P. Gil, *Codifying Wildness: Wild Behaviour for Improving Human-Robot Interaction*. *Int J of Soc Robotics* 15, 825–834 (2023). <https://doi.org/10.1007/s12369-023-00986-2>. [Not related to this research].

A.2 Conferences

E. Aguado, V. Gómez, M. Hernando, C. Rossi, and R. Sanz, *Category Theory for Autonomous Robots: The Marathon 2 Use Case*. In: L. Marques, C. Santos, J. Lima, D. Tardioli, M. Ferre (eds) ROBOT 2023: Sixth Iberian Robotics Conference. ROBOT 2023, Springer Lecture Notes in Networks and Systems series, vol. 976. Springer, Cham. [ACCEPTED]. ArXiv preprint: <https://doi.org/10.48550/arXiv.2303.01152>.

E. Aguado, R. Sanz, and C. Rossi, *Self-awareness for Robust Miner Robot Autonomy*, EGU General Assembly 2022, Vienna, Austria, 23–27 May 2022, EGU22-2716, <https://doi.org/10.5194/egusphere-egu22-2716>.

J. Päßler, **E. Aguado**, G. R. Silva, S. L. T. Tarifa, C. H. Corbato, and E. B. Johnsen, (2022). *A Formal Model of Metacontrol in Maude*. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation. Verification Principles. ISoLA 2022. Lecture Notes in Computer Science, vol. 13701. Springer, Cham. https://doi.org/10.1007/978-3-031-19849-6_32.

D. Bozhinoski, **E. Aguado**, M. G. Oviedo, C. Hernandez, R. Sanz, and A. Wasowski, *A Modeling Tool for Reconfigurable Skills in ROS*, 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE), Madrid, Spain, 2021, pp. 25-28, <https://doi.org/10.1109/RoSE52553.2021.00011>.

E. Aguado, R. Sanz, and C. Rossi, *Ontologies for Run-time Self-adaptation of Mobile Robotic Systems*, in XIX Conference of the Spanish Association for Artificial Intelligence, Málaga, Spain, 2021. ISBN 978-84-09-30514-8 https://caepia20-21.uma.es/inicio_files/caepia20-21-actas.pdf

R. Sanz, J. Bermejo-Alonso, C. Rossi, M. Hernando, K. Irusta, and **E. Aguado**, (2020). *An Apology for the “Self” Concept in Autonomous Robot Ontologies*. In: Silva, M., Luís Lima, J., Reis, L., Sanfeliu, A., Tardioli, D. (eds) Robot 2019: Fourth Iberian Robotics Conference. ROBOT 2019. Advances in Intelligent Systems and Computing, vol 1092. Springer, Cham. https://doi.org/10.1007/978-3-030-35990-4_34

E. Aguado, V. Gómez, M. Hernando, C. Rossi, and R. Sanz, *Category Theory for Autonomous Robots: The Sys-Self Model*, Compositional Robotics: Mathematics and Tools (ICRA 2023 Workshop), London, UK, June, 2023. https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Research_Frazzoli/workshops/ICRA2023CRMT_paper_3.pdf

A.3 Book Chapter

E. Aguado and R. Sanz, *Using Ontologies in Autonomous Robots Engineering*, Robotics Software Design and Engineering, p. 71, 2021. <https://doi.org/10.5772/intechopen.97357>

A.4 Patent

V. Gomez, M. Hernando, C. Rossi, **E. Aguado**, D. Bajo, (2022). *Sistema mecánico de actuación para un brazo robótico flexible*. ES2907801.

Appendix *B*

Implementation for an Underwater Mine Explorer Robot

Here we provide more details on the implementation discussed in Section 7.2.1. In this application, the reconfiguration available in the UX-1 is the selection of a force allocation matrix. Each Function Design in the UX-1 ontological model corresponds to a different force allocation matrix, and it depends on which thrusters are used according to the following dynamical model. The non-linear equation in (B.4) models the motion for an Underwater Unmanned Vehicle (UUV) using the motion representation vectors in (B.1)–(B.3).

$$\boldsymbol{\nu} = [u, v, w, p, q, r]^T \quad (\text{B.1})$$

$$\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^T \quad (\text{B.2})$$

$$\boldsymbol{\tau} = [X, Y, Z, K, M, N]^T \quad (\text{B.3})$$

$$\mathbf{M} \dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) = \mathbf{B}\boldsymbol{\tau} \quad (\text{B.4})$$

where $\boldsymbol{\nu}$ is the linear and angular velocity vector, $\boldsymbol{\eta}$ is the position and orientation vector, and $\boldsymbol{\tau}$ is used to describe the forces and moments acting on the vehicle. The motion model matrix are \mathbf{M} , which is the system inertia matrix, $\mathbf{C}(\boldsymbol{\nu})$, the Coriolis and Centripetal term matrix, $\mathbf{D}(\boldsymbol{\nu})$, the total hydrodynamic damping matrix. $\mathbf{g}(\boldsymbol{\nu})$ is the vector of hydrostatic forces and moments for the gravitational and buoyant forces acting on the vehicle and \mathbf{B} is used as a mapping matrix for thruster configuration. Further detail of this model can be found in [222].

The action taken by the reasoner is directed at the thruster configuration matrix \mathbf{B} . This matrix is used to define how the thruster configuration affects the dynamics of the UX-1 robot. The UUV is actuated with eight symmetrically allocated thrusters on each side of the vehicle. \mathbf{B} is a 6 x 8 matrix, the rows are the six DOF, $\{X, Y, Z, K, M, N\}$, and the columns correspond to each thruster, $\{T_0, \dots, T_7\}$.

Based on the dynamics of the system and the effect of the thrusters, the matrix B is defined in (B.5).

$$B = \begin{bmatrix} 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & -l & 0 & l & 0 & l & 0 & -l \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ l & 0 & -l & 0 & -l & 0 & l & 0 \end{bmatrix} \quad (\text{B.5})$$

with

$$l = \sin(\delta) \sqrt{\sigma_1^2 + \sigma_2^2} \quad (\text{B.6})$$

where σ_1 is the distance from the axis of the thrusters to the geometrical center of the UX-1, σ_2 is the distance from each thruster to the middle lateral point, and $\delta = \arctan \frac{\sigma_2}{\sigma_1}$ is the rotation angle of the moments generated on the UUV.

After experimental tests, this matrix is adapted to force limitations and particularities in the final thruster disposition. The real B used when all thrusters are well-functioning is presented in Equation (B.7); this is the configuration used with function designs *fd_surge_all* and *fd_heave_all*. Note that this matrix is independent of the direction of movement. When surging, the system will use the information in the first row (X); whereas when heaving, it will use the third row (Z).

$$B_{real} = \begin{bmatrix} 0.5 & 0 & -0.5 & 0 & 0.5 & 0 & -0.5 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 & -0.25 & -0.25 & -0.25 & -0.25 \\ 0 & 0.5 & 0 & -0.5 & 0 & 0.5 & 0 & -0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 0 & -0.25 & 0 & -0.25 & 0 & 0.25 & 0 \end{bmatrix} \quad (\text{B.7})$$

When a thruster is disabled, for example T_0 , its corresponding column is set to zero. The force allocation depends on the movement direction, when one thruster is not functioning, the sum of forces made by one side needs to be equal to the other side to preserve symmetry. If T_0 is disabled, T_2 could double its force to compensate. However, for security reasons, it is preferable to preserve the nominal workload in T_2 and divide it by half in the other side, T_4 and T_6 , see Figure B.1.

The adaptation of the B optimal matrix in (B.7) to the case of the thruster T_0 disabled while surging, results in the matrix (B.8).

$$B_{noT0} = \begin{bmatrix} 0 & 0 & -0.5 & 0 & 0.25 & 0 & -0.25 & 0 \\ 0 & 0.25 & 0.25 & 0.25 & -0.125 & -0.25 & -0.125 & -0.25 \\ 0 & 0.5 & 0 & -0.5 & 0 & 0.5 & 0 & -0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.25 & 0 & -0.125 & 0 & 0.125 & 0 \end{bmatrix} \quad (\text{B.8})$$

The same force-conservative approach is taken when any thruster is disabled, adapting the force reallocation to their motion contribution, surge in (T_0, T_2, T_4, T_6) and heave in (T_1, T_3, T_5, T_7) .

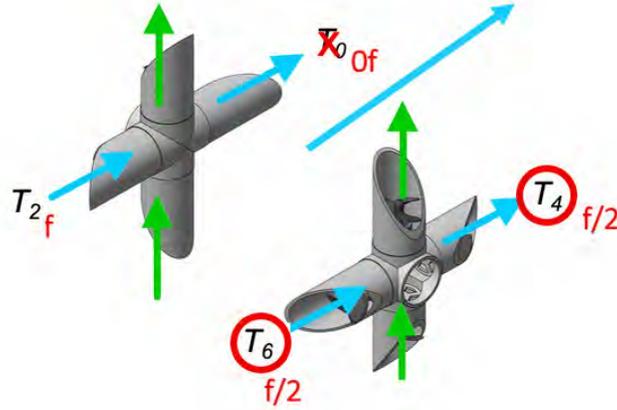


Figure B.1: Force allocation in thrusters T_0, T_2, T_4, T_6 when surging.

When the reasoner creates a Function Grounding entity from the selection of the most suitable Function Design, the reconfiguration is triggered. The reconfiguration consists of the adaptation of the B matrix according to the availability of thrusters. This matrix is stored in a comma-separated values (CSV) file. The name of the file corresponds to the Function Design reified by the Function Grounding. Since this implementation uses *ROS*, the reconfiguration is triggered by publishing the resulting matrix to a *ROS* topic for the low-level controllers, which will use this matrix to adapt the motion to the runtime situation, according to the motion model in (B.4).

B.1 Experimental Results

Experiments follow a software-in-the-loop approach, through the combination of the Gazebo [223] simulator and a realistic model of the UX-1 robot. The position of the robot was acquired from Gazebo's ground truth measurements, further disturbed with random Gaussian noise accumulated over time, to better mimic the positioning system of the real submersible based on noisy instant relative measurements and dead-reckoning. The controller used for the experiments is the Feedback Linearization (FL) controller developed for the UX-1 platform and presented and validated in detail in [222]. The experiments were carried out on a 64-bit Ubuntu 16.04 PC with an Intel i7-6700 2.6 GHZ processor and 16 Gb of memory and using *ROS* Kinetic as middleware.

Three different experiments were performed, all of them reproducing the setting and steps explained above, but with different thrusters enabled/disabled. The first experiment, denoted as **I**, was performed with all the thrusters working. The second experiment, denoted as **II**, was performed with a simulated failure of one of the thrusters in charge of the *heave*

movement (T_0, T_2, T_4, T_6). Finally, the third experiment, denoted as **III**, was performed with a simulated failure of one of the thrusters in charge of the *surge* movement (T_1, T_3, T_5 , or T_7).

Experiments were carried out in an analogous setting, and the commanded path consisted of waypoints containing the desired location in space and the orientation of the robot, and were entered in the following format $[x, y, z, \phi, \psi]$, where the location variables x , y , and z follow the usual *North (x)-East (y)-Down (z)* convention for marine navigation, and ϕ and ψ represent, respectively, the pitch and yaw of the robot. The complete reference path was composed of the following 6 waypoints (x, y, z in meters, ϕ, ψ in degrees).

$$\begin{aligned} A : (0, 0, 2, 0, 0), B : (2, 0, 2, 0, 0), C : (3.5, 0, 2, 0, 0), \\ D : (1.5, 0, 2, 0, 0), E : (0, 0, 2, 0, 0), F : (0, 0, 0.5, 0, 0) \end{aligned} \quad (B.9)$$

In the experiment **II**, corresponding to the fault situation, the said failure of the thruster T_0 was triggered during the *surge* movement while going away from the deployment location when the submersible reached the position $x = 1.7$ m. In the experiment **III**, thruster failure T_1 was triggered during the *heave* movement while leaving the deployment location when the submersible reached the position $z = 0.8$ m. The time elapsed between the failure trigger and the system reconfiguration, referred to as latency, in the experiment **II** was 1.81 s, and in the experiment **III** was 1.09 s. The measurement of these elapsed times was performed by comparing the timestamps of ROS messages, and therefore its precision depends on the timestamps' quality. Since all the nodes were run on a single PC, we assume that there is no considerable delay caused by ROS middleware. The latency is depicted in Table B.1c.

The odometry measurement of each experiment is shown in Figure B.2, as well as the reference waypoints depicted in alphabetical order corresponding to (B.9). The root-mean-square deviation (RMSD) of submersible's position with respect to the ideal path, described by linking the commanded waypoints, is depicted in Table B.1a. It can be noted that the RMSD for all experiments has similar values. This result is in line with expectations and confirms the assumption of redundancy of the UX-1 motion system, showing that the submersible can successfully perform the desired maneuvers (*surge* and *heave*, in this particular proof-of-concept) despite the failure of one of the thrusters.

Figure B.3 depicts the command for the force reference over time in all three experiments: that is, the force in each direction demanded to the thrusters by the controller, and Table B.2a depicts mean values of each reference force. Figure B.4 depicts the forces produced by each thruster in charge for *surge* movement, and Figure B.5 shows the forces produced by each thruster in charge for *heave* movement. Table B.2b summarized the mean values of the force produced per thruster.

It can be seen that the mean commanded force has almost identical values in all three tests: however, the forces actually produced by the thrusters are lower in the experiments **II** and **III**, compared to the experiment **I** in which all thrusters were working properly and with full power. The lower accomplished forces by the thrusters cause, as expected, the longer duration of the experiment (see Table B.1b). The considerable difference in duration of the

experiment **II** seems to be the result of the relative lengths of the vertical and horizontal sections of the experimental setting: the length of the reference path is longer in *North* (x) direction than in *Down* (z) (7 m versus 3 m); the longer distance traveled with reduced power in the thrusters in charge in that direction leads to the observed longer duration for completing the desired maneuver.

The green graphs in Figure B.4, corresponding to the experiment **II**, visually depict the effect of the successfully performed runtime system adaptation of the submersible due to the thruster failure. The time instance when the thruster failure is triggered is shown with a vertical black line. After the waypoint **A** is reached and before the moment of the thruster failure, Function Design *fd_surge_all* is used, implying that all the thrusters are working properly and with full power. When failure is simulated in the thruster T_0 , the reconfiguration is triggered and the new Function Design is selected, that is, the one that does not require the malfunctioning thruster T_0 , *fd_surge_no_t0*. This Function Design implies the use of the opposite thruster on the same side of the hull, T_2 , operating at the same power (Figure B.4b), and the two thrusters on the other side of the hull, T_4 and T_6 , operating at half power (Figure B.4c,d). The reduction in power by half can be perceived by comparing it with the experiment **I** (graph in purple) when all thrusters were working properly.

Analogously, the blue graphs in Figure B.5, corresponding to the experiment **III**, illustrate the successfully performed runtime system adaptation due to the failure of thruster T_1 .

Finally, the performed experiments, **II** and **III**, are compared to the initial naive approach consisting of disabling the symmetric thruster, i.e. the thruster in the same position but on the opposite side of the robot. Table B.3 summarizes the numerical comparison between the experiments performed using the proposed metacontroller and the experiments performed using the naive approach. Table B.3a shows mean forces, Table B.3b duration, and Table B.3c latency of each experiment. It can be noted that the produced mean forces have similar values; however, higher values are obtained in the experiments with the metacontroller. This result is expected since the metacontroller avoids deactivation of a proper-functioning thruster; contrary to the naive approach. This result directly affects the tests' duration, causing the longer duration of the experiments done with the naive approach. Finally, the latency of the system response to thruster failure is compared and depicted in Table B.3c. Both experiments have similar values; however, slightly shorter latency is obtained with the naive approach due to the usage of fewer ROS nodes.

These experiments demonstrate the viability of the model-based approach for reconfiguring the deployed system. This approach provides benefits that manifest both in a) the mission fulfillment (in this case, shorter duration of the experiment in comparison with the most basic contingency handling, i.e., disabling the symmetrical thruster of the one malfunctioning) and b) the generality of the systems engineering process. Reduction of engineering time is a clear advantage of model-based reasoners; however, from the point of adaptation response time, it is possible that most fault-handling results for a specific UUV in the literature will have better metric values (RMSD and/or latency) than ours, as they are designed specifically for a robot and its concrete operation.



Figure B.2: Odometry measurements during the thruster failure experiments. Yellow squares depict the reference waypoints.

(a) The root-mean-square deviation (RMSD) of the vehicle position with respect to the reference trajectory		
Test	RMSD	Units
I	0.12	[m]
II	0.19	[m]
III	0.23	[m]
(b) Duration of each test		
Test	Duration	Units
I	70.2	[s]
II	168.7	[s]
III	88.3	[s]
(c) Latency between the thruster failure trigger and the system response		
Test	Latency	Units
II	1.81	[s]
III	1.09	[s]

Table B.1: Comparison between tests. **I** represent the test with all thrusters working. **II** represent the test with the failure of T_0 . **III** represent the test with the failure of T_1 .

(a) The root-mean-square deviation (RMSD) of the vehicle position with respect to the reference trajectory		
Test	RMSD	Units
I	0.12	[m]
II	0.19	[m]
III	0.23	[m]
(b) Duration of each test		
Test	Duration	Units
I	70.2	[s]
II	168.7	[s]
III	88.3	[s]
(c) Latency between the thruster failure trigger and the system response		
Test	Latency	Units
II	1.81	[s]
III	1.09	[s]

Table B.2: Mean force summary.

(a) Mean force			
	II	III	
Test	Mean F		Unit
With MC	0.21	0.4	[N]
Without MC	0.16	0.29	[N]
(b) Duration of the test			
	II	III	
Test	Duration		Unit
With MC	168.7	88.3	[s]
Without MC	203.2	113.1	[s]
(c) Latency between the thruster failure trigger and the system response			
	II	III	
Test	Latency		Unit
With MC	1.81	1.09	[s]
Without MC	0.91	0.72	[s]

Table B.3: Numerical comparison of the experiments performed with and without the metacontroller (MC). **II** represent the test with the failure of T_0 . **III** represent the test with the failure of T_1 .

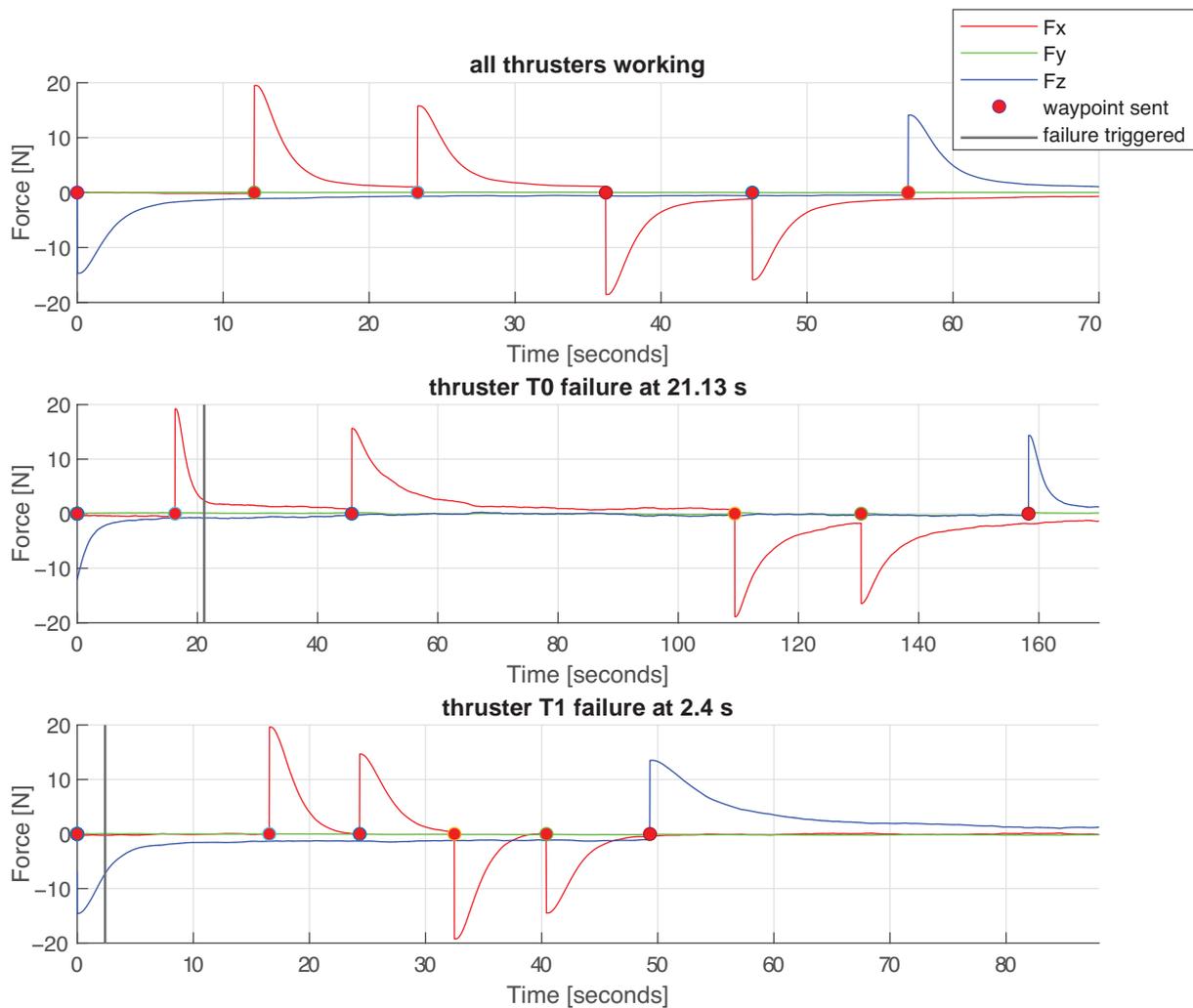


Figure B.3: Force reference commands obtained by the FL controller. Uppermost figure represent the *I* test, middle the *II* and bottom the *III*. The time instance when each waypoint is sent is depicted with a red circle, and the moment of the thruster failure is depicted with a vertical black line.

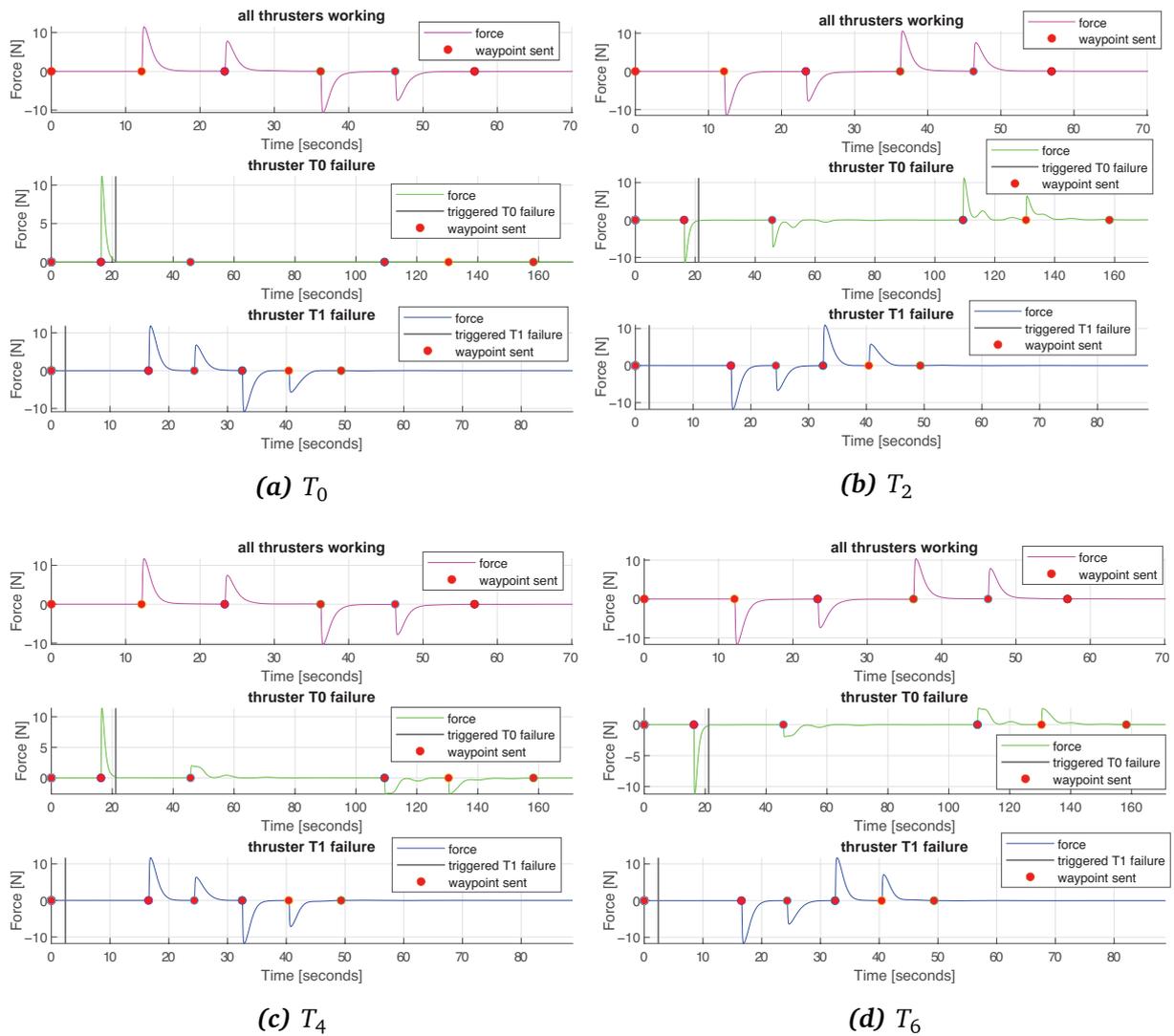


Figure B.4: Forces produced by thrusters in charge for surge movement in all three experiments. Purple graphs represent the I test, green the II and blue the III. The time instance when each waypoint is sent is depicted with a red circle, and the moment of the thruster failure is depicted with a vertical black line.

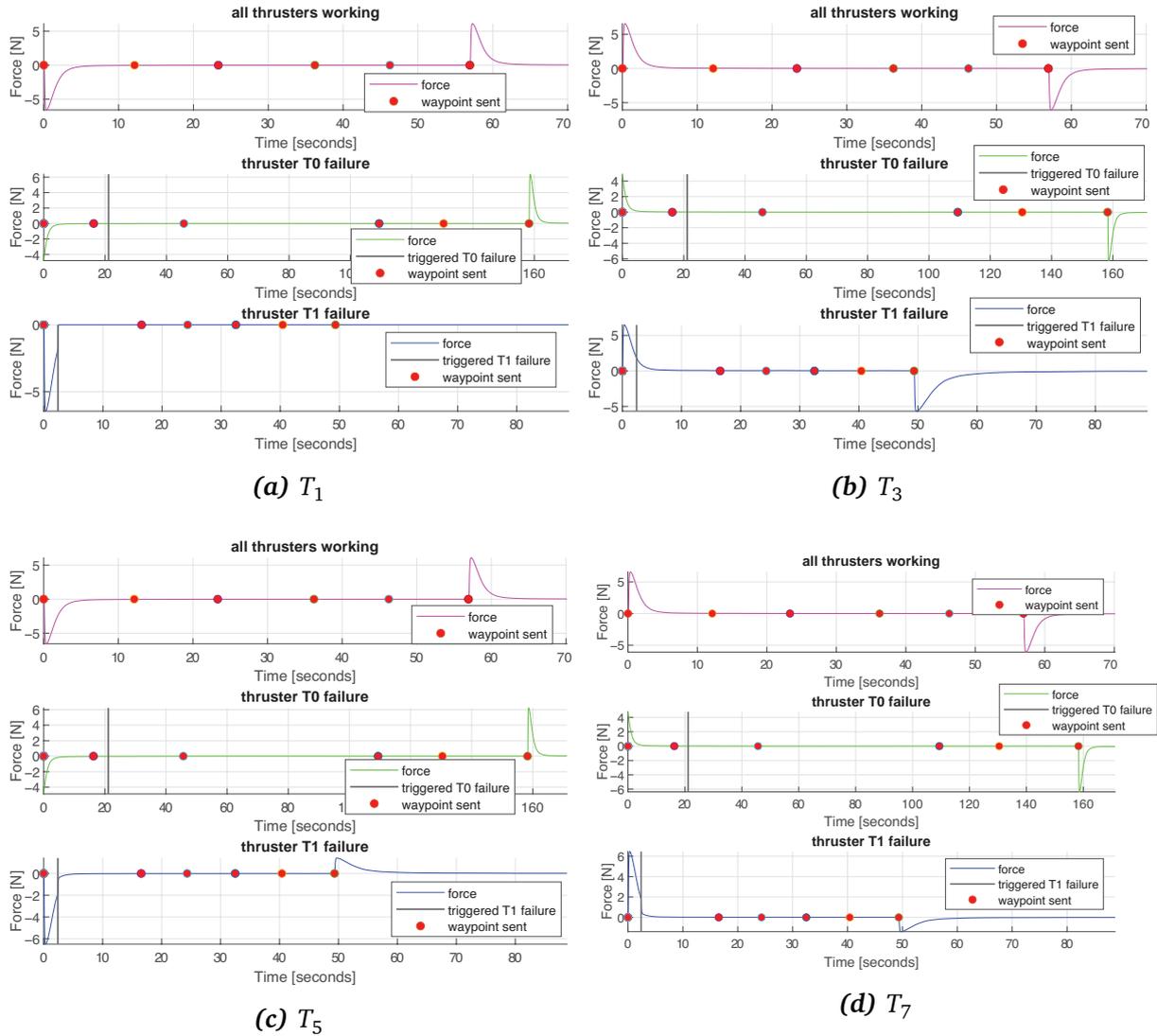


Figure B.5: Forces produced by thrusters in charge for heave movement in all three experiments. Purple graphs represent the *I* test, green the *II* and blue the *III*. The time instance when each waypoint is sent is depicted with a red circle, and the moment of the thruster failure is depicted with a vertical black line.

Bibliography

- [1] R. Brachman, “Systems that know what they’re doing,” *IEEE Intelligent Systems*, vol. 17, no. 6, pp. 67–71, 2002, doi: <https://doi.org/10.1109/MIS.2002.1134363>.
- [2] D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, “Systems engineering handbook. a guide for system life cycle processes and activities v4.0,” International Council on Systems Engineering (INCOSE), Hoboken, NJ, USA, Technical Publication INCOSE-TP-2003-002-04, 2015.
- [3] B. Boehm, “A spiral model of software development and enhancement,” *SIGSOFT Softw. Eng. Notes*, vol. 11, no. 4, p. 14–24, aug 1986, doi: <https://doi.org/10.1145/12944.12948>.
- [4] D. D. Walden and International Council on Systems Engineering, Eds., *INCOSE Systems Engineering handbook Version 5.0*. Wiley, 2023.
- [5] L. v. Bertalanffy, *General system theory : foundations, development, applications*. New York: George G. Braziller, 1968.
- [6] K. E. Boulding, “General systems theory—the skeleton of science,” *Management science*, vol. 2, no. 3, pp. 197–208, 1956, doi: <https://www.doi.org/10.1287/mnsc.2.3.197>.
- [7] G. J. Klir, *An approach to general systems theory*. New York: Van Nostrand Reinhold, 1969.
- [8] —, “The emergence of two-dimensional science in the information society,” *Systems Research*, vol. 2, no. 1, pp. 33–41, 1985, doi: <https://doi.org/10.1002/sres.3850020107>.
- [9] R. Rosen, “Some comments on systems and system theory,” *International Journal of General Systems*, vol. 13, no. 1, pp. 1–3, 1986, doi: <https://doi.org/10.1080/03081078608934949>.
- [10] G. A. Bekey, *Autonomous Robots: From Biological Inspiration to Implementation and Control (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [11] S. Franklin and A. Graesser, “Is it an agent, or just a program? a taxonomy for autonomous agents,” in *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ser. ECAI ’96. Berlin, Heidelberg: Springer-Verlag, 1996, p. 21–35.
- [12] J. M. Beer, A. D. Fisk, and W. A. Rogers, “Toward a framework for levels of robot autonomy in human-robot interaction,” *J. Hum.-Robot Interact.*, vol. 3, no. 2, p. 74–99, jul 2014, doi: <https://doi.org/10.5898/JHRI.3.2.Beer>.
- [13] R. Sanz, F. Matia, and S. Galan, “Fridges, elephants, and the meaning of autonomy and intelligence,” in *IEEE International Symposium on Intelligent Control - Proceedings*, Patras, Greece, 2000, pp. 217–222, doi: <https://doi.org/10.1109/isic.2000.882926>.

- [14] B. T. Clough, “Metrics, schmetrics! how the heck do you determine a uav’s autonomy anyway,” in *Proceedings of the Performance Metrics for Intelligent Systems Workshop*, 2002, p. 12–20.
- [15] F. P. B. Osinga, *Science, Strategy and War: The Strategic Theory of John Boyd*. Routledge, 2007.
- [16] H. Huang, K. Pavek, J. Albus, and E. Messina, “Autonomy levels for unmanned systems (ALFUS) framework: an update,” in *Unmanned Ground Vehicle Technology VII*, G. R. Gerhart, C. M. Shoemaker, and D. W. Gage, Eds., vol. 5804, International Society for Optics and Photonics. SPIE, 2005, pp. 439 – 448, doi: <https://doi.org/10.1117/12.603725>.
- [17] P. J. Durst, W. Gray, and United States. Army Test and Evaluation Command and United States. Army. Corps of Engineers and Engineer Research and Development Center (U.S.) and Geotechnical and Structures Laboratory (U.S.), *Levels of autonomy and autonomous system performance assessment for intelligent unmanned systems*, ser. ERDC/GSL SR, U. A. Test and E. Command, Eds., 2014.
- [18] P. J. Durst, W. Gray, and M. Trentini, “Development of a non-contextual model for determining the autonomy level of intelligent unmanned systems,” in *Unmanned Systems Technology XV*, R. E. Karlsen, D. W. Gage, C. M. Shoemaker, and G. R. Gerhart, Eds., vol. 8741, International Society for Optics and Photonics. SPIE, 2013, p. 874111, doi: <https://doi.org/10.1117/12.2014352>.
- [19] SEBoK Editorial Board, “The guide to the systems engineering body of knowledge (SEBoK), v. 2.8,” www.sebokwiki.org, Hoboken, NJ, USA, 2023, last Accessed [August, 2023].
- [20] D. Fisher, R. Linger, H. Lipson, T. Longstaff, N. Mead, and R. Ellison, “Survivable network systems: An emerging discipline,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-97-TR-013, 1997.
- [21] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955,” *AI Magazine*, vol. 27, no. 4, p. 12, Dec. 2006, doi: <https://doi.org/10.1609/aimag.v27i4.1904>.
- [22] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, MA: MIT Press, 1996.
- [23] S. J. Russell and P. Norvig, *Artificial Intelligence: a modern approach*, 4th ed. Pearson, 2021.
- [24] D. Vernon, *Artificial Cognitive Systems: A Primer*. The MIT Press, 2014.
- [25] P. Langley, J. E. Laird, and S. Rogers, “Cognitive architectures: Research issues and challenges,” *Cognitive Systems Research*, vol. 10, no. 2, pp. 141–160, 2009, doi: <https://doi.org/10.1016/j.cogsys.2006.07.004>.
- [26] C. Hernández, J. Bermejo-Alonso, and R. Sanz, “A self-adaptation framework based on functional knowledge for augmented autonomy in robots,” *Integrated Computer-Aided Engineering*, vol. 25, pp. 157–172, 2018, doi: <https://doi.org/10.3233/ICA-180565>.

- [27] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, “Introduction to diagnosis and fault-tolerant control,” in *Diagnosis and Fault-Tolerant Control*. Berlin, Heidelberg: Springer, 2006, pp. 1–32, doi: https://doi.org/10.1007/978-3-540-35653-0_1.
- [28] P. Jalote, *Fault tolerance in distributed systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1994.
- [29] A. Aviziens, “Fault-tolerant systems,” *IEEE Transactions on Computers*, vol. C-25, no. 12, pp. 1304–1312, 1976, doi: <https://doi.org/10.1109/TC.1976.1674598>.
- [30] G. E. Dullerud and F. Paganini, *A Course in Robust Control Theory: A Convex Approach*. New York, NY: Springer New York, 2000, doi: <https://doi.org/10.1007/978-1-4757-3290-0>.
- [31] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, “Software engineering for self-adaptive systems: A research roadmap,” in *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26, doi: https://doi.org/10.1007/978-3-642-02161-9_1.
- [32] O. N. Robert Hirschfeld, Pascal Costanza, “Context-oriented programming,” *Journal of Object Technology*, vol. 7, no. 3, pp. 125–151, 2008, doi: <http://doi.org/10.5381/jot.2008.7.3.a4>.
- [33] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, “Engineering self-adaptive systems through feedback loops,” in *Software Engineering for Self-Adaptive Systems. Lecture Notes in Computer Science*, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer, 2009, pp. 48–70, doi: https://doi.org/10.1007/978-3-642-02161-9_3.
- [34] D. Garlan, S.-W. Cheng, and B. Schmerl, “Increasing system dependability through architecture-based self-repair,” in *Architecting Dependable Systems*, R. de Lemos, C. Gacek, and A. Romanovsky, Eds. Berlin, Heidelberg: Springer, 2003, pp. 61–89.
- [35] R. S. Burns, *Advanced Control Engineering*. Oxford: Butterworth-Heinemann, 2001, doi: <https://doi.org/10.1016/B978-0-7506-5100-4.X5000-1>.
- [36] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Camara, R. Calinescu, M. B. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, H. A. Müller, R. Rouvoy, C. M. F. Rubira, E. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. M. Villegas, T. Vogel, and F. Zambonelli, “Software engineering for self-adaptive systems: Research challenges in the provision of assurances,” in *Software Engineering for Self-Adaptive Systems III. Assurances*, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Cham: Springer International Publishing, 2017, pp. 3–30, doi: https://doi.org/10.1007/978-3-319-74183-3_1.
- [37] S. Dobson, S. Denazis, A. Fernández, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, “A survey of autonomic communications,”

- ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, p. 223–259, dec 2006, doi: <https://doi.org/10.1145/1186778.1186782>.
- [38] P. Horn, “Autonomic computing: IBM’s Perspective on the State of Information Technology,” IBM, 2001.
- [39] J. Kephart and D. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41–50, 2003, doi: <http://doi.org/10.1109/MC.2003.1160055>.
- [40] IBM Corporation, “An architectural blueprint for autonomic computing,” White Paper, Tech. Rep., 2005.
- [41] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura, “Incorporating architecture-based self-adaptation into an adaptive industrial software system,” *Journal of Systems and Software*, vol. 122, pp. 507–523, 2016, doi: <https://doi.org/10.1016/j.jss.2015.09.021>.
- [42] R. Sanz, I. López, J. Bermejo, R. Chinchilla, and R. Conde, “SELF-X: THE CONTROL WITHIN,” *IFAC Proceedings Volumes. 16th IFAC World Congress*, vol. 38, no. 1, pp. 179–184, 2005, doi: <https://doi.org/10.3182/20050703-6-CZ-1902.01071>.
- [43] O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, and M. van Steen, *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.
- [44] R. Sanz, I. López, and J. Bermejo-Alonso, “A rationale and vision for machine consciousness in complex controllers,” in *Artificial Consciousness*, A. Chella and R. Manzotti, Eds. Imprint Academic, 2007, pp. 141–155.
- [45] S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, *Self-Aware Computing Systems*. Cham: Springer International Publishing, 2017.
- [46] F. Crick and C. Koch, “The Problem of Consciousness,” *Scientific American*, vol. 267, no. 3, pp. 152–159, 1992.
- [47] A. Chella and R. Manzotti, *Artificial Consciousness*, 2nd ed. Imprint Academic, 2013.
- [48] J. E. Tardy, *The Creation of a Conscious Machine: The Quest for Artificial Intelligence*, 4th ed. Sysjet Inc., 2018.
- [49] A. Cardon, *Beyond Artificial Intelligence: From Human Consciousness to Artificial Consciousness*. Wiley, 2018.
- [50] T. R. Besold, L. Zaadnoordijk, and D. Vernon, “Feeling functional: A formal account of artificial phenomenology,” *Journal of Artificial Intelligence and Consciousness*, vol. 08, no. 01, pp. 147–160, 2021, doi: <https://doi.org/10.1142/S2705078521500077>.
- [51] P. R. Lewis, “Self-aware computing systems: From psychology to engineering,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1044–1049, doi: <https://doi.org/10.23919/DATE.2017.7927144>.

- [52] N. S. Voros, W. Mueller, and C. Snook, “An introduction to formal methods,” in *UML-B Specification for Proven Embedded Systems Design*. Boston, MA: Springer US, 2004, pp. 1–20, doi: https://doi.org/10.1007/978-1-4020-2867-0_1.
- [53] A. Hall, “Seven myths of formal methods,” *IEEE Software*, vol. 7, no. 5, pp. 11–19, 1990, doi: <https://doi.org/10.1109/52.57887>.
- [54] K. S. Schweiker, S. Varadarajan, D. I. Spivak, P. Schultz, R. Wisnesky, and P. Marco, “Operadic analysis of distributed systems,” NASA Center for AeroSpace Information, Tech. Rep., 2015.
- [55] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
- [56] M. Griffin, “How do we fix system engineering?” *AI Proceedings of the 61st International Astronautical Congress*, September 2010.
- [57] J. Holt and S. Perry, “Part 1 – introduction,” in *SysML for Systems Engineering: A Model-Based Approach*, ser. Computing. Institution of Engineering and Technology, 2018, pp. 3–13, doi: https://doi.org/10.1049/PBPC020E_ch1.
- [58] “ISO/IEC/IEEE International Standard - Systems Engineering – System Life Cycle Processes,” *ISO/IEC 15288 First edition 2002-11-01*, pp. 1–70, 2001, doi: <https://doi.org/10.1109/IEEESTD.2001.8684399>.
- [59] “ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes,” *ISO/IEC/IEEE 15288:2023(E)*, pp. 1–128, 2023, doi: <https://doi.org/10.1109/IEEESTD.2023.10123367>.
- [60] The International Council on Systems Engineering (INCOSE), “INCOSE Definitions,” *INCOSE Definitions*, 2019.
- [61] H. Chestnut, *Systems Engineering Tools*, ser. Systems Engineering Tools. Wiley, 1965, no. 41.
- [62] Federal Aviation Agency (USA FAA), *Systems Engineering Manual [definition contributed by Simon Ramo]*. Federal Aviation Agency (USA FAA), 2004.
- [63] H. Eisner, *Essentials of Project and Systems Engineering Management*, 3rd ed. Chichester, UK: Wiley, 2008.
- [64] G. Shea, “NASA Systems Engineering Handbook Revision 2,” 2017.
- [65] S. A. Sheard and A. Mostashari, “Complexity types: From science to systems engineering,” *INCOSE International Symposium*, vol. 21, no. 1, pp. 673–682, June 2011, doi: <https://doi.org/10.1002/j.2334-5837.2011.tb01235.x>.
- [66] J. Holt and S. Perry, “Introduction to sysml and systems modelling,” in *SysML for Systems Engineering: A Model-Based Approach*, ser. Computing. Institution of Engineering and Technology, 2018, pp. 81–128, doi: https://doi.org/10.1049/PBPC020E_ch4.
- [67] “ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering,” *ISO/IEC/IEEE 29148:2018(E)*, pp. 1–104, 2018, doi: <https://doi.org/10.1109/IEEESTD.2018.8559686>.

- [68] J. Holt, S. Perry, and M. Brownsword, *Model-Based Requirements Engineering*, ser. Computing and Networks. Institution of Engineering and Technology, 2011, doi: <https://doi.org/10.1049/PBPC009E>.
- [69] ISO/IEC/IEEE 42020:2019, “ISO/IEC/IEEE International Standard - Software, systems and enterprise – Architecture processes,” *ISO/IEC/IEEE 42020:2019(E)*, pp. 1–126, 2019, doi: <https://doi.org/10.1109/IEEESTD.2019.8767004>.
- [70] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: Systems Modeling Language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.
- [71] “ISO/IEC/IEEE International Standard - Systems and software engineering – Software life cycle processes,” *ISO/IEC/IEEE 12207:2017(E) First edition 2017-11*, pp. 1–157, 2017, doi: <https://doi.org/10.1109/IEEESTD.2017.8100771>.
- [72] “ISO/IEC/IEEE International Standard - Systems and software engineering - Life cycle management - Part 1:Guidelines for life cycle management,” *ISO/IEC/IEEE 24748-1:2018(E)*, pp. 1–82, 2018, doi: <https://doi.org/10.1109/IEEESTD.2018.8526560>.
- [73] Object Management Group, “What Is OMG SysML?” <http://www.omg.sysml.org> [Last Accessed October, 2023].
- [74] S. T. Force, “OMG Systems Modeling Language(SysML),” Object Management Group (OMG), Tech. Rep., 2023. [Online]. Available: <https://www.omg.org/spec/SysML/2.0/Language>
- [75] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML, Third Edition: The Systems Modeling Language*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.
- [76] S. C. Shapiro, “Encyclopedia of cognitive science.” Macmillan Publishers Ltd., 2003, vol. 2, ch. Knowledge Representation, pp. 671 – 680.
- [77] W. Ertel, *Introduction to Artificial Intelligence*, 2nd ed. Springer Publishing Company, Incorporated, 2018.
- [78] J. A. Robinson, “A machine-oriented logic based on the resolution principle,” *J. ACM*, vol. 12, no. 1, p. 23–41, jan 1965, doi: <https://doi.org/10.1109/IROS45743.2020.9341207>.
- [79] S. Rudolph, “Foundations of description logics,” in *Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 76–136, doi: https://doi.org/10.1007/978-3-642-23032-5_2.
- [80] I. Horrocks, O. Kutz, and U. Sattler, “The even more irresistible sroiq,” in *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, ser. KR’06. AAAI Press, 2006, p. 57–67, doi: <https://doi.org/10.5555/3029947.3029959>.
- [81] L. S. Sterling and E. Shapiro, *The art of PROLOG: Advanced programming techniques*. MIT Press, 1992.

- [82] D. McDermott, M. Ghallab, A. E. Howe, C. A. Knoblock, A. Ram, M. M. Veloso, D. S. Weld, and D. E. Wilkins, “PDDL—the planning domain definition language,” 1998.
- [83] D. Bryce, “6th international planning competition: Uncertainty part,” <https://api.semanticscholar.org/CorpusID:18368139>, 2008, last Accessed September, 2023.
- [84] H. L. S. Younes and M. L. Littman, “PPDDL 1.0 : An extension to PDDL for expressing planning domains with probabilistic effects,” School of Computer Science. Carnegie Mellon University, Tech. Rep. CMU-CS-04-167, 2004.
- [85] J. Hoffmann and R. I. Brafman, “Conformant planning via heuristic forward search: A new approach,” *Artificial Intelligence*, vol. 170, no. 6, pp. 507–541, 2006, doi: <https://doi.org/10.1016/j.artint.2006.01.003>.
- [86] D. L. Kovács, “A multi-agent extension of PDDL3.1,” 2012.
- [87] F. Martín, J. Ginés, F. J. Rodríguez, and V. Matellán, “PlanSys2: A Planning System Framework for ROS2,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - October 1, 2021*. IEEE, 2021.
- [88] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, “Rosplan: Planning in the robot operating system,” in *International Conference on Automated Planning and Scheduling*, vol. 25, Apr. 2015, pp. 333–341, doi: <https://doi.org/10.1609/icaps.v25i1.13699>.
- [89] M. A. González-Santamarta, F. J. Rodríguez-Lera, C. Fernández-Llamas, and V. Matellán-Olivera, “MERLIN2: MachinEd Ros 2 pLanINg,” *Software Impacts*, vol. 15, p. 100477, 2023, doi: <https://doi.org/10.1016/j.simpa.2023.100477>.
- [90] M. Mayr, F. Rovida, and V. Krueger, “Skiros2: A skill-based robot control platform for ros,” 2023.
- [91] T. R. Gruber, “A translation approach to portable ontology specifications,” *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993, doi: <https://doi.org/10.1006/knac.1993.1008>.
- [92] I. Horrocks, “Ontologies and the semantic web,” *Commun. ACM*, vol. 51, no. 12, p. 58–67, dec 2008, doi: <https://doi.org/10.1145/1409360.1409377>.
- [93] W3C OWL Working Group, “OWL 2 Web Ontology Language Document Overview (Second Edition),” World Wide Web Consortium (W3C), Tech. Rep., 2012. [Online]. Available: <http://www.w3.org/TR/owl2-overview/>
- [94] M. A. Musen, “The protégé project: a look back and a look forward,” *AI Matters*, vol. 1, no. 4, pp. 4–12, 2015, doi: <https://doi.org/10.1145/2757001.2757003>.
- [95] J.-B. Lamy, “Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies,” *Artificial Intelligence in Medicine*, vol. 80, pp. 11–28, 2017, doi: <https://doi.org/10.1016/j.artmed.2017.07.002>.

- [96] F. Lehmann and E. Y. Rodin, *Semantic networks in artificial intelligence*. Oxford: Pergamon Press, 1992.
- [97] P. Hitzler, “A review of the semantic web field,” *Commun. ACM*, vol. 64, no. 2, p. 76–83, jan 2021, doi: <https://doi.org/10.1145/3397512>.
- [98] A. Singhal, “Introducing the knowledge graph: things, not strings,” 2012, <https://www.blog.google/products/search/introducing-knowledge-graph-things-not/> [Last Accessed December, 2023].
- [99] M. Färber, “The microsoft academic knowledge graph: A linked data source with 8 billion triples of scholarly data,” in *The Semantic Web – ISWC 2019*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Eds. Cham: Springer International Publishing, 2019, pp. 113–129.
- [100] A. Fan, C. Gardent, C. Braud, and A. Bordes, “Using local knowledge graph construction to scale Seq2Seq models to multi-document inputs,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4186–4196, doi: <https://doi.org/10.18653/v1/D19-1428>.
- [101] IBM Research, “Knowledge Graph construction gets big boost from AI,” 2021, <https://research.ibm.com/blog/knowledge-graph-ai> [Last Accessed December 2023].
- [102] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, “Industry-scale knowledge graphs: Lessons and challenges,” *Commun. ACM*, vol. 62, no. 8, p. 36–43, jul 2019, doi: <https://doi.org/110.1145/3331166>.
- [103] A. Hogan, E. Blomqvist, M. Cochez, C. D’amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, “Knowledge graphs,” *ACM Comput. Surv.*, vol. 54, no. 4, jul 2021, doi: <https://doi.org/10.1145/3447772>.
- [104] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč, “Foundations of modern query languages for graph databases,” *ACM Comput. Surv.*, vol. 50, no. 5, sep 2017, doi: <https://doi.org/10.1145/3104031>.
- [105] O. Hartig, P.-A. Champin, G. Kellogg, R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carroll, and B. McBride, “RDF 1.2 Concepts and Abstract Syntax,” W3C Member Submission, W3C, Tech. Rep., 2023. [Online]. Available: <https://www.w3.org/TR/rdf12-concepts/>
- [106] S. Harris, A. Seaborne, and E. Prud’hommeaux, “SPARQL 1.1 Query Language,” W3C Member Submission, W3C, Tech. Rep., 2013. [Online]. Available: <https://www.w3.org/TR/sparql11-query/>
- [107] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, “Cypher: An evolving query language for property graphs,” in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1433–1445, doi: <https://doi.org/10.1145/3183713.3190657>.

- [108] M. A. Rodriguez, “The gremlin graph traversal machine and language (invited talk),” in *Proceedings of the 15th Symposium on Database Programming Languages*, ser. DBPL 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 1–10, doi: <https://doi.org/10.1145/2815072.2815073>.
- [109] R. Angles, M. Arenas, P. Barcelo, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, O. van Rest, and H. Voigt, “G-core: A core for future graph query languages,” in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1421–1432, doi: <https://doi.org/10.1145/3183713.3190654>.
- [110] L. Ehrlinger and W. Wöß, “Towards a definition of knowledge graphs.” *Proc. of SEMANTiCS Posters & Demos.*, vol. 48, no. 1-4, p. 2, 2016.
- [111] N. Guarino, *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*, 1st ed. NLD: IOS Press, 1998.
- [112] M. Hepp, D. Bachlechner, and K. Siorpaes, “Ontowiki: Community-driven ontology engineering and ontology usage based on wikis,” in *Proceedings of the 2006 International Symposium on Wikis*, ser. WikiSym ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 143–144, doi: <https://doi.org/10.1145/1149453.1149487>.
- [113] R. Sanz, I. Alarcon, M. Segarra, J. A. Clavijo, and A. de Antonio, “Progressive domain focalization in intelligent control systems,” *Control Engineering Practice*, vol. 7, no. 5, pp. 665–671, 1999, doi: [https://doi.org/10.1016/S0967-0661\(99\)00012-X](https://doi.org/10.1016/S0967-0661(99)00012-X).
- [114] A. Pease, *Ontology: A Practical Guide*. Angwin, CA: Articulate Software Press, 2011.
- [115] I. Niles and A. Pease, “Toward a Standard Upper Ontology,” in *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, C. Welty and B. Smith, Eds., 2001, pp. 2–9.
- [116] C. E. Brown, A. Pease, and J. Urban, “Translating SUMO-K to Higher-Order Set Theory,” in *Frontiers of Combining Systems (FroCoS)*, to appear, 2023.
- [117] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, R. Oltramari, and L. Schneider, “Sweetening Ontologies with DOLCE,” in *In: Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2002, pp. 166–181.
- [118] V. Mascardi, V. Cordì, and P. Rosso, “A comparison of upper ontologies (technical report disi-tr-06-21),” Dipartimento di Informatica e Scienze dell’Informazione (DISI) and Universidad Politécnic de Valencia, Tech. Rep., 2006.
- [119] R. Arp, B. Smith, and A. D. Spear, *Building Ontologies with Basic Formal Ontology*. The MIT Press, 07 2015, doi: <https://doi.org/10.7551/mitpress/9780262527811.001.0001>.
- [120] M. Bunge, *Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World*. Boston, MA: Reidel, 1977.
- [121] Y. Wand and R. Weber, “On the ontological expressiveness of information systems analysis and design grammars,” *Information Systems Journal*, vol. 3, no. 4, pp. 217–237, 1993.

- [122] D. B. Lenat, “Cyc: A large-scale investment in knowledge infrastructure,” *Commun. ACM*, vol. 38, no. 11, p. 33–38, nov 1995, doi: <https://doi.org/10.1145/219717.219745>.
- [123] R. Lukyanenko, V. C. Storey, and O. Pastor, “Foundations of information technology based on bunge’s systemist philosophy of reality,” *Software and Systems Modeling*, vol. 20, no. 4, pp. 921–938, 2021, doi: <https://doi.org/10.1007/s10270-021-00862-5>.
- [124] S. R. Fiorini, J. Bermejo-Alonso, P. Gonçalves, E. Pignaton de Freitas, A. Olivares Alarcos, J. I. Olszewska, E. Prestes, C. Schlenoff, S. V. Ragavan, S. Redfield, B. Spencer, and H. Li, “A suite of ontologies for robotics and automation [industrial activities],” *IEEE Robotics & Automation Magazine*, vol. 24, no. 1, pp. 8–11, 2017, doi: <https://doi.org/10.1109/MRA.2016.2645444>.
- [125] E. Prestes, J. L. Carbonera, S. Rama Fiorini, V. A. M. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Goncalves, M. E. Barreto, M. Habib, A. Chibani, S. Gérard, Y. Amirat, and C. Schlenoff, “Towards a core ontology for robotics and automation,” *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1193–1204, 2013, doi: <https://doi.org/10.1016/j.robot.2013.04.005>.
- [126] IEEE SA, “IEEE Standard Ontologies for Robotics and Automation,” *IEEE Std 1872-2015*, pp. 1–60, 2015, doi: <https://doi.org/10.1109/IEEESTD.2015.7084073>.
- [127] “Ieee approved draft standard for robot task representation,” *IEEE P1872.1/D5, October 2023*, pp. 1–35, 2024.
- [128] S. Balakirsky, C. Schlenoff, S. Fiorini, S. Redfield, M. Barreto, H. Nakawala, J. Carbonera, L. Soldatova, J. Bermejo-Alonso, F. Maikore, P. Goncalves, E. Momi, S. Ragavan, and T. Haidegger, “Towards a robot task ontology standard,” in *Proceedings of the Manufacturing Science and Engineering Conference (MSEC), Los Angeles, CA, US, 2017-06-08 04:06:00 2017*.
- [129] IEEE SA, “IEEE Standard for Autonomous Robotics (AuR) Ontology,” IEEE, Standard IEEE Std 1872.2-2021, 2022, doi: <https://doi.org/10.1109/IEEESTD.2022.9774339>.
- [130] J. Bermejo-Alonso, R. Sanz, M. Rodríguez, and C. Hernández, “An ontology-based approach for autonomous systems’ description and engineering,” in *Knowledge-Based and Intelligent Information and Engineering Systems*, R. Setchi, I. Jordanov, R. J. Howlett, and L. C. Jain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 522–531.
- [131] D. Beßler, R. Porzel, M. Pomarlan, A. Vyas, S. Höffner, M. Beetz, R. Malaka, and J. Bateman, “Foundations of the Socio-physical Model of Activities (SOMA) for Autonomous Robotic Agents,” in *Formal Ontology in Information Systems - Proceedings of the 12th International Conference, FOIS 2021, Bozen-Bolzano, Italy, September 13-16, 2021*, ser. Frontiers in Artificial Intelligence and Applications, B. Brodaric and F. Neuhaus, Eds. IOS Press, 2021, accepted for publication. [Online]. Available: <https://ai.uni-bremen.de/papers/bessler21soma.pdf>
- [132] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan, R. Chou, J. Glanville, J. M.

- Grimshaw, A. Hróbjartsson, M. M. Lalu, T. Li, E. W. Loder, E. Mayo-Wilson, S. McDonald, L. A. McGuinness, L. A. Stewart, J. Thomas, A. C. Tricco, V. A. Welch, P. Whiting, and D. Moher, “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews,” *BMJ*, vol. 372, 2021, doi: <https://doi.org/10.1136/bmj.n71>.
- [133] R. Gayathri and V. Uma, “Ontology based knowledge representation technique, domain modeling languages and planners for robotic path planning: A survey,” *ICT Express*, vol. 4, no. 2, pp. 69–74, 2018, doi: <https://doi.org/10.1016/j.ict.2018.04.008>.
- [134] R. Gayathri and V. Uma, “A review of description logic-based techniques for robot task planning,” *Studies in Computational Intelligence*, vol. 771, pp. 101–107, 2019, doi: https://doi.org/10.1007/978-981-10-8797-4_11.
- [135] M. Cornejo-Lupa, R. Ticona-Herrera, Y. Cardinale, and D. Barrios-Aranibar, “A survey of ontologies for simultaneous localization and mapping in mobile robots,” *ACM Computing Surveys*, vol. 53, no. 5, 2020, doi: <https://doi.org/10.1145/3408316>.
- [136] S. Manzoor, Y. Rocha, S.-H. Joo, S.-H. Bae, E.-J. Kim, K.-J. Joo, and T.-Y. Kuc, “Ontology-based knowledge representation in robotic systems: A survey oriented toward applications,” *Applied Sciences (Switzerland)*, vol. 11, no. 10, 2021, doi: <https://doi.org/10.3390/app11104324>.
- [137] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li, “A review and comparison of ontology-based approaches to robot autonomy,” *Knowledge Engineering Review*, 2019, doi: <https://doi.org/10.1017/S0269888919000237>.
- [138] M. Tenorth and M. Beetz, “Knowrob — knowledge processing for autonomous personal robots,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 4261–4266, doi: <https://doi.org/10.1109/IROS.2009.5354602>.
- [139] —, “Knowrob: A knowledge processing infrastructure for cognition-enabled robots,” *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013, doi: <https://doi.org/10.1177/0278364913481635>.
- [140] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels, “Know rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 512–519, doi: <https://doi.org/10.1109/ICRA.2018.8460964>.
- [141] M. Waibel, M. Beetz, J. Civera, R. D’Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schießle, M. Tenorth, O. Zweigle, and R. V. De Molengraft, “Roboearth,” *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011, doi: <https://doi.org/10.1109/MRA.2011.941632>.
- [142] L. Riazuelo, M. Tenorth, D. Di Marco, M. Salas, D. Gálvez-López, L. Mösenlechner, L. Kunze, M. Beetz, J. D. Tardós, L. Montano, and J. M. M. Montiel, “Roboearth semantic mapping: A cloud enabled knowledge-based approach,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 432–443, 2015, doi: <https://doi.org/10.1109/TASE.2014.2377791>.

- [143] M. Tenorth, A. Clifford Perzylo, R. Lafrenz, and M. Beetz, “The roboearth language: Representing and exchanging knowledge about actions, objects, and environments,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1284–1289, doi: <https://doi.org/10.1109/ICRA.2012.6224812>.
- [144] J. Crespo, R. Barber, O. M. Mozos, D. BeBler, and M. Beetz, “Reasoning systems for semantic navigation in mobile robots,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5654–5659, doi: <https://doi.org/10.1109/IROS.2018.8594271>.
- [145] M. Beetz, L. Mösenlechner, and M. Tenorth, “CRAM — A Cognitive Robot Abstract Machine for everyday manipulation in human environments,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1012–1017, doi: <https://doi.org/10.1109/IROS.2010.5650146>.
- [146] M. Beetz, M. Tenorth, and J. Winkler, “Open-EASE – a knowledge processing service for robots and robotics/ai researchers,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, 2015, finalist for the Best Cognitive Robotics Paper Award.
- [147] M. Diab, A. Akbari, M. Ud Din, and J. Rosell, “PMK—A Knowledge Processing Framework for Autonomous Robotics Perception and Manipulation,” *Sensors*, vol. 19, no. 5, 2019, doi: <https://doi.org/10.3390/s19051166>.
- [148] G. H. Lim, I. H. Suh, and H. Suh, “Ontology-based unified robot knowledge for service robots in indoor environments,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 3, pp. 492–509, 2011, doi: <https://doi.org/10.1109/TSMCA.2010.2076404>.
- [149] M. Diab, M. Pomarlan, S. Borgo, D. Beßler, J. Rosell, J. Bateman, and M. Beetz, “Failrecont – an ontology-based framework for failure interpretation and recovery in planning and execution,” in *Proceedings of the Joint Ontology Workshops co-located with the Bolzano Summer of Knowledge (BOSK 2021), Virtual & Bozen-Bolzano, Italy, September 13th to September 17th, 2021*, ser. CEUR Workshop Proceedings. CEUR-WS.org, 2021, accepted for publication.
- [150] M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. Bateman, and M. Beetz, “An ontology for failure interpretation in automated planning and execution,” in *Robot 2019: Fourth Iberian Robotics Conference*, M. F. Silva, J. Luís Lima, L. P. Reis, A. Sanfeliu, and D. Tardioli, Eds. Cham: Springer International Publishing, 2020, pp. 381–390.
- [151] L. Antanas, P. Moreno, M. Neumann, R. P. de Figueiredo, K. Kersting, J. Santos-Victor, and L. De Raedt, “Semantic and geometric reasoning for robotic grasping: a probabilistic logic approach,” *Autonomous Robots*, vol. 43, no. 6, pp. 1393–1418, Aug 2019, doi: <https://doi.org/10.1007/s10514-018-9784-8>.
- [152] E. Aguado, Z. Milosevic, C. Hernández, R. Sanz, M. Garzon, D. Bozhinoski, and C. Rossi, “Functional self-awareness and metacontrol for underwater robot autonomy,” *Sensors*, vol. 21, no. 4, pp. 1–28, 2021, doi: <https://doi.org/10.3390/s21041210>.

- [153] D. Bozhinoski, M. G. Oviedo, N. H. Garcia, H. Deshpande, G. van der Hoorn, J. Tjerngren, A. Wařowski, and C. H. Corbato, “MROS: runtime adaptation for robot control architectures,” *Advanced Robotics*, vol. 36, no. 11, pp. 502–518, 2022, doi: <https://doi.org/10.1080/01691864.2022.2039761>.
- [154] I. H. Suh, G. H. Lim, W. Hwang, H. Suh, J.-H. Choi, and Y.-T. Park, “Ontology-based multi-layered robot knowledge framework (omrkf) for robot intelligence,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 429–436, doi: <https://doi.org/10.1109/IROS.2007.4399082>.
- [155] X. Li, S. Bilbao, T. Martın-Wanton, J. Bastos, and J. Rodriguez, “SWARMs Ontology: A Common Information Model for the Cooperation of Underwater Robots,” *Sensors*, vol. 17, no. 3, 2017, doi: <https://doi.org/10.3390/s17030569>.
- [156] K. B. Laskey, “MEBN: A language for first-order Bayesian knowledge bases,” *Artificial Intelligence*, vol. 172, no. 2, pp. 140–178, 2008, doi: <https://doi.org/10.1016/j.artint.2007.09.006>.
- [157] X. Sun, Y. Zhang, and J. Chen, “RTPO: A Domain Knowledge Base for Robot Task Planning,” *Electronics*, vol. 8, no. 10, 2019, doi: <https://doi.org/10.3390/electronics8101105>.
- [158] G. Burroughes and Y. Gao, “Ontology-based self-reconfiguring guidance, navigation, and control for planetary rovers,” *Journal of Aerospace Information Systems*, vol. 13, no. 8, pp. 316–328, 2016, doi: <https://doi.org/10.2514/1.1010378>.
- [159] R. Chandra and R. P. Rocha, “Knowledge-based framework for human-robots collaborative context awareness in usar missions,” in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2016, pp. 335–340, doi: <https://doi.org/10.1109/ICARSC.2016.50>.
- [160] L. Huang, H. Liang, B. Yu, B. Li, and H. Zhu, “Ontology-based driving scene modeling, situation assessment and decision making for autonomous vehicles,” in *2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, 2019, pp. 57–62, doi: <https://doi.org/10.1109/ACIRS.2019.8935984>.
- [161] X. Sun, Y. Zhang, and J. Chen, “High-level smart decision making of a robot based on ontology in a search and rescue scenario,” *Future Internet*, vol. 11, no. 11, 2019, doi: <https://doi.org/10.3390/fi11110230>.
- [162] M. Diab, Muhayyuddin, A. Akbari, and J. Rosell, “An ontology framework for physics-based manipulation planning,” in *ROBOT 2017: Third Iberian Robotics Conference*, A. Ollero, A. Sanfeliu, L. Montano, N. Lau, and C. Cardeira, Eds. Cham: Springer International Publishing, 2018, pp. 452–464.
- [163] S. Lemaignan, R. Ros, L. Mosenlechner, R. Alami, and M. Beetz, “Oro, a knowledge management platform for cognitive architectures in robotics,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3548–3553, doi: <https://doi.org/10.1109/IROS.2010.5649547>.
- [164] A. Olivares-Alarcos, S. Foix, S. Borgo, and Guillem Alenya, “OCRA – An ontology for collaborative robotics and adaptation,” *Computers in Industry*, vol. 138, p. 103627, 2022, doi: <https://doi.org/10.1016/j.compind.2022.103627>.

- [165] D. S. Chang, G. H. Cho, and Y. S. Choi, “Ontology-based knowledge model for human-robot interactive services,” in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2029–2038, doi: <https://doi.org/10.1145/3341105.3373977>.
- [166] M. J. Huber, “Jam: A bdi-theoretic mobile agent architecture,” in *Proceedings of the Third Annual Conference on Autonomous Agents*, ser. AGENTS '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 236–243, doi: <https://doi.org/10.1145/301136.301202>.
- [167] Z. Ji, R. Qiu, A. Noyvirt, A. Soroka, M. Packianather, R. Setchi, D. Li, and S. Xu, “Towards automated task planning for service robots using semantic knowledge representation,” in *IEEE 10th International Conference on Industrial Informatics*, 2012, pp. 1194–1201, doi: <https://doi.org/10.1109/INDIN.2012.6301131>.
- [168] M. Stenmark and J. Malec, “Knowledge-based instruction of manipulation tasks for industrial robotics,” *Robotics and Computer-Integrated Manufacturing, Special Issue on Knowledge Driven Robotics and Manufacturing*, vol. 33, pp. 56–67, 2015, doi: <https://doi.org/10.1016/j.rcim.2014.07.004>.
- [169] T. Hoebert, W. Lopuschitz, M. Vincze, and M. Merdan, “Knowledge-driven framework for industrial robotic systems,” *Journal of Intelligent Manufacturing*, Aug 2021, doi: <https://doi.org/10.1007/s10845-021-01826-8>.
- [170] A. Perzylo, N. Somani, M. Rickert, and A. Knoll, “An ontology for cad data and geometric constraints as a link between product models and semantic robot task descriptions,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4197–4203, doi: <https://doi.org/10.1109/IROS.2015.7353971>.
- [171] M. Merdan, T. Hoebert, E. List, and W. Lopuschitz, “Knowledge-based cyber-physical systems for assembly automation,” *Production & Manufacturing Research*, vol. 7, no. 1, pp. 223–254, 2019, doi: <https://doi.org/10.1080/21693277.2019.1618746>.
- [172] R. Bernardo, R. Farinha, and P. J. S. Gonçalves, “Knowledge and tasks representation for an industrial robotic application,” in *ROBOT 2017: Third Iberian Robotics Conference*, A. Ollero, A. Sanfeliu, L. Montano, N. Lau, and C. Cardeira, Eds. Cham: Springer International Publishing, 2018, pp. 441–451.
- [173] S. Borgo, A. Cesta, A. Orlandini, and A. Umbrico, “Knowledge-based adaptive agents for manufacturing domains,” *Engineering with Computers*, vol. 35, no. 3, pp. 755–779, Jul 2019, doi: <https://doi.org/10.1007/s00366-018-0630-6>.
- [174] R. Sanz, J. Bermejo, M. Rodríguez, and E. Aguado, “The Role of Knowledge in Cyber-Physical Systems of Systems,” *TASK Quarterly*, vol. 25, no. 3, pp. 355–373, 2021.
- [175] R. S. Bravo, J. B. Alonso, J. Morago, and C. Hernández, “Ontologies as backbone of cognitive systems engineering,” in *Proceedings of CAOS - Cognition And Ontologies Workshop at AISB 2017*, Bath, UK, 2017.
- [176] D. I. Spivak, *Category Theory for the Sciences*. The MIT Press, 2014.

- [177] T.-D. Bradley, “What is applied category theory?” *arXiv preprint arXiv:1809.05923*, 2018, doi: <https://doi.org/10.48550/arXiv.1809.05923>.
- [178] K. A. Lloyd, “Category theoretic foundations for systems science and engineering,” in *Handbook of Systems Sciences*, G. S. Metcalf, K. Kijima, and H. Deguchi, Eds. Singapore: Springer Singapore, 2021, pp. 1227–1249, doi: https://doi.org/10.1007/978-981-15-0720-5_65.
- [179] G. Bakirtzis, C. H. Fleming, and C. Vasilakopoulou, “Categorical semantics of cyber-physical systems theory,” *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 3, jul 2021, doi: <https://doi.org/10.1145/3461669>.
- [180] D. Kozen, C. Kreitz, and E. Richter, “Automating proofs in category theory,” in *Automated Reasoning*, U. Furbach and N. Shankar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 392–407.
- [181] F. Lawvere and S. Schanuel, *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, 1997.
- [182] S. M. Lane, *Categories for the Working Mathematician*, ser. Graduate Texts in Mathematics. Springer New York, NY, 1998.
- [183] S. Awodey, *Category Theory*, 2nd ed., ser. Oxford Logic Guides. Oxford: Oxford University Press, 2010.
- [184] M. Barr and C. Wells, *Category Theory for Computing Science*, ser. Prentice-Hall International series in Computer Science. Prentice Hall, 1990.
- [185] B. Fong and D. I. Spivak, *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Cambridge University Press, 2019.
- [186] G. Bakirtzis, “Compositional cyber-physical systems theory,” Ph.D. dissertation, School of Engineering and Applied Science, University of Virginia, April 2021, doi: <https://doi.org/10.18130/xn8v-5d89>.
- [187] P. Selinger, “A survey of graphical languages for monoidal categories,” in *New Structures for Physics*. Springer Berlin Heidelberg, 2010, pp. 289–355, doi: https://doi.org/10.1007%2F978-3-642-12821-9_4.
- [188] D. Yau, *Operads of Wiring Diagrams*. Springer International Publishing, 2018, doi: <https://doi.org/10.1007/978-3-319-95001-3>.
- [189] T. Leinster, *Higher Operads, Higher Categories*, ser. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004, doi: <https://doi.org/10.1017/CBO9780511525896>.
- [190] K. A. Lloyd, “A category-theoretic approach to agent-based modeling and simulation,” Watt Systems Technologies Inc., Tech. Rep., 2010.
- [191] N. Tsuchiya and H. Saigo, “A relational approach to consciousness: categories of level and contents of consciousness,” *Neuroscience of Consciousness*, vol. 2021, no. 2, 10 2021, doi: <https://doi.org/10.1093/nc/niab034>.

- [192] A. Stepanov and D. Rose, *From Mathematics to Generic Programming*. Pearson Education, 2014.
- [193] D. I. Spivak, “Ologs: A categorical framework for knowledge representation,” *PLoS ONE* 7(1): e24274, 2012, doi: <https://doi.org/10.1371/journal.pone.0024274>.
- [194] D. I. Spivak and R. Wisnesky, “Relational foundations for functorial data migration,” in *Proceedings of the 15th Symposium on Database Programming Languages*, ser. DBPL 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 21–28, doi: <https://doi.org/10.1145/2815072.2815075>.
- [195] E. Patterson, “Knowledge representation in bicategories of relations,” *ArXiv*, 2017, doi: <https://doi.org/10.48550/arXiv.1706.00526>.
- [196] A. Aguinaldo, E. Patterson, J. Fairbanks, and J. Ruiz, “A categorical representation language and computational system for knowledge-based planning,” *arXiv preprint arXiv:2305.17208*, 2023, doi: <https://doi.org/10.48550/arXiv.2305.17208>.
- [197] A. Censi, “A mathematical theory of co-design,” *ArXiv*, 2015, doi: <https://doi.org/10.48550/arXiv.1512.08055>.
- [198] G. Zardini, D. Milojevic, A. Censi, and E. Frazzoli, “Co-design of embodied intelligence: A structured approach,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 7536–7543, doi: <https://doi.org/10.1109/IROS51168.2021.9636513>.
- [199] S. Breiner, B. Pollard, E. Subrahmanian, and O. Marie-Rose, “Modeling hierarchical system with operads,” *Electronic Proceedings in Theoretical Computer Science*, vol. 323, pp. 72–83, sep 2020, doi: <https://doi.org/10.4204/2Feptcs.323.5>.
- [200] J. D. Foley, S. Breiner, E. Subrahmanian, and J. M. Dusel, “Operads for complex system design specification, analysis and synthesis,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 477, no. 2250, p. 20210099, 2021, doi: <https://doi.org/10.1098/rspa.2021.0099>.
- [201] C. Hernández, “Model-based self-awareness patterns for autonomy,” Ph.D. dissertation, Escuela Técnica Superior de Ingenieros Industriales - Universidad Politécnica de Madrid, 2013.
- [202] E. Aguado, R. Sanz, and C. Rossi, “Ontologies for run-time self-adaptation of mobile robotic systems,” in *XIX Conference of the Spanish Association for Artificial Intelligence*, 2021.
- [203] D. Bozhinoski, E. Aguado, M. G. Oviedo, C. Hernandez, R. Sanz, and A. Wąsowski, “A modeling tool for reconfigurable skills in ros,” in *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*, 2021, pp. 25–28, doi: <https://doi.org/10.1109/RoSE52553.2021.00011>.
- [204] E. Aguado and R. Sanz, “Using Ontologies in Autonomous Robots Engineering,” in *Robotics Software Design and Engineering [Working Title]*. IntechOpen, apr 2021, doi: 10.5772/intechopen.97357.

- [205] L. Lopes, N. Zajzon, B. Bodo, S. Henley, G. Zibret, and T. Dizdarevic, “UNEXMIN: developing an autonomous underwater explorer for flooded mines,” *Energy Procedia. European Geosciences Union General Assembly 2017, EGU Division Energy, Resources & Environment (ERE)*, vol. 125, pp. 41 – 49, 2017, doi: <https://doi.org/10.1016/j.egypro.2017.08.051>.
- [206] S. Latré, J. Famaey, and F. D. Turck, “A semantic context exchange process for the federated management of the future internet,” *International Journal of Network Management*, vol. 24, no. 1, pp. 1–27, 2014, doi: <https://doi.org/10.1002/nem.1840>.
- [207] B. Henderson-Sellers, “Bridging metamodels and ontologies in software engineering,” *Journal of Systems and Software*, vol. 84, no. 2, pp. 301–313, 2011, doi: <https://doi.org/10.1016/j.jss.2010.10.025>.
- [208] “IEEE Standard for Software and System Test Documentation,” *IEEE Std 829-2008*, pp. 1–150, 2008, doi: <https://doi.org/10.1109/IEEESTD.2008.4578383>.
- [209] IEEE Std 1220-2005, “IEEE Standard for Application and Management of the Systems Engineering Process,” *IEEE Std 1220-2005 (Revision of IEEE Std 1220-1998)*, pp. 1–96, 2005, doi: <https://doi.org/10.1109/IEEESTD.2005.96469>.
- [210] G. J. Roedler and C. Jones, “Technical Measurement. A Collaborative Project of PSM, INCOSE, and Industry,” Practical Software & Systems Management (PSM)/International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-020-01, Tech. Rep., 2005.
- [211] N. Johnson and D. Yau, *2-Dimensional Categories*. Oxford University Press, 2021.
- [212] S. Macenski, F. Martín, R. White, and J. G. Clavero, “The marathon 2: A navigation system,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2718–2725, doi: <https://doi.org/10.1109/IROS45743.2020.9341207>.
- [213] D. Fox, “Kld-sampling: Adaptive particle filters,” in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14. MIT Press, 2001.
- [214] K. Konolige, “A gradient method for realtime robot control,” in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 1, 2000, pp. 639–646 vol.1, doi: <https://doi.org/10.1109/IROS.2000.894676>.
- [215] R. Sanz, “Envisioning conscious controllers,” 2004, keynote Speech at the IFAC/IFIP Workshop on Real-Time Programming (WRTP04).
- [216] K. Dentler, R. Cornet, A. ten Teije, and N. de Keizer, “Comparison of reasoners for large ontologies in the owl 2 el profile,” *Semant. Web*, vol. 2, no. 2, p. 71–87, apr 2011, doi: <https://doi.org/10.3233/SW-2011-0034>.
- [217] V. Gomez, M. Hernando, E. Aguado, R. Sanz, and C. Rossi, “Robominer: Development of a highly configurable and modular scaled-down prototype of a mining robot,” *Machines*, vol. 11, no. 8, 2023, doi: <https://doi.org/10.3390/machines11080809>.

- [218] E. Aguado, R. Sanz, and C. Rossi, “Self-awareness for robust miner robot autonomy,” in *EGU General Assembly*, 23–27 May 2022, pp. EGU22–2716, doi: <https://doi.org/10.5194/egusphere-egu22-2716,2022>.
- [219] V. Gomez, M. Hernando, E. Aguado, D. Bajo, and C. Rossi, “Design and kinematic modeling of a soft continuum telescopic arm for the self-assembly mechanism of a modular robot,” *Soft Robotics*, vol. 0, no. PMID: 37878327, p. null, 0, doi: <https://doi.org/10.1089/soro.2023.0020>.
- [220] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007, doi: <https://doi.org/10.1109/TRO.2006.889486>.
- [221] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI*. CRC Press, jul 2018, doi: <https://doi.org/10.1201/9780429489105>.
- [222] R. A. Suarez Fernandez, D. Grande, A. Martins, L. Bascetta, S. Dominguez, and C. Rossi, “Modeling and control of underwater mine explorer robot UX-1,” *IEEE Access*, vol. 7, pp. 39 432–39 447, 2019, doi: <https://doi.org/10.1109/ACCESS.2019.2907193>.
- [223] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2149–2154 vol.3, doi: <https://doi.org/10.1109/IROS.2004.1389727>.

List of Figures

1.1	Methodology phases of the iterative process followed in this thesis, main milestones to achieve.	6
2.1	An engineered system and its environment. The system comprises a device and some external sensors in the environment. The device is further decomposed into software and hardware; the hardware includes actuators and sensors for both the device and its surroundings. The external sensors are used by the device's software.	11
2.2	ALFUS model in which each axis corresponds to a metric value for mission complexity, environmental complexity, and human intervention (inverse).	13
2.3	Architecture of a fault-tolerant control system, adapted from [27].	27
2.4	Garlan's et al. architectural adaptation framework, adapted from [34]. The monitor mechanisms on the executing system trigger the architecture manager, which decides whether an adaptation to the system's execution structure is required.	29
2.5	Autonomic element composed of a managed element entity and an automatic manager which executes the MAPE-K loop, adapted from [40].	31
3.1	Hierarchical decomposition of a System-of-Interest composed of several systems and atomic elements, adapted from [59].	44
3.2	Example of a life cycle progression, adapted from [72].	47
4.1	Relationship between the representation and world dimensions. Sentences represent some aspects of the world which can be derived to extract new facts through entailment. This entailment should actually follow from the aspects in the real world.	56
4.2	Example of a symbolic representation for a robotic task. The provided information includes the robot initial pose, the considered safe distance from a wall, and the designated next waypoint for the robot. These details are derived using predefined rules to compute the recommended speed. In the robot world, a controller can leverage this knowledge to move the robot base.	56
4.3	Example of the Apple ontology.	65
5.1	Conceptualization of an autonomous agent interacting with the environment and upon itself.	72
5.2	PRISMA flow diagram, adapted from [132].	74
6.1	Passengers and seats category.	117
6.2	Functors for the passengers and seats category.	118
6.3	Functors and natural transformation for the passengers and seats category.	120
6.4	Example of a wiring diagram representing a robot in a navigation task. It has distance to objects and available battery as resources and velocity command as product.	121
6.5	Example of an operad composed of a f cell with 5 ports, a g cell with 4 ports, a h cell with 4 ports, a k cell with 1 ports. They form an external cell with 4 ports.	122
6.6	Pushout in a Robot category.	124

7.1	Overall metacontrol workflow.	131
7.2	Main elements of TOMASys metamodel, the * symbol represents the possible multiplicity in the destination relationship.	133
7.3	The UX-1 prototype during operation in Kaatjala Mine (Finland). Image credits: UNEXMIN project.	136
7.4	Thrusters allocation, the green arrows represent the thrusters implicated on heave movement (T_1 , T_3 , T_4 and T_7) and the blue ones, the thrusters responsible of forward movement (T_0 , T_2 , T_4 and T_6).	136
7.5	TIAGo robot in a university setup. Image credits: MROS project.	138
7.6	Reasoning time in (ms) according to the number of individuals in the ontology. Also shown its corresponding exponential trendline ($y = 681e^{0.0002x}$ with R^2 of 0.99).	140
7.7	Navigation 2 engineering knowledge represented in the SysSelf category.	148
7.8	Equivalence in terms of component functionality.	150
7.9	SysSelf metamodel, main elements and relationships.	151
8.1	Architecture of a system using a metacontroller to face contingencies, inspired from [27].	153
8.2	Main classes and relationships of the SysSelf ontology.	155
8.3	Classes and relationships of the SysSelf ontology related with metrics and stakeholders.	156
8.4	Additional classes and relationships of the SysSelf ontology related with constraints, status and interfaces.	156
8.5	ABox example: Application model using the SysSelf metamodel.	158
8.6	Activity diagram for the metacontroller execution.	160
8.7	Component individuals and morphisms to find equivalences.	162
8.8	Implicit morphism emergence by tracing morphisms and functors between categories.	163
8.9	Metric propagation to determine how changes in components are expected to affect value MOEs.	164
8.10	Activity diagram of the metacontroller and the observer nodes. Note that ROS 2 nodes are cyclic and they finish with an interruption. The activity “find adaptation mechanism” depicted in gray, corresponds to the main metacontroller execution as explained in Section 8.2.2 and depicted in Figure 8.6.	165
8.11	ROS 2 wrapper for metacontroller, composed of a Python class and two ROS 2 nodes, an observer and the metacontroller itself.	166
8.12	Processes required to use the SysSelf framework. The systems engineers shall provide the information necessary to (i) produce a system model, (ii) monitor the system and (iii) execute the pertinent adaptation actions. Dotted lines corresponds to the software provided by the framework. The main constituents—the metamodel and the metacontroller—are highlighted in gray, while those specific to the ROS 2 implementation—the observer and adaptation executor—are highlighted in blue. In scenarios where the ROS 2 infrastructure is not utilized, developers must provide the metacontroller with information regarding the entity affected by a contingency and subsequently execute the reconfiguration action determined by the metacontroller.	167
9.1	Full-scale prototype of the miner robot during field tests in the open pit, active, oil shale mine in Kunda, Estonia. Image credits: ROBOMINERS project.	170

9.2	Scaled-down prototype of the miner robot composed of highly reconfigurable modular platforms, adapted from [217].	170
9.3	Robotic module description with the docking ports to manually attach the locomotion submodules and self-assembly ports to connect additional sensors or modules, adapted from [217].	171
9.4	Miner robot module configuration examples including different locomotive systems, using archimedean screws or continuous track wheels, and possible configurations combining sensing or cutting headers or other locomotive modules. . .	172
9.5	Component decomposition of the miner robot concept. Components in blue are involved in Scenario 1, while components in purple are associated with Scenario 2. Additional components of the miner robot concept are shown in gray. Solid lines represent structural connections on the robot from the beginning of the robot's task, while dashed lines represent dynamic links that may be created during robot operation.	172
9.6	Navigation 2 software architecture. ROS topics start with a “\” symbol. The input elements are depicted in purple: goal input and map server. The output is the robot system depicted in dark blue. The main element is the Nav2 subsystem, depicted in gray, which is further decomposed into the light blue boxes: localization, path planner, and controller.	174
9.7	Setup for the failure in critical sensor scenario for the miner robot.	175
9.8	Nominal ontological model providing a specification of the LiDAR component, incorporating performance metrics, interfaces, as well as involved capabilities, values, and goals. Other components such as camera and interface adaptor are not used at this moment. Note that not all relationships are visualized for improved readability. Active elements are denoted in blue, whereas inactive ones are presented in gray; in bold, main relationships.	176
9.9	Equivalence between LiDAR and camera sensor through point cloud to laser scan interface adaptor.	176
9.10	Comparison of a map generated with different sensors.	177
9.11	Information displayed in the terminal of the metacontroller node for the failure in critical sensor scenario, colored text represent the specific application elements on the model from the SysSelf metamodel.	178
9.12	Ontological model post-adaptation, showcasing the substitution of the LiDAR with the camera and interface adaptor. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue, while inactive ones are shown in gray; affected metrics are represented by purple arrows; in bold, main relationships.	179
9.13	Graphic visualization of relevant nodes and topics for the adaptation running on the system. In red, highlighted the the scan topic coming from the LiDAR and the observer and metacontroller nodes.	180
9.14	Graphic visualization of nodes and topics running on the system involved after adaption. In green, highlighted the nodes and topics affected. Now the scan message comes from the camera after a processing in the point cloud to laser scan node. The scan topic is still monitored by the observer.	181
9.15	Setup for the underachieved capability scenario. On the left, a robotic module equipped with a cutting head that is used for selective mining and, on the right, a spare robotic module.	182

9.16	Nominal ontological model providing a specification for the robominer mining module and the extract capability it realizes, along with the corresponding goal and value. Note that not all relationships are visualized for improved readability. Active elements are denoted in blue, whereas inactive elements are presented in gray; in bold, main relationships.	183
9.17	Support capability required to solve the lack of force contingency, requires to connect a robominer module through an interface adaptor.	184
9.18	Ontological model post-adaptation, showcasing the use of an additional robotic module to increase the system support. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue; affected metrics are represented by purple arrows; in bold, main relationships.	184
9.19	Information displayed in the terminal of the metacontroller node for the lack of force scenario, colored text represent the specific application elements on the model from the SysSelf metamodel.	185
9.20	Setup after adaptation. The robotic module equipped with a cutting head has connected a spare robotic module to have more force. Now it can continue the mining task.	186
9.21	Setup for the mission unachievable scenario. There is a robotic module performing selective mining when the robot realizes that it cannot extract the desired amount of mineral from that deposit.	186
9.22	Nominal ontological model in the case of the extracting mineral goal scenario. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue; inactive, in gray; in bold, main relationships. . . .	187
9.23	Ontological model post-adaptation, it shows the change of goal to continue the operation. Note that not all relationships are illustrated for enhanced readability. Active elements are depicted in blue, inactive in gray; affected metrics are represented by purple arrows; in bold, main relationships.	188
9.24	Information displayed in the terminal from the observer and metacontroller nodes for the unachievable quantity of mineral extraction, colored text represent the specific application elements on the model from the SysSelf metamodel.	189
9.25	Mobile robot testbed Pioneer 2-AT8 with its main sensors, LiDAR and RGB-D camera, mounted.	190
9.26	Breakdown of the main components involved in the mobile robot tests. The main computer is represented in dark blue, while the remaining components are illustrated in light blue.	191
9.27	Setup for the failure in critical sensor scenario for the mobile robot.	191
9.28	Graphic visualization of nodes and topics running on the system involved before and after adaption. In red, the laser node that was substituted, in green the point cloud to laser scan transformer that continues providing scan messages.	192
9.29	Nominal ontological model for the navigation task in the unsolvable capability error scenario. Note that not all relationships are visualized for improved readability. Active elements are denoted in blue, whereas inactive ones are presented in gray; in bold, main relationships.	193
9.30	Information displayed in the terminal from the observer and metacontroller nodes for the communication failure, colored text represent the specific application elements on the model from the SysSelf metamodel.	194
B.1	Force allocation in thrusters T_0 , T_2 , T_4 , T_6 when surging.	211

B.2	Odometry measurements during the thruster failure experiments. Yellow squares depict the reference waypoints.	214
B.3	Force reference commands obtained by the FL controller. Uppermost figure represent the I test, middle the II and bottom the III . The time instance when each waypoint is sent is depicted with a red circle, and the moment of the thruster failure is depicted with a vertical black line.	216
B.4	Forces produced by thrusters in charge for <i>surge</i> movement in all three experiments. Purple graphs represent the I test, green the II and blue the III . The time instance when each waypoint is sent is depicted with a red circle, and the moment of the thruster failure is depicted with a vertical black line.	217
B.5	Forces produced by thrusters in charge for <i>heave</i> movement in all three experiments. Purple graphs represent the I test, green the II and blue the III . The time instance when each waypoint is sent is depicted with a red circle, and the moment of the thruster failure is depicted with a vertical black line.	218

List of Tables

4.1	Truth tables for the five logical connectives.	59
5.1	Use of ontologies in the manipulation domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).	84
5.2	Part 1: Use of ontologies in the navigation domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).	95
5.3	Part 2: Use of ontologies in the navigation domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).	96
5.4	Use of ontologies in the social domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).	103
5.5	Use of ontologies in the industrial domain for perception and categorization (P/C), decision-making and planning (DM/P), prediction and monitoring (P/M), reasoning (R), execution (E), communication and coordination (C/C), interaction and design (I/D) and learning (L).	108
5.6	Summary of other aspects of the framework: concrete encoding languages used, incorporation of temporal conceptualizations, and whether the framework is built upon other works or utilizes an upper-level ontology.	112
7.1	Semantic Web Rule Language rules for TOMASys implementation. The first one sets the <code>Function Grounding</code> in error if it uses a faulty <code>Component</code> , the second one sets the <code>Objective</code> in error if the <code>Function Grounding</code> is in error, and the third one marks as unreachable the <code>Function Designs</code> that require unavailable <code>Components</code>	134
7.2	UX-1 optimal thrusters used according to the movement direction, if one thruster is disabled the system must adapt to use the remaining thrusters in the column.	137
7.3	TOMASys ontology models of different sizes used to evaluate runtime reasoning.	140
7.4	Main concepts of SysSelf metamodel and relationships between them.	143
9.1	Average Maintenance Time (AMT) in the evaluation scenarios after 50 iterations and standard deviation. Scenarios have been evaluated that execute (i) meta-control (MC) and (ii) metacontrol and navigation (MC and Nav) in the same processor to measure the impact of other tasks.	196
B.1	Comparison between tests. I represent the test with all thrusters working. II represent the test with the failure of T_0 . III represent the test with the failure of T_1	214
B.2	Mean force summary.	215

B.3 Numerical comparison of the experiments performed with and without the meta-controller (MC). **II** represent the test with the failure of T_0 . **III** represent the test with the failure of T_1 215

Acronyms

- ACL** Autonomous Control Levels. 12
- AFRL** Air Force Research Laboratory. 12
- AI** Artificial Intelligence. 3, 53, 63
- ALFUS** Autonomy Levels for Unmanned Systems. 13
- AMCL** Adaptive Monte Carlo Localization. 173
- APIs** Application Programming Interfaces. 66
- AuR** Autonomous Robot. 71
- BREP** Boundary REPresentation. 102
- BT** Behavior Tree. 173, 194
- CAD** Computer-Aided Design. 78
- CORA** Core Ontology for Robotics and Automation. 70, 80
- CRAM** Cognitive Robot Abstract Machine. 78, 109, 113
- CT** Category Theory. 115, 132, 142, 144, 154
- DL** Description Logics. 61, 65, 126, 133
- DOLCE** Descriptive Ontology for Linguistic and Cognitive Engineering. 70, 105
- DUL** DOLCE+DnS Ultralite Ontology. 70, 99
- FailRecOnt** Failure Interpretation and Recovery in Planning and Execution Ontology-Based Framework. 80
- FMs** Formal Methods. 34
- FOL** First Order Logic. 59
- FOND** Fully Observable Non-Deterministic Planning. 63
- GNC** Guidance, Navigation, and Control. 83
- GST** General Systems Theory. 9, 199
- IEEE ORA** Ontologies for Robotics and Automation. 70
- INCOSE** International Council on Systems Engineering. 10, 39, 50

- ISRO** Intelligent Service Robot Ontology. 99
- KB** Knowledge Base. 53, 55, 61, 78, 134, 159
- KerML** Kernel Modeling Language. 50
- KR&R** Knowledge Representation and Reasoning. 53, 63, 132
- MAPE-K** Monitor, Analyze, Plan, Execute sharing a Knowledge base. 31, 85, 134
- MBSE** Model-Based Systems Engineering. 49, 115, 116, 132, 141
- MEBN** Multi-Entity Bayesian Network. 86, 109
- MOE** Measure of Effectiveness. 143, 150, 155
- MOP** Measure of Performance. 143, 150, 155
- MROS** Metacontrol for Robot Operating System. 137
- NCAP** Non-Contextual Autonomy Potential. 14
- NEEMs** Narratively-Enabled Episodic Memories. 72, 78
- OASys** Ontology for Autonomous Systems. 71, 83
- OCRA** Ontology for Collaborative Robotics and Adaptation. 99
- OMG** Object Management Group. 50
- OMRKF** Ontology-based Multi-layered Robot Knowledge Framework. 85, 94
- OODA** Observe, Orient, Decide and Act. 12
- OpsCon** Operational Concept. 48
- ORO** OpenRobots Common Sense Ontology. 98
- OUR-K** Ontology-based Unified Robot Knowledge. 94
- OWL** Ontology Web Language. 65, 66, 133, 140, 154
- PDDL** Planning Domain Definition Language. 63
- PL** Propositional Logic. 58, 59
- PLM** Probabilistic Logic Module. 81
- PMK** Perception and Manipulation Knowledge. 79
- PPDDL** Probabilistic PDDL. 63
- PPOS** Planning with Partial Observability and Sensing. 63
- PRISMA** Preferred Reporting Items for Systematic reviews and Meta-Analyses. 74, 237

- Prolog** PROgramming in LOGic. 62
- QA** Quality Attribute. 132
- RAS** Robotics and Automation Society. 70
- RDF** Resource Description Framework. 66
- ROS** Robot Operating System. 62, 64, 164, 211
- ROSETTA** ROBot control for Skilled ExecuTion of Tasks. 102
- RTPO** Robot Task Planning Ontology. 88
- SAR** Search and Rescue. 92
- SE** Systems Engineering. 24, 25, 39, 115, 127, 142, 146
- SEBoK** Systems Engineering Body of Knowledge. 40
- SLAM** Simultaneous Localization and Mapping. 74, 92, 173
- SLD** Selection rule driven Linear resolution for Definite clauses. 59
- SMC** Symmetric Monoidal Categories. 120
- SNAME** Society of Naval Architects and Marine Engineers. 135
- SoI** System-of-Interest. 44, 45
- SOMA** Socio-physical Model of Activities. 71
- SoS** Systems-of-Systems. 44, 115
- STRIPS** Stanford Research Institute Problem Solver. 101
- SUMO** Suggested Upper Merged Ontology. 69, 80
- SUO-KIF** Standard Upper Ontology Knowledge Interchange Format. 69
- SWARMS** Smart and Networked Underwater Robots in Cooperation Meshes. 86
- SWRL** Semantic Web Rule Language. 65, 134, 140, 159
- SysML** Systems Modeling Language. 50, 115, 121, 127
- TAMP** Task and Motion Planning. 79
- TMR** Triple Modular Redundancy. 24
- TOMASys** Teleological and Ontological Model for Autonomous Systems. 83, 132, 202
- TPM** Technical Performance Metric. 143, 150, 155
- UAV** Unmanned Aerial Vehicle. 12
- UML** Unified Modeling Language. 50
- USAR** Urban Search and Rescue. 90

Glossary

A

agent Any entity, physical or virtual, that act. 17

autonomy Quality of a system which provides the ability to pursue objectives independently, free from human intervention. These objectives can either be commanded or fixated by the system itself to provide further value. 3, 11

availability Probability that a system or system element is operational at a given point in time, under a given set of environmental conditions. 15

B

boundary Line of demarcation between the system under consideration and its environment, including users, operators, enabling systems, and other surrounding elements. 45

C

Category Theory General mathematical theory for describing structures and maintaining control over which aspects are preserved when performing abstractions. 115

cognitive system System based on knowledge. 17

D

dependability Ability to deliver the system intended services with a certain level of assurance, encompassing factors like availability, reliability, safety, and security. 16

E

emergence Manifestation of properties attributed to the whole system that individual elements, in isolation, do not possess. 45

engineered system A system designed or adapted to interact with an anticipated operational environment to achieve one or more intended purposes while complying with applicable constraints. 10

error Property of the system state which may lead to failure. 24

F

failure Deviation of the system behavior from its specification. 24

fault Defect with the potential to produce an error. Faults cause changes in component and system characteristics, which produce undesired modes of operation or performance outcomes. 24

fault tolerance Engineering domain dedicated to develop methodologies for prevent, mitigate and recover from faults. 24

formal methods Engineering methodology for specifying, developing, and verifying software systems through the use of a mathematically grounded notation and language. It constitutes a tool to systematically and, in many cases, automatically assess the software model for three critical qualities: consistency (by eliminating ambiguity), completeness, and correctness. 34

H

hierarchy Definition of the composing elements of a system only in relation to higher order elements while suppressing all interaction and interrelations with other parts. 44

K

knowledge graph Representation method for data as a graph, utilizing a set of nodes to represent entities and edges to capture relations between these entities. 66

Knowledge Representation and Reasoning Sub-area of artificial intelligence that concerns analyzing, designing and implementing ways of representing information in computational formats so agents can use it to derive implicit facts. 53

L

life cycle Evolution of a system, product, service, project or other human-made entity from conception through retirement. 47

life cycle model Framework of processes and activities concerned with the life cycle. 47

M

maintainability Probability that a system or system element can be repaired in a defined environment with defined resources within a specified period of time. Increased maintainability implies shorter repair times. 16

O

ontology Abstract, simplified conceptual perspective of a specific area of interest. It describes a shared vocabulary at both textual and conceptual levels, including concepts' properties and the relations between them. 64

R

reasoning Process of manipulating symbols and sentences to deduce new sentences in a given world. 55

reliability Ability of a system or an element to perform its required functions under stated conditions without failure for a given period of time. 15

resilience Ability to maintain capability in the face of adversity either by avoiding the cause of stress, withstand this degradation or recovering from it. 3, 16

robot Physical agents endowed with sensors and physical effectors that perform tasks by manipulating objects in the world. 17

robustness Ability to resist degradation of capabilities under adverse conditions by avoiding or withstanding the cause of stress. 16

S

safety Property of a system of not causing damage as it behaves when being used and sustained in a specific way in a specific environment. 16

security Ability to protect against non-authorized access. A composite of four attributes—confidentiality, integrity, availability, and accountability—plus aspects of a fifth, usability, all of which have the related issue of their assurance. 16

self-adaptation Capability of a system to dynamically adjust its structure and behavior during runtime in response to its perception of both the environment and its internal state.. 27

self-awareness Capability that supports understanding and decision-making towards a value. It usually implies having information about its own internal states, being able to differentiate itself from external world information and having sufficient knowledge to perceive how its components influences other parts of the system. 33

self-X Capability of a system to manage itself to complete its tasks according to an administrator's goals. 32

stage Period within the life cycle of an entity that relates to the state of its description or realization. 47

system A "whole" consisting of interacting "parts" in which the arrangement of these parts gives rise to properties or behaviors that are not attainable individually. 9, 10

Systems Engineering A transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods. 39

T

traceability Capability to establish associations between system elements and concepts such as needs, requirements, architectural elements, and verification artifacts. 45

V

value Amount of benefit, feature or capability provided to various stakeholders. It considers the optimum set of requirements to deliver customer satisfaction. 17, 26

Y

Yoneda lemma Category theory result which prove that two objects in a category can be considered equivalent if and only if all the relationships that one of them holds with others in the category are the same as those of the other object. 126, 149, 161



Title **Systems that know what they are doing**
Subtitle A Model-Based Formal Specification for Robust Autonomy
Author Esther Aguado
Date July 2024
Reference ASLAB-R-2024-007
Release 1.0 Final
URL <http://www.aslab.upm.es/doc/controlled/ASLAB-R-2024-007.pdf>
© 2024 Autonomous Systems Laboratory
aslab.upm.es