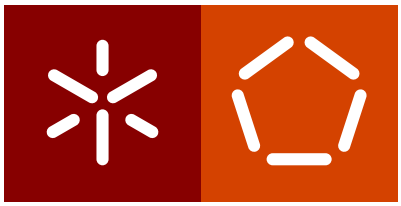
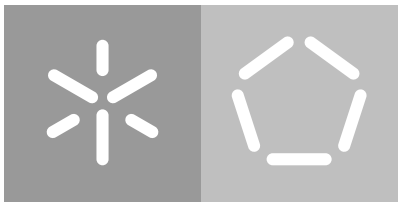


University of Minho
Engineering School
Department of Industrial Electronics

Ricardo Jorge Magalhães Teixeira

Ontology-Driven Metamodeling Towards Intelligent Motion Control System Design Automation

October 2017



University of Minho
Engineering School
Department of Industrial Electronics

Ricardo Jorge Magalhães Teixeira

Ontology-Driven Metamodeling Towards Intelligent Motion Control System Design Automation

Master dissertation
Master Degree in Industrial Electronics and Computers Engineering

Dissertation supervised by
PhD Professor Adriano Tavares (University of Minho)
Professor Quangang Wen (Zhuhai College of JLU)

October 2017

ACKNOWLEDGMENTS

This Masters Dissertation could not have been completed without the support and help of many people. First and foremost, I would like to thank my family for their patience and enduring faith over the challenging and strenuous years of my studies and, in particular, this year of my Masters.

Naturally, my supervisor Professor Adriano Tavares had the most influence on the results and progress of my work and my development as a professional. He had the original idea for the interdependent topic, and contributed long hours of discussions about the direction of my research. I particularly would like to thank him for the unfailing trust in me and my abilities, and for the vast encouraging support on different ways. Obviously, he also had the greatest part in proofreading of my Masters Dissertation, which he did with great precision and diligence.

I am deeply indebted to my academic co-advisor, Professor Quangang Wen, who made it possible to develop this Masters Dissertation in cooperation with an industrial research institute, the Jilin University. Professor Wen, Professor Liang and the remaining Intelligent Motion Control Systems' Platform (IMCSP) development team, did not only endorse the topic, but showed an incredible interest in my work and actively encouraged me to pursue my research. For that, for the exceptional reception and for the fulfilling ERASMUS+ period in China, thanks.

My friends and colleagues at the University of Minho, Jilin University and Zhuhai College of JLU, played a major role in making this time enjoyable and rewarding. Miguel Abreu, José Martins, José Lopes, João Alves, Pedro Pereira, David Almeida, Miguel Macedo and all the Embedded Systems Research Group (ESRG) team deserve a special mention for providing unique perspectives to the work developed in the Masters Dissertation, and amazing interdependent collaborative groups during the last years.

To Juliana Meireles and to my lasting group of friends from SB, thanks for being always present and for all the fruitful discussions and motivating moments.

Last but not least, I wish to express my warmest thank you to my girlfriend, Beatriz Meneses, for all her love, abiding patience and magnificent support.

RESUMO

A evolução dos Sistemas Inteligentes de Controlo de Movimento (SICM) desafia a sua manutenção, capacidade de atualização, variabilidade e automação. Desta forma, é imperativo que o seu processo de desenvolvimento seja complementado com uma perspectiva abstrata, que permita reduzir a complexidade da configuração e implementação do sistema, através de paradigmas, ferramentas de software ou linguagens de programação. De acordo com um dos mais prominientes paradigmas de *software*, Model-Driven Software Development (MDSD), modelar sistemas complexos requere o uso de diversas linguagens específicas de domínio para modelação, resultando em múltiplos modelos de domínio específico, o que aumenta o custo de desenvolvimento do sistema e favorece a ambiguidade de modelos. Esta Dissertação de Mestrado explora uma abordagem à modelação de SICM enriquecida por tecnologias semânticas, utilizando a Semantically-enriched Modeling Language (SeML) para automatizar o processo de configuração e implementação destes sistemas através da geração de código de forma generativa. Para tal, esta Dissertação introduz a SeML e explora exaustivamente o desenvolvimento de uma ontologia prescritiva para o domínio dos SICM. Finalmente, através de um caso de estudo, é demonstrada a viabilidade da abordagem proposta através da expansão da gramática SeML com o conhecimento semântico contido na ontologia desenvolvida. Além disso, a viabilidade da mesma foi complementada com a modelação de um SICM, o sistema de produção de bobinas desenvolvido na Universidade de Jilin, com o objetivo de automatizar a configuração e implementação deste sistema e validar a SeML ao nível do domínio.

ABSTRACT

The evolution of Intelligent Motion Control Systems (IMCSs) challenges its maintenance, *upgradability*, variability and automation. Concerning this, it's imperative to complement its development with an abstract perspective to reduce the complexity of its configuration and implementation, through paradigms, tools or programming languages. As one of the most prominent paradigms advocates, MDSD, modeling complex systems usually requires several different Domain-specific Modeling Languages (DSMLs), resulting in diverse domain-specific models, and thus augmenting systems' development costs and ensuing ambiguous models. This Masters Dissertation presents an approach to IMCSs' modeling process enriched by semantic technologies, using the SeML to automate systems' customization and implementation with generative code generation. To this end, this Dissertation introduces the SeML and exhaustively explores the development of a prescriptive ontology for the IMCSs' domain. The feasibility of this approach is demonstrated by extending the SeML's grammar with semantic knowledge contained in the developed ontology, conducive to the creation of an IMCS's model, a coil winding machine developed at Jilin University, aiming at automate its customization and implementation, and validate the SeML at a domain-level.

CONTENTS

ABBREVIATIONS

AIM Agent Information Manager.
ALFUS Autonomy Levels for Unmanned Systems.
API Application Programming Interface.
AST Abstract Syntax Tree.

BWW Bunge-Wand-Weber.

CFo Common Framework object.
CORA Core Ontology for Robotics and Automation.
CORAX Core Ontology for Robotics and Automation X.
COTS Commercial Off-The-Shelf.

DDA Digital Differential Analyser.
DIOD Device Input Output Driver.
DL Description Logics.
DoD Device object Driver.
DOLCE Descriptive Ontology for Linguistic and Cognitive Engineering.
DSL Domain-specific Language.
DSML Domain-specific Modeling Language.
DUL Dolce UltraLite.

ERM Entity-Relationship Model.
ESRG Embedded Systems Research Group.
ESRG-OT Embedded Systems Research Group Ontology Team.

FP6 European Union's Sixth Framework Programme.
FPGA Field-Programmable Gate Array.
FSMC Flexible Static Memory Controller.

GFO General Formal Ontology.
GOL General Ontological Language.
GPL General-purpose Language.

HMI Human Machine Interface.

IDE Integrated Development Environment.
IEEE Institute of Electrical and Electronics Engineers.

IMCS Intelligent Motion Control System.
IMCSP Intelligent Motion Control Systems' Platform.
IoS Internet of Services.
IoT Internet of Things.
IRIs International Resource Identifier.
IT Information Technology.

KIF Knowledge Representation Framework.
KnowRob Knowledge Processing for Autonomous Personal Robots.
KOS Knowledge Organization Systems.

LUT Look-Up-Table.

MDA Model-Driven Architecture.
MDE Model-Driven Engineering.
MDSD Model-Driven Software Development.
MLCOF Multi-layered Context Ontology Framework.
MTBF Mean Time Between Failures.

NIST National Institute of Standards and Technology.

OCOA Ontology based Component Oriented Architecture.
ODP Ontology Design Pattern.
ODSD Ontology-Driven Software Development.
OGC Open Geospatial Consortium.
OMG Object Management Group.
OMRKF Ontology-Based Multi-Layered Robot Knowledge Framework.
OpenCyc Open Source Cyc technology.
ORAWG Ontologies for Robotics and Automation Working Group.
ORO OpenRobots Common Sense Ontology.
OWL Web Ontology Language.
OWL-DL Web Ontology Language Description Logics.
OWL-Full Web Ontology Language Full.
OWL-Lite Web Ontology Language Lite.
OWL2 Web Ontology Language 2.

PBP Point-By-Point.
POS POSition Ontology.
PROTEUS Platform for RObotic Modeling and Transformation for End-Users Scientific communities.

RDF Resource Description Framework.

RDFS Resource Description Framework Schema.
RoSta Robotic Standards and Reference Architectures.
RPARTS Robot Parts Ontology.
RuleML Rule Markup Language.

SAN Semantic Actuator Network.
SeML Semantically-enriched Modeling Language.
SeSE Semantically-enriched Software Engineering.
SICM Sistema Inteligente de Controlo de Movimento.
SRAM Static Random Access Memory.
SSN Semantic Sensor Network.
SSN-XG Semantic Sensor Network Incubator Group.
SSO Stimulus-Sensor-Observation.
SUMO Suggested Upper Merged Ontology.
SUOWG Standard Upper Ontology Working Group.
SWRL Semantic Web Rule Language.

UAV Unmanned Aerial Vehicle.
UFO Unified Foundational Ontology.
UML Unified Modeling Language.
UNA Unique Name Assumption.
URI Uniform Resource Identifier.

W3C World Wide Web Consortium.
WSML Web Service Modeling Language.

XML Extensible Markup Language.

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1

INTRODUCTION

1.1 MOTIVATION

For decades, software systems have become more and more *complex*, and control systems, as part of software systems, are becoming more complex too. As a sub-domain of control systems, intelligent motion control systems contain hundreds or thousands of sub-systems, each consisting of incalculable entities and entities' relations, designed to ensure feasible and rational motion in the neighbourhood of kinematic and dynamic singularities. With the ever-developing hardware technologies and ensuing increasing computation power of today's [Wir95], customer demands and expectations towards software continue to rise as well. Additionally, as a result of platform technologies' metamorphosis and due to business requirements, the developed software is likely to become outdated much faster than previously. Therefore, there is a growth in the need to manage the systems' complexity and the desire to integrate the developed software/hardware systems with a tool/framework that, besides efficiently and effortlessly enhances systems' variability, allows the automation of systems' configuration and implementation.

In pursuance of reducing software complexity, and consequently reduce the software maintenance cost [Kos03, BDZ89], the software engineering industry has developed frameworks, tools, programming languages and paradigms, focused on describing software systems through an abstract perspective. In particular, MDSD advocates the use of diverse DSMLs to describe software systems [BL07], increasing systems' *development speed* through its automation, providing a correct *separation of concerns* [Ram03] (which will grant, among other things, better maintainability of software systems through redundancy avoidance and manageability of technological changes) and enabling the existence of a *productive environment* in technology, engineering, and management fields.

In development scenarios that require different DSMLs, the development of multiple domain-specific models does not suffice to ensure the consistency and validity between them, because the semantic links between constructs from different modeling languages remain unspecified, and therefore:

- The modeling productivity decreases and the precise definition of concerns within a particular domain is hampered;
- The semantic rigour in partial systems' specifications decreases, which often causes inconsistencies and semantic incompatibilities between the individual viewpoints;
- The generation of the code on a particular platform is usually precluded due to insufficient information about the system.

In conjunction with these disadvantages, the MDSD approach has a major disadvantage - the development cost of DSMLs will be as big as the number of unavoidable domain-specific models. Although, this approach provides a crucial advantage to the DSMLs' developers, which is the fact that the information about the meaning of a model's element in relation to one another is usually hidden [BL07], but available. Merging the advantages and the disadvantages of the MDSD approach, it's easily identifiable that only one of the Guizzardi's modeling language properties [Gui13] is satisfied: the *domain appropriateness*, which refers to truthfulness of the language to the domain; the other property, domain *comprehensibility appropriateness*, which refers to the pragmatic efficiency of the language to support communication, understanding and reasoning in the domain, is not fulfilled because the resulting domain-specific models are not semantically connected.

A solution to this core problem resides in semantics. Semantics apply "techniques that support and exploit semantic information (as opposed to syntax and structure) to enhance information systems" [SR03]. A formal technique to define DSLs' semantics is through the use of ontologies to describe domain knowledge. The increasing interest on ontologies, the introduction of the Semantic Web framework and the development of its own language, Web Ontology Language (OWL) (by the World Wide Web Consortium (W3C)) were the key enablers to a new paradigm: ODSD, which will have an significant role in this Dissertation. The combination of ontologies and DSMLs for complex software systems' modeling, and consequently combination of the ODSD and MDSD paradigms, enabled the development of a *unique* DSML with *shapeshifting* capability, the SeML, which allows systems' designers to model systems from different domains (or different applications within the same domain) with a single DSML.

Advocating the opinion that the ERASMUS+ programme, namely the KA107 International Credit Mobility, is an excellent opportunity to develop my engineering, technological and social skills, the IMCSP developed at Jilin University (integrated in the Chinese government's "Made in China 2025" initiative to comprehensively upgrade Chinese industry) was the chosen application (and domain) to the validation of the SeML.

Therefore, in its first phase, this Masters Dissertation explores the development of the SeML, beginning with its ontology for software/hardware systems. Subsequently, and as an instance of the referred ontology, a domain ontology for the intelligent motion control

systems' domain is developed. Lastly, to validate the whole process and the SeML itself, an application ontology for the coil winding machine (one of the intelligent motion control systems developed at Jilin University) is created, and its metamodel is *built* using the SeML and the developed ontologies, concerning the automation of its customization and implementation following generative approach.

1.2 CONTEXT

This Dissertation has been written in the context of a Masters Dissertation at the University of Minho, Portugal, in cooperation with the Embedded Systems Research Group Ontology Team (ESRG-OT) and the IMCSP's development team from Jilin University. This Dissertation investigates how the integration of ontologies with a SeML can assist the automation of systems' customization and implementation.

At the University of Minho, the ESRG-OT embraced a project concerning the SeML's development, its application and validation in the intelligent motion control systems' and in the hypervisors' domains. Within the members of the referred team, each one has an assortment of objectives regarding the realization of the interdependent global project, which was branched in the following Masters Dissertations:

1. Semantically-enriched Modeling Language:
   - SeML's Infrastructure - Miguel Abreu;
   - A Graphical Metamodeling Environment - Nuno Afonso.
2. SeML's application and validation:
   - Intelligent motion control systems:
     - Ontology-driven metamodeling towards intelligent motion control system design automation - Ricardo Teixeira.
   - Hypervisor:
     - Ontology-driven metamodeling towards hypervisor design automation:
       * Kernel Infrastructure - José Martins;
       * Secure Design Environment - Miguel Macedo;
       * Secure Boot - David Almeida;
       * Secure Monitoring - Pedro Pereira;
       * Runtime Security - Pedro Lopes;
       * Secure IPC - João Alves.

At Jilin University, the IMCSP's development team is composed by MSc and PhD students from distinct research fields, and aims at developing an intelligent motion control systems' platform to provide a *reference architecture* for the intelligent motion control systems' domain. The development team is divided through the Changchun and Zhuhai campus, both belonging to Jilin University. As advocated by Guzzo and Dickson, the students themselves are interdependent because of the tasks they perform as members of small teams, who are embedded in one larger team [Guz96]. Due to the magnitude of this project, it's imperative to complement its development with an abstract perspective that will improve systems' complexity management.

The present Dissertation is a contribution to the existing research in the semantics of the intelligent motion control systems' domain and, hence, aspires to lay the foundation to future students' Thesis and the global project. A small part of this Dissertation is therefore dedicated to the classification and categorization of existing approaches, in order to precisely identify shortcomings and weaknesses in previous solutions, to place the work developed into context and highlight this Dissertation's contribution.

1.3 AIM AND SCOPE

This Masters Dissertation has one main goal, which is the development of a domain and an application ontology, both designed to be integrated with the SeML, in order to allow the modeling of an intelligent motion control system, developed at Jilin University. Among others, this is expected to yield the following advantages:

- Reduces system's complexity, by allowing the system's designer to define system's parameters and behaviours, and thus automate its configuration and implementation;
- By reusing concepts and relations from different levels of abstraction (between core and domain ontology; between domain and application ontology), provides a semantically rich system's description that can be leveraged for:
  - Automatic reasoning;
  - Consistency checking between domain-specific/application-specific models;
  - System's updatability and upgradability, facilitating the integration of new modules/functionalities, and so, enhance its variability.
- Reduces system's development time, and subsequently reduces its development costs.

In order to correctly explore the inherent complexity of this topic, this Masters Dissertation's main goal is branched into seven distinct objectives, following the top-down approach partially illustrated in the Figure 1. These objectives will be discussed in depth in Chapter 3.3, but in here a brief summary is provided as an overview. The first objective is to analyse

the most prominent upper ontologies, in an attempt to identify the essential concepts necessary for a semantic integration of diverse DSLs for modeling software/hardware systems. Therefore, the second objective is contributing to the development of the SeML's infrastructure, aiming at design an ontology for software/hardware systems. Subsequently, the next objective is to develop a domain ontology, focused on the intelligent motion control systems' domain, which will decompose it into concepts, relations and axioms. The last objective before being able to model a IMCS is to design an application ontology that extends the domain ontology, and thus provides knowledge about the application within the referred domain.

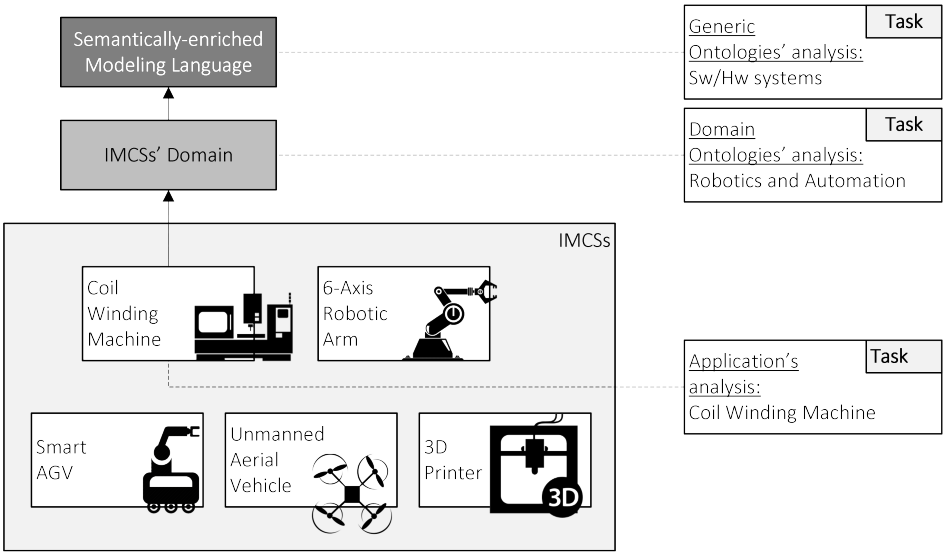


Figure 1.: Masters Dissertation's main sub-objectives through a top-down approach.

To demonstrate industrial applicability, an intelligent motion control system, the coil winding machine (a system that belongs to the IMCSP and which is already employed at the Mktech factory, in China), will be modelled with the SeML and the imported domain and application ontologies. This last step validates the developed ontologies and, consequently, the SeML in a higher abstraction level, the domain-level.

1.4 Organization of this Dissertation

1.4.1 *Chapter Structure and Outline*

Similar to a typical scientific paper, this Masters Dissertation's structure begins by reviewing the foundations and the theoretical background in Chapter 2, and then, in Chapter 3, the problem is analysed and the solution proposed. As one of the most relevant parts of this dissertation, Chapter 4 discusses the related work, before presenting the final results

of the developed work, strategy used and the concrete evaluation through a case study, in Chapter 5. Finally, in Chapter 6, the main conclusions about the entire project are realized and the future work is revealed. After this briefly outline of contents of each chapter, a more extensive one is following:

CHAPTER 2
Reviews the distinct emerging software development paradigms that this Masters Dissertation uses as foundation. In particular, the ODSD is the analysis' focus, and a common vocabulary for the remainder of this Dissertation is established by introducing and defining the most relevant concepts.

CHAPTER 3
Motivates the Dissertation and highlights the arising challenges. Then, an overview of the solution approach is provided and the expected benefits are described. Lastly, multiple research hypotheses are defined, as well as seven concrete objectives to guide the reader through the remainder work.

CHAPTER 4
Deeply analyses and compares related work in each of the problem domains that belong each objective. As aforementioned, this chapter is not a mere survey but instead intends to provide the conceptual foundation for all the developed work.

CHAPTER 5
Presents the theoretical insights and practical developments gained from the research conducted in this Masters Dissertation. Specially, it introduces the developed ontologies and the interdependencies between them. To round off the discussion, the ontologies are integrated with the SeML, being this process evaluated through a case study.

CHAPTER 6
Finally, it concludes this Masters Dissertation. Particularly, this document assess the feasibility of the approach by highlighting the advantages of the SeML as a modeling tool, and the developed ontologies as knowledge bases for the SeML. Furthermore, the Dissertation closes by identifying different directions of future work, both in terms of this project and in terms of the development of other projects directly related with the developed one.

1.4.2 *Typographical Conventions*

In order to provide a structured and emphatic visual guidance to the reader, this Masters Dissertation adopts the following typographical conventions:

- *Italic script* highlights scientific terms and proper names of patterns, methods and tools carrying special meaning in the software engineering literature.
- **Bold face** highlights important keywords;
- When referring to elements included in diagrams and other illustrations, Sans-serif font is used.
- In the PDF version of the Masters Dissertation document, the green text color represents bibliography entries, and the blue text color highlights chapters, sub-chapters, figures, etc.
- For code listings, name of classes and other code fragments the `Typewriter font` is used.

2

FOUNDATIONS AND BACKGROUND

This chapter contemplates and reviews diverse emerging software development paradigms that are of extreme relevance for the entire project developed by the ESRG-OT. In turn, the MDSD' goals, techniques and process building blocks that allows this paradigm to raise the level of abstraction in software engineering are discussed, in order to understand how this paradigm grants multi-domain and multi-application development approaches. Lastly, this chapter finishes with a more detailed introduction to the motivation, by exploring the techniques of the ontology-driven software paradigm. Thus, this chapter is extremely important, particularly because the rest of the Masters Dissertation is based on the terminology and vocabulary defined here.

2.1 MODEL DRIVEN SOFTWARE DEVELOPMENT

This section introduces the paradigm of MDSD, alternatively Model-Driven Engineering (MDE), beginning with an abstract view of this paradigm and being followed by the identification of the elements that compose a model-driven process. Subsequently, the core concept of MDSD is reviewed, **domain architecture**, by discussing the construction of domain architectures, introducing techniques and best practices that are important for its development, and approaching the multi-domain development processes. Finally, this section investigates the relevant characteristics and divergences between two levels of abstraction, domain and application level.

2.1.1 *Overview*

Each software has its own inner structure - inherent development paradigms, expressed in the source code. This structure directly influences the performance, maintainability and interoperability of the software, and so, it has an enormous economical impact. Nowadays, these paradigms are hardly identifiable on a programming language level, because of their low level of abstraction. To solve this problem, most software development tools were designed with a *Round-trip* engineering functionality (*e.g.* [Nie12, Alt]), or *reverse engineering*,

which synchronizes two or more related software artefacts. This is a functionality that most of the Unified Modeling Language (UML) tools offer, improving the graphical visualization of the code (through UML syntax), but not raising its level of abstraction, stagnating productivity [VSB+06, p. 12].

Model-driven Software Development intends to raise the level of abstraction in software engineering [Per14], creating models that have the exact meaning of program code and allow the final implementation to be generated from them, rather then just class and method skeletons. To this end, MDSD promotes models as *parts of the software*, constituting a decisive factor in increasing the performance of the process of developing software.

Model-Driven Architecture (MDA), an initiative [Gro03, Gro16a] of the Object Management Group (OMG), advocates a stratified approach, where models are mapped from higher levels of abstraction to levels that are close to the final implementation, prevailing the interoperability and software portability as main goals [KWB03, p. 9]. By contrast, MDSD aims mainly at:

- Increase the **development speed** through the automation of systems' code generation;
- Enhance **software quality**, as a consequence of the use of automated transformations and DSLs focused in modeling systems, or DSMLs;
- Provide a well-structured *separation of concerns* [Ram03], improving the **maintainability** of software systems;
- After conceiving the architectures and modeling languages, expand the acquired knowledge in order to increase **reusability** [JW05];
- **Manageability of complexity** through abstraction, using modeling languages;
- By combining the referred goals and the MDSD best practices, offer a **productive environment** in the engineering field.

After exploring the main differences between the MDA and the MDSD paradigms, the MDSD terminology should be the focus, in order to identify and define the concepts and terms that, contiguous with the MDSD processes, allow the subsistence of the aforementioned goals. **Domain** should be the genesis of the classification, and so is described as *a bounded field of action, thought, influence or knowledge* (adapted from [Dic17]) that can contain other domains, the **subdomains**. Now that domain was defined, a term should be created to define the *thing* that supports the realization of a domain, the **platform**, representing components that aggregate the shared parts of a software system family. The last relevant concept is **metamodel**, being the prefix "*meta*" to represent a higher level of description and the **model** to represent a function, behaviour or system's abstract representation. Thus, "*metadata* are data about data, and a metamodel is a model used to describe other models.

Metadata are descriptions of data, and metamodels are descriptions used to characterize models." [DDV13, p. 72].

In the following sections, these concepts and terms are elaborated, using them to explore this paradigm in great detail.

2.1.2 *The Elements of a Model-Driven Process*

This subsection explores the core parts of a MDSD process, in order to provide an abstract analysis of this complex topic, using the referred process to introduce MDSD concepts that are of extremely relevancy. A complete and comprehensive coverage of this topic was elaborated by Thomas Stahl and Markus Völter in [VSB+06], Holger Krahn in [KRV14] and Jorn Bettin in [Bet04].

Domain-Specific Modeling

A DSML is a DSL which domain is modeling systems. Therefore, is a specialized language that, besides raising the abstraction level of software [MHS05], embodies the concepts and relations that represent domain knowledge, not forcing systems analysts to reconstruct them from scratch, and thus optimizes systems' development process [CE00]. A DSML already includes domain's relations and rules that would otherwise have to be added manually, contributing to the model integrity. Therefore, the creation of models with a DSML allows the design, configuration and implementation of complete systems in a complex [FRBS+13], but very productive [Voe11], set of activities usually referred as **domain-specific modeling processes**.

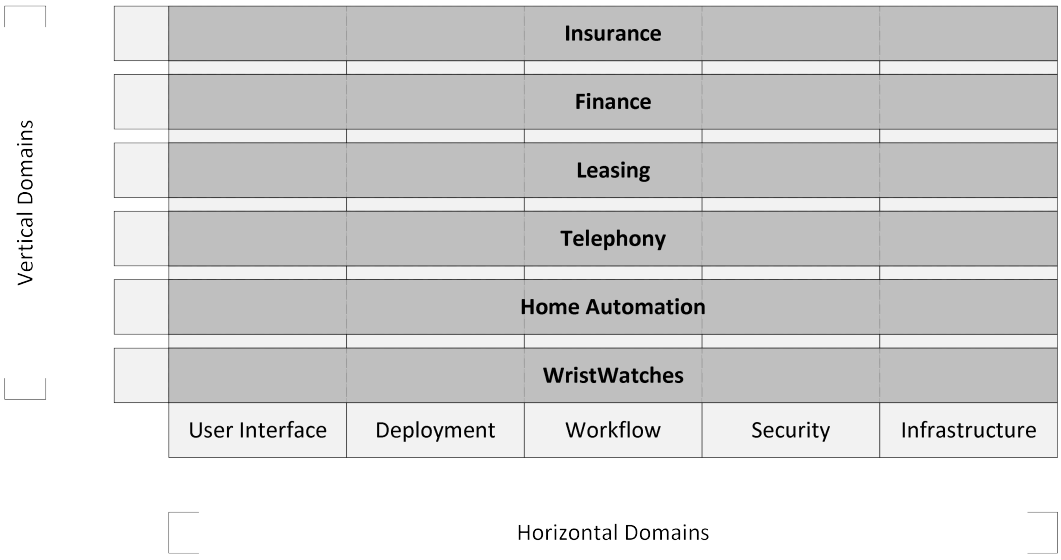


Figure 2.: Classification of horizontal and vertical domains.

A domain can be categorized in *vertical* and *horizontal* domains, according to several criteria. Vertical domains refer to complete domains, being alternatively referred as *technical domains*. Antagonistically, horizontal domains describe only sub-domains of systems, being alternatively referred as *solution domains*. Figure 2 illustrates the categorization referred above. In order to correctly understand the ontological foundations of a language's constructs, this dissertation must classify different DSLs. Being the semantic integration of DSLs one of the central goals of this dissertation, and due to the analysis of a multi-domain scenario that it requires, the DSLs' classification will focus on DSLs that model horizontal, technical domains. As a basis for the classification, Greenfield and Short (in [GS03]) classified DSLs in three dimensions with several sub-categories each. Their work was summarized in [ABm07], resulting in the classification presented in Table 1.

Table 1.: The domain-specific modeling languages classification, developed by Greenfield and Short

Category	Sub-category	Description
Types of information	Behavior	Describes dynamic execution of a software and interaction with other systems.
	Structure	Describes organization and composistion
	Artifacts	Describes artifacts' development, deployment, and execution
Abstraction level	Physical	Captures information concerning to constructs that exists outside the realm of execution
	Logical	Captures information concerning to constructs within the realm of execution
	Conceptual	Captures information concerning to constructs that are implied by the execution
	Platform Independence	Determines whether language constructs depend on technologies supplied by the platform
	Computation Independence	Determines whether language constructs describe information and processes in the real world or their automation in a computer system
Specification	Declarative	Describes the results of execution
	Imperative	Describes instructions to be executed
	Complementary	Combines declarative and imperative language
	Models with code	Adds imperative code to declarative models

This classification is the foundation of the Three-Dimensional Framework developed by Aßmann in [ABm07], and divides the classification of DSMLs between three categories, as illustrated in Figure 3, the **level of abstraction** (purposes a classification for levels of abstraction between computation-independent, platform-independent, or platform-specific), the **domain** (branched into structure, behaviour and user interface, in order to represent the aforementioned horizontal domains) and the **ontological kind of models** (that refers to the nature of the models' elements, and is partitioned into Type and Token).

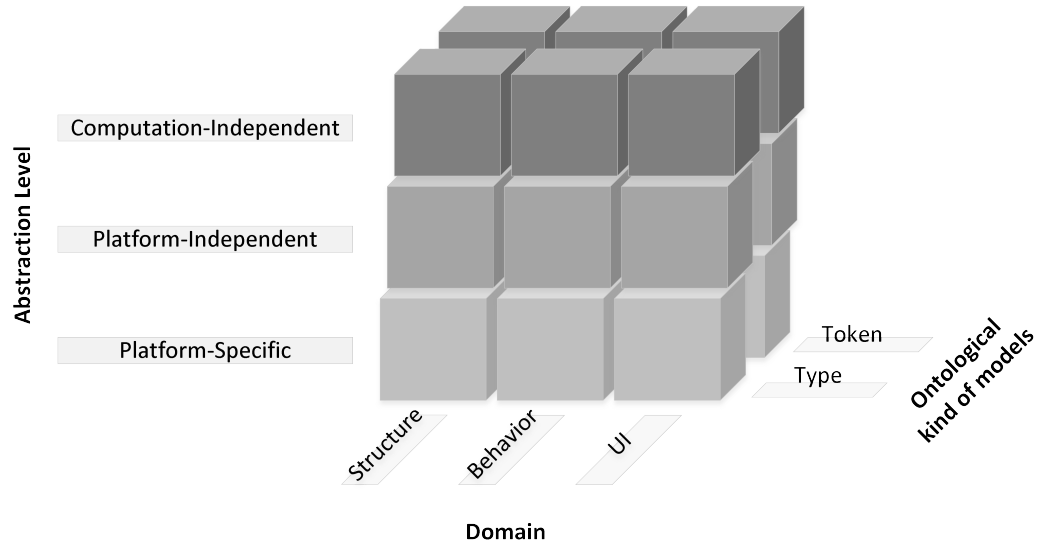


Figure 3.: The three-dimensional framework to classify DSLs (adapted from [ABm07])

The aforementioned framework will be used as a reference for DSMLs' classification in this Masters Dissertation.

Model Transformations

After reviewing the basics about DSMLs and defining the MDSD's terminology, this paragraph connects both and explores what is a MDSD's **transformation**. For Schmidt [Sch06], a transformation is a mechanism that "analyses certain aspects of models and then synthesizes various types of artefacts, such as source code, simulation inputs, XML deployment descriptions, or alternative model representations.", *videlicet* a mapping [GS04, p. 45] between two artefacts, *e.g.* two domains. As domains, transformations can have different classifications, *e.g.* [SK03], but in this Dissertation they'll be characterized as either vertical or horizontal [GS03]. Vertical transformations constitute a specification of domain, mapping an abstract domain into a more concrete one or to code. Contrarily, horizontal transformations can constitute a process of ***refactoring*** [FB99], which tries to improve the design without changing its meaning, or a process of ***delocalization*** [CE00], where the aim is to optimize an implementation. In a MDSD process, illustrated in Figure 4, designing and implementing models are the domain's designer responsibility, while the mapping between models and the application code are the application's designer responsibility. Lastly, the engine's designer is in charge of the transformation engine development and maintenance.

This classification will be very useful in the remaining document, because it will allow the differentiation between diverse responsibility levels, and thus, different goals within the ESRG-OT.

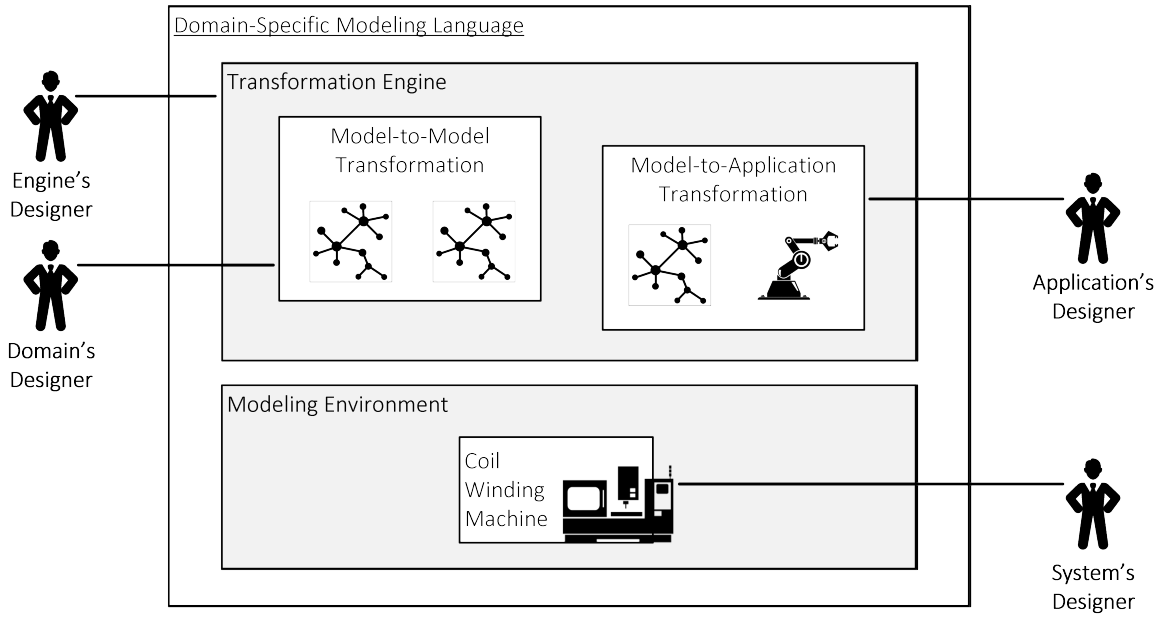


Figure 4.: Model transformations in a MDSD process.

2.1.3 *Domain Architectures*

After describing the MDSD's core concept, *domain architecture*, this subsection approaches its practical side, the *construction* of domain architectures. To this end, some techniques and best practices that are relevant for its development are explored, beginning with the **metamodeling** and the multi-domain modeling processes. Lastly, the difference between domain and application architecture is clarified.

Metamodeling and Multi-level Metamodeling

As aforementioned, metamodels are descriptions used to characterize models, therefore they must contain the knowledge to describe the abstract syntax of a DSL and to validate the models through the inferred metamodeling constraints and mapping rules. Figure 5 illustrates the relation between the instantiation (or real world elements in [VSB+06, p. 86]), the model, the metamodel and the meta metamodel. Instances, models, metamodels and meta metamodels have a class-instance relation and so, each *level* of this hierarchy represents an instance of the level above, constituting a stratified diagram that can be continued in a infinite way, usually referred as a **multi-level modeling process**, also called **deep metamodeling**. Besides being in use by the OMG [Obj13] for almost two decades, the referred stratified diagram raises some concerns identified in [ABM99] (see also [AKG11a, AKG11b]), due to the use of strict metamodeling that restricts the instance-of and describes to only be approved between pairs of conterminous layers and never within a layer. Therefore, when relations

between non-conterminous layers are an indispensable resource, the procedures developed by Juan de Lara and Esther Guerra in [LGC14], should be followed.

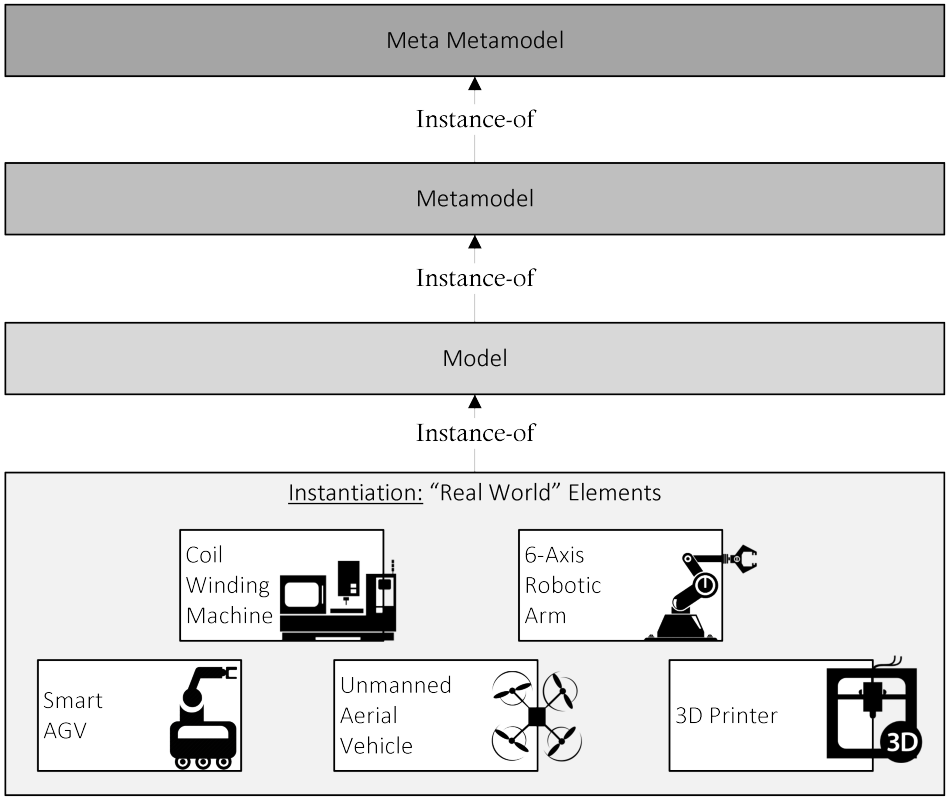


Figure 5.: Relation between the real world elements, models, metamodels and meta metamodels.

Building Domain Architectures

After exploring the stratified diagram that allows us to follow a multi-level metamodeling approach when modeling a system, the focus should now be the *construction* of a domain architecture. Accordingly with Thomas Stahl and Markus Völter [VSB+06, pag. 143], the prime challenge in building a domain architecture is to choose/develop a suitable DSL, taking into account the trade-offs between the following decision-points:

- Variability - Level of *freedom* needed to describe an application;
- Configuration and construction - There are two types of *not-disjoint* DSLs, those DSLs that grant support for creative constructions and those that provide the engineering resources to allow systems' configuration, both illustrated in Figure 6;
- The capability of modeling - Between modeling structural (component structures or persistence mapping) and/or behaviour aspects of systems;

- Concrete syntax - Since metamodels are descriptions used to characterize models, a DSL is the *interface* for the metamodel, and so, its syntax must be carefully designed to enable applications' developers to understand the models properly;
- Continuous validation of the metamodel - The stakeholders must be able to validate the model at any time, and thus, the grammar should compose valid sentences in the domain, avoiding misunderstandings.

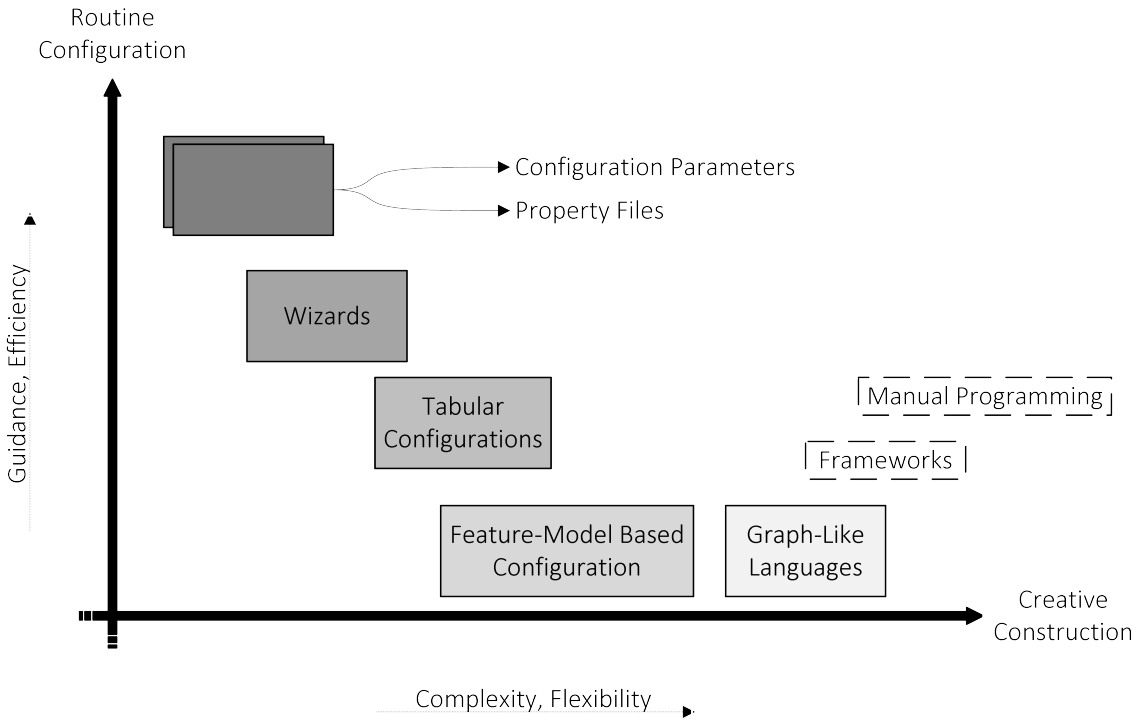


Figure 6.: Inherent trade-off in a DSL building process (adapted from [VSB+06, pag. 144]).

Regarding the work developed by Thomas and Markus, the remaining identified challenges address the development of a DSL instead of directly address the *building* of domain architectures, and so they are not relevant to the focus of this Masters Dissertation.

According to RUP [Kru03], a **reference architecture** is:

> "A predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed, and proven for use in particular business and technical contexts, together with supporting artefacts to enable their use. Often, these artefacts are harvested from previous projects."

Thus, a domain model is, in its basic essence, a reference architecture because it provides a proven architecture pattern both in business and technical contexts to its instantiation. Therefore, in order to *build* a domain architecture, two good practices should be respected,

being the development a suitable DSML the main one, and the formation of an **architecture development group** the last one. This group should be constituted by members from various IT organizational departments, in order to design the domain/reference architecture respecting the existent tools, technology and work developed within the organization [Ree02].

Multi-Domain Modeling Process

The previous sections highlighted the relevance of the multi-level metamodeling approach and the development of domain architectures as reference architectures. Subsequently, the exploration of the **multi-domain development process**, illustrated in Figure 7, enables the ESRG-OT to develop the techniques to model two independent domains simultaneously with the same tool, the SeML.

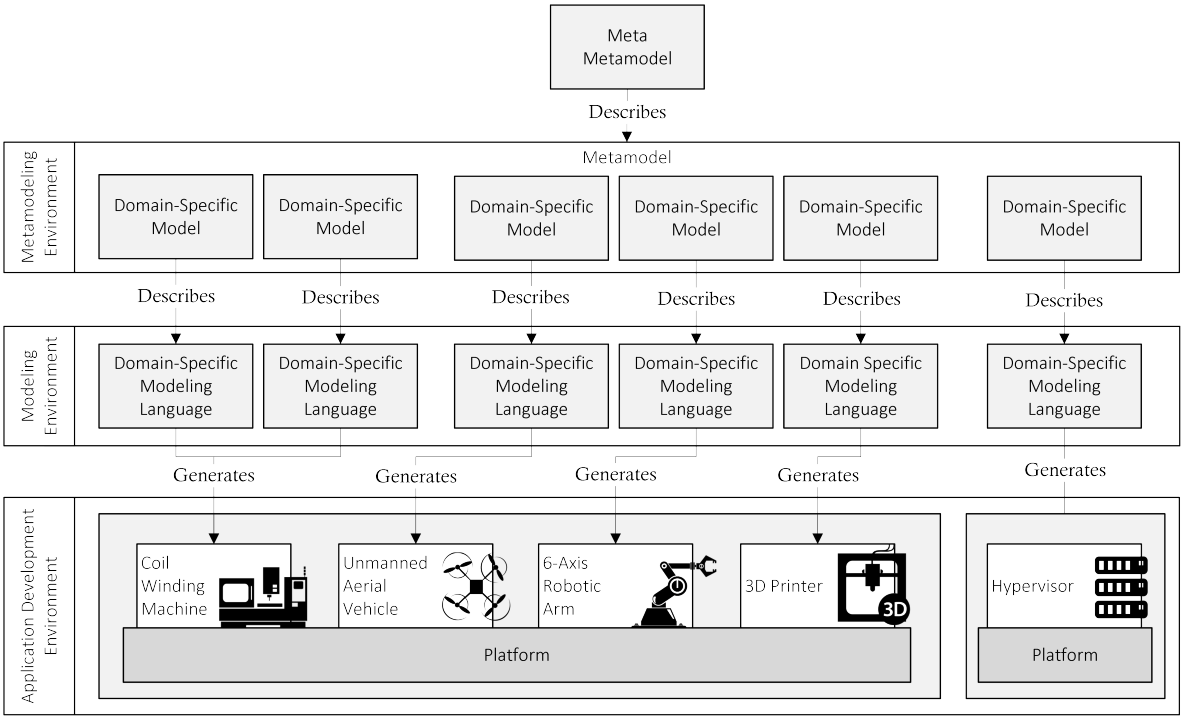


Figure 7.: Overview of a multi-domain modeling process.

During a multi-domain modeling process, the DSL designer is responsible for its maintenance, providing a metamodeling environment, while the domain designer and the application designer (or the aforementioned architecture development group) are responsible for *building* the reference architecture, in a modeling environment, allowing the application's code to be generated for a specific platform in an application development environment.

Domain and Application Architecture

The domain architecture, or reference architecture, provides support and formalizes a domain. Therefore, a domain architecture must be completely independent from an application, allowing it to cover a software system family.

In order to elucidate the reader on the main difference between application and domain architecture, Figure 8 illustrates the formal modeling process, which denotes the process that allows the application's designer to develop an application architecture, using a DSML. To do so, the application's designer creates a mapping between the concrete application's concepts with the concepts provided by the domain architecture, and uses the DSML to develop the model and to **vertically transform** the developed model into application code, as illustrated in Figure 4.

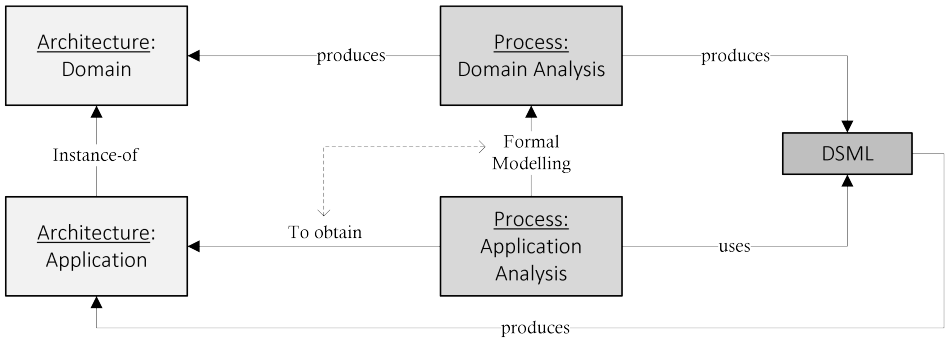


Figure 8.: Domain's and application's architecture development threads.

2.2 SEMANTICALLY-ENRICHED SOFTWARE ENGINEERING

This section introduces Semantically-enriched Software Engineering (SeSE), a paradigm that represents the act of applying techniques to support and explore the semantics within information (the opposite of syntax and structure), conducive to complement and improve information systems [SR03]. It begins by investigate and review **ontologies** as a semantic technology to define and explore DSML's inherent semantics. Therefore, this section approaches the theoretical foundations of ontologies as a knowledge representation technology, and its classification. It continues with the introduction of the **Semantic Web**, identifying diverse standards and languages as the prime technology used in the practical developments of this Masters Dissertation. Lastly, the fundamental ontologies' development methodologies are introduced and the description of a SeSE's research field, the ODSD (an emerging development paradigm that constitutes a foundation of this Dissertation) concludes this chapter.

2.2.1 *Ontologies*

The "**ontology**" term, originally borrowed from philosophy, has an increasing popularity over the computer science and information systems communities, mostly because of the type of problems that ontologies aim to solve in these domains, *e.g.* interoperability between multiple reality representations and real reasoning capabilities that reduce the gap between humans and computer systems. Although, as explored by Guarino and Giaretta [GG95], this term has multiple definitions when applied to different domains, and thus, an ontology can represent:

- A philosophical discipline;
- An informal conceptual system;
- A formal semantic account;
- A specification of a conceptualization;
- A representation of a conceptual system via a logical theory:
- Logical theory's vocabulary;
- A **meta-level** specification of a logical theory.

Therefore, there is still a lot of inconsistency in the usage of the "ontology" term, in particular in limits between the computer science and information systems domains of research, and mostly because of the following three foundational ontology aspects [HHDL+08]:

- Truth *versus* consensus - The decision between develop an ontology that reflects the **truthfulness** of structures, and thus being context independent, or a consensual ontology;
- Formal logic *versus* other modalities - For most of the researchers, an ontology is not only a logical account, but also a textual definition of its elements.
- Specification *versus* conceptual system - The trade-off between define an ontology as a representation of a conceptual system or its specification, and thus, decide the abstraction level of an ontology.

This Dissertation's research agrees that an ontology should reflect the **truthfulness** of structures, in order to be context independent, and therefore include not only logical account but also a textual definition of its elements, being able to describe a domain at every abstraction level required. To conclude this overview, the definition of ontology is stated as follows - a means to properly and *declaratively* represent a body of knowledge, and thus,

it's considered that "they form the foundation for building knowledge bases, representing domain terminology and enabling communication and knowledge mediation between heterogeneous agents in knowledge-based systems" [Dev02].

The following Subsections explore and expose in more detail the definition of ontology and their comparison with the actual knowledge representation methods and modeling techniques.

Ontologies versus Hierarchies of Knowledge

Sometimes, ontologies are regarded as hierarchies, *e.g.* taxonomies, that provide *domain knowledge*. However, due to the development of languages (OWL [OWL12], Resource Description Framework Schema (RDFS) [DRM12], Web Service Modeling Language (WSML) [dBFK+12], etc.) and another technologies that exploit ontologies' knowledge, it's possible to express the semantics of individual concepts in a way that simple taxonomies/hierarchies can't do.

Due to the fact that the aforementioned languages and other tools can be used both to create ontologies and to create **knowledge bases**, sometimes there are a misunderstanding between the divergences between ontologies and knowledge bases. An OWL file can be used to describe both ontologies and knowledge bases. Nevertheless, an ontology contains the vocabulary and the formal specification of the vocabulary (domain rules and axioms), which can be used to express a knowledge base, and aims at the interoperability between diverse knowledge bases. Thus, as Hepp suggests in [HHDL+08], a differentiation between ontological and data individuals should be assumed, with the latter representing part of a knowledge base within a domain and the former being part of the *specification* of a domain.

Extensible Markup Language (XML) schemas [BPSM+08] are a way to define a representation syntax for a particular domain through sequential and hierarchical fields, aiming at moulding context-independent categories of things. Thus, as knowledge bases, they are confused with ontologies too, but since XML schemas does not represent explicitly the semantics within a domain, they are not ontologies.

Besides XML schemas and knowledge bases, the relation between Knowledge Organization Systems (KOS) with ontologies is a source of misconceptions too, due to the fact that KOS "are means for structuring storage of knowledge assets for better retrieval and use" [HHDL+08, p. 7]. Although, KOS are not able to correctly define the *subclass relations* or the meaning of *being an instance*, and thus, since a key property of ontologies is the meaning of subclass/instance and its context-independent notion, these two methods of knowledge representation and classification diverge.

Ontologies - An Emerging Modeling Technique

A formal ontology relies on languages to express its axioms, using the formal semantics of these languages (typically based on abstract mathematical notions) to develop a domain knowledge base. This knowledge base can be used to *enrich* a DSML, by providing the concepts, properties, relations and constraints that enable the ontology-DSML combination to be a novel modeling technique [GHW03, Obe06]. This emerging technology in the conceptual modeling techniques domain, such as UML [Obj08] and the Entity-Relationship Model (ERM) [Che76] (or one of its variants [Bro75]), have fundamental differences to the traditional modeling techniques in terms of scope, aim and foundations. Summarizing Oberle [Obe06]:

- Ontologies have unambiguous semantics;
- An ontology aims at an agreement on the vocabulary terms, properties, constraints and axioms, between different knowledge bases of heterogeneous domains;
- Ontologies support languages, which enable reasoning and querying capabilities.

As referred in the Section 2.1, the MDA infrastructure aims at the creation of models and metamodels, managing metadata and the transformation between models (vertical transformations) or between models and applications (horizontal transformations). Based on the aforementioned advantages, the *integration* of ontologies with a DSML is a necessity in the perspective of this Dissertation, in order to reduce language ambiguity, and enable validation and automated consistency checking [TPO+06].

Conceptualization and Classification

According with Gruber [Gru93, p. 199], ontologies' development involves their provision in multiple formalisms, adopting an understanding of ontology as a *conceptualization* instead of its *specification* in a particular language.

The term ontology was already defined in the diverse domains where it can be applied. Subsequently, this Paragraph defines what is exactly a **conceptualization** and how can a conceptualization be explicitly described. In this Dissertation, a conceptualization is an "implicit and language-independent view on a domain of interest", [ABm07], committed by a logical language through an ontological commitment, which allows diverse agents to share vocabulary in a consistent and meaningful manner. Therefore, an ontology is expressed using a logical language which ontologically commits to a conceptualization, which in turn contains the logical theory of an ontology. However, an ontology may contain only a limited *perspective* of a conceptualization [GG95], and thus cannot guarantee completeness regarding a certain conceptualization, but only consistency [Gru95]. Although, there are some *directives* that should be followed when developing an ontology:

1. Defining a precise formalism to provide the correct interpretation of the vocabulary, properties, constraints and axioms within an ontology, and its exchange at a syntax level;

2. Developing the referred formalisms aiming at optimizing the reasoning capability;

The best practices and *directives* to develop ontologies will be approached in the next Paragraph, followed by the constraints of ontologies' development processes, but this brief explanation allows us to define a classification for ontologies. There exist diverse approaches to classify ontologies, namely by Lassila and McGuinness [LM01], by Oberle [Obe06, p. 43.47], by Uschold/Gruninger [UG96] and by Hepp [HHDL+08]. Lassila and Mcguiness developed an ontology *spectrum* to classify ontologies accordingly with their formal semantics degree, while Oberle, Uschold/Gruninger and Hepp classified ontologies through a multi-dimensional framework.

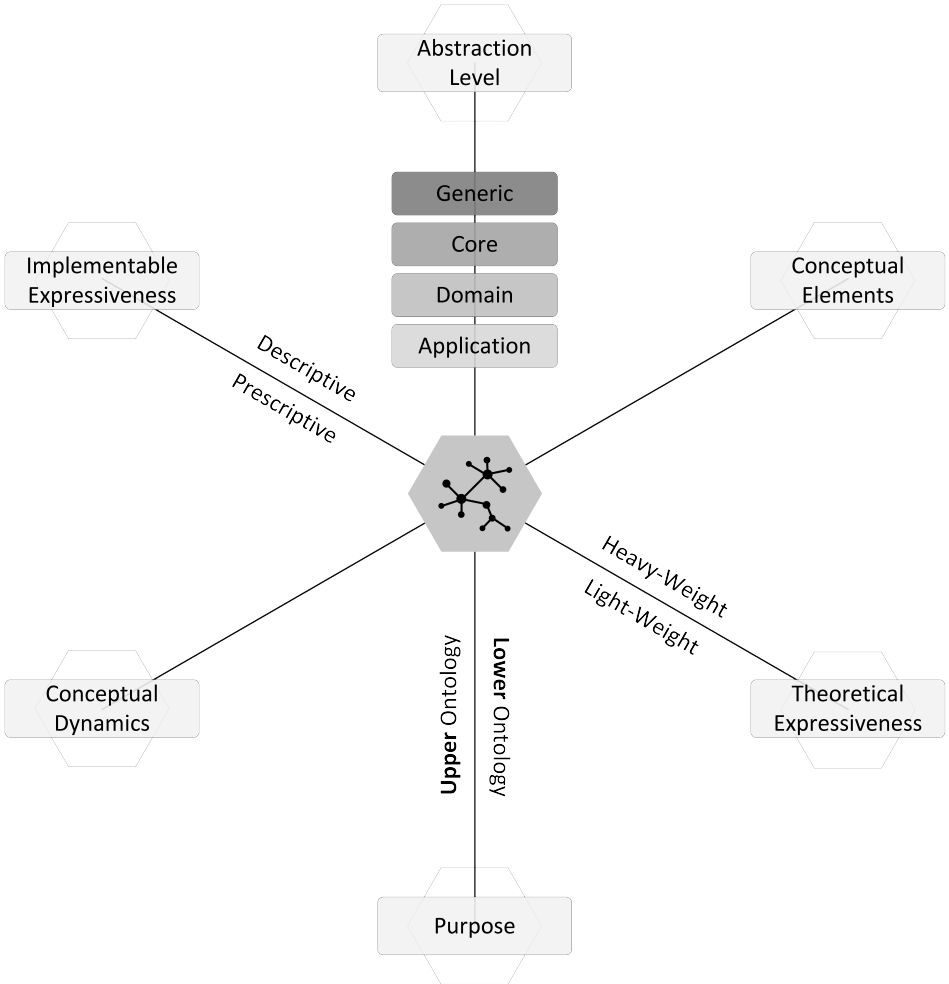


Figure 9.: HexOntology, a six-dimensions classification framework of ontologies.

This Masters Dissertation developed and follows a multi-dimensional classification framework, illustrated in Figure 9, to evaluate ontologies through six properties:

- **Purpose** - Allows the differentiation between an **upper ontology**, that enables an agreement between agents belonging to the same metalevel, and a **lower ontology**, which is an ontology without instances in the form of ontologies;
- **Abstraction level** - Refers to the degree of specificity of an ontology, denoting if it is a **generic ontology**, which covers diverse and heterogeneous fields, a **core ontology**, which covers some interrelated domains, a **domain ontology**, which enables an agreement between application ontologies, or an **application ontology**, which is composed by the semantics within a specific application;
- **Implementable expressiveness** - This property can range from *prescriptive* to *descriptive*;
- **Theoretical expressiveness** - This property can range from light-weight to heavy-weight, and represents the expressiveness of the formalism needed to create an ontology.
- **Conceptual dynamics** - The relation between the recently created conceptual elements and the old ones, per period of time. Thus, representing the relation between the knowledge capture by an ontology and the actual knowledge existent in a specific domain at some abstraction level;
- **Conceptual elements** in the domain - The number of elements constituting an ontology.

Due to the fact that there are some *interdependencies* between this framework's properties, this classification framework can be simplified, allowing an agile comparison between ontologies. Figure 10 illustrates another perspective of the HexOntology, highlighting the relations between the purpose, implementable expressiveness and abstraction level.

Each layer in the alternative HexOntology classification framework denotes a different abstraction level, from highly specific (descriptive application ontology or prescriptive application ontology) to very general (descriptive generic ontology), providing a vertical classification of ontologies:

Generic Ontology. By defining general concepts that are applicable in almost all domains, aims to be a foundational ontology inspired by philosophical considerations regarding the nature of things and their relations in general, and provides a *complete* perspective of reality.

Core Ontology. Is an *instance-of* a generic ontology and allows an agreement between a limited number of similar domains.

Domain Ontology. Is an *instance-of* a core ontology and contains domain specific concepts, constraints and relations, enabling an agreement between application ontologies.

Application Ontology. Is an *instance-of* a domain ontology and the last metalevel before the application's model. Therefore, it specifies the knowledge from the domain ontology into application-specific concepts, relations and rules.

Besides the vertical categorization, exists an horizontal categorization, in the ranges of the implementable expressiveness property, which describes a fundamental feature of a model [Sei03], and can be either **descriptive** or **prescriptive**:

Descriptive Ontology. Describes reality but reality is not constructed from it, and, due to its capability to describe and conceptualize *things*, are used in analysis and re-engineering processes.

Prescriptive Ontology. Describes a model that is a specification of reality (describes the structures and/or behaviours of reality and reality is constructed according to the model), and thus focus on the specification, control and generation of systems.

These are the definitions that will be used in this Masters Dissertation, in opposition from the analysis developed by Aßmann/Zschaler, which culminated with the proposal for the role of ontologies in a meta-pyramid [CRP06, p. 266]. They assume that an ontology is "a shared, descriptive, structural model, representing reality by a set of concepts, their interrelations, and constraints under the open-world assumption", directly advocating that an ontology cannot be either descriptive or prescriptive. Besides Aßmann/Zschaler, this Dissertation contradicts Gruber ([Gru93]) and Devedzic too ("Generally, an ontology is a meta-model describing how to build models." [Dev02]), due to the fact that they advocate that ontologies are a design or implementation model, and so, cannot be descriptive. Lastly, another source of misconceptions is the term **upper ontology**, which is used to denominate any ontology in a **metalevel n** that has an instance ontology in a **metalevel (n-1)**, *e.g.* in Figure 10, domain ontology is an upper ontology for the application ontology. This term must not be confused with the way that Aßmann uses it in [ABm07, p. 22] (in his work, an upper ontology is the top-level ontology, and so, a Generic Ontology in this classification).

After exhaustively exploring some of the most relevant definitions in this Dissertation and the ontologies' classification, the next subsections will approach the methodologies and constraints of ontologies' development processes.

Ontological Development Methodologies

In the last decades, ontologies have been known as consensual models of domain of discourse, either focused in systems' specification or in reality description. Although, as advocated by Martin Hepp in [Hep07], the number of real and useful ontologies available in

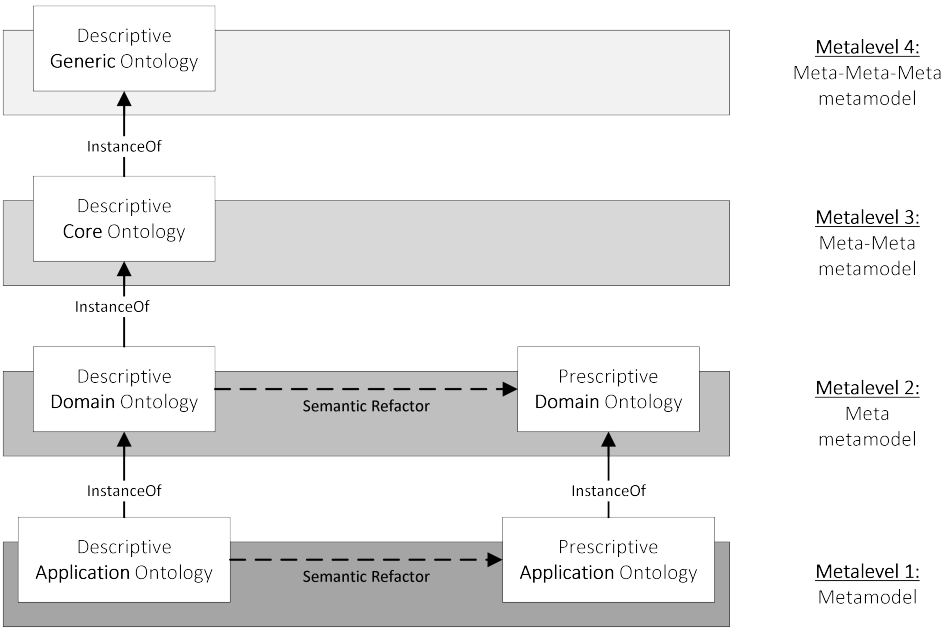


Figure 10.: Alternative perspective of the HexOntology.

the Web is very low. In order to contradict this tendency, the science community should regard the constraints of developing an ontology, explored in the next subsection, and the methodologies already developed to improve the *quality* of an ontology.

This subsection does not pretend to transform completely the ontological art in engineering, but aims at identify and clarify some methodologies and requirements that were used to developed the ontologies approached in this Masters Dissertation. Therefore, in here is presented an overview of the work developed by Fernandez [FLGPJ97], Guarino [GW02] and Jansen [JS11]. Notwithstanding, there are other outstanding works on how to develop ontologies methodologically, *e.g.* Uschold/King in [UK95], Gruninger/Fox in [UG96], On-To-knowledge in [SSSS01], among others.

METHONTOLOGY

Developed at the Universidad Politécnica de Madrid, by the Ontological Engineering group, **METHONTOLOGY** aims at enable the construction of ontologies at the knowledge level, and has its roots in some main activities explored by the IEEE software development process [Sch97] and in other methodologies [MS88]. This methodology guides in how to carry out the whole ontology development process through a set of well-structured activities, illustrated in Figure 11. The METHONTOLOGY activities does not intend to imply an order of execution of its activities, but instead highlight the list of activities to be completed:

1. Planification - Explore the details about the resources and schedule needed to build the ontology;

2. Specification - Define the goal of the ontology;
3. Conceptualization - Acquire and/or systematize knowledge to conceptualize it in a model that describes the domain's problems and solutions;
4. Formalization - Convert the conceptual model into a formal model, using description logic representation systems;
5. Integration - Integrate the developed ontologies with well-structured ontologies, allowing an extension of the knowledge representation and an agreement between different domains/communities;
6. Implementation - Implement the ontology in a formal language;
7. Evaluation - Evaluate the developed and implemented ontology;
8. Documentation - Document the finished ontology, allowing other developers to be able to understand and use it;
9. Maintenance - Maintain the ontology using maintenance guidelines.

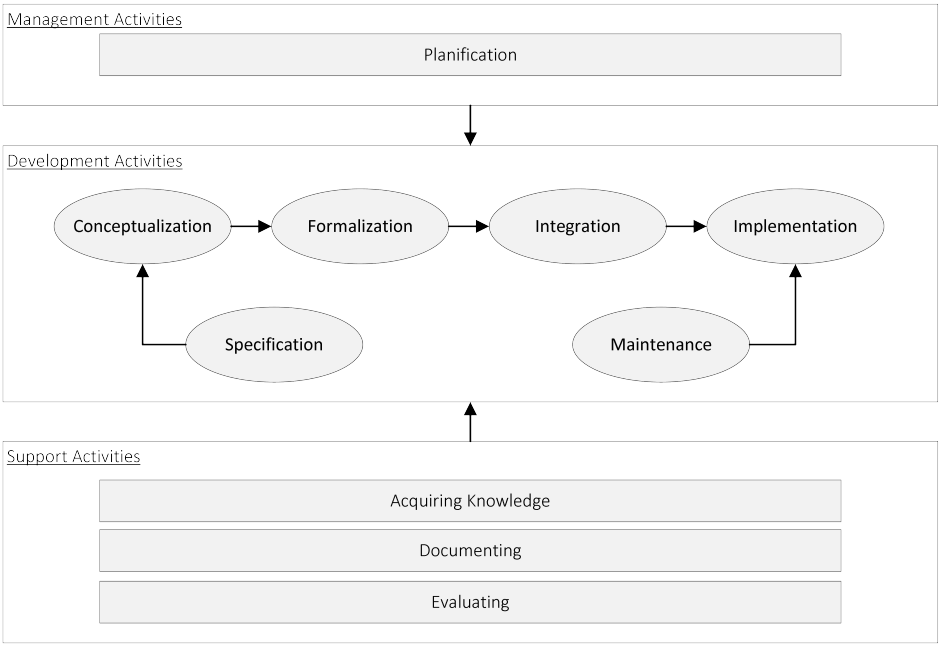


Figure 11.: Activities in ontologies' development following the METHONTOLOGY methodology.

This activities constitute the METHONTOLOGY methodology to correctly develop ontologies and to reduce the existing gap between the ontological art and the ontological engineering.

ONTOCLEAN

The OntoClean methodology, developed by Nicola Guarino and Chris Welty, aims at provide guidance two solve two questions:

1. What kind of ontological decisions need to be made?
2. How these decisions can be evaluated?

And by answering this questions, the OntoClean, provides a support for the ontology development process based on highly general ontological notions drawn from *analytic metaphysics*, a field from the philosophical ontology domain. To do so, this methodology uses **formal notions** to define a set of **metaproperties** (domain-independent properties of classes) and to characterize relevant aspects on the properties, classes and relations within an ontology. Therefore, OntoClean classifies an entity's property accordingly with its:

- Essence and rigidity - A property can be **rigid**, if it is essential to all of its instances, **non-rigid**, if it is essential to some entities and not essential to others, or **anti-rigid**, if it is never essential;
- Identity - Capability of recognizing individual entities in the world as equal or different;
- Unity - Capability of recognizing all parts that constitute an individual entity.

This methodology of classifying entities' properties allows the ontology's developer to identify the *backbone taxonomy* and to clearly define when and where not use *subsumption* in the taxonomy [GW02].

THE TEN COMMANDMENTS OF ONTOLOGICAL ENGINEERING

These commandments aim at compile and condense ontology development rules following a realist and engineer perspective to provide higher probability of sustainability, interoperability and adaptability to a specific domain. These commandments are categorized in two approaches, an abstract and a specific one:

- Realism, Multi-perspectivalism and *adequatism*:
  1. Realism - The prime goal of an ontology is the description of the world, and not the description of words or meaning-entities;
  2. Integration Capability - Be a part of a *perspective* and not the *perspective* itself, and thus, developing an ontology that can describe the world through a perspective while integrating with others ontologies with different ones, instead of advocating that there is only one perspective;

3. Domain boundaries - Respect your domain, regarding its boundaries and granularity;

- Terms, definitions and relations:

4. Appropriate conceptualizations - By not providing classes or relations' labels somewhat artificial and not commonly used in oral or written communication;
5. Unambiguous definitions - Avoid ambiguity when defining classes or relations;
6. Upper ontology need - The integration with an upper ontology reduces the ontology ambiguity and provides mapping between ontologies, allowing an agreement between different domains/communities;
7. Aim at interoperability - Avoid *eclecticism*, by choosing the correct types in an ontology, and *parochialism*, by considering that the domain to be covered or the entities described in the ontology itself are not all there are in the world;
8. Ontology *versus* epistemology - Describe a reality through a perspective (ontology's goal) and not the perspective itself (epistemology's goal).
9. Ontology's artefacts - Ontologies should avoid classes that are not extensions of types.
10. Ambiguous relations - Avoid using the same terminology for relations if the meaning is different, in particular with *is-a* relation, implicit quantifiers and singular numerous.

Although several methodologies have been proposed, no methodology has established as a standard [PM04]. Therefore, this Masters Dissertation will merge these three methodologies in order to fill the individual *gaps* and use a merged methodology to develop the domain ontology for intelligent motion control systems and the respective application ontology, aiming at modeling the coil winding machine developed at the Jilin University. The methodology advocated by METHOTONTOLOGY's developers has a high *using-cost*, although should be followed. Nevertheless, this methodology does not specifies the development methods which allow an ontology to improve its probability of sustainability, interoperability and adaptability to a specific domain, and so, should be complemented with the 10 commandments methodology. Lastly, this two methodologies should be complemented with the OntoClean to fill the *gap* that exists in the *metaproperties* analysis, essential due to the fact that the system's model is composed by the application ontology's instances.

Ontology Development Constraints

Although diverse methodologies exist for *building* ontologies, they (even after being merged) do not answer questions related to ontology-engineering practices, such as:

1. Can we develop ontologies fast enough to reflect domain's evolution (**conceptual dynamics**)?

2. Does the automation provided by ontologies justifies the development cost?

3. Is the ontology perceptible and meaningful to the user?

To answer the first question, it is an absolute necessity to understand that almost every domain has a continuous degree of conceptual dynamics, being this degree increased with the *specificity* (the aforementioned **abstraction level**) of modeling. Therefore, if the ontology does not follow the principal of minimal ontological commitment, does not maintain the ontology-engineering/ontology-maintenance lags small and does not limit the degree of conceptual dynamics, the ontology developers will face an *obstacle* identified by Hepp [Hep07, Figure 1], as having an ontology that, after the initial domain capture, will always have a *conceptual discrepancy* (*e.g.* domains such as pharmaceuticals, and chemical substances have a high degree of conceptual dynamics and thus, constitute a problem to ontology developers [Hep13]).

The economic question embodies a huge dimension in the ontologies' development processes, due to the fact that the creation of an ontology is characterized as a high development-cost process. Simperl and Tempich developed ONTOCOM [KVV06] in order to predict the referred costs and to generate a cost model specialized in a particular ontology, but this cost model aims at resource-consumption aspects rather than relevant economic driving forces. Ontology's inherent costs are a result of both those who create or contribute to the ontologies and the users of it, due to **familiarization and commitment costs**. Thus, the production of good documentation about the ontology, alongside with the good practices already identified, helps increase the probability of its use/reuse and reduces the development cost.

An ontology does not represents *the perspective* of reality but *a perspective* of reality, and therefore the user will be compelled to accept the inherent perspective of *things* and eventual discrepancies that it could have. Nevertheless, the number of ontologies' users (also referred as community size) will not only be determined by its *quality* (and thus, the quality of its inherent perspective), but also by its *complexity*, being the latter reflected by the size [Hep07, Figure 5], detail, expressiveness and conceptual dynamics of it [Hep07, Figure 4].

2.2.2 *The Semantic Web*

Promoted by the W3C and supported by a large community of researchers, the **Semantic Web** is a framework that aims at enable data's sharing, reuse and integration across the limits of applications and enterprise [Koi13]. To do so, it defines semantic techonologies (*e.g.* languages) to capture this data, represent it, and specify its semantics to be published. This

languages are XML- or Resource Description Framework (RDF)-based, as referred in the last subsection, and have a logical foundation in ontologies and knowledge representation systems.

Theoretical and Technological Foundations

The W3C vision comprises the Semantic Web as a layered architecture framework, illustrated in Figure 12, that can provide improved interoperability (compared to traditional XML approaches) due to Semantic Web languages' inherent richer *expressivity*, reusability (on a global scale) and unambiguous vocabulary to users and developers. To do so, the Semantic Web works as an *open-world* [KOTW06], in which new information can be added to the existing resources, linking everything. Therefore, it is **impossible to rule if a specific statement is true or will be true**, resulting in the so-called "**open world assumption**", which means that **a statement is not necessarily false if it cannot be proved true**. This implies a mindset for the *squared* developers, used to the closed and finite modeling domains, and offers the aforementioned advantages.

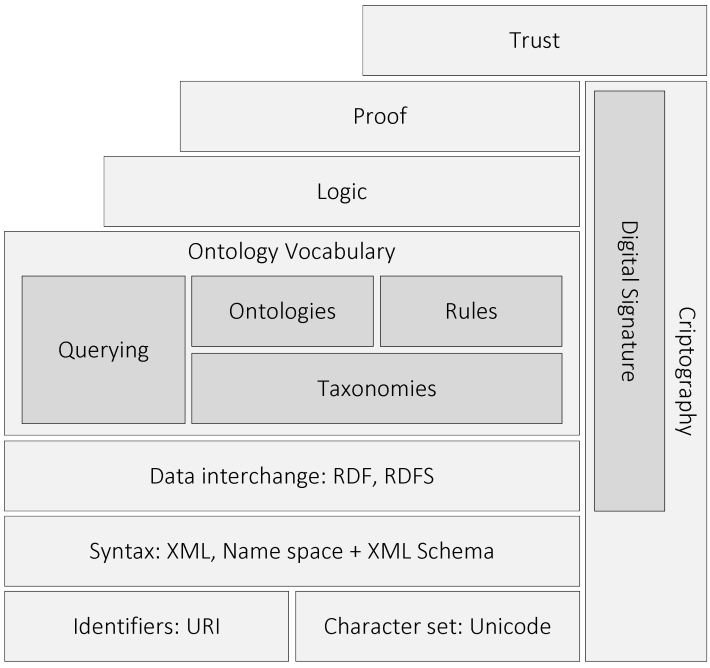


Figure 12.: Semantic Web's architecture (adapted from [BLoo, FM01]).

The referred W3C vision is only possible with the development of semantic-enabling technologies to foster an early industry's adoption and development tools support. At the moment of writing of this Masters Dissertation, the Semantic Web community developed a set of semantic-enabling technologies to develop, maintain, use and share domain models for software engineering, allowing an agreement up to the **ontologies, rules and querying layer**. A brief overview of these technologies is presented below:

- RDF - Provides a basic data model to express formally associations among resources that can be identified via a Identifier (*e.g.* Uniform Resource Identifier (URI)). This framework [Guu] is intended for situations in which information needs to be processed by applications, and thus, constitutes a semantic-enabling technology to express and exchange the referred information without loss of meaning. To do so, this framework allows the creation of statements about resources, represented as sets of triples resembling the subject, predicate and object of an elementary phrase, such as International Resource Identifier (IRIs), literals, blank nodes and multiple graphs;

- RDFS - Is the RDF vocabulary description language, extending it to include the basic features required to describe ontologies. Therefore, RDFS specifies an object-oriented model for RDF, defining how classes, subclass, relations, properties and datatypes are represented, and enabling the creation of ontologies based on taxonomic relations;

- OWL - Is a richer and powerful ontology language that extends RDFS and uses the RDF syntax as its base grammar. As advocated by Ian Horrocks in [Horo8], "A key feature of OWL is its basis in Description Logics (DL), a family of logic-based knowledge-representation formalisms that are descendants of Semantic Networks and KL-One, but that have a formal semantics based on first-order logic", adding expressiveness [Saho7] through vocabulary to specify classes based on property or cardinality restrictions, to support boolean combinations of classes and to allow properties to be transitive, unique or inverse. OWL is constituted by three increasingly expressive sublanguages [DSB+04], namely Web Ontology Language Lite (OWL-Lite), Web Ontology Language Description Logics (OWL-DL) and Web Ontology Language Full (OWL-Full);

- Web Ontology Language 2 (OWL2) - Is an extension and revision of OWL [Gro16b] with a new syntax [HPS12], three new profiles [MGH+], new features and new *rationales* [WG12];

- Semantic Web Rule Language (SWRL) - Being a W3C member, combines unary/binary datalog sublanguages of the Rule Markup Language (RuleML) [PBZ+12] with OWL-Lite and OWL-DL. SWRL [HPsB+04] extends the set of OWL axioms to comprise horn-like clause rules, improving the reasoning capability through additional expressiveness.

For the practical development of this Masters Dissertation, OWL2 was the representation format of choice, due to the fact that is an extension and a review of an already solid theoretical grounding and relatively mature tool support, OWL. Therefore, the following definitions, adapted from [BKMPS09, HKP+12, DSB+04], allows an agreement between the vocabulary to explore OWL2 ontologies:

Class. Represents a group of resources with similar characteristics. Like RDF classes, where resource (which is the object of an rdf:type triple) is itself an instance of the rdf:class resource, every OWL class is related with a set of individuals, denominated **class extension**, where each individual is called an **instance** of the class. The class extension is related with the class intentional meaning, but that does not implies that its underlying concept is defined by it. Thus, "two classes can have the same class extension, but still be different" [DSB+04]. In order to describe a class, a developer may establish singular **class descriptions**, or combine them to *build* **class axioms**.

Individuals. Members of a class description, as aforementioned. Individuals are defined through **axioms** (also referred as *facts*), that can be about class membership and property values of individuals, or about the individual entity.

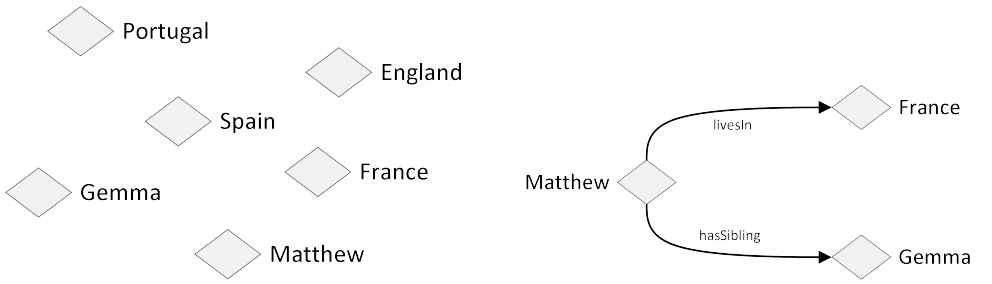
Properties. Through binary predicates and a defined *domain*/*range* pair, a property can semantically connect individuals (**object property**), individuals to data values (**datatype property**), or individuals to annotations (**annotation property**). Additionally, an object property can be declared as **functional** property, **inverse** property, **transitive** property, **reflexive** property, **asymmetric** property, **symmetric** property and/or **inverse functional** property. Besides that, a datatype property can be described as an **equivalent** property, a **disjoint** property or a **pairwise disjoint** property.

Datatypes. Properties that links individuals to data values. A datatype property consists of a **RDF datatype** specification, a **rdfs:literal** (can be either plan or typed and it is used for values such as strings, numbers and dates - [CWL14]) or an **enumerated datatype** (to define a range of values).

Annotations. Properties to encode information about the ontology, rather then about the domain of interest.

Figure 16, depicts a practical ontology composed of excerpts of an ontology used to model pizza's domain [PHK+11]. In here, are introduce two *standard notations* to represent ontologies during this Masters Dissertation. The first notation illustrated in Figures 13 and 14 is used to illustrate theoretical foundations of OWL, to explore ontologies from other developers and/or represent theoretical ontologies' implementations, and the second one, illustrated in Figure 16, is an ontology generated by OntoGraf [Fal10] (using Protégé [GHH+]), and will be used to illustrate the ontologies developed and integrated with the SeML.

Contrarily to what happens with **Protégé**, which is the tool chosen to *build* this Masters Dissertation's ontologies, OWL does not use the Unique Name Assumption (UNA), meaning that two different names could actually refer to the same individual. Therefore, in OWL, if not explicitly stated that two individuals are different or equal, they *might* be



(a) Standard representation of individuals.

(b) Standard representation of properties.

Figure 13.: Standard representations in this Masters Dissertation.

the same or different. Figure 13a shows the representation of some individuals in some domain, represented as small gray diamonds.

As aforementioned, properties are binary relations between individuals. Figure 13b shows the notation used to represent properties, through an example with the property hasSibling linking the individual Matthew to the individual Gemma, and the property livesIn linking the individual Matthew to the individual France. Properties are also known as *slots*, in Protégé, or as *roles*, in DL, or as *attributes* in some other formalisms.

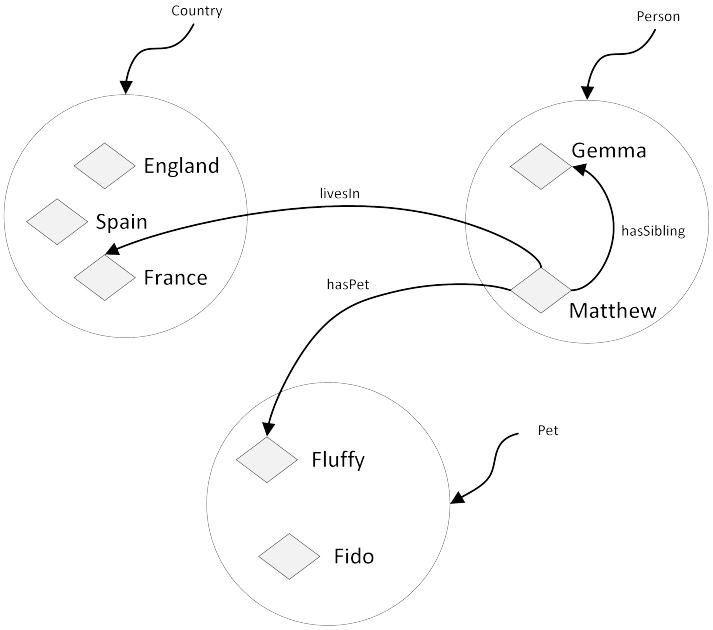


Figure 14.: Standard representation of classes (containing individuals).

In the definition of classes, this Dissertation explicitly referred that classes "represents a group of resources with similar characteristics". Thus, classes are interpreted as groups of individuals. For example, the class **dog** contains the individuals that are dogs in the domain. Classes may be organized into a class-subclass hierarchy, also referred as **taxon-**

omy. Thus, considering the class **dog**, it's possible to infer that all members of the class **dog** are members of the class **animal**, or **dog** is a subclass of **animal**. This hierarchies can be computed automatically by a reasoner - approached in the next subsection. Figure 14 illustrates three classes, in this notation, as circles or ovals: Person, Pet and Country, each one of them with the respective individuals.

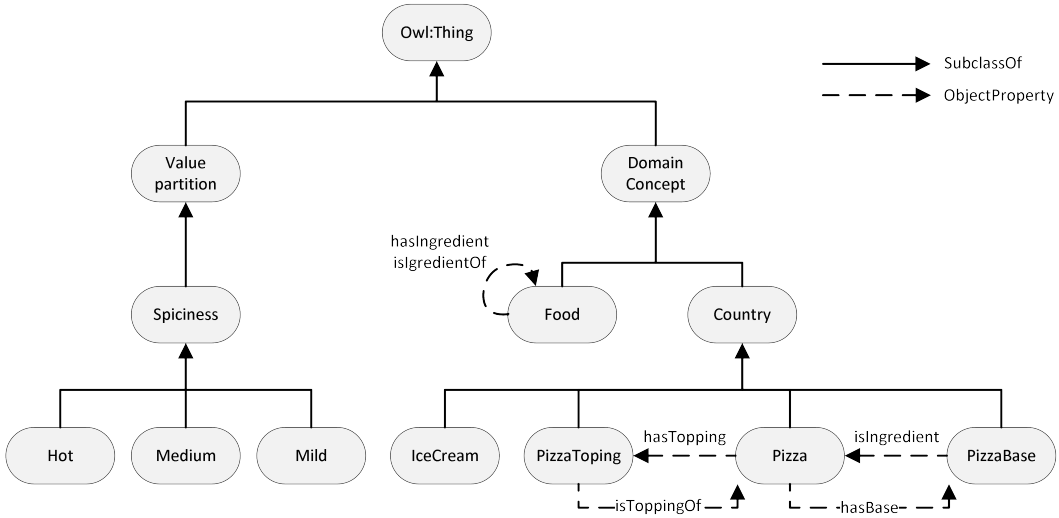


Figure 15.: Standard representation of classes (without individuals).

The defined annotation creates a clear distinction between illustrations that represent ontologies that were integrated with the SeML to accomplish this Dissertation's objectives, which will be illustrated as in Figure 16, and ontologies from other developers or therotical ontologies, that will be illustrated as in Figure 15.

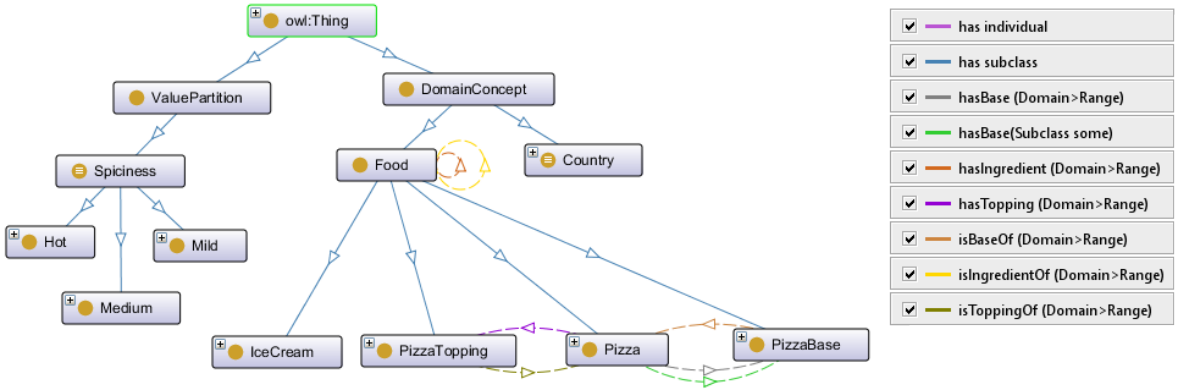


Figure 16.: Standard representation of Ontologies developed and integrated with the SeML, using OntoGraf.

Reasoner and Reasoning Capability

Besides providing precise and unambiguous meaning to domains' descriptions, formal semantics allow the development of **reasoning algorithms**, useful to correctly answer arbitrarily complex queries about the domain. The design and implementation of this algorithms comprises one of the most relevant aspects of DL, because results in highly optimized **reasoning systems**, also referred as **resoners**, that can be used by applications to help them understand the knowledge captured in a DL-based ontology.

Reasoner. A reasoner is a program that infers logical consequences from a list of explicit asserted facts or axioms, and typically provides automated support for reasoning tasks such as classification, debugging and querying.

This subsection provides a brief analysis to some reasoners, evaluating their performance and choosing a suitable reasoner for the ontologies developed in the practical part of this Masters Dissertation. Therefore, the Pellet [CP13] and HermiT [oO] reasoners will be compared, alongside with FaCT++ [prod] and RacerPro [HHMW12], to provide an additional reference point. This set of reasoners has several common characteristics which seem to be essential to the final application. In terms of practical usability, they are accessible through the OWL Application Programming Interface (API) and can be plugged into Protégé, except RacerPro. Since they possess the same interfaces, they can be interchanged, providing development flexibility and allowing their comparison. Also, all the referred reasoners implement *tableux* algorithms which guarantees **soundness** - all the inferred statements are correct - and **completeness** - how many correct statements are inferred. Another important reasoner characteristic is rule support. However, the FaCT++ does not provide SWRL rule support and HermiT is not capable of dealing with SWRL built-ins. A small description of each reasoner follows:

- **HermiT** [oO] - This Java-based OWL reasoner is based on a new *tableaux* reasoning algorithm, and can be integrated into Protégé and Java applications using the OWL API;
- **Pellet** [CP13] - This is an open-source, Java-based OWL DL reasoning engine that supports a majority of the constructs of OWL, including those introduced in OWL2. Pellet is developed and commercially supported by Clark and Parsia;
- **FaCT++** [prod] - An open-source reasoner, which is implemented in C++, supports a large subset of OWL DL and is based on optimized *tableaux* algorithms;
- **RacerPro** [HHMW12] - Renamed ABox and Concept Expression Reasoner, supports a large subset of OWL DL and is implemented with Common Lisp programming language.

Table 2.: Reasoners' performance comparison, in seconds [DCTD11].

(a) Classification performance.

	HermiT	Pellet	FaCT++
GO	6.48	3.41	20.75
NCI	11.75	14.84	11.10
S CT	6,793.76	1,345.65	700.87

(b) Consistency checking performance.

	HermiT	Pellet	FaCT++
GO	0.00	0.27	0.36
NCI	0.00	0.38	0.71
S CT	0.00	16.78	15.3

PERFORMANCE COMPARISON

The performance benchmarked in [DCTD11] uses two typical reasoner tasks, **consistency checking** and **classification**. Consistency checking verifies whether exists a relational structure that satisfies all axioms in a given ontology, that is, if there are no contradictory statements in the ontology. Classification is the computation of the concepts' hierarchy and relations, being this often used as the main performance indicator. The ontology should be checked for consistency and classified regularly, with every modification made to it.

Using several ontologies in the biomedical field, Kathrin Dentler [DCTD11] evaluates the performance of several reasoners (including HermiT, Pellet and FaCT++) according to the previously mentioned reasoner tasks, among other characteristics. These ontologies have the same OWL profile but different sizes in order to evaluate a possible trade-off between ontologies' complexity and reasoners' performance. The ontologies used were GO [GO14], NCI [fBI13] and SNOMED CT [Into6], which are listed in ascendant order of size in tables 3a and 3b. Accordingly with the data provided in [DCTD11], all tested reasoners succeeded in classifying SNOMED CT, and FaCT++ was the only reasoner who classified the NCI faster than the GO ontology. HermiT reasoner had the worst performance when classifying the SNOMED CT ontology, taking approximately two hours to perform the task.

Consistency checking performance per reasoner is depicted in Table 3b. The recorded times are mostly insignificant except for the SNOMED CT ontology, where HermiT outperforms the other reasoners by a large margin. It should be noted that, independently of the used reasoner, the performance decreases with increasing ontology size and complexity.

In [BHJV08], a different kind of analysis is performed. Reasoners are compared according to their **Load** and **Response Time**. Load time is the time required to perform some important preparation before classification, consistency checking and querying ontologies. Response time starts with querying execution and ends when all the results are stored in local variables. In the experiments by [DCTD11], the tested ontologies were implemented according to the same OWL profile. The study by [BHJV08] takes a different approach and tests ontologies implemented in different OWL profiles, which appear in ascendant expressiveness order in tables 5a and 5b.

Table 4.: Reasoners' performance comparison, in seconds [BHJV08].

(a) Load time performance.

	HermiT	Pellet	FaCT++
VICODI	0.889	0.563	0.110
SWRC	0.776	0.518	0.092
LUBM	0.708	0.404	0.072
WINE	1.090	0.835	0.0168

(b) Response time performance.

	HermiT	Pellet	FaCT++
VICODI	0.180	1.145	0.080
SWRC	0.046	0.346	0.056
LUBM	0.046	0.253	0.365
WINE	0.465	9.252	0.607

By looking at the above tables it is clear that, although Pellet and RacePro show better performance at load time, they also show significant less performance than HermiT when queries are executed, especially for higher complexity ontologies. This comparison depends on whether the application frequently operates on preloaded ontologies or has to re-loaded them at each classification, since in the former case load times becomes irrelevant.

FINAL THOUGHTS

Based on the literature, two benchmarks ([DCTD11] and [BHJV08]) and the respective results were analysed to select the reasoners that best fit the requirements of the ontologies here developed. Between Pellet and HermiT, at first sight the HermiT reasoner shows a better performance than Pellet. Another important point for reasoner's selection are the target ontology's characteristics, such as size or expressiveness.

Since reasoner performance is largely dependent on ontologies' and applications' characteristics, when developing an ontology and its application, the various reasoners should be tested regularly in order to decide which one bests suits the ontologies. Because of this, an important characteristic of the selected reasoners is that they should provide an interface via a common ontology API, such as the OWL API, and have a Protégé plugin, which will allow to easily interchange them at development time for performance testing. Regarding the domain and application ontologies it is noted that, if a reasoner must be selected, that reasoner should be Pellet. Firstly, Pellet shows good performance at load and classification time for low/medium theoretical expressiveness ontologies. Also HermiT does not support SWRL rule built-ins, on which the future work of this Masters Dissertation greatly depends.

2.2.3 *Ontology-Driven Software Development*

ODSD is an emerging field of software engineering that has the objective of researching how ontologies and other Semantic Web technologies can contribute to software develop-

ment. In particular, this Masters Dissertation provides a way to model an intelligent motion control system using a semantic technology, the SeML. To do so, a domain ontology for intelligent motion control systems and an application ontology for a specific system within the domain (the coil winding machine) are developed and integrated with the SeML, to allow sound and complete reasoning for consistency checking, domain knowledge agreement and sharing, aiming at the automation of this system's configuration and implementation.

Figure 17 illustrates the integration bridge (adapted from the M3 Integration Bridge [PSA+13, p. 258]) that enables the system's designer to model an intelligent motion control system *using* an ODSD approach.

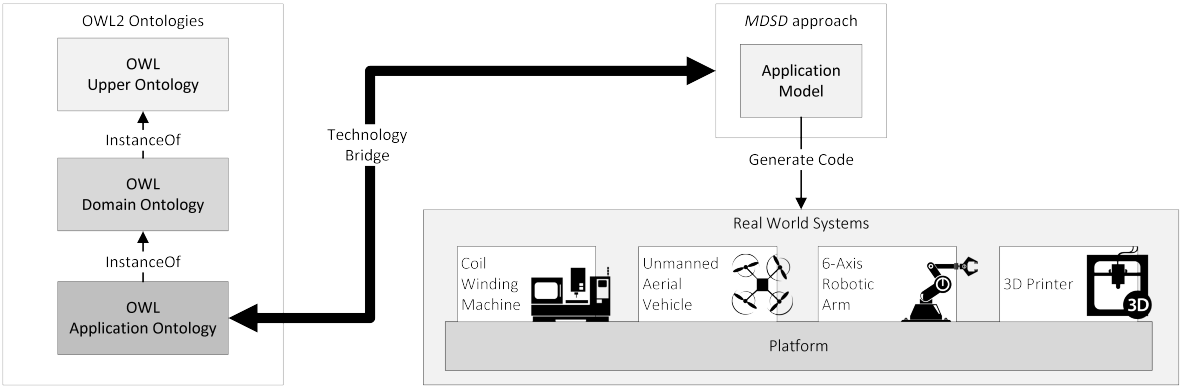


Figure 17.: ODSD foundational technologies' bridge.

This bridge integrates the MDSD approach (Figure 7), which addresses the specification of abstract syntax for modeling languages using formalisms like metamodels and metamodeling languages, and the *ontology world*, which addresses the structural, semantic representation and validation of data using formalisms like OWL2 ontologies.

Advantages of Ontology-Driven Software Development

The advantages of ODSD were deeply explored by Happel and Seedorf in [HS06], but in here are summarized the most relevant ones in this Dissertation's context:

- **Interoperability and flexibility** - Being an ontology a description of domain knowledge and being that knowledge the result of an agreement about the concepts, relations, properties and constraints that constitute an ontology, its extension and reuse are both effortless processes.
- **An improved consistent perspective** - As referred before, *an ontology is not the perspective but a perspective* of a domain. Ontologies incorporate knowledge from multiples perspectives and different levels of abstraction, and, by doing so, they provide an increased consistency between the different viewpoints, facilitating the integration of domains envisaged by MDSD.

- **Unambiguous and semantically-shared knowledge** - Ontologies precisely describe domain knowledge and its semantics. Therefore, the relation of model's elements and their dynamic relations to other models is correctly defined, allowing them to *communicate*.

- **Reasoning capability** - By inferring logical consequences from a list of explicit asserted facts or axioms, and by providing automated support for classification, debugging and querying, reasoners uncover hidden relations between elements from different domain models and facilitates its automatic adjustment.

Divergence from Objected-Oriented Modeling

The last Subsections already referred some of differences between the Semantic Web languages and the object-oriented languages. A deep analysis to this question was made by [KOTW06], but in Table 6 the resume of some key-differences that will be important in this Masters Dissertation is presented.

Table 6.: Relevant differences between Semantic Web languages and object-oriented languages.

	Object-oriented Languages	OWL and RDF
Domain Models	Consist of classes, properties and instances (individuals).	
Classes hierarchies	Classes can be arranged in a subclass hierarchy with inheritance.	
Properties	Properties can take objects or primitive values (literals) as values.	
Classes and Instances	Classes are regarded as types for instances, that cannot be shared with other classes.	Classes are regarded as set of individuals, that can belong to multiple classes, and may be used by a reasoner for classification and consistency checking.
Properties, Attributes and Values	Properties are defined locally to a class and instances can have values only for the attached properties	Properties are stand-alone entities that may be associated with arbitrary classes via their domain restrictions.
	Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.	Open world: If there is not enough information to prove a statement true, then it may be true or false.

3

PROBLEM ANALYSIS AND SOLUTION SCOPING

This chapter provides a brief analysis of the problem this Dissertation aims to address. It motivates the project with the analysis of a intelligent motion control system developed at Jilin University, the coil winding machine, and highlight the arising challenges. Additionally, it sketches the solution approach and describes the expected benefits. Lastly, it defines a number of research hypotheses as well as this Dissertation's objectives.

3.1 THE IMCSP'S PROBLEM ANALYSIS

The foundation of control engineering lies on feedback theory, linear systems' analysis and integration of network knowledge and communication theory. Thus, three relevant conceptualizations can be defined in the context of this Masters Dissertation:

Control System. A control system is an **interconnection** of components assembling a **system's architecture** that provides a desired system's response [DB11].

The referred system's architecture may be a reference architecture if this system constitutes a platform, *e.g.* the IMCSP developed at Jilin University, and the referred system's response is conducted through an automation process.

Automation Process. An automation process is executed by a **control system**, and consists of a process controlled using automatic mechanisms rather than manual.

Therefore, a control system is classified through its *intelligence*, the capability to control a process without help (*e.g.* human intervention), or through its **purpose**, according with desired system's response (*e.g.* an intelligent motion control system is a type of control systems due to its capability to move parts of the machine in a controlled manner).

Intelligent Motion Control System. An intelligent motion control system is a sub-field of automation systems, encompassing systems involved in moving parts of machines, respecting the kinematics and dynamics aspects, in order to accomplish some objective or manufacture some product.

The intelligent motion control system developed at Jilin University was designed to constitute a reference architecture, and so, a platform for diverse intelligent motion control systems. Thus, the final objective of this platform is to have a network of implemented and integrated systems that can be controlled by it. In the Figure 18 is illustrated a group of intelligent motion control systems that are intended to be supported by this platform. From the presented systems, some of them are already implemented, *e.g.* the coil winding machine and the Unmanned Aerial Vehicle (UAV), but at the time of writing of this Masters Dissertation not all of them were integrated with the platform, *e.g.* the UAV and one of them, the robotic 6-axis arm, due to the complexity processes that constitutes its implementation, will be the last system to be integrated.

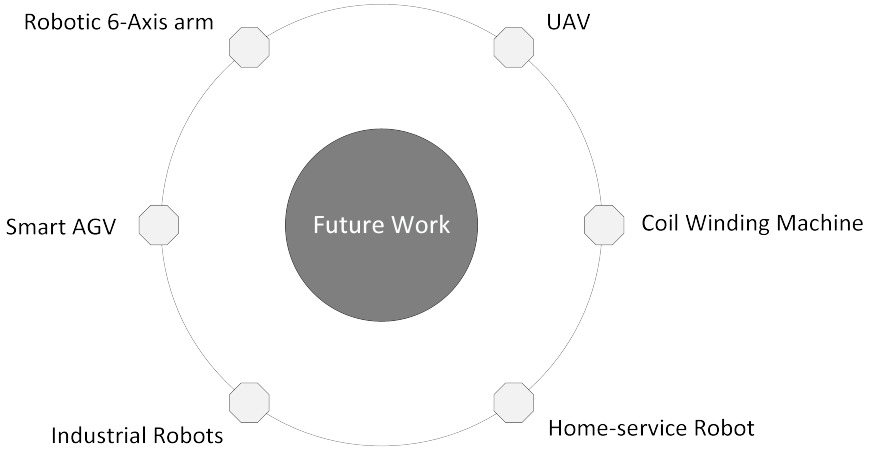


Figure 18.: Intelligent motion control systems to be developed and integrated with the IMCSP.

The reference architecture designed at Jilin University by the IMCSP's engineers comprise a set of features that provides the appropriate platform's flexibility in order to grant a constant *workflow*, independent of the desire behaviours, configurations and/or requirements. Aiming at its objectives, the platform requires to be loaded with the system's configurations (Motion parameters and G-encoder code) and the system's requirements, as illustrated in the Figure 19.

So far, some of the so-called *nuclear definitions* of this domain were explored and the aim of the IMCSP developed at Jilin University was defined. To contextualize, one of the systems developed and integrated in the IMCSP will be used as a case study in this Masters Dissertation, validating the domain and application ontologies, and the system's model. Therefore, its **structure** and **behaviour** should be analysed.

A technology stack seeks to build a bridge between business, technical and operational requirements, while optimizing common quality attributes, such as performance constraints. The IMCSP's technology stack is represented in the Figure 20, with the Application Software in the top-level. This layer integrates all the software dedicated to the user interface, either

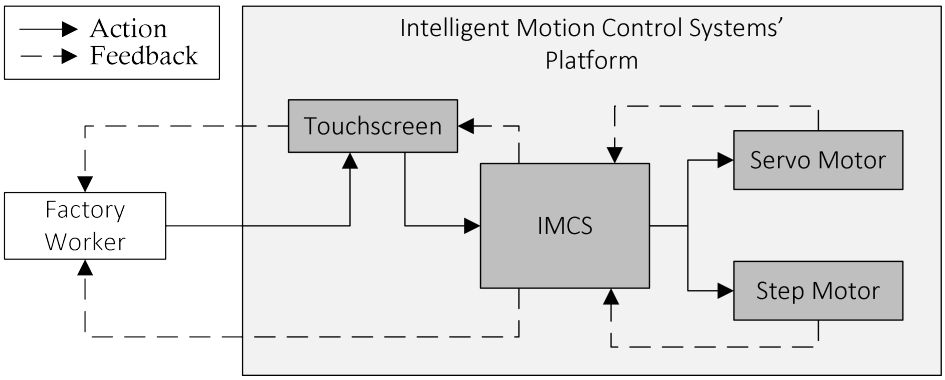


Figure 19.: Overview of the IMCSP's workflow.

to specify system's goals or to provide system's feedback. Sequentially, the communication layer specifies the existent internal and external communication protocols.

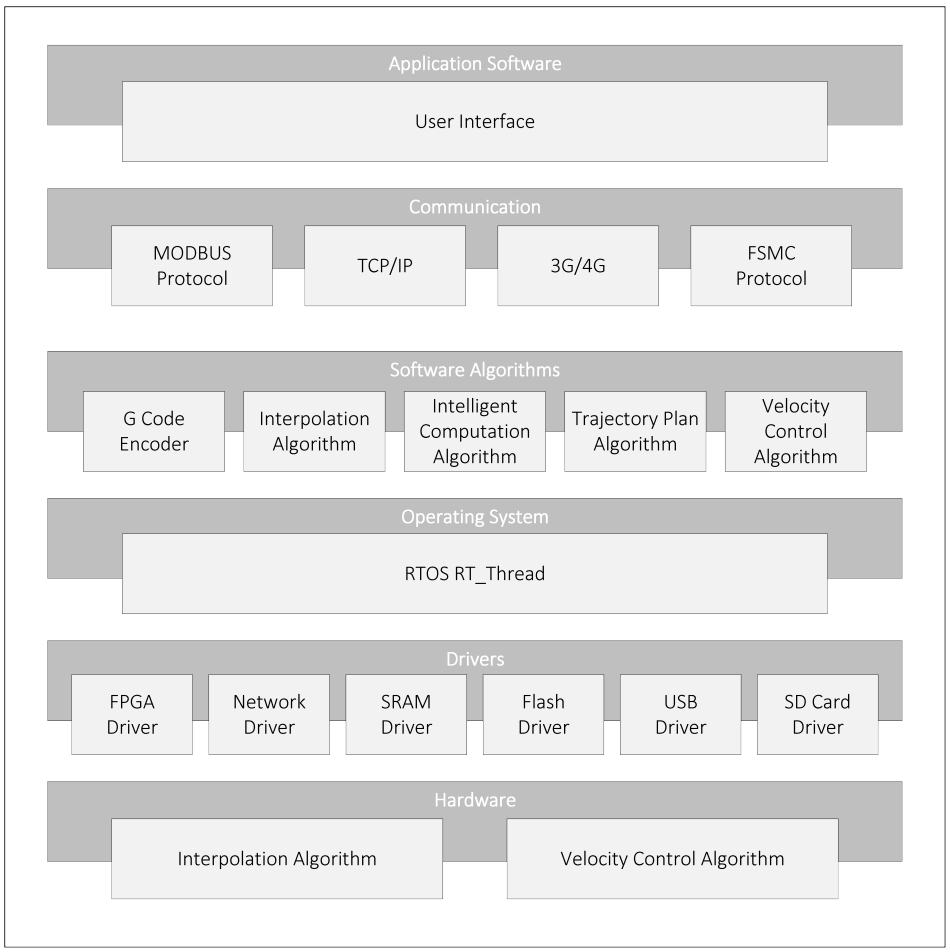


Figure 20.: IMCSP's technology stack.

Intelligent motion control systems is an interdisciplinary domain that ranges in scope from the design of mechanical and electrical components to sensor technology [FGL87], computer systems, and artificial intelligence. To perform a desired task, an intelligent motion control system must ensure feasible motion in the neighbourhood of dynamic and kinematic singularities. To do so, the third layer of the software stack comprises the software algorithms that accurately support the system's actuators control, being this action impacted by the feedback received through the system's sensing components. These algorithms integrate the methodologies to efficiently control this *multi-variable system* (jerk, chord error, tracking efficiency, fee rate fluctuations, etc.) towards a single purpose, which can be the best performance in terms of execution time, accuracy, resources efficiency, reliability, stability, or a trade-off between them, and so, constitute the area with most substantial variability elements. The forth layer represents the operating system, RTOS RT-Thread. The layer below is composed by the device drivers that provide a software interface to each I/O device, allowing the access to hardware functions by the upper layer, the operating system. The last layer is the hardware layer, comprising the system's hardware and the software algorithms that were migrated to hardware, constituting the hardware accelerators intended to increase system's performance.

The hardware architecture of the IMCSP is divided in two main parts:

- STM32F4 - Comprises the application software, operating system, communication management, software algorithms and device drivers;
- FPGA Cyclone - Includes the hardware accelerators, e.g. High-accuracy interpolation algorithm or velocity control algorithms.

These hardware parts and the required hardware logic units are integrated in a single board, which results in the hardware architecture illustrated in Figure 21. The architecture, and subsequently the entire system, are in constant development, mainly due to a huge variety of intelligent motion control algorithms that can be applied to each motion environment, depending on which attributes the user requires from the system. Besides differing in the requirements that provides to the system, these algorithms can be implemented in software or in hardware, depending on the system's requirements. Due to IMCSP's complexity and ever-increasing variability, its integration with a tool that provides customization and automation for the systems' implementation is an indispensable process, beneficial to reduce systems' configuration and implementation complexity, which in turn reduces systems' development time, and subsequently reduces the development costs and enhances systems' variability.

In the next Section, the SeML's workflow is analysed to explore the integration of a domain ontology for intelligent motion control systems and an application ontology for the coil winding machine with the SeML itself, in order to enable the metamodeling of the coil

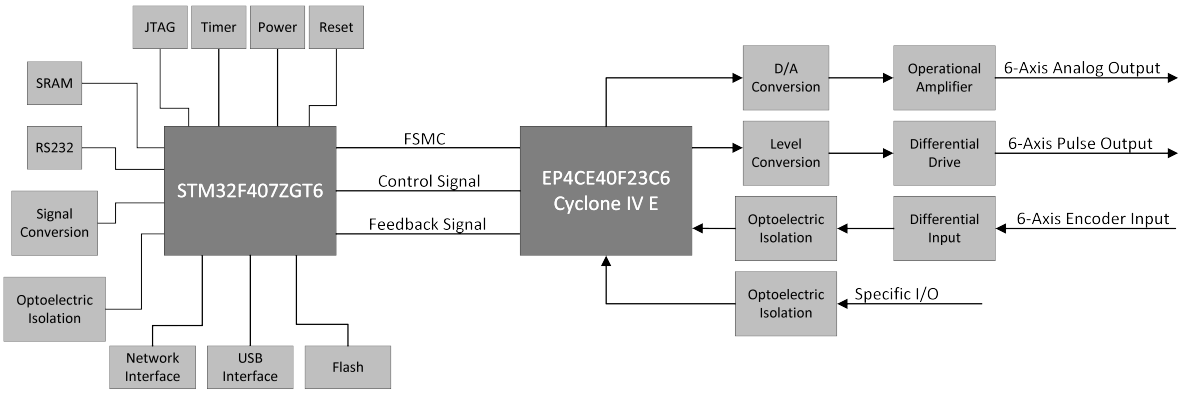


Figure 21.: Overview of the IMCSP's hardware architecture.

winding machine with a semantically-enriched DSML that will provide the aforementioned benefits.

3.2 SOLUTION APPROACH

3.2.1 *SeML as a Semantic Connector*

The nuclear idea of the upper ontology developed and integrated in the SeML by the engine designer (see Subsection 2.1.2) is to establish semantic *links* between different domains, the intelligent motion control systems' domain and the hypervisor's domain, using semantic technologies, in particular, ontologies. Thus, the upper ontology works as a semantic connector, forming a knowledge-base for software/hardware systems and enabling a domain's designer to develop the desired domain ontologies from it, which subsequently will allow the application's designer to develop an application ontology that reflects the mental image that a system's designer conceives for the application when designing the individual model.

Intelligent Motion Control Systems' Domain Ontology

Concerning the modeling of the coil winding machine, a domain ontology for intelligent motion control systems should be developed and integrated with the SeML, through the creation of semantic-relations between the domain ontology and the upper ontology. This ontology will provide a methodology for knowledge representation and reasoning in the domain, allowing the existence of a common vocabulary alongside with clear and concise definitions for it, through the creation of concepts, relations and axioms.

Being the development of a domain ontology one of the most relevant objectives of this Masters Dissertation, it's important to evaluate the diverse possible implementations, select

the methodology with the best ontological performance and, lastly, choose a reasoner that fits the application. Strictly speaking, the behaviour of each existent possibility should be evaluated, adopting a trade-off between reasoning performance and ontological knowledge level, to select the best approach between the following hypotheses:

1. Develop a prescriptive core ontology (complementing the prescriptive core ontology of the SeML) to describe the Robotics and Automation domain, a prescriptive domain ontology for intelligent motion control systems and a prescriptive application ontology for the coil winding machine;

2. Create an instance of the prescriptive core ontology of the SeML and from it developing a prescriptive domain ontology for intelligent motion control systems and a prescriptive application ontology for the coil winding machine;

3. Complement the second hypothesis with a descriptive domain ontology that would assist in the creation of the prescriptive domain ontology. In this case, the descriptive domain ontology requires the development of a descriptive generic ontology from scratch;

4. Follow the third hypothesis, but instead of develop a descriptive generic ontology from scratch, create the descriptive domain ontology as an instance of a recognized descriptive generic ontology, inhering all of its semantic-knowledge.

In Chapter 2, a methodology to follow during the practical part of this Masters Dissertation was defined and the reasoner that fits the ontologies to be developed was established. After doing so, the last hypothesis is now choosing the approach that provides a complete, concise and unambiguous knowledge base to *build* a model for the coil winding machine, aiming at automate its configuration and implementation.

IMCS Application Ontology

The clear, concise and correct definition of the concepts, relations and axioms in the IMCS domain is required by the SeML's *shapeshifting* process to enable the creation of a model for the coil winding machine. The workflow of the SeML is represented in Figure 22, and identifies the system's code implementation as the main goal. Firstly, the developed application ontology, already semantically-connected to the domain ontology (and subsequently to the upper ontology) is integrated in the SeML. The second step is to use keywords, constraints and semantics provided by the ontologies to model the coil winding machine. Lastly, the SeML generates the system's code, accordingly to the system's behaviours, configurations and requirements provided by the system's designer when developing the application's model. The application ontology is essential to this last step, due to the fact that it contains

the semantic knowledge that enables the SeML to know **how to** and **where to** change the application's code.

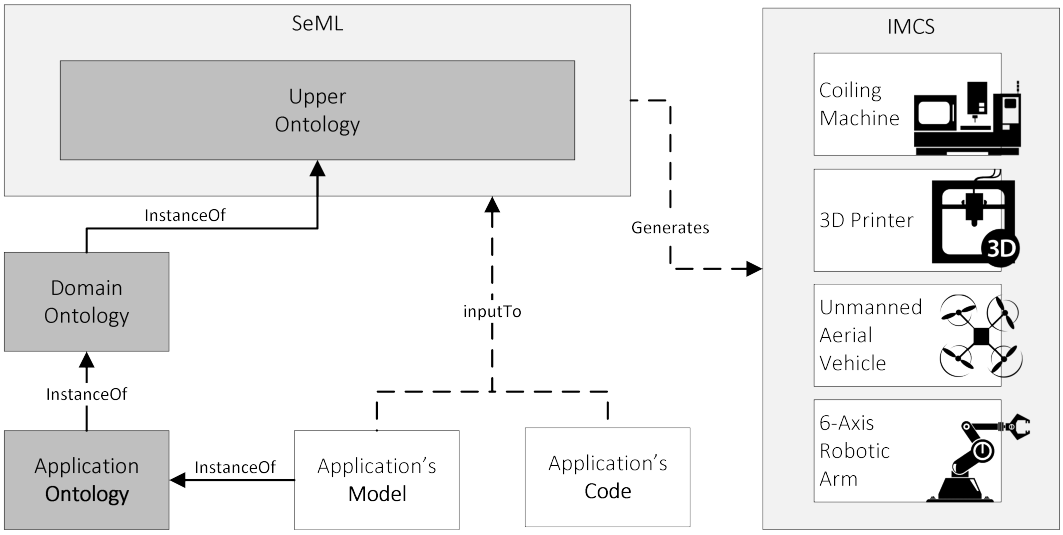


Figure 22.: Overview of SeML's workflow.

3.3 Masters Dissertation's Objectives

The previous section introduced the solution approach and specified four hypotheses that can be suitable to solve the issues that this Masters Dissertation addresses. Starting from there, a list of concrete objectives to aim at can be defined. These objectives also serve to scope and structure the analysis of related work in the next chapter as well as define the case study.

Contribute to the SeML's development

An ontology is a branch of metaphysics concerned with the nature of being and relations of a specific domain. Unlike taxonomies, which provide only a set of vocabulary and a single type of relations between concepts, an ontology provides an exhaustive set of relations, constraints and rules. Therefore, to develop a foundational ontology for software/hardware systems, as the SeML's upper ontology, engineers need to be aware of the essential notions and viewpoints during the domain analysis, preventing ambiguity in the ontology's design. Thus, a group of six ESRG-OT's students embodied a particular team responsible to provide a methodology for knowledge representation and reasoning in the software/hardware systems' domain, culminating in an upper ontology for the referred domain. The first objective of this dissertation is, along with the referred team, contributing for this ontology's development, and so, to the development of the SeML itself. The supervisor of the referred group is Miguel Abreu, and the remaining elements are João Alves, José Martins, Miguel

Macedo, Pedro Lopes and Ricardo Teixeira. The upper ontology's development constitutes an iterative process, and thus, this ontology was continuously updated and upgraded during the development of this Masters Dissertation.

DOMAIN ONTOLOGY'S DEVELOPMENT AND SEML'S INTEGRATION

Concurrently with the development of the upper ontology, the intelligent motion control systems' domain ontology will be developed. Since the upper ontology will be constantly updated and upgraded, the domain ontology will follow this state of metamorphosis too, allowing the correct relation between these two hierarchy levels. Towards the definition of a domain ontology that allows the representation, reasoning about, and communication of knowledge in the intelligent motion control systems' domain, this dissertation must focus on the related work in this field, particularly in the already developed ontologies, opening the spectrum of domain's perspectives as well as their concepts, attributes, constraints, and relations. The development of the domain ontology will be complemented with the merged methodology establish in the last chapter, adjusting the activities and techniques that support its development and exposing inappropriate and inconsistent modeling choices.

APPLICATION ONTOLOGY'S DEVELOPMENT

The application ontology will be developed by the application's designer, in order to specialize the concepts from the intelligent motion control systems' domain ontology with application-specific variants. The application's designer is the role of this Dissertation, assisted by Professor Quangang Wen, a specialist in the intelligent motion control systems' domain.

This ontology is a key technology in this Masters Dissertation, because even being only a specification of a (much complexer) domain ontology, *builds* an interface between the SeML and the system's designer (see Subsection 2.1.2), enabling the development of the system's model and the automation of system's code configuration and implementation. The system's designer is also Professor Quangang Wen, which leads the IMCSP's development team at Jilin University, composed by diverse sub-teams, each one responsible for a specific research field that theoretically composes the coil winding machine and the IMCSP's project.

APPLICATION'S MODEL USING THE SEML

Concluded the integration of the domain and application ontologies with the SeML, the following objective is the modeling of the already implemented and tested coil winding machine. In order to accomplish such complex objective, the system's designer will use the SeML to develop the coil winding machine's model, while assisted by the semantics provided by the upper, domain and application ontologies.

validation of the coil winding machine's model

The final objective of this Masters Dissertation is the automation of the coil winding machine's configuration and implementation. After accomplishing this objective, the code generated by the SeML must be validated, in order to verify if it respects the model built by the system's designer. To do so, the following task must be accomplished:

1. Verify if the code contains the model's specifications - System's behaviours (accordingly with system's requirements), system's interfaces and system's properties;
2. When employed in the coil winding machine, verify if the system's generates a viable and desirable final product - it must be precisely identical with and without the SeML integration in the system's implementation process;
3. Verify if the software modifications required to integrate the system's code and if the SeML do not introduce any overhead in the system;

On the assumption that all the above conditions are respected, the coil winding machine's model is validated, and subsequently, the application and domain ontologies are validated too.

seml's validation at domain-level

Finally, if all the above objectives are successfully accomplished, the SeML is partially validated. In order to achieve a full validation, the SeML should verify the same behaviour with its application on a different domain, the hypervisor's domain. This objective rely on ESRG-OT's teamwork, where the strengths of individuals and the support of ESRG-OT's members are directed towards meaningful goals.

plan and write this masters dissertation

The *roadmap* to complete a Masters of Science degree contemplates the writing of a Masters Dissertation. To accomplish this objective, such task was analysed and planned in a thoughtful way, scrutinizing every detail of its elaboration in a plan where the Masters Dissertation structure was carefully delineated through all the chapters, sub-chapters, typographical conventions, etc.

4

ANALYSIS OF RELATED WORK

This chapter thoroughly analyses and compares related work in each of the domain problems belonging to the identified sub-objectives of this Masters Dissertation. Therefore, a perspective of how relevant the existent upper ontologies were in the process of developing an upper ontology (the SeML's core ontology for software/hardware systems) is provided, and subsequently diverse well-structured and well-defined domain ontologies for the robotics domain that were conductible to the development of the domain ontology are explored in great detail.

4.1 REVIEW OF UPPER ONTOLOGIES

An ontology is, summarily, an aggregation of trivial assertions about the world, aiming at describing things that the human mind can understand but computers can't, in order to provide a knowledge foundation to this machines and thus expect them to act and *think* as humans, *videlicet* shaping them as intelligent agents that act through artificial intelligence processes. In Figure 23, similarly to Giancarlo Guizzardi in [Gui07], this Dissertation illustrates where ontologies intervenes in the process of translating an existing term to an existing object, concerning the explanation that an ontology shouldn't focus in the terms but in the way this terms can be represented, forming concepts through a computable language (*e.g.* OWL2).

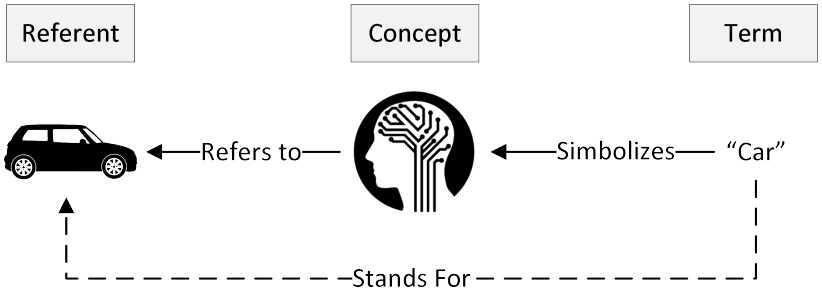


Figure 23.: Relation between terms, ontologies' concepts and real-world referents.

Subsequently, an Upper Ontology is a semantically-enriched database structure that provides a knowledge representation agreement between its instances, ensuring clear and consistent domain conceptualization, granting correct correlation between classes and subclasses (instances), expressing knowledge through a computable language and deducing new *facts* from existing *facts* and *rules* (inference mechanism). Therefore, an upper ontology works in the conceptualization of domain terms and rules, allowing it to replace human intelligence by constituting a foundational knowledge base that is easily expandable (and thus, is able to follow the real world knowledge expansion), language independent, machine understandable, concise, consistent, precise and unambiguous. Such characteristics are essential to develop an agreement about the real-world assertions included in an upper ontology, and thus enable an agreement between every instance that is connected to it. By doing so, an upper ontology provides:

- Guidance for how to *place* information in relation to other information in the domain;
- A source of structure and controlled vocabularies helpful for disambiguating context;
- A more effective basis for information extraction or content clustering, and thus, a structure for browsing within a domain;
- An explicit scope, definition, language and semantics of a given domain, and therefore a basis to reason and/or infer over it.

Despite the huge number of developed upper ontologies, perspectives and subsequent differences between them, there are some general concepts and relations that constitute an agreement on many issues [CJ99]:

- There are *objects* in the world;
- Objects have properties or attributes that can have values;
- Objects can exist in various relations with each other;
- Properties and relations can change over time;
- Events can occur at different time instants;
- Objects participate in processes, and that processes can occur over time;
- The world and its objects can be in different *states*;
- Events can cause other events or states as effects;
- Objects can have parts.

Lastly, to finish this introduction and motivate its *use*, this section makes an analogy to software development methodologies nowadays. **Reuse** is an absolute necessity and part of the best practices in any modern software development (otherwise, every application would imply the development of its own operating system system, device driver, etc.) to avoid high development cost processes. Accordingly, during an information system design, *reuse* should be maintained as a fundamental best practice. Thus, when developing a domain ontology, the respective ontology should be mapped to a core or generic ontology.

Naturally, a full coverage of all existing generic ontologies is not the focus of this Masters Dissertation, and thus this section converges the analysis' concerns on the proposals that are related to the problem of establishing ontological foundations for modeling languages, or that contain semantic-knowledge useful for the development of a *stable* upper ontology for hardware/software systems. Therefore, in the next subsections, the BWW, UFO and SUMO ontologies are analysed in order to explore the development of foundational ontologies through the last decades, aiming at clearly identify the insufficiencies, *strengths* and/or *weaknesses* of the existing ontologies to emphasize this Dissertation's contribution to the semantic world. Lastly, an upper ontology is established, to be used as a semantic connector to the descriptive domain ontology, which development will later be explored.

4.1.1 *Bunge-Wand-Weber*

The BWW ontology, which dates from around 1990, is until now the most discussed approach to ontology-driven conceptual modeling, with over 100 publications in various contexts. Aware of the basis of conceptual modeling, Wand and Weber adapted the two-volume "treatise" on formal ontologies by Bunge [Bun77, Mar80], a physicist and philosopher of science, and developed the BWW. This process was documented in a series of articles [WW88, WW90a, WW95].

The BWW ontology advocates that an "information system is an artificial representation of a real-world system as perceived by humans" [WW88], in order to make a distinction between the representation and the perceived system. Accordingly to Wand and Weber, an ontology should be the technology used to model an information system due to its ability to describe the structure of the world, and thus to encompass both the statics (structure) and the dynamics (behaviour) of it, through a 3-phase process described in Figure 24. In the analysis phase, a perspective of the world is forged into a formal model of its perception, resulting in the so called **model of reality**, which is equivalent to a descriptive application ontology. Even being only a human perspective of the real-world and not a way to define its constituents and its relations, this *transformation* should provide a structured and unambiguous model of reality. Later, in the design phase, the model of reality is *transformed* to a **model of representation** (model of the information system), constituting an interface layer

between the model of reality and the system's implementation. Lastly, the model of representation is *transformed* into a realization of an information system, and so, a real-world system.

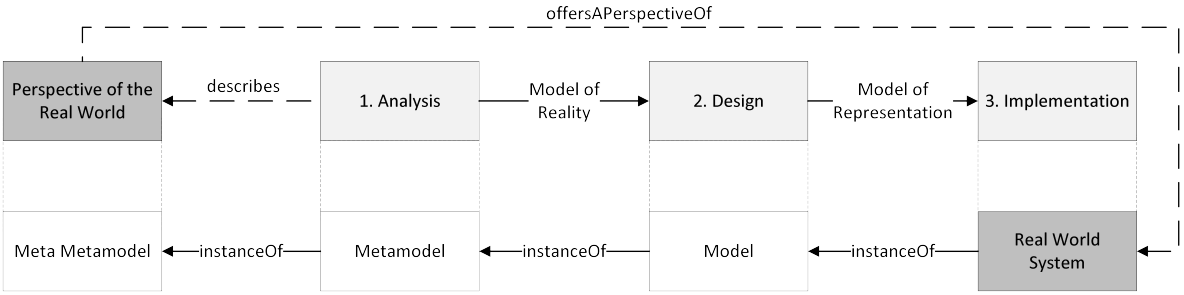


Figure 24.: The transformations of BWW during an ODSD process.

In the perspective adopted by Wand and Weber, "the world is made of things, that have properties by which they are known". A *thing* is modelled by a set of **functions** that assign **values** to its **properties** (referred as functional schema). This properties can be agglomerated as sets (that should represent its structure) to constitute a **state** of the referred *thing*, allowing the model to describe the dynamics of a thing through a state machine. Two different *things* are not independent if their history (set of states of thing during a period of time) demonstrates some dependency, namely, if the history of one of them changes with the presence/absence of another. Therefore, a **system** is a *thing* composed by sets of interacting *things*, and thus has its own state machine that is dependent on the interactions between the state machines of things that compose it. Besides that, a system's state change can be implied by an external *thing* (a *thing* that is not part of the **things** that compose the system, but is part of the *things* that compose the **environment** of the system), constituting an external event. Regarding possible systems' states, two assumptions were created [WW88]:

- Stability - A state change happens if and only if the system is in an unstable state;
- Unique response assumption - A system in an unstable state changes to a stable state uniquely defined by the unstable state.

Taking into account that develop a formal model for all phenomena associated with information systems is an impossible task, Wand and Weber focused in the deep structure phenomena of the internal view of an information system [WW95], as illustrated in Figure 25.

The external view of an information system models the system with an external perspective by treating it as a black box that provides specific necessary services to an organization, affecting users and evoking managerial concerns. This view allows researchers to define system's requirements, process stakeholders and organizational issues. Divergently from

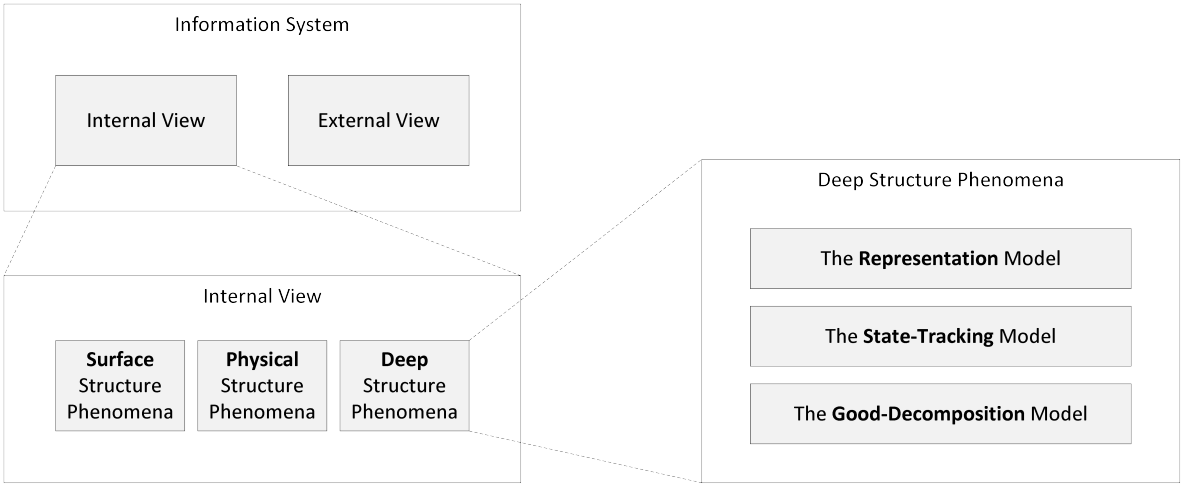


Figure 25.: The domain of BWW models.

this perspective, the internal view takes system's requirements as fulfilled and concerns about the characteristics that the system must have if they are to fulfil the referred requirements, and so, it regards the structure and behaviour of the system to provide the required functionality. The later view can be branched into three distinct phenomena - deep, surface and physical structures. The surface structure phenomena represents the way the system is presented to its users. The second phenomena, deep structure phenomena, expresses the *meaning* of the real-world systems that the information system pretends to model. The last, physical structure phenomena, manifests the nature and physical structure of the technology used to *build* the system.

Advocating that the core of information systems' design resides in ensuring a good description of the deep structures of information systems, by providing the meaning of the real-world systems that they pretend to model, the BWW approach focused in these phenomena and provides three formal models, illustrated in Figure 25, based on two premises:

- Physical-symbol system - A physical-system possesses sufficient properties for it to be able to represent real-world meaning;
- Representation - Information systems provide an artificial representation of real-world systems;

The representation model is constituted by two evaluation criteria, the *completeness* and *clearness*, which evaluates if a designed grammar is completed (and so enable to model any real-world phenomenon) and the capability of its constructs to have a one-to-one correspondence with one of the developed ontological constructs, respectively. The ontological constructs that constitute the representation model are illustrated in Figure 26, as a result of a concise analysis to [WW95, Table 1].

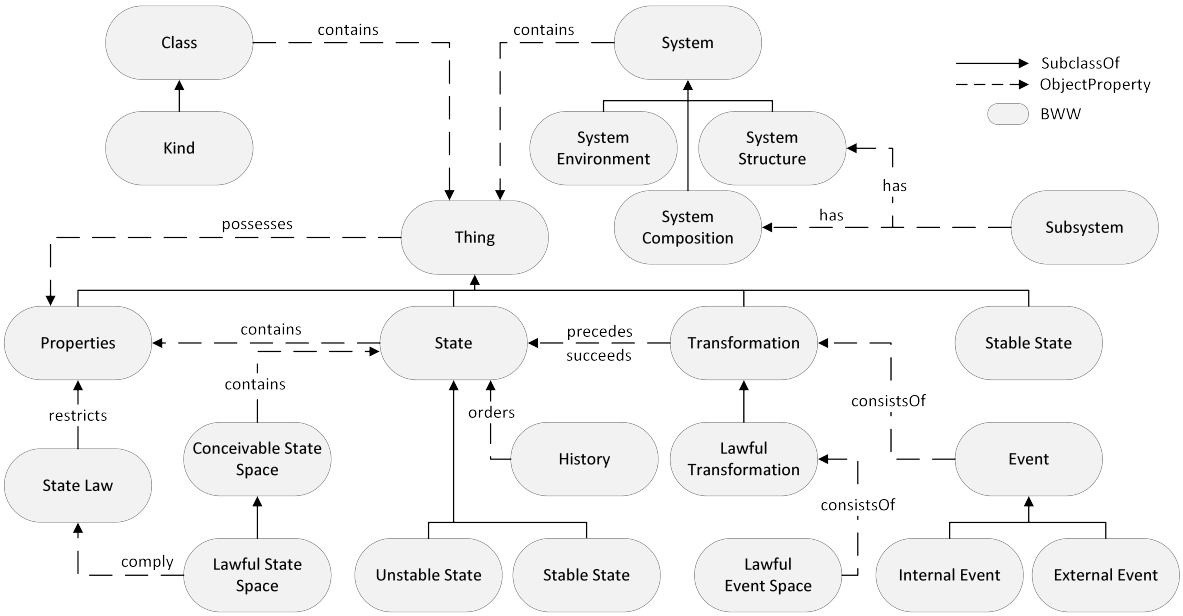


Figure 26.: BWW constructs and summarized ontology.

The second model, the state-tracking model, pretends to specify four requirements that any information system must satisfy in order to enable state tracking [WW88]:

1. **Mapping** - A real system's state has at least one correspondent state in the model of representation;
2. **Tracking** - The model of representation's laws (mappings that might change the state of a system) replicate the real system's laws;
3. **Reporting** - Real system's events should have a reflection in the model of representation;
4. **Sequencing** - The real system's state machine should be respected in the model of representation.

The last model, The good-decomposition model, specifies three conditions that an information system must hold in order to allow the reliability of a decomposition process:

1. **Determinism** - A decomposition is reliable if, for every subsystem, each external event at the system's level is defined either as specified external event (manifest the influence of the environment on the subsystem), or as well-defined internal event (state changes that occur internally to a system as a result of an external event);
2. **Minimality** - A decomposition is reliable only if every subsystem's descriptions are defined through non-redundant variables (a redundant variable is a variable that is never *used* during the lifetime of the subsystem).

3. ***Losslessness*** - A decomposition is reliable only if all emerging variables in the system (state variable describing a property of the composite system) result from a function of properties of, at least, one subsystem in this process.

Figure 27 illustrates an overview over the most relevant concepts in the BWW ontology. By presenting Thing as a pivotal concept, this ontology establishes that "the world is made up of things" [WW90b]. This development arises with the OWL too, where "every individual in OWL worlds is a member of the class owl:thing" [Gro16b].

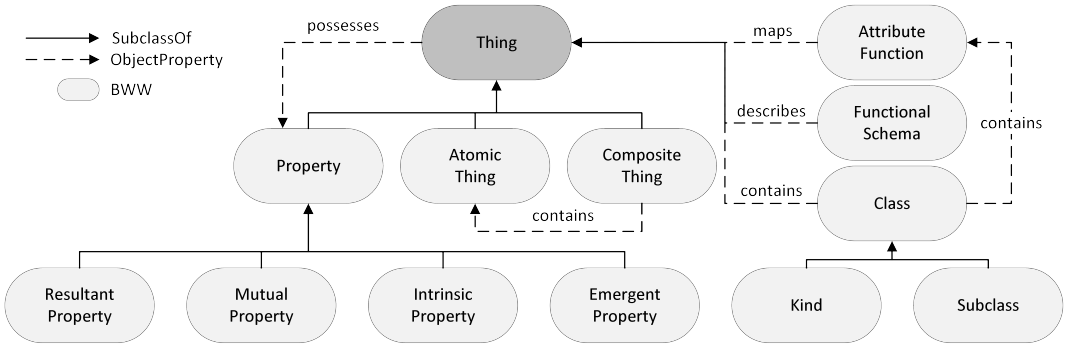


Figure 27.: A selection of structural concepts in the BWW ontology.

The compromise adopted by Wand and Weber in branching *Thing* into *Component Thing* and *Composite Thing* is an appealing decision for software/hardware ontologies' designers, because it allows such ontologies to follow a structural design pattern as the **composite design pattern**. Thus, in the presented ontology, *Composite Thing* can form a group of *Things*, which may be made up of other *Things* (or a single *Thing*), providing a tree structured architecture to represent part-whole hierarchies. Nevertheless, both this decision and the decision of using a state machine to represent the behaviour of the system were initially adopted in the SeML' upper ontology, as explained in detail in the next chapter. This research agrees with Wand and Weber about the relevance of the Deep Structure Phenomena when modeling information systems, and so, the development of the ontologies should be made by focusing in all the constituents of the internal view, due to the importance that the specification of the physical and the surface structure encompasses when modeling software/hardware systems.

4.1.2 *Unified Foundational Ontology*

The UFO is the result of the synthesis process illustrated in Figure 28, which merged two foundational ontologies, General Ontological Language (GOL)/General Formal Ontology (GFO) and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE), since it derives from the referred ontologies but focus on a different domain, business [GW04], in an attempt to resolve each ontology's shortcomings and insufficiencies.

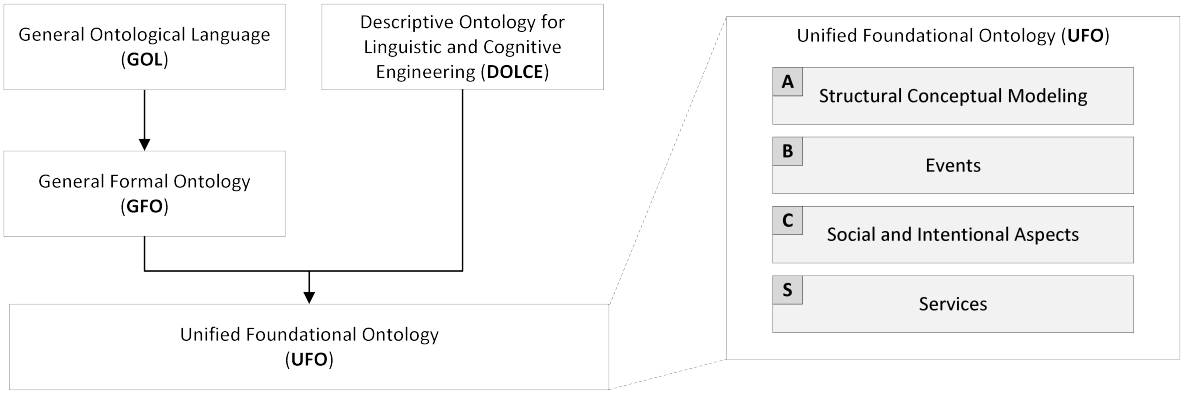


Figure 28.: The history of UFO.

GOL/GOF

Advocating that "every domain-specific ontology must use as a framework some upper-level ontology which describes the most general, domain-independent categories of reality" [DHHS01], GOL was developed with an upper-ontology as its foundational base of knowledge, providing reasoning capabilities about time, space, inheritance, instantiation, identity, process, event, attribute, relation, and so on, thus being considered a descriptive ontology.

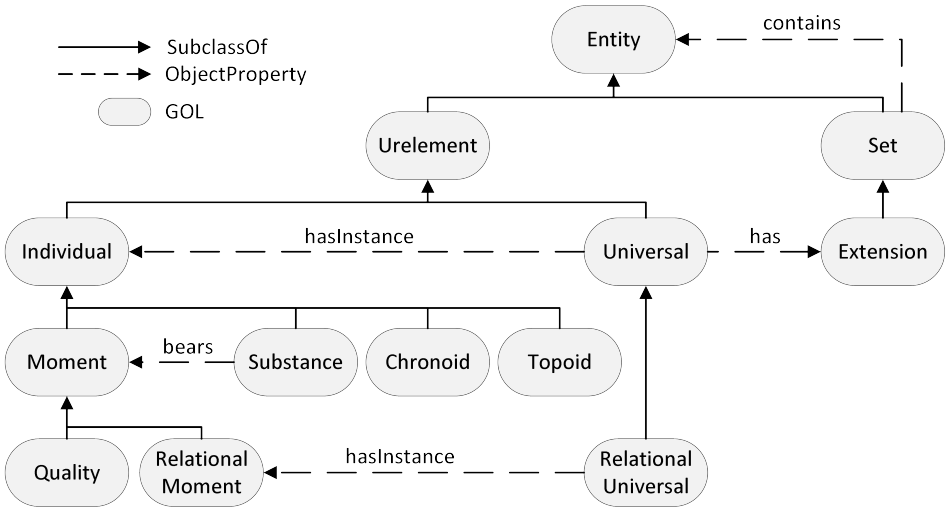


Figure 29.: Basic concepts of GOL (adapted from [GHW02b]).

As illustrated in Figure 29, GOL branches the entities from the real-world into sets and urelements, considering that everything that is not a set is a urelement. Subsequently, urelements are divided into individuals and universals, in order to distinguish between something that is not a set and exist within the confines of space and time, from something that is not set and can be instantiated simultaneously by a multiplicity of different individuals that share some characteristics. Individuals may be substances, moments, processes, chronoids, or

topoids. Similarly with thing in the BWW ontology, the concept substance in GOL is based in the Aristotelian idea of substance [Ack88, p. 7]:

> "A substance - that which is called a substance most strictly, primarily, and most of all - is that which is neither said of a subject nor in a subject, e.g. the individual man or the individual horse. The species in which the things primarily called substances are, are called secondary substances, as also are the genera of these species. For example, the individual man belongs in a species, man, and animal is a genus of the species; so both man and animal are called secondary substances."

Therefore, a substance "is that which can exist by itself" [GHW02a], an so its independent from other individuals, *e.g.* me, you, the sun or a golf ball. Contrarily, a moment can only exist in other individuals (*e.g.* passions or an electrical charge). Lastly, chronoids and topoids are instances of the universals *time* and *space*, respectively.

DOLCE

Although being part of the WonderWeb library [Gua], DOLCE is not considered a descriptive generic ontology, but a starting point for comparing and elucidating the relations with other modules of the library, and also to clarify the hidden assumptions underlying existing ontologies or linguistic resources, such as WordNet. Nevertheless, this ontology is analysed in this Masters Dissertation due to its relevance in the development of the UFO and because it constitutes one of the most thoroughly researched contemporary generic ontologies.

DOLCE "aims at capturing the ontological categories underlying natural language and human common sense" [Gua01a], focusing on specific domains. Therefore, a fundamental ontological distinction between universals and particulars is clarified in this ontology by taking the relation of instantiation as a primitive (particulars are entities from which no instance is created and universals, contrarily, are entities that can have instances). Besides this foundational ontological choice, DOLCE has another two core ontological commitments that allowed it, partially illustrated in Figure 30, to become a standard [MBG+03]:

- Enduring *versus* perduring entities - An endurant lives in time by participating in some perdurants;
- Multiplicative approach - Different entities can be co-located in the same space-time.

The final objective of this ontology, as a formal tool for the semantic integration of data, is to integrate knowledge from different domains like computational linguistics, agriculture, medicine, cultural resources, banking, mobile robotics, etc., covering all possible subjects that exist. To do so, and in order to ensure a constant quality level during its maintenance and expansion, DOLCE was suggested to divide the foundational ontology into submodules of domain knowledge [Gua01b].

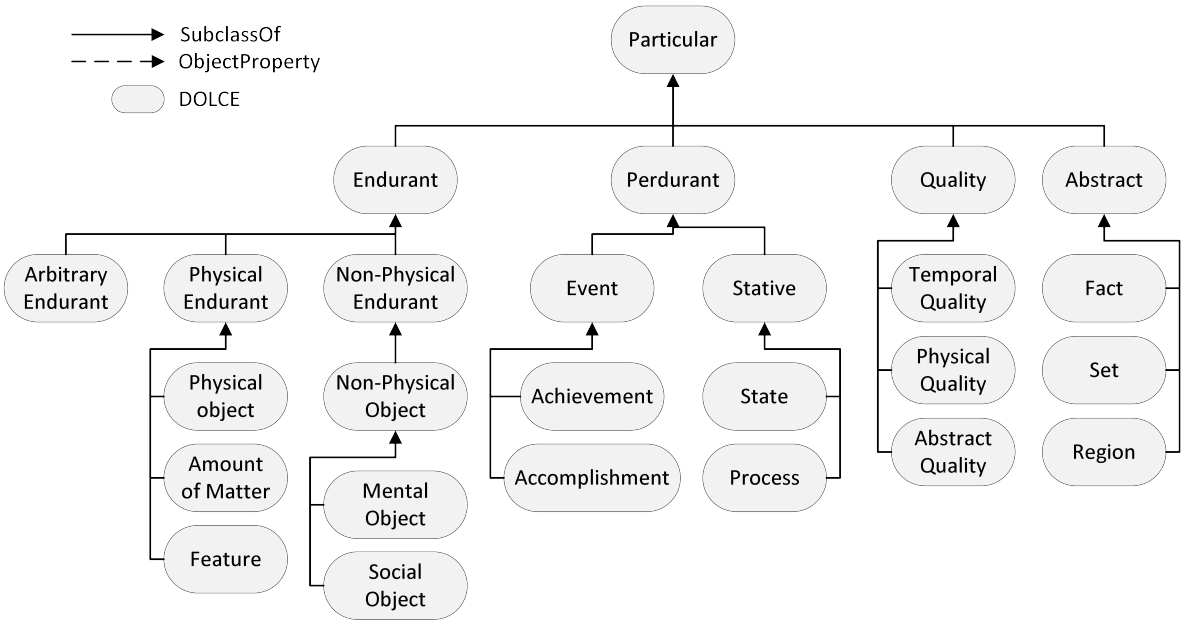


Figure 30.: Basic concepts of DOLCE's taxonomy (adapted from [MBG+03, p. 13]).

THE EMERGING OF THE UFO

The term UFO has first been used in [GW04, GW05] and only later the ontology was successfully applied in the analysis of several conceptual modeling constructs [GG05]. The UFO was consistently developed after a deep research on theories originating from areas such as Formal Ontology in philosophy, cognitive science, linguistics and philosophical logics, comprising a number of micro-theories to address fundamental conceptual modeling notions, and originating the layered ontology illustrated in Figure 31.

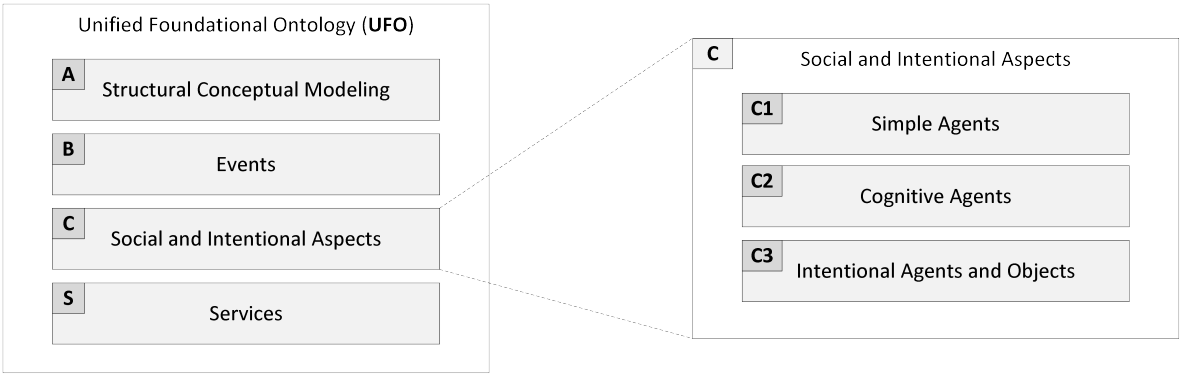


Figure 31.: The architecture of UFO.

The structure of this ontology aims at addressing different aspects of reality, namely [GWAG15]:

- UFO-A - An ontology of *endurants*, aiming at structural conceptual modeling aspects;

- UFO-B - An ontology of *perdurants*, dealing with aspects related with events and processes;
- UFO-C - An ontology of intentional and social entities, which addresses notions such as beliefs, intentions, goals, actions, commitments, desires, social roles, among others;
- UFO-S - An ontology of services, addressing the commitments established between customers and services providers [NFA+15].

Accordingly with the MDA approach (from the OMG [SO00]), a business model is a "computation-independent model" because it should be solely expressed in terms of business language, and not in Information Technology (IT) concepts. Therefore, the development of an upper ontology for business, presents many challenges due to the variability and complexity of the business domain. The analysis of the UFO leverages this research's ontological expertise, and allows the identification of methods that provide the necessary IT-abstraction to develop an upper ontology and a descriptive domain ontology.

Figure 32 illustrates the main concepts in the UFO, and clearly identifies *Thing* as the foundational concept, being "anything perceivable or conceivable [ISO:object]" [GW04], concurrently to the terms OWL:*Thing* and BWW:*Thing*. This concept is then branched into set and entity, creating a XOR relation that represents disjointness of the corresponding class extensions.

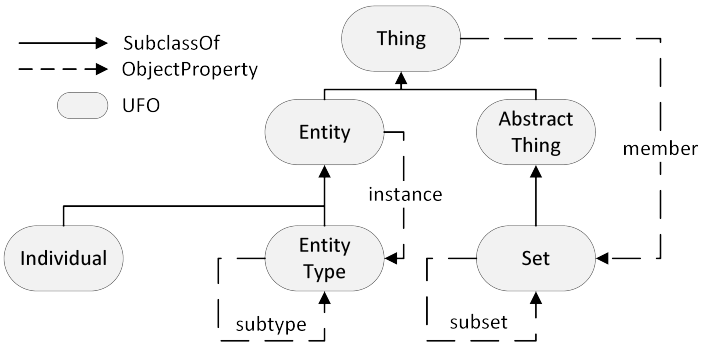


Figure 32.: A selection of UFO's structural concepts (adapted from [GW04]).

In this metaphysical approach, Guizzardi and Wagner defined set as *Things* that contains other *Things*, providing some conflict with the BWW:*Composite Thing* definition. Therefore, this Dissertation proposes a different *Set* definition in the upper ontology (being this notion hidden under an ontology's annotations interpretation mechanism developed in the SeML).

4.1.3 *Suggested Upper Merged Ontology*

SUMO is an open source descriptive generic ontology that has been proposed by the Standard Upper Ontology Working Group (SUOWG), an IEEE-sanctioned working group of

collaborators from the fields of engineering, philosophy, and information science, as a result of merging a number of existing generic ontologies ([Sow00, BGM96, BGM97, J.F84, Smi96, Lab, Ass]), as illustrated in Figure 33.

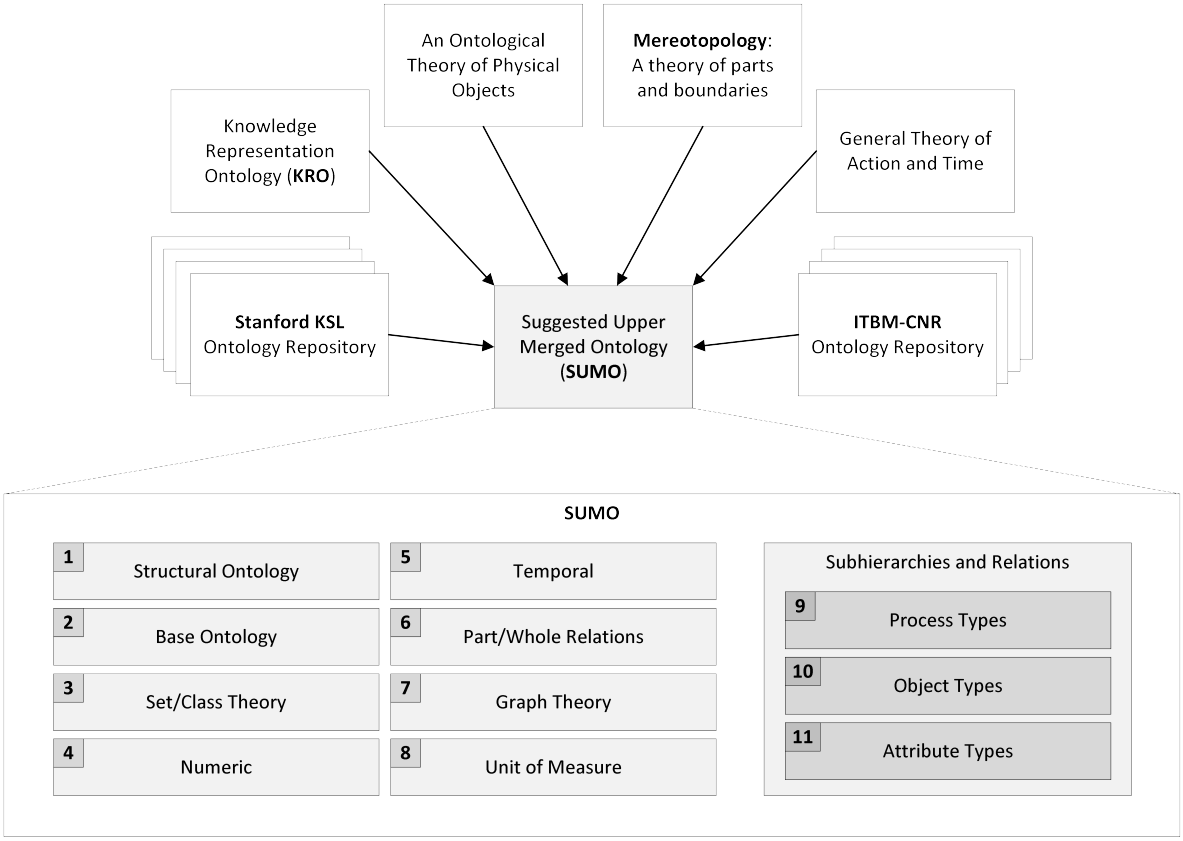


Figure 33.: The origin of SUMO and its architecture's overview.

The referred merging process, grants a huge diversity and quantity of concepts, axioms and relations in the resulting ontology, and thus, during its creation, a stratified architecture was proposed in order to keep it organized in different sections [PNL02], namely into 11 sections with rooted interdependencies carefully documented by Adam Pease in [NP01, Pea06].

Being a generic ontology (see Figure 9), SUMO proposes definitions for general-purpose concepts and acts as a foundation for core and domain ontologies, thus creating a comprehensive and cohesive ontological structure. The referred ontology uses Knowledge Representation Framework (KIF) to embody the semantic knowledge necessary to describe a considerable number of entities and axioms, intended to define the meaning of those entities. Although SUMO was initially developed to clarify the meaning of more specific terms (making them reusable in a large-scale) and to realize a (language independent and easily understandable by a machine [Pea11]) knowledge representation method, at the time of

writing this Masters Dissertation, the combination of all SUMO concepts result in a ontology composed by around 25,000 terms and 80,000 axioms.

Contrarily to what was observed in the BWW and UFO ontologies, SUMO proposes entity as the root node, and physical and abstract as disjointed concepts, Figure 34. Physical is further branched into object and process. Object exists in space and keeps its identity in time (has spatial parts but not temporal stages). Process is a class of instances that happen in time (have temporal stages), meaning that instead of a *perdurantist* perspective, SUMO follows an *endurantist* perspective. Therefore, in this ontology an object can change through the time, keeping its identity. Abstract is further branched into quantity, attribute, set, relation and proposition. As explained in [NP01]:

> "Set is the ordinary set-theoretic notion, and it subsumes class, which, in turn, subsumes relation. A class is understood as a set with a property or conjunction of properties that constitute the conditions for membership in the class, and a relation is a class of ordered tuples."

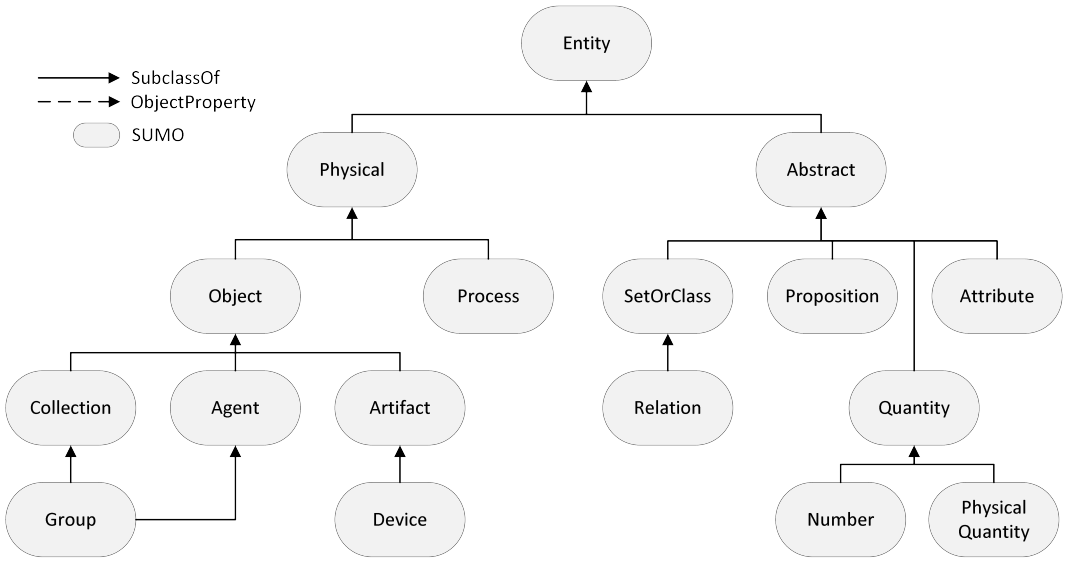


Figure 34.: An elementary SUMO's taxonomy.

During the development of this ontology, and due to the adjustment of its focus, referred by Adam Pease in [Pea11], SUMO increased rapidly in order to cover all possible domains (as any generic ontology should do). Although, instead of maintain a constant abstraction level, it eventually focused at specific domains (*e.g.* hotel/hospitality) in a way that confused the initial well-organized structure. Therefore, this Masters Dissertation will focus on the perspective given by SUMO as a generic ontology and abstract above the specificity of some domains that, despite not relevant to the development of the ontologies, constitute a part of the final architecture of SUMO [Pea06], illustrated in Figure 35.

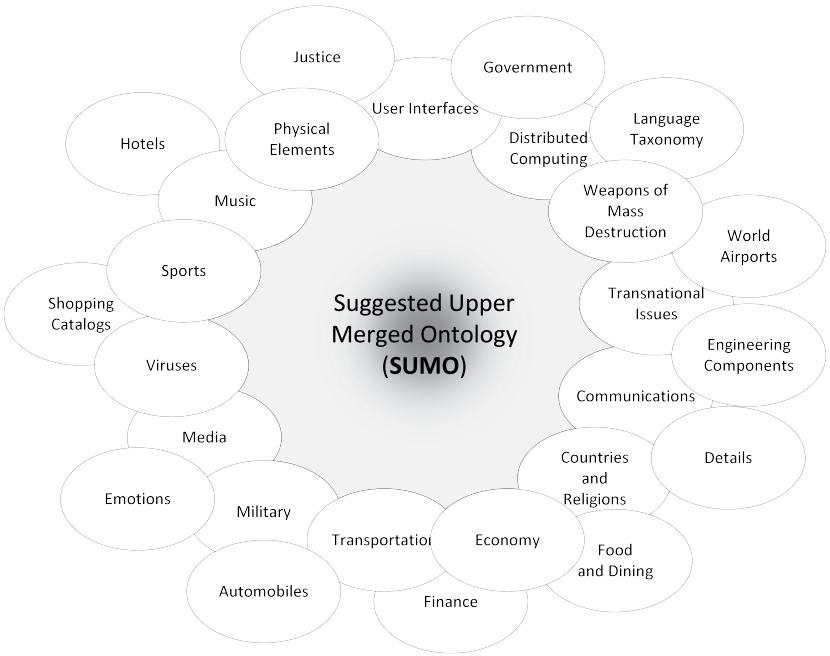


Figure 35.: Domain knowledge added to SUMO after the conclusion of its initial architecture (presented in Figure 33).

As a result of its ontological-roots and as the outcome of a huge research (which in turn was concluded with the integration of domain knowledge in the generic ontology), SUMO provides definitions for general-purpose terms and acts as a foundation for more specific domain ontologies, providing the correct integration of domains, an agreement of real-world perspectives, and also being more *flexible* than the remaining generic ontologies analysed in this subsection.

In this Masters Dissertation, SUMO is the descriptive generic ontology from which the descriptive domain ontology was created. Thus, in the next subsection, the essential ontologies from the **robotics and automation domain** that were useful in the creation of the descriptive domain ontology as an instance of SUMO will be analysed. Besides the flexible characteristic granted by this generic ontology (that was used as an upper ontology - see Figure 9), the process of developing a descriptive domain ontology from a well-defined and widely *used* ontology (as SUMO) allows an agreement on the semantic knowledge inherent to other core/domain ontologies derived from it.

4.2 Review of Domain Ontologies

The last section reviewed generic ontologies, briefly discussing their aim, background, structure and areas of application, which enables the comparison between them and the upper ontology developed in this Masters Dissertation using the HexOntology (see Figure 9), as

illustrated in Table 7. Although the analysis of the explored generic ontologies constitute a relevant research in order to develop the core ontology (which works as an upper ontology to the instantiated domain ontologies), this ontology intends to operate at different abstraction level, with a light-weight theoretical expressiveness and a prescriptive implementable expressiveness, aggregating a variety of characteristics which distinguish it from the analysed generic ontologies.

Table 7.: Comparison between the generic ontologies analysed and the SeML's ontology, using the HexOntology (see Figure 9).

	BWW	UFO	SUMO	SeML
Purpose	Reference	Reference	Reference	Reference
Abstraction Level	Generic	Generic	Generic	Core
Theoretical Expressiveness	Heavy	Heavy	Heavy	Light
Implementable Expressiveness	Descriptive	Descriptive	Descriptive	Prescriptive

Regarding the IMCS's domain ontology development, which in turn (since is an instantiation of the SeML's prescriptive upper ontology) must be a prescriptive domain ontology, the focus of this Dissertation should now be the development of a descriptive version of the referred ontology, avoiding the lack of domain knowledge inherent in the process of directly creating a prescriptive domain ontology. Therefore, as illustrated in Figure 36, a descriptive domain ontology must be developed as an instantiation of a descriptive generic ontology (identified in the last Subchapter as being SUMO), and later must be transformed into a prescriptive domain ontology through a **semantic-refactoring process**.

Ontology's development should aim at the reuse of knowledge, and thus, when concluded, an ontology should be at least reusable in its own domain. In pursuance of this modularization [Obio7], the development of a new ontology should be complemented with the knowledge and perspectives provided from the existing ones, allowing vocabulary agreement and knowledge sharing between two different applications in the same domain.

In the domain of robotics and automation, ontologies have been applied for robot description and operation, generally as a knowledge base used to describe and characterize the domain, the tasks and/or the surrounding environment. This domain contains diverse and rich hierarchies in terms of knowledge, and most of the ontologies about it convene on the following sub-domains:

- Environment - Characterization of the environment surrounding the robot as a repository of objects [PSB+04] or as knowledge about the location in which their are moving into [BS04];

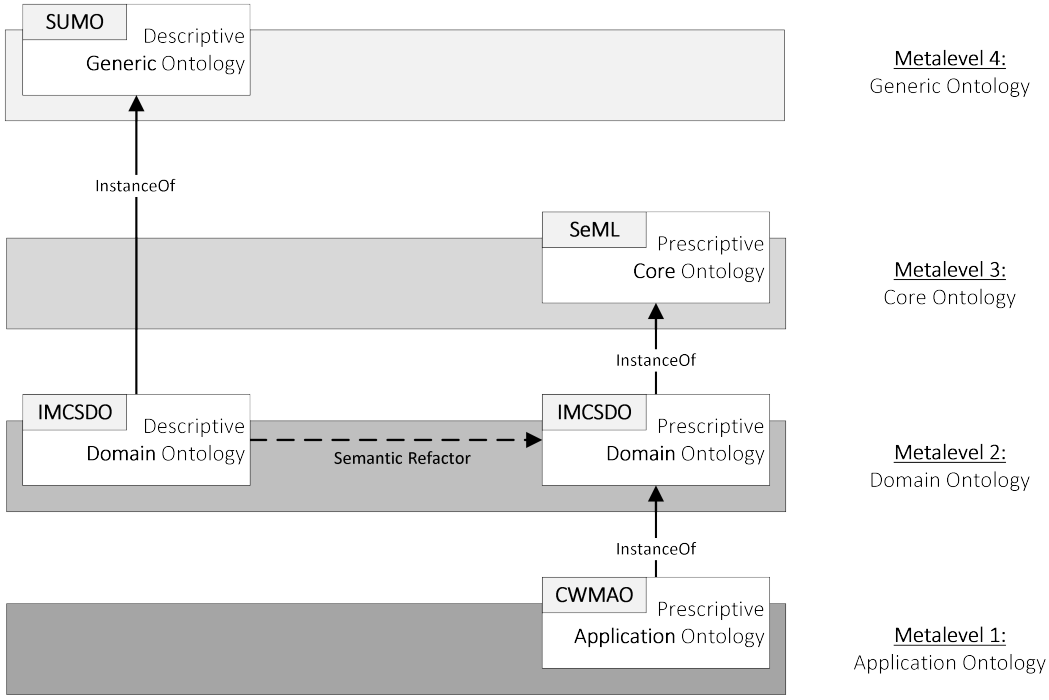


Figure 36.: Contextualization of the developed ontologies, using the alternative perspective provided by the HexOntology (see Figure 10).

- Reasoning about actions/tasks - Description of tasks and actions through a stratified knowledge structure [BAM+04], to describe learning methods and heuristics [Eps04] and define concepts related to the intervenients (actors, actions and behavioural policies) [JBK+04] or task-oriented concepts [Woo04];
- Shared knowledge - Knowledge is shared to allow robots and its modules to communicate, using an ontology-based representation-neutral language [CT+04], allowing them to perform the require actions or to learn about their own techniques [IBGM08];
- Structural, functional and behavioural knowledge - Ontologies to describe structural, functional or/and behavioural features of robots [SM05, DDLF+11];
- Hierarchical knowledge - Characterization of domains and sub-domains of robotics [HB06].

In this Dissertation's evaluation, projects that are either unspecific about their ontological commitments, defines too many domain concepts to be useful for this research's proposes, or target a different application area were excluded, such as:

- A Robot Ontology for Urban Search and Rescue [SM05] - In a effort to improve the development speed of robotic tools for urban search and rescue responders, the National

Institute of Standards and Technology (NIST) developed an ontology that aims at populate a neural knowledge representation by capturing information about robots and their capabilities to assist in the development and testing of effective technologies for sensing, mobility, navigation, planning, integration and operator interaction within the search and rescue domain. In the time of writing this document, the referred ontology presented a specific perspective in the search and rescue domain, and thus is not profitable to this Dissertation;

- RobotEarth [M. 11] and KnowRob [TB09] - This project aims at representing a world wide database repository for robots to share information about their actions, abstracting from their hardware specifications. To do so, RobotEarth stores semantic information encoded in OWL using typed links and URIs based on the linked data principles. The created platform enables robots to store and share information, do the required offload computation (through Knowledge Processing for Autonomous Personal Robots (KnowRob), which uses OWL for task modeling and a prolog interpreter to reach symbolic goals) and collaborate with another robots. The conjunction of the KnowRob ontology [Prob, Proa] and the RobotEarth [Proc] presents a limited perspective for an Generic/Core ontology, reducing the possible spectrum of instantiated ontologies (domain/application ontologies).

- Ontology-Based Multi-Layered Robot Knowledge Framework (OMRKF) [SLH+07] and the Multi-layered Context Ontology Framework (MLCOF) [BASRH13] - Its main purpose is to help robots in object identification tasks, and thus includes 6 knowledge layers (KLayers): image, 1D Geometry, 2D Geometry, 3D Geometry, object and space. Each KLayer includes a meta-ontology, an ontology and an ontology of instances, and its composed of concepts, relations, functions of relations, hierarchies of concepts, relations of hierarchy and axioms. MLCOF composes a perspective too abstract to be considered in this Masters Dissertation. OMRKF is an extension of MLCOF, organized in a knowledge structure composed by 4 levels (perception, model, context and activity), each one of them organized in three layers (high, intermediate and low levels). This ontology aims at the description of robots, but presents an unspecific definition about their ontological commitments;

- OpenRobots Common Sense Ontology (ORO) [TB09] (complemented with KnowRob [TB09]) - Largely based on Open Source Cyc technology (OpenCyc) generic ontology [Len95], shares most of its concepts with the KnowRob ontology to provide an ontology-based framework for knowledge representation for cognitive robotic applications, focusing on the interaction with robots through object recognition, natural language interpretation, cooperation, task planning and replanning, and therefore is out of the range of the analysis.

In the next Subsections, some of the prominent domain ontologies for robotics and automation are explored, in order to comprehend and summarize the inherent domain knowledge and to enrich the ontological expertise required to develop a descriptive ontology, from which a prescriptive application ontology will be instantiated, conducive to model the coil winding machine.

4.2.1 *IEEE Standard Ontologies for Robotics and Automation*

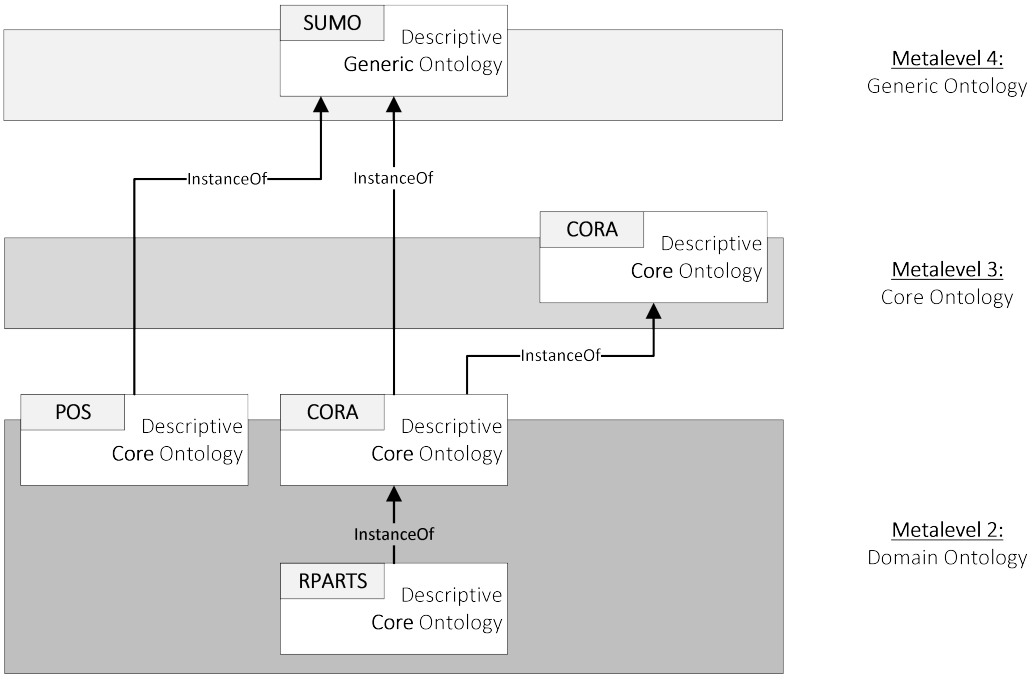


Figure 37.: IEEE hierarchical structure of ontologies.

The origins of the IEEE Standard Ontologies for robotics and automation lie in the requirement of having a common understanding in this community and facilitating more efficient integration/transfer of information. The development of this standard was documented in a series of articles [SPM+12, PCRF+13a, FCG+15, CFP+13], and the standard itself was branched between a conjunction of five ontologies. Following a top-down approach in the standard's structure, illustrated in Figure 37, the IEEE is composed by a generic ontology explored in the last section, SUMO, a core ontology CORAX, and three domain ontologies: POS, CORA and Robot Parts Ontology (RPARTS).

CORAX ONTOLOGY

As referred in [FCG+15], at the time of creation of this ontology, SUMO does not cover for every possible aspect of reality, only a perspective of it. Although, the knowledge missing in

the generic ontology could not be *added* in a domain ontology for robotics and automation due to the violation of the abstraction level that this process requires. Therefore, a core ontology was developed to interconnect the generic ontology to the domain ontology, CORAX. This ontology approaches the specification of some conceptualizations from SUMO, *e.g.* design, physical environment, interaction and artificial systems, resulting in the concepts and relations presented in Figure 38. A full explanation of this ontology can be found in the IEEE standard [fRG15].

One of the possible hypotheses to the problem specified in 3.2 is to develop a side-by-side upper ontology that has the objective of complementing the SeML's upper ontology, concerning its semantic connection to the intelligent motion control system's ontology. This approach was structured based on the CORAX development idea. CORAX tries to resolve some shortcomings and insufficiencies on SUMO that emerges when trying to semantically connect it to a robotics and automation ontology.

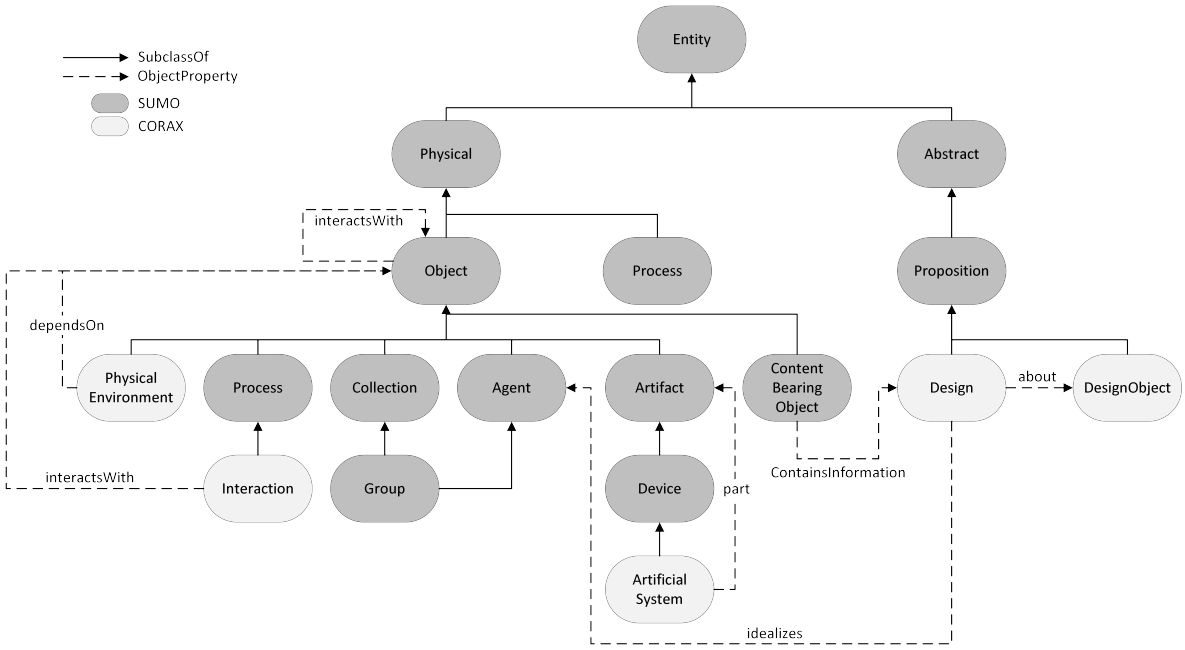


Figure 38.: The primary semantic relations between CORAX and SUMO.

CORA ontology

Despite the name, during this analysis CORA is classified as a domain ontology due to the fact that does not express the required knowledge to be considered a core ontology (and thus uses CORAX to complement the generic ontology, SUMO). This ontology provides the concepts and axioms that appear across the different applications of a domain, including concepts as robot, robotic system, robot part, etc., and aims at providing consistency among different sub-domains of knowledge. As referred by Edson Prestes in [PCRF+13b], in order

to maintain the required consistency, future ontologies about industrial and service robotics to be included in the standard shall commit to the already existing concepts/definitions in CORA. Otherwise, inconsistencies and wrong inferences might appear. This is a common problem in the ontology world, and is based on it that the effort required by the process of **semantic refactor** between a descriptive and a prescriptive ontology (illustrated in Figure 36) presents a huge difficulty and relevancy in this Masters Dissertation.

Figure 39 illustrates the main concepts and relations in CORA. This ontology unfolds around the term robot [PCRF+13a], considering that a robot:

- Is an **agent** (SUMO - performs tasks by acting on the environment or themselves) and a **device** (SUMO - an artefact with the purpose of serving as an instrument in a subclass of process);
- Has other devices as parts (further explained in the RPARTS ontology), that represent suitable mechanical and electrical parts [PCRF+13a];
- Interacts with the surrounding world through an interface;
- Can be classified accordingly with Autonomy Levels for Unmanned Systems (ALFUS)'s classification [HMA03];
- Can form social groups, *e.g.* a robotic football team;
- Together with other devices can form a robotic system, and thus *equip* a robotic environment.

ALFUS [HMA03] defines autonomy as generally **dependent on the degree of human intervention and context**, being the latter characterized by the type of global objective and environment. Based on this, the Ontologies for Robotics and Automation Working Group (ORAWG) classifies a robot through its *modus operandi* as a fully autonomous robot, a semi autonomous robot, a teleoperated robot, a remote controlled robot or an automated robot.

RPARTS ontology

This ontology is another domain ontology, not completely independent from the CORA ontology [PCRF+13b]. RPARTS ontology specifies the notions related to a specific robot part, defining it as a *role* played by a device while it is connected to a robot. Therefore, RPARTS provides only an aggregation of specific types of roles that specialize the general role of robot part, defining the requirements that have to be reached for a device representing the role of each robot part [fRG15]:

- Robot sensing part - A measuring device that is connected to a robot;
- Robot actuating part - A device that enables the robot to move and act in the surrounding environment;

- Robot communicating part - Any device used to communicate between two robots or between robots and humans;
- Robot processing part - A device used to process information.

POS ONTOLOGY

In one of the first approaches to the development of CORA, some sub-domains were identified as future work [PCRF+13a], *e.g.* tasks, positioning and physical structure. Nevertheless, this ontology was later extended [FCG+15] by the POS ontology, which aims at capturing the main concepts, relations and axioms underlying the notions of position, orientation and pose. In this ontology, two types of positional information is explored, **quantitative** (position represented by a given point in a given coordinate system) or **qualitative** (position represented as a region defined as a function of a reference object). By considering that a **position** can be attributed to an object, this ontology assumes that there is a **measure** that relates a given robot to a position measurement, either a position point (for quantitative position) or a position region (for qualitative position). Similarly to the conceptualization of position, this ontology provides the conceptualization of **orientation**, which alongside with the position constitutes a **pose**. Accordingly to [FCG+15]:

> "The pose of an object is the description of any position and orientation simultaneously applied to the same object. Often, a pose is defined with a position and an orientation referenced to different coordinate systems/reference objects. In addition, since objects can have many different positions and orientation, they can also have many different poses."

Figure 39 summarizes the integration of the CORAX, CORA and POS ontology with SUMO, illustrating the main concepts and relations.

The present ontology, that since 2015 constitutes a standard ontology for robotics and automation, had a significant relevance in this Dissertation's approach for the development of a descriptive domain ontology. Firstly, this standard procedures conducive to connect a domain to a generic ontology, provided a guidance to establish an assortment of research possibilities to the development of the prescriptive domain ontology, as explored in Section 3.2. Secondly, ALFUS [HMA03] classification system, elucidated how to correctly classify a robot through its autonomy level, and so, enabled the creation of a classification of intelligent motion control systems through its intelligence, allowing a correct distinction between each application developed within the IMCSP's project at Jilin University. Although constituting a conjunction of different abstraction level ontologies (generic, core and domain), specific conceptualizations are not complete, *e.g.* there is no a robot part to provide power to the robot, even being this possible by the semantic-connection to the generic on-

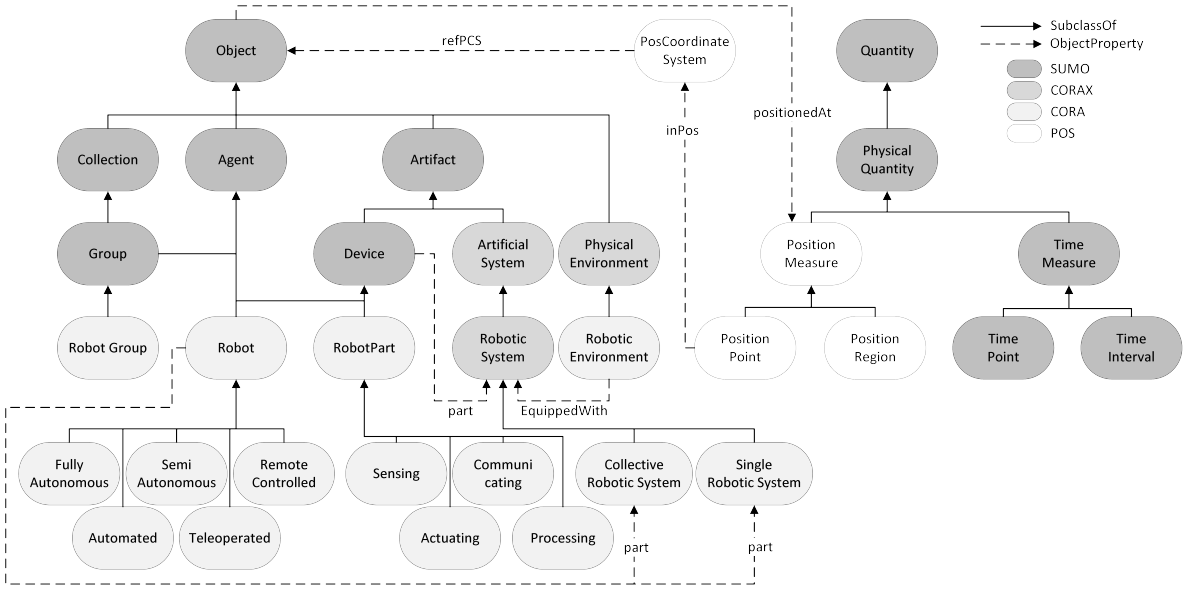


Figure 39.: Core concepts in CORAX, CORA, POS ontologies and its semantic connection to SUMO (adapted from [fRG15] and [PCRF+13a]).

tology. Therefore, the descriptive domain ontology here developed, should try to provide the knowledge to compensate the insufficiencies and weaknesses of this standard.

4.2.2 *OCOA*

OCOA [CG02] is an open software robot architecture based on behaviours and architectures, comprising four types of objects that manage and share information with each other on a distributed peer-to-peer basis, which enables it to describe control architectures for a very specific component model.

Disagreeing from Shuo and Xinjun [YMY+17], Casas and García act in accordance with a robotic software architecture's classification divided in three categories:

- **Hierarchical** [ALM88] - This architecture aims at providing an abstract perspective by restricting low-level horizontal communications (and so has reduced *flexibility*), making it difficult to adapt to modern robots (due to the iterative and reflexive characteristics presented in modern robots control processes);
- **Deliberative** [Bro86] - Opposed to the aforementioned category, this architecture comprises several behaviours (also referred as modules) that run concurrently through communication and through the environment, and thus complicating its high-level representation;
- **Hybrid** [SA01] - As implied by the name, this architecture combines the hierarchical and the deliberative architectures, resulting in a complex architectural process.

Based on the referred disadvantages of the current robotic software architectures, OCOA project presents a new architecture following the **component-based architecture** software development paradigm and using ontologies as a knowledge base, aiming at increasing the reusability and compatibility of the architecture's components, and enabling the ability to perform structured and complex coordinated reactive/deliberative behaviours.

The OCOA is composed by three types of components, illustrated in Figure 40.

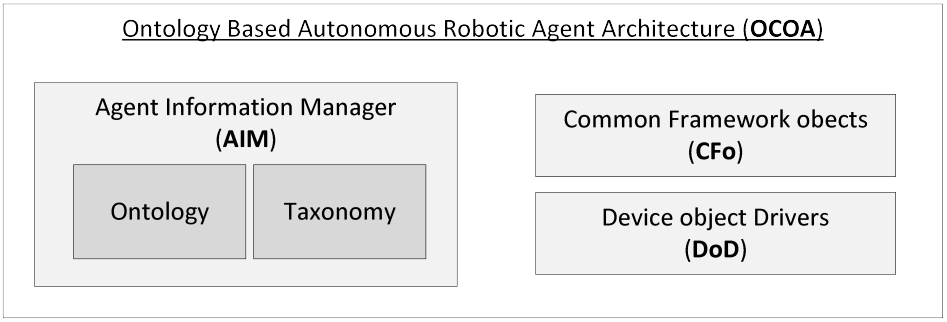


Figure 40.: OCOA's project architecture.

The main component, and also core of the OCOA architecture, is the Agent Information Manager (AIM). Through a notification system that interacts with an internal ontology and taxonomy, this component manages the agent capabilities, allowing the architecture knowledge base (composed by the interaction between the ontology and the taxonomy) to be built dynamically. Common Framework object (CFo) provides services for high level logical processing and for the infrastructure that shares raw and ontological architectural information. This component is extended by the Device object Driver (DoD) component, in order to interact indirectly (through Device Input Output Driver (DIOD)) to physical devices, and thus contains device and platform dependent code. The architecture of the OCOA is presented in Figure 41, encompassing the interaction between the different aforementioned components to create a basic architecture. Since each component have implicit its own goals and tasks, the creation of a coordinated process that identifies the architecture's common or final goal/task is enabled by OCOA, through its expression at an ontological level (as preconditions, postconditions, execution deadline and execution priority), and later by its registration in the AIM.

The OCOA's project, although aiming at a similar core objective (develop a domain ontology to describe robotic architectures), diverges from the approach of this Masters Dissertation in some key developments. The first one is the fact that, this Masters Dissertation agrees with Shuo and Xinjun [YMY+17] when considering that a a robotic architecture can be classified as reactive control, deliberative control, hybrid control or behaviour-based control, instead of the classification referred before. The second disagreement resides in the ontology developed. Firstly, the OCOA domain ontology presents a high abstraction level for a domain/core ontology, concurrently with a very light-weight theoretical expressiveness,

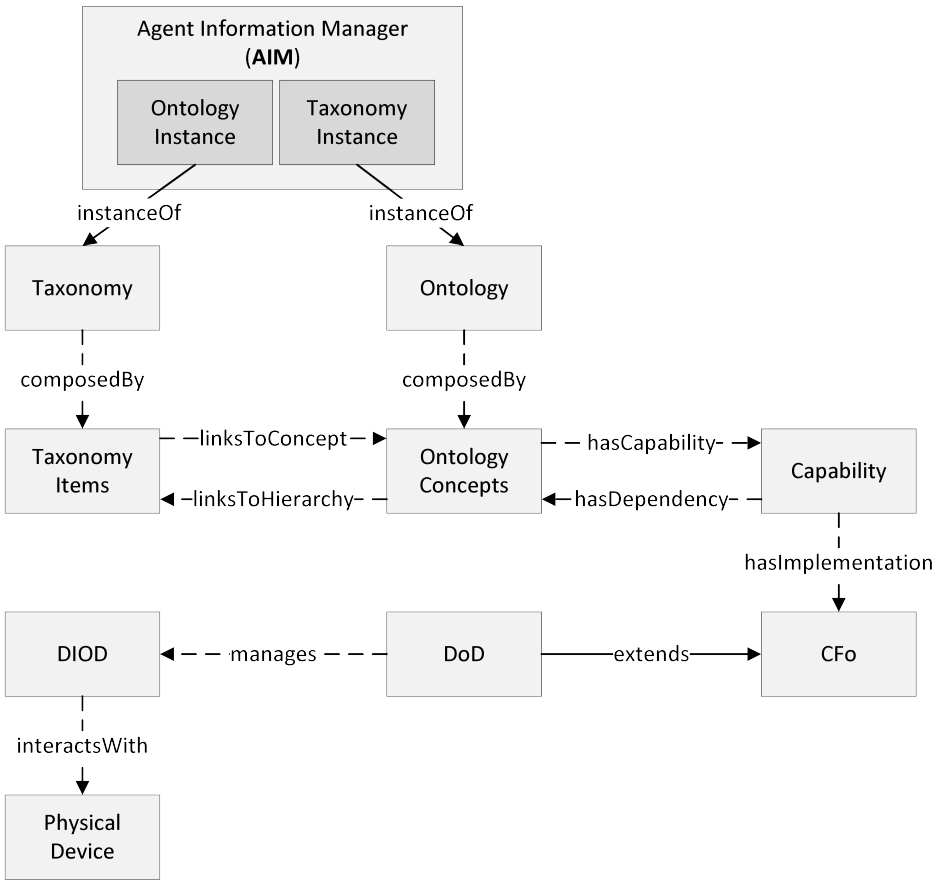


Figure 41.: OCOA's architectural ontology and its components modular description (adapted from [CG02]).

even for a prescriptive ontology, not including the required semantics to be considered a knowledge based for robotic architectures. Although the divergences, this project's foundational background includes the component-base architecture software paradigm, emphasizing the relevance of the composite design pattern for software/hardware systems, already advocated by some of the analysed ontologies in this Masters Dissertation.

4.2.3 *RoSta*

RoSta is a coordination action funded under the European Union's Sixth Framework Programme (FP6) from 2007 to 2009, aiming at becoming the main international agreement for robots standards and reference architectures in service robotics [RoS]. In this context, they have defined three objectives [045]: The creation of an action plan for the definition of a standard, the creation of an open-source driven community, and the creation of conditions to an agreement through this standard in the international community. In order to proac-

tively accomplish these goals, this team established and concluded some activities in the domain of mobile manipulation and service robots [HD09]:

- Creation of a glossary/**ontology** - Advocating that ontologies are a means to discuss and formally describe advanced robotics systems in terms of their requirements, functional components, performance, architectures, models and methods, the first goal of this project is to develop a domain ontology to provide a comprehensive and widely accepted glossary's agreement.

- Specification of **reference architecture** principles - Due to increasing number of developed *home-grown* or *ad-hoc* architectures, which leads to a divergence rather than to a convergence of robot's technology (as referred in the last section) and a set of counterproductive processes for scientific progress, the second goal is to develop reference architecture principles (rather than a specific reference architecture to cover all the domain systems, which in turn is impossible due to the flexibility demanded on the systems' architecture accordingly with its final application). These principles aim at promoting efficient engineering and reuse of components for technical platforms, enabling the development of specific architectures.

- Specification of a **middleware** - Due to the complexity of robotic systems, which in turn are composed by heterogeneous hardware components with a broad range of communication requirements with diverse real-time requirements and distributed processing power over a network of embedded computers, the third goal is the integration and communication of the various systems' components through a suitable middleware.

- *Benchmarking* - Since measuring performance of robotic systems and subsystems represents an important process to facilitate the communication between research and industry, to document research progress and to trigger research towards specific functionality performance, the last goal is to develop and use specific benchmarks on different systems' levels - from evaluating performance of robotics' functions via behaviours to assessing full systems in their environments.

Figure 42 illustrates an overview of the glossary used to structure the RoSta ontology, which in turn is fully explained in the set of *deliverables* issued by RoSta's development team [045].

Arguing, in the deliverable 2.1, that the current state of the art for the robotic systems' reference architectures appears to be quite disappointing (due to the little reuse and application of the developed reference architectures), in the deliverable 2.3 they reinforce the idea that "a common reference architecture is a mission impossible", and therefore present a set of technical and non-technical principles to consider when developing a reference

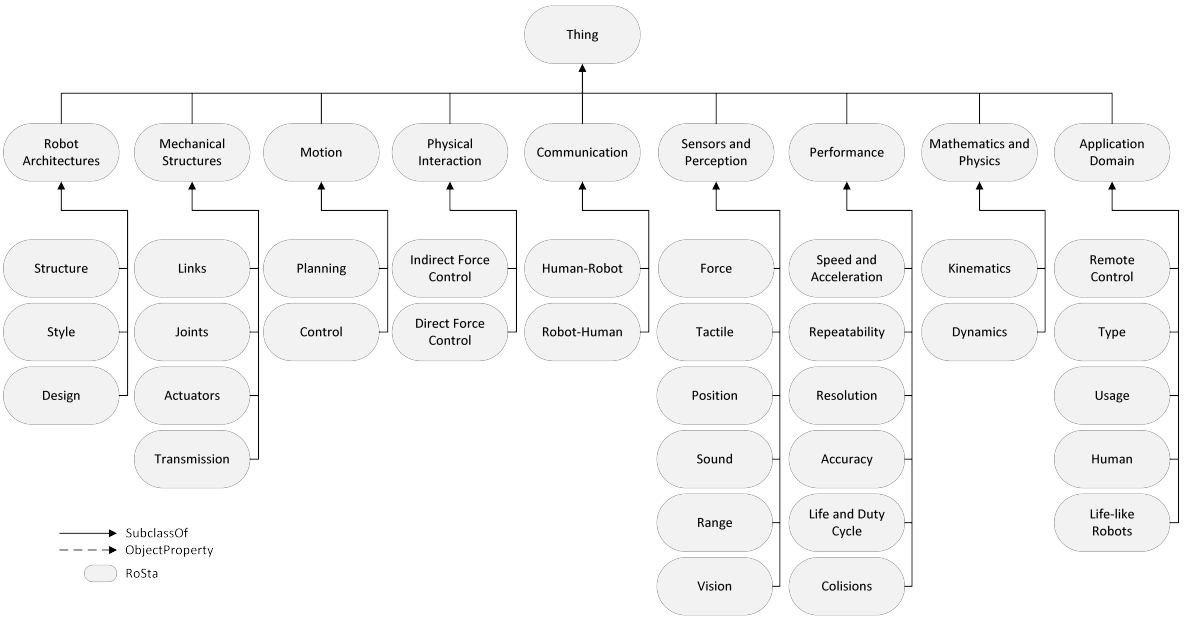


Figure 42.: RoSta's glossary structure (adapted from [RoS]).

architecture. Although relevant to the RoSta project, the non-technical principles are not extremely relevant to this Dissertation's procedures, and thus this research focus on the technical considerations:

1. Communication and coordination - A suitable coordination and separation of purposeful tasks, specially on communication between the middleware and the architecture side, allowing system's dynamic configuration, and thus enabling a robot to be configured with real-time requirements.

2. Architecture and platform independent computations - RoSta research proposes components on three different levels: Formal algorithms, portable source code and binary components. By doing so, and following model driven engineering, they aim at making use of high-levels models to generate the source code, abstracting from hardware or platform issues;

3. Knowledge structures for flexible interoperability and learning - Grant integration of artificial intelligence, cognition and semantic web, allowing the migration of code/algorithms for robots acting in the so called intelligent spaces;

4. *Compositionality* and reuse of components - Being aware of the trade-off of what goes into models/implementations and what goes into the engineering tools, RoSta advocates that *compositionality* should be achieved, by putting together the list of key-concepts (Open Systems, interoperability, loose coupling, separation of concerns,

declarative descriptions, modularity and compositionality) and architecture mechanisms [RoS09] that enable components' and systems' reuse.

The analysis of the RoSta's ontology does not enlighten this research's perspective due to the lack of information available about it. Although, the second part of the RoSta deliverables [045] provided us the specifications on reference architectures' development principles, which contains key-information in order to correctly develop the model of the coil winding machine (developed in the Jilin University), one of this Masters Dissertation's objectives (see Section 3.3).

4.2.4 *PROTEUS*

PROTEUS is a project that aims at creating a platform to foster exchanges, co-operations and interactions among the members of the French Robotic Community, enhancing its members' capability to share their skills, know-hows and research findings, and thus facilitating the transfer between industry and academic worlds [Pro12]. Conducive with their project core ideology, PROTEUS's development team presented a set of scientific and technical goals, based on the creation of an innovative technological infrastructure - the identification of problems (as specifications or simulations) and the respective architectures and/or algorithms that should be developed as solutions, the comparison and publication of these solutions in PROTEUS's database, and lastly the validation on real-world systems. The result of referred ideology was the development of individual shared products that constitute a large platform:

- Ontologies - Domain ontologies to cover the domains of aerial, ground, marine/submarine and humanoid robotics;
- DSML - Through the integration of the ontology, the existing DSML is capable of describing problems for specific parts of the robotic domain;
- Reference robotic environments - Scenarios described using the resulting DSML;
- Tools - The tools that constitute the platform and allow its consistent workflow;
- Tests - Develop validation tests to challenge and verify the reliability of the created products.

In the analysis of PROTEUS's project, the focus will be the developed ontologies and its respective integration with the DSML, due to the fact that this process follows the same principles followed in this Masters Dissertation. The ontologies developed in PROTEUS should reflect the knowledge (trough concepts, relations and axioms) that allow the DSML to model the following domains:

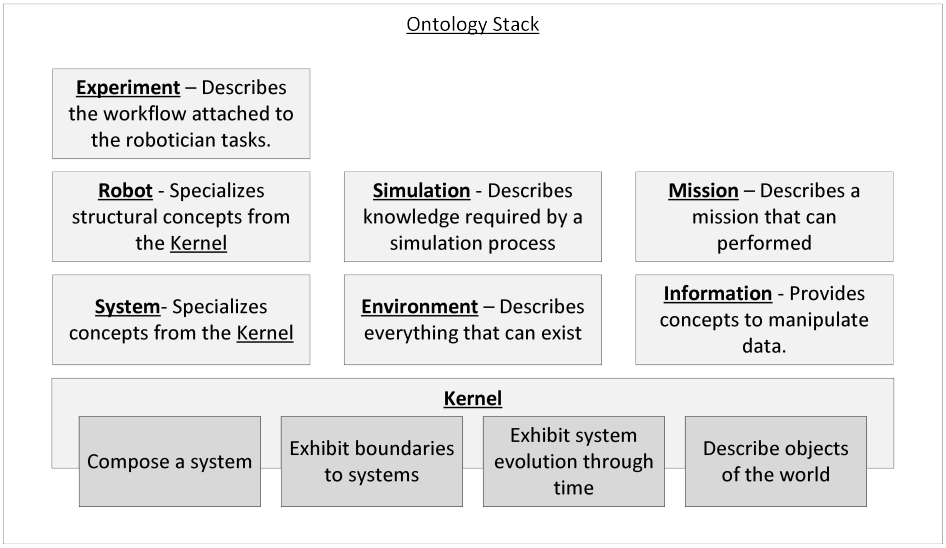


Figure 43.: Summary of PROTEUS's ontologies stack.

- Robotic architectures - Specify robot's architecture and the components that form the architecture (sensors, actuators, planners);
- Robotic interfaces - Create communication mechanisms between the components, allowing them to interact in a meaningful way to achieve the desired goal;
- Robotic implementations - Represent system's and component's behaviours.

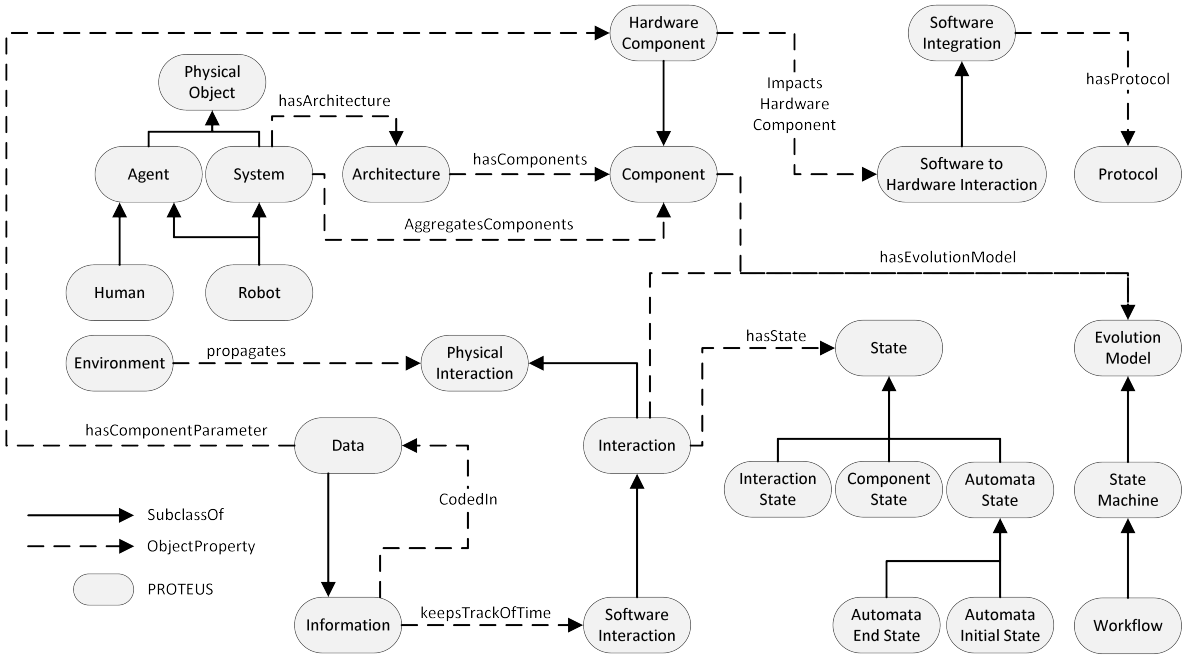


Figure 44.: Overview of PROTEUS's ontology main concepts.

To this end, an ontology stack was created, illustrated in Figure 43, including both kernel-related knowledge to describe dynamic systems moving in a specific environment, and the several modules devoted to specific aspects of the PROTEUS's platform - information, environment, mission, robot and simulation. Despite being presented as a stack, this individual modules are linked in order to provide the required domain knowledge to the resulting DSML. The diverse domain ontologies developed in this project are fully explored in [Faro09], and in the Figure 44 the main concepts of the PROTEUS ontology are illustrated. From the relations, this research easily infers that PROTEUS advocates that a system has an architecture which is composed of components, which can be software or hardware components. Similarly to the reviewed ontology, PROTEUS follows a component-based architecture, where hardware and software components represent hardware and software parts of the robotic systems, respectively. With the knowledge provided by the ontologies, the DSML offers specific notations and abstractions within the domain that increase programmers' productivity, allowing the programmer to quickly and precisely implement novel software solutions to the emerging problems. Figure 45 illustrates the transfer of knowledge between PROTEUS's ontologies and the resulting DSML, enabling the programmer to create a robotic architecture based on the composite design pattern, where for each hardware component a software component is associated. By doing so, the architect creates a relation of component-behaviour that allows the robotic architecture to express both the static and dynamic aspects of systems.

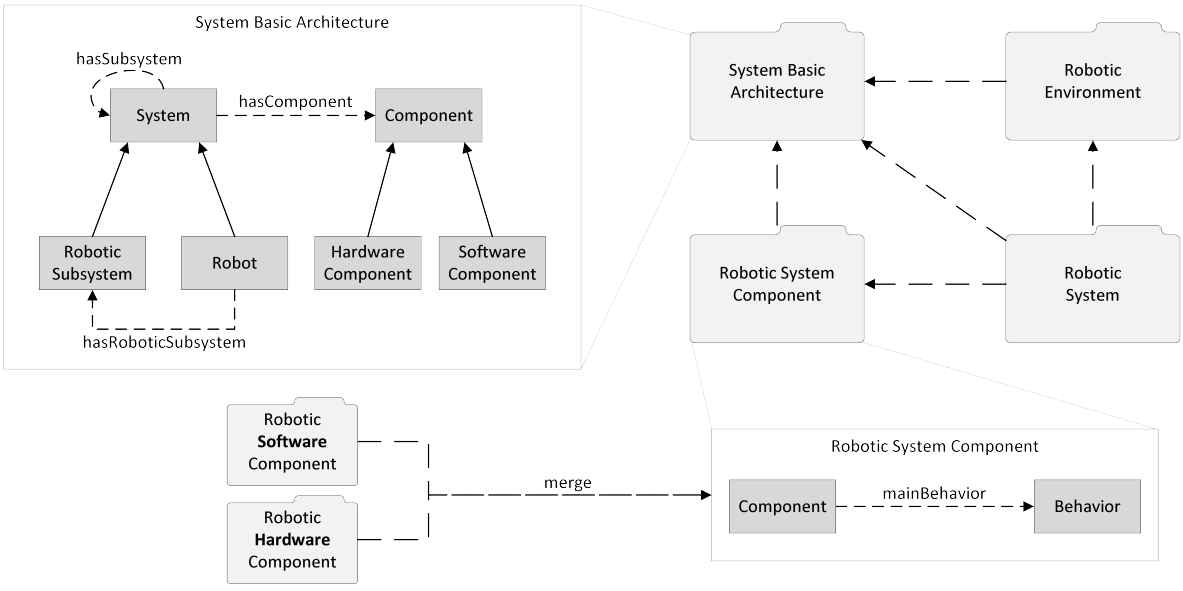


Figure 45.: PROTEUS's DSML architecture.

The approach followed by this project is the same advocated in this Masters Dissertation. Although, since this project aims at providing a platform to foster exchanges, cooperations and interactions among the members of a small community, the core objective

is not the same. This aimed agreement inside a small community is inherent in the PROTEUS ontologies' development procedures - instead of connecting the domain ontology to a well-structured and thoroughly agreed generic ontology, they developed a light-weight generic ontology (which is directly integrated with the core and domain ontologies), that besides providing a reduced knowledge-base about the real-world, reduces the probability of an agreement between the developers' community (that is the ontologies' core objective). Contrarily, as a final goal, this Dissertation intends to provide a DSML (resulting from a shape-shifting process of the SeML with the developed ontologies) that can be used to model intelligent motion control systems, and thus, it aims at an agreement by providing a perspective of the world (through ontologies) based on the research of the ontologies and standards already implemented. Nevertheless, the analysis of this project, specially the composition of its ontologies, thoroughly elucidated us about some key-procedures in the development of the descriptive domain ontology, specially in the specific relation existent between the hardware and the software parts of a robotic system, which enabled the correct differentiation between software and hardware properties of a robotic system (which is a dependency studied during the ontologies' design, explored in the next chapter).

4.2.5 *Semantic Sensor Network*

This subsection researched and reviewed some prominent domain ontologies in the robotics and automation domain, exploring architectures, behaviours and environments of robotic systems. Nevertheless, following the composite design pattern, a robotic system is composed of another systems and subsystems, which in turn forces the analysis of the knowledge inherent to each individual system that possibly constitutes a robotic system. Therefore, in this subsection the SSN ontology is explored, in order to provide the required abstraction to create a knowledge base and an agreement not only for a sensor subsystem but for every subsystem that possibly composes a robot. The SSN ontology [CBB+12], developed by the W3C Semantic Sensor Network Incubator Group (SSN-XG), aims at describing sensors and observations in terms of capabilities, measurement processes, observations, and deployments, conducive to standardize the SSN ontology, to allow its use in a **linked sensor data** context. It also enables an agreement in the Internet of Things (IoT) and in the Internet of Services (IoS) domains, while fostering the adoption of the SSN ontology in the Open Geospatial Consortium (OGC) community. This ontology was built around a core Ontology Design Pattern (ODP) to describe relations between sensors, stimulus, and observations, the Stimulus-Sensor-Observation (SSO) pattern [JC10], and therefore presents four perspectives - **sensor** (what senses, how it senses and what is sensed), **observation** (observed data and related metadata), **system** (systems of sensors and deployments) and **feature and property** (what senses a particular property or what observations have been

made about a specific property). The result was a descriptive domain ontology organized, conceptually but no physically, into ten modules, encompassing 41 concepts and 39 object properties, directly inheriting from 11 Dolce UltraLite (DUL)'s concepts and 14 DUL's object properties. Figure 46 illustrates the ten modules and the relation between them.

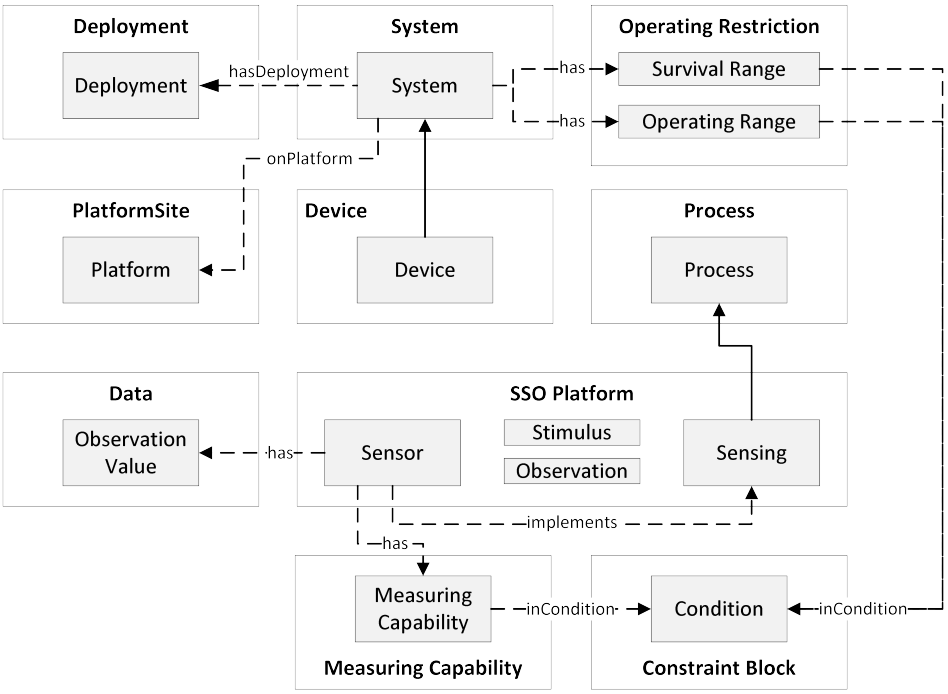


Figure 46.: The SSN's ontology, key concepts and relations, split into conceptual modules (adapted from [CBB+12]).

The SSN ontology advocates that *stimulis* are changes or states (DUL:Event) in an environment that a sensor senses through a sensing process. In order to capture the knowledge involved in a sensing process, they created the concept **observation**, which links the **act of sensing** and the event that represents the **stimulus** (DUL: includes events), the sensor (ssn: observed by), the **method** (SSN: sensing method used), the **result** (SSN: observation result), the observed **feature** (SSN: feature of interest) and the observed **property** (SSN: observed property). This concept is complemented with a sensor's perspective, referring that, for any property observed by a sensor, the sensor's performance might be affected by environmental conditions, that can be related (or not) to the property observed, being this performance modelled by the measurement capabilities of a sensor, as illustrated in Figure 47. To complement the sensor's perspective, the SSN ontology constructed a system's perspective around the concept system, defining system as an agglomerated of components (composite design pattern), from which the devices and sensing devices (subclass of devices) have operating and survival ranges - an operating range describes the environmental characteristics in which the system is intended to operate, and the survival range represents the

environmental conditions to which a device can be exposed and continue to operate as per defined measurement capabilities.

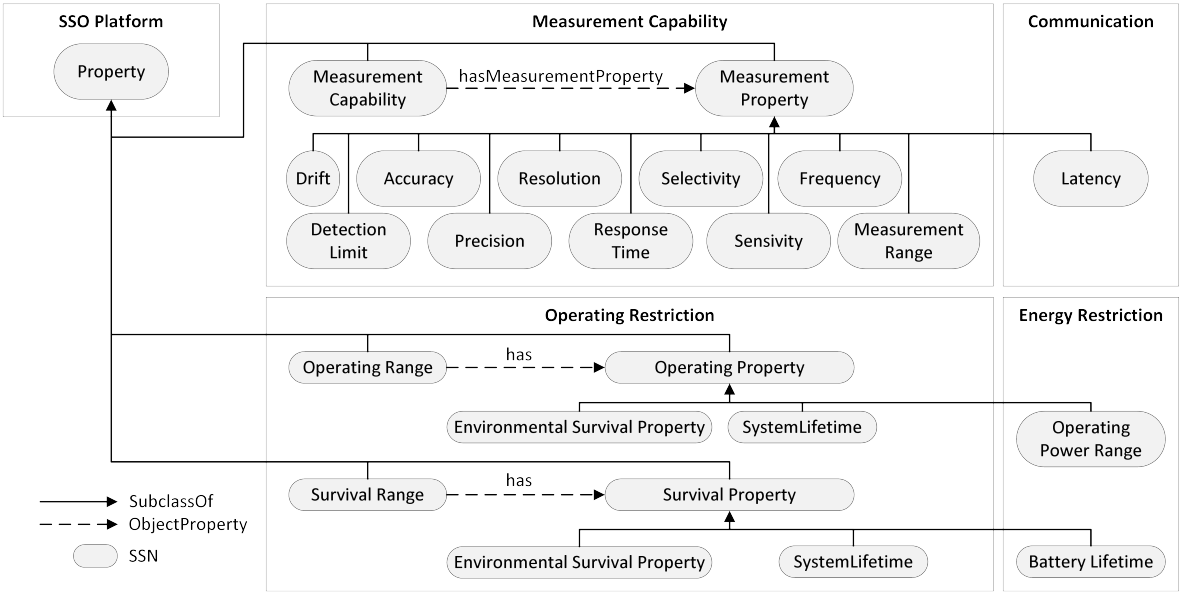


Figure 47.: Enumeration of the SSN's ontology measurement, environmental and survival properties (adapted from [W3C14]).

By abstracting from the sensing domain, this research can extract from this ontology a rich knowledge network to describe any subsystem of a robotic system in terms of its hardware properties and environmental behaviour. One example of this is the Semantic Actuator Network (SAN) ontology [SAD+], an actuators ontology based on the SSN ontology. Therefore, the knowledge provided by the SSN and the SAN ontologies will be used in the development of the descriptive domain ontology.

5

RESULTS AND PRACTICAL EVALUATION

In this chapter, the theoretical insights and practical results gained from the research conducted in this Masters Dissertation are presented.

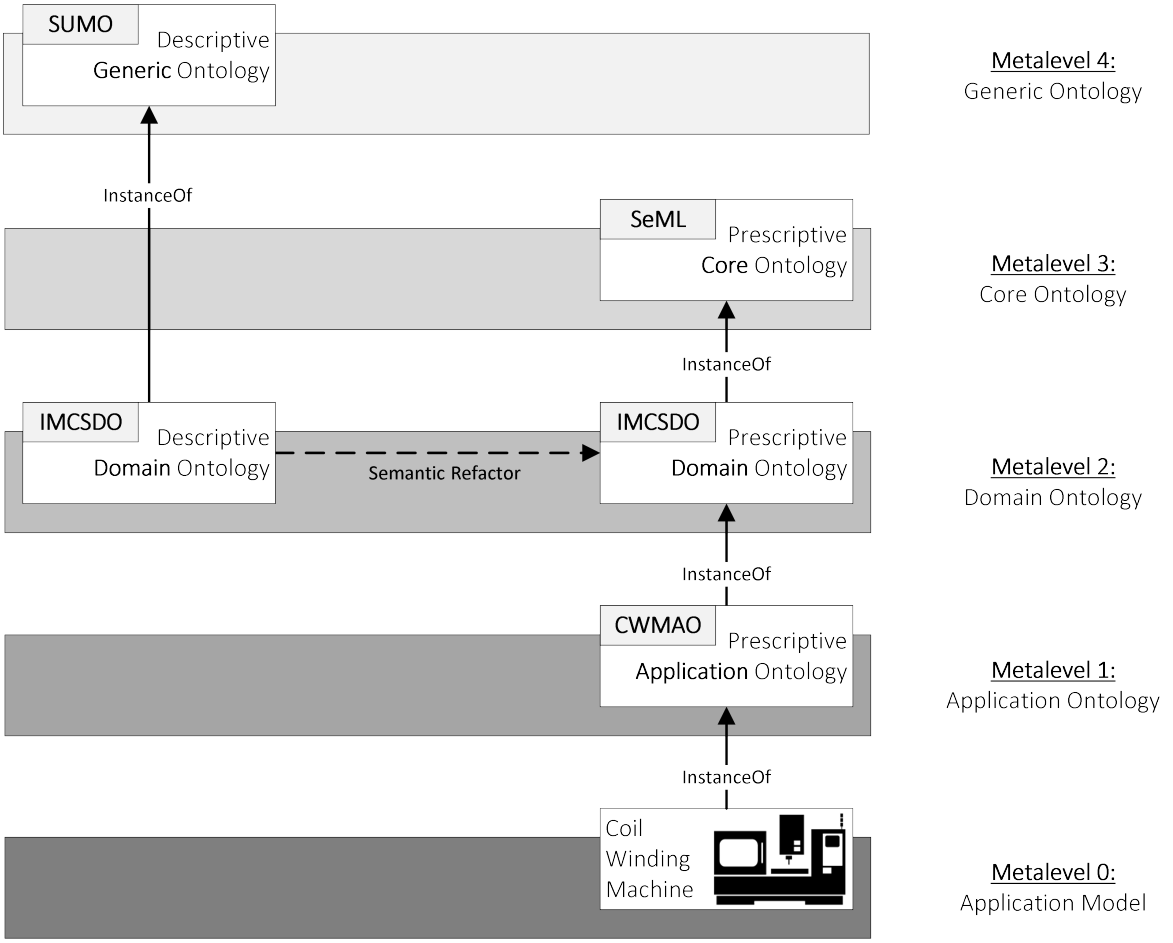


Figure 48.: Contextualization of the developed ontologies and their relation with the application's model, using the alternative perspective of the HexOntology (see Figure 10).

Therefore, and based on the findings from the last chapters, the first section discusses the design of the SeML. Besides a comprehensive description of its grammar and workflow, the

developed mechanisms embedded on this tool that allow the automation of systems' implementation and configuration, with generative code generation, are reviewed. Subsequently, the second section elaborates on the practical challenges for a complete implementation of a descriptive domain ontology for the intelligent motion control systems' domain, from which a prescriptive domain ontology was developed, as illustrated in Figure 48. Lastly, based on the mentioned ontology, a prescriptive application ontology is *built* and explored. To round off the chapter's approach, the developed ontologies are evaluated in a case study realized with a real-world system developed at Jilin University, the coil winding machine.

5.1 Design of the Semantically-enriched Modeling Language

The SeML is a DSL which domain is modeling systems. To do so, it acquires semantic knowledge from ontologies and provides an infrastructure that enables a system's designer to create a model, describing its:

- Requirements - Through the requirements, the designer indirectly describes machine's capabilities that provide the behaviour demanded for solving a problem or achieving an objective;
- Behaviours - The software implementation of each one of the software components that compose the system's model;
- Properties - Code execution properties that characterize the system's behaviours.

As a result, the SeML generates the system's code that meets the requirements described by its designer, in a process similar to the one illustrated in Figure 49.

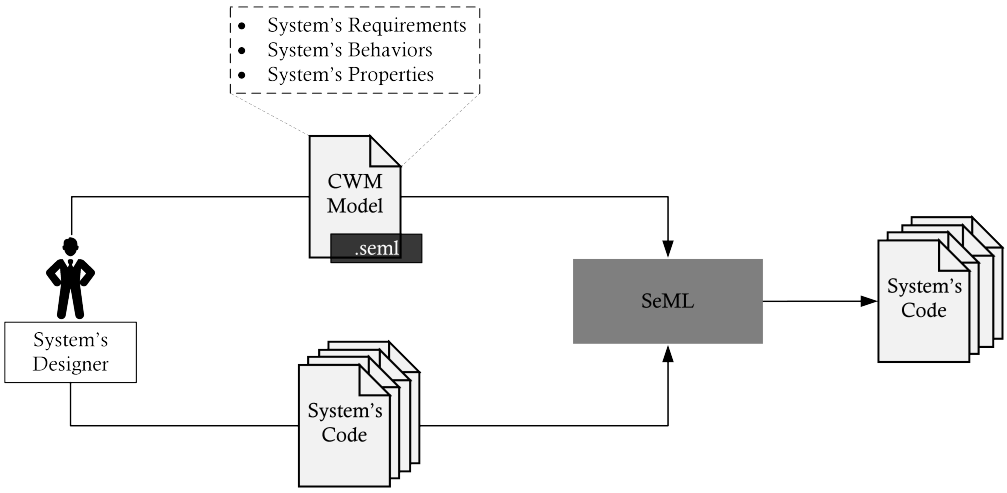


Figure 49.: The interaction between the system's designer and the SeML, conducive to the creation of the system's model.

Developing a programming language forces its designer to compose a processing system capable of read and understand the code written on it. Contrarily to the processing system of a DSL, the processing system of a GPL includes the phases illustrated in Figure 50 [ASU86]. In this complex process, the preprocessor converts the source program, usually divided into modules stored in diverse files, into source language statements. Subsequently, the compiler uses the source program to generate assembly code that is translated by an Assembler into machine code and then linked with some libraries into executable target code, by the Linker.

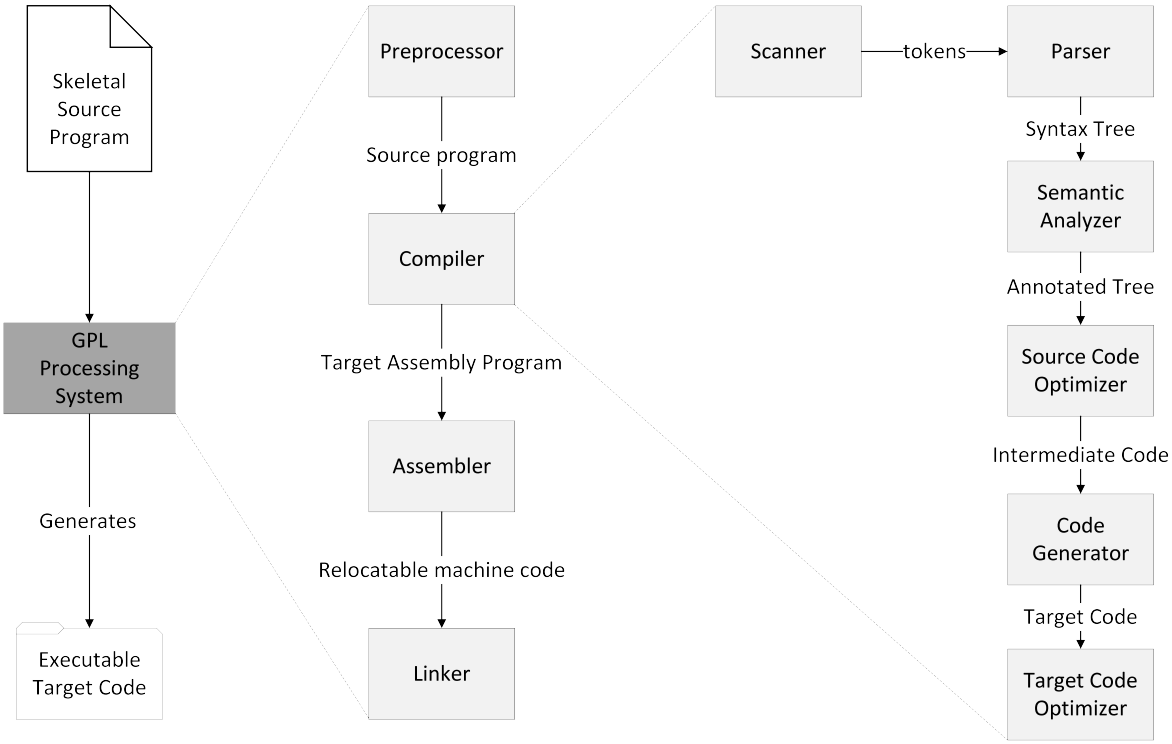


Figure 50.: The phases of a GPL processing system.

DSLs are programming languages that target a specific problem domain, and so, the opposite of GLSs that are meant to provide features to solve a diverse variety of problems, *e.g.* C, C++ and Java. A program or a specification written in a DSL can be either interpreted/-compiled into a general purpose language or represent data that will be processed by other systems. Implementing a DSL encompasses the development of a program that is able to read text written in that DSL, parse it, process it, and possibly interpret it or generate code into a target language [Bet16], as illustrated in Figure 51.

During the compilation phase, the analysis consists of four phases:

- Linear analysis - Also referred as lexical analysis or scanning (when performed by a **scanner**), is the first phase of compilation, in which a stream of characters in the

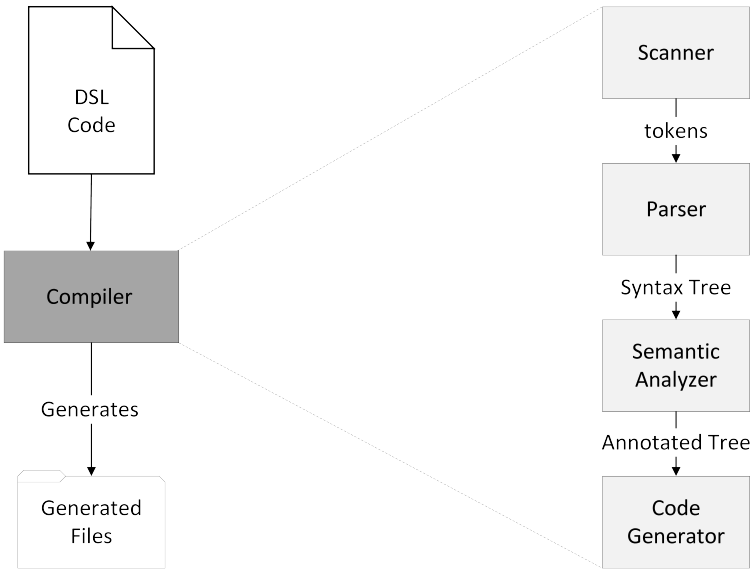


Figure 51.: The phases of a DSL processing system.

source program is read and grouped into tokens, composing sequences of characters that have a collective meaning;

- Hierarchical analysis - Also called syntax analysis, it is realized by the **parser** and hierarchically groups the tokens into grammatical phrases with collective meaning;
- Semantic analysis - Ensures that the grammatical phrases, composed during the previous phase, fit together meaningfully, gathering information for the subsequent phase;
- Code generation - Uses the Abstract Syntax Tree (AST) to generate code in the target language.

Since the implementation of a parser and respectively the scanner represents very complex and costly processes, tools were created to deal with these tasks. In particular, there are DSLs (usually referred as parser generators or compiler-compilers) that generate code for the lexer and the parser from the specification of a language's grammar, which is a set of rules that describe the form of the elements that are valid according to the syntax of a DSL [Bet16, p. 11]. To develop the SeML, the Eclipse Integrated Development Environment (IDE) was used, providing an aggregation of features that enrich the user-experience of the DSL itself, *e.g.* syntax highlighting, background parsing, error markers, content assist, *hyperlinking*, *quickfixes*, outline and automatic build. Eclipse was the IDE chosen because it integrates the **Xtext framework** [ES16] to implement programming languages and DSLs. The advantage of using this framework is that, from the specification of the grammar, it generates the lexer, the parser, the AST model, the construction of the AST to represent the parsed program, and the Eclipse Editor with all the IDE features.

As indicated by the name, SeML is more than a elementary DSL for modeling systems. This programming language is semantically-enriched, which means that, in its background, has an ontologies (a semantic technology) providing the semantic knowledge required to model a system, as illustrated in Figure 52.

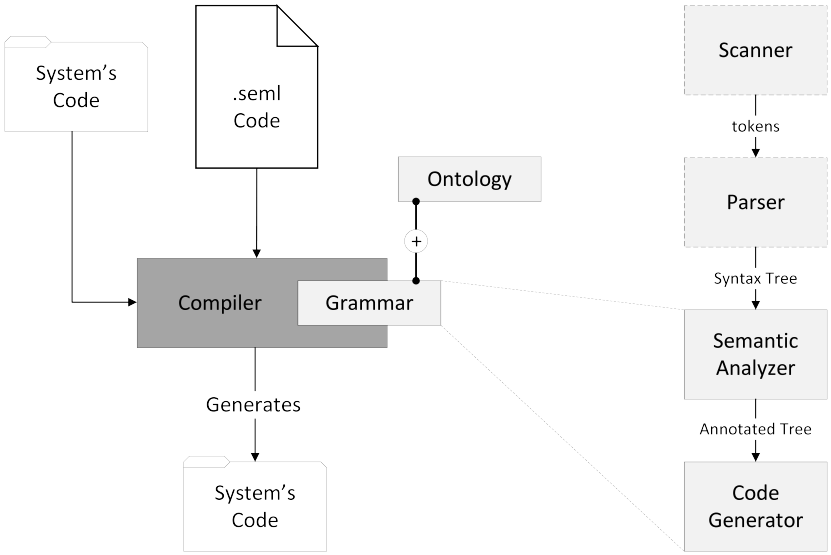


Figure 52.: The phases of the SeML processing system.

Ontologies are composed by concepts, relations and axioms that are used to enrich the keywords and the set of rules that describe the SeML's grammar, and by doing so, enable the language to **shapeshift** according to the knowledge inherent in the provided ontologies. Therefore, if the system's designer provides a domain ontology for robotics (an ontology that describe the robotics domain through concepts, relations and axioms) and, subsequently, an application ontology for a specific robotic machine (*e.g.* 6-axis robotic arm), it will be able to model that robotic machine.

In order to be able to use the semantically-enriched metamodeling environment provided by the SeML to model a specific system, the system's designer must provide the ontologies that represent the domain and application knowledges required to model the system, and these ontologies must be *instance-of* of the Core Ontology provided by the SeML (see Figure 48), which in turn provides core knowledge about hardware/software systems, illustrated in Figure 53.

The core ontology for software/hardware systems follows the composite design pattern, also used by the BWW ontology and the IEEE Standard Ontologies for Robotics and Automation analysed in Chapter 4. By doing so, this ontology branches owl:*Thing* into (SeML:) problem, (SeML:) characteristic and (SeML:) component, advocating that a (SeML:) component **is composed of** other (SeML:) components. A (SeML:) component can have a (SeML:) problem and can be demanded by a (SeML:) characteristic, which in turn solves a (SeML:)

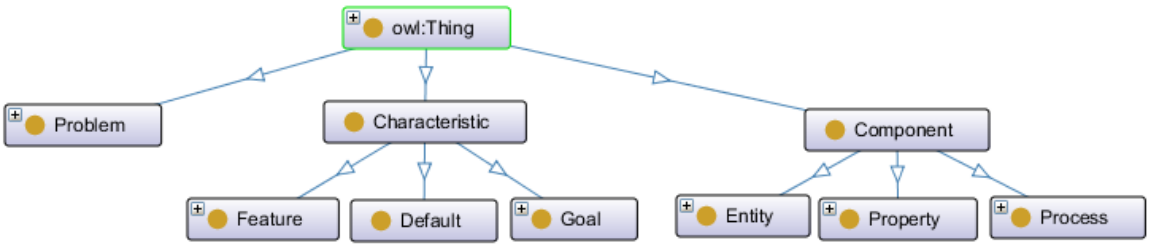


Figure 53.: The SeML's Core Ontology.

problem. By doing this, the SeML core ontology creates a group of object properties that focus on describing system's variability.

A (SeML:) component is either a (SeML:) property (describes every kind of property related with code execution, and so, it can be constants or variables), an (SeML:) entity (A container for every type of components) or a (SeML:) process (a procedure that can have inputs and outputs, and so, it describe continuous behaviour, *e.g.* function, method or sequential instruction).

A (SeML:) characteristic is intended to model a characteristic of a system through the conceptualization of features (system's attributes), defaults and goals (system's objectives), and thus might be used to:

- Require/reject the instantiation of other (SeML:) characteristics;
- Impose existential restrictions on (SeML:) component individuals;
- Influence solution proposals;
- Trigger SWRL rules.

Besides enabling the description of systems' variability, the SeML has a mechanism to related each part of the ontology to external tools that can have different roles in the code generation, being this accomplished through the annotations created in the application ontology.

The main focus of this Masters Dissertation is to metamodel an intelligent motion control system following an ontology-driven process. To do so, this Dissertation *uses* to the SeML, that provides a semantically-enriched environment to model software/hardware systems and enable the automation of its configuration and implementation by *using* the description of its variability.

After exploring the development and characteristics of the SeML, is now obvious that the decision of following the fourth hypothesis from the ones indicated in Section 3.2 is based on the insufficiencies and weaknesses of the remaining hypotheses:

1. Hypothesis 1 - The extension of the core ontology provided by the SeML would indeed enhance the knowledge within the ontology, allowing a broad description of the

intelligent motion control systems' domain, but would constitute an overhead in the global process of modeling an intelligent motion control system (since the additional knowledge provided by the extension would not be recognized by the SeML);

2. Hypothesis 2 - Develop a prescriptive domain ontology without develop a descriptive domain ontology first, deceives the perspective applied to the ontology, due to the fact that this perspective would not include the perspectives of other ontologies for the domain, compromising the agreement required;

3. Hypothesis 3 - Solves the problem of the previous hypothesis, but creates the same problem for the development of a descriptive generic ontology, since an ontology at this abstraction-level should not be developed without a huge team of experts from different and diverse knowledge areas that provide the necessary awareness and required perspectives to develop a well-structure generic ontology, *e.g.* SUMO;

4. Hypothesis 4 - By creating the descriptive domain ontology from a descriptive generic ontology, this hypothesis enables an agreement between domains and within the domain, solving the problem referred in the third hypothesis. Since the creation of the prescriptive domain ontology is the result of the *semantic refactor* of the descriptive domain ontology, this hypothesis also solves the problem mentioned in the second hypothesis. After the *semantic refactor*, the resulting ontology only contains knowledge inherited from the SeML's prescriptive core ontology (see Figure 48), and so, the problem introduced in the first hypothesis is solved too, making this hypothesis the one to be followed.

In order to follow the fourth hypothesis, and to correctly use the SeML, a **descriptive domain** ontology, a **prescriptive domain** ontology and a **prescriptive application** ontology were developed, as explored in the next Sections.

5.2 Design of the Intelligent Motion Control Systems' Domain Ontology

The main goal of this section is to define a prescriptive domain ontology that is expressive enough to represent the common semantics of the intelligent motion control systems' domain. To do so, a descriptive domain ontology will be develop, describing reality through the conceptualization of *things* and their relations. Figure 8 classifies both ontologies, according to the HexOntology (see Figure 10), and thus allows the comparison between them.

The descriptive ontology is connected to the SUMO ontology, in order to obtain a almost complete coverage of reality, characterized by a heavy-weight theoretical expressiveness, that results of the huge amount of concepts, relations and axioms used to **describe** the things in the real-world. Contrarily to the descriptive ontology, the prescriptive ontology

Table 8.: Comparison between the descriptive and prescriptive domain ontologies for the intelligent motion control systems' domain.

	Intelligent Motion Control Systems	
	Descriptive Ontology	Prescriptive Ontology
Purpose	Reference	Reference
Abstraction Level	Domain	Domain
Theoretical Expressiveness	Heavy	Light
Implementable Expressiveness	Descriptive	Prescriptive

defines reality, and therefore must only contain implementable concepts, relations and axioms. To convert a descriptive ontology into a prescriptive one, a *semantic refactor* process will review and rearrange all the non-implementable information in the descriptive ontology, as will be further explain.

In the next subsections, a detailed description of the methodologies and techniques used to develop each one of the referred ontologies is presented. Nevertheless, since the descriptive ontology contain a big aggregation of knowledge, represented as concepts, relations and axioms, a huge part of its composition is explored in the annex A.

5.2.1 *Descriptive Domain Ontology*

Following the methodologies for ontologies' development, mentioned in the Subsection 2.2.1 (*METHONTOLOGY*, the ten commandments and OntoClean), firstly it's necessary to explore the details about the needed resources and then build the desired ontology. This task will be accomplished with the creation of a Gantt diagram, presented in the annex C. Aiming at reducing the gap between the ontological art and the ontological engineering, the second step of the development of the descriptive domain ontology is equivalent to the specification phase in *METHONTOLOGY*. It consists in understand the domain and define use-case scenarios from design experts' perspective, aiming at specifying the global goals of the ontology. Table 9 presents the specification and use-case scenarios of the referred ontology.

The process of developing a descriptive domain ontology must aim at the creation of abstract and general concepts, relations and axioms, rather than the creation of a particular and independent massive domain ontology, increasing its flexibility, modularization and maintainability. Therefore, the third step is the reuse of existent ontologies (if applicable), allowing knowledge to be shared, and thus an agreement between domains. To do so, this ontology will be semantically connected to the SUMO ontology, analysed in great detail in Subsection 4.1.3.

Table 9.: Domain specification and use-case scenarios for the intelligent motion control systems' descriptive ontology.

Use-case scenarios for the Intelligent Motion Control Systems' Descriptive Ontology	
What is the domain?	• Intelligent Motion Control Systems
What can the ontology be used for?	• Understand systems' properties, features and hierarchical breakdown structure • Describe systems' working flow • Improve systems' design and development • Classify an Intelligent Motion Control System accordingly with its autonomy level
What questions should the ontology answer?	• What are the main components in a Intelligent Motion Control System? • What kinds of properties a component possesses and how they influence the overall systems' performance? • What are the dependencies between systems' processes and hardware components ?
Who will use and maintain it?	• It will be used, populated and maintained by its developer in order to be integrated with another established ontologies, aiming at full coverage of the robotics and automation domain.

A descriptive ontology describes reality but reality is not constructed from it, constituting a shared and structured knowledge foundation represented by a set of concepts, their interrelations, and axioms under the *open-world assumption*. As aforementioned, the SeML is a prescriptive ontology, an thus a descriptive ontology cannot be constructed from it due to the lack of knowledge expressed. Therefore, aiming at the creation of a descriptive domain ontology, SUMO was used as a upper and generic ontology, inheriting all the knowledge that is described on it. SUMO describes, through its designers' perspective, the real-world and its *things*, creating a consensus between diverse domains and applications (granting the sustainability, interoperability and adaptability aimed by the *10 Commandments*), and thereupon, allow the descriptive ontology for intelligent motion control systems to be connected to other domain ontologies through the core abstraction level that represents robotics and automation, *e.g.* the domains of object recognition, speech recognition, space mapping, human-robot communication etc.

In order to build the core structure of the descriptive ontology, and following the approached advocated by the ORAWG, the combination of the concepts agent (SUMO: something or someone that can act on its own and produce changes in the world) and device (SUMO: A device is an artefact whose purpose is to serve as an instrument in a specific subclass of process) was used to conceptualize robot and develop a taxonomy that allows its classification through its autonomy level, as illustrated in Figure 54:

- Automated system - An automated system is the agent of an automated process. 'Automated' simply means that actions can be carried out by something other than a cognitive agent (SUMO);
  - Automated robot - An automated robot can accomplish a goal or solve a problem to which the robot was automated to, not adapting to changes in the surrounding environment;
  - Teleoperated robot - A teleoperated robot can accomplish a goal or solve a problem with the intervention of an human operator, either by controlling the actuators using sensory feedback or by assigning new goals on a continuous basis;
  - Remote controlled robot - A remote controlled robot can accomplish a goal or solve a problem when the control of the tasks' execution belongs to the human operator on a continuous basis, resulting from direct or indirect observation of the robotic system;
  - Semi-autonomous robot - A semi-autonomous robot can accomplish a goal or solve a problem with human intervention at diverse levels, in order to induce the robot to plan and execute the purposed task, through the execution of the required processes.
- Cognitive system - An automated system is the agent of a cognitive process;
  - Fully-autonomous robot - A fully-autonomous robot can accomplish a goal or solve a problem without human intervention, while adapting to operational and environmental conditions.

Respecting the realism, *multi-perspectivalism* and *adequatism* advocated by the **Ten Commandments of Ontological Engineering**, this Dissertation's perspective encompasses robot as a composite component designed to perform purposeful actions, through the execution of processes, in order to accomplish a goal or solve a problem. To accomplish the referred actions, a robot is composed of parts that can play the role of a platform part, an interfacing part, a powering part, a processing part or a storing part:

- Interfacing part - An interfacing part allows the robot to sense, act and interact with the surrounding environment.
  - Actuating part - Allows the robot to move and act in the surrounding environment, through an actuating process;
  - Communicating part - Provides communication among robots and humans through a communication process;
  - Sensing part - Senses the surrounding environment, through a sensing process, in order to measure a physical quantity;

- Platform part - Framework of the robot that holds all the robot parts together, allowing the robot to have the required performance;
- Powering part - Provides power to the robot through a powering process, accordingly with its own powering properties;
- Processing part - Responsible for processing data and information, through a processing process.
- Storing part - Provides storing capabilities, by allowing the robot to read/write and control a storing part, through a storing process.

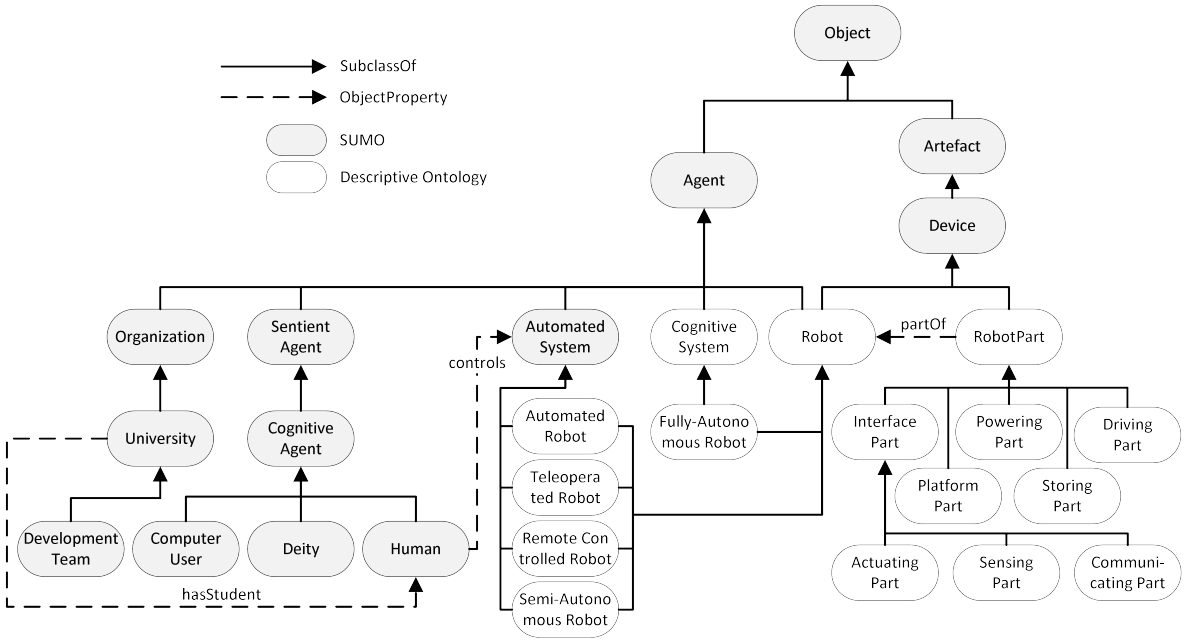


Figure 54.: Main concepts of the descriptive ontology for intelligent motion control systems.

Thus, a robot part is defined as an **atomic** or **composite** component that represents a relation between a given component and a robot, indicating that the referred component is playing the *role* of a robot part when composes the robot composite component. By doing so, in this ontology, a robot part is a piece of hardware that acts on his own when is not connected to a robot, and acts as a robot part when is integrated in the robot, performing is own role, likewise in the IEEE Standard Ontology for Robotics and Automation [fRG15] (explored in detail in the Subsection 4.2.1).

As mentioned in the previous section, the SeML follows the composite design pattern, allowing a component to be composed by other components. Correspondingly with this approach, the IEEE Standard Ontology for Robotics and Automation [fRG15] advocates that the composite design pattern is the most advantageously way to approach the structure of such complex systems. Equivalently, in this descriptive domain ontology, a robot

(SUMO:Agent + SUMO:Device) is a composite component composed of parts that represent composite or atomic components. Therefore, at a higher level (and since a robot is an agent and agents can form social groups), a robot can be a **part of** a group of robots (or other kind of complex system) that work like a team to achieve a goal. By doing so, a Robot can be part of a single robotic system and a robot group can be a part of a collective robotic system (being a robotic system an artificial system formed by robots and devices intended to support the robots in achieving their goals), both aiming at equipping a robotic environment (physical environment equipped with robotic systems). These relations are illustrated in Figure 55.

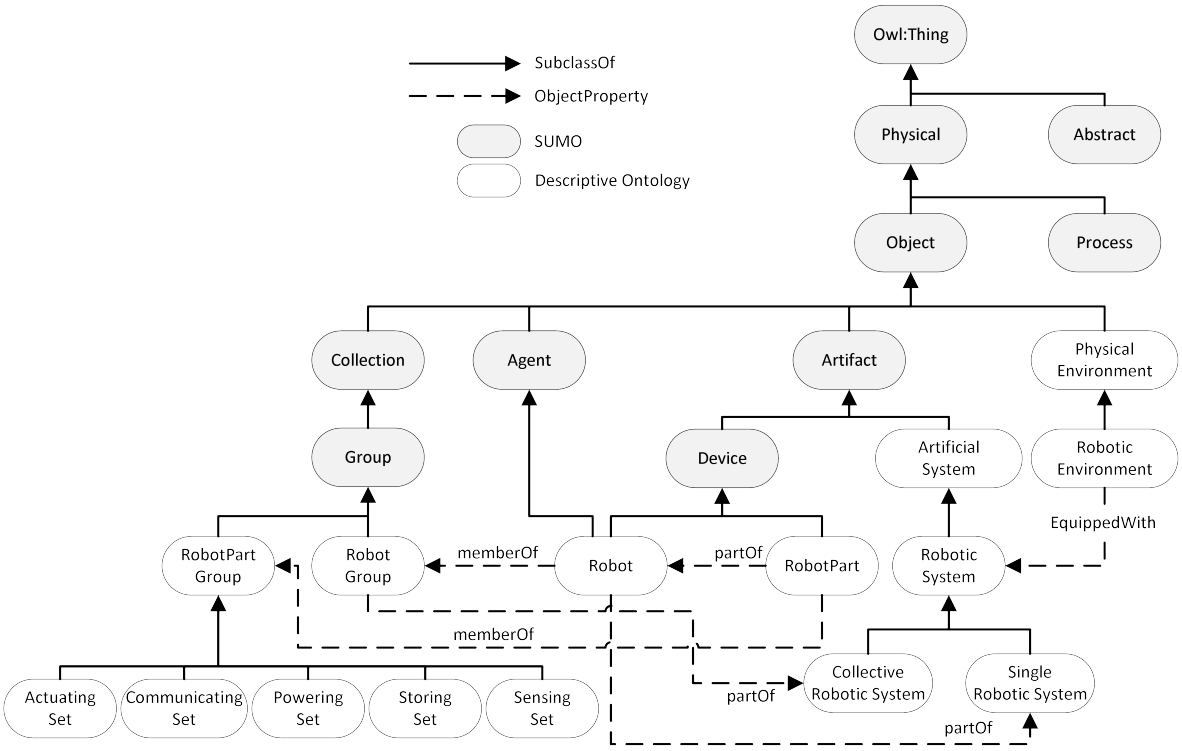


Figure 55.: Part of the descriptive domain ontology showing concepts related to Robot Group conceptualization.

The autonomy level of a specific robot will be determined by its ability to execute an aggregation of processes in order to achieve the final goal. Therefore, a relevant part of this descriptive domain ontology is the conceptualization of the diverse kinds of processes that a intelligent motion control system should be able to perform. SUMO has created a extensive and exhaustive taxonomy to describe process (SUMO:Process - the class of things that happen and have temporal parts or stages), and from it this Dissertation explores a group of conceptualizations that are relevant in the limits of the ontology's domain:

- SUMO:Power generation - Class of processes in which some kind of power is generated either for immediate use in a device or to be stored for future use, being classified

as fossil fuel, nuclear, electrical, or other types (in order to include any type of emerging power source). The power generated in any of these processes can later be used to supply (supplying process) or charge (charging process) a robot/robot part;

- SUMO:Nature process - A process that takes place in nature spontaneously, and therefore is the opposite of a powering process;
- SUMO:Motion - Conceptualizes any process of movement, and thus includes the motion control processes required to control an intelligent motion control system;
- SUMO:Automated process - An automated process is some process that is not a natural process and does not require the agent to be some cognitive agent. The agent will usually be an automated system. Respecting the boundaries of the domain, the descriptive ontology describes communicating process, sensing process, processing process, actuating process and storing process as a combination of conceptualizations between automated process and another specific process identified by SUMO, as illustrated in Figure 56;
- SUMO:Communication - A social interaction that involves the transfer of information between two or more cognitive agents. A communicating process, besides being an automated process, involves the transfer of information between two robots or a robot and an human;
- SUMO:Computer process - Process which manipulates data in the computer. Depending on the abstraction level at which a process is observed, any process can manipulate data in a computer, due to the fact that any actuating, sensing, communicating, storing or processing process relies on manipulated data to perform. An exception to this rule is a supporting process executed by a platform part, in order to hold all the parts of the robot together, allowing the robot to have the required performance;
- SUMO:Making - The subclass of creation in which an individual artefact or a type of artefact is made. A motion control process can be a making process in which an artefact (SUMO: An abject that is the product of a making) is produced.

The achievement of a goal is accomplished through the aggregation of simple and complex processes, described using the conceptualizations explored so far, but the performance of the robot or the performance of one of its parts is directly dependent on its properties, both hardware (describes an observable quality of a robot or one of its robot parts) and software properties, illustrated in Figure 57. A property can be ontologically described accordingly with its quantity kind (an abstract classifier that explores the concept of kind of physical quantity, representing the essence of a quantity without a numerical value or unit [CBB+12], *e.g.* frequency, velocity) and its physical unit of measurement (an abstract

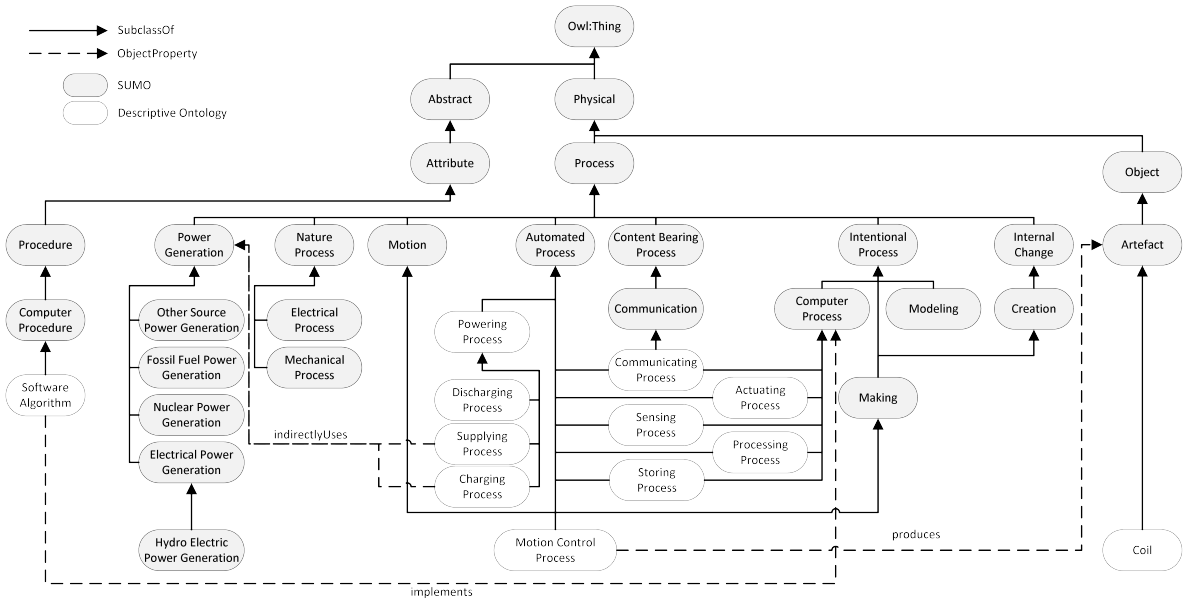


Figure 56.: Part of the descriptive domain ontology showing concepts related to process conceptualization.

classifier that represents a "real scalar quantity, defined and adopted by convention. This way, any other quantity of the same kind can be compared to express the ratio between of the two quantities as a number" [W3C14], *e.g.* Hz, m/s).

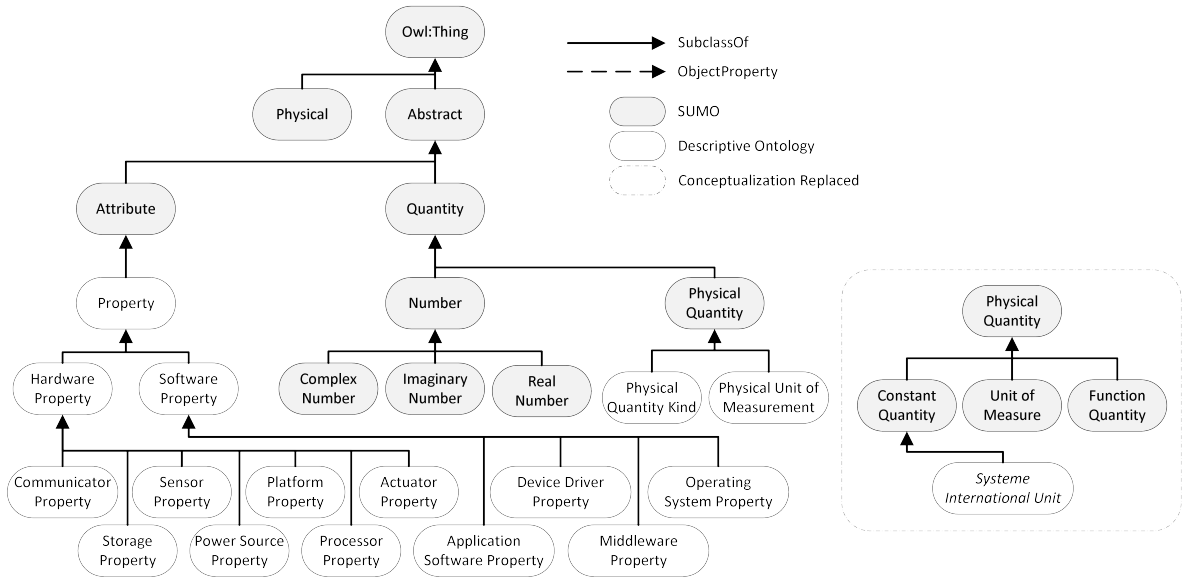


Figure 57.: Part of the descriptive domain ontology showing concepts related to property conceptualization.

A robot has an operating range (a hardware property that specifies the environmental conditions and characteristics of a robot part/robot's normal operating environment) and a

survival range (a hardware property that specifies the conditions a robot part/robot can be exposed to without damage - if the robot part/robot is damaged, its capability specifications may no longer be the same). Besides the operating and the survival ranges, each robot part is described through a unique group of properties, as illustrated in Figure 58 for a sensing part and in Figure 59 for an actuating part. This separation of concerns allows the specification of the characteristics of each robot part, increasing the knowledge inherent in the process and granting an *accurate* description on real-world *things*. Therefore, an actuating part is characterized through its control signal (which describes the source of the control signal that controls the referred actuating part as electrical, hydraulic or pneumatic), energy source (which describes the source of energy that provides power to an actuating part as alternative, electric current, Hydraulic Pressure, Magnetic Energy, mechanical force, pneumatic pressure or thermal energy), and actuating properties (describes a physical actuating metric of a robot part, *e.g.* efficiency and actuating time), while a sensing part is characterized only through measuring properties (*e.g.* accuracy, detection limit and resolution). The referred properties and the properties of the remaining robot parts are illustrated in the annex A.

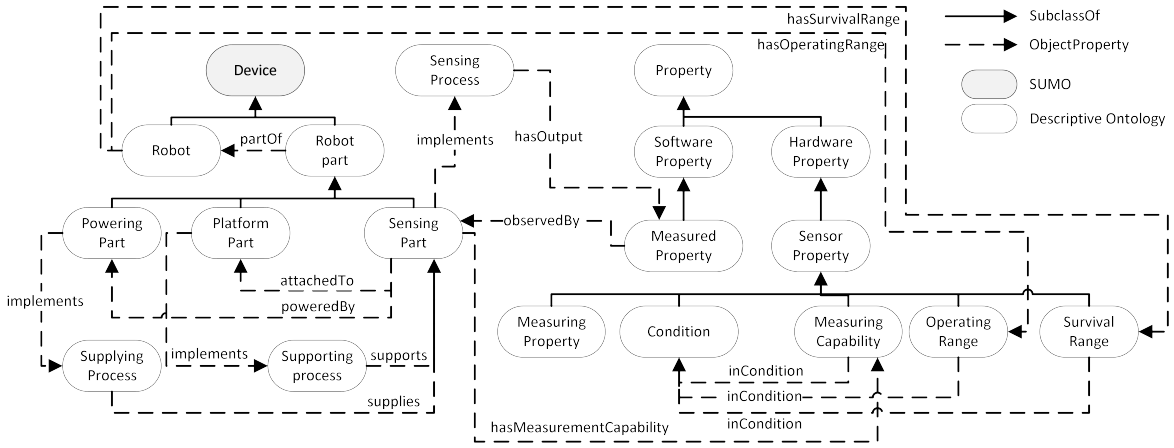


Figure 58.: Part of the descriptive domain ontology showing concepts related to sensing property conceptualization.

The relation between robot, robot part and hardware property allow this ontology to accurately describe a intelligent motion control system's hardware structure. Nevertheless, besides the hardware, a robot is composed by software too. Therefore, and in order to reduce the gap between the ontological perspective of the system and the software embedded in the real-system, a relation between robot, robot part, process and event was created and illustrated in Figure 60. This relation enlighten the system's designer about two kind of events that any robot part can possess - an **observation** or an **order**. In this Dissertation perspective, and following the design patterns created by the remaining ontologies in this domain, an event can be either a command (an action issued by a robot part and requested by a process, in order to move and/or act in the surrounding environment, or in the robot

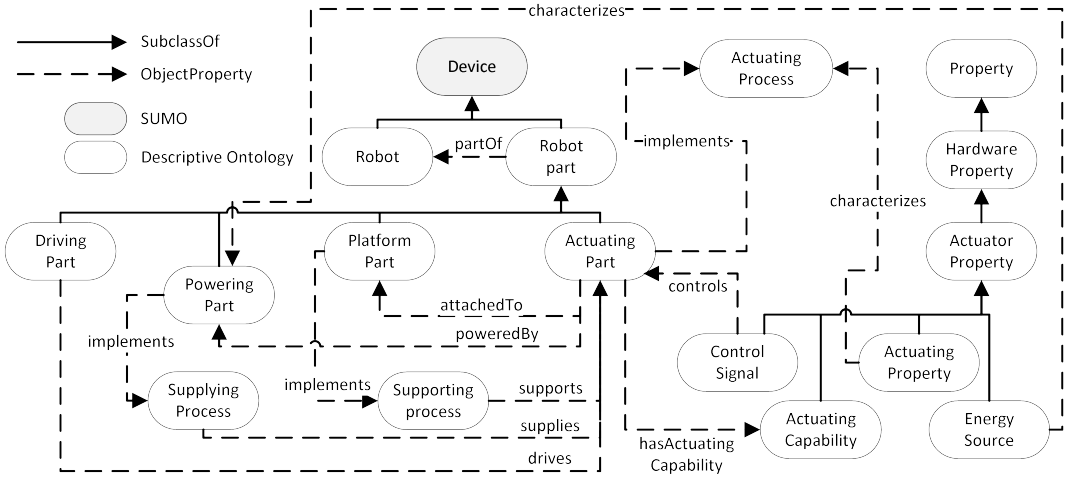


Figure 59.: Part of the descriptive domain ontology showing concepts related to actuating property.

itself) or an observation (a situation that describes the observation of a software property, controlled by the process which requested the observation, through a robot part).

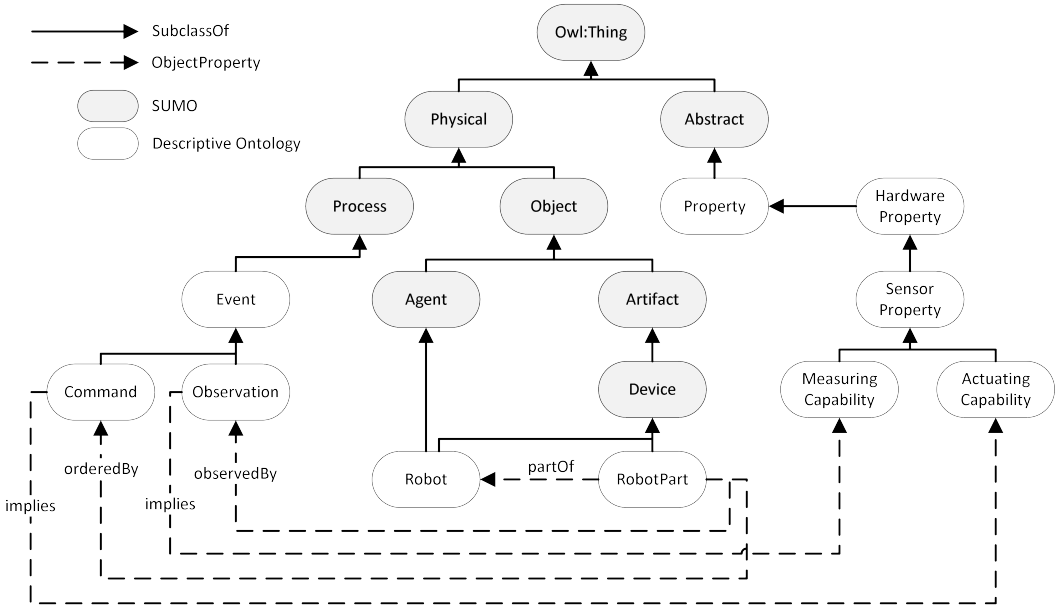


Figure 60.: Part of the descriptive domain ontology showing concepts related to Event conceptualization.

This ontology describes intelligent motion control systems' architecture, behaviour, physical environment, interaction with other systems/humans and working flow. Following the methodologies referred in this Masters Dissertation, the last step is the documentation and implementation of the ontology - this subsection documents the main parts of this ontology (and the remaining is explored in the annex A) and the implementation will be concluded after the creation of the prescriptive version of this descriptive ontology.

5.2.2 *Prescriptive Domain Ontology*

The prescriptive domain ontology for intelligent motion control systems, contrarily to a descriptive domain ontology, is a metamodel that originates a model, and subsequently a domain-specific modeling language, that describes the structure and behaviour of reality and **reality is constructed according to it**, constituting a specification of *reality*.

To create a prescriptive ontology, another use-case scenario was developed, allowing the prescriptive ontology to inherit some features implemented in the SeML, in order to optimize the ontology for the modeling tool. This subsection describes the *semantic refactoring* process necessary to do the referred conversion, and as a result, a prescriptive domain ontology is created, explored and implemented using *Protégé*.

Table 10.: Domain specification and use-case scenarios for the intelligent motion control systems' prescriptive ontology.

Use-case scenarios for the Intelligent Motion Control Systems' Prescriptive Ontology	
What is the domain?	• Intelligent Motion Control Systems
What can the ontology be used for?	• Understand system's properties, features and hierarchical breakdown structure • Describe system's variability • Classify an Intelligent Motion Control System accordingly with its autonomy level • Customization and automation for Intelligent Motion Control System's implementation with generative code generation
What questions should the ontology answer?	• What are the main components of a Intelligent Motion Control System? • What are the characteristics of a Intelligent Motion Control System? • What problems can an Intelligent Motion Control System solve? • How is an Intelligent Motion Control System's feature related to a goal? • What are the possible behaviors of a Intelligent Motion Control System?
Who will use and maintain it?	• It will be used, populated and maintained by its developer in order to be integrated with another established ontologies, aiming at full coverage of the robotics and automation domain.

As aforementioned, the SeML provides a semantically-enriched domain-specific modeling language that aims at describing software/hardware system's variability, concerning the customization and automation for system's implementation with generative code generation. Therefore, the concepts, relations and rules created in the descriptive domain ontology that are not focused in the description of system's variability constitute an overhead in the semantic analysis and interpretation of the ontology, executed by the SeML. Consequently, the **semantic refactor** necessary to convert a descriptive domain ontology into a prescriptive domain ontology (see Figure 10) focused on the following steps:

1. Ontology's core restructuring - The upper ontology integrated in the SeML allows a concise but brief description of system's hardware structure, which in turn forces the ontology's designer to rethink and reformulate the descriptive domain ontology's core structure by removing the hardware properties and retain only the software properties (system's properties related with code execution). Nevertheless, some hardware properties, depending on the application, can be directly related with code execution, and thus the hardware properties should only be removed from the ontology after analysing the application and correctly identifying the properties that are related with code execution. Consequently, this part of the semantic refactor will be postponed until the development of the prescriptive application ontology, in the next section;

2. Object properties and rules' adaptation - During the process of import and semantically analyse an ontology, the SeML will process every kind of concept, relation and axiom, but only the object properties **recognized by the upper ontology** will aid in the process of automate systems' customization and implementation. Therefore, every object property that have not a direct correlation in the upper ontology or does not increases the domain knowledge conducive to the description of system's variability, will be eliminated, while the remaining object properties will be maintained or reimplemented;

3. Process analysis - Equivalently to the semantic refactor executed with the core structure of the ontology, the conceptualization of process was modified, in order to describe only software processes, instead of describing hardware processes too, *e.g.* Charging Process, Supplying Process, Supporting Process, etc.

To implement the prescriptive domain ontology, a semantic refactor was applied to the descriptive domain ontology and used Protégé, alongside with the Pellet as a editor/framework and reasoner, respectively. This step supported the development of an unambiguous and precise domain ontology, and enabled the capability of reasoning and querying. This feature allowed the ontology to understand the inherent knowledge through logical inferences, useful for its classification, debugging and querying. Since the prescriptive application ontology is an instance-of the prescriptive domain ontology and results from the descriptive domain ontology already illustrated and explored in the last subsection, the prescriptive domain ontology's implementation will be illustrated during the analysis of the prescriptive application ontology.

5.3 CASE STUDY: IMCS'S METAMODELING

In this section, a detail analysis on a specific application in the intelligent motion control systems' domain is provided, in particular an intelligent motion control system implemented at Jilin University as part of the project presented in Chapter 3. Conducive to this objective, the hardware/software structure, behaviour and variability of a **coil winding machine** will be explored, resulting in the development of prescriptive application ontology, which in turn will lead to the creation of the application's model.

5.3.1 *System's Overview*

Developed under the Chinese industry plan "Made in China 2025", the coil winding machine follows the principles of this initiative, constituting an innovation-driven, efficient and high quality system that is integrated in the IMCSP developed at Jilin University. The long term objective is the development of a platform capable of being used as a hardware and software reference architecture for diverse kinds of intelligent motion control systems, *e.g.* 6-axis robotic arm, hexacopter, coil winding machine, UGV, etc. Since each application has different implementation's adversities, a development plan was created, illustrated in Figure 61. Considering the pitfalls and obstacles inherent in the development of each system, the coil winding machine, because it only presents problems that can be directly solved by adjusting the motion control algorithms, was the first system to be implemented. As a final objective, the implementation of the 6-axis robotic arm stands as the biggest engineering and economic challenge, outstripping the hexacopter (which has its size, weight and portability as a huge design challenge). The coil winding machine explored in this and in the following Subsections is a consummated industry product, already applied with success in a Mktech factory which focuses on the production of electronic components, *e.g.* a coil.

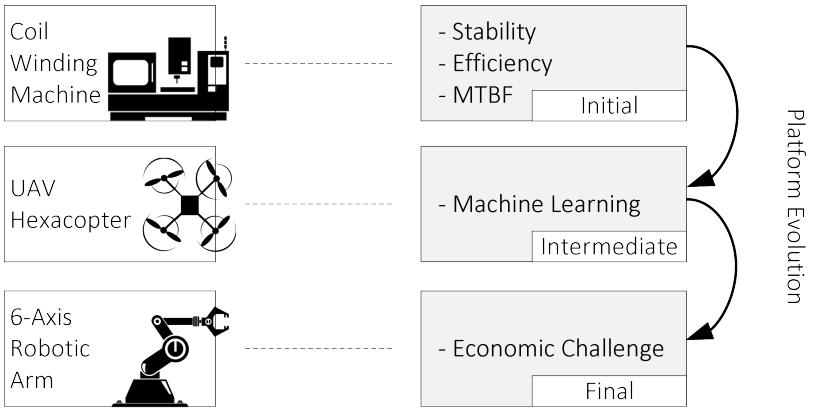


Figure 61.: IMCSP's evolution and future work.

5.3.2 *System's Structure*

The coil winding machine is divided between two subsystems, each one of them with specific functions and features, aiming at building a coil and at providing specific feedback about its implementation to the industrial worker, in order to allow a correct management of the global industrial process. Figure 62 illustrates the coil winding machine's workflow, indicating the specific inputs, outputs and processes.

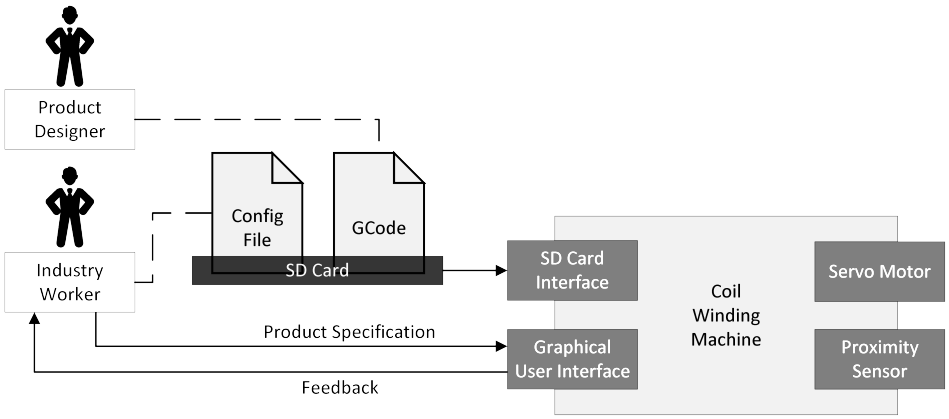


Figure 62.: Coil winding machine working flow's overview.

The first part of the coil winding machine is responsible for managing all the peripherals and processes that allow the product designer and the industry worker to interact meaningfully with the system. Analysing the Figure 62, it's clear that this interaction is a three step interaction:

1. System's properties definition - While the product designer provides a G-Code file with all the G-Code commands to realize each one of the coils available in the product line of the company, the industry worker contributes with a file that includes the following configurations:
    - Modbus protocol - An internal communication protocol for which variables must be initialized. Examples of Modbus properties are the aggregation of 136 toggle objects, mode, slave address, port, *baudrate*, parity, number of stop bits and word width;
    - G-Code - Depending on the G-Code files available, the memory allocated for G-Code and the memory to store it must be defined;
    - Interpolation - Define the memory allocated for the interpolation buffer and the memory to story it;
    - Servo motor - Define the actuators' maximum velocity, acceleration, jerk and range.

2. Product specification - The industry worker uses the graphical user interface to choose a product between the products available in the file provided by the product designer during the first step;

3. Feedback - During the production of the chosen product, the system provides continuous feedback (through the graphical user interface) to the industry worker, allowing the management of all the industry process through the following parameters:
   - 3-axis real-time position;
   - Angular displacement;
   - Rotational speed - Speed in loops/minute;
   - Period - Time to complete a loop;
   - Required loops - Presents a relation between the number of loops required to complete the product and the number of loops concluded.
   - Required file - Presents the name of the file that contains the G-Code of the product chosen.
   - Steps - Presents a relation between the number of G-Code steps required to complete the product and the number of steps concluded.

Apparently simple as an overview, Figure 62 hides a complex hardware architecture that composes the coil winding machine. Aiming at performing accordingly with the characteristics provided by each motion control algorithm, the coil winding machine is composed by two distinct parts:

- STM32F103ZET6 - This 32-bit Cortex M3 high-density performance line microcontroller is the *heart* of the user interface, using:
  - The Modbus protocol to integrate a Human Machine Interface (HMI) MT6070iH;
  - Standard Digital Input Output interface to support an external memory expansion (SD Card);
  - Flexible Static Memory Controller (FSMC) protocol to interact with two external memories, a IS61LV25616 (a 256K x 16bits Static Random Access Memory (SRAM)) and a K9XXG08UXB (a 128M x 8bits NAND FLASH memory);
  - RT-Thread, a Chinese open-source real-time operating system for embedded systems, as a provider of the necessary device drivers, board support package, middleware and a 8, 32 or 256 priority multi-threading scheduling with object oriented programming style support.

- Cyclone IV EP4CE40F23C6 - A 328 IOs Field-Programmable Gate Array (FPGA), responsible for the execution of the hardware accelerators, both the interpolation and

the velocity control algorithms. As an alternative for the software implementation, the hardware implementation increases sytem's performance. To do so, the FPGA is directly connected with at least four servo motors (HBS2206), one for each axis (x,y,z) and one for the rotational motion (θ), and receives feedback from a set of proximity sensors (EE-SX672).

To perform and achieve the desired goals, this hardware architecture is complemented with a technology stack, illustrated in Figure 63. Within the illustrated technology stack, the sources of system's variability is manifested in the software algorithms, allowing the performance of the system to be directly dependent on the algorithms selected to control the actuators, *videlicet* the algorithms that combine to create the motion control algorithm, which dependencies are illustrated in Figure 64.

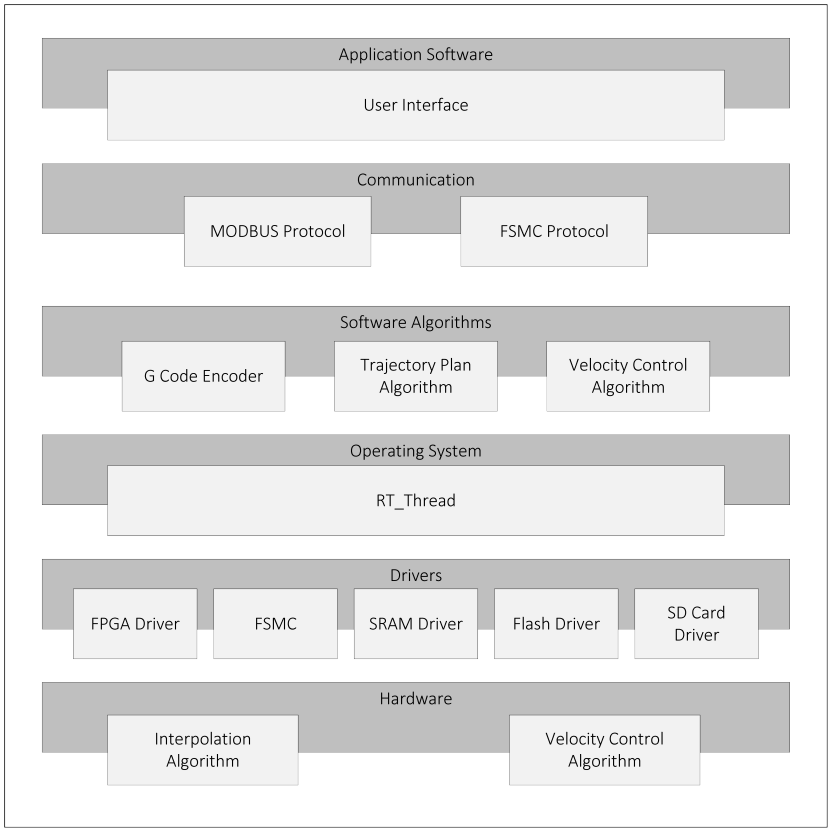


Figure 63.: The technology stack of the coil winding machine.

The trajectory plan, the interpolation algorithm and the velocity control algorithm are combined to create a motion control algorithm, aiming at controlling the four actuators that integrate their actions to create the coil for which the G-Code available in the SD Card (provided by the industry worker). G-Code provides information about the tool and the type of movement to perform: where to move, how fast and what path to follow. After encoding the G-Code, the G-Code encoder saves the result, the **interpolation points**, that subsequently

are used by the interpolation algorithm to **divide a complex segment** into small and individual segments (identified by the **start point**, **end point** and **distance** between them). Lastly, the velocity control algorithm uses these variables to calculate the velocity curve to apply to the servo motor (HBS2206) in order to complete the referred segment, as illustrated in the left side of the Figure 64. Depending on system's requirements, the Interpolation Algorithm and the Velocity Control Algorithm require different implementations. The next subsection explores the possible implementations for each one of these algorithms and the relation between them and the system's performance.

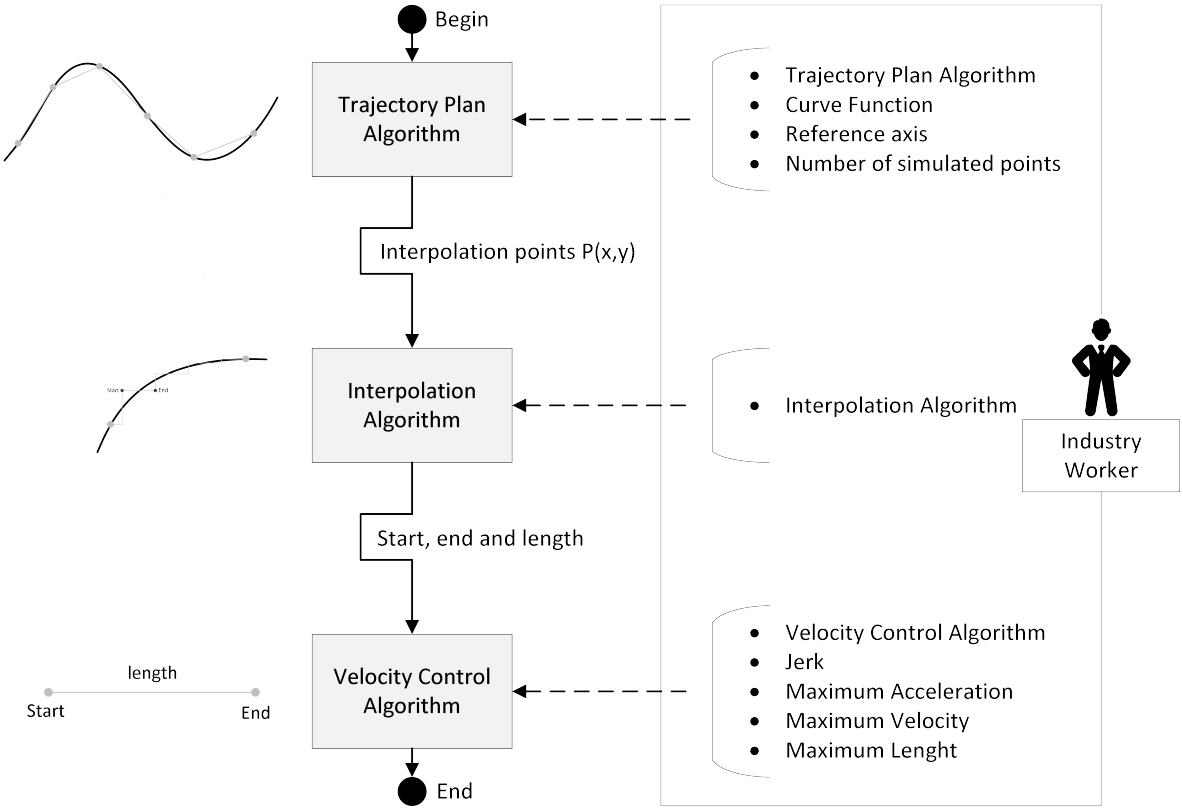


Figure 64.: Dependencies between the diverse algorithms that compose a motion control process.

5.3.3 *System's Variability*

In a motion control process, the trajectory plan algorithm optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality, and, as a result, it provides an aggregation of points that will serve as input to an interpolation algorithm. This algorithm is extremely costly in terms of processing resources, and therefore is not executed by a coil winding machine, but by a general processor, which subsequently provides the G-Code file to be used in each product (that later is stored in an external

memory by the product designer, as illustrated in Figure 62). Therefore, contrarily to the interpolation and velocity control algorithms, the trajectory plan is part of the Commercial Off-The-Shelf (COTS) component of the IMCSP's project.

The next paragraphs resume the work of Prof. Quangang (project manager of the IMCSP and co-advisor of this Masters Dissertation), which resulted in the submission of 4 Journal papers. Since some of the work developed to create and improve this algorithms is under a non-disclosure agreement, this subsection only presents an overview of the analysis of the corresponding algorithms.

INTERPOLATION ALGORITHMS

The interpolation algorithm uses the interpolation points, computed by the G-Code Encoder from the G-Code file, and calculates the start, the end and the length of each segment that composes the path to be followed by the Robot. To do so, the coil winding machine presents four different algorithms implemented in hardware, using the FPGA Cyclone IV EP4CE40F23C6. The variability inherent in an interpolation algorithm resides mainly in choosing between a low or high accuracy implementation, due to the fact that, since all the algorithms are implemented in hardware, their performance is very similar but the resources' consumption level diverges. In an initial phase of this project, the Digital Differential Analyser (DDA) and the PBP algorithms were the only alternatives, and so, due to their low accuracy, an improved alternative for both algorithms was developed, verifying that the Improved PBP algorithm presents better performance in terms of accuracy and chord error, as illustrated in Figure 65.

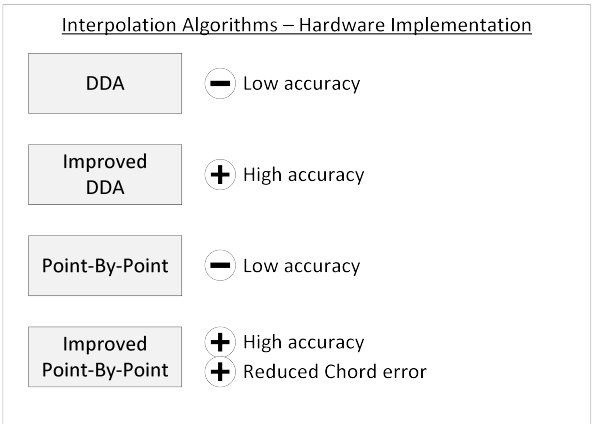


Figure 65.: Advantages and disadvantages of each IMCSP's interpolation algorithms.

The PBP algorithm provides a step-by-step linear or circular interpolation, being the decision of the following step made after the conclusion of the current step. As illustrated in Figure 66, the PBP algorithm provides three options in each point. As an example, lets consider **O(0,0)** the start point, **P(x,y)** the final point (the objective), and **N** as the current

point. At **N**, the next step can be suggested to follow the x-axis (reaching **C**), the y-axis (reaching **A**) or both (reaching **B**).

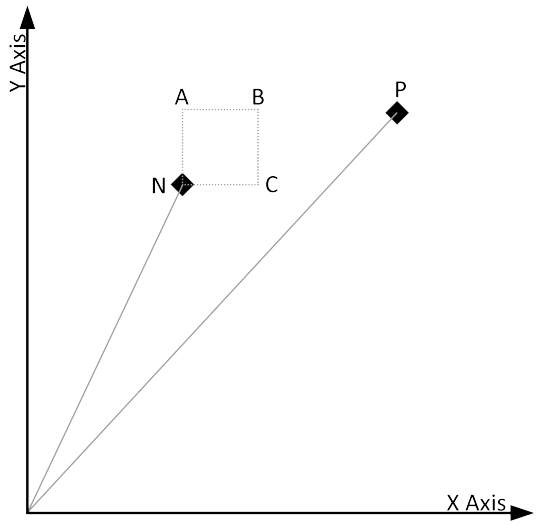


Figure 66.: PBP algorithm's working fundamentals.

The following equation defines the path that should be followed:

$$S_n = X_p Y_n - Y_p X_n$$

- S_n >0 - In this case, the point N is above the objective line (OP), and thus the next step will be taken towards the x-axis (X_n+1,Y_n);
- S_n = 0 - In this case, the point N is in the objective line (OP), and thus the next step will be taken towards the x-axis (Xn+1,Yn) or the y-axis (X_n,Y_n+1);
- S_n <0 - In this case, the point N is under the objective line (OP), and thus the next step will be taken towards the y-axis (X_n,Y_n+1);

Nevertheless, this algorithm requires both multiplications and additions, constituting a costly process when deployed in a FPGA. Therefore, an improved PBP algorithm was developed, which is based in the iterative relation existent between the next step and the current step, and allows the reduction of logic gates necessary to implement this algorithm. Besides decreasing the amount of hardware resources, this algorithm presents a reduced chord and trajectory errors, when compared with the PBP, thus resulting in a optimized performance in terms of time.

DDA is an interpolation algorithm used for interpolation of variables over an interval between two specific points, being useful for *rasterization* of lines, triangles and polygons, and thus useful for motion control processes. Accomplished using floating-point or integer arithmetic, the implementation of this algorithm constitutes a complex process when

deployed in a FPGA. This algorithm solves two types of interpolation, circular and linear, and it was improved on the following points:

- Linear Interpolation:
  1. Reduction of the equivalent pulse - Reducing the pulse directly decreases the size of each step, and thus reduces the calculation time and the chord error associated to the interpolation.
  2. Adjust the accumulator's capacity.
- Circular Interpolation:
  1. The use of secant instead of tangent increases the path accuracy rate but duplicates the computational time.

As an overall performance, the improved DDA represents a better accuracy, due to the reduction of the chord error between 20%-50% when compared to the DDA.

VELOCITY CONTROL ALGORITHMS

Contrarily to the interpolation algorithm, the velocity control algorithm is implemented in both software and hardware. The hardware implementation of this algorithm was successfully accomplished through three methods - the trigonometric Look-Up-Table (LUT), the CORDIC iteration and the CORDIC pipeline. Nevertheless, aiming at solving the insufficiencies and weaknesses of the existing methodologies, a forth algorithm was implemented, the trigonometric iteration, as illustrated in Figure 67.

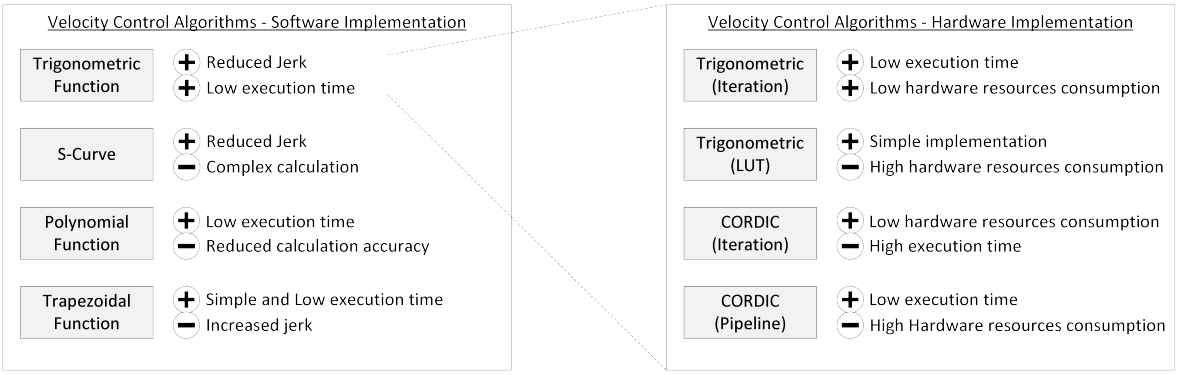
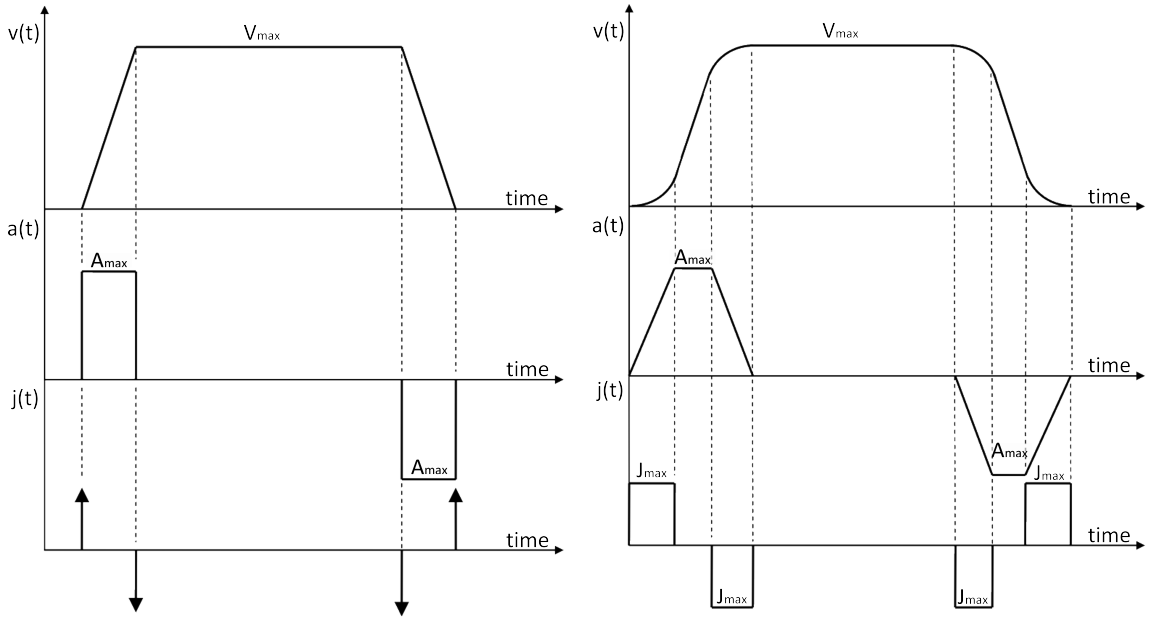


Figure 67.: Advantages and disadvantages of each IMCSP's velocity control algorithms.

These algorithms control one or diverse servo motors in order to accomplish the desired task, *e.g.* moving an object from one point to another. Hypothetically speaking, to achieve the best performance (finish a goal in the minimum possible time), the motor would attain top speed, V_{max} instantaneously, would reach its destination at V_{max} and, lastly, comeback to halt position instantaneously too. This case is impossible due to the infinite acceleration

that would be necessary to accomplish this requirements, as can be demonstrated by the following explanation.



(a) Case1: Jerk assumes an infinite value.

(b) Case2: Jerk is limited.

Figure 68.: Two distinct scenarios where the relation between the jerk and the acceleration is analysed.

The distance between two points is denoted as d and has units of m, being the instantaneous value of the displacement represented as a function of time, d(t). This displacement can be obtained at any point in time by integrating the velocity equation, v(t):

$$d(\tau) = \int_0^\tau v(t)\ dt$$

Subsequently, the velocity can be evaluated by integrating the acceleration, which means that the velocity is equal to the area under the acceleration-time curve:

$$v(\tau) = \int_0^\tau a(t)\ dt$$

Lastly, trough the integration of the acceleration-time curve the jerk is obtained, which has units of m/s^2 and represents the rate of change of acceleration, and thus, it indirectly measures the impact in the motors and in the machine (being rapid changes in acceleration a factor that can lead to equipment wear and result in uneven performances):

$$a(\tau) = \int_0^\tau j(t)\ dt$$

Therefore, the jerk applied to a specific machine (*e.g.* the coil winding machine) is directly influenced by the velocity control algorithm applied to its motors, as illustrated in Figure 68. In the first case (68a), a trapezoidal function is used, which generates infinite jerk in a specific time interval and subsequently produces a huge impact in the equipment, reducing its lifetime expectations and its Mean Time Between Failures (MTBF). However, if instead of an abrupt change of velocity, this change happens gradually (as showed in case 68b) the jerk value is limited to a certain value, being this value as low as the slop of the velocity change curve. Therefore, the best approach to control a servo motor is using a trigonometric function, providing the necessary reliability and stability to the machine. As illustrated in Figure 67, the coil winding machine has 4 different hardware implementations of the trigonometric function, being the Trigonometric Iteration the most advantageous, due to the fact that its development combines the techniques of the other approaches and solves their insufficiencies and weaknesses.

5.3.4 *Design of the Coil Winding Machine Prescriptive Application Ontology*

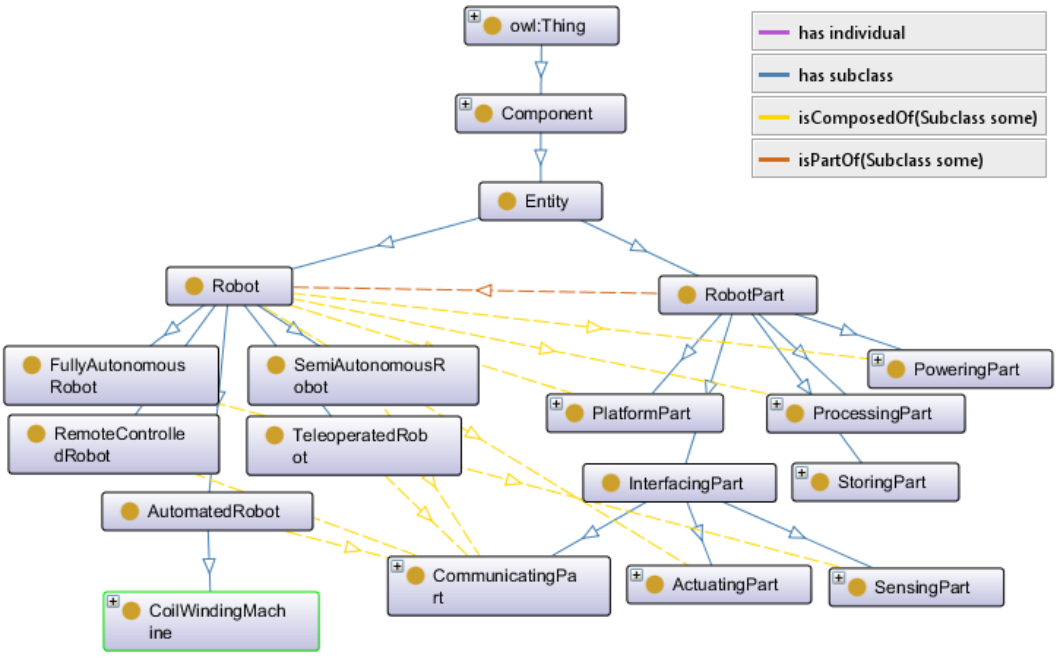


Figure 69.: Conceptualizations that result from entity's conceptualization specification.

The prescriptive ontology for the coil winding machine is an instance of the prescriptive domain ontology presented and discussed in the last section, inheriting and specifying all the semantic knowledge inherited from it. By doing so, this ontology explores the coil

winding machine following the prescriptive core ontology's perspective, provided by the SeML.

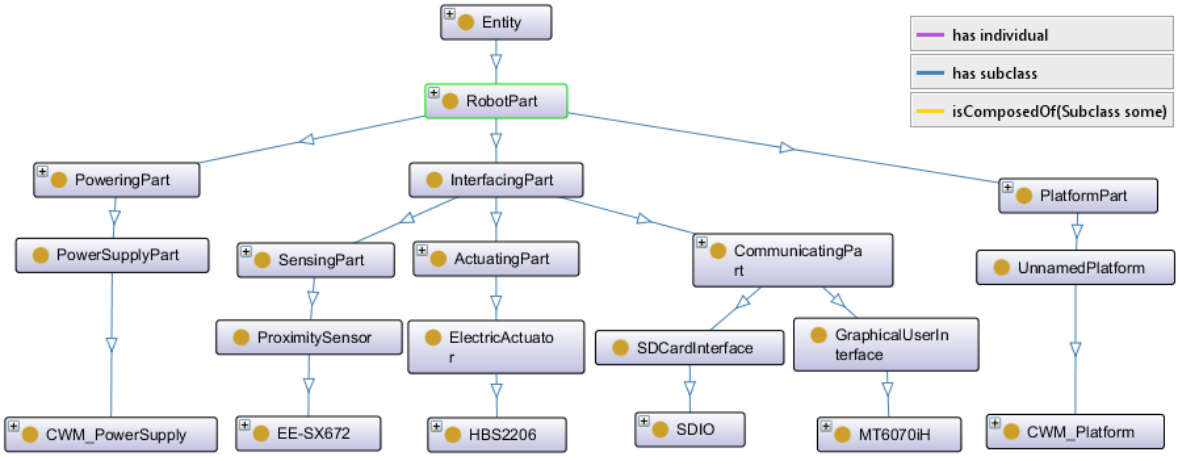


Figure 70.: Hierarchy of robot parts that can compose a robot.

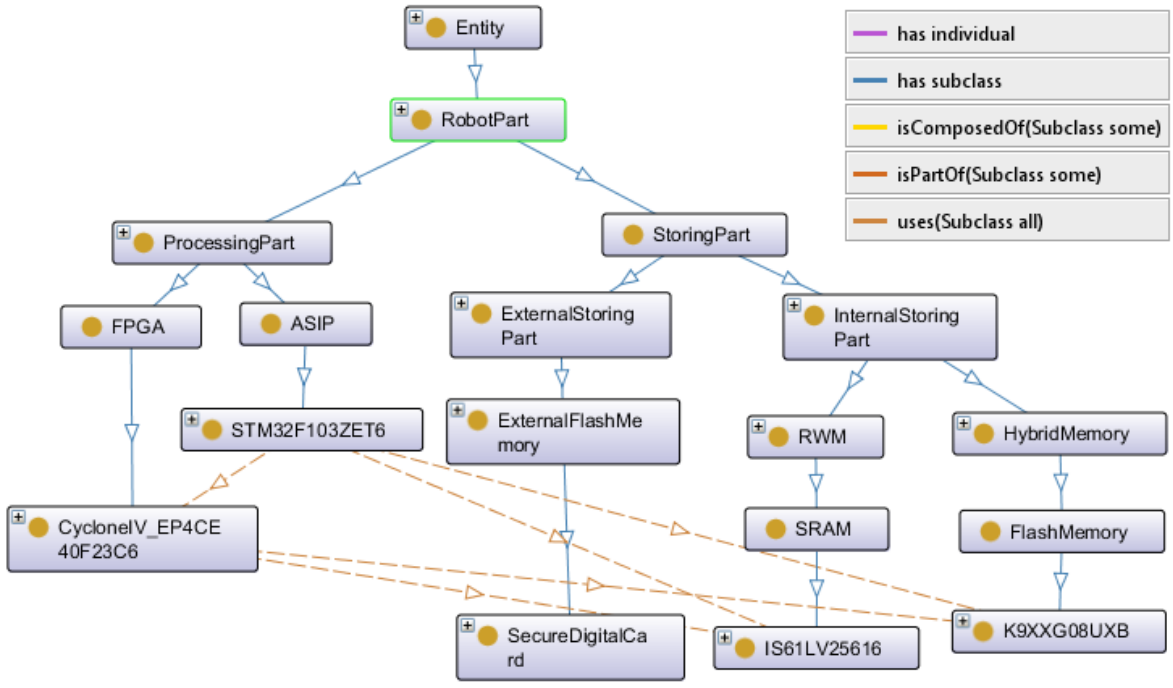


Figure 71.: Hierarchy of robot parts that can compose a robot.

Accordingly with the construction of the prescriptive domain ontology, the first conceptualizations should consider the entities existent in this application, and thus should be the creation of subclasses of each robot part that compose the coil winding machine and the classification of the robot itself. Figure 69 illustrates the classification of the coil winding machine as an automated robot, since it is a robot that can accomplish a goal or solve a problem

to which it was automated to, not adapting to changes in the surrounding environment. Subsequently, the composition of the Robot is specified accordingly to its hardware and software structure, as illustrated in Figure 70 and 71.

Since the knowledge about the classification and the structure of the coil winding machine is already integrated in the ontology, the next step should be the description of the characteristics (requirements/goals and features) of this application, and the respective problems to which they relate. As mentioned in Section 5.1, the objective of the SeML is to provide a semantically enriched environment for system's modeling accordingly with its variability, using the relation features-goals-problems to allow the system's designer to create the model. Therefore, when establishing the features, goals and problems of an application, a perspective over the application's variability should be provided. In the coil winding machine, as explored in the previous sections, the variability of the system is directly dependent on the motion control algorithms (trajectory plan algorithm, interpolation algorithm and velocity algorithm) and the properties that each one of them implies for desired system's behaviour. Therefore, the features, goals, and problems of the coil winding machine, are the same as the features, goals and problems of the motion control algorithms and its properties, due to the fact that the remaining parts of the system are static and does not influence significantly the system's performance. Figure 72 illustrates the application's characteristics and problems, which connected to the structure, processes and properties of the system, allow this ontology to provide the semantic knowledge necessary to extend the SeML grammar conducive to the creation of the application's model.

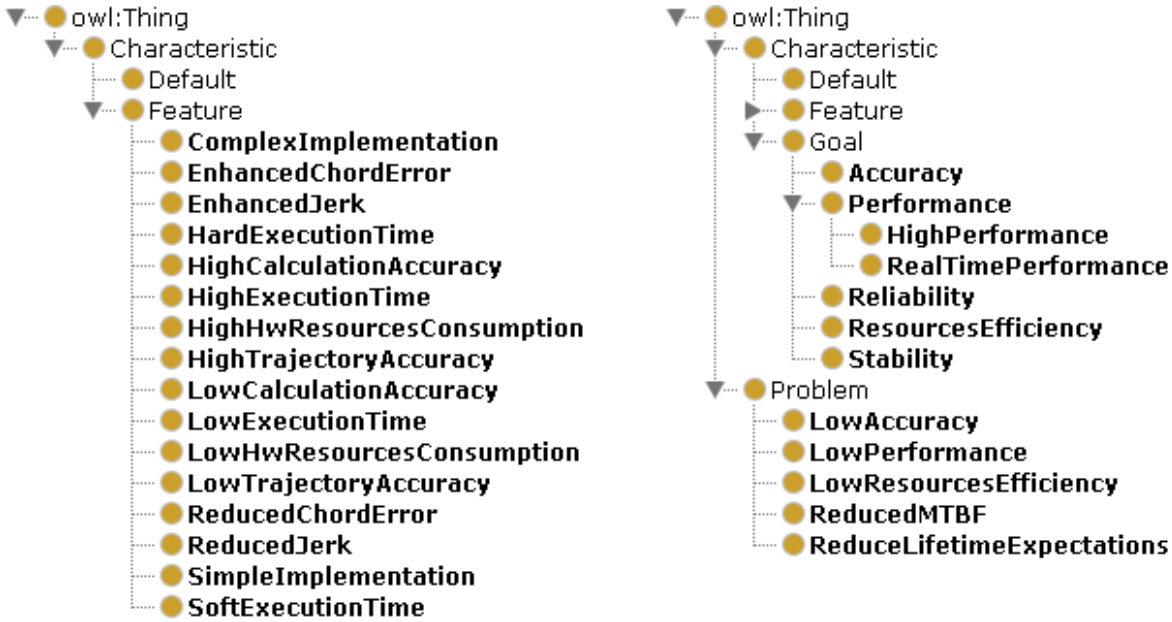


Figure 72.: Hierarchy of characteristics and problems of the coil winding machine.

The conceptualization of Process (inherited from the SeML's prescriptive core ontology), was specified as illustrated in Figure 73 in the prescriptive domain ontology.

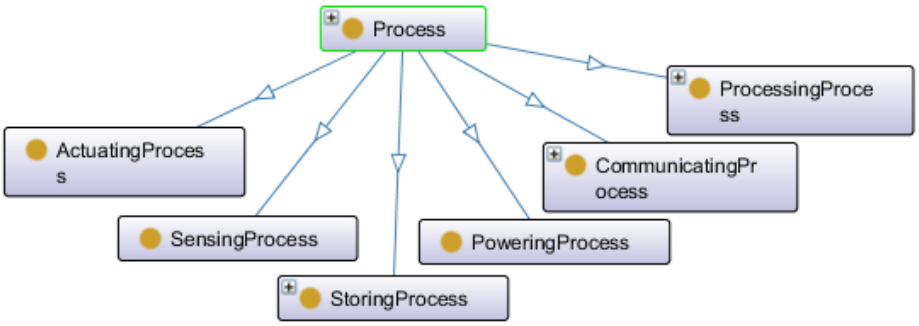


Figure 73.: Process's conceptualization in the prescriptive domain ontology.

The semantic refactor removed the specification of the processes that contain only knowledge about the physical behaviour of an intelligent motion control system, erasing the specification of the actuating process, sensing process, powering process and supporting process, and adapting the concepts of processing process, communicating process and storing process. Being one of the subclasses of process, processing process explores all the processes that are responsible for processing data and information necessary to proper and required operation of the robot, and so, the RT-Thread operating system (and subsequently the its middleware, device drivers and board support package), all the software/hardware algorithms and communication processes between system's parts (MODBUS and FSMC protocols) are covered by this concept, as illustrated in Figure 74.

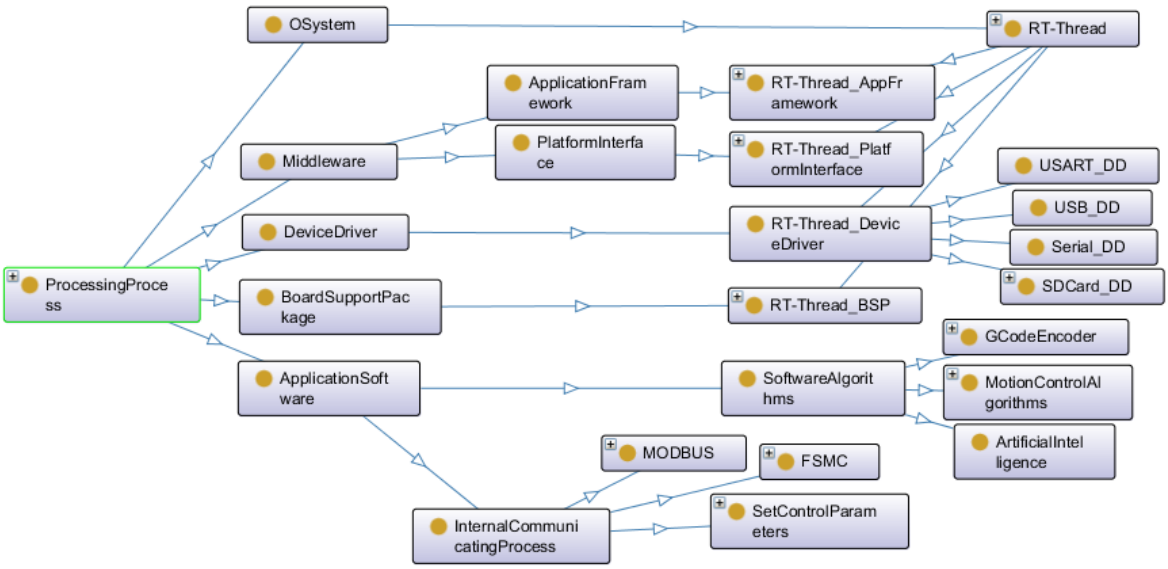


Figure 74.: ProcessingProcess's conceptualization in the prescriptive domain ontology.

A storing process is a process responsible for allowing the interaction between a robot and its storing part, and therefore, this concept encompasses knowledge about the software processes to access both the IS61LV25616 (SRAM memory) and the K9XXG08UXB (flash memory) storing parts, as illustrated in Figure 75.

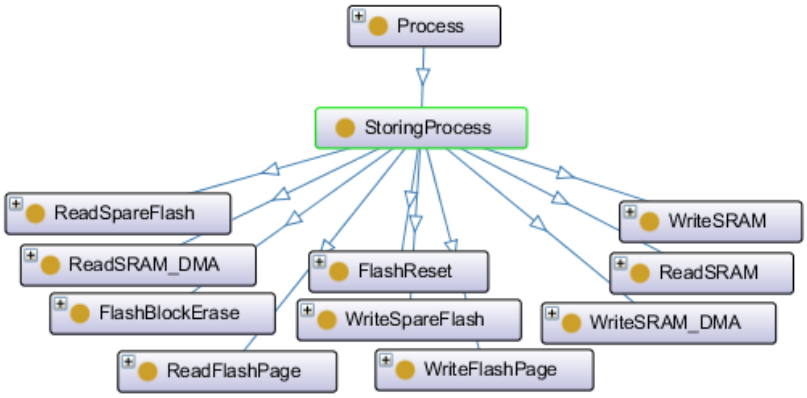


Figure 75.: StoringProcess's conceptualization in the prescriptive domain ontology.

Lastly, the communicating process provides knowledge about the software processes that enable the factory worker to interact with the machine, in order to specify the final product and show the machine's working parameters, as illustrated in Figure 76.

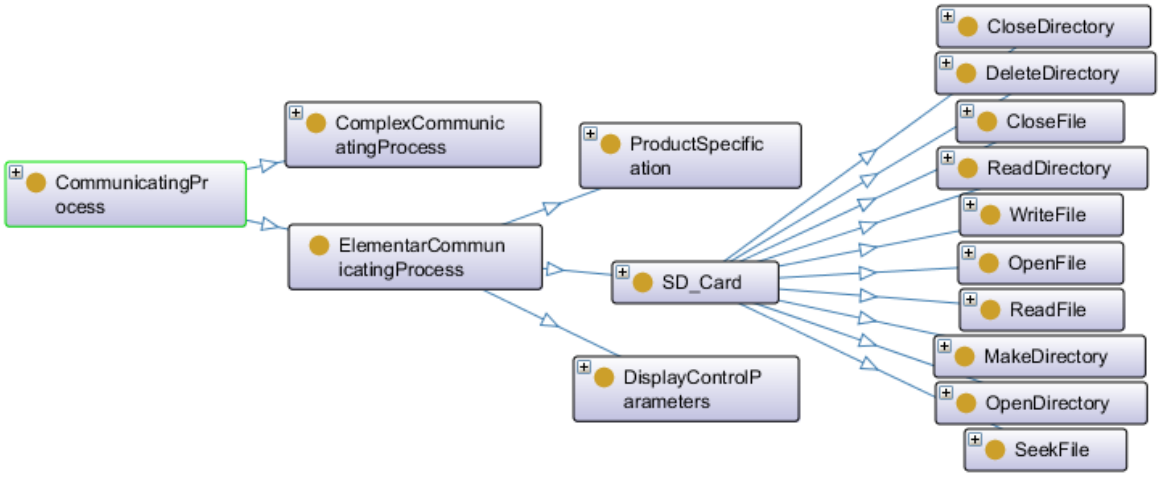


Figure 76.: CommunicatingProcess's conceptualization in the prescriptive domain ontology.

After specifying all the knowledge referent to the structure and behaviours of the coil winding machine, the prescriptive application ontology specifies the SeML's ontology concept property, and branches it between the properties of the software algorithms, the system's initial configurations and the properties of the communication protocols, as illustrated in Figure 77. From this knowledge, the properties of this application's intelligent motion control algorithms are explicitly defined as illustrated in Figure 78. Finally, to conclude the

core structure of the application's ontology, the properties of the internal communication protocols were identified as illustrated in Figure 79.

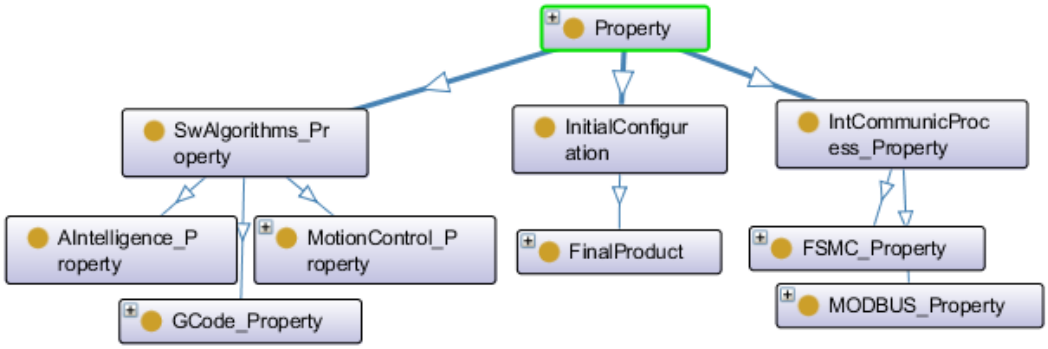


Figure 77.: SeML's property's conceptualization in the prescriptive domain ontology.

Figure 78.: Properties of a velocity control algorithm that has direct influence in system's variability.

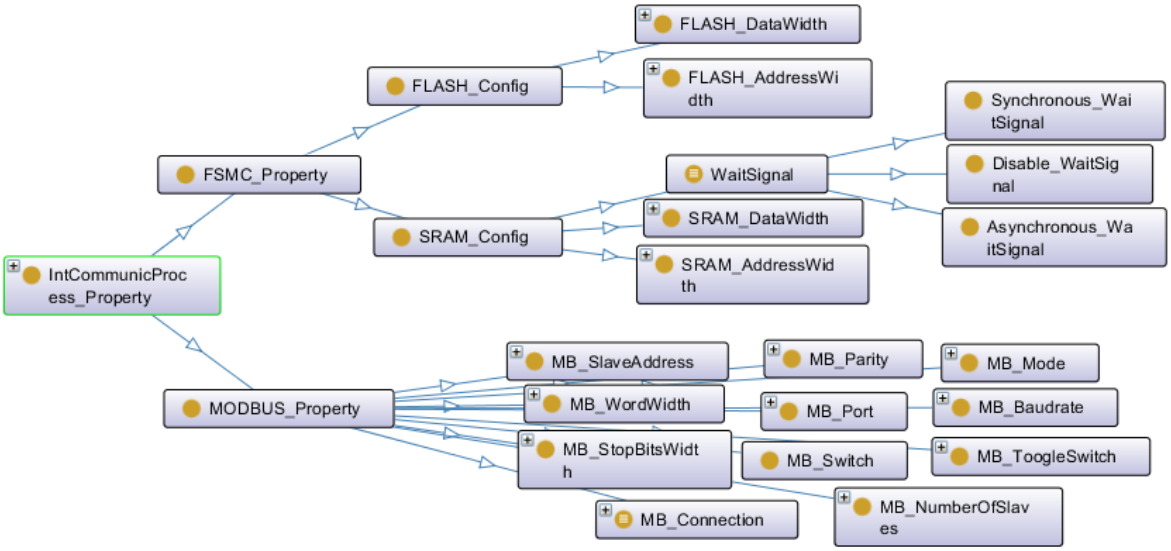


Figure 79.: Properties of the MODBUS and FSMC protocols that directly influence system's variability.

After specifying all the knowledge referent to the structure, behaviours and properties of the coil winding machine, the prescriptive application ontology must relate all the parts of the ontology to describe system's variability through the object properties existent in

the SeML's prescriptive core ontology. To do so, each goal is related (requires) to a feature, which in turn is associated with specific processes (demands) that solves a problem. Figure 80 illustrates a single example of the aforementioned relations implemented with Protégé.

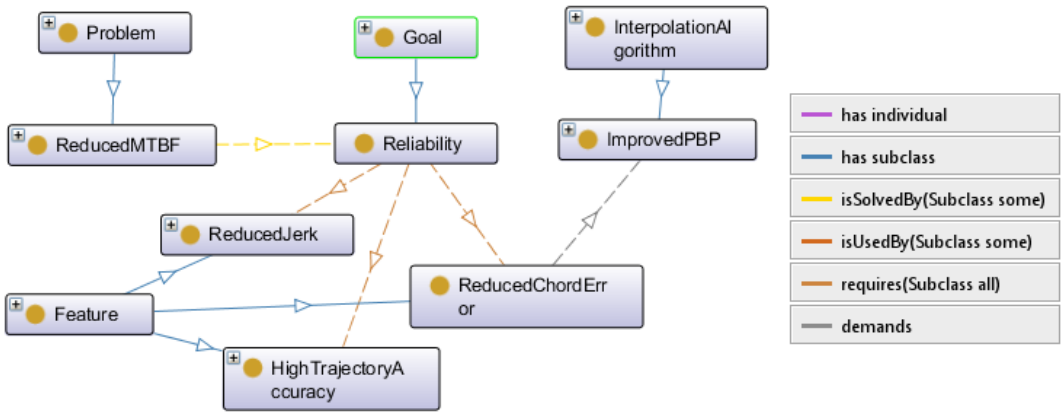


Figure 80.: Relation between Feature, Goal and Problem, using OntoGraf.

Figure 81 illustrates all the relations between the features and the processes of the coil winding machine, which is used to force the system's designer to specify (in the model) the correct process to accomplish the desired requirements or goals, as will be explored in the next section.

The prescriptive application ontology is concluded with the introduction of the aforementioned relations to describe system's variability. Nevertheless, it is necessary to create specific annotations in the ontology (using Prétége) that will allow the SeML to relate the system's model to the system's code, and enable the automation of the configuration and the implementation of the coil winding machine. The SeML provides tools, implemented in java, that enable the following functionalities:

- **Replace** - This tools is capable of replacing a specific keyword in the system's code by value/string specified in the system's model or by a default value specified in the prescriptive application ontology (in case the system's designer does not mentioned it in the system's model);

- **CheckExistence** - Verifies if a function is implemented in the system's code.

The code generation is divided in two main parts, the verification of the system's code interfaces and the assignment of the properties specified in the system's model. In the prescriptive application ontology, every process must contain an annotation that forces the SeML's compiler to check if a specific interface is implemented in the code, as illustrated in Figure 82 for one of the motion control algorithms. By doing this, if the system's designer specifies the Trigonometric_Iteration as the velocity control algorithm in the

Figure 81.: Relation between processes and features of the coil winding machine (adapted from Prétége).

system's model, during the code generation process the SeML generates the file Trigonometric_Iteration.v in the files' directory ApplicationSoftware/VelocityControl, and checks if the Trigonometric_Iteration function exists.

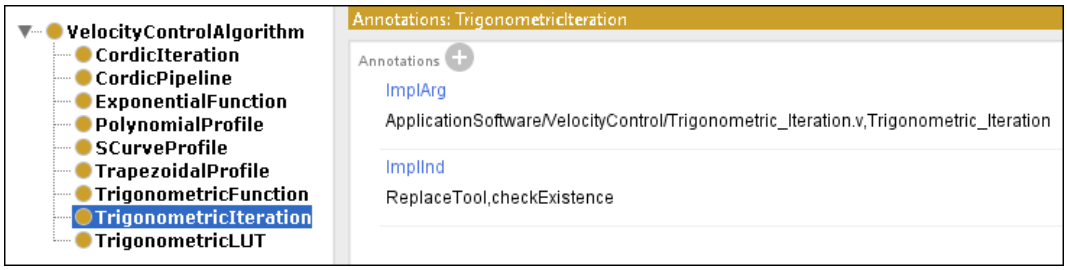


Figure 82.: Annotation that indicates the necessity of the Trigonometric_Iteration.v file's generation and the verification of Trigonometric_Iteration function's interface.

Subsequently, the ontology should specify the files and the location of each one of the system's properties related with code execution, as illustrated in figures 83 and 84. Figure 83 demonstrates how to create an annotation in the prescriptive application ontology in

order to replace a keyword in the system's code with a string. The compiler will search for the keyword in the file ApplicationSoftware/InitialConfig.h and replace it with the product specification provided by the system's designer in the system's model.

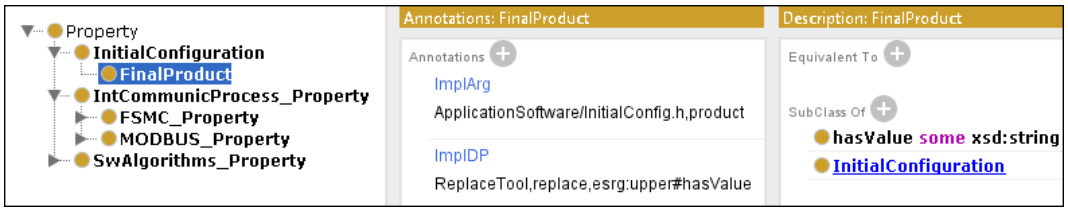


Figure 83.: Annotation that assists the SeML on the process of replacing the FinalProduct variable in system's code.

Figure 84 provides an alternative for the configuration of the system, which is the default values provided by the ontology. If, in the system's model, the system's designer does not configure one of the required system's properties, they will be replaced in the generated code with the default values defined in the ontology, *e.g.* in this specific case, the value of all the software switches of the MODBUS protocol will assume the default value 1.

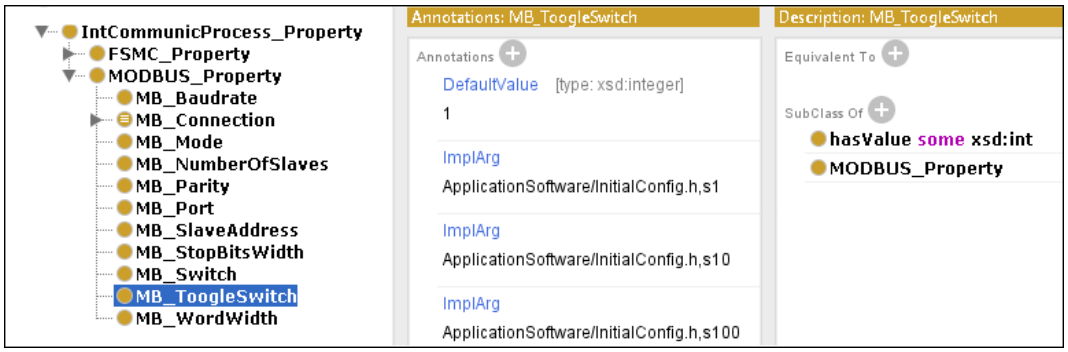


Figure 84.: Annotation that assists the SeML on the process of replacing a set of variables in system's code with default values.

This section explored the prescriptive application ontology, its relation with the prescriptive domain ontology and the mechanisms that allow the SeML compiler to associate the system's model (built with the knowledge provided by the hierarchy of ontologies) with the system's code. From the integration of these ontologies with the SeML results a DSML with the following characteristics (accordingly with the DSML framework discussed in the Subsection 2.1.2):

- Abstraction-level - Since the prescriptive application ontology contains knowledge directly related with system's code, the DSML is **platform-specific**;
- Domain - Due to the SeML's ontology's capability to describe system's variability, the resulting DSML can model systems' **behaviours** and its **structure**;

- Ontological kind of models - This language abstracts from the system's implementation by classifying several components based on their properties and/or behaviours, and thus, is a **type model language**.

In the next Section, the system's model will be developed and analysed, to understand how the developed ontologies assist the development of the model and the automation of system's configuration and implementation.

5.3.5 *Coil Winding Machine Metamodeling using the SeML*

The objective of this section is not only to demonstrate how the SeML was used to model the coil winding machine, proving that both the prescriptive domain and application ontologies are suitable for its purpose, but also to elucidate how the compiler (semantically-enriched by these ontologies and by the SeML's prescriptive core ontology) assists in the development of the system's model.

```
import "D:CWM/CWMO.owl"
```

Listing 5.1: Import of the prescriptive application ontology with the SeML.

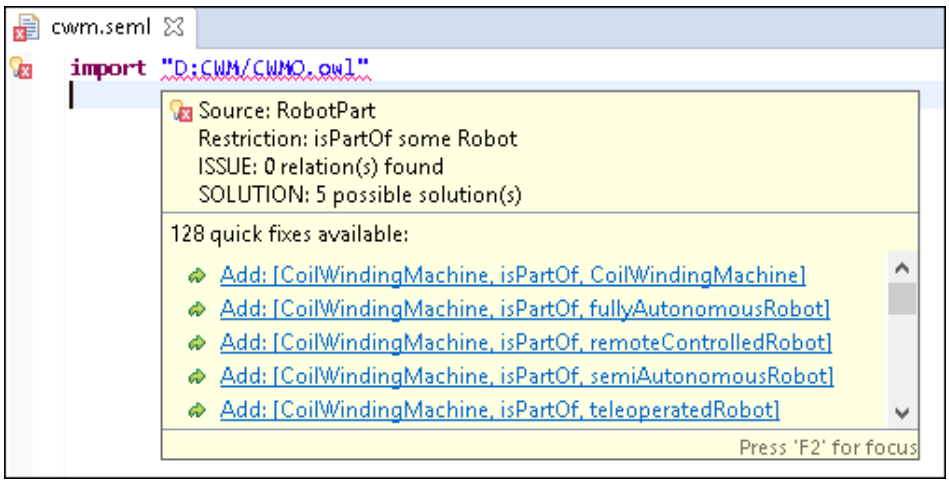


Figure 85.: SeML's compiler error that forces the system's designer to instantiate an intelligent motion control system.

In order to begin the modeling of the coil winding machine, the system's designer must import the application ontology that provides the domain-knowledge required, and since the prescriptive application ontology is an instance of the prescriptive domain ontology, which in turn is an instance of the prescriptive core ontology, the SeML imports all of the ontologies. Importing the prescriptive application ontology (CWMO.owl), as illustrated

in Listing 5.1, makes the compiler interpret it and extend the SeML's grammar with the knowledge provided by it, aiming at assisting the system's designer in the system's model development. The following figures illustrate the assistance provided by the SeML after each step of the system's model development process. After extending its grammar, the compiler shows an error, illustrated in Figure 85. This error forces the user to create an instance of an intelligent motion control system, which in turn can have one of the illustrated classifications. Since the prescriptive application ontology is for a coil winding machine, in this ontology the coil winding machine was already instantiated and classified as an automated robot in the ontology, using Protégé. Therefore, this informs the compiler about the compromise of creating this system, forcing the system's designer to use the instantiation created on Protégé in the model and specify its composition, as in the following Listing.

```
CoilWindingMachine isComposedOf
  CWM_PowerSupply,
  CWM_Infrastructure,
  EE-SX672,
  HBS2206,
  IS61LV25616,
  K9XXG08UXB,
  MT6070iH,
  STM32F103ZET6,
  SDIO,
  CycloneIV_EP4CE40F23C6,
  SecureDigitalCard

STM32F103ZET6 uses
  CycloneIV_EP4CE40F23C6,
  MT6070iH,
  SDIO,
  EE-SX672,
  HBS2206,
  IS61LV25616,
  K9XXG08UXB


CycloneIV_EP4CE40F23C6 uses
  EE-SX672,
  HBS2206,
  IS61LV25616,
  K9XXG08UXB

SDIO uses SecureDigitalCard
```

Listing 5.2: Description of the hardware structure of the coil winding machine using the SeML.

Subsequently, the compiler forces the designer to create the dependencies between the components, by establishing the relations between its processing parts, the STM32F103ZET6 and the CycloneIV_EP4CE40F23C6, and the remaining robot parts. These robot parts, *e.g.*

MT6070iH or EE-SX672, are related (through an object property) to processes. For example, the HBS2206 (a servo motor) is connect to the interpolation and the velocity control algorithms explored in the last subsections, forcing the user (during the specification of the system's behaviors) to mention which one of the existent algorithms (both to the interpolation and the velocity control algorithms) will control this actuating part in the system's implementation. Since this system contains a huge variety of alternatives for each process that is directly related with the motion control algorithms, in the prescriptive application ontology exists semantic knowledge **to indicate** the compiler **to ask** the designer **to specify** the system's requirements, because with them the compiler can assist the designer in the process of specifying the system's behaviours for each of its robot parts. This development is verified by the compiling error illustrated in Figure 86.

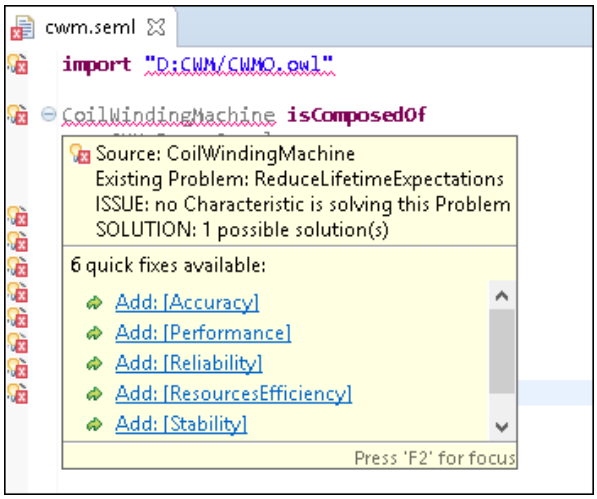


Figure 86.: SeML's compiler error that forces the system's designer to specify system's requirements.

So, the next step should be the specification of the system's requirements, elaborated in the following listing. As analysed in the last subsection, in the ontology each characteristic is associated with a problem and with the process that solves that problem. Therefore, by choosing the requirements of the system (or the characteristics), the system's designer is indirectly specifying the processes that should compose the implementation of the system.

```
use Accuracy, Reliability, Performance, Stability, ResourcesEfficiency
```

Listing 5.3: Specification of system's requirements, using the SeML.

Listing 5.3 represents the most important line in the system's model, since it allows the compiler to use the knowledge contained in the prescriptive application ontology to propose the remaining system's model composition. Hidden in this step, is the key for the automation of the system's configuration and implementation, since from it, the designer

will be restricted to define system's behaviours and properties that respect the requirements, and so, the generated code will respect them too.

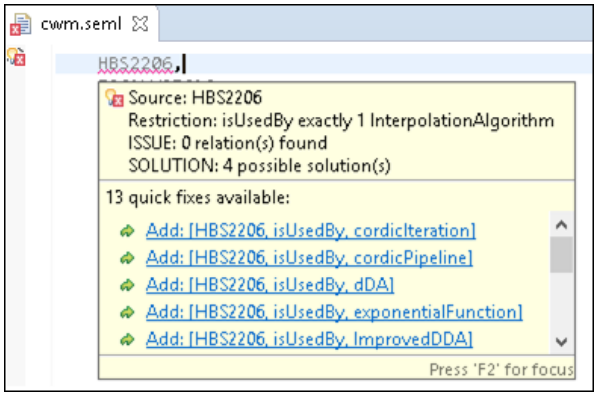


Figure 87.: SeML's compiler error that forces the system's designer to instantiate the application's software structure.

Following the proposes of the compiler, the system's designer specifies each process of the coil winding machine, as coded in the Listing 5.4. Since this model constitutes a reference architecture for the coil winding machine, the interface of each selected process must be verified. In order to simplify the programming effort and provide a *cleaner* model's view for the designer, the mechanisms that verify the interfaces of the reference architecture in the system's code work directly with the semantic knowledge of the ontology, avoiding redundancies in the model.

```
SecureDigitalCard isUsedBy
  DeleteDirectory,
  MakeDirectory,
  ReadDirectory,
  CloseDirectory,
  SeekFile,
  OpenDirectory,
  WriteFile,
  ReadFile,
  CloseFile,
  OpenFile

MT6070iH isUsedBy
  DisplayControlParameters,
  ProductSpecification

IS61LV25616 isUsedBy
  ReadSRAM,
  ReadSRAM_DMA,
  WriteSRAM,
  WriteSRAM_DMA

K9XXG08UXB isUsedBy
```

```
  FlashBlockErase,
  FlashReset,
  ReadSpareFlash,
  WriteFlashPage,
  WriteSpareFlash

HBS2206 isUsedBy ImprovedDDA
```

Listing 5.4: Description of the software structure of the coil winding machine using the SeML.

The last subsection explored the relation between system's requirements and its processes, which explicitly stated that the ImprovedDDA algorithm does not respects neither the reliability or the stability requirements, due to its enhanced chord error. Therefore, and since the system's designer chosen those requirements, by specifying this process (ImprovedDDA algorithm), he must expect the compiler to throw the error illustrated in the Figure 88.

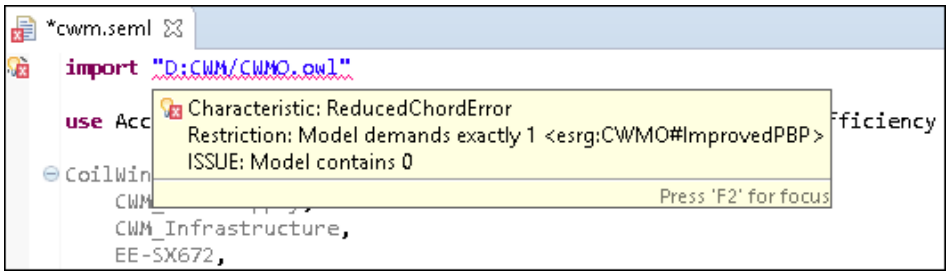


Figure 88.: SeML's compiler error that forces the system's designer to respect the specified system's requirements.

The referred interpolation algorithm should be replaced by an algorithm that represents the best trade-off for the requirements expressed, the ImprovedPBP.

```
HBS2206 isUsedBy ImprovedPBP
HBS2206 isUsedBy TrigonometricLUT
```

Listing 5.5: Description of the software structure of the coil winding machine using the SeML.

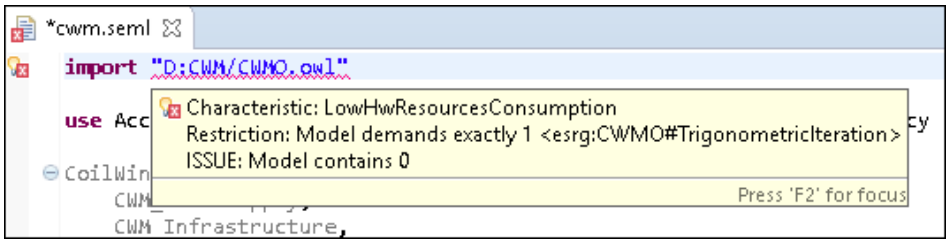


Figure 89.: SeML's compiler error that forces the system's designer to respect the specified system's requirements.

Concurrently, the TrigonometricLUT does not offers the best trade-off between the selected requirements, and so, the following error is thrown by the SeML compiler. In order to respect the specified requirements, the system's designer must select both the TrigonometricIteration and the ImprovedPBP, as expressed in the next listing.

```
HBS2206 isUsedBy TrigonometricIteration
HBS2206 isUsedBy ImprovedPBP
```

Listing 5.6: Description of the software structure of the coil winding machine using the SeML, accordingly with the specified requirements.

Lastly, the compiler throws an error expressing the necessity of having the specification of some properties, as illustrated in Figure 90.

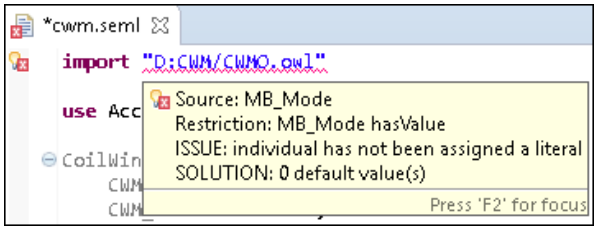


Figure 90.: SeML's compiler error that forces the system's designer to instantiate and specify the system's properties.

To conclude the system's model, the system's designer specifies the following properties, as required by the ontology (and thus by the SeML compiler).

```
FinalProduct = "Coil1"

MB_Mode = "MB_RTU"
MB_Parity = "EVEN"
MB_Baudrate = 9600
MB_Port = 1
MB_SlaveAddress = 8
MB_StopBitsWidth = 1
MB_WordWidth = 8

MaxAcceleration = 150
MaxVelocity = 150
Jerk = 15000
```

Listing 5.7: Description of system's properties using the SeML.

Since the model is finished, the system's designer can now ask the SeML to implement the system, and so, to generate the system's code with the specified structure, behaviours and properties. Conducive to this accomplishment, the SeML interprets the annotations

existent in the ontology for each one of the processes and properties, as explored in the last section, and generates the files' directory illustrated in Figure 91.

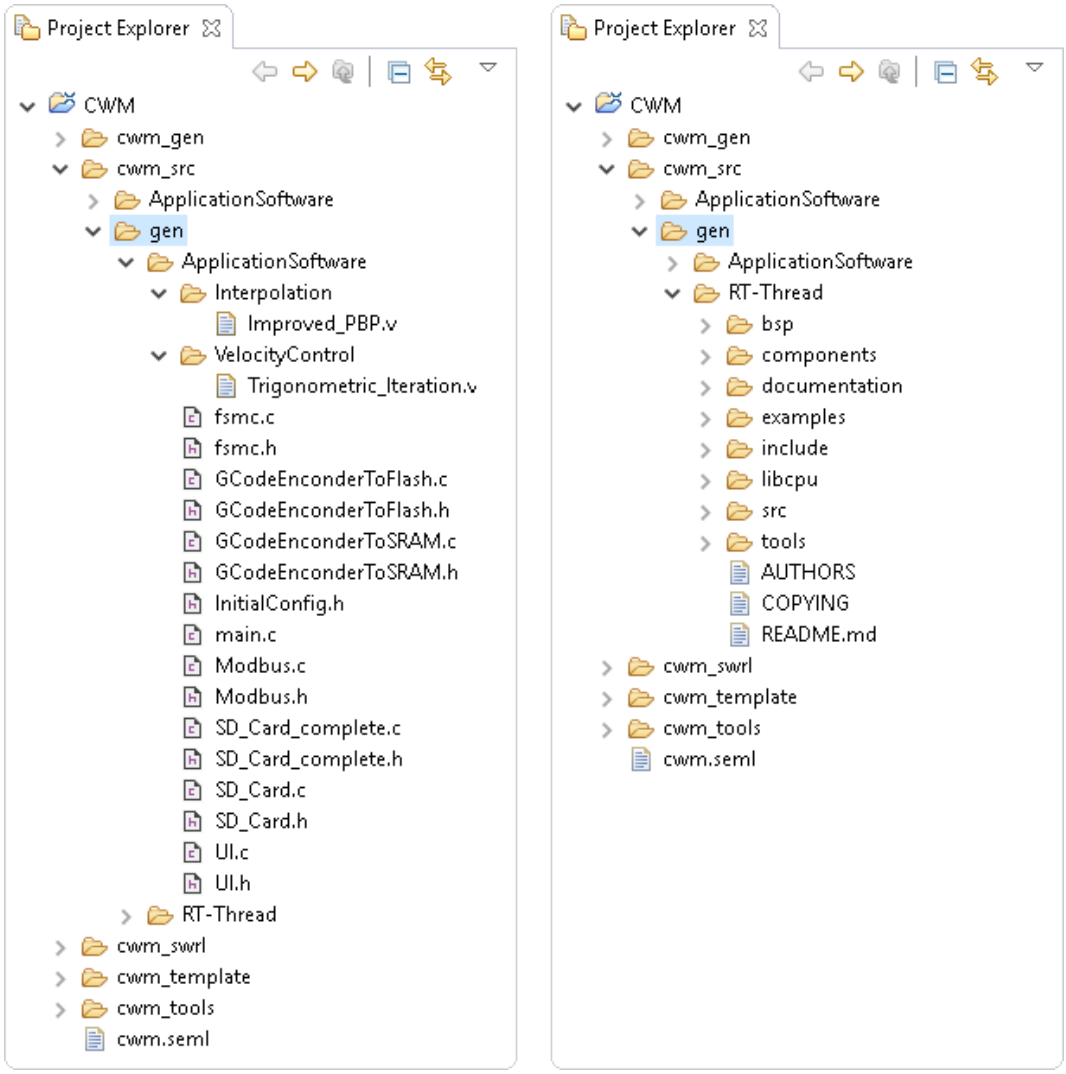


Figure 91.: Directory of files generated by the SeML.

Concurrently to the generation of the files' directory, the SeML verifies if the functions are in fact implemented inside the files, following the annotation provided by the prescriptive application ontology for each process. Accordingly to the annotation illustrated in Figure 82, the SeML generates the Trigonometric_Iteration.v file and checks if the function Trigonometric_Iteration is implemented, which in turn has the following implementation.

```
function Trigonometric_Iteration;
  input integer StartPoint;
  input integer EndPoint;
  input integer Distance;
```

```
  // Output driving wires to drive between 1-8 motors
  output wire servo1_control;
  output wire servo2_control;
  output wire servo3_control;
  // ...

endfunction
```

Listing 5.8: Trigonometric_Iteration implementation in Verilog, in a file named Trigonometric_Iteration.v that was generated by the SeML.

After this step, the SeML replaces the system's properties in the code with the values assigned by the system's designer during the system's model development. Following the examples of the figures 83 and 84, the SeML generates the InitalConfig.h file with the code presented in the listing 5.9.

```
#ifndef InitConfig
#define InitConfig

// Product Specification
#define FinalProduct Coil1

// Switch States
#define switch1   1
#define switch2   1
#define switch3   1
// ...

#endif
endfunction
```

Listing 5.9: InitalConfig.h file generated by the SeML, accordingly with the system's model.

By providing the knowledge (through concepts, relations, rules and annotations) about the system's hardware/software structure, the system's variability, the system's files directory, the files themselves and the processes and properties contained in that files, the prescriptive application ontology extends the SeML grammar and enables the automation of system's configuration and implementation. Thus, the integration with the SeML provides an abstract perspective of the system's implementation that allows the system's designer to model the system with domain-knowledge, reducing the complexity of system's configuration and implementation, and so, reducing its development cost.

6

DISCUSSION AND OUTLOOK

This chapter discusses the results of this Dissertation and concludes on the work developed. First, an initial evaluation of the semantically-enriched metamodeling process is provided, based on the insights gained from the case study in the last chapter. Then, the contributions of this Dissertation are explored, regarding both to the SeML and the IMCS projects. The Dissertation closes by pointing to the future work necessary in this project as well as providing some general research directions for ODSD.

6.1 EVALUATION

Based on the insights gained from the case study, this section provides an evaluation of the semantic technologies developed and used in this Masters Dissertation. A number of advantages that have proved to be of particular value for the semantically-enriched metamodeling process of software/hardware systems are highlighted, and lastly some critical issues and open questions that still need to be resolved to leverage the full potential of the followed approach are indicated.

6.1.1 *Advantages of the SeML*

Metamodeling environment
Since the SeML was implemented using the Xtext/Xtend Framework and Eclipse IDE, this DSML provides a rich modeling environment to develop a system's model. Through syntax highlighting, background parsing, error markers, content assist, hyperlinking and quick-fixes, the system's designer is able to use diverse features of the language to help in the development of the model. This modeling-assisted process is enabled both by the Xtext/Xtend framework and by the semantic knowledge that extends the language's grammar. Some examples of these features were illustrated during the creation of a system's model, in the previous chapter.

Composite design pattern

This design pattern describes a component as being atomic or composite. A composite component is composed of another components, while an atomic component is composed only by itself. The SeML prescriptive core ontology was created following this pattern, and, since the prescriptive domain ontology is an instance of this ontology, the knowledge described in the prescriptive domain ontology for intelligent motion control systems, follows a composite structure. This perspective of a software/hardware systems, creates a hierarchy of part-whole relations that reduces the complexity of a system's implementation analysis, and thus the complexity of creating a model. Besides that, since the SeML sees everything as a component (which in turn can be a property, an entity or a process), the system's model can be easily updated or upgraded by replacing or editing a specific component without changing the whole architecture. Subsequently, a system's model developed by a system's designer represents a reference architecture, and thus can be applied in others systems and changed its behaviour and properties without changing its core structure.

Goal-oriented software architecting

Alongside with the metamodeling environment and the composite design pattern, a goal-oriented architecture provides to the system's designer a straightforward way to almost automatically change the behaviour and properties of a system, by changing its requirements. Thus, as demonstrated in the previous chapter, this feature reduces the system's model complexity and development time.

6.1.2 *Advantages of Descriptive Domain Ontology for Intelligent Motion Control Systems*

Metamodeling with domain- and application-knowledge

Instantiated from the SeML prescriptive core ontology, the prescriptive domain ontology contains the concepts, relations and rules that constitute domain-knowledge about the intelligent motion control systems' domain. Being an instance of the latter ontology, the prescriptive application ontology for the coil winding machine provides specific concepts, relations and axioms that specify the inherited ones. Then, this structure of semantic knowledge extends the SeML's grammar and provides domain and application concepts, relations and axioms that enhance the metamodeling process through which the system's designer develops the system's model.

Excellent reuse support

Ontology technological space facilitates an effective reuse of classes (concepts) and properties declared in the prescriptive domain ontology [ABmo7], and so the integration of new prescriptive application ontologies can be effortlessly accomplished, increasing the knowl-

edge within the domain and allowing an agreement between different applications.

System's configuration and implementation automation
As validated in the previous chapter, system's configuration and implementation automation is enabled by the integration of the prescriptive application ontology. By doing so, the system's model development using SeML reduces system's development costs.

6.1.3 *Critical Issues*

Domain-knowledge coverage
As referred in the last chapter, the semantic refactor process required to convert a descriptive domain ontology to a prescriptive domain ontology reduces the knowledge within the ontology, and therefore, inhibits the system's model to contain more information about the hardware structure and the hardware properties of the system, which clearly is a disadvantage of the prescriptive domain ontology. Nevertheless, since the SeML is focused on enabling system's configuration and implementation automation through system's software variability description, this process represents an imperative convergence in the domain-knowledge's coverage to reduce the semantic-knowledge's overhead during the SeML's compiling processes.

SeML compiler's performance
The SeML compiling performance is directly influenced by the size of the ontology (in terms of concepts, relations, rules and annotations), and thus a really complex ontology besides being hardly reusable (as explained in Subsection 2.2.1), requires a higher reasoning and processing time from the reasoner and the SeML compiler, respectively. Though, Miguel Abreu (the SeML developer), developed and integrated SeML's mechanisms that enable a system's model development process to not be completely dependent on the ontologies' size. Conducive to this task, the referred mechanisms create a trade-off between the reasoning time (and compiling time) and the availability of some features of the rich metamodeling environment aforementioned.

6.2 CONTRIBUTE OF THIS DISSERTATION

The contributions of this Dissertation can be split into two different categories - contributions to the SeML's project, and contributions to the IMCSP's project.

6.2.1 *Contribute for the SeML's Project*

In Section 1.2, the context of the project developed in this Masters Dissertation was explored. Alongside with the ESRG-OT, this Master Dissertation contributes to the SeML by providing a prescriptive domain ontology that validates at the domain-level this technology and the work developed by this team, since it proofs that the SeML is capable of providing a technology to model a system based on semantic-knowledge in two different domains, the intelligent motion control systems' and the hypervisor's domain. Lastly, the validation at the application-level will be the objective of the ESRG-OT elements branched between the different sub-domains of the hypervisor's domain.

6.2.2 *Contribute for the IMCSP's Project*

At Jilin University, the IMCSP's project encompasses a hardware/software platform to control diverse intelligent motion control systems. The project lacked a technology that enabled the system's designer to automate the process of configuration and implementation of a specific system within the platform, and thus reduce the development time and the development costs. The ontologies developed in this Masters dissertation, alongside with its integration with the SeML, provide the tool required for the desired objectives, enabling the system's designer to create a system's model with domain-knowledge, and configure the system's implementation by expressing its requirements.

6.3 FUTURE WORK

Naturally, a single Masters Dissertation cannot exhaustively cover all questions and problem that arise from the huge and diverse complexity of this project. In fact, this Masters Dissertation has multiple directions for future work. Without a doubt, the most important goal in the near future is to be able to develop a system's model through a **graphical meta-modeling environment**, which in turn is a project already being developed by a member of the ESRG-OT. This integration aims at providing a user-friendly interface that can facilitate and enhance the system's model development.

In addition, and directly in focus with the objectives of this Masters Dissertation and both projects (the SeML and the IMCS), the second direction for future work is the development of **prescriptive application ontologies** for the remaining IMCSs developed for the IMCSP. This would allow us to create an agreement between different applications, and thus, enabling the existence of a vast catalogue of reference architectures for diverse intelligent motion control systems that can be used at any time. Lastly, this work would reinforce the validation of the SeML and the descriptive domain ontology at an application-level.

Finally, an interesting topic for future research is the expansion of the descriptive domain ontology for intelligent motion control systems to a descriptive domain ontology for **robotics and automation**, and thus the creation of an agreement between the actual ontology and other descriptive domain ontologies developed as instances of the SUMO ontology. This would enable the coverage of systems focused on speech recognition, object recognition, transport, etc. As a final objective, this ontology would be metamorphosed through a semantic refactor and integrated with the SeML, allowing it to model a wide range of systems in the robotics and automation domain, enabled by the merging of multiple perspectives provided by the diverse ontologies.

6.4 SUMMARY AND CONCLUSIONS

The goal of this Masters Dissertation was to develop a semantic technology that can be integrated with the SeML to enable the development of an intelligent motion control system's model and subsequently allow the automation of its customization and implementation. This goal was realized by tackling four distinct objectives in turn.

The first objective was to develop a technology to enable a DSML to model systems in any sub-domain of the software/hardware systems' domain. To do so, this Dissertation contributed to the development of a prescriptive core ontology based on the composite design pattern and the goal-oriented architecture, that can integrate other ontologies and extend the DSML's grammar to be able to model domain-specific applications. The result was the creation of the SeML.

The second objective was to developed an ontology to describe the domain-knowledge of intelligent motion control systems. To this end, the research of this Dissertation included a comprehensive literature survey to identify commonly used descriptive generic ontologies and descriptive domain ontologies for robotics and automation, studying their structures and comprehending as well their semantic rules and relations. An outcome of this study was the development of a descriptive domain ontology for intelligent motion control systems, as an instance of a well-structured and acknowledge descriptive generic ontology, SUMO.

The third objective was to *find an appropriated* ontology that could be integrated with the SeML to enable its *shapeshifting* capability and be able to model an intelligent motion control system. Therefore, through a semantic refactor process, the descriptive domain ontology was converted into a prescriptive domain ontology, containing only knowledge strictly necessary for the *shapeshifting* of the SeML.

Finally, the fourth objective was to evaluate the capability of the prescriptive domain ontology to provide the semantic knowledge to the SeML in order to model a domain-specific application. Therefore, an prescriptive application ontology was developed as an

instance of the prescriptive domain ontology, focused on a specific application - the coil winding machine. Subsequently, the model of the coil winding machine was created, and the automation of the system's configuration and implementation was validated by the code generated.

In conclusion, this Masters Dissertation has, to a large extent, achieved its goals. It clarified and explored many different aspects of ontologies' development, and clearly identified the most critical questions that need to be addressed in order to achieve its integration with a DSML, conducive to systems' configuration and implementation automation. As stated in Section 1.2, this Masters Dissertation represents a relevant work within both to the SeML and the IMCS projects. As a result, this work has become a broad coverage of many different issues related to the enrichment of DSML using semantic technology and ODSD in general. Whenever possible, this Dissertation strived to address the inherent complexity of this topic by providing frameworks to classify and evaluate both the ontologies, as a semantic technology, and the resultant DSML. Hopefully, this Masters Dissertation will represent a good foundation for future work and assist others Masters students to familiarize themselves with the backgrounds and state of the art of this project.

A

INTELLIGENT MOTION CONTROL SYSTEMS' DESCRIPTIVE DOMAIN ONTOLOGY

As a reference, this annex provides an overview over all the concepts and relations that were only referred during the analysis completed in the Subsection 5.2.1.

A.1 ROBOT PARTS' CLASSIFICATION

This section provides an overview of the classification of each part that can composes a robot. Conducive to the creation of a reduced taxonomy that would allow the classification of an actuating part, diverse approaches were suggested by the team of domain experts in the Jilin University. Nevertheless, in this ontology, an actuating part can be classified accordingly with its function:

- **Electric actuator** - An electric actuator is an actuating part which has electric current as energy source, and so it converts that energy into mechanical actions;
- **Hydraulic actuator** - A hydraulic actuator is an actuating part which has hydraulic Pressure as energy source, and so it converts that energy into mechanical actions;
- **Magnetic actuator** - A magnetic actuator is an actuating part which has magnetic energy as energy source, and so it converts that energy into mechanical actions;
- **Mechanical actuator** - A pneumatic actuator is an actuating part which has mechanical force as energy source, and so it converts that energy into mechanical actions;
- **Pneumatic actuator** - A pneumatic actuator is an actuating part which has pneumatic pressure as energy source, and so it converts that energy into mechanical actions;
- **Thermal actuator** - A thermal actuator is an actuating part which has thermal energy as energy source, and so it converts that energy into mechanical actions.

Similarly, a classification was created to classify a sensing part, accordingly with its functionality:

- **Acoustic sensor** - Specialized on measure, through a sensing process, some physical quantity in a acoustic environment;
- **Automotive sensor** - Specialized on measure, through a sensing process, some physical quantity in a automotive environment;
- **Chemical sensor** - Specialized on measure, through a sensing process, some physical quantity in a chemical environment;
- **Electrical sensor** - Specialized on measure, through a sensing process, some physical quantity that describes some electrical attribute;
- **Flow Sensor** - Specialized on measure, through a sensing process, some physical quantity in an environment evolving some fluid movement;
- **Force sensor** - Specialized on measure, through a sensing process, some physical quantity in an environment where the observed Force, density, or level is applied;
- **Ionization sensor** - Specialized on measure, through a sensing process, some physical quantity in an environment involving ionizing radiation or subatomic particles;
- **Movement sensor** - Specialized on measure, through a sensing process, some physical quantity involving position, angle, displacement, distance, speed or acceleration;
- **Navigation sensor** - Specialized on measure, through a sensing process, some physical quantity in a navigational environment;
- **Optical sensor** - Specialized on measure, through a sensing process, some physical quantity involving optical vision, light, imaging and photons;
- **Pressure sensor** - Specialized on measure, through a sensing process, some physical quantity in an environment where the observed pressure is applied;
- **Proximity sensor** - Specialized on measure, through a sensing process, some physical quantity in an environment where the observed proximity or presence needs to be controlled;
- **Thermal sensor** - Specialized on measure, through a sensing Process, some physical quantity in an Thermal environment.
- **Specific application sensor** - Specialized on measure, through a sensing process, some physical quantity in a specific environment, not specified by none of the referred sensing parts.

Accordingly with its portable characteristic, a classification was create for the powering part:

- **Battery** - Capable of providing power to a Robot (through a **supplying process**) and classified accordingly to its charging/discharging properties as:
  - Rechargeable - Battery which discharges through a discharging process and can be charged through a charging process multiple times, until the end of the battery lifetime (an operating property of a battery);
  - Non-rechargeable - Battery that discharges through a discharging process and can only be used one time, and so can't go through a charging process or a supplying process.
- **Power supply** - Capable of providing power to a robot (through a **supplying process**), or charging a **battery** (through a **charging process**).

A platform part can be classified into several categories:

- **UAV** - Unmanned aerial vehicle is a platform part for a robot that flies without a human pilot aboard and without requiring input from an operator (and so a fully autonomous robot);
- **AUV** - Autonomous underwater vehicle is a platform part for a robot that travels underwater without requiring input from an operator (and so, a fully autonomous robot);
- **UGV** - Unmanned ground vehicle is a platform part for a robot that operates while in contact with the ground, without a human pilot aboard and without requiring input from an operator (and so a fully autonomous robot);
- **Unnamed platform** - A platform part intended for any robot that is not included in the AUV, UAV and UGV subclasses.

Accordingly to its specificity, a processing part can be classified using the following taxonomy:

- **ASIC** - Application-Specific Integrated Circuit is a hardware processing part customized for a particular application or domain, rather than intended for general-purpose use (as is the case of the generic processor);
- **ASIP** - Application-Specific Instruction-Set Processor is a hardware processing part which includes a minimum ISA (Instruction Set Architecture) and a configurable logic that can be used to design new instructions. Compared with an **ASIC** is more flexible and more expensive. Compared with an FPGA is less flexible, achieves better performances and is cheaper;
- **Generic processor** - A generic processor is the hardware processing part dedicated to general purpose applications, such as personal computers and workstations;

- **FPGA** - Field-Programmable Gate Array is an hardware processing part that is designed to be configured and reprogrammed to desired application or functionality requirements after manufacturing.

Lastly, a storing part can be classified as illustrated in Figure 92.

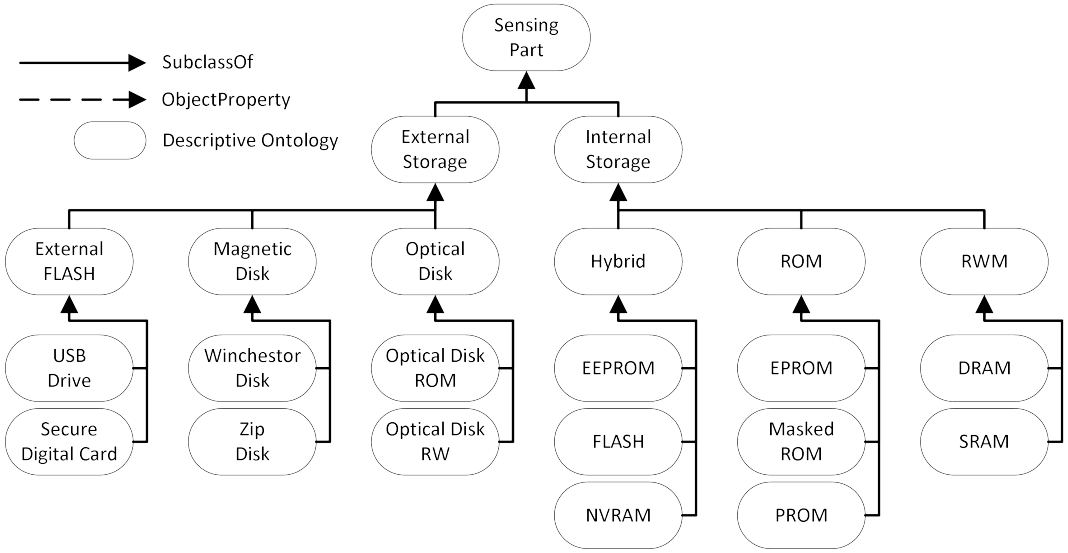


Figure 92.: Part of the descriptive domain ontology showing taxonomy that allows the classification of a Storing Part.

- **External storage** - A storing part that is not directly accessible by a software processing part. This kind of storing part provides a way of communicate indirectly with the Robot.
  - External flash memory - A non volatile and randomly accessed external storage;
  - Magnetic disk - A non volatile and sequentially accessed external storage. A magnetic disk can be accessed through a read/write, using a electromagnet reader;
  - Optical disk - A non volatile and sequentially accessed external storage. A optical disk can be accessed through a read/write, using a laser reader.
- **Internal storage** - A storing part that is directly accessible by a software processing part, through a storing process.
  - RWM - A volatile, mutable and randomly accessed internal storage;
  - ROM - A non volatile, non mutable and randomly accessed internal storage;
  - Hybrid - A non volatile, mutable and randomly accessed internal storage.

A.2 PROPERTIES

This section illustrates the conceptualization of property in the descriptive domain ontology, exploring the properties of each robot part. As described in the Subsection 5.2.1, both robot part and robot have survival and operating ranges, which attributes depend directly on the environmental conditions experimented by both. Thus, a robot (as a composite system) has a survival range and an operating range that is an aggregation of the survival ranges and operating ranges of the atomic or composite robot part that compose it.

The following definitions describe the main conceptualizations that allow the specification of robot part/robot's hardware properties, while the software properties are defined in the application ontology (due to its specific relation with the system's code):

- Capability - A capability is a property that collects together properties and environmental conditions in which those properties hold, composing a specification of a robot part's capability in those conditions. Therefore, its branched in:
  - Measuring capability;
  - Powering capability;
  - Supporting capability;
  - Actuating capability;
  - Processing capability;
  - Storing capability;
  - Powering capability;
- Hardware property - Physical describes a robot part;
- Operating range - An operating range is a hardware property that specifies the environmental conditions and characteristics of a robot part/robot's normal operating environment;
- Operating property - A hardware property that identifies the environmental characteristics and other conditions in which the robot part/robot is intended to operated;
- Survival range - A hardware property that specify the conditions a robot part/robot's can be exposed to without damage. If the robot part/robot's is damaged, its capabilities' specifications may no longer be the same;
- Survival property - A hardware property intended to identify the characteristics that represent the extent of the robot part/robot's useful life;
- Condition - A hardware property which specify ranges for other hardware property, defining the behaviour of a robot part/robot in the surrounding environment.

Equivalently to the operating range and survival range, the majority of the properties presented in the following figures are described through the existent relation with the conceptualizations of Physical Quantity Kind and Physical Unit of Measure, allowing each property to have a meaning and a unit in the *Systeme International Unit*. Nevertheless, some properties require a functional relation and a covering axiom to allow its description, as explained during the implementation of the prescriptive domain ontology, in the Subsection 5.2.2. Examples of this *special* properties are the Mutability and Volatility, both illustrated on Figure 93.

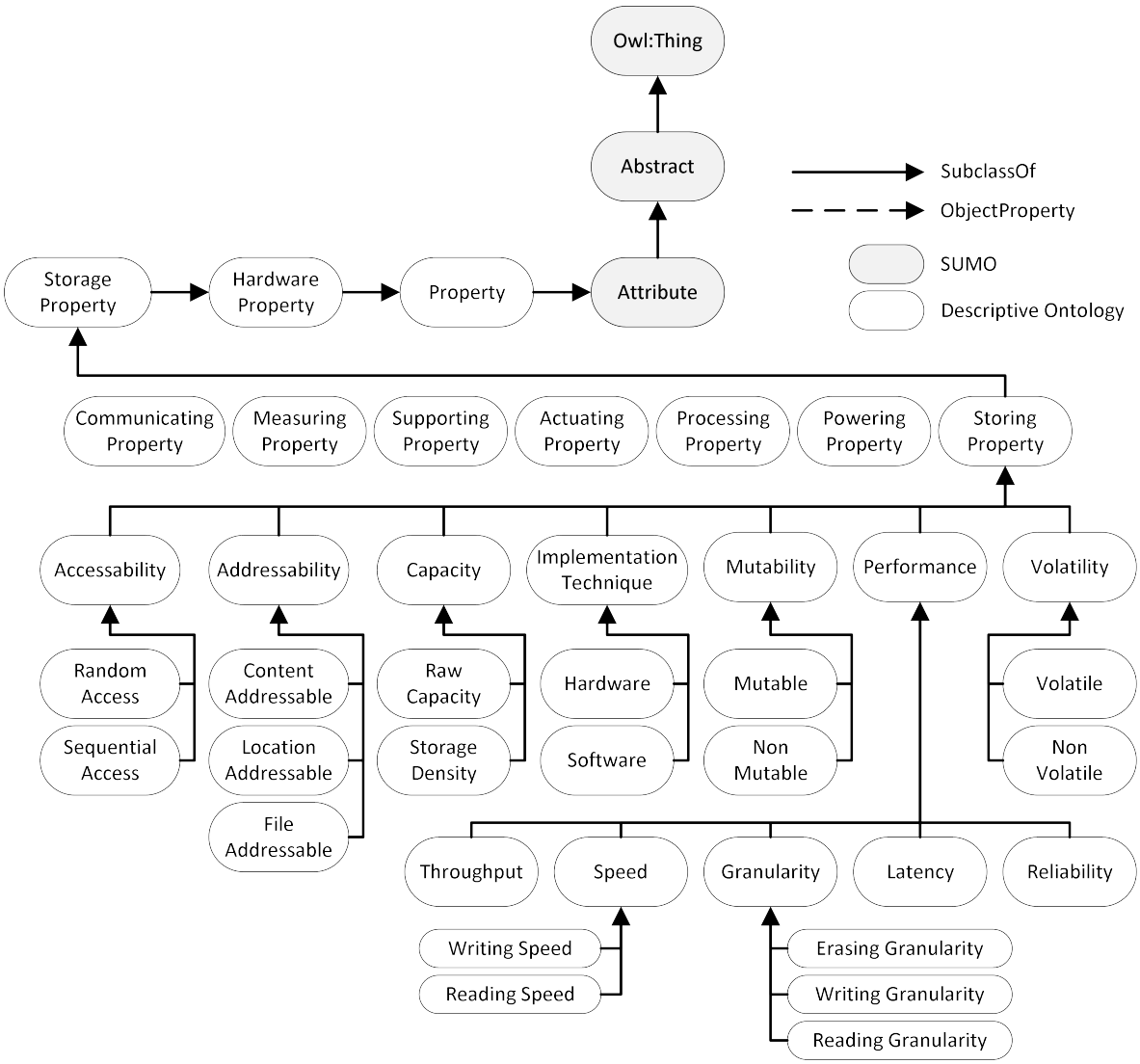


Figure 93.: Part of the descriptive domain ontology showing the main concepts related to Storing Property concept.

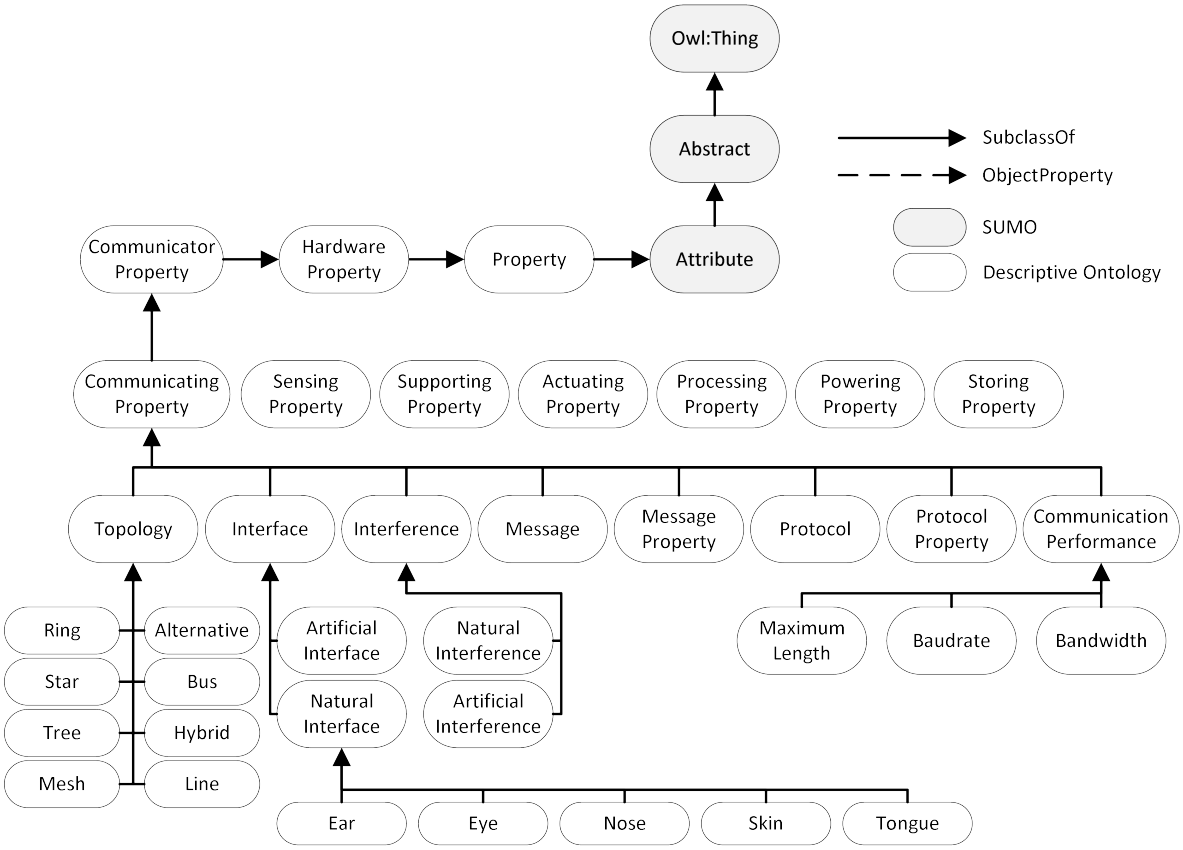


Figure 94.: Part of the descriptive domain ontology showing the main concepts related to Communicating Property concept.

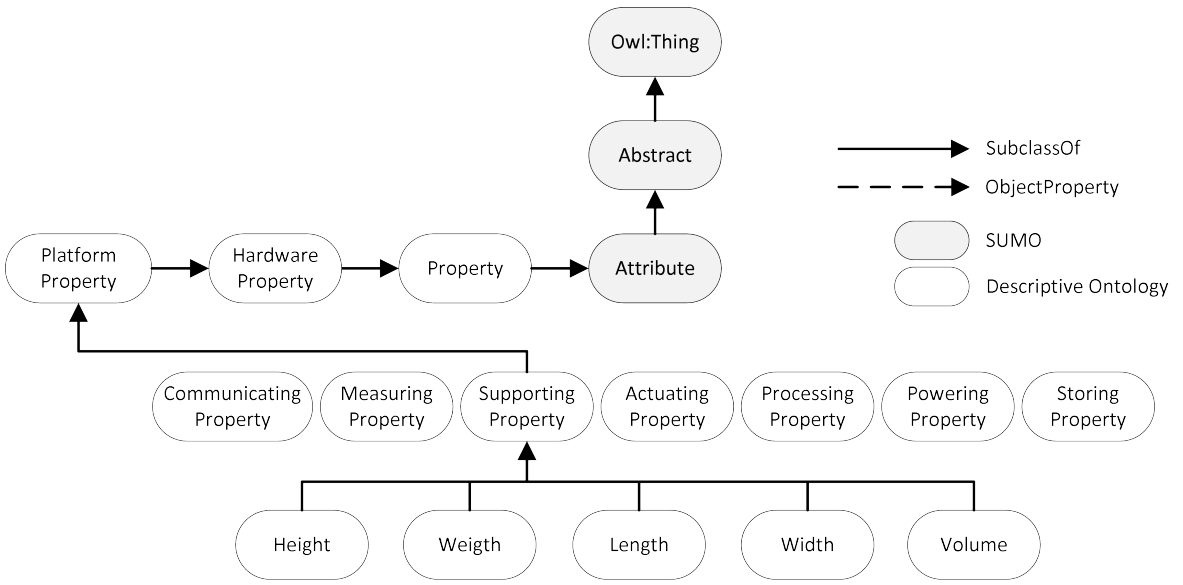


Figure 95.: Part of the descriptive domain ontology showing the main concepts related to Supporting Property concept.

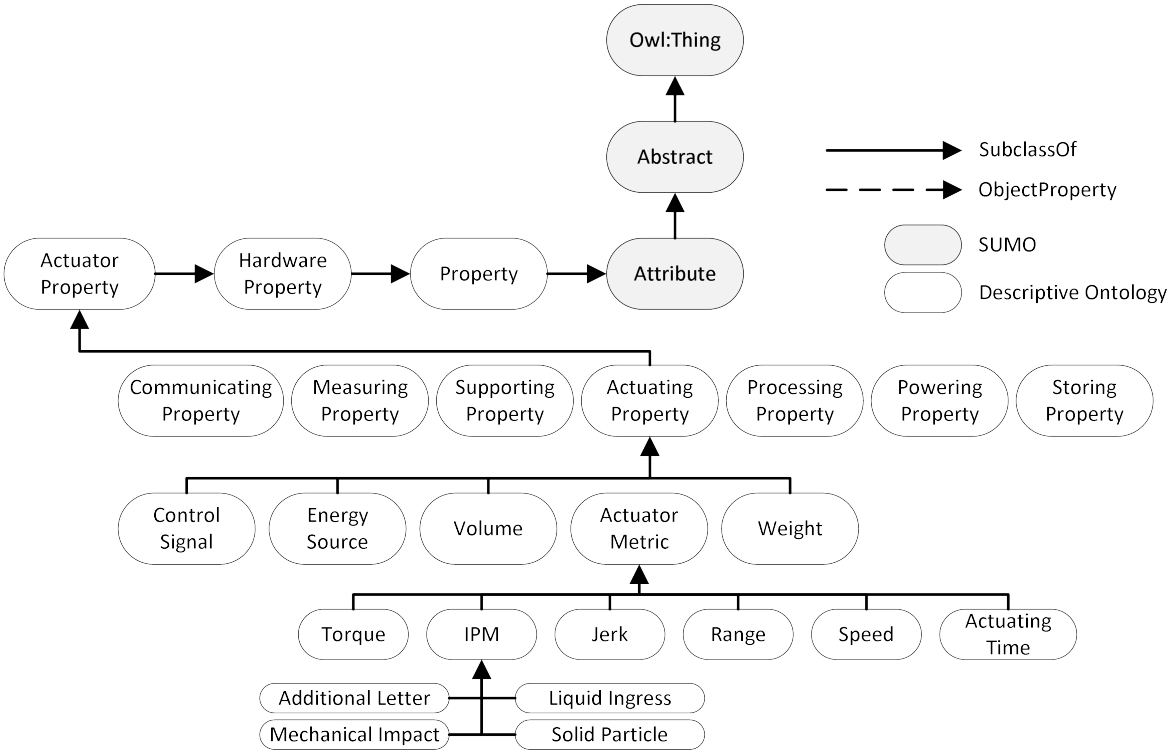


Figure 96.: Part of the descriptive domain ontology showing the main concepts related to Actuating Property concept.

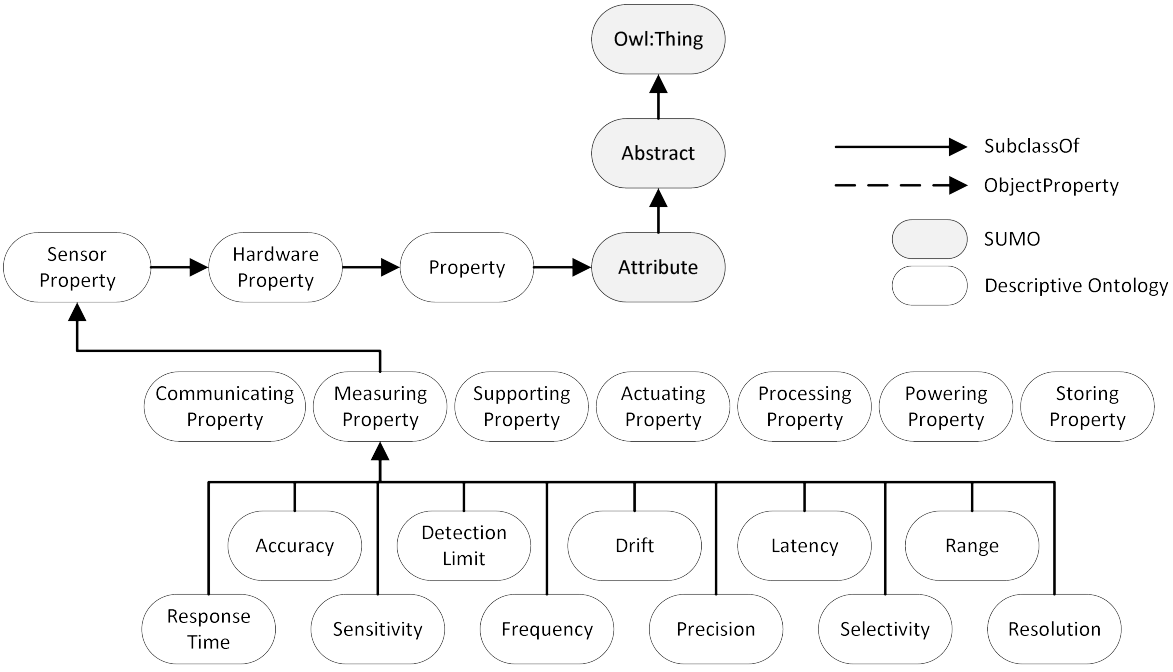


Figure 97.: Part of the descriptive domain ontology showing the main concepts related to Measuring Property concept.

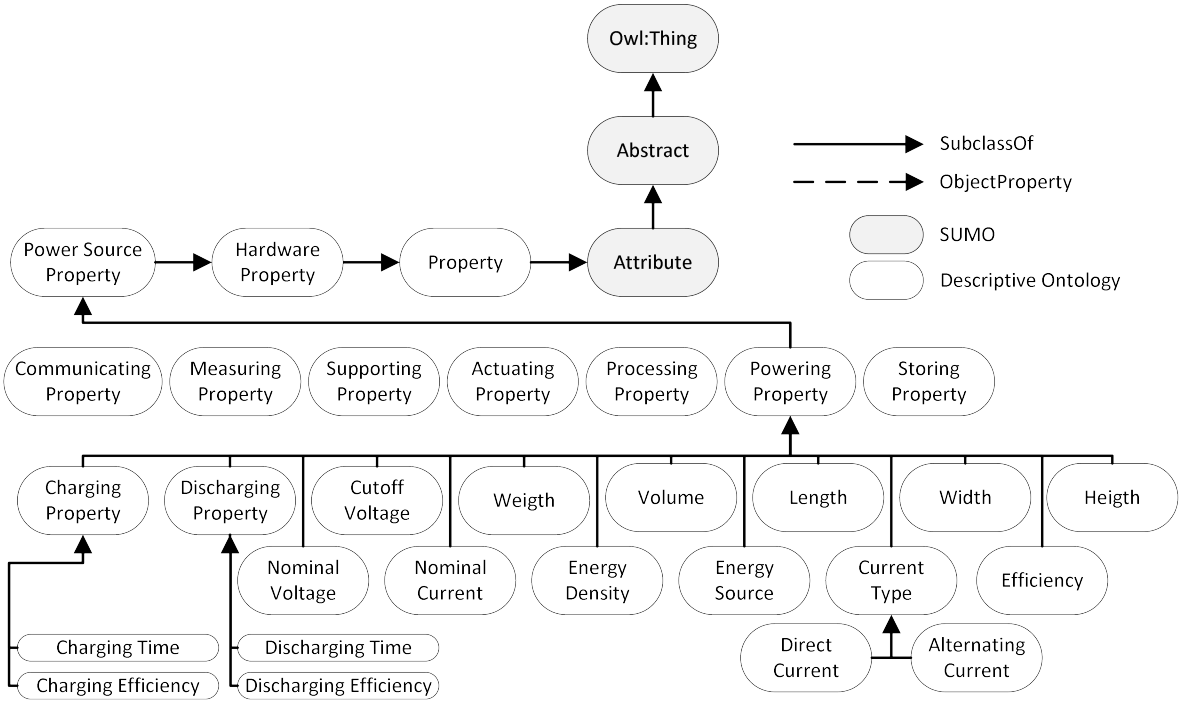


Figure 98.: Part of the descriptive domain ontology showing the main concepts related to Powering Property concept.

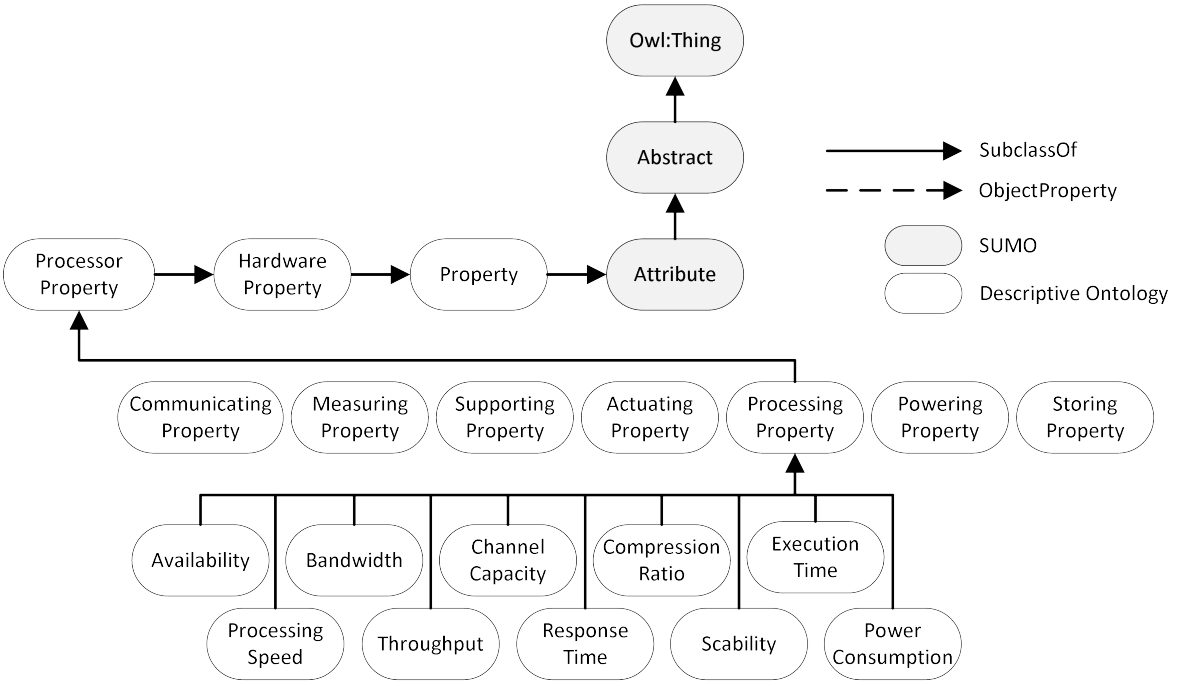


Figure 99.: Part of the descriptive domain ontology showing the main concepts related to Processing Property concept.

A.3 PROCESSES

This section provides an overview of the taxonomy of processes that a robot can execute through a specific robot part, illustrated in Figure 100:

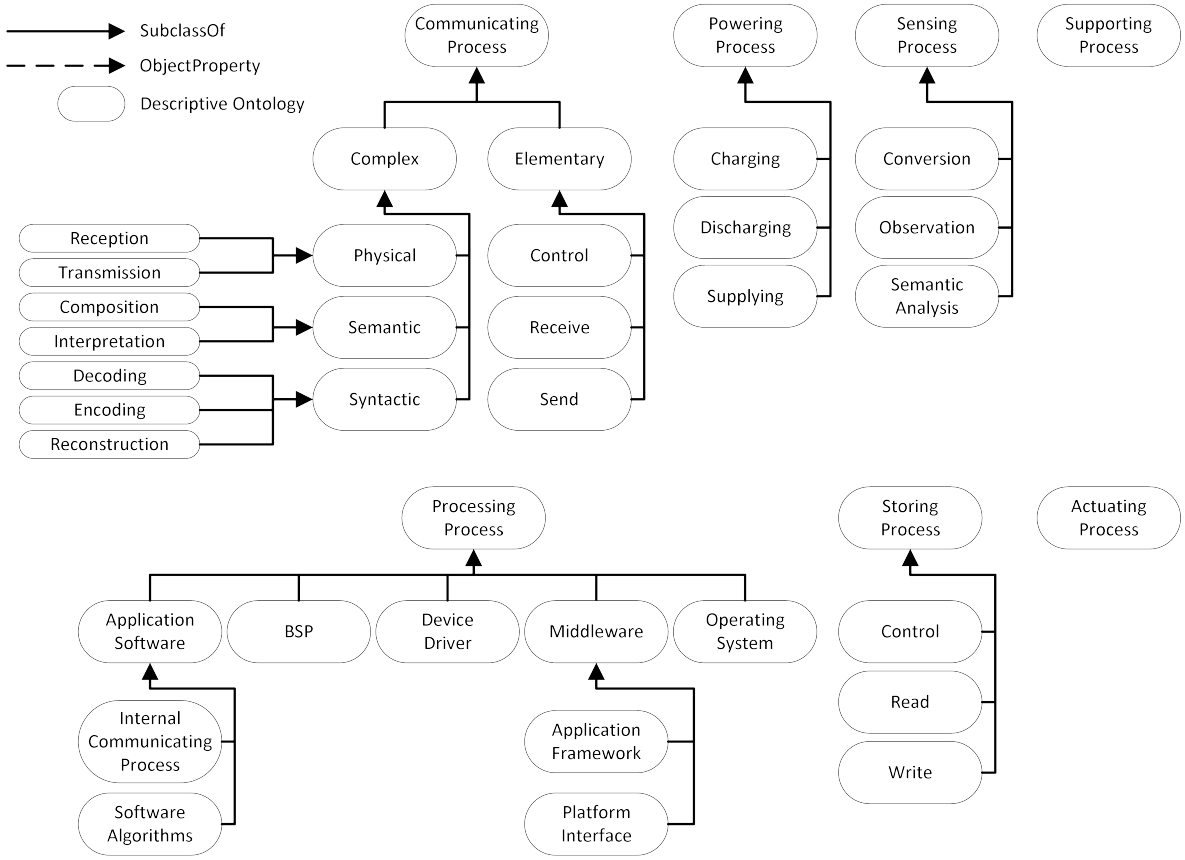


Figure 100.: Part of the descriptive domain ontology showing the main concepts related to Process conceptualization, depending on the Robot Part.

- Communicating process - A process that interacts with a communicating part, in order to provide interaction among robots and humans:
  - Complex:
    - Physical - Responsible for configuring the system and proceed with the required actions to enable the reception and transmission of messages;
    - Syntactic - Allows a robot to decode (decoding), encode (encoding) or reconstruct (reconstruction) a message, in order to assemble it in a syntactically correct way.
    - Semantic - Responsible for interpreting (interpretation) [or infer (composition)] the semantic knowledge of a (to a) message, making it meaningful.

  - Elementary:
    * Control - Procedure to interact with a communicating part in order to configure or control its behaviour;
    * Receive - Procedure to wait for a message;
    * Send - Procedure to send a message.

- Powering process - Process which allows a powering part to be the robot's power supply or to be the power supply of another powering part:
  - Charging - Represents the action of charging another powering part;
  - Discharging - Represents the action of discharging of a powering part;
  - Supplying - Represents the action of being the power supply of a robot part/robot.

- Sensing process - A process which is aware of an observation to allow the robot to measure a physical quantity in the surrounding environment, through a sensing part:
  - Conversion - Responsible for converting the measured physical quantity into a signal, in order to allow the measured values to be processed;
  - Observation - Observes a predefined physical quantity in the surrounding environment or in the robot itself;
  - Semantic Analysis - Infers meaning to the measured physical quantity.

- Processing process - Process responsible for processing the data and information necessary to the proper and required operation of the robot:
  - Application Software - Process focused on the application software:
    * Internal communication;
    * Software algorithms.
  - Board support package - A process dedicated to provide support for a given board that conforms to a given operating system;
  - Device driver - Process focused on the **device drivers**, and thus responsible for implementing a device driver;
  - Middleware - Process focused on the middleware's software:
    * Application framework - Defines a standard structure for application software development;
    * Platform interface - Defines an API that enables the application software to interact with the operating system, and so provides services to the application software;

  - Operating system - Process that constitutes the operating system and manages its operation.

- Storing Process - A process responsible for enabling the interaction between the robot and its storing part.
  - Control - Interaction with a storing part that is neither read and write;
  - Read;
  - Write;
- Supporting process;
- Actuating process - Process that interact with the actuating part, in order to move and act in the surrounding environment.

B

COIL WINDING MACHINE'S MODEL

```
import "D:CWM/CWMO.owl"

/* System's Requirements [Goals] */
use Accuracy, Reliability, Performance, Stability, ResourcesEfficiency

/* Hardware Structure [RobotParts] */
CoilWindingMachine isComposedOf
  CWM_PowerSupply,
  CWM_Infrastructure,
  EE-SX672,
  HBS2206,
  IS61LV25616,
  K9XXG08UXB,
  MT6070iH,
  STM32F103ZET6,
  SDIO,
  CycloneIV_EP4CE40F23C6,
  SecureDigitalCard

STM32F103ZET6 uses
  CycloneIV_EP4CE40F23C6,
  MT6070iH,
  SDIO,
  EE-SX672,
  HBS2206,
  IS61LV25616,
  K9XXG08UXB

CycloneIV_EP4CE40F23C6 uses
  EE-SX672,
  HBS2206,
  IS61LV25616,
  K9XXG08UXB

SDIO uses SecureDigitalCard

/* Software Structure [Processes] */
SecureDigitalCard isUsedBy
  DeleteDirectory,
  MakeDirectory,
```

```
  ReadDirectory,
  CloseDirectory,
  SeekFile,
  OpenDirectory,
  WriteFile,
  ReadFile,
  CloseFile,
  OpenFile


MT6070iH isUsedBy
  DisplayControlParameters,
  ProductSpecification


IS61LV25616 isUsedBy
  ReadSRAM,
  ReadSRAM_DMA,
  WriteSRAM,
  WriteSRAM_DMA

K9XXG08UXB isUsedBy
  FlashBlockErase,
  FlashReset,
  ReadSpareFlash,
  WriteFlashPage,
  WriteSpareFlash

HBS2206 isUsedBy TrigonometricIteration
HBS2206 isUsedBy ImprovedPBP


/* Properties */
FinalProduct = "Coil1"
MB_Mode = "MB_RTU"
MB_Parity = "EVEN"
MB_Baudrate = 9600
MB_Port = 1
MB_SlaveAddress = 8
MB_StopBitsWidth = 1
MB_WordWidth = 8
MaxAcceleration = 150
MaxVelocity = 150
Jerk = 15000
```

Listing B.1: Application's model using the SeML.

C

ONTOLOGIES' DEVELOPMENT - GANTT DIAGRAM

As a reference, this annex provides an overview of the main tasks (Figure 101) and respective schedule (Figure 102) in the development of the descriptive domain ontology, prescriptive domain ontology and the prescriptive application ontology. The scheduling of this ontologies' development follows the **METHONTOLOGY** methodology and includes the validation of each of the ontologies by the 10 commandments and the Ontoclean methodologies.

Task	Begin	End
Specification	**01/09/16**	01/12/16
Descriptive domain ontology's specification	01/09/16	01/10/16
Prescriptive domain ontology's specification	01/10/16	01/11/16
Prescriptive application ontology's specification	01/11/16	01/12/16
Acquiring Knowledge	**01/09/16**	**19/06/17**
Coil winding machine's analysis	28/02/17	19/06/17
IMCSProject's analysis	01/10/16	29/05/17
Descriptive core & generic ontologies' analysis	**01/09/16**	**03/12/16**
Descriptive domain ontologies' analysis	**01/09/16**	**03/12/16**
Conceptualization	**12/11/16**	05/01/17
Formalization	**26/11/16**	**01/05/17**
Validation using 10 Ccommandments	**26/11/16**	**01/05/17**
Validation using Ontoclean	**26/11/16**	**01/05/17**
Integration	**03/12/16**	**01/01/17**
Descriptive domain ontologies' integration with SUMO	**03/12/16**	**01/01/17**
Implementation	**01/01/17**	**01/05/17**
Descriptive domain ontologies' development	01/01/17	01/03/17
Prescriptive domain ontologies' development	02/03/17	**01/05/17**
Evaluation	**01/05/17**	**19/06/17**
Prescriptive application ontology's development	01/05/17	29/05/17
Modeling the coil winding machine	29/05/17	19/06/17
Documentation	**01/09/16**	31/08/17

Figure 101.: Ontologies' development tasks, accordingly with the methodologies explored in Subsection 2.2.1.

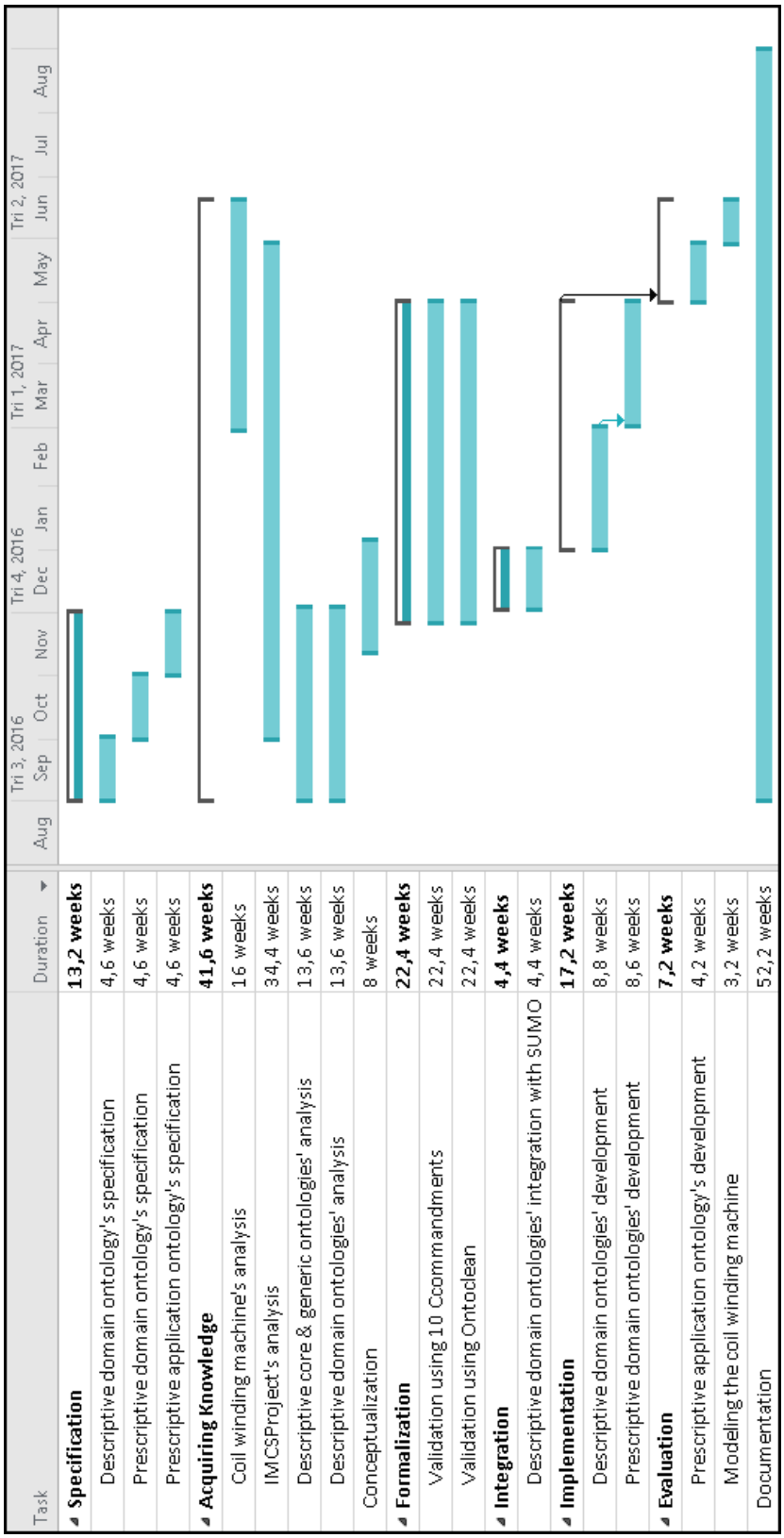


Figure 102.: Gantt's diagram illustrating ontologies' development scheduling, respecting the methodologies explored in Subsection 2.2.1.

BIBLIOGRAPHY

[045] RoSta Project 045304. RoSta: Project. http://www.robot-standards.org/index.php?id=8. [Online; visited on 14-April-2017].

[ABM99] Colin Atkinson, Jean Bézivin, and Pierre-Alain Muller. *Supporting and Applying the UML Conceptual Framework*, pages 21–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[ABm07] Uwe ABmann. *Design of a Semantic Connector Model for Composition of Metamodels in the Context of Software Variability*. Phd thesis, Technische Universitat Dresden, 2007.

[Ack88] J.L. Ackrill. *A New Aristotle Reader*. Princeton University Press, 1988.

[AKG11a] Colin Atkinson, Bastian Kennel, and Bjorn GoB. *The Level-Agnostic Modeling Language*, pages 266–275. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[AKG11b] Colin Atkinson, Bastian Kennel, and B Goß. Supporting Constructive and Exploratory Modes of Modeling in Multi-Level Ontologies. *7th International Workshop on Semantic Web Enabled Software Engineering*, 2011.

[ALM88] J. S. Albus, R. Lumia, and H. Mccain. Hierarchical control of intelligent machines applied to space station telerobots. *IEEE Transactions on Aerospace and Electronic Systems*, 24(5):535–541, 1988.

[Alt] Altova. UML Round Trip Engineering. http://www.altova.com/umodel/uml-round-trip.html. [Online; visited on 27-March-2017].

[Ass] European Health Telematics Association. ITB CNR Institute for Biomedical Technologies (Italy) eHealth Portal for Europe. https://www.ehtel.eu/join-ehtel/member-profiles/member-itb-cnr. [Online; visited on 24-April-2017].

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[BAM+04] T. Barbera, J. Albus, E. Messina, C. Schlenoff, and J. Horst. How task analysis can be used to derive and organize the knowledge for the control of autonomous vehicles. *Robotics and Autonomous Systems*, 49(1-2 SPEC. ISS.):67–78, 2004.

[BASRH13] Julita Bermejo-Alonso, Ricardo Sanz, Manuel Rodríguez, and Carlos Hernández. *Ontology Engineering for the Autonomous Systems Domain*, pages 263–277. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[BDZ89] R. D. Banker, S. M. Datar, and D. Zweig. Software complexity and maintainability. *Proceedings of the tenth international conference on Information Systems - ICIS '89*, 11(5.6):247–255, 1989.

[Bet04] J Bettin. Model-Driven Software Development Activities. *The Process View of an MDSD Project, SoftMetaWare*, pages 1–16, 2004.

[Bet16] Lorenzo Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd, 2 edition, 2016.

[BGM96] S Borgo, N Guarino, and C Masolo. A Pointless Theory of Space Based on Strong Connection and Congruence. In *Proceedings of Principles of Knowledge Representation and Reasoning,*, pages 220–229. Morgan Kaufmann, 1996.

[BGM97] Stefano Borgo, Nicola Guarino, and Claudio Masolo. An Ontological Theory of Physical Objects. *Qualitative Reasoning 11th International Workshop*, pages 223–231, 1997.

[BHJV08] Jürgen Bock, Peter Haase, Qiu Ji, and Raphael Volz. Benchmarking OWL reasoners. *CEUR Workshop Proceedings*, 350, 2008.

[BKMPS09] Jie Bao, Elisa F. Kendall, Deborah L. McGuinness, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language Quick Reference Guide. http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/, 2009. [Online; visited on 04-May-2017].

[BL00] Tim Berners-Lee. Semantic Web - XML2000. http://www.w3.org/2000/Talks/1206-xml2k-tbl, 2000. [Online; visited on 04-April-2017].

[BL07] Matthias Bräuer and Henrik Lochmann. Towards semantic integration of multiple domain-specific languages using ontological foundations. *Proceedings of 4th International Workshop on (Software) Language Engineering (ATEM 2007) co-located with MoDELS*, 2007.

[BPSM+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. {Extensible Markup Language}. https://www.w3.org/XML/, 2008. [Online; visited on 25-July-2017].

[Bro75] A. P. G. Brown. Modelling a real world system and designing a schema to represent it. In *IFIP TC-2 Special Working Conference on Data Base Description*, pages 339–348, 1975.

[Bro86] Rodney A. Brooks. A Robot Layered Control System For a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[BS04] Stephen Balakirsky and Chris Scrapper. Knowledge representation and planning for on-road driving. *Robotics and Autonomous Systems*, 49(1):57 – 66, 2004. Knowledge Engineering and Ontologies for Autonomous Systems 2004 AAAI Spring Symposium.

[Bun77] Mario Augusto Bunge. *Treatise on Basic Philosophy - Onthology I: The Furniture of the World*, volume 3. Springer Netherlands, 1977.

[CBB+12] Michael Compton, Payam Barnaghi, Luis Bermudez, et al. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32, 2012.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[CFP+13] Joel Luis Carbonera, Sandro Rama Fiorini, Edson Prestes, et al. Defining positioning in a core ontology for robotics. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1867–1872, 2013.

[CG02] Feliciano Manzano Casas and L. A. García. *OCOA: An Open, Modular, Ontology Based Autonomous Robotic Agent Architecture*, pages 173–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Che76] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, March 1976.

[CJ99] B. Chandrasekaran and R. Josephson. What are ontologies, and Why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[CP13] Clark and Parsia. Pellet's reasoner, 2013. [Online as a Github repository; available on 25-July-2017].

[CRP06] C. Calero, F. Ruiz, and M. Piattini. *Ontologies for Software Engineering and Software Technology*. Springer Berlin Heidelberg, 2006.

[CT+04] Nicholas L. Cassimatis, J. Gregory Trafton, et al. Integrating cognition, perception and action through mental simulation in robots. *Robotics and Autonomous Systems*, 49(1-2 SPEC. ISS.):13–23, 2004.

[CWL14] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts, 2014. [Online; visited on 25-July-2017].

[DB11] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice-Hall, Inc., 12th edition, 2011.

[dBFK+12] Jos de Bruijn, Dieter Fensel, Uwe Keller, et al. WSML - Web Service Modeling Language. https://www.w3.org/Submission/WSML, 2012. [Online; visited on 01-April-2017].

[DCTD11] Kathrin Dentler, Ronald Cornet, Annette Ten Teije, and Nicolette De Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web*, 2(2):71–87, 2011.

[DDLF+11] Saadia Dhouib, Nicolas Du Lac, Jean-Loup Farges, et al. Control Architecture Concepts and Properties of an Ontology Devoted to Exchanges in Mobile Robotics. In *6th National Conference on Control Architectures of Robots*, page 24 p., Grenoble, France, May 2011. INRIA Grenoble Rhône-Alpes.

[DDV13] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Archite and Ontology Development*, volume 53. Springer, 2013.

[Dev02] Vladan Devedzić. Understanding ontological engineering. *Commun. ACM*, 45(4):136–144, April 2002.

[DHHS01] Wolfgang Degen, Barbara Heller, Heinrich Herre, and Barry Smith. GOL: Toward an axiomatized Upper-Level Ontology. *Proceedings of the international conference on Formal Ontology in Information Systems - FOIS '01*, 2001(November):34–46, 2001.

[Dic17] Dictionary.com. Definition of the domain term, 2017. [Online; visited on 25-July-2017].

[DRM12] Google Dan Brickley, Google R.V. Guha, and Brian McBride. RDF Schema 1.1. https://www.w3.org/TR/rdf-schema/, 2012. [Online; visited on 01-April-2017].

[DSB+04] M Dean, G Schreiber, S Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, D McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein.

OWL Web Ontology Language Reference, 2004. [Online; visited on 23-June-2017].

[Eps04] Sl Epstein. Metaknowledge for Autonomous Systems. *Proceedings of AAAI Spring Symposium on Knowledge Representation and Ontology for Autonomous Systems. AAAI.*, 2004.

[ES16] Sven Efftinge and Miro Spoenemann. Xtext - Language Engineering Made Easy! http://www.eclipse.org/Xtext/, 2016. [Online; visited on 02-July-2017].

[Fal10] Sean Falconer. OntoGraf - Protege Wiki. http://protegewiki.stanford.edu/wiki/OntoGraf, 2010. [Online; visited on 13-June-2017].

[Far09] Jean-Loup Farges. Robotic Ontology and Modelling - 3rd version, 2009. [Online - PDF version; visited on 10-July-2017].

[FB99] Martin Fowler and Kent Beck. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[fBI13] National Cancer Institute Center for Biomedical Informatics. National Cancer Institute Thesaurus. http://bioportal.bioontology.org/ontologies/NCIT, 2013. [Online; visited on 04-June-2017].

[FCG+15] Sandro Rama Fiorini, Joel Luis Carbonera, Paulo Goncalves, et al. Extensions to the core ontology for robotics and automation. *Robotics and Computer-Integrated Manufacturing*, 33:3–11, 2015.

[FGL87] K S Fu, R C Gonzalez, and C S George Lee. *ROBOTICS : Control, Sensing, Vision, and Intelligence*. McGraw-Hill Education (India) Pvt Limited, 1987.

[FLGPJ97] M Fernández-López, A Gómez-Pérez, and Natalia Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. *AAAI-97 Spring Symposium Series*, SS-97-06:33–40, 1997.

[FM01] Dieter Fensel and Mark A. Musen. The semantic web: A brain for humankind. *IEEE Intelligent Systems and Their Applications*, 16(2):24, 2001.

[FRBS+13] Ulrich Frank, Iris Reinhartz-Berger, Arnon Sturm, et al. *Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines*, pages 133–157. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[fRG15] Ontologies for Robotics and Automation Working Group. Ieee standard ontologies for robotics and automation. *IEEE Std 1872-2015*, pages 1–60, April 2015.

[GG95] Nicola Guarino and Pierdaniele Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In *Towards very Large Knowledge bases: Knowledge Building and Knowledge sharing*, pages 25–32. IOS Press, 1995.

[GG05] G. Guizzardi and Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, 10 2005.

[GHH+] Rafael Gonçalves, Josef Hardi, Matthew Horridge, et al. protégé. http://protege.stanford.edu/. [Online; visited on 06-May-2017].

[GHW02a] Giancarlo Guizzardi, Heinrich Herre, and Gerd Wagner. On the general ontological foundations of conceptual modeling. *Conceptual Modeling ER 2002, Lecture Notes in Computer Science*, 2503(Er 2002):65–78, 2002.

[GHW02b] Giancarlo Guizzardi, Heinrich Herre, and Gerd Wagner. *Towards Ontological Foundations for UML Conceptual Models*, pages 1100–1117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[GHW03] Giancarlo Guizzardi, Heinrich Herre, and Gerd Wagner. *On the General Ontological Foundations of Conceptual Modeling*, pages 65–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[GO14] GO. Documentation — Gene Ontology Consortium. http://www.geneontology.org/page/ontology-documentation, 2014. [Online; available on 25-July-2017].

[Gro03] Object Management Group. Mda guide version 1.0, 2003. [Online; visited on 25-July-2017].

[Gro16a] Object Management Group. Mda - the architecture of choice for a changing world, 2016. [Online; visited on 25-July-2017].

[Gro16b] OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). https://www.w3.org/TR/owl2-overview/, 2016. [Online; visited on 04-June-2017].

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5):907 – 928, 1995.

[GS03] Jack Greenfield and Keith Short. Software Factories - Assembling Applications with Patterns, Models, Frameworks and Tools. *OOPSLA '03 Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27, 2003.

[GS04] J. Greenfield and K. Short. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Timely, practical, reliable. Wiley, 2004.

[Gua] Nicola Guarino. WONDERWEB - Ontology Infrastructure for the Semantic Web — Institute of Cognitive Sciences and Technologies. http://www.istc.cnr.it/project/wonderweb-ontology-infrastructure-semantic-web. [Online; visited on 23-April-2017].

[Gua01a] Nicola Guarino. WONDERWEB Report Summary: DOLCE, 2001. [Online - PDF version; visited on 16-April-2017].

[Gua01b] Nicola Guarino. WONDERWEB Report Summary: DOLCE Modules, 2001. [Online - PDF version; visited on 02-June-2017].

[Gui07] Giancarlo Guizzardi. On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. In *Proceedings of the 2007 Conference on Databases and Information Systems IV: Selected Papers from the Seventh International Baltic Conference DB&IS'2006*, pages 18–39, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[Gui13] Giancarlo Guizzardi. *Ontology-Based Evaluation and Design of Visual Conceptual Modeling Languages*, pages 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Guu] VU University Amsterdam Guus Schreiber. RDF 1.1 Primer. https://www.w3.org/TR/rdf11-primer/. [Online; visited on 04-April-2017].

[Guz96] M. Guzzo, R. & Dickson. Team in organizations: Research on performance and effectieness. *Annual Review of Psychology*, 47:307–338, 1996.

[GW02] Nicola Guarino and Christopher Welty. Evaluating ontological decisions with OntoClean. *Commun. ACM*, 45(2):61–65, 2002.

[GW04] Giancarlo Guizzardi and Gerd Wagner. A Unified Foundational Ontology and some Applications of it in Business Modeling. In *CAiSE Workshops*, pages 129–143, 2004.

[GW05] Giancarlo Guizzardi and Gerd Wagner. *Towards Ontological Foundations for Agent Modelling Concepts Using the Unified Fundational Ontology (UFO)*, pages 110–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[GWAG15] Giancarlo Guizzardi, Gerd Wagner, João Paulo Andrade Almeida, and Renata Guizzardi. Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story. *Applied Ontology*, 10(3-4):259–271, 2015.

[HB06] John Hallam and Herman Bruyninckx. An ontology of robotics science. *Springer Tracts in Advanced Robotics*, 22:1–14, 2006.

[HD09] Martin Hägele and Knut Drachsler. RoSta - Final Deliverable DMa4, 2009. [Online; visited on 18-June-2017].

[Hep07] Martin Hepp. Possible ontologies: How reality constrains the development of relevant ontologies. *IEEE Internet Computing*, 11(1):90–96, 2007.

[Hep13] M. Hepp. *Güterklassifikation als semantisches Standardisierungsproblem*. Deutscher Universitätsverlag, 2013.

[HHDL+08] Martin Hepp, Martin Hepp, Pieter De Leenheer, Aldo De Moor, and York Sure. *Ontologies: State of the Art, Business Potential, and Grand Challenges*, pages 3–22. Springer US, Boston, MA, 2008.

[HHMW12] V Haarslev, K Hidde, R Möller, and M Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web Journal*, 1:1–5, 2012.

[HKP+12] Pascal. Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL2 Web Ontology Language Primer (Second Edition). https://www.w3.org/TR/owl2-primer/, 2012. [Online; visited on 05-April-2017].

[HMA03] Hui-Min Huang, Elena Messina, and James Albus. Toward a generic model for autonomy levels for unmanned systems (ALFUS). *Performance Metrics for Intelligent Systems (PerMIS) Workshop*, 2003.

[Hor08] I Horrocks. Ontologies and the semantic web. *Communications of the ACM*, 51:58–67, 2008.

[HPS12] Matthew Horridge and Peter F. Patel-Schneider. Owl 2 web ontology language manchester syntax (second edition). https://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/, 2012. [Online; visited on 09-June-2017].

[HPsB+04] Ian Horrocks, Peter F Patel-schneider, Harold Boley, et al. SWRL : A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member submission 21*, pages 1–20, 2004.

[HS06] Hans-jörg Happel and Stefan Seedorf. Applications of Ontologies in Software Engineering. *In 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006*, pages 1–14, 2006.

[IBGM08] J Ierache, M Bruno, and R García-Martínez. Ontolog{í}a para el aprendizaje y compartici{ó}n de conocimientos entre sistemas aut{ó}nomos. *VII Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento 2008, Guayaquil, Ecuador, January 30*, 2008.

[Into6] SNOMED International. SNOMED - The Global Language of Healthcare. http://www.snomed.org/snomed-ct, 2006. [Online; visited on 18-July-2017].

[JBK+04] Hyuckchul Jung, Jeffrey M Bradshaw, Shri Kulkarni, et al. An Ontology-Based Representation for Policy-Governed Adjustable Autonomy. In *Proceedings of the AAAI Spring Symposium on Knowledge Representation and Ontology for Autonomous Systems.*, 2004.

[JC10] Krzysztof Janowicz and Michael Compton. The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology. In *Proceedings of the 3rd International Conference on Semantic Sensor Networks - Volume 668*, SSN'10, pages 64–78, Aachen, Germany, Germany, 2010. CEUR-WS.org.

[J.F84] J.F.Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[JS11] Ludger Jansen and Stefan Schulz. The Ten Commandments of Ontological Engineering. In *OBML Workshop*, pages 41–46, 2011.

[JW05] Jan Jürjens and Stefan Wagner. Model Driven Software Development in the Context of Embedded Component Infrastructures. *Lecture Notes in Computer Science*, pages 320–344, 2005.

[Koi13] E. Koivunen, M. R., & Miller. W3C Semantic Web Activity. http://www.w3.org/2001/12/semweb-fin/w3csw, 2013. [Online; visited on 27-Feb-2017].

[Kos03] Jussi Koskinen. Software Maintenance Costs. *Information Technology Research Institute, University of Jyv{\"a}skyl{\"a}*, 200, 2003.

[KOTW06] Holger Knublauch, Daniel Oberle, Phil Tetlow, and Evan Wallace. A Semantic Web Primer for Object-Oriented Software Developers, 2006. [Online; visited on 15-May-2017].

[Kru03] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003.

[KRV14] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Roles in software development using domain specific modeling languages. *CoRR*, abs/1409.6618, 2014.

[KVV06] Markus Krötzsch, Denny Vrandečić, and Max Völkel. ONTOCOM: A Cost Estimation Model for Ontology Engineering. *The Semantic Web - ISWC 2006*, 4273(February 2016):935 – 942, 2006.

[KWB03] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Lab] Knowledge Systems AI Laboratory. Stanford Knowledge Systems, AI Laboratory. http://www-ksl.stanford.edu/. [Online; visited on 09-June-2017].

[Len95] Douglas Lenat. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[LGC14] Juan De Lara, Esther Guerra, and Jes'us S'anchez Cuadrado. When and How to Use Multilevel Modelling. *ACM Trans. Softw. Eng. Methodol.*, 24(2):12:1—-12:46, 2014.

[LM01] Ora Lassila and Deborah L. McGuinness. The Role of Frame-Based Representation on the Semantic Web. *Review of Accounting Studies*, 12(3):369, 2001.

[M. 11] M. Waibel and M. Beetz and J. Civera and R. D'Andrea and others. RoboEarth-A World Wide Web for Robots. *Robotics Automation Magazine, IEEE*, 18(June):69–82, 2011.

[Mar80] Mario Augusto Bunge. *Treatise on Basic Philosophy - Ontology II: A World of Systems*, volume 4. Springer Netherlands, 1980.

[MBG+03] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. WonderWeb Deliverable D18, 2003.

[MGH+] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, et al. OWL 2 Web Ontology Language Profiles (Second Edition). http://www.w3.org/TR/owl2-profiles/. [Online; visited on 23-July-2017].

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[MS88] J.L. Maté and J.P. Sierra. *Ingeniería del conocimiento: diseño y construcción de sistemas expertos*. Informática (SEPA). Sepa, 1988.

[NFA+15] Julio Cesar Nardi, Ricardo De Almeida Falbo, João Almeida, et al. A commitment-based reference ontology for services. *Information Systems*, 54(June):263–288, 2015.

[Nie12] Engelbert Niehaus. Javascript class generator. https://niebert.github.io/JavascriptClassCreator, 2012. [Online as a Github repository; visited on 25-July-2017].

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, pages 2–9, New York, NY, USA, 2001. ACM.

[Obe06] D. Oberle. *Semantic Management of Middleware*. Advances in the Semantic Web And Beyond. Springer, 2006.

[Obi07] Marek Obitko. Ontolgies and Semantic Web - Modularization of Ontologies, 2007. [Online; visited on 24-May-2017].

[Obj08] Object Management Group. Welcome to Papyrus UML web site. http://www.uml.org/, 2008. [Online; visited on 01-Feb-2017].

[Obj13] Object Management Group. MetaObject Facility (MOF) Home Page. http://www.omg.org/mof/, 2013. [Online; visited on 03-March-2017].

[oO] Information Systems Group University of Oxford. HermiT Reasoner: Home. http://www.hermit-reasoner.com/. [Online; visited on 25-July-2017].

[OWL12] OWL Working Group. OWL - Semantic Web Standards. https://www.w3.org/OWL/, 2012. [Online; visited on 01-Feb-2017].

[PBZ+12] Adrian Paschke, Harold Boley, Zhili Zhao, et al. Reaction RuleML 1.0: Standardized semantic reaction rules. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7438 LNCS, pages 100–119, 2012.

[PCRF+13a] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, et al. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.

[PCRF+13b] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, Vitor A. M. Jorge, et al. Towards a core ontology for robotics and automation. *Robot. Auton. Syst.*, 61(11):1193–1204, November 2013.

[Pea06] A Pease. The suggested upper merged ontology (sumo)-ontology portal. http://www.adampease.org/OP/, 2006. [Online; visited on 23-April-2017].

[Pea11] Adam Pease. Formal Ontology and the Suggested Upper Merged Ontology (SUMO), Adam Peace, 20110822 - YouTube. https://www.youtube.com/watch?v=EFQRvyyv7Fs&t=1251s, 2011. [Online - San Francisco Bay ACM's talk; viewed on 20-April-2017].

[Per14] Branko Perisic. Model Driven Software DevelopmentState of The Art and Perspectives. *Infoteh- Jahorina*, 13(March):1237–1248, 2014.

[PHK+11] Using Prot, Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, Simon Jupp, Georgina Moulton, Nick Drummond, and Sebastian Brandt. A Practical Guide To Building OWL Ontologies Using Protégè 4 and CO-ODE Tools Edition 1.3, 2011. [Online; visited on 25-July-2017].

[PM04] Helena Sofia Pinto and João P. Martins. Ontologies: How can they be built? *Knowledge and Information Systems*, 6(4):441–464, 2004.

[PNL02] Adam Pease, Ian Niles, and John Li. The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, volume 28, pages 7–10, 2002.

[Proa] KnowRob's Project. KnowRob - Upper Ontology File. http://knowrob.org/kb/knowrob.owl. [Online; visited on 09-April-2017].

[Prob] KnowRob's Project. Overview of the knowrob upper ontology. http://knowrob.org/doc/knowrob_taxonomy. [Online; visited on 25-July-2017].

[Proc] RobotEarth's Project. RobotEarth - UpperOntology File. http://knowrob.org/kb/roboearth.owl. [Online; visited on 14-June-2017].

[prod] WonderWeb project. FaCT++ reasoner — OWL research at the University of Manchester. http://owl.cs.manchester.ac.uk/tools/fact/. [Online; visited on 30-May-2017].

[Pro12] Proteus Project. Platform for RObotic modeling and Transformations for End-Users and Scientific communities. http://www.anr-proteus.fr/?q=node/110, 2012. [Online; visited on 29-April-2017].

[PSA+13] Jeff Z. Pan, Steffen Staab, Uwe Aßmann, et al. *Ontology-driven software development*. Springer-Verlag Berlin Heidelberg, Berlin, 2013.

[PSB+04] Ron Provine, Craig Schlenoff, Stephen Balakirsky, et al. Ontology-based methods for enhancing autonomous vehicle path planning. *Robotics and Autonomous Systems*, 49(1-2 SPEC. ISS.):123–133, 2004.

[Ram03] Laddad Ramnivas. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., 2003.

[Ree02] Paul Reed. Reference Architecture: The best of best practices, 2002. [Online; visited on 07-July-2017].

[RoS] RoSta. RoSta. http://wiki.robot-standards.org/index.php/Main_Page. [Online; visited on 28-April-2017].

[RoS09] RoSta IST-045304. Deliverable D 2.2: Reference Implementation of Key Architectural Abstractions and Mechanisms, 2009.

[SA01] Alexander Stoytchev and Ronald C. Arkin. Combining deliberation, reactivity, and motivation in the context of a behavior-rased robot architecture. *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA*, pages 290–295, 2001.

[SAD+] Nicolas Seydoux, Mahdi Ben Alaya, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. SAN (Semantic Actuator Network). https://www.irit.fr/recherches/MELODI/ontologies/SAN. [Online; visited on 30-May-2017].

[Sah07] Goutam Kumar Saha. Web ontology language (OWL) and semantic web. *Ubiquity*, 2007(February):1–1, 2007.

[Sch97] David J. Schultz. Ieee standard for developing software life cycle processes. IEEE Std 1074-1997, The Institute of Electrical and Electronics Engineers, New York, USA, 1997.

[Sch06] Douglas C Schmidt. Model-Driven Engineering. *Computer*, 39(February):25–31, 2006.

[Sei03] E. Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, Sept 2003.

[SK03] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.

[SLH+07] Il Hong Suh, Gi Hyun Lim, Wonil Hwang, Hyowon Suh, et al. Ontology-based multi-layered robot knowledge framework (OMRKF) for robot intelligence. *IEEE International Conference on Intelligent Robots and Systems*, pages 429–436, 2007.

[SM05] Craig Schlenoff and Elena Messina. A robot ontology for urban search and rescue. In *Proceedings of the 2005 ACM Workshop on Research in Knowledge Representation for Autonomous Systems*, KRAS '05, pages 27–34, New York, NY, USA, 2005. ACM.

[Smi96] Barry Smith. Mereotopology: A theory of parts and boundaries. *Data and Knowledge Engineering*, 20(3):287–303, 1996.

[SO00] Richard Soley and OMG Staff Strategy Gropu. Model driven architecture. *OMG white paper*, pages 1–12, 2000.

[Sow00] John F. Sowa. *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 2000.

[SPM+12] Craig Schlenoff, Edson Prestes, Raj Madhavan, Paulo Goncalves, et al. An IEEE standard Ontology for Robotics and Automation. *IEEE International Conference on Intelligent Robots and Systems*, pages 1337–1342, 2012.

[SR03] Amith P Sheth and Cartic Ramakrishnan. Semantic (Web) Technology In Action: Ontology Driven Information Systems for Search, Integration and Analysis. *IEEE Data Engineering Bulletin*, 26(December):1–10, 2003.

[SSSS01] Steffen Staab, Rudi Studer, Hans-Peter Schnurr, and York Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, January 2001.

[TB09] Moritz Tenorth and Michael Beetz. KNOWROB - Knowledge processing for autonomous personal robots. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 4261–4266, 2009.

[TPO+06] Phil Tetlow, Jeff Z. Pan, Daniel Oberle, Evan Wallace, Michael Uschold, and Elisa Kendall. Ontology driven architectures and potential uses of the semantic web in systems and software engineering. W3C Working Draft Working Group Note 2006/02/11, W3C, 03 2006.

[UG96] M. Uschold and M. Gruninger. *Ontologies: Principles, Methods and Applications*. AIAI-TR. Artificial Intelligence Applications Institute, University of Edinburgh, 1996.

[UK95] Mike Uschold and Martin King. Towards a methodology for building ontologies. In *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995.

[Voe11] Markus Voelter. From Programming to Modeling-and Back Again. *IEEE Software*, 28(6):20–25, 2011.

[VSB+06] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, K. Czarnecki, and B. von Stockfleth. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley Software Patterns Series. Wiley, 2006.

[W3C14] W3C. Semantic sensor network ontology, 2014. [Online; visited on 28-June-2017].

[WG12] Evan K. Wallace and Christine Golbreich. OWL 2 Web Ontology Language New Features and Rationale (Second Edition). http://www.w3.org/TR/owl2-new-features/, 2012. [Online; visited on 01-Feb-2017].

[Wir95] Niklaus Wirth. A plea for lean software. *Computer*, 28(2):64–68, February 1995.

[Woo04] Sharon Wood. Representation and purposeful autonomous agents. *Robotics and Autonomous Systems*, 49(1-2 SPEC. ISS.):79–90, 2004.

[WW88] Yair Wand and Ron Weber. An ontological analysis of some fundamental information systems concepts. *Proceedings of the Ninth International Conference on Information Systems*, 1988:213–226, 1988.

[WW90a] Yair Wand and Ron Weber. An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*, 16(11):1282–1292, 1990.

[WW90b] Yair Wand and Ron Weber. Toward a theory of the deep structure of information systems. *Conference on Information Systems*, 5:61–71, 1990.

[WW95] Y Wand and R Weber. On the deep structure of information systems. *Information Systems Journal*, 5(3):203–223, 1995.

[YMY+17] Shuo Yang, Xinjun Mao, Sen Yang, Zhe Liu, et al. Towards A Robust Software Architecture for Autonomous Robot Software. In *2017 6th International Conference on Software and Computing Technologies (ICSCT 2017)*, 2017.