TU WIEN Informatics

Knowledge Base for Reconfigurable Safety Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

David Reitgruber, BSc
Matrikelnummer 11777716

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Projektass. Dipl.-Ing. Dr.techn. Thomas Frühwirth, BSc
Projektass. Dipl.-Ing. (FH) Dieter Etz, MBA

Wien, 20. September 2022

David Reitgruber	Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Knowledge Base for Reconfigurable Safety Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

David Reitgruber, BSc
Registration Number 11777716

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Projektass. Dipl.-Ing. Dr.techn. Thomas Frühwirth, BSc
Projektass. Dipl.-Ing. (FH) Dieter Etz, MBA

Vienna, 20th September, 2022

David Reitgruber | Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

David Reitgruber, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. September 2022

David Reitgruber

Danksagung

An erster Stelle möchte ich Thomas Frühwirth, Dieter Etz und Wolfgang Kastner für ihre großartige Unterstützung bei der Erstellung dieser Arbeit danken. Für die hilfreichen Anregungen und die konstruktive Kritik während der Erstellung dieser Diplomarbeit möchte ich mich ganz herzlich bedanken. Außerdem danke ich dem Center for Digital Production GmbH für die Finanzierung dieser Diplomarbeit. Ein besonderer Dank gilt meinen Kommilitonen, die mich während meines gesamten Studiums großartig unterstützt haben und ohne die meine Zeit an der TU Wien nicht so spannend gewesen wäre. Abschließend danke ich meinen Eltern, die mich während meiner gesamten Studienzeit unterstützt haben, meinem Bruder, den ich immer um Hilfe bitten konnte, und meiner Freundin, die immer ein offenes Ohr für mich hatte.

Acknowledgements

First and foremost, I would like to thank Thomas Frühwirth, Dieter Etz, and Wolfgang Kastner for their great support while creating this thesis. I would like to express my sincere thanks for the helpful suggestions and constructive criticism during the preparation of this thesis. Further, I would like to thank the Center for Digital Production GmbH for financing this thesis. A special thanks goes to my fellow students, who provided great support throughout my studies and without whom my time at TU Wien would not have been so exciting. Finally, I would like to thank my parents for supporting me throughout my university time, my brother, whom I could always ask for help, and my girlfriend for always having an open ear for me.

Kurzfassung

In den letzten Jahrzehnten war es charakteristisch, dass industrielle Fertigungssysteme für die Herstellung eines einzelnes Produktes entwickelt wurden. Das Durchführen von Änderungen am System, um ein anderes Produkt herzustellen, war meist langsam und umständlich. In der heutigen Zeit unterziehen sich diese Systeme einem Wandel. Mit Industrie 4.0 sind dezentralisierte Systeme üblicher geworden, womit stark vernetzte Umgebungen entstehen. Intelligente Fertigungssysteme gewinnen an Popularität und ein Trend weg von Massenfertigung ist erkennbar. Kunden möchten individuelle Produkte, womit eine Anpassung der Fertigungssysteme erforderlich wird. Bei solchen veränderbaren Systemen stellt sich die Frage, wie die funktionale Sicherheit innerhalb dieser gewährleistet werden kann. Für ein System, das eine schnelllebige Produktionsumgebung ermöglicht, muss sich auch das Sicherheitssystem entsprechend ändern können. Mit dem aktuellen statischen Ansatz zur funktionalen Sicherheit ist dies aber nicht möglich. Es muss ein dynamisches Sicherheitssystem geschaffen werden.

Diese Arbeit befasst sich mit einigen der Probleme, die bei diesem dynamischen Ansatz auftreten. Es wird eine Wissensbasis entwickelt, welche sicherheitsrelevante Informationen über das Fertigungssystem speichert. Die relevanten Domänen werden beschrieben und bestehende Ontologien, die diese Domänen abdecken, werden recherchiert und bewertet. Für Domänen, die von bestehenden Ontologien nicht ausreichend abgedeckt werden, werden neue Ontologien erstellt. Die Wissensbasis wird dann anhand zuvor definierter Kompetenzfragen evaluiert und in einem Anwendungsszenario eingesetzt. In diesem Anwendungsszenario stellt die Wissensbasis die Konfiguration für das Sicherheitssystem bereit, welches Time Sensitive Networking (TSN) und Open Platform Communications Unified Architecture (OPC UA) verwendet, um die Kommunikation zwischen Komponenten zu ermöglichen.

Abstract

In the last few decades, it was typical for industrial manufacturing systems to be developed for the creation of a single product. Performing changes to the system to create a different product was usually slow and cumbersome. In today's time, those systems are changing. With Industry 4.0, decentralized systems have become more common, which creates a highly interconnected environment. Smart manufacturing systems are gaining popularity, and the trend of dedicated manufacturing systems for only one product is decreasing. Customers wish to individualize products, which means an adaptation of the manufacturing system is required. With these changing systems, the question of how safety can be ensured within them arises. For a system to allow for a fast-paced production environment, the safety system must also be able to change accordingly. With the current static approach for functional safety, this is not possible. A dynamic safety system needs to be developed.

This thesis addresses some of the problems that arise with this dynamic approach. A knowledge base is developed, which stores safety-relevant information about the manufacturing system. The relevant domains are described, and existing ontologies covering these domains are researched and rated. For domains that are not suitably covered by existing ontologies, new ontologies are created. The knowledge base is then evaluated based on before-defined competency questions and used in an application scenario. In this application scenario, the knowledge base provides the configuration for the safety system, which uses Time Sensitive Networking (TSN) and Open Platform Communications Unified Architecture (OPC UA) to enable communication between components.

Contents

CHAPTER 1

Introduction

The fourth industrial revolution changes the approach to organizing automation systems. Their structure changes from a pyramid to a pillar, shown in Figure 1.1, resulting in a decentralized system connected via real-time communication [33]. This new structure creates a high demand for interoperability and connectivity of the different components within a system. It is also the basis for smart manufacturing systems, which utilize this connectivity to collect and analyze data provided by the components. The information gathered can then be used for different purposes, e.g., to check the state of a component or even to increase the overall system's productivity. Time Sensitive Networking (TSN) is a technology that provides the required properties for communication in such a decentralized system. It allows for high-speed, low-latency communication within the field level and connectivity level while carrying low-priority background traffic without slowing down the time-critical traffic. In order to allow for high interoperability and connectivity of the different components, the Open Platform Communications Unified Architecture (OPC UA) standard can be used. It defines transport mechanisms for how data between two components can be exchanged [51]. Further, it defines an information model specifying rules and objects on how data is exposed. OPC UA services are the link between the transport mechanisms and the information model. They define how data from the information model can be read.

Another change that comes with the transition to Industry 4.0 is the convergence of the Information Technology (IT) and Operational Technology (OT) domain. In Industry 3.0, a distinction between IT and OT can be made. IT systems cover data-centric computing, including creating, processing, storing, and exchanging of data. OT systems are used in an industrial environment. They monitor processes, events, and devices and adjust industrial operations. Over the last few years, the distinction between those domains has become increasingly unclear. With Industry 4.0, no clear difference between them can be made. Concepts from OT are brought into IT. For example, physical components, such as sensors, can upload data to a central server for monitoring and analysis. Results can

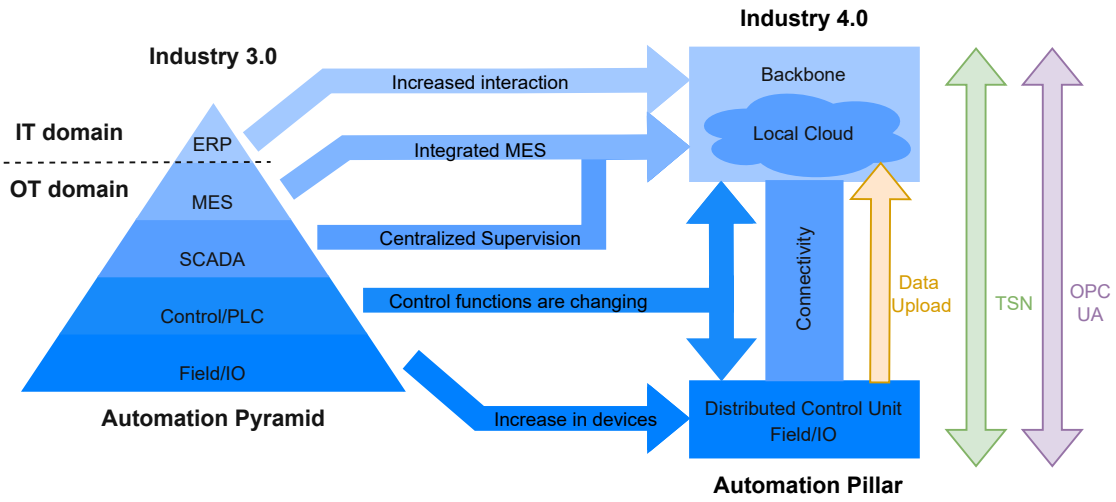


Figure 1.1: Transition of automation systems organization (adapted from [7])

then be passed down, allowing for the autonomous operation of independent components. Using knowledge representation concepts, it is also possible to store information regarding the OT domain and use it within the IT domain. A knowledge base can store relevant information about a system, which can then be provided to operators or other system components when needed.

An essential property of Industry 4.0 is the on-demand individualization as described by Lasi et al. in their paper [48]. This trend causes an increasing individualization of products, resulting in a dynamic change in the production needs of a manufacturing system. Such flexible manufacturing systems are one of the main advantages of Industry 4.0, as they allow for adaptive production, which is enabled by the connectivity and interchangeability of the components. With the interchangeability comes the need for reconfigurable systems to adjust the parameters to fit the new configuration. One thing to keep in mind with these systems is the safety aspect. A dynamic safety system that could be reconfigured at runtime would greatly benefit the flexibility of manufacturing systems.

1.1 Motivation

Figure 1.2 shows an example of this reconfigurability within a manufacturing system described by Etz et al. in [22]. Mobile devices such as Automated Guided Vehicles (AGVs) can aid the flexibility of a manufacturing system. To allow the AGV to enter a production cell, *muting* can be used. It defines a safe, automatic, and temporary bypass of non-contact safety devices, such as a light barrier. AGVs can be equipped with safety components, such as laser scanners or an emergency stop button. Consequently, when an AGV enters or leaves a production cell, the safety configuration needs to be adapted.

This adaptation should be made without going into a safe state to keep the production running. This automatic reconfiguration would need a mechanism that affiliates the AGV to a specific safety area. The process of this reconfiguration has to be as follows. The AGV has to notify the system before it leaves the current cell to remove the device from the safety configuration. Once this is done, the AGV can move to its new destination. On arrival, the system needs to be notified to add the device to the safety configuration. By doing so, the overall safety concept of the system then includes the safety components of the AGV.

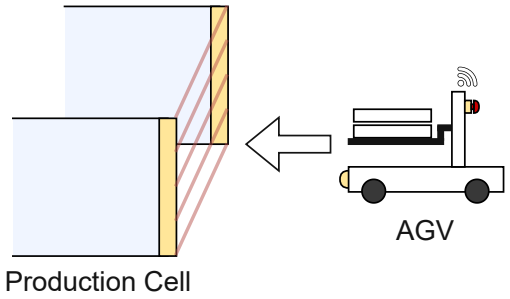


Figure 1.2: Example of a reconfiguration of an manufacturing system (adapted from [22])

This example shows why dynamic and reconfigurable safety systems are required for Reconfigurable Manufacturing Systems (RMSs). It greatly benefits the flexibility and productivity of manufacturing systems. Including and excluding new components without disrupting the production would allow engineers to perform maintenance tasks on parts of the systems while the others still operate. Not only could an AGV enter and leave safety areas, but safety sensors and actuators could be added on the fly, aiding the system's safety and keeping the throughput high.

1.2 Problem Statement & Goal

Current functional safety systems cannot provide the desired properties needed for Industry 4.0 [23]. The usual approach to functional safety was to assess and analyze possible system risks and provide measures to prevent or reduce them adequately. This approach was a static one, performed at design time, and the configuration would not change once the system was in operation, the configuration would not change. With flexible and reconfigurable systems, the configuration is no longer static and needs to change while the system operates.

Etz et al. present in [23] a model for a self-organizing safety system, the goal of which is to assist the safety engineer with generating the configuration. As shown in Figure 1.3, it consists of two main components, the so-called Flexible Safety System (FSS) and the Cyber-Physical Production System (CPPS). The authors define four tools to assist the safety engineer within the FSS. The automatic *Discovery* identifies all safety-related devices within the system. Based on the discovered devices, the *Safety Configuration*

Generator creates a suitable safety configuration. The generated configuration then undergoes a *Plausibility Check* before it gets passed to the *Deployment*, which transfers the configuration to the correct devices. In order to ensure the highest standards of the functional safety system, an *Adaptation Validation* building block is added. Over a *Human Control Interface* and a *Human Machine Interface (HMI)*, the generated safety configuration must be verified by a human for completeness and correctness before it can be deployed. A key component connected to all these building blocks is the *Knowledge-Based System (KBS)*. It contains information needed for each of the tools, such as a *Machine Model* or *Legal Regulations*, which are represented by the *Knowledge Base* block. With the *Inference Engine*, it is possible to derive new information from the knowledge base.

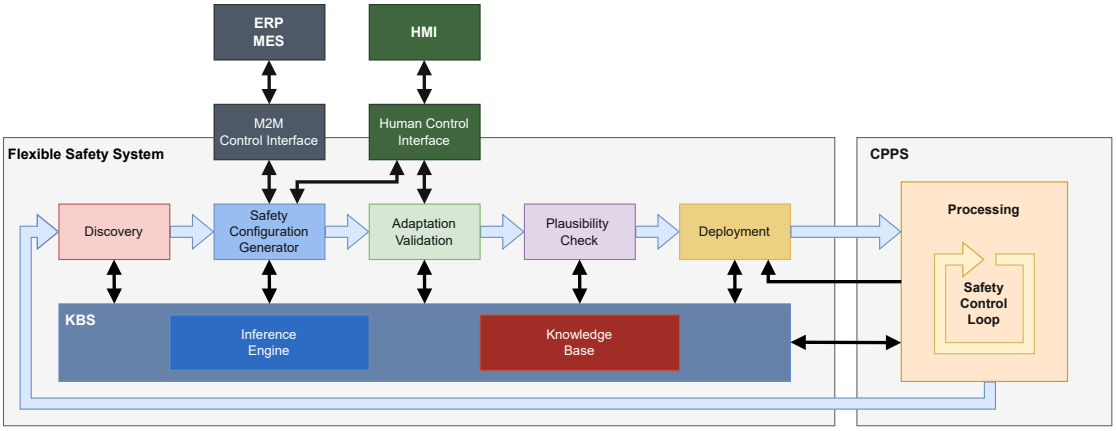


Figure 1.3: Self-organizing safety system model (adapted from [23])

For the safety system to be complicit with an RMS, it needs to be adapted and extended, as stated by Etz et al. [22]. They propose the definition of an Reconfigurable Safety System (RSS) and identify six core characteristics that an RSS has to possess. The core characteristics *Modularity* and *Integrability* overlap with those of an RMS as defined by Koren et al. in [44]. Modularity describes the system's decomposability into smaller parts that can be designed and produced independently but can be combined to work together. Integrability specifies the ability of a system to integrate new modules without problems. *Flexibility* describes the capability of modifying a system's hardware or software configuration. This modification can be done by adding, removing, or changing safety components. This characteristic also aligns with a core characteristic customization for an RMS defined in [44]. With *Interoperability*, the capability of exchanging information between different devices is defined. *Referenceability* specifies that changes made to a system must be documented for future reference. Lastly, *Comprehensibility* defines the ability to enhance an individual's understanding of a system. The authors then convert these six abstract characteristics of an RSS, in combination with those of an RMS, and the building blocks provided by the FSS model into five service groups. These

five service groups are *Knowledge Representation*, *Discovery*, *Configuration*, *Visualization & Modification* and *Deployment*.

Based on the structure shown in Figure 1.3 and corresponding to the knowledge representation service, this thesis will develop a knowledge base for a system where relevant information can be stored and accessed by the different components via interfaces. The main focus of this knowledge base will be on the system's safety aspect.

1.3 Methodological Approach

The methodological approach for this diploma thesis consists of the following steps:

1. **Literature review:** An extensive review and analysis of relevant literature will be done to get a deeper knowledge of this topic and establish state of the art. A short dive into the methodology of developing ontologies will be performed. The results of this review will provide comprehensive information on currently used approaches. This will define the starting point and needed building blocks for developing the knowledge base.

2. **Requirements analysis:** Following the literature review, a requirements analysis will be performed. It should show which function the ontology needs to provide and what the interfaces should look like to enable other components to access the knowledge base. Here, the so-called competency questions will be defined, which specify questions the developed ontology should be able to provide answers to.

3. **Development of a Knowledge Base:** Based on the results from the literature review and the analysis, a knowledge base will be developed. The knowledge base must fulfil the requirements stated in the analysis. In the development process, already existing ontologies should be used as the base building blocks and are then extended to fit the desired needs.

4. **Evaluation:** The evaluation of the developed knowledge base will be done in two parts. First, the ontology will be evaluated using the previously defined competency questions. Here, it will be assessed if the ontology can be used to answer the given questions. The second part will be the evaluation using a real-world example. Here, the developed knowledge base will be used within a simple example system, such that the primary function of the ontology itself and the interfaces to the different components of the system can be evaluated.

1.4 Structure of This Work

In Chapter 1, an introduction to the topic was given, followed by the motivation for this work. Then, the problem statement was specified, and the goal this thesis should achieve was set. Chapter 2 provides information about the technical background relevant to this

work. First, some context about RMSs is given. Then, information about functional safety and knowledge engineering is provided. Finally, some related work is presented. Chapter 3 specifies the methodology used for the knowledge base creation. In Chapter 4, a requirements analysis is performed to define the requirements for the knowledge base. Chapter 5 is focused on the creation of the needed ontologies. Existing ontologies are rated, and new ones are created before they are combined into the final ontology. This ontology is then evaluated in Chapter 6 using predefined competency questions and integrating it into an RSS. Finally, in Chapter 7, the conclusion is stated, and an outlook for future work is presented.

CHAPTER 2

Technical Background

The following sections present some relevant technical background and related work. First, some historical information on manufacturing systems is shown. Details about the evolution of such a system over the last decades are given, and the different concepts are explained. Then, a definition for RMSs is given, and the differences to other designs are presented. After this introduction on RMSs, information about functional safety is given. This thesis starts with a general definition of the topic, then shows how functional safety applies to manufacturing systems. After this, it presents the relevant standards and explains how they relate. The information these standards provide is needed as a starting point for creating a knowledge base. Following the standards is a section about knowledge engineering. Here, information about the representation and modelling of knowledge is provided, which is a key aspect of creating a knowledge base. Finally, more details about ontologies and important terms in this context are given.

2.1 Reconfigurable Manufacturing Systems

The term *manufacturing system* was used as early as 1815 when it meant a factory system [59]. Since then, the term's meaning has changed as it had no scientific basis [76]. Nowadays, the term defines the conversion of raw material into the finished product, which is planned and controlled by a management system [31]. The definition shows two aspects of such systems: the conversion and management systems. Concerning the former, four different manufacturing systems can be distinguished: Dedicated Manufacturing Systems (DMSs), Adjustable Manufacturing Systems (AMSs), Flexible Manufacturing Systems (FMSs) and Reconfigurable Manufacturing Systems (RMSs).

DMSs, which are built since the early 20th century, are sometimes also called a transfer line [76, 45]. The main architecture of these systems consists of several special-purpose machines connected by a conveying system, and a dedicated control system controlled all the parts. This design is aimed to produce a single part fast at a high volume. DMSs are

a main contributor to mass production and have the advantage that they make products cheaper and more efficient to produce, and they provide a fixed quality. The disadvantage is that products can not be customized and individualized based on the customer. Once decided on a design, the system can produce only this product. The system has to be changed to allow for a new product to be produced. This rigidity makes it unfit for many of today's production needs where often a small volume is desired.

AMSs are built since the 1940s and consist of several adjustable machines [76]. This structure allows for slight adjustments in the production of parts. Machines used in such systems are spindle-head-changing machines, program-controlled machines, and unit-built machines. As the name states, with spindle-head-changing machines, it is possible to change the head of the spindle to produce different workpieces. Program-controlled machines have a preset program control for their operation sequences. Unit-built machines consist of standard units and a few special-design units. This modularization brings advantages to the production system. It allows for easier changes in the system, cuts down on the design period, and simplifies the production, among others. These properties also make them suitable for mass production.

FMSs started to be used in the 1960s and reached their height in the 70s and 80s [76]. These systems consist mainly of general-purpose Computer Numerical Control (CNC) machines and programmable automation machines like automated guided vehicles or industrial robots [45]. The high flexibility of FMSs allows them to manufacture multiple products but in smaller batch sizes. Because of these small batch sizes and the high cost of CNC machines, FMSs are unsuitable for mass production. By using multi-group CNC, it is possible to adapt FMSs for mass production, but it still results in high production costs. So the main disadvantages of this system are cost, low production rate, and complexity. They are therefore not fit for some of today's desired production concepts.

The aforementioned systems are at a disadvantage regarding the new production demands. This is why Koren et al. describe a new class of systems: RMSs [45]. This new type aims to combine the high throughput of DMSs with the flexibility of FMSs. They achieve this by designing the system according to the following two principles. First, design the system to be adaptable to new products and scalable in response to market demand. Second, to design it around the part family, so it is possible to produce all parts of the part family. This leads us to the following definition of RMSs:

> *"A Reconfigurable Manufacturing System (RMS) is designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in market or in regulatory requirements."[45]*

Compared to DMSs and FMSs, RMSs bring improvement in production ability and cost [76]. DMSs have a fixed production ability. If the demand is higher than it can produce, the system needs to be redesigned, resulting in high costs. On the contrary, RMSs can be adjusted to the market need because of its reconfigurability. Since the system is dynamic and can be developed continuously, it can supply a high ratio of functions over a long period. It is important to define the term reconfigurability in this context as it is not unlimited. Here, it refers to the change of a system to produce different parts within a part family.

Figure 2.1 shows the evolvement of manufacturing systems over time. It shows how the product variety is connected to the product volume per model over time. Starting in the 1850s, craft production with great variety and smaller volume was the main focus. For this, general-purpose machine tools were used. The graph then gradually shifts to lower product variety with higher volume. DMSs came into play, and mass production became more common. From there, a decline in volume and increase in variety can be observed. With mass customization in the 1960s, where FMSs were prominent, it then led to individual products and RMSs, which were proposed since the 1980s.

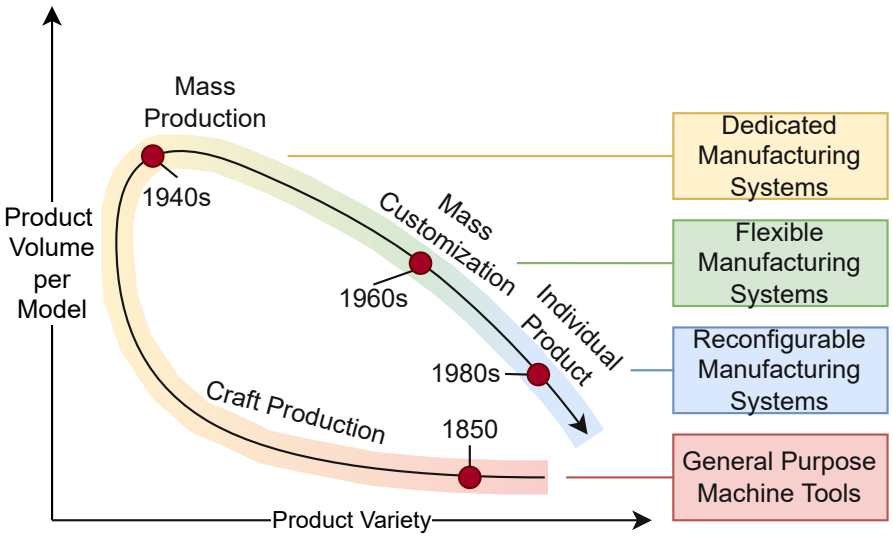


Figure 2.1: Overview of different manufacturing systems (adapted from [49])

2.2 Functional Safety

With the rise of Industry 4.0 and the growing complexity of industrial safety systems, functional safety is more relevant than ever. The goal of functional safety is defined by the IEC 61508 as *"freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment" [34].* Industrial automation systems achieve this goal by implementing a

safety system consisting of one or more safety functions. A safety function is defined by the ISO 12100 as a function of a machine whose failure can immediately increase the risks [37]. For example, if a machine has safety doors to provide access to the machining area, a safety function can be to stop the machine if a door is opened.

An essential document regarding the safety of machinery in the European Economic Area (EEA) is the Directive 2006/42/EC of the European Parliament and of the Council, also called Machinery Directive [20]. Its central intent is to regulate a common safety level for machinery placed on the market within the EEA. It further provides minimum standards regarding health and safety requirements, which are refined through harmonized standards such as the IEC 62061 [36] or the ISO 13849–1 [39].

In order to ensure the safety of machinery, the risk needs to be identified, and possible countermeasures need to be taken. The ISO 12100 [37] provides an iterative process to do so, which consists of the following five steps:

1. Establishing the limits of a system, including the intended use and reasonably foreseeable misuse
2. Identifying hazards and related hazardous situations
3. Estimating the risk for each of the identified hazards and situations
4. Evaluating the risk and deciding the need for risk reduction
5. Eliminating the hazard or reducing the risk associated with the hazard with protective measures

Safety requirements are defined based on the detected risks which need to be realized. So for each identified hazard, a safety function needs to be implemented.

The hazards are classified to give a guideline for implementing these safety functions. There are two main metrics when it comes to classifying hazards. First, the Safety Integrity Level (SIL), which is defined in the IEC 61508 [34]. Second, the Performance Level (PL) as defined by the ISO 13849–1 [39]. With the SIL, the category can be defined using four criteria: severity of the damage, frequency and duration of the suspension, probability of occurrence, and the possibility of avoidance. The corresponding SIL is assigned based on these values. Starting with *SIL1*, which is used to protect from small risks, up to *SIL4* for high-risk application. The PL classification uses the same four criteria but utilizes a different granularity for each. This results in five PLs, starting with *a*, which represents small risk, up to *e* for the highest risk. It is possible to convert from one to the other using the Probability of dangerous Failure per Hour (PFH_d), which is defined for each level of both classifications. The connection between SIL and PL is shown in Table 2.1.

SIL	$PFH_d[1/h]$	PL
—	$10^{-5} \leq PFH_d < 10^{-4}$	a
SIL1	$3 \cdot 10^{-5} \leq PFH_d < 10^{-5}$	b
SIL1	$10^{-6} \leq PFH_d < 3 \cdot 10^{-5}$	c
SIL2	$10^{-7} \leq PFH_d < 10^{-6}$	d
SIL3	$10^{-8} \leq PFH_d < 10^{-7}$	e

Table 2.1: Connection SIL/PL [3]

Once the safety functions are implemented, it is essential to determine their integrity. As a final step, the safety system is validated against the defined safety plan. This validation then results in documented evidence of compliance with the safety requirements. [27]

The differences in the industrial sectors make it hard to define a universal standard covering the topic of safety. These differences are reflected by the number of standards that exist in this context. The IEC 61508 functions as a base standard for safety and applies to all industries. For the safety of machinery, there is the IEC 62061, which covers Electrical, Electronical, and Programmable Electrical (E/E/PE) control systems. Further relevant to this topic is the ISO 13849, which includes hydraulic and pneumatic machinery. Then there are standards that cover specific branches of industry, e.g., the IEC 61513 for nuclear power plants, the IEC 50128 for the rail industry, or the IEC 61800 for electrical power drive systems. Figure 2.2 shows how the different standards are connected.

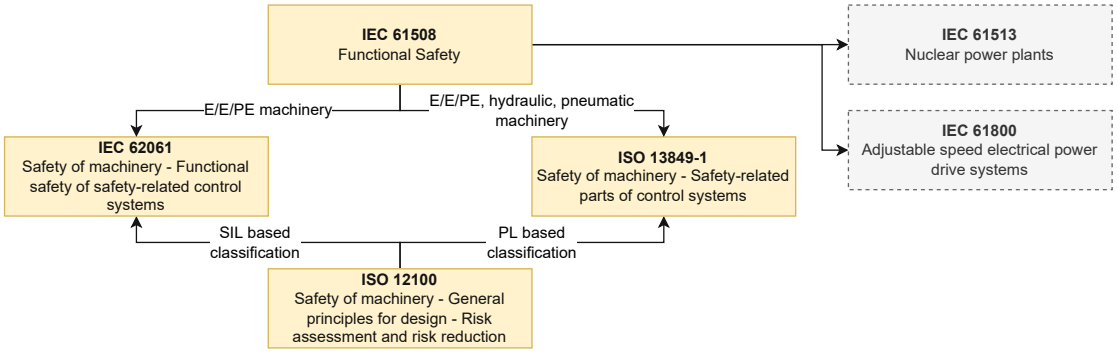


Figure 2.2: Relation of safety standards (adapted from [1])

The following sections give a deeper insight into relevant standards for this thesis, as they provide a more general view of functional safety.

2.2.1 IEC 61508

The IEC 61508 [34] can be considered the base standard for functional safety. This central standard provides information concerning functional safety for automation systems regardless of the application. It covers the complete lifecycle of such systems, from development up until retirement. The standard is split into seven parts.

Part 1 provides an introduction to the concept of functional safety. Starting with some information on when to apply this standard, how it is related to other standards, and some meta information, it then shows the documentation process necessary to achieve functional safety in all lifecycle phases. After this, it provides a way to define the responsibilities of the people who manage the functional safety system. Then, the central part of the standard follows, which provides the requirements for the safety life cycle. Here, the necessary steps to achieve functional safety of a system are provided in detail, starting from the concept, over the risk analysis, to the implementation, the validation, the usage, until the system's retirement. The last part of this standard is concerned with assessing the achieved functional safety based on the relevant chapters of the standard itself.

Part 2 of the IEC 61508 covers the process of achieving functional safety with the focus on E/E/PE systems. It is structured in the same way as Part 1, with the difference that the central part, which provides the requirements for the safety life cycle, focuses on E/E/PE-systems.

Part 3 provides the information needed to create functionally safe software. Again, following the structure of Part 1, it then focuses on the requirements for software used in automation systems to achieve functional safety over the whole life cycle.

Part 4 provides terms and abbreviations. In Part 5, some examples of evaluating the SILs are given. To further specify the application of Part 2 and 3, Part 6 shows policies on when and how to apply them. Lastly, Part 7 contains usage instructions on procedures and measures of Parts 2 and 3.

2.2.2 IEC 62061

The IEC 62061 [36] is an important standard for the safety of machinery. It is a sector-specific standard below the IEC 61508 and covers the functional safety of Safety-related Control Systems (SCSs) for E/E/PE systems. The design process of an SCS and the management of functional safety are defined in it. The design process is an iterative one. The safety functions are designed starting with information gathered from the risk assessment. First, the requirements are defined, and then the safety integrity is determined before an SCS is designed to implement the safety function, which a combination of subsystems can achieve. After this design step, the subsystems are combined and then validated to check whether all requirements are met and achieve the desired safety integrity. This process is repeated for all safety functions. Once finished, the information is passed back to the risk assessment. The management of functional safety is done by

utilizing a functional safety plan. This plan contains information about the specification, implementation, or modification of an SCS and is intended to provide measures against an incorrect execution.

It is important to note that the currently harmonized version of the IEC 62061 was published in 2015. In February 2021, a new revision was released, which not only provides an update of the standard but no longer restricts its application to E/E/PE systems. It can now be applied to all kinds of technology, like pneumatic and hydraulic systems. This update is now similar to the ISO 13849, but it is not harmonized, yet.

2.2.3 ISO 13849

Similar to the IEC 62061, the ISO 13849 is concerned with the safety of machinery. As mentioned above, it is not focused on E/E/PE systems, but its application field also includes hydraulic and pneumatic systems. It is divided into two parts.

Part 1 [39] provides a methodology and requirements for the design and integration of Safety-Related Parts of Control Systems (SRP/CSs), which also includes the design of the software. The design process is again an iterative one and follows a similar pattern as it is shown in the IEC 62061. After performing a risk analysis, the safety functions are specified, including the determination of the PL. Then, the SRP/CS is designed to implement the safety function. Next, the PL is evaluated and verified if the required one is achieved. As the last step, the safety function is validated. This process is then repeated for all safety functions. Once finished, the information is passed back to the risk assessment.

Part 2 [38] addresses the validation of SRP/CSs. The validation is performed according to a validation plan and consists of two main steps: analysis and testing.

2.3 Knowledge Engineering

Knowledge Engineering focuses on the development of intelligent systems [55]. In the beginning, it was seen as a transfer process, the goal of which was to transfer human knowledge into a knowledge base. The primary assumption behind this was that the required knowledge already existed and only needed to be acquired and stored. One way to do so was by interviewing domain experts.

Maintenance of such implemented knowledge bases was complex and was thus only feasible for small systems as it did not scale well for large ones [72]. Another problem with the assumption that the knowledge only had to be collected was the missing tacit knowledge which is applied to problem-solving. This resulted in a paradigm shift from the transfer to the modelling approach.

The goal of this new approach was to build a computer model that could provide the same problem-solving capabilities as a domain expert. Knowledge needs to be built up and structured during the acquisition process to provide these capabilities. This means

it was no longer a transfer process but a model construction process. The following three consequences result from this modelling view [15, 54]:

- The model is only an approximation of the real world since, in principle, the modelling process is infinite.
- The modelling process is a cyclic one. New observations can lead to a refinement of the model, whereas the model can help acquire new knowledge.
- The modelling process is subjective to the modeller and, therefore, typically faulty. An evaluation of the model in comparison to the real world is essential.

While performing an analysis of systems based on the transfer approach, Clancey discovered that they describe a common problem-solving behaviour [14]. He was able to abstract this behaviour to a generic inference pattern. With this heuristic classification, as it was called, it was possible to describe the problem-solving behaviour on an abstract level. The knowledge level, as Newell called it in [56], made it possible to describe reasoning by defining goals, the actions to reach them, and the knowledge required for those actions. This leads to the definition of a Problem-Solving Method (PSM), which describes the reasoning process of a KBS in a domain- and implementation-independent way [8, 72]. Birmingham and Klinker describe three ways to characterize a PSM [9]:

- The PSM specifies which inference actions need to be performed to solve some task.
- The PSM specifies in which sequence these actions have to be performed.
- Knowledge roles define what role the domain knowledge has in each inference action. These roles specify a domain-independent generic terminology.

PSMs can be used for knowledge engineering differently. One way would be to use the contained inference actions which need specific knowledge. They can be used as a guideline to gather static domain knowledge. Another way is to use them for validation of a KBS since PSMs can describe the reasoning process of a KBS. Furthermore, since PSMs can be reused, it is possible to create libraries of them, which can help with the construction of KBSs [72].

2.3.1 Knowledge Representation

Knowledge representation is another crucial element of knowledge engineering. It aims to represent information so that a computer system can use it to reason and complete tasks. The information over a specified domain is modelled so that symbols represent this domain's elements. Elements of a domain can be physical objects, dependencies between objects, and so on [70]. At the core of knowledge representation is the knowledge base, which stores the symbols of the domain of interest in statements. A system can then

reason over this domain by manipulating these statements and posing questions to the knowledge base.

There are several ways to represent knowledge, such as frame logics, description logics, and conceptual graphs. Frame logic [41] combines the data modelling capabilities of object-oriented data models with the declarative semantics expressiveness of deductive database languages [47]. Conceptual graphs and description logics are subtypes of semantic networks based on first-order logic. Semantic networks are used to depict semantic information using a graphical representation.

Ontologies

Ontologies build on the aforementioned notions of knowledge representation. They are conceptual models, meaning they use intentional descriptions of concepts and their interrelations to represent the knowledge. The term *Ontology* is used to refer to the shared understanding of some domain [74]. Ontologies can be visualized as a semantic network. Figure 2.3 shows an example of a simple ontology. The rectangles represent *classes* of this ontology, sometimes also called *concepts*. The arrows define the relationships between them, called *properties*. Instances of classes are called *individuals*. Relationships between classes (and individuals) to one another are called *object properties*. Classes (and individuals) can also be related to data values. These relations are called *data properties*. Nodes connected by a relation create a *triple* in the form of *Subject Predicate Object*. To represent ontologies, different Semantic Web languages can be used, such as Resource Description Framework (RDF) or Web Ontology Language (OWL) [4]. RDF uses triples as the base building blocks and serves as a foundation for other Sematic Web languages. OWL builds on top of RDF and adds language constructs, which results in a greater expressiveness. Databases that hold triples are called *Triple Stores* or *RDF Stores*. To query information from these databases, SPARQL Protocol and RDF Query Language (SPARQL) can be used. More information on ontologies, the semantic web, and SPARQL can be found in [13, 32, 60].

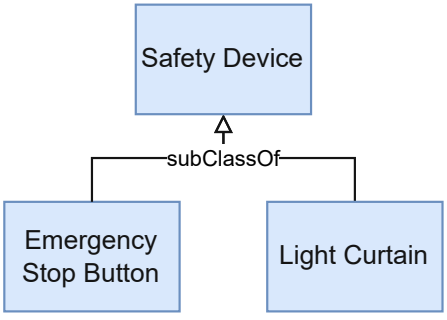


Figure 2.3: Simple ontology

2.3.2 Approaches

The following section describes different approaches used in knowledge engineering. First, two historical ones are explained, which influenced newer modelling approaches. Then, some more well-known ones are presented in more detail. Finally, this thesis briefly mentions other approaches, which are not further explained.

Role-Limiting Methods

The first historical relevant approach was *Role-Limiting Methods* [52]. These methods utilize the reusability of PSMs. This approach can be seen as a shell approach, where a shell holds an implementation of a PSM. This had the downside of only using it for appropriate tasks where the PSM can be applied. The PSM further defined the roles knowledge plays during the problem-solving process and gives a fixed representation for it. The knowledge engineer only had to instantiate these roles' concepts and relations and provide the required knowledge. One problem with this approach is that it is unclear whether a specific task can be solved by a given role-limiting method. Another problem with the fixed structure of these methods is that they do not provide a good basis to solve a task if it can only be solved by combining multiple PSMs. To overcome this problem, *Configurable Role-Limiting Methods* have been proposed in [63].

Generic Tasks

The second historical relevant approach was *Generic Tasks* [11]. Such tasks can be seen as building blocks that can be reused to create KBSs. Each task specifies a generic description of its in- and output and a fixed scheme of the structure of the needed domain knowledge. It also includes a fixed problem-solving strategy, which specifies the inference steps. This approach has no clear distinction between the notion of a task and the PSM used to solve the task. It was also unclear what the granularity of these building blocks should be. The *Task Structure* approach was proposed [12] to overcome this problem. This approach made a clear distinction between a task and a method. The concept of task-method-decomposition is nowadays reused in other knowledge-engineering methodologies [72].

CommonKADS

Schreiber et al. present another knowledge modelling approach with *KADS* [66] and its successor *CommonKADS* [67]. The main idea behind the KADS approach is to create multiple models, where each of them captures specific details about the KBS and its environment. CommonKADS consists of the following models:

- The *Organization Model* contains information about the organizational structure and a specification of which functions each organizational unit performs. It also holds information about current deficiencies of the business process and how it can be improved by the KBS.

- For each organizational unit, the *Task Model* provides a hierarchical description of its performed tasks. It also specifies which agent performs a task.
- The *Agent Model* provides the capabilities of each agent for a currently executed task. Agents can be humans or software systems.
- The *Communication Model* defines the interaction between the different agents, e.g., what information is exchanged.
- In the *Expertise Model* three different types of knowledge are stored, which can be structured into layers:
  - The *Domain layer* holds domain-specific knowledge.
  - The *Inference layer* specifies the reasoning process of the KBS by using PSMs. It also describes inference actions and the roles the domain knowledge has. A connection between those is achieved with an inference structure.
  - The *Task Layer* defines tasks broken down into subtasks. It includes inference actions, a goal specification, and how these goals are achieved.
- The *Design Model* defines the system architecture and the computational mechanisms for implementing the inference actions. The structure of the Expertise Model should be reflected in the Design Model as much as possible

The first four models cover the organizational environment of the KBS and the tasks performed in the organization. The last two cover functional and non-functional components of the KBS.

VITAL

The VITAL approach [69] is based on KADS multiple model approach to describe the KBS. It differs from the KADS approach by putting its main focus on the notion of a process product. A project using a KBS can provide four of these process products:

- The *Requirements specification* describes an application's expected functions and constraints.
- The *Conceptual model* holds domain-relevant entities, task structures, and expert problem-solving behaviour.
- The *Design models* consist of a functional-design model and a technical-design mapping. The functional design gives an implementation-independent description of the KBS. The technical design provides an implementation-dependent mapping from the functional design to executable code.
- The *Executable code* is the final process product, which are the different software components of an application.

In addition to the process products, the methodology includes the following three components:

- *Knowledge-engineering methodology*: defines the production of the process products for a specific application
- *Life-cycle model*: controls and monitors the entire project
- *Life-cycle configuration*: links the former two, it defines how each process product is developed in the context of the project

PROTÉGÉ-II

PROTÉGÉ-II aims to provide a knowledge-engineering environment for developers. With the PROTÉGÉ-II approach [65, 21], the main focus is placed on reusing PSMs and ontologies for the development of KBSs. As with Generic Tasks, this approach utilizes the task-method-decomposition. Tasks are decomposed into subtasks by applying a PSM. This decomposition is done until a level is reached where primitive methods can solve the subtasks. The input and output of a method are defined in the *method ontology*. It specifies concepts and relations used by the PSM. Another ontology used is the *domain ontology*. It defines the domain in a shared conceptualization. PROTÉGÉ-II now proposes a third ontology, the *application ontology*, which extends the domain ontology with PSMs specific concepts and relations. The application ontology is connected to the method ontology through different types of mapping relations [28]:

- *Renaming mappings*: translate domain-specific into method-specific terms
- *Filtering mappings*: select a subset of domain instances used for the corresponding method concept
- *Class mappings*: provides a function to compute instances of method concepts from application concept definitions rather than instances

Figure 2.4 shows the connection between the ontologies and PSMs.

Other Approaches

Another approach would be MIKE [6], which provides a method covering all steps of developing a KBS. It proposes an engineering framework that integrates formal and semiformal specification techniques and prototyping. Another would be EXPECT [29], which aims to provide a tool for developing KBS independent of the inference structure and PSM of the task. Commet [71] also aims to provide a framework for knowledge modelling. It assumes that a description can be created from three perspectives: tasks, methods, and models.

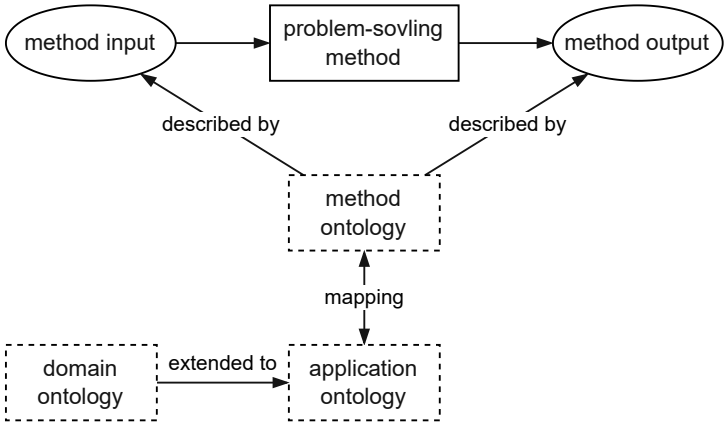


Figure 2.4: Structure of ontologies in PROTÉGÉ-II (adapted from [72])

2.3.3 Ontology Creation Methodologies

This work aims to use ontologies as the core element of the knowledge base. There exist many different approaches how to develop an ontology. Cristani and Cuel provide a survey on these ontology creation methodologies in [16]. They state that the creation of an ontology deals with three problems. First, providing the general terms of the domain to be used to describe classes and relations. Second, ordering these terms into a taxonomy of classes by the ISA relation. Last, explicitly describing the constraints that make the ISA pairs meaningful. This thesis will briefly summarize some of the methodologies for solving these problems.

Noy and McGuinness present the *Ontology Development 101* methodology in [57]. It consists of a simple iterative seven-step guide. The first step is to *determine the domain and scope of the ontology*. Here, questions such as what domain the ontology will cover or what types of questions the ontology should provide answers to, also called competency questions, will be defined. The second step is to *consider reusing existing ontologies*. In this step, literature and the Web are searched for ontologies that might already cover the described domain or parts of it. The next step *enumerates important terms in the ontology*. The goal is to create a list of terms used within the ontology and to describe their meanings and properties. As the fourth step, the *classes and class hierarchy is defined*. Different approaches can be taken when describing the hierarchy. A top-down approach starts with the most general concepts and then specializes. With the bottom-up approach, the opposite is performed. However, a combination of both is also possible. The fifth step is to *define the properties of classes-slots*. Here, all the properties are added that are needed to answer the competency questions. In the next step, the *facets of the slots are defined*. There exist different facets which can describe value type, allowed values, and other features that the value of a slot can take. The final step of this methodology is to *create instances*. In this step, individuals of the classes are created, and the slot values are assigned. The final result is the ontology and the modelled domain.

Another important methodology is *Methontology* presented by Ferández-López et al. in [24]. They define activities that need to be performed when creating an ontology. They developed a flow of the creation process for three different processes: management, technical, and support. This methodology consists of the following phases, as shown in [16]. In the first phase, the *planning* is done. It specifies which tasks need to be performed, when they are performed, and what resources they need. The next phase handles the *specification*. This step aims to identify the ontology's goals and define its purpose and scope. Phase three deals with the *conceptualization*. Here, a conceptual model is formed that describes the problem and its solution. The outcome of this phase includes a glossary of terms containing important concepts, verbs, and properties. Following this phase is the *formalization*, where the conceptual model is transformed into a formal model using specific logic representation systems. In the *integration* phase, existing ontologies are included, and a refinement process is performed. The *implementation* phase then creates a computable ontology from the formal model. Here, the desired target language is selected. After this phase follows the *maintenance* phase. There, modification and inclusion processes are performed on the ontology. In the *acquisition* phase, knowledge sources are searched for and listed. This acquisition can be made through interviewing domain experts or text analysis. The *evaluation* phase deals with the technical judgment of the created ontology to a frame of reference. Lastly, in the *documentation* phase, the necessary information is recorded.

The third methodology presented in this work is the *Toronto Virtual Enterprise (TOVE)* methodology, as shown in [16]. It consists of six steps. The starting point is a *motivating scenario*. It describes a set of problems encountered in an enterprise. In the next step, *informal competency questions* are defined. These questions guide what the ontology should be able to answer. The third step is the *terminology specification*. This step specifies the ontology's objects, attributes, and relations. Next follows the definition of *formal competency questions*, based on the previously specified terminology. The fifth step handles the *axiom specification*. Here, the axioms are specified that define terms and pose constraints on their interpretations. Lastly, *completeness theorems* are added. This evaluation step defines the conditions under which the competency questions are complete.

The three presented methodologies have in common that they all take a specific task as a starting point, which helps choose the domains and categories for a correct representation. In Ontology 101, the ontology's domains and scope are specified. Methontology identifies the ontology's goals and defines its purpose and scope. The TOVE methodology starts with defining a motivating scenario, which describes a specific problem. Cristani and Cuel state that there are two different types of methodologies. Stage-based models are used when the purpose and requirements of the ontology are clear. TOVE is an example of this kind of methodology. Evolving prototype models are used when the environment is dynamic and difficult to understand. Methontology is an example of this type.

The three presented methodologies also differ in the kind of approach they take. TOVE implements a bottom-up approach. Methontology, on the other hand, uses a more

top-down approach. With Ontology 101, both approaches are possible. All three are sophisticated approaches, meaning they assist the knowledge engineer in creating the ontology.

2.4 Related Work

The following section presents some related work to this topic, starting with works focusing on ontology design for RMSs, going over to safety in RMSs, ontology design for safety, and lastly, the use of ontologies in other domains.

2.4.1 Ontology Design for RMSs

Lepuschitz et al. show how a manufacturing system can be reconfigured automatically [50]. They describe an automation agent architecture that manufacturing systems can use for self-reconfiguration. The focus is placed on the low-level layer. The Low-Level Control (LLC) is responsible for controlling the physical components, supervising sensors and actuators, and transferring information to the High-Level Control (HLC). In their reconfiguration infrastructure, they use a reconfiguration application that controls the reconfiguration process of the control application. It ensures that no wrong input is given to the process and that all state changes are done correctly. Between the two applications, services are implemented to handle the communication tasks, such as monitoring events or changing data values. They then tested this infrastructure via two experiments and showed that they could achieve reconfiguration without disturbing the operation (i.e., keeping a pendulum straight up). Further, they then combined this infrastructure with an automation agent. Every component in a manufacturing system is controlled by an automation agent, that has to coordinate its activities to ensure correct functioning. The agent's architecture is split into an HLC and LLC. Using an ontological representation of the low-level functionality in the HLC makes it possible to reason and initiate a reconfiguration of the LLC. Finally, they perform a case study based on this architecture to show that their framework is suitable for self-reconfiguration.

In [68], Seyedamir et al. show an approach to using ontologies in manufacturing systems. For this, they develop an ontology with the help of the ISA-95. This standard was developed by the International Society of Automation (ISA) in the 1990s and is used for integrating enterprise and control systems. The main focus was to reduce the effort required to implement interfaces between them. The authors use this standard to help define the required functions and data flows to integrate enterprise and control systems. The resulting Enterprise Control Ontology (ECO), as it is called in the paper, consists of three sub ontologies. First up is the hierarchy ontology, which is based on parts one and two of the ISA-95. It is split into the role-based hierarchy and physical asset equipment models. This distinction separates the assets used in the manufacturing process based on activity, composition, physical location, and finance-related aspects. The second part is the operation type ontology. It is used to define a set of management operations. The standard's four main management operation activities are production, maintenance,

quality, and inventory. The paper focuses on the production management operation. The final sub-ontology is the resource ontology. It is used to define resources that can provide capabilities for some actions. Such resources are personnel, material, equipment, and process segment, as defined in the standard. Seyedamir et al. then add semantic rules to this solution to enhance the explicit knowledge of the proposed model. Finally, they test their solution by modelling a simple factory line.

2.4.2 Safety in RMSs

In his paper [42], Koch shows an approach to how a safety configuration for a robot application can be created automatically. He states that with the paradigms of Industry 4.0, which call for a dynamic and reconfigurable system, the system can easily run into uncertainties not covered by the safety configuration. Thus, making safety a bottleneck in an RMS. To overcome this problem, he proposes an approach to create a computer-automated safety configuration. The approach consists of three main parts: the PPR-H model, a safety behaviour model, and the safety state machine. The Product-Process-Resource (PPR) model covers information about the existing robot application and, as the name suggests, gives insights on the product, performed processes, and used resources. This model is then extended to the PPR-H model to also provide information about hazards since this is needed to create the safety configuration. Within the safety behaviour model, the safety functions are stored. These safety functions are created during risk reduction using the information about hazards stored in the PPR-H model. The approach's last and central component is the safety program as a safety state machine. The state-of-the-art programming provides a global safety input to output allocation focusing on safety measures. Koch now proposes safety states which are process-oriented. Here, in a specific process step, a safety state is triggered. Each safety state can contain multiple safety functions that implement the safety measures. In his proposal, he makes two assumptions. First, if the safety state is in itself safe, then the process control can trigger it. This means that even though the trigger signal may be unsafe, no unsafe state can be adopted. The second assumption is that an intrinsic safe state can be found for each robot application task.

What kind of challenges are posed concerning safety in RMSs is shown by Koo et al. in [43]. Their paper discusses problems and requirements regarding safety in RMSs. They found four main points of concern. First, a new assessment must be made to identify possible new hazards whenever a system is reconfigured. Second, the reconfigured system's safety measures must be fulfilled. On this notion, their third point of concern is that new measures need to be implemented where needed. Lastly, the availability of safety experts to perform the risk assessment must be considered. They then go on to give possible reasons as to why current standards for assessing and certifying RMSs do not take flexibility and reconfigurability into consideration. One of them would be the highly dynamic changes in RMSs and the possible unknown safety concerns resulting from them. After they describe the current assessment process, they explain which parts of this process can possibly be automated to reduce effort: data collection, data allocation,

risk assessment, documentation, and certification. Using these five process steps, they recognize six challenges for automating the risk assessment. The first challenge is the need for standards describing the safety features of modular machines. Such standards would enable manufacturers to provide the correct information, such as machine features, the direction of movement, or working range, needed to automate the risk assessment process. As a second challenge, they state that a standardized way to interpret existing standards is needed to automate risk assessment. Here, relevant standards for the assessment have to be converted into a set of rules such that the systems can be assessed quantitatively. The third challenge is hazard recognition and classification. For current systems, there exist several analysis methods like Fault Tree Analysis (FTA) or Failure Mode and Effects Analysis (FMEA), which safety engineers perform to identify hazards. New techniques need to be developed for RMSs to automate this process. Challenge four is the interdependency of different modular machines and human involvement. This interdependency means that for a feasible automated risk assessment, the connection between different machines and humans involved in the work process must be considered. The fifth challenge is the accuracy of such a risk assessment and whether the automated certification process will be generally accepted. The last challenge is transitioning from the current state of engineering to RMSs. For this transition to be successful, the new concepts must be mature, cost-efficient, produce quality products, and be safe. They then propose concepts for an automated assessment system. The first concept is partially automated, which requires an operator to assist the process. Then a fully automated concept is presented, where the operator will only start the assessment and confirm the outcome.

2.4.3 Ontology Design for Safety

On the notion of adaptable safety functions, in [10] Brecher et al. show an ontology-based approach for data-management in a CPPS which is then used for adaptable safety functions. CPPSs utilize various data sources since the requirements change for different parts of a system regarding the type, amount, and usage of the available data. To handle these data sources, they propose the application of an ontology. Its main parts are information about the production site, data sources, and facts. For the production site, they use the model given by the IEC 62264–1 [35], which contains a hierarchical representation of an enterprise. The data sources part represents databases, data streams, data servers, and datasets. With this, they cover elements like relational databases, Message Queuing Telemetry Transport (MQTT) messages or OPC UA servers. The last main part of this ontology is the concept of a fact, which is used to describe the data that is provided by the different sources. A so-called Ontology-Based Data Management (OBDM) service was implemented to enable the user to interact with the ontology and the data sources. With it, a user can request data and query the ontology. This service is then used to query data needed for a safety function. The safety logic and a client were written to repeatedly check if the data needed for the safety function had changed. Based on this data, it is then decided if an action concerning the system's safety is needed.

2.4.4 Ontologies in Other Domains

The use of ontologies for automated safety planning in the construction domain is shown by Zhang et al. in [77]. They propose an ontology containing construction safety knowledge, which is then connected to the Building Information Model (BIM) to allow for more significant interaction between safety management and BIM. The ontology consists of three main domain ontology models: construction process model, construction product model, and construction safety model. The construction process model stores information regarding the project's construction plan with its needed resources. Building element information, such as wall or column information, is stored in the construction product model. This model also provides the main interface to the BIM. The construction safety model holds knowledge concerning construction safety, including potential hazards or specifications from regulations. The developed ontology is then evaluated for semantics by interviewing subject experts and an application scenario.

In their paper [53], Melik-Merkumians et al. show how to integrate an ontology-based knowledge base in runtime failure detection for industrial automation systems. Based on the three major lifecycle phases of such systems, namely design, deployment, and runtime phase, they present a domain-specific Engineering Knowledge Base (EKB). One of the main problems with runtime failure detection is the absence of relevant design-time information, such as the system's layout or the manufactured product. The EKB provides a place to hold and access this information during runtime, aiding the detection of failures or unplanned situations. The different components can query the ontology with SPARQL or Semantic Web Rule Language (SWRL), which allows components to derive new facts by reasoning. With the aid of the design-time information, assertions can be defined which are checked during runtime, and a notification is given if any of these assertions are violated. It is further possible to store deployment information to enhance further deployments. To test their EKB, they describe two real-world scenarios where failures occur. The evaluation shows that their approach is feasible to detect failures for which a combination of sensor and design-time information is necessary.

CHAPTER 3

Methodology

The applied methodology for this thesis can be seen in Figure 3.1. It consists of three main phases:

1. *Requirements Analysis*: This phase defines the concepts and functions the knowledge base should provide. First, an application scenario is described to define where and how the knowledge base is used. Based on this application scenario, the different domains covered by the knowledge base are derived. Also starting from the application scenario, requirements in form of competency questions [30] are defined. These two parts then merge together and the questions are matched into the domains to refine missing parts.

2. *Ontology Creation*: In the second phase, the ontologies for the knowledge base are developed. Here, it is differentiated between reusing existing ontologies for specific domains and modelling the missing domains and concepts. Multiple existing ontologies are considered for each of the different domains. These ontologies are rated based on different criteria before the best fitting one is chosen. The existing ontologies are then combined with the newly created ones and relations between them are modelled.

3. *Evaluation*: The last phase is the evaluation, which aims to check if the created knowledge base fulfills the defined requirements of phase one. One element of the evaluation is based on answering the predefined competency questions. The second element is integrating the knowledge base into a reconfigurable safety system and testing if it can provide the required functionality.

Each of the steps of the methodology shown in Figure 3.1 will now be defined in greater detail. The required input, the performed tasks within, and the produced output will be explained for each step.

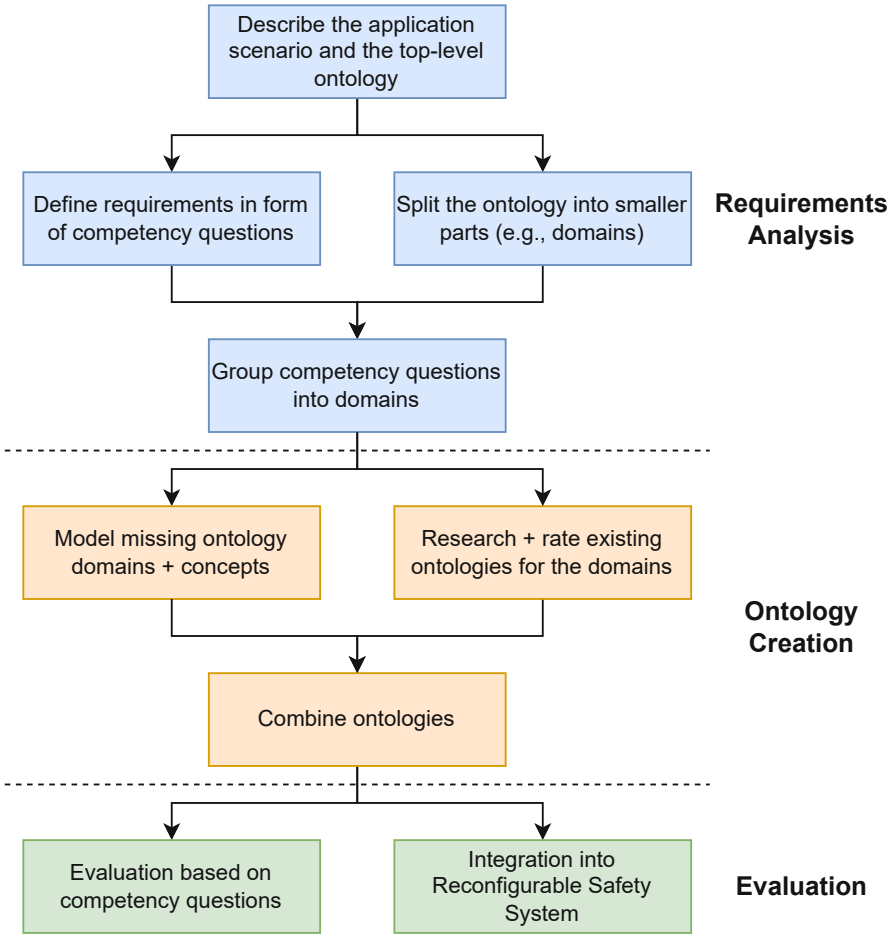


Figure 3.1: Methodology for the knowledge base creation

Describe the application scenario and the top-level ontology is the first step of the requirements analysis. The motivation example presented in Section 1.1 and the described problem in Section 1.2 can be seen as the input for this step. This basic information is needed to know for which specific field the application scenario should be described. The example presented in Section 1.1 already specifies a first use case for the knowledge base. The self-organizing safety system model shown in Figure 1.3 provides information in which context the knowledge base will be used. Thus further specifying restrictions for the application scenario. By using the given information, first, a wider view of where and how the knowledge base can be used is shown. Then more specific use cases are described in more detail, which provides examples of the usage of the knowledge base. These use cases give a base idea for the top-level ontology.

The defined use cases from the previous step are now used as input for the step *Define requirements in form of competency questions*. Here, the competency questions for the knowledge base are defined. With the application scenario in mind, a set of questions is

defined that the knowledge base should provide answers to. In addition to extracting competency questions from relevant literature, an interview with domain experts is performed. A result of these activities is a set of competency questions specifying requirements for the knowledge base.

Parallel to the definition of competency questions, the step *Split the ontology into smaller parts (e.g., domains)* is executed. Again, using the previously described use cases, specific domains that the knowledge base should cover are defined. Here, the main focus is on the safety system and finding useful domains in this context. Risk analysis processes provided by standards are taken into consideration when defining the domains. A result of this step is a set of domains that the knowledge base should cover.

The output of the previous two steps is now combined in the step *Group competency questions into domains*. Here, each of the competency questions will be matched into the domains. Because of the form of the competency questions, one question might correspond to multiple domains. The goal of this grouping process is to identify possible missing domains or questions. Missing concepts will later be added to corresponding sets until no more are found. This step results in a complete list of competency questions and domains matched to each other.

The ontology creation starts with two parallel steps. *Research + rate existing ontologies for the domains* takes the before-defined domains and reviews literature containing ontologies that might cover those domains. An initial selection of ontologies is made by simple Internet research and checking citations of different papers. Furthermore, ontologies provided by domain experts are taken into consideration. For each of the found domains, some ontologies are explained in detail. The core concepts and relations covered by the ontology are presented. These presented ontologies are then rated based on different criteria, such as domain coverage and overhead before making a final selection [26]. The result of this step is a set of ontologies covering specific domains.

Domains not covered by already existing ontologies are created in *Model missing ontology domains + concepts*. The missing concepts are modeled and added for domains that are only partially covered by an ontology. For domains that no ontology sufficiently covers, a new one is created. While performing this creation process, the previously defined competency questions are used as requirements. Resulting from this step are new ontologies covering the missing domains.

The existing and newly created ontologies are then combined in *Combine ontologies*. Here, relations between the different ontologies are created to further enrich the information content of the knowledge base. The goal is to add concepts and relations between ontologies to be able to answer the competency questions. The result of this step is the finalized knowledge base.

This knowledge base is now evaluated in the next two steps. The first evaluation is performed in *Evaluation based on competency questions*. Before the evaluation starts, an application scenario in which the knowledge base should be evaluated is described. Within this scenario, the previously defined competency questions are used to evaluate

the information content of the knowledge base. If any of the questions can not be answered, then another round of modelling missing concepts is performed. This process is done iteratively until all questions can be answered. The evaluation itself is performed expressing competency questions as SPARQL queries.

The *Integration into Reconfigurable Safety System* is the second part of the evaluation. Here, the knowledge base is integrated into the safety system of a production cell to test if it can provide the required functionality. There, the use case of initial commissioning is performed, and the knowledge base should provide the needed configuration for the TSN and OPC UA components. This information is acquired using SPARQL queries.

If both of the previous evaluation steps are completed with positive results, then the knowledge base has proven to fulfill its desired usage as defined by the requirements analysis. Therefore, the created knowledge base is suitable to be used within an RSS.

CHAPTER 4

Requirements Analysis

Before the ontology for the knowledge base can be developed, it is crucial to define where and how it will be used later. The functions it should provide and the information it has to hold need to be defined. For this, a requirements analysis needs to be performed. This analysis is done via two different approaches. The first specifies an application scenario from which the needed domains will be extracted. The second utilizes competency questions, which the knowledge base should be able to answer. As a final step, the found questions are matched with the specified domains to find possible missing elements.

4.1 Application Scenario

The starting point for this requirements analysis is set by describing an application scenario in which the knowledge base can be used. Again, it is possible to make two significant distinctions regarding how and where the knowledge is used based on the development lifecycle. The first would be by integrating the knowledge base into the digital twin of a manufacturing system. By doing so, the provided information can be used within the simulation and help with its analysis before the real system is implemented and deployed. The second would be a manufacturing system, such as a production line. Here, the knowledge can be used during operation to provide current information about the system.

In addition to the use case described in Section 1.1, Etz et al. describe two other use cases for an RSS in [22]. They first describe a plug and produce use case, where an individual component can be added or removed rapidly. The authors provide the example of a handling robot used for machine loading, which is added to an existing manufacturing cell. A scenario in which this might occur would be when the manufacturing cell does not have a robot, or it is deactivated, and the machine operator manually produces individual parts. The robot might also be activated during the night shift, whereas during the day, the work is handled by an operator. These scenarios pose a problem for the safety

system since the robot is equipped with an emergency stop button that must be added to it when it is in use. This addition requires a reconfiguration on two levels. First, the safety function must be extended to include all available safety equipment. Second, the communication system must be reconfigured to ensure timely communication between safety components.

Another use case described in [22] is concerned with moving container-based applications. Virtual machines or containers are used to achieve higher flexibility and scalability in edge or cloud computing. In the context of RMS, this means that fog nodes host various applications used for control processes or communication between cells. An example of this could be the safety logic of a cell handling the safety functions. This safety logic might be moved to a different fog node because the current one is overloaded or crashed. When moving the safety logic, new connections need to be formed, e.g., to the emergency stop button. Since the container holding the safety logic has new addresses, they need to be passed to the safety logic, corresponding to the mapping of relevant ports. The move of such a container also requires changes in the surrounding network, for example, TSN.

4.2 Domains

The goal is now to find what kind of information the knowledge base needs to hold to provide a benefit in the described scenarios. The focus of this thesis is primarily placed on functional safety requirements and the safety system. To be able to describe a safety system, the *System Architecture* is needed. Knowledge about the system's architecture is always required for the simulation or the application in a real system. In both cases, information about the used components is necessary. The *Asset Properties* are closely connected to the architecture and components. They include limits, size, or location of a component. These properties are essential considering the safety system. As mentioned in Section 2.2, establishing a system's limits is the first step for risk analysis and reduction, as specified by the ISO 12100. Therefore, by storing this information, the knowledge base can aid the safety analysis process.

This process also mentions the intended use of components. Information about the usage can be given with the notion of *Capabilities*. It makes it possible to define for each system element what capabilities it possesses, meaning what kind of jobs it is meant to do. This idea of capabilities can also be used to provide information for a person working on or with the system about the purpose of a machine.

On this thought, it is essential to know which person is allowed to perform which task regarding the system. This restriction is where *Management* comes into play. It is important to know which worker has which specific role. If changes are made within the system, it is necessary to know who is allowed to make these changes. Again, in the context of safety, this is essential because only a safety engineer can declare if by changing a system in a certain way, it will still be safe.

A *Network* is needed to enable the different components to communicate with each other or allow workers to retrieve internal information. The network consists of hardware elements, such as switches and links, and software elements, such as configurations and routing information. Assigning these elements to the system architecture and asset properties domain is possible. Nevertheless, in this case, it is feasible to distinguish between elements actively contributing to the manufacturing process and the communication components since it facilitates the reusability of the ontologies covering each of these domains.

In the context of network communication, TSN and OPC UA are two important standards, which are gaining acceptance in the industry [23]. TSN enables deterministic real-time and highly reliable communication provided by a fault-tolerance mechanism. The concepts of TSN are represented within the *Network* domain. OPC UA allows for cross-platform communication and easy data exchange between components. These are the reasons why OPC UA is used in this work. The manufacturing system will be extended with OPC UA concepts to enable the data exchange and communication between system parts. An *Information Model* is needed to represent these concepts.

As mentioned in Section 2.2, different *standards* apply when it comes to the safety of machinery. The knowledge they provided can be used to check whether a given configuration of the safety system is valid or even generate a new configuration. This additional information would shorten the deployment time of new safety configurations as they no longer need to be created or checked manually by a safety engineer. However, modelling these standards into concepts and relations to form an ontology is a difficult and time-consuming task and would go beyond the scope of this thesis. Nevertheless, it will be kept in mind while modelling the other domains to allow for a future knowledge base extension.

Figure 4.1 gives a summarized view of the described domains. It shows the sub-ontologies needed for a knowledge base for an RSS. The green domains will be covered in this work, the grey one will be covered in future work.

4.3 Competency Questions

Based on the before defined application scenario and to further determine the scope of the knowledge base, a list of competency questions [30] is defined. These are questions that the knowledge base should be able to answer. A benefit of using competency questions is that they can be used in the evaluation to check if the knowledge base contains enough information. Since the competency questions are used as a sketch to define a frame of the needed information, they do not need to be exhaustive. [57]

As mentioned in Chapter 3, these questions were obtained in two ways. One part was extracted from relevant literature. While reading related papers, it became apparent what questions the knowledge base should be able to answer in regards to reconfigurability and functional safety. The others were gathered while attending a workshop and speaking to domain experts, which gave requirements to the knowledge base and stated the expected

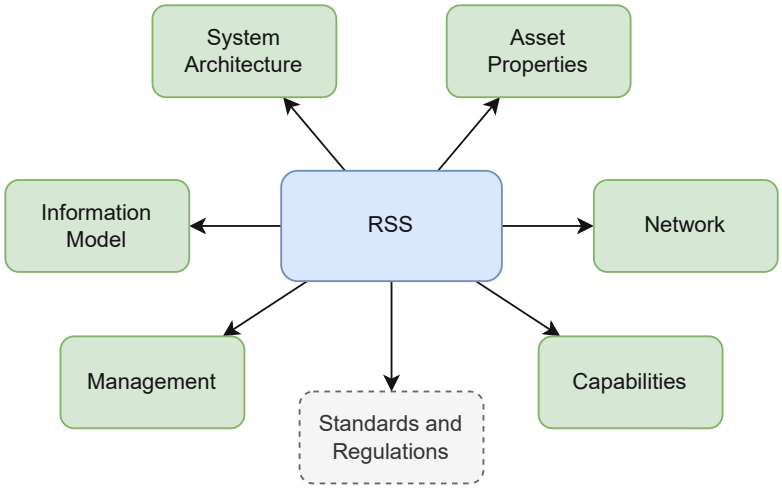


Figure 4.1: Domains of an RSS

information it should provide. Because of the modular structure of the knowledge base, it can easily extended in case additional application scenarios yield further competency questions.

For the domain of reconfigurable safety systems, the found competency questions are:

1. What is the current safety configuration for this system?
2. When should the new safety configuration be deployed?
3. Is the configuration validated by a safety engineer?
4. Is the configuration currently active?
5. Has there been an error while deploying the configuration?
6. Is a new configuration available?
7. How big is the distance between two components?
8. What are the speed limits of a component?
9. What are the components of the manufacturing system?
10. What are the components of the safety system?
11. What are the components of the network?
12. Does a component have a specific capability?
13. Does a machine require human interaction?

14. Where is a component located?
15. What are the properties (MAC, IP, ...) of a component?
16. What SIL does a component have?

4.4 Grouping of Competency Questions Into Domains

As stated in the methodology, the competency questions are now grouped into the identified domains. The goal of this grouping process is to further define the scope of the ontology. If a competency question does not match into any of the identified domains, then either a domain is missing or the competency question might not fit in this context. Conversely, if a domain is not matched with any competency question, then either a key domain was overlooked at the competency question definition, or the domain is not needed. Because of the way that the questions are formulated and the domains are structured, one question may require information from different domains to be answered.

Competency Question	System Architecture	Asset Properties	Network	Capabilities	Management	Information Model
1.			x			x
2.			x			
3.			x		x	
4.			x			
5.			x			
6.			x			
7.		x				
8.		x				
9.	x					
10.	x					
11.			x			
12.				x		
13.				x	x	
14.		x				
15.		x	x			x
16.		x				

Table 4.1: Matching of competency questions into domains

CHAPTER 5

Ontology Creation

The following chapter explains the structure of the created ontologies and the top-level ontology. Different existing ontologies covering the specified domains will be discussed, and it will be decided which are used for this application. After this, the missing ontologies and concepts will be modelled. Finally, all the ontologies will be combined within the top-level ontology, and properties that connect multiple domains will be defined.

5.1 Existing Ontologies

The next step in the knowledge base creation is to check for already existing ontologies. The main reference point for finding these ontologies is based on the before defined domains. Since there exist multiple ontologies for each domain, an initial selection was made based on relevance and establishment of ontologies. Overall, around 70 papers were taken into consideration. A lot of these papers were only very little relevant for this work. They mostly touched topics, such as relevant domains or the usage of ontologies in these domains, very briefly. Another elimination criterion for some papers was, that they focused very precisely on a specific application of an ontology. Therefore, they were not studied in greater detail. To further narrow down the selection, it was checked how often and in which context an ontology was referenced in other papers. In addition to this selection, ontologies recommended by domain experts were taken into consideration. These criteria allowed to narrow down the set of suitable ontologies. Table 5.1 reflects this data for some appropriate ontologies. In the following sections, these ontologies are explained in more detail.

ECO

As already mentioned in Section 2.4.1, Seyedamir et al. present the Enterprise Control Ontology (ECO) in [68]. The ontology is developed with the help of the ISA-95 standard. It consists of three sub-ontologies, which are combined to the ECO.

Ontology	Citations	Recommendation
ECO	11	x
CORA	201	
AMLO	9	x
Ontology for IT Services	23	
Network Ontology for Computer Network Management	3	
Organization Ontology	—	x
Upper ontology for manufacturing service description	102	
MaRCO	100	
OPC UA NodeSet Ontology	19	

Table 5.1: Criteria for selection of existing ontologies

The first sub-ontology, shown in Figure 5.1, describes different hierarchy models, their object models, and attributes following Parts 1 and 2 of the ISA-95 standard. *Role Hierarchy* and a *Asset Hierarchy* are modelled to distinguish the assets of an enterprise. The assets can be distinguished based on activity, composition, physical location, or finance-related aspects. Elements of both hierarchies may correspond to each other. This correspondence allows for an asset to be modelled from both viewpoints. The role hierarchy is further refined and contains the concepts *Enterprise*, *Site*, *Area*, *WorkCenter*, and *WorkUnit*. The object property *contains* specifies that *Role Hierarchy* or subclasses of it may be contained in themselves.

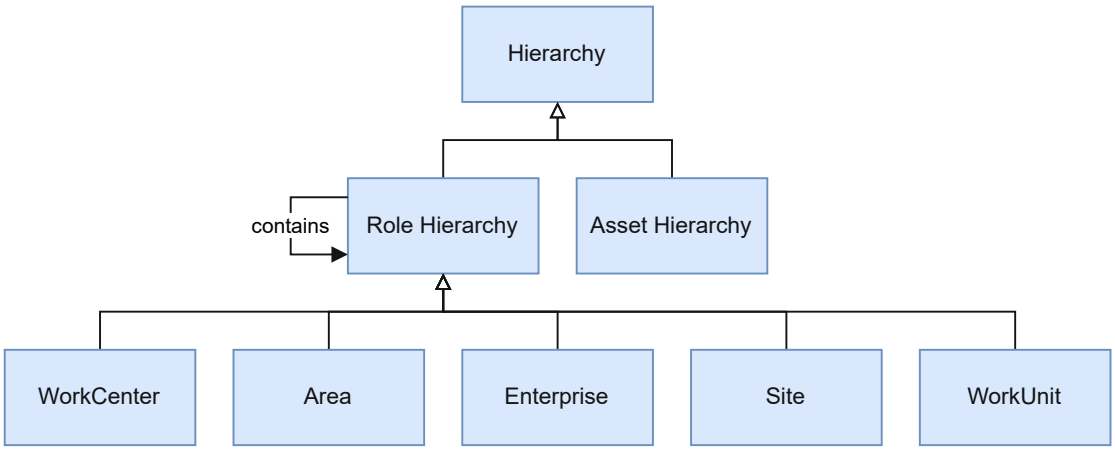


Figure 5.1: Structure of the hierarchy ontology

Figure 5.2 shows the second sub-ontology, which is based on the MES layer. In this ontology, different generic management operations can be defined. They follow a basic request and response cycle that starts with the request, which is then transformed into a work schedule before the necessary resources are dispatched in the correct order for the operation, followed by the execution and finally, the recording and providing of the data as part of the response. The standard defines four main operation activities: *Production*, *Maintenance*, *Quality*, and *Inventory*. However, there are others, such as security and information. Figure 5.2 does not show all subclasses of *Production*. The ones depicted show the object properties of this ontology. It connects *WorkPerformance* to *WorkSchedule* via the *CorrespondsToWorkSchedule* relation. Further, *JobOrder* is connected to *WorkMaster* using the *referencesWorkMaster* property.

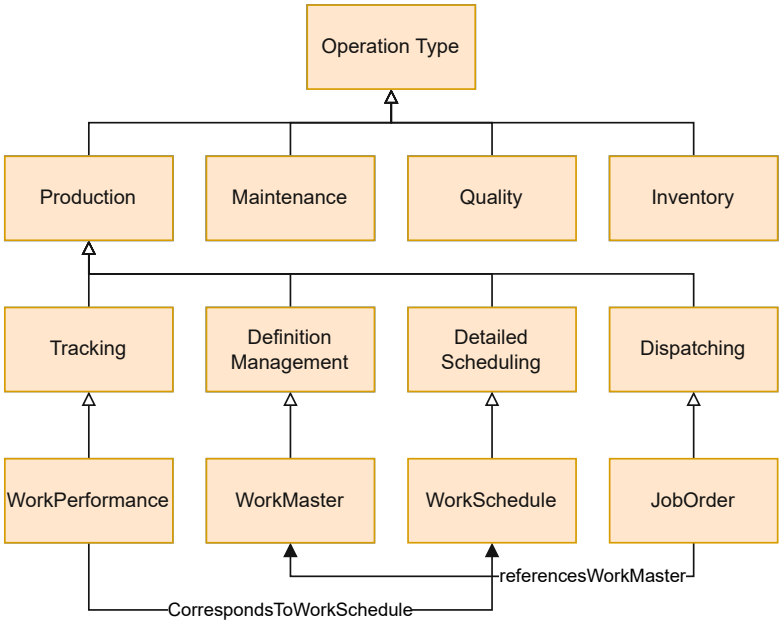


Figure 5.2: Structure of the operation type ontology

The third sub-ontology contains the resources, defined by the standard as entities providing capabilities for activities or processes. Figure 5.3 shows the main classes of this ontology. They are split into *Personnel*, *Material*, *Equipment*, and *ProcessSegment*. This ontology is an intermediary between the hierarchy and operation type ontology. Therefore, a mapping for the communication between them must be added when necessary. The object properties connect the *ProcessSegment* to the other classes. The relations *RequiresMaterialDefinition*, *RequiresPersonnel* and *RequiresPhysicalAsset* connect to their respective classes.

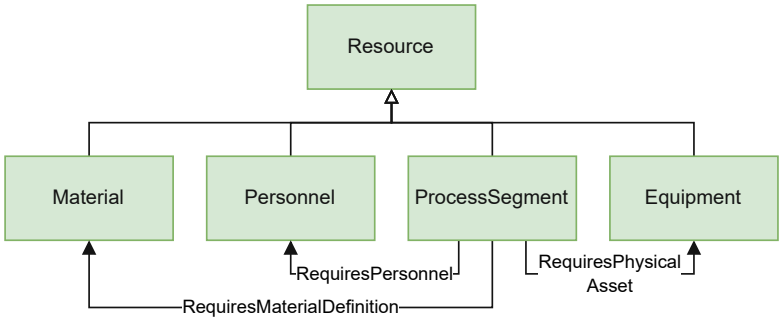


Figure 5.3: Structure of the resource ontology

CORA

The Core Ontology for Robotics and Automation (CORA) [64] was created by the Ontologies for Robotics and Automation (ORA) working group. Its main goal is to provide an ontology that contains commonly used terms in robotic and automation. Figure 5.4 gives an overview of the CORA.

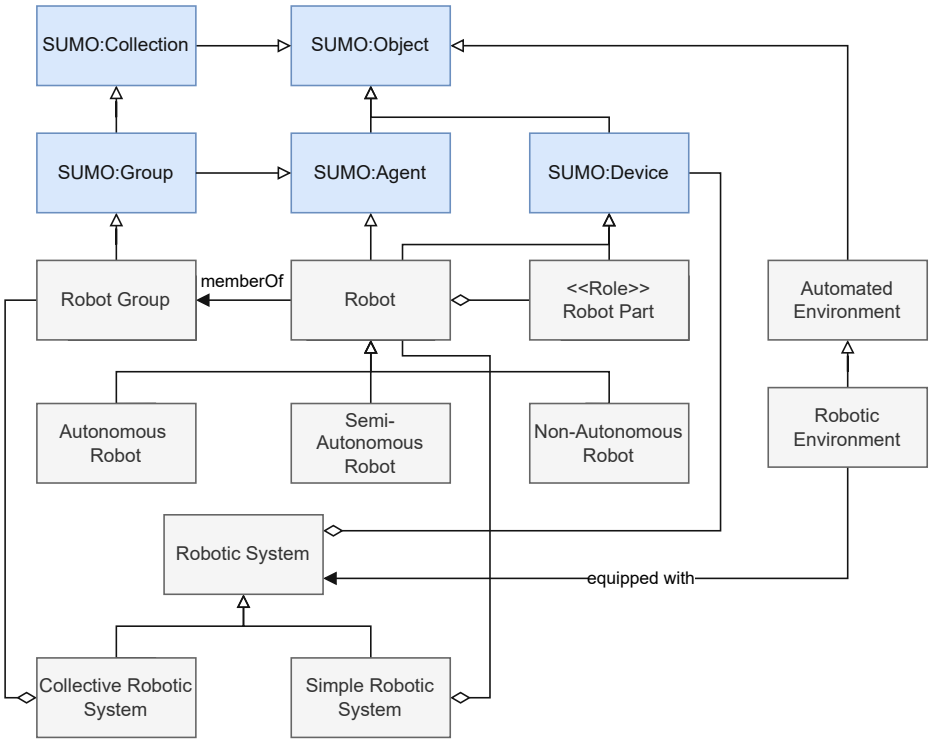


Figure 5.4: Structure of the CORA (adapted from [64])

The central concept in this ontology is the *Robot*. To further help define the domain, other entities are defined which surround the robot, and concepts from the Suggested

Upper Merged Ontology (SUMO) are used. They are classified by their complexity. The main four entities of the ontology are *Device*, *Robot*, *Robot Group*, and *Robotic System*. A *Device* is used as an instrument within a specific process subclass, such as manufactured tools or machinery. Examples of a *Device* are a hammer, sensor, actuator, or transistor. A *Robot* is a *Device* because it is a manufactured instrument to perform some task. However, a robot can also have multiple devices, which are then classified as *Robot Parts*. Prestes et al. specify in their paper that the concept of a *Robot Part* is a domain of its own and will be developed in a sub-ontology.

Nevertheless, they abstract the notion of *Robot Part* as a role to provide a way to classify a *Device* as a *Robot Part*. A role is a concept that some instances might assume dynamically based on the context. For example, a power supply can be a *Device* on its own but can become a *Robot Part* when it is connected to a robot.

A robot is also an *Agent* as defined by the SUMO. An agent can act on its own and produce a change in the world [64]. Since a robot can perform tasks by acting on the environment or itself, it can be considered as an agent. Three significant distinctions can be made depending on how autonomous a robot is, meaning how many inputs it requires from an operator to perform its task. The paper defines autonomous, semi-autonomous, and non-autonomous robots. There are some difficulties in defining where to draw the line between non-autonomous robots and controlled machinery. The details concerning this distinction can be found in the paper [64].

A robot can also be part of a *Robot Group*. An example of such a robot group would be a group of soldering robots in a factory. A *Robotic System* consists of robots and devices intended to help the robots with their tasks. In the example of soldering robots working on a part, the helping devices could be movement sensors or conveyor belts. Such a robotic system is deployed within a *Robotic Environment*.

AMLO

In [46], Kovalenko et al. present an AutomationML Ontology (AMLO). It covers the Computer Aided Engineering Exchange (CAEX) part of the AutomationML standard. With this standard, it is possible to model complete systems, starting from single automation components to entire complex production systems. It enables the representation of different aspects of these systems, such as geometry, kinematics, and topology. Figure 5.5 gives an overview of the AMLO.

The core element of the ontology is the *CAEXFile*. It corresponds to the AutomationML document. The metadata related to this document, such as version, name, and release, can be stored using the *AdditionalInformation* class.

The next important concept are so-called container classes, which consist of *InterfaceClassLib*, *RoleClassLib*, *SystemUnitClassLib*, and *InstanceHierarchy*. The purpose of these concepts is to group the following classes: *InterfaceClass*, *RoleClass*, *SystemUnitClass*, and *InternalElement*.

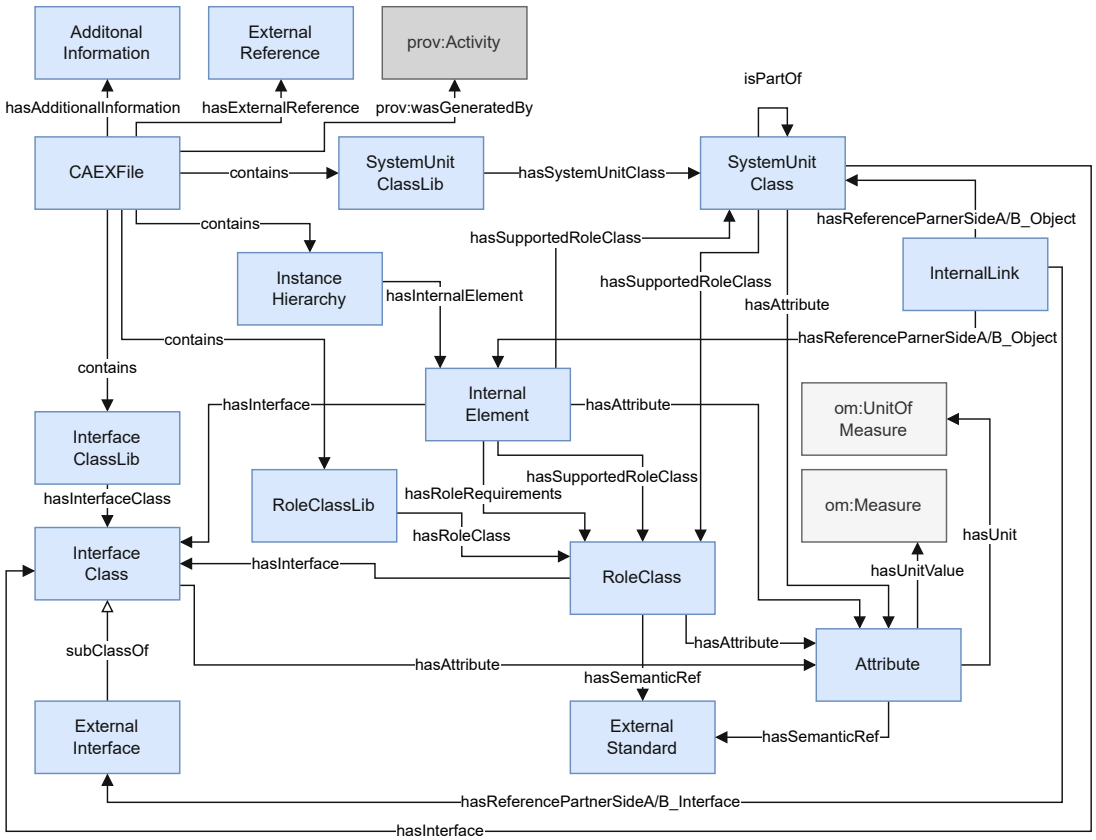


Figure 5.5: Structure of the AMLO (adapted from [46])

The *InterfaceClass* represents relations between topology elements or references to external information sources. A *RoleClass* defines vendor-independent functionalities, which equipment elements can provide. With these classes, it is possible to assign semantic functionality to *SystemUnitClass* and *InternalElement* instances. They describe the functional capabilities of elements. The *SystemUnitClass* defines the realization of a physical or logical object specific to a vendor. These classes can contain instances of themselves, which is shown with the *isPartOf* relation. The *InternalElement* describes a piece of specific equipment used within a project. They can again contain instances of themselves. Further, it can be specified what *SystemUnitClass* is instantiated by a specific internal element. To define properties of a CPPS, the ontology contains the *Attribute* class. Such attributes can be an element's length, speed, or force. Some concepts from the Ontology of Units of Measurements are imported to define the unit for an attribute. Finally, the *InternalLink* class represents a connection between two constructs.

Ontology for IT Services

In [25], Freitas et al. present an ontology for IT Services. The ontology covers aspects from service at different abstraction levels to infrastructure management services. It consists of five main packages: *Services*, *Delivery*, *Support*, *Enterprise Architecture*, and *Participants*. Figure 5.6 gives an overview of this ontology.

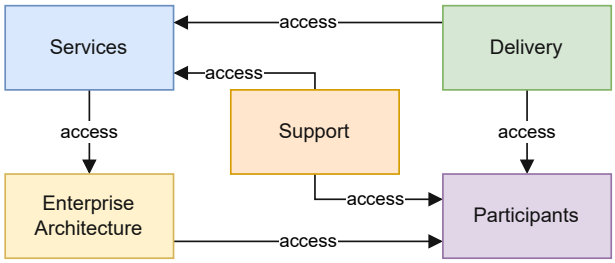


Figure 5.6: Structure of the ontology for IT services (adapted from [25])

The structure of the service package is shown in Figure 5.7. *Services* can be associated with product deployments. Therefore, the authors generalized the concept *Deliverable*. This concept includes both *Product* and *Service*. Deliverables can be hierarchically composed and are further refined by assigning them a type of the *TypeOfDelivery* class with the *hasType* relation. The ontology also contains a *DeliverableCatalog*, which specifies services from which the deliverables can be chosen. The deliverables can have a set of characteristics, defined by *CharacteristicOfService*, such as performance or availability. An *Indicator* can quantitatively evaluate these characteristics. A *Service Level Agreement (SLA)* defines admissible values for these indicators, which are then compared with elements of the *Observation* class.

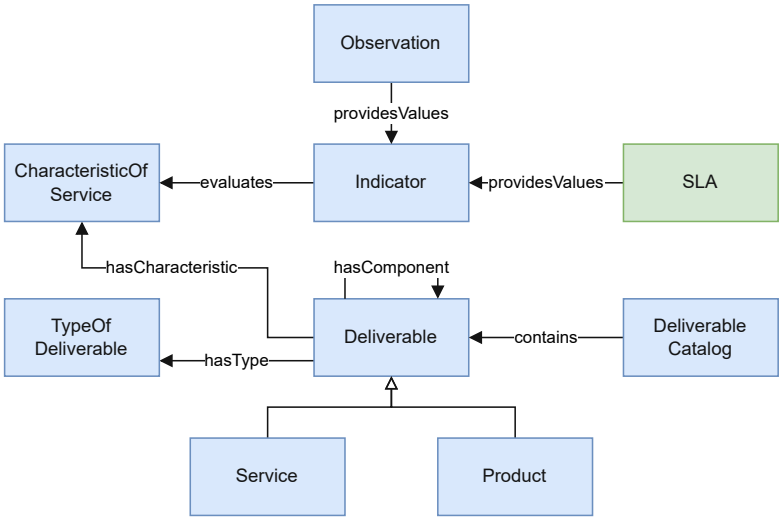


Figure 5.7: Services package (adapted from [25])

These SLAs are contained in the *Delivery* package, the structure of which is shown in Figure 5.8. It contains the class *TemplateSLA*, which can then be used to generate specific ones. Such a specific SLA is called a *ContractualizedSLA*. With this class, a *Contract* between a *Customer* and *Provider* is established. Each SLA contains services for which the client or the provider must take responsibility. This is represented by the *Responsibility* class and the *hasResponsibility* object property.

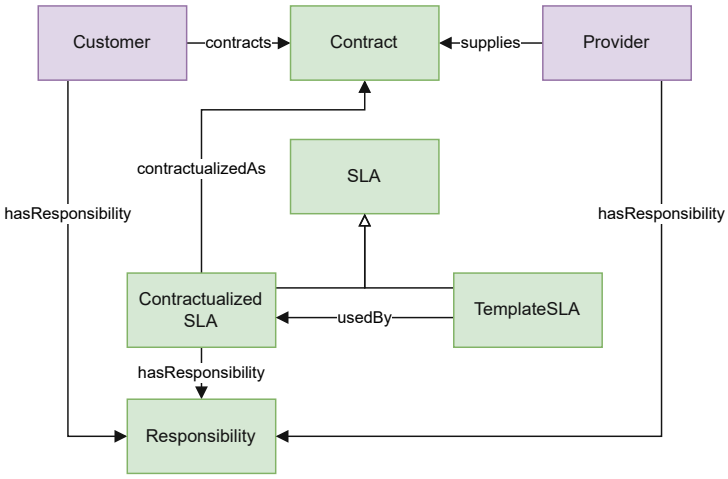


Figure 5.8: Delivery package (adapted from [25])

The *Participants* package contains the two main concepts: *Person* and *Organization*, as shown in Figure 5.9. Both concepts can be hierarchically structured to depict management and organizational structures. A person can be an *EndUser* or a *Worker* and be associated with an organization. An organization can be a *Provider* or a *Customer*. A worker can further be *assignedTo* a *ServiceAccessPoint*.

The organizational goals are defined within the *EnterpriseArchitecture* package. One or more instances of the class *Process* need to be implemented to achieve a *Goal*. Processes can be hierarchically structured and can have interdependencies. A process *consumes* some deliverables and produces others as output. The paper states that this package is still incomplete and will be reworked in the future.

The central concept of the *Support* package is the *ServiceAccessPoint* (e.g., help-desk, ticket system). The service access point has relations to the *Deliverable* and *Action* class. An action, such as incident reporting or assistance queries, can be generated by an end-user and might be prioritized by a worker.

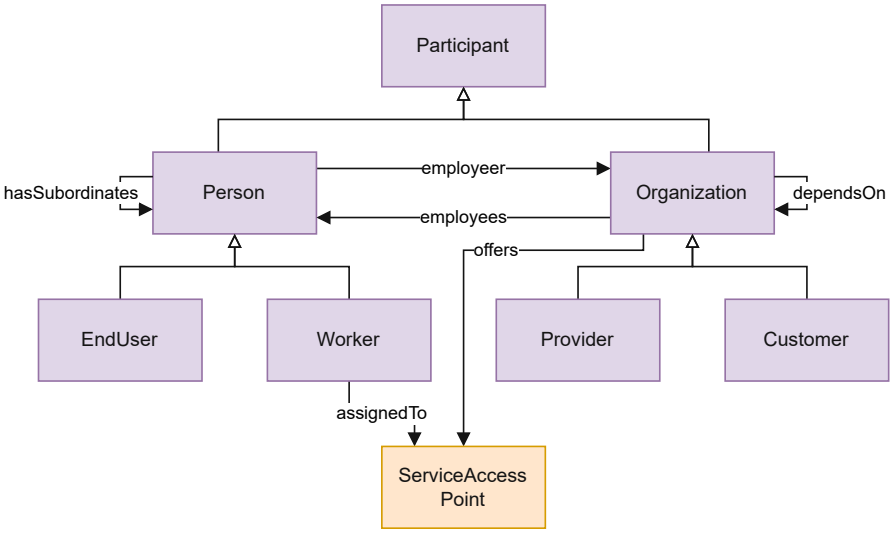


Figure 5.9: Participants package (adapted from [25])

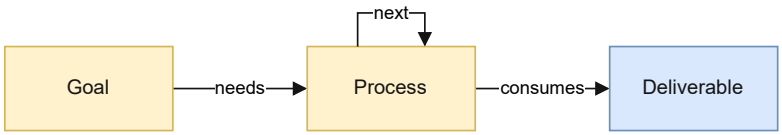


Figure 5.10: EnterpriseArchitecture package (adapted from [25])

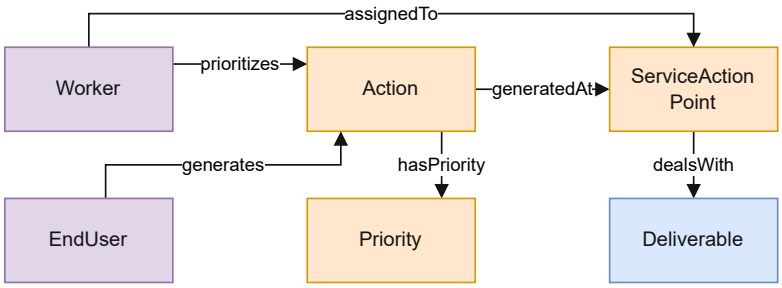


Figure 5.11: Support package (adapted from [25])

Network Ontology for Computer Network Management

In [19], De Paola et al. present an ontology for computer network management. This ontology aims to cover the aspects of the network domain necessary for monitoring and controlling purposes. With the knowledge stored in the ontology, it should be possible to perform tasks such as fault diagnosis and recovery and plan actions to improve the quality of service. Because this ontology is quite extensive, this thesis will only summarize the concepts and relations. For a full explanation of the ontology, see [19].

The *NetEntity* class represents the communication infrastructure of a network. The subclasses and relations are shown in Figure 5.12. A distinction between hardware and software entities is made. The concept of a *Node*, interface (*IFace* class), and *Link* belong to the hardware entities. Examples for software entities are a *RoutingTable* or a *Queue*. The paper defines relations between the classes to represent the infrastructure properly. With the *hasIFace* and *belongsToNode* relation it is defined which interface belongs to which node. The *hasNeighbor* relation is implemented to show that two nodes are neighbors. Lastly, a connection between software and hardware entities is made with *hasRoutingTable*, which defines the routing information for a node, and *hasInQueue/hasOutQueue*, which defines the queues for an interface.

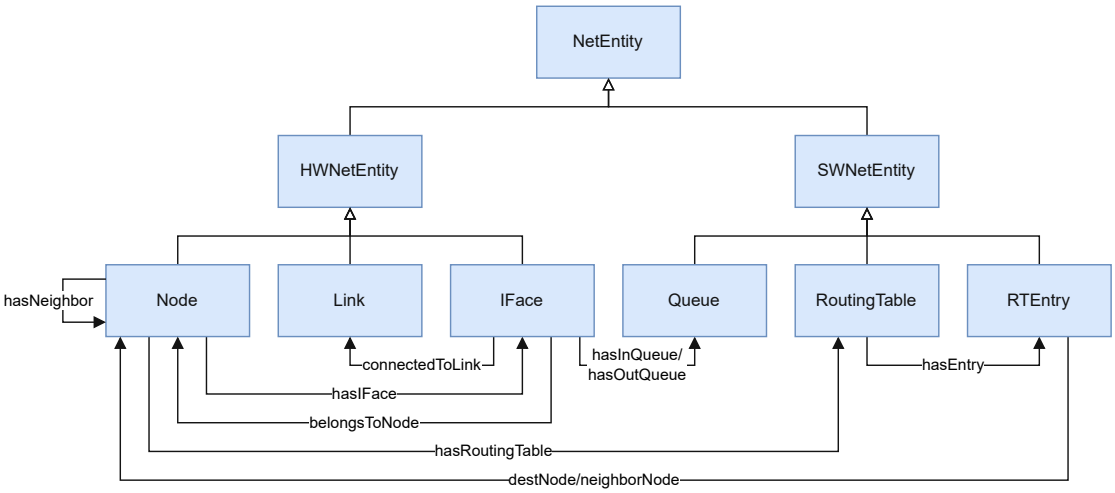


Figure 5.12: Communication infrastructure (adapted from [19])

The *TrafficEntity* class describes the traffic within a network. Figure 5.13 shows the main concepts to define traffic and shows some relations to other classes. *TrafficEntity* contains two subclasses: *Flow* and *Datagram*. The *Flow* class uses a higher abstraction to define the traffic. A flow can contain multiple datagrams. The *Routing* concept is used to define how the routing in the network functions. Depending on this routing, a flow is connected to a *Routing* via the *dependsOnRouting* object property. With the *Demand* concept, the hypothetical traffic load in the network is represented if no resource restrictions exist.

Further important for the network is the state it is currently in. For this purpose, the *Event* class is used. It can be used to define different events that can happen in the network. The authors define three subclasses of events, shown in Figure 5.14. First is the lost packet event, for when the router discards a packet. The second is the state change event, which indicates a status change in a network entity, such as a router powering down. Last, the traffic event describes the traffic data flowing through the network. These events are connected to a *NetEntity*, like a link or node, to associate them.

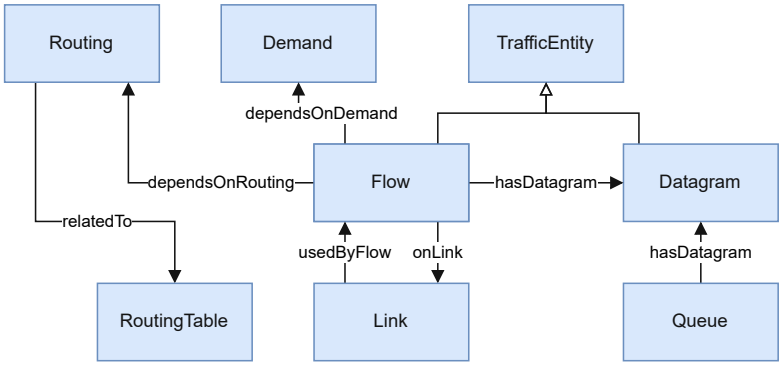


Figure 5.13: Traffic information (adapted from [19])

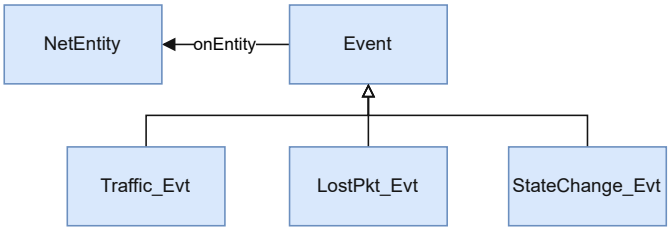


Figure 5.14: Events (adapted from [19])

The *Abnomality* class is used to describe a misfunctioning state of the network. It is refined to describe concepts such as *NetDisconnection*, *Congestion*, *RTCorruption*, and *Loops*. The structure is shown in Figure 5.15. Compared to events, which describe an instantaneous manifestation of the change, an abnormality provides more global information.

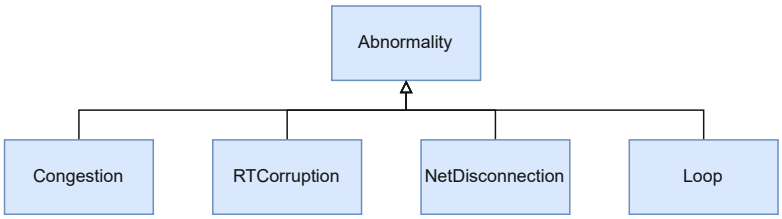


Figure 5.15: Abnormalities (adapted from [19])

The last part of this ontology is concerned with actions and tools for the execution of management tasks. *Management Tools* perform management tasks within the network. *Sensor* is a subclass of them and can be used to capture events, represented by the *catchesEvent* relation. Another subclass would be *Capsule*, which represents active packets containing code to be executed within the network nodes. The ontology further contains the class *Actor*, which represents managing agents that play an active role. Such actors are reasoners and programmable local agents. The *Database* class is used to store

information. It contains two subclasses, *LocalDatabase* and *GlobalDatabase*. As stated by the name, the former is used more locally, e.g., by sensors to store their information, while the latter is used globally by different agents. *Action* is used by actors and represents their performed effects on the network. Actions are used within management tasks. Lastly, the *TrafficStatistic* class represents the traffic load and distribution throughout the network.

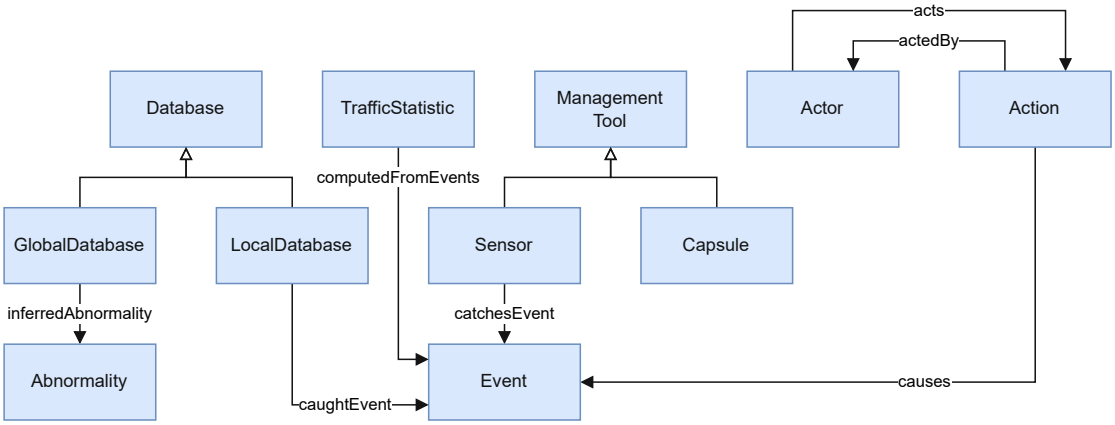


Figure 5.16: Management tools and actions (adapted from [19])

Organization Ontology

In [18], Reynolds describes a core ontology for organizational structures. It represents the main concepts needed to describe organizations and relations within them. Figure 5.17 gives an overview of the main components of this ontology.

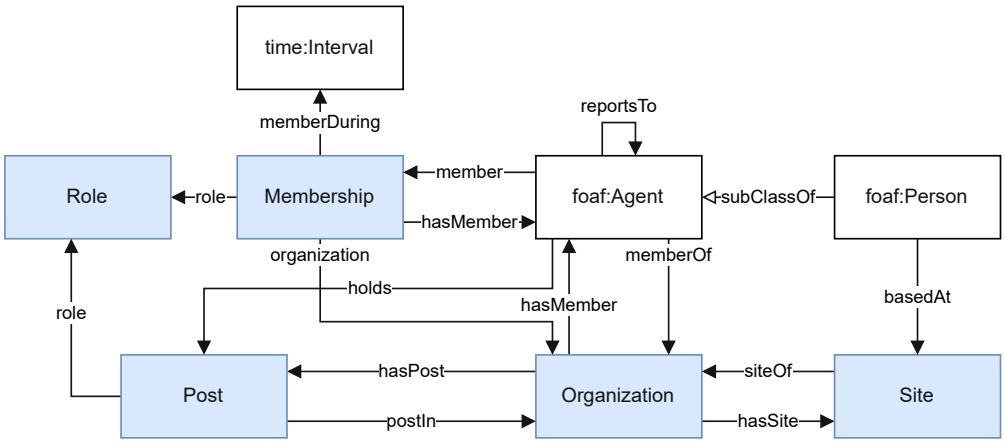


Figure 5.17: Structure of the Organization Ontology (adapted from [18])

The core class of this ontology is *Organization*. It is intended to represent a collection of people organized together in some structure. The ontology further contains subclasses of

organization to help classify them better. The subclass *OrganizationalUnit* represents departments within an organization, such as the IT department. A *FormalOrganization* indicates an organization recognized in legal jurisdiction and has rights and responsibilities. The last subclass, *OrganizationalCollaboration*, represents a collaboration between organizations.

With the *Role* class, a specific organizational role can be assigned to a *Person* or an *Agent*. To show an agent's affiliation to an organization, the *Membership* class is used. It uses a time interval to establish the duration of the membership. A *Post* is a position within the organization that exists independently of a person filling it. Multiple persons can fill a post at the same time. The *Site* class represents an office or any other location where the organization is located. *Person* is related to *Site* with the *basedAt* relation. An organization can have multiple sites.

MaRCO

In [40], Järvenpää et al. developed an ontology for describing the capabilities of manufacturing resources. The overall goal of this ontology was to support automatic matchmaking between resource capabilities and product requirements. The analysis showed that four distributed ontologies were needed to allow for matchmaking, as shown in Figure 5.18. The *Process Taxonomy Model* defines the hierarchical categorization of different manufacturing processes, e.g., "milling" is classified as "machining" and can be further classified as "material removing". The *Product Model* holds product characteristics and manufacturing requirements. Within the *Capability Model*, the capability names, parameters, and relations between them are specified. Lastly, the *Resource Model* defines the resources and the system composed of them. The paper focuses on the latter two models and is thus referred to as Manufacturing Resource Capability Ontology (MaRCO).

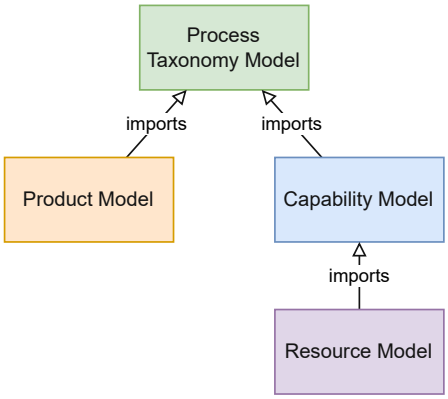


Figure 5.18: Distributed ontologies of MaRCO (adapted from [40])

Figure 5.19 shows the main classes and relations of the *Capability Model*. The main concept is *Capability*, with its two subclasses *SimpleCapability* and *CombinedCapability*. They are used to define the functionality of a resource. Capabilities can relate to each other via the

hasInputCapability relation. Most of the parameters defining a capability are stored using datatype properties. The class *CapabilityParameterAdditional* is introduced for the ones that cannot adequately be represented this way. They are then referenced with object properties. An advantage of this representation is the reusability of parameters. The classes *ItemSize*, *MovementRange*, *ShapeAndSizeDefinition* and *BasicResourceInformation* are used to refine the additional capability parameters. *ItemSize* stores the size and mass of elements. It has a relation with capabilities via the *hasItemSize_min/max* object property. *MovementRange* defines a capability's movement range, such as linear or rotational. This relation is shown by the *hasRotationalMovementRange* and *hasLinearMovementRange*. With the *ShapeAndSizeDefinition* class, the shape and size on a coarse level can be defined. Subclasses of this would be *BoxShape* or *CylinderShape*. *BasicResourceInformation* stores a collection of basic information about a resource. The *Workspace* class is used to define the parameters of the workspace and store details about it.

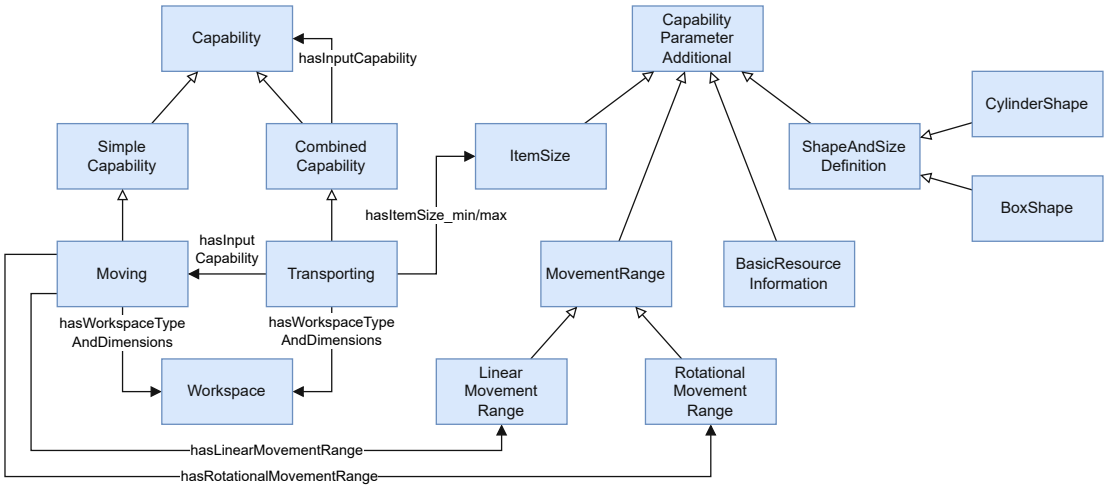


Figure 5.19: Capability model (adapted from [40])

Figure 5.20 shows the main classes and relations of the *Resource Model*. The core concept is *Resource* and is used as the parent for all resource-related classes. These subclasses include *FactoryUnit* and *Device* as shown in Figure 5.20, but also the classes *Human*, *RawMaterial* and *Software* not shown in the figure. The *FactoryUnit* class is used to represent the physical place where the operation takes place. To further specify the physical location, the subclasses *Site*, *Area*, *Line*, *Cell* and *Station* are defined. These subclasses are related via object properties to define the physical location, as shown in Figure 5.20. The *FactoryUnit* is related to a *Resource* via the *hasResource* object property. The *Device* class is the parent to all device-related classes. Devices include machines, equipment, and tools. The main distinction for devices is made with the three subclasses *DeviceBlueprint*, *IndividualDevice*, and *DeviceCombination*. A *DeviceBlueprint* contains catalogue information about devices and has multiple subclasses to specify the devices further. It is related to the *Capability* class by the *hasCapability* object property. An *IndividualDevice* represents the actual device present on the factory floor. It stores life

cycle information and updates the device's capability properties. This update is modelled with the *hasCapabilityUpdated* relation. The *hasDeviceBlueprint* relation connects an individual device to its blueprint. Lastly, the *DeviceCombination* class represents a combination of multiple devices.

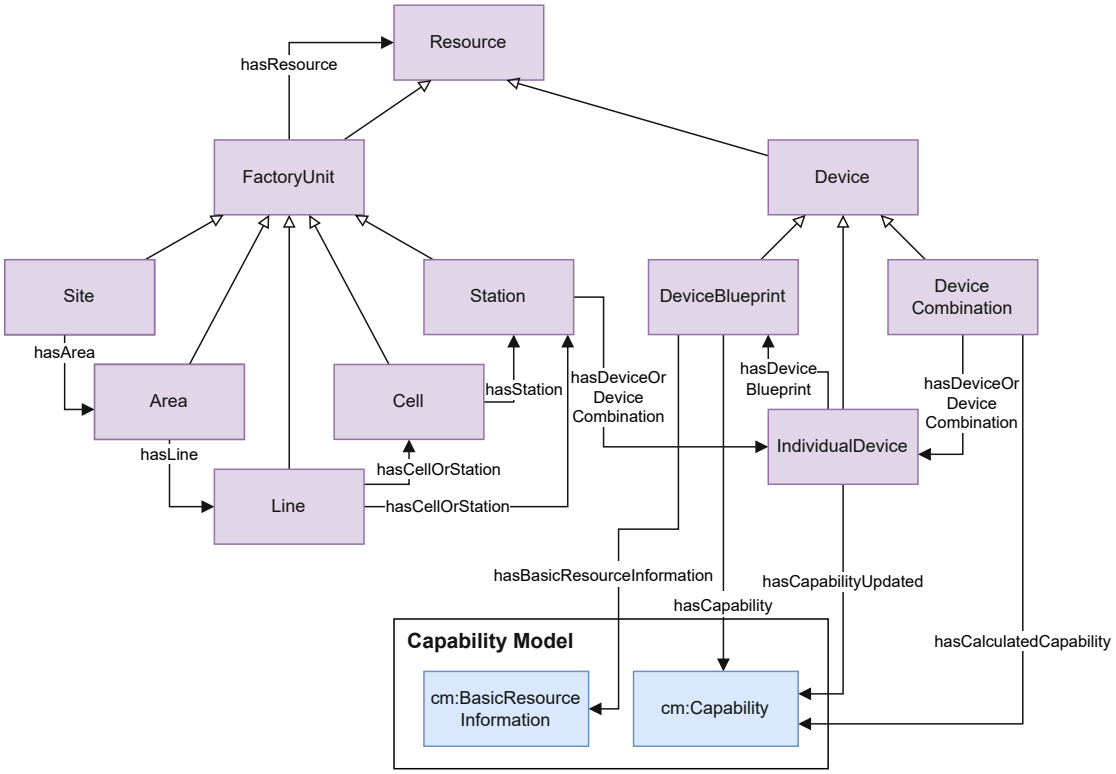


Figure 5.20: Resource model (adapted from [40])

Upper Ontology for Manufacturing Service Description

Ameri and Dutta introduce a Manufacturing Service Description Language (MSDL) as an ontology for the representation of manufacturing services in [5]. The ontology covers the core building blocks to describe a broad spectrum of services. Figure 5.21 shows the main concepts of this upper ontology for manufacturing service description.

Supplier is the central class of this ontology. This class is required since a service can not exist independently and needs a supplier to be delivered through. Based on the field of expertise of a supplier, different object properties, such as *hasCustomerFocus*, *hasProductFocus*, and *hasIndustryFocus*, can be set. These properties connect the *Supplier* class with *Customer*, *Product*, and *Industry*, respectively. *Supplier* and *Customer* are subclasses of *Actor*, which is imported from the OWL-S ontology [17].

For the following classes, the authors use *Mfg* as an abbreviation for *Manufacturing*. The *MfgService* class represents manufacturing services. It is connected to exactly one

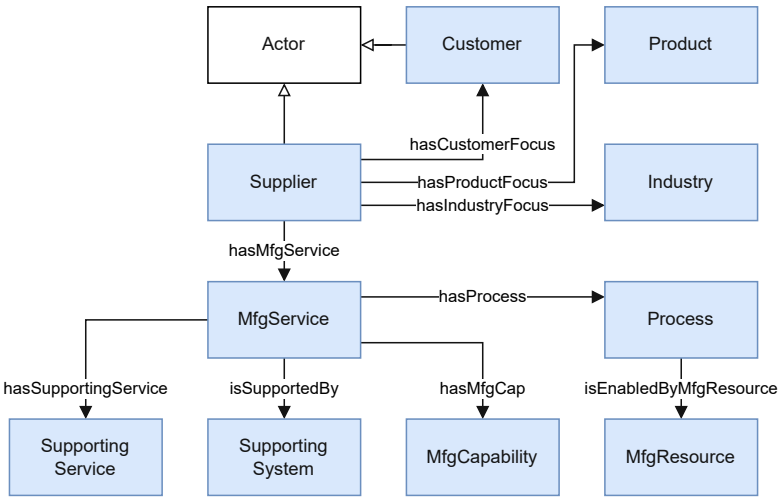


Figure 5.21: Structure of the upper ontology for manufacturing service description (adapted from [5])

Process by the *hasProcess* relation. The service is characterized by multiple capabilities, represented by the *MfgCapability* class. A service needs resources such as workstations or machines to be enabled. These resources, defined by *MfgResource*, are connected to the *Process* by the *isEnabledByMfgResource* object property. A service can also have supporting systems, like a material handling system. This relation between a *SupportingSystem* and *MfgService* is shown by the *isSupportedBy* object property. The relation *hasSupportingService* connects a service to some *SupportingService*. These services may include CAD modelling or shipping.

MSDL does not contain detailed descriptions of capabilities. A reason for this is that different vendors might describe the same capability differently. The authors, therefore, used an extensible and modular approach when creating this ontology. This approach allows to extend the ontology as needed and makes it possible to provide different levels of abstraction based on the imports.

OPC UA NodeSet Ontology

Perzylo et al. present an OPC UA NodeSet ontology in [61]. Their goal was to use the information models used by OPC UA in combination with a geometry and kinematic model of a resource to create a digital twin. For the geometry model, they rely on their OntoBREP ontology [62], which allows them to define the faces, edges, and vertices of objects based on an infinite geometry with corresponding bounds to make it finite. This twin could then be used for virtual prototyping and automatic deployment of manufacturing tasks. Apart from the ontology itself, the authors also provide a software tool to automatically convert Extensible Markup Language (XML)-based NodeSet specifications

into OWL-based OPC UA NodeSet descriptions. Figure 5.22 shows the main classes needed to represent the OPC UA information model.

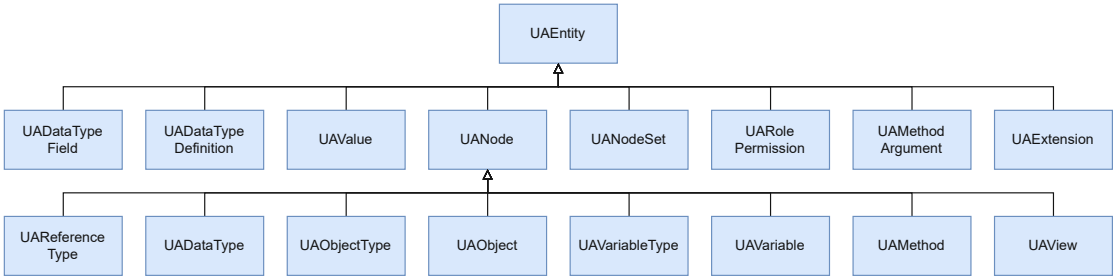
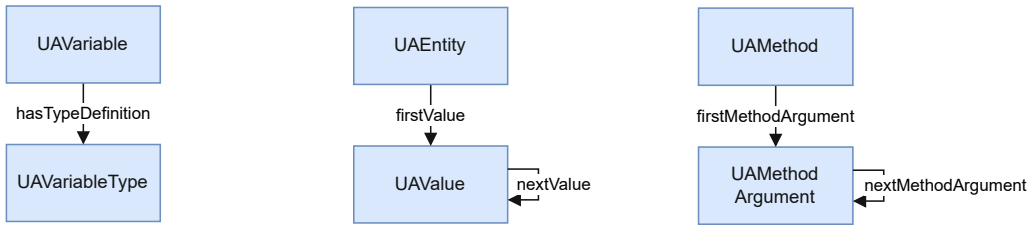


Figure 5.22: Structure of the OPC UA NodeSet ontology (adapted from [61])

The base model of OPC UA as specified by the standard is represented by eight classes. These eight classes define the different possible nodes and consist of *UAReferenceType*, *UADataType*, *UAObjectType*, *UAObject*, *UAVariableType*, *UAVariable*, *UAMethod*, and *UAView*. Each node has a set of mandatory and optional attributes that provide further information about the node's properties. The most important attribute is the *nodeId*, which is unique for each node and used to identify it within the address space. Such properties are represented in the ontology via data properties. An information model also contains binary relations between nodes. One example would be the *hasTypeDefinition* relation shown in Figure 5.23a, which can be used to link *UAVariable* to *UAVariableType*.

The authors introduce additional classes and properties to properly represent NodeSets in an OWL compatible way. These classes consist of *UADataTypeDefinition*, *UADataTypeField*, *UAValue*, *UANode*, *UANodeSet*, *UARolePermission*, *UAMethodArgument*, and *UAExtension*. One change from NodeSets to the OWL representation is made with value arrays, which are converted to single linked lists of *UAValue*. The connection between them is made with the *nextValue* object property, shown in Figure 5.23b. The same principle is applied for *UAMethods* were the arguments, represented by the *UAMethodArgument* class, are linked to the method using the *firstMethodArgument* and *nextMethodArgument* object property. This connection is shown in Figure 5.23c.



(a) Type definition (b) Value assignment (c) Method definition

Figure 5.23: Object properties of OPC UA NodeSet ontology

5.2 Rating of Existing Ontologies

The existing ontologies are now rated based on the following properties:

Domain coverage describes to which extent an ontology covers one or multiple domains. Depending on how well the coverage of the ontology is, less work has to be put into adding missing concepts or additional ontologies.

Overhead describes, whether an ontology contains a lot of overhead. An ontology with little overhead makes the modelling process easier, because less concepts have to be considered.

Competency question coverage describes to which extent an ontology covers one or multiple competency questions. As with the domain coverage, depending on how well the coverage of the ontology is, less work has to be put into adding missing concepts or additional ontologies.

Combinability describes to which extent an ontology can be combined with others. Because of the modular design of the knowledge base and to keep it easily extendable, ontologies, which can be simply combined with others, are preferred.

Starting with the ECO ontology, it is possible to cover the *System Architecture* fully. The hierarchy sub-ontology can be used to create a model of the components of the system. Further, it is possible to model the shop floor and its layout. A disadvantage of the ECO ontology is that the *Asset Properties* can not be modelled in a simple way. This domain would need to be added to the ECO ontology. An advantage of this ontology is the availability of operation types and resources. With them, parts of the *Capability* domain would be covered, as it would allow modelling what kind of manufacturing processes the system can perform. By looking at the competency questions, it becomes apparent that, even though the ECO ontology can cover some domains, additional concepts and relations would need to be added to answer them. A combination with other ontologies covering the missing concepts would be possible. However, as it will be shown, other ontologies provide better coverage of the domains without the overhead of requiring additional concepts.

As stated by the name, the CORA ontology is specified for robotic systems. It provides well-defined concepts and relations to describe the hierarchy and components of such a system. Regarding this work, it could be used to cover the *System Architecture* domain. However, since this ontology is specified for robots, it provides a disadvantage when trying to model more general systems. It is also capable of answering some competency questions. Overall, because of its specification for robot systems, this ontology is not suited for this work. However, if a specialization in robots is desired, the usage of this ontology would be beneficial.

Since AMLO is based on AutomationML, which is used to describe and model automation systems, it fits perfectly to represent just this. With this ontology, it is possible to describe

the *System Architecture* and *Asset Properties*. It allows modelling the architecture freely while giving almost no restrictions on the design. The modeller decides how detailed the system should be represented. Additionally, it is possible to define the properties, such as speed or length of an element, in a simple way. Again no restrictions are posed on the modeller, which properties shall be represented, and in what detail they are defined. So overall, this gives many possibilities to the modeller in designing a manufacturing system. Because it covers two domains fully, this ontology can provide answers to more competency questions compared to the ECO or CORA ontology. These are the reasons why in this work the AMLO will be used to represent manufacturing systems.

Regarding the network domain, two ontologies were presented. The ontology for IT services with its five packages provides an excellent structure representing enterprises and their IT services. It is possible to describe the participants within a network and their hierarchical dependencies. In general, this ontology is focused on the services of a network. It defines concepts of contracts and SLAs, which help describe the services but are not needed in the domain of an RSS. Defining the network topology and connection properties is more beneficial in this context. This ontology's missing concepts are another disadvantage since it provides no way to define network infrastructure such as switches, nodes, or hosts. Also, it has no way of defining the properties of a network, such as connections or configurations. This problem could be overcome by modelling the infrastructure using the AMLO. However, this brings the problem of mixing different domains. In the end, the decision of whether to mix the domains lies with the system modeller. Nevertheless, to support modularization, it is good practice to keep them separate.

The network ontology for computer network management provides an easy way to define the network infrastructure as it has concepts for hardware and software entities. With its object properties, it allows to model the network topology. It further provides ways to represent connections between nodes using flows. With these concepts and relations, this ontology already covers the network domain almost wholly. It also provides answers to some of the competency questions. Nevertheless, the ontology contains much overhead not needed in an RSS. Because of this overhead, this work will introduce a new network ontology in Section 5.3, which is based on this network ontology. The new ontology will be extended to better fit into the context of an RSS.

Only one ontology was presented covering the management domain. One of the reasons for this is that the organization ontology is recommended by the World Wide Web Consortium (W3C). The W3C is an international community that develops open standards. This community's goal is to ensure the web's long-term growth. Another reason is the coverage of the management domain that this ontology provides. It holds concepts and relations that allow displaying memberships and roles of persons. This kind of information is needed in an RSS when it comes to different roles with different functions, e.g., if a safety engineer validated a safety configuration. The ontology contains some overhead, such as concepts representing organizational structures, but it is relatively small.

For the domain of capabilities, two ontologies were presented. The upper ontology for manufacturing service description holds the core concepts to describe a broad spectrum of manufacturing services. In the ontology, a service is connected to a process. Such a process holds multiple resources. With these connections, it is possible to describe the manufacturing process and the resources involved accurately. With the additional capabilities definition for each service, a clear picture of the system, its processes, and possibilities can be modelled. However, the ontology's structure makes it inconvenient to answer the competency questions since a capability can only be assigned to a service and not a device.

The second presented ontology for the capability domain was the MaRCO. This ontology aimed to provide automatic matchmaking between capabilities and product requirements. Because of this goal, the MaRCO consists of four models. Relevant to this work and covering the capability domain are the capability model and, to some extent, the resource model. The capability model provides concepts to define a variety of manufacturing capabilities. It contains simple and combined capabilities and allows for the definition of additional parameters. In the resource model, it is shown how devices relate to capabilities. Concepts such as site, area, or device are already included in other ontologies. Therefore, it is impractical to import the resource model as it would result in redundant concepts within the RSS ontology. Thus, only the capability model will be used, and the relations between the resource and capability model will be added based on the definition in the MaRCO.

Regarding the information model domain, the OPC UA NodeSet ontology was presented. It covers the domain fully since the goal of this ontology was to represent the information model, which could then be used within a digital twin. Because the work also introduces a tool to convert from NodeSet to OWL, it is a good candidate for use in the context of this thesis. A tradeoff that has to be made when using this ontology, and more specifically the conversion tool, is the created overhead. On one hand, when automatically converting a NodeSet file to OWL, many instances might be created that are not necessary. On the other hand, the engineer does not have to perform the conversion process by hand. Because OPC UA is also a part of the network structure, a connection between the information model and the network domain must be created. These relations are shown in Section 5.4.

Table 5.2 gives a summary of the rated ontologies. For every property, each ontology is assigned a grade from one to five, one being the best and five the worst. The table shows that the chosen ontologies have excellent domain coverage, while they might lack in other sections.

Domains	Ontologies	Domain coverage	Overhead	Competency question coverage	Combinability
System Architecture & Asset Properties	ECO	3	3	2	2
	CORA	2	1	2	1
	AMLO	1	2	1	1
Network	Ontology for IT Services	4	4	5	4
	Network Ontology for Computer Network Management	1	4	1	2
Management	**Organization Ontology**	1	2	1	1
Capabilities	Upper ontology for manufacturing service description	3	2	3	2
	MaRCO	1	3	2	1
Information Model	**OPC UA NodeSet Ontology**	1	1	3	2

Table 5.2: Rating of ontologies (1 .. best, 5 .. worst)

5.3 New Ontologies

As mentioned above, a new ontology will be created to represent the network better. The ontology will be created using Protégé, a free, open source ontology editor provided by Stanford University [2]. This ontology is based on the network ontology for computer network management from [19] and takes some of the concepts from there. One main reason for the new design was to reduce the overhead contained in the original ontology. This overhead would have meant that the modeller needed to understand all the existing concepts and relations. This additional needed understanding would have made the usage of the knowledge base cumbersome. Another reason to newly create an ontology for the network domain was to represent its desired concepts better. The competency questions ask about safety configurations and properties. These concepts could not conveniently be represented using the network ontology for computer network management.

Figure 5.24 shows the main communication infrastructure concepts of the ontology. Some concepts were taken from [19] as shown in Section 5.1. The main change is the removal of the software entities like queues and routing tables. Instead of these classes, the *Configuration* class was added. It has two subclasses: *OPC UAConfiguration* and *TSNConfiguration*. These concepts aim to represent the network configurations. While modelling the *Configuration* concept, the problem occurred regarding how to represent a configuration's details. Two different approaches were considered. The first one was to model the properties needed by each configuration explicitly. This approach would provide more implicit knowledge but restricts the modeller to the predefined concepts. The second approach was to use a general class for defining the properties. This resulted in the *Attribute* class. The class uses the data properties *hasAttributeName* and *hasAttributeValue* to define the details of a configuration. Using such a general concept makes it possible for the modeller to decide which properties get defined for a configuration. In the case of the OPC UA configuration, a link to the relevant elements in the information model will also be established. *ListElement* defines the element of a list and is related to the configuration via the *hasListElement* object property. This concept is needed because some configurations can contain a list of values.

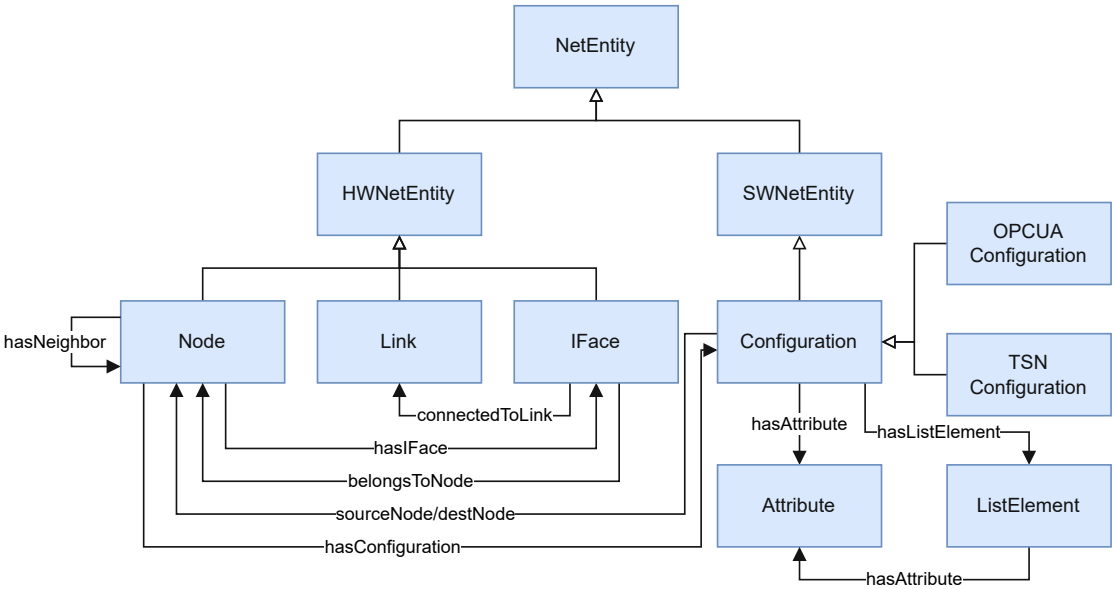


Figure 5.24: Communication infrastructure

Figure 5.25 shows the subclasses of *ListElement*, which are *QueueMaxSduTable*, *PortDataSet*, *Component*, *BridgeVLan*, *TrafficClassV2*, and *AdminControlList*. The connection between list elements is made with the *next* object property.

Figure 5.26 show the main concepts used to represent the traffic information. The main difference to the concepts presented in [19] is that the *Flow* class is replaced with the *Connection* class. In the RSS domain, a static view of the network connections is preferred

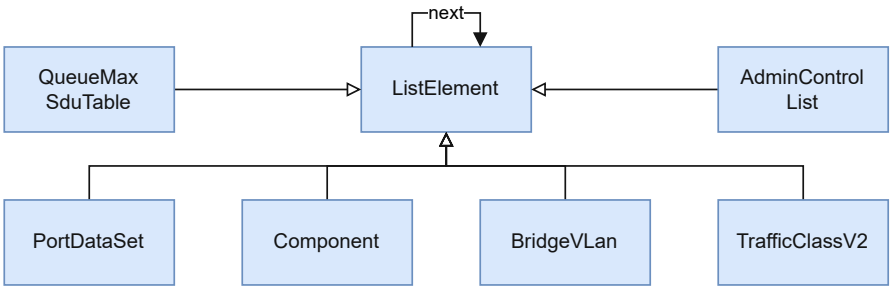


Figure 5.25: List elements

over the abstract flow definition. *Connection* contains two subclasses: *OPCUA* and *TSN*. As with the *Configuration*, the *Connection* details can be set using the *Attribute* class. The *containsConnection* relation associates a connection with a configuration. A connection also *contains* a node to show which nodes are used by the connection. The *Datagram* class was adopted from the original ontology and is used to represent a datagram running over a connection.

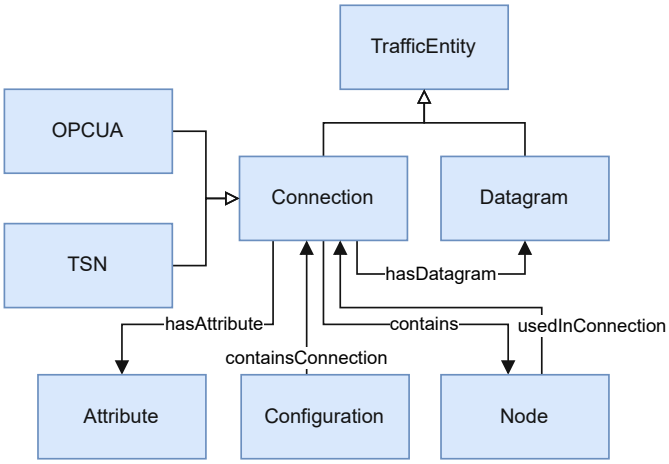


Figure 5.26: Traffic information

5.4 Combination of Ontologies and Final Knowledge Base

Figure 5.27 shows which ontologies cover the different domains. The grey shaded blocks indicate that the full ontology is used. The violet one means only a part of the ontology was taken. In this case, only the capability model was used from the MaRCO ontology. Lastly, the orange shaded block indicates the newly created ontology. Here, only some concepts from other works were used.

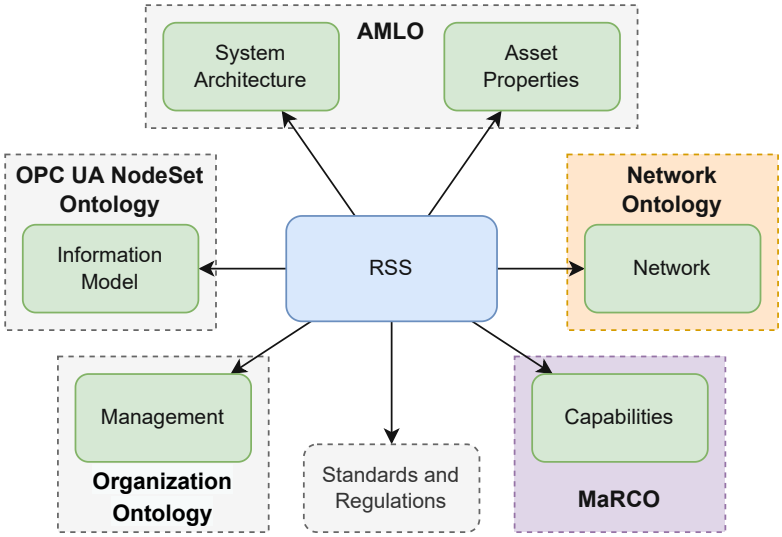


Figure 5.27: Domains of an RSS matched with ontologies

The last step of creating the knowledge base is combining the existing and newly created ontologies. The final ontology which imports the others is called *Reconfigurable Safety System Ontology (RSSO)*[1]. In this ontology, relations between the different ontologies are defined. This addition is done to further enrich the information content of the knowledge base. Figure 5.28 shows the import structure of the RSSO. It further shows the prefixes for each of the ontologies. These prefixes are used in the following figures to clarify to which ontology the classes belong.

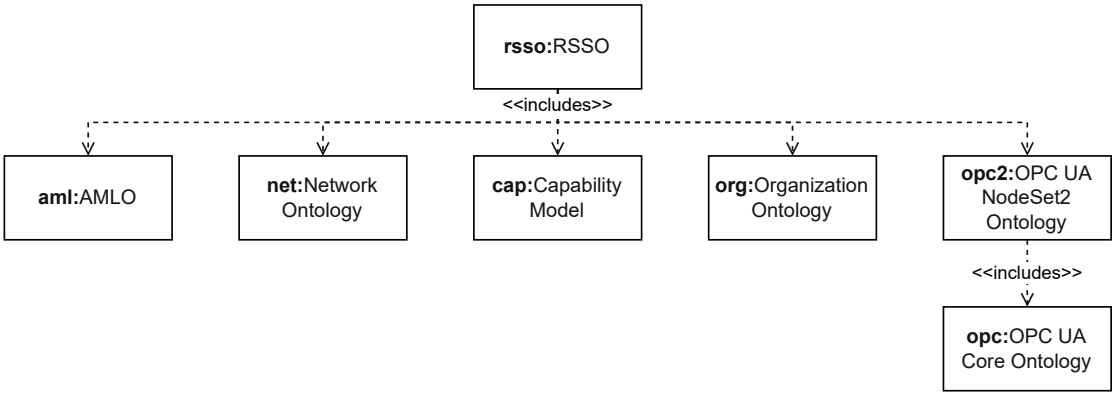


Figure 5.28: Import structure of RSSO

Starting with AMLO and the MaRCO ontology, relations are added that assign the capabilities to the system architecture. Figure 5.29 shows these relations. The added

[1] https://git.auto.tuwien.ac.at/safety/knowledgebase

object properties assign a capability to the three classes in AMLO used to represent elements of a system via the *hasCapability* relation. It also features the inverse object property *providedBy* allowing a search for elements that provide a specific capability.

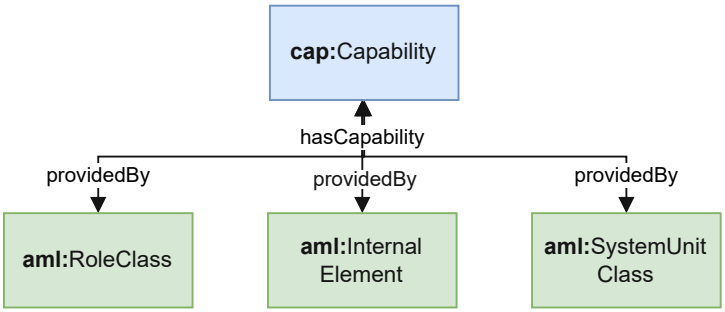


Figure 5.29: Relations between AMLO and MaRCO

The next step is adding relations between the network and OPC UA NodeSet ontology. With this addition, it is possible to connect the network configuration with the corresponding concepts in the OPC UA information model. Figure 5.30 shows these relations. The connection between *Configuration* and *UANode* is made to show which elements are contained in a configuration. As shown in Section 5.1, the *UANode* class contains subclasses such as *UAObject* and *UAVariable*, which might be relevant to a configuration. Further, the relation to *UAValue* is established. The instances of this class hold the values used for the OPC UA configuration, such as IDs or method arguments.

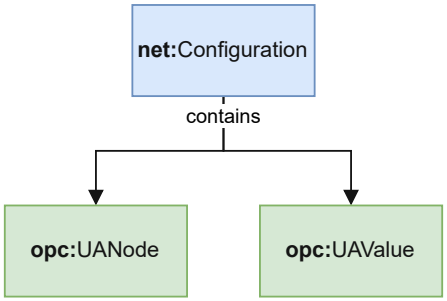


Figure 5.30: Relations between network and OPC UA NodeSet ontology

This work uses concepts of OPC UA, such as servers and clients. Therefore, to properly represent the OPC UA communication between elements, it is necessary to assign them to the corresponding components of the system. Figure 5.31 shows these relations. The *runsOn* object property specifies which *Node* runs on which *InternalElement* or *SystemUnitClass*. This specification enables to assign an OPC UA server to an element of the manufacturing system. Subsequently, because of the relations of the *Node* class within the network ontology, it becomes clear which components are connected and what their configuration is.

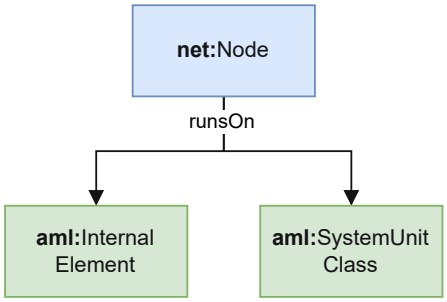


Figure 5.31: Relations between AMLO and network ontology

Next, a connection between the OPC UA NodeSet ontology and AMLO ontology is added. The OPC UA information model is used to define physical entities, systems, and applications so that this information can be used within the CPPS. These details mean that the information model holds a representation of the physical system architecture and properties of them. To be able to connect them to the corresponding elements of AMLO, the relations shown in Figure 5.32 are added. An *UAObject* can represent components of the system architecture. *UAVariable* represents the attributes of those components.

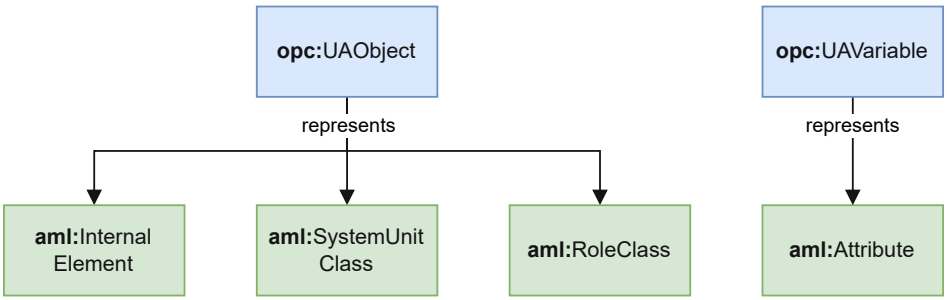


Figure 5.32: Relations between OPC UA NodeSet ontology and AMLO

Another connection can be made between the organization ontology and AMLO, as shown in Figure 5.33. To be able to know which machine components belong to which site, a connection between *Site* and the classes *InternalElement* and *SystemUnitClass* is made via the *contains* relation. The ontology should also be able to model whether a machine needs human interaction. In order to do so, the *operates* object property is defined. It connects *foaf:Person* to *InternalElement* and *SystemUnitClass*.

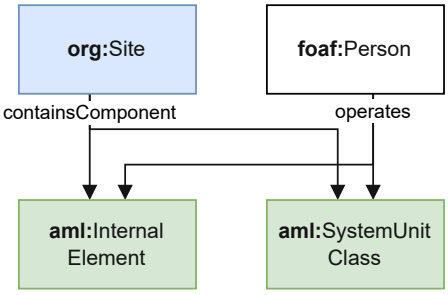


Figure 5.33: Relations between organization ontology and AMLO

The same connection can also be made between the organization and network ontology. Figure 5.34 shows these connections. As with the machine components, *contains* specifies whether a *Node*, *Link* or *IFace* are located at a *Site*. With the *operates* object property it is possible to assign a *foaf:Person* to a *Node*, for example if a host is assigned to specific worker. To ensure that a safety *Configuration* is correct, it needs to be approved by a safety engineer. This approval is reflected in the ontology by the *canApprove* relation, which specifies which *Role* a person must hold to approve a *Configuration*. The *isValidatedBy* relation defines which *foaf:Person* validated a *Configuration*.

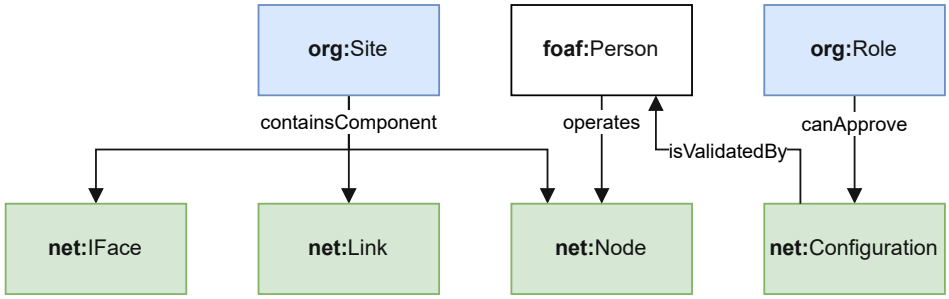


Figure 5.34: Relations between organization and network ontology

CHAPTER 6

Evaluation

Before the evaluation is performed, a simple manufacturing system where the knowledge base will be used is defined. Then, the evaluation setup is explained and some base information is given. The evaluation process of the created knowledge base is then split into two parts. First, the knowledge base is used to answer the previously defined competency questions. Second, the knowledge base is used within the RSS to provide the safety configuration. After this evaluation, the results are summarized, and a discussion is performed.

6.1 Evaluation Setup

To perform the evaluation, the developed knowledge base will be integrated into a manufacturing system. A sketch of this system is shown in Figure 6.1. The manufacturing part of this system consists of two machine tools, a robot and an AGV which transports workpieces between them. Concerning the safety of this system, multiple components are integrated. Each machine, robot, and the AGV is equipped with an emergency stop button. Walls surround the machines on three sides. On the fourth, a light curtain is used to make sure no person can enter the manufacturing cell. For the network, one TSN switch connects all components of the production cell. Each component is also equipped with an OPC UA Publisher and Subscriber, and an OPC UA SafetyProvider and SafetyConsumer. For more information about OPC UA Safety see [58].

SPARQL queries are used to perform the evaluation process. These queries are executed on a Fuseki server. Apache Jena Fuseki [73] provides a standalone SPARQL server. The application takes the ontology as an OWL file and creates a SPARQL endpoint. It provides a Web interface for the queries to be entered and executed. In addition to this interface, it also provides a REST API which allows to execute the queries using HTTP requests.

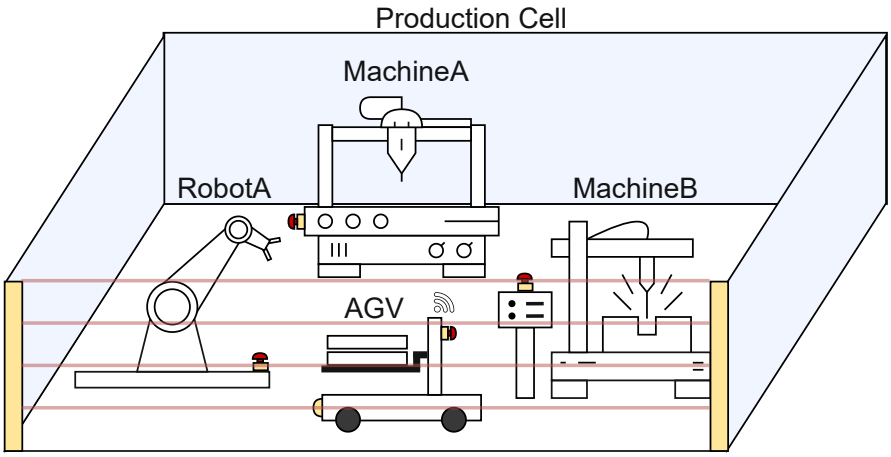


Figure 6.1: Sketch of a manufacturing system

Listing 6.1 gives an example of a simple SPARQL query. This query returns a list of all triples contained in the ontology. Before the query is given, it is possible to define prefixes for a used Internationalized Resource Identifier (IRI). The query is similar to a Structured Query Language (SQL) query and consists of two parts – first, the *SELECT* clause, which defines the variables that should appear in the query result. The second is the *WHERE* clause, which provides the pattern for which to search for in the ontology.

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3
4 SELECT * WHERE {
5     ?sub ?pred ?obj .
6 }
```

Listing 6.1: Simple SPARQL query

Listing 6.2 shows the used prefixes in this work. *rdf* contains general concepts and relations such as *type* or *subClassOf*. *opc* contains the base elements of OPC UA, whereas *opc2* contains the concepts and relations needed to fully represent a NodeSet file. The other prefixes correspond to the ontologies as shown in Figure 5.28 and contain the corresponding concepts. The prefix definitions are omitted in the following queries to keep them shorter.

```
1 PREFIX aml: <https://w3id.org/i40/aml#>
2 PREFIX cap: <http://resourcedescription.tut.fi/ontology/capabilityModel#>
3 PREFIX net: <http://www.semanticweb.org/david/ontologies/2022/4/Network#>
4 PREFIX org: <http://www.w3.org/ns/org#>
5 PREFIX opc: <http://www.fortiss.org/kb/opcua/OpcUaCore.owl#>
6 PREFIX opc2: <http://www.fortiss.org/kb/opcua/OpcUaNodeSet2.owl#>
7 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
8 PREFIX rsso: <http://www.semanticweb.org/david/ontologies/2022/4/RSSO#>
```

Listing 6.2: Prefix definitions

6.2 Competency Questions

The first part of the evaluation is to answer the competency question in the context of the provided manufacturing system shown in Figure 6.1. To get a better understanding of the following queries, Figure 6.2 shows an excerpt of the classes and corresponding individuals. It contains concepts from the AMLO ontology and capability model. Multiple boxes behind each other represent multiple different individuals of the same class.

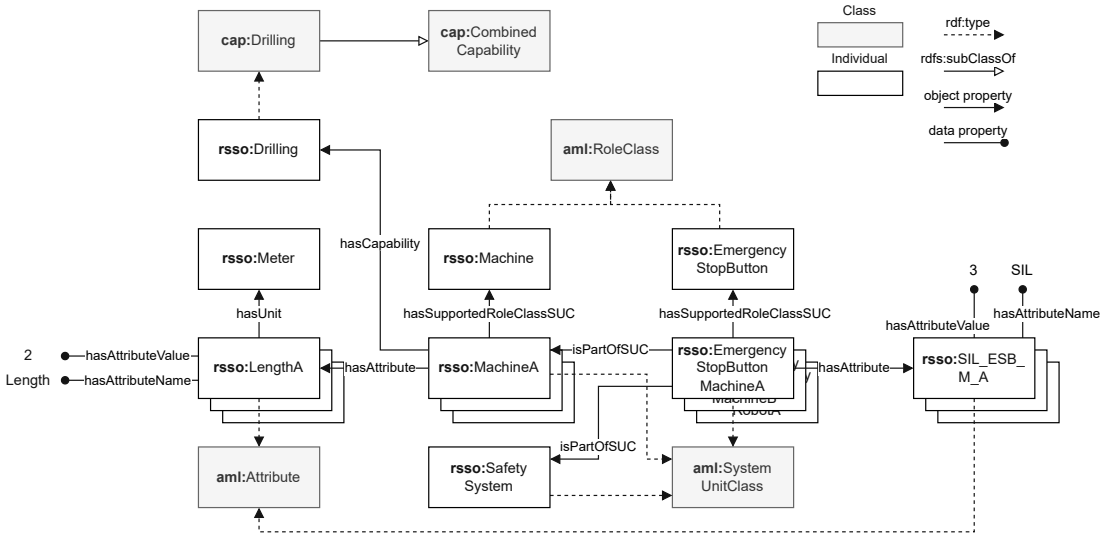


Figure 6.2: Excerpt of individuals describing the system architecture and asset properties

In the following, some of the SPARQL queries used for answering the competency question are presented. The set of evaluated competency questions was chosen such that in combination with the evaluation performed in Section 6.3, every ontology is queried at least once.

Starting with the question: What are the components of the safety system? Listing 6.3 shows the corresponding query. It consists of a simple where clause, which searches for components part of the system unit class “SafetySystem”. Table 6.1 shows the result of this query. As expected, it returns the emergency stop buttons and the light curtain.

```
1 SELECT ?component WHERE {
2     ?component aml:isPartOfSUC rsso:SafetySystem .
3 }
```

Listing 6.3: Query to get the components of the safety system

Component
EmergencyStopButtonAGV_A
EmergencyStopButtonMachineA
EmergencyStopButtonMachineB
EmergencyStopButtonRobotA
LightCurtainA

Table 6.1: Result of query shown in Listing 6.3

For the next query, the SILs of the safety components are asked. Listing 6.4 shows the SPARQL query, which corresponds to the competency question: What SIL does a component have? The query first asks for each component to have an attribute, as shown in line 2. Then it is narrowed down further with lines 3 and 4, where the ?sil variable from before is used. It is searched for the attribute name "SIL" and the result is stored in the ?value variable. To make the output of this query clearer, the ?value variable is renamed to SIL with the *AS* keyword. The result of this query is shown in Table 6.2, which contains the SILs of the safety components.

```
1 SELECT ?component (?value AS ?SIL) WHERE {
2     ?component aml:hasAttribute ?sil .
3     ?sil aml:hasAttributeName "SIL" .
4     ?sil aml:hasAttributeValue ?value
5 }
```

Listing 6.4: Query to get the SILs of the safety components

Component	SIL
EmergencyStopButtonAGV_A	3
EmergencyStopButtonMachineA	3
EmergencyStopButtonMachineB	3
EmergencyStopButtonRobotA	3
LightCurtainA	3

Table 6.2: Result of query shown in Listing 6.4

In the next query, the capability of a machine is asked. Listing 6.5 shows the query, which corresponds to the competency question: Does a component have a specific capability? First, all the components for which a capability is defined are queried. Then, the *BIND* command is used to return true if a component has the *Drilling* capability, otherwise it returns false. Table 6.3 shows the result of this query. As expected, it returns true for MachineA and false for the other components.

```
1 SELECT DISTINCT ?component ?hasCapability WHERE {
2     ?component rsso:hasCapability ?obj .
3     BIND (exists{?cap rdf:type cap:Drilling .
4                  ?component rsso:hasCapability ?cap} AS ?exists)
5     BIND (IF(?exists, "true", "false") AS ?hasCapability)
6 } ORDER BY ASC(?component)
```

Listing 6.5: Query to check if a component has a specific capability

Component	hasCapability
MachineA	true
MachineB	false
RobotA	false

Table 6.3: Result of query shown in Listing 6.5

To get a better understanding of the next query, Figure 6.3 shows another excerpt of the classes and corresponding individuals. The contained concepts are from the network and organization ontology.

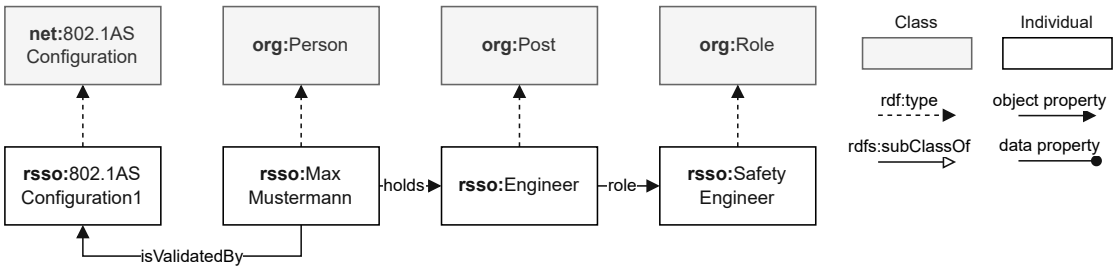


Figure 6.3: Excerpt of individuals describing the network and organization

The last query in this section evaluates the competency question: Is the configuration validated by a safety engineer? This query is shown in Listing 6.5. In this example, the query first asks for all configurations of type *802.1ASConfiguration*. To check the other types of configuration, simply the object in line 2 has to be changed. It is then further refined to only return configurations which are validated by a person holding any post that has the role of a *SafetyEngineer*. Table 6.4 presents the results of this query, which show that this configuration was validated by Max Mustermann.

```
1 SELECT ?configuration (?person AS ?isValidatedBy) WHERE {
2     ?configuration rdf:type net:802.1ASConfiguration .
3     ?configuration rsso:isValidatedBy ?person .
4     ?person org:holds ?post .
5     ?post org:role rsso:SafetyEngineer .
6 }
```

Listing 6.6: Query to check if a configuration is validated by a safety engineer

Configuration	isValidatedBy
802.1ASConfiguration	MaxMustermann

Table 6.4: Result of query shown in Listing 6.6

6.3 Integration in Reconfigurable Safety System

The application scenario in which the knowledge base shall now be used is with the initial commissioning of the components. Here, the safety system configuration is deployed, and the different safety components are added to the system. This deployment is done piecewise, meaning one after the other safety component gets initialized, and the configuration gets deployed. The goal of the knowledge base in this scenario is to provide the correct configuration for each component.

The evaluation of the knowledge base shows only an excerpt of the initial commissioning. Zainzinger [75] provides in depth information about this application scenario. As stated in his work, a safety connection needs to be configured on three levels. First, the TSN network must be configured. Then the OPC UA Publisher/Subscriber connection must be created. Finally, the OPC UA safety layer needs to be configured. Each of these levels has a specific set of parameters needed for configuration. In addition to those parameters, usually an IP address, username, and password are also needed. The full list of those parameters with a detailed explanation can be found in Zainzinger's thesis.

6.3.1 TSN Configuration

The TSN network has to be configured on three levels. The first sets properties for time synchronization, the second sets basic properties on the bridge, and the third is used to set up the interfaces. Thirty-two parameters are needed for the configuration combined over those three levels. Similar to before, Figure 6.4 shows an excerpt of the classes and individuals needed for the TSN configuration. It contains concepts from the AMLO and network ontology. To keep the structure of the figure clearer, attributes are only shown for the 802.1AS configuration and omitted for the others. Also the attribute name and value are not shown.

Since it is possible for multiple configurations to be stored in the knowledge base, it is important to use the newest one. To do so, the timestamp of all available configurations is read. The following queries are then performed on the configuration with the newest timestamp. The query to find this time stamp is shown in Listing 6.7 for the *802.1AS-Configuration*. For all the following configurations, this step is omitted, as it is identical for all of them. Table 6.5 shows the result of this query. Because the query is ordered by the timestamp, the newest one is in the first row.

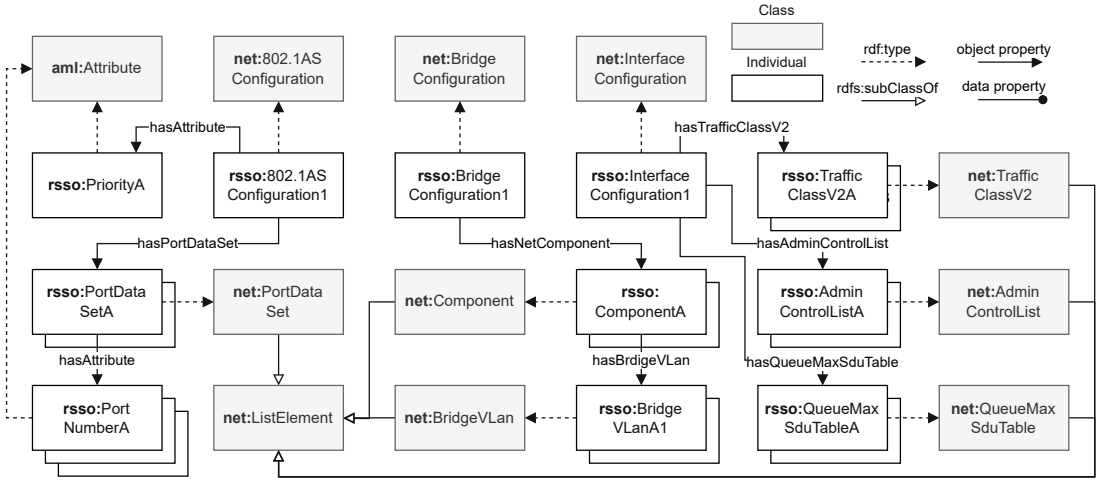


Figure 6.4: Excerpt of individuals describing the TSN configuration

```
SELECT ?configuration ?timestamp WHERE {
    ?configuration rdf:type net:802.1ASConfiguration .
    ?configuration net:hasTimestamp ?timestamp .
} ORDER BY DESC(?timestamp)
```

Listing 6.7: Query to get the timestamp

Configuration	Timestamp
802.1ASConfiguration1	2022-08-26T13:20:54Z
802.1ASConfiguration	2022-07-26T09:48:31Z

Table 6.5: Result of query shown in Listing 6.7

First, the parameters for the time synchronization are queried from the knowledge base. The *priority* parameter is read first via Listing 6.8. The right configuration, called *802.1ASConfiguration1*, was determined based on the timestamp, as shown in Listing 6.7. From this configuration, the attributes are read, and the name and value are returned as a result as shown in Table 6.6. It is possible to define two priorities for the time synchronization, however, in this case both have the same value. Therefore, only one value is stored in the knowledge base.

```
SELECT ?name ?value WHERE {
    rsso:802.1ASConfiguration1 aml:hasAttribute ?att .
    ?att aml:hasAttributeName ?name .
    ?att aml:hasAttributeValue ?value
}
```

Listing 6.8: Query to get the priority

Name	Value
Priority	1

Table 6.6: Result of query shown in Listing 6.8

Next, the *PortDataSets* are queried via Listing 6.9. Again, all the elements of the *802.1ASConfiguration1* are gathered. Then the contained *PortDataSets* are read, and the corresponding attributes with their name and value are returned as a result. The returned data, shown in Table 6.7, is then ordered by the *PortDataSet*. To keep the table a reasonable size, only one of four *PortDataSets* is shown.

```
1 SELECT ?dataset ?name ?value WHERE {
2     rsso:802.1ASConfiguration1 net:hasPortDataSet ?dataset .
3     ?dataset aml:hasAttribute ?att .
4     ?att aml:hasAttributeName ?name .
5     ?att aml:hasAttributeValue ?value
6 } ORDER BY ASC(?dataset)
```

Listing 6.9: Query to get the PortDataSet

Dataset	Name	Value
PortDataSetA	Hs	0
PortDataSetA	InitialLogAnnounceInterval	0
PortDataSetA	InitialLogSyncInterval	-3
PortDataSetA	Ls	655360
PortDataSetA	Ms	0
PortDataSetA	PortNumber	1

Table 6.7: Result of query shown in Listing 6.9

The next step is to read the configuration of the TSN bridge. Listing 6.10 shows the query to get the name and address of the bridge. This query is similar to Listing 6.8. The correct configuration is again found using the timestamp, as shown in Listing 6.7, but this time it is asked for elements of the type *BridgeConfiguration*. From the resulting configuration, called *BridgeConfiguration1*, the attributes with their corresponding names and values are queried. The result is shown in Table 6.8.

```
1 SELECT ?name ?value WHERE {
2     rsso:BridgeConfiguration1 aml:hasAttribute ?att .
3     ?att aml:hasAttributeName ?name .
4     ?att aml:hasAttributeValue ?value
5 }
```

Listing 6.10: Query to get the name and address

Name	Value
Address	61-67-55-23-25-55
Name	DefaultBridgeName

Table 6.8: Result of query shown in Listing 6.10

Next, the list of components with their parameters is read. Listing 6.11 shows the corresponding query. Again, using the *BridgeConfiguration1*, first all components of this configuration are read. The query then returns the attribute name and value for each of the components. In this case, only one component is needed. Table 6.9 shows the result of this query.

```
1 SELECT ?component ?name ?value WHERE {
2     rsso:BridgeConfiguration1 net:hasNetComponent ?component .
3     ?component aml:hasAttribute ?att .
4     ?att aml:hasAttributeName ?name .
5     ?att aml:hasAttributeValue ?value
6 } ORDER BY ASC(?component)
```

Listing 6.11: Query to get components

Component	Name	Value
ComponentA	Name	DefaultComponentNameA
ComponentA	Address	30-02-23-36-40-57
ComponentA	Id	0
ComponentA	Type	c-vlan-component

Table 6.9: Result of query shown in Listing 6.11

Each component also contains a list of *Virtual Local Area Networks (VLANs)*. The read those values, the following query in Listing 6.12 can be used. It is similar to the query shown in Listing 6.11, with the addition that the VLANs get read for each component. For the defined manufacturing system, only one *VLAN* is needed. The result of the query is shown in Table 6.10.

```
1 SELECT ?vlan ?name ?value WHERE {
2     rsso:BridgeConfiguration1 net:hasNetComponent ?component .
3     ?component net:hasBridgeVLan ?vlan .
4     ?vlan aml:hasAttribute ?att .
5     ?att aml:hasAttributeName ?name .
6     ?att aml:hasAttributeValue ?value
7 } ORDER BY ASC(?component)
```

Listing 6.12: Query to get VLANs

VLAN	Name	Value
BridgeVLanA1	VId	1
BridgeVLanA1	Name	TestVlan1

Table 6.10: Result of query shown in Listing 6.12

The configuration of the interface needs two sets of parameters. First, some basic configuration to enable the standard functions of the interface is needed. Second, parameters defining the TSN gate control list are required to configure the TSN stream according to IEEE 802.1Qbv. Listing 6.13 shows the query to read those parameters from the knowledge base. Again, first the correct configuration is found by modifying the query shown in Listing 6.7 to search for configurations of the type *InterfaceConfiguration*. For this configuration, called *InterfaceConfiguration1*, the attributes are read and the name and value for each are returned. To keep the table to a reasonable size, the results shown in Table 6.11 contain only one of three parameter sets.

```
1 SELECT ?name ?value WHERE {
2     rsso:InterfaceConfiguration1 aml:hasAttribute ?att .
3     ?att aml:hasAttributeName ?name .
4     ?att aml:hasAttributeValue ?value
5 }
```

Listing 6.13: Query to get interface parameters

Name	Value
Enabled	true
Name	InterfaceA
Gate-enabled	true
Default-priority	0
Description	Generic Description
Bridge-Name	DefaultBridgeName
Component-Name	DefaultComponentName
Pvid	1
Admin-cycle-time	1/1
Admin-gate-states	255
Config-change	false
Admin-base-time	100/0

Table 6.11: Result of query shown in Listing 6.13

The configuration also contains lists of structs to define traffic classes, maximal Service Data Unit (SDU) size, and the gate control list. Listing 6.14 shows how to read the traffic classes defined for an interface. Starting from the correct configuration,

InterfaceConfiguration1, all the contained traffic classes are read. Then the attributes for each traffic class are read and the name and value are returned. The results are shown in Table 6.12. In case of the other two lists, the query is similar. Only the object property in line 2 needs to be changed to *hasQueueMaxSduTable* and *hasAdminControlList*, respectively.

```
1 SELECT ?trafficclass ?name ?value WHERE {
2     rsso:InterfaceConfiguration1 net:hasTrafficClassV2 ?trafficclass .
3     ?trafficclass aml:hasAttribute ?att .
4     ?att aml:hasAttributeName ?name .
5     ?att aml:hasAttributeValue ?value
6 } ORDER BY ASC(?trafficclass)
```

Listing 6.14: Query to get traffic classes

TrafficClass	Name	Value
TrafficClassV2A	Traffic-class	1
TrafficClassV2A	Priority	2
TrafficClassV2B	Traffic-class	2
TrafficClassV2B	Priority	3

Table 6.12: Result of query shown in Listing 6.14

6.3.2 OPC UA Configuration

The OPC UA configuration consists of four parts. A configuration for the OPC UA Publisher, Subscriber, SafetyConsumer, and SafetyProvider is needed. This section presents the SPARQL queries required for the OPC UA Publisher configuration. Similar queries can be defined for the remaining parts but are not discussed in detail. As before, Figure 6.5 shows some of the individuals needed for the OPC UA configuration. It contains concepts from the OPC UA NodeSet and network ontology. To keep the figure reasonably sized, only parts of the configuration are shown. Properties and values are shown representative for some of the individuals and omitted for others.

The configuration of an OPC UA Publisher consists of two parts, the published data and the connection. To see which components of the information model correspond to these concepts, the query in Listing 6.15 can be used. The first step is to get the correct configuration of the type *PublisherConfiguration* by modifying the query shown in Listing 6.7. For this configuration, called *PublisherConfiguration1*, the contained objects are read and the browseName is returned. The ?name variable is renamed to ?object, to make it clear that the returned elements are objects. Executing this query returns the results shown in Table 6.13. As expected, it returns two objects, a connection and data set folder, which contain the relevant parameters for the OPC UA Publisher.

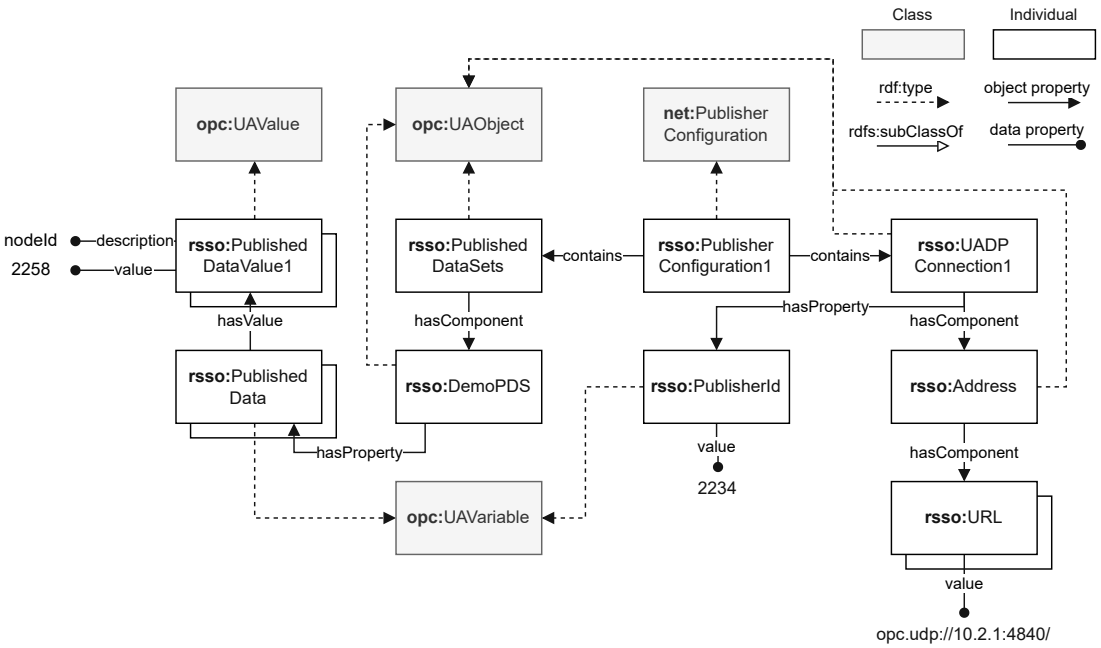


Figure 6.5: Excerpt of individuals describing the OPC UA configuration

```
SELECT (?name AS ?object) WHERE {
    rsso:PublisherConfiguration1 aml:contains ?obj .
    ?obj opc:browseName ?name
}
```

Listing 6.15: Query to get contents of the OPC UA Publisher configuration

Object
UADP Connection 1 PublishedDataSets

Table 6.13: Result of query shown in Listing 6.15

The information contained in the data set folder is used to configure which values from the address space get published. For this, the nodeIds and aliases of the fields are needed. To get the configuration parameters from the data set folder, the query shown in Listing 6.16 can be used. Utilizing the previously found name of the data set folder, the contained dataset is read using the *hasComponent* property. The dataset contains the needed parameters, which are stored in two properties. These properties are read using the *hasProperty* relation. From each of these properties, the relevant information is extracted.

Table 6.14 shows the result of this query. The first two values are the namespaceIndex and nodeId of the published variable. The third value represents an alias for this variable. It is important to note that, because the ontology stores arrays as linked lists, this query only return the values of the first variable. If more than one variable is published, the object property *nextValue* is used to create a connection between them. This object property can then be used to iteratively get from the first to the last value.

```
SELECT ?datasetname ?propertyname ?description ?value WHERE {
    rsso:PublisherConfiguration1 aml:contains ?obj .
    ?obj opc:browseName "PublishedDataSets" .
    ?pds opc:browseName ?datasetname .
    ?pds opc2:hasProperty ?prop .
    ?prop opc:browseName ?propertyname .
    ?prop opc:hasValue ?opcval .
    ?opcval opc:description ?description .
    ?opcval opc:value ?value
}
```

Listing 6.16: Query to get parameters for the dataset

Dataset Name	Property Name	Description	Value
Demo PDS	PublishedData	namespaceIndex	0
Demo PDS	PublishedData	nodeId	2258
Demo PDS	DataSetMetaData	fieldAliases	Server localtime

Table 6.14: Result of query shown in Listing 6.16

Next, the information for the connection is read from the knowledge base. First, the Publisher identifier is read via the query shown in Listing 6.17. The first step is to get the connection object from the Publisher configuration using the previously found name. Then, the property with the browse name “PublisherId” is read. Table 6.15 shows the result.

```
SELECT ?name ?value WHERE {
    rsso:PublisherConfiguration1 aml:contains ?obj .
    ?obj opc:browseName "UADP Connection 1" .
    ?obj opc2:hasProperty ?prop .
    ?prop opc:browseName "PublisherId" .
    ?prop opc:browseName ?name .
    ?prop opc:value ?value .
}
```

Listing 6.17: Query to get the PublisherId

Name	Value
PublisherId	2234

Table 6.15: Result of query shown in Listing 6.17

Listing 6.18 shows the query to read the URL of the Subscriber. Again, the connection object is read from the Publisher configuration. Then, the address component is read. From the address component, the sub-component with the browse name "URL" is read, and its value is returned. Table 6.16 shows the result.

```
SELECT ?name ?value WHERE {
    rsso:PublisherConfiguration1 aml:contains ?obj .
    ?obj opc:browseName "UADP Connection 1" .
    ?obj opc:hasComponent ?address .
    ?address opc:browseName "Address" .
    ?address opc:hasComponent ?subcomponent .
    ?subcomponent opc:browseName "URL" .
    ?subcomponent opc:browseName ?name .
    ?subcomponent opc:value ?value .
}
```

Listing 6.18: Query to get the URL

Name	Value
URL	opc.udp://10.2.1:4840/

Table 6.16: Result of query shown in Listing 6.18

The configuration for the OPC UA Subscriber needs almost the same parameters as the Publisher, thus resulting in nearly identical queries. The same is valid for the parameters of the SafetyConsumer and SafetyProvider. Therefore, these queries are omitted, but can easily be created by viewing the required parameters presented in [75] and alter the above shown queries accordingly.

CHAPTER 7

Conclusion & Outlook

The following chapter provides a summary of this thesis. After the initial problem is stated, a summarized view of all the performed tasks to create and evaluate the knowledge base is given. This section is followed by the conclusion, where the main findings and takeaways are described. Finally, a brief outlook is given as to what future work might cover.

7.1 Summary

The fourth industrial revolution brings changes to automation systems. New systems become more decentralized and have a high demand for interoperability and connectivity between different components. These changes also result in a convergence of the IT and OT domain. OT concepts are brought into the IT world and vice versa. OT components, such as sensor, can upload data directly to a central server for monitoring and a knowledge base can be used to represent the OT system and use it in the IT domain. Another essential property of Industry 4.0 is on-demand individualization. To satisfy this property, RMSs are needed, as they allow for adaptive production. This dynamic approach poses a requirement for the safety system, as it also needs to be reconfigurable.

Presented with the problems that arise with Industry 4.0 and its dynamic production environment, this thesis provided some solutions to the problems that safety systems face in this context. This work resulted in a knowledge base providing safety-relevant information for manufacturing systems. First, some historical context regarding manufacturing systems was given. Then, base knowledge was provided by describing the technical background. Next was the main part of this thesis with the knowledge base creation. A requirements analysis was performed to set the delimitations for the knowledge base. This analysis was done by describing an application scenario in which the knowledge base gets used and then deriving the domains from it. In addition to the scenario, competency questions were defined, which specify questions that the developed knowledge base should

be able to answer. Matching these questions to the domains helped finding missing questions or domains before the next step was performed. There, existing ontologies covering the found domains were researched. The more fitting ones were then described in more detail and the main relevant concepts were shown. After rating the presented ontologies, new ones were created covering the remaining domains. All of these ontologies were then combined to create the final result. Connections between the different ontologies were made to further enrich the information contained in the knowledge base.

The finished knowledge base was then evaluated in two ways. First, the predefined competency questions were answered. This evaluation was done by populating the knowledge base with example data and then querying it using SPARQL. The OWL file was deployed to a Fuseki server, and the queries were executed on the corresponding web interface. A few of the queries used to answer the competency questions were presented and explained in more detail. The result of these queries showed that the created knowledge base could provide answers to the questions.

The second evaluation was performed using the developed knowledge base within an application scenario. The scenario was the initial commissioning of a manufacturing system where the safety system configuration was deployed. The role of the knowledge base was to provide the configuration for the different devices used in the safety system. One part of this system was made up by the TSN network. The knowledge base was queried to provide the TSN configuration. Part of these queries were provided and explained in greater detail. These queries showed that the knowledge base could provide the configuration. It is important to note that multiple queries are necessary to get the configuration, and some knowledge of the domains and ontologies is advantageous.

After the parameters for the TSN network of the system were read, the configuration for the OPC UA Publisher was queried from the knowledge base. As before, some queries were presented and explained in more detail. The result of these queries contained the required parameters to initialize a Publisher correctly. Also needed for OPC UA are the Subscriber, SafetyConsumer, and SafetyProducer parameters. These parameters can be retrieved with similar queries to the ones used for the Publisher and were therefore omitted in this work.

7.2 Conclusion

Overall, the developed knowledge base was able to provide answers to questions coming from the covered domains. It is beneficial to know the domains and the relations between them to use the knowledge base with all its information to its full extent.

The creation of the knowledge base followed the methodology shown in Figure 3.1. This methodology has proven to work well for the given task. However, while the methodology was presented in a strictly linear manner, applying it for the given use case required multiple iterations. This was the case in the requirements analysis, where while merging the competency questions into the defined domains, it became apparent that for some

domains questions were missing. The same problem was observed between ontology creation and evaluation. The evaluation of the knowledge base within an RSS showed that some concepts needed for the network configuration were missing. To accommodate for such scenarios the methodology could be extended to represent these needed iterations.

Following the methodology, a good set of requirements was provided such that the creation of the ontology could be easily performed. One problem that arose while researching existing ontologies to use in this context, was the initial selection of appropriate ones. There exists a lot of literature utilizing ontologies for specific tasks within a domain, however, only a subset of these papers proved to be useful for this work. Another problem while researching existing ontologies, was acquiring the ontology file for them. In some papers, a link to download the file was included. However, with some of them, no resources were provided. In this case, an additional search through the Web was necessary to try and find the file, which was not always successful.

While modelling the concepts for the newly created ontologies, some additional research was necessary to correctly represent the required classes. This was the case, for example, with the modelling of the TSN and OPC UA configurations. It was necessary to know how the different parameters were structured to be able to correctly represent them in the ontology. Configurations often use lists of structs or arrays to store parameters. In both cases, this was modelled in the ontology using linked lists.

The provided methodology can also be used outside of the RSS domain. By keeping the steps abstract, the methodology can easily be applied for the creation of a knowledge base in a different domain. Adaptations only have to be made in the evaluation step, as an integration into an RSS does not fit in another domain. This step can be replaced by evaluating the created knowledge base in another fitting real-world example.

Because of the modularization of the knowledge base, ontologies can easily be replaced and new ones added. This allows for an adaption of the knowledge base to cover additional use cases.

7.3 Future Work

In future work, the knowledge base could be extended to feature a more precise geometry model, such as OntoBREP shown by Perzylo et al. in [62]. Such a geometry model could be needed in the safety evaluation of a manufacturing system. It can help with hazard analysis and enables the safety engineer to precisely define the system's border. Precise borders might be needed if an automatic safety configuration creation process is developed.

A more significant addition that can be made to this knowledge base would be the addition of ontologies covering different relevant safety standards. By including them in the knowledge base, information regarding risk identification and hazard prevention is available to the safety engineer. Standards such as the IEC 61508, IEC 62061, and ISO 13849-1 are crucial for functional safety. In the case of an automatic safety configuration,

the tool could easily get the required information from the knowledge base. However, the modelling of standards is elaborate work and in-depth knowledge of them is required.

Future work could perform research on requirements for ontologies and triple stores such that they are safe. A question that arises in this context is, whether data has to be stored immutable and redundant for it to be considered safe. A similar problem occurs at the interface between the knowledge base and components accessing data from it. When querying data, it is important to make sure that it is not changed along the way. Here, an analysis could be performed, on whether it is sufficient to define specific requirements for the interfaces to solve this problem or if a validation and plausibility check should be implemented in a later step.

The main use of the knowledge base in this work is to store relevant information regarding reconfigurable manufacturing and safety systems. The use of features such as reasoning or rule checking could prove useful in this context. Shapes Constraint Language (SHACL) could be used to create constraints on the content and structure of the knowledge base to provide the safety engineer with a frame of how the data needs to be structured. SHACL rules can also add inferencing capabilities. They define which statements can be derived from existing ones.

List of Figures

List of Tables

Listings

Acronyms

AGV Automated Guided Vehicle. 2, 3, 63

AMLO AutomationML Ontology. 36, 39, 40, 52, 53, 55, 58–61, 65, 68, 81, 82

AMS Adjustable Manufacturing System. 7, 8

BIM Building Information Model. 24

CAEX Computer Aided Engineering Exchange. 39

CNC Computer Numerical Control. 8

CORA Core Ontology for Robotics and Automation. 36, 38, 52, 53, 55, 81

CPPS Cyber-Physical Production System. 3, 23, 40, 60

DMS Dedicated Manufacturing System. 7–9

E/E/PE Electrical, Electronical, and Programmable Electrical. 11–13

ECO Enterprise Control Ontology. 21, 35, 36, 52, 53, 55

EEA European Economic Area. 10

EKB Engineering Knowledge Base. 24

FMEA Failure Mode and Effects Analysis. 23

FMS Flexible Manufacturing System. 7–9

FSS Flexible Safety System. 3, 4

FTA Fault Tree Analysis. 23

HLC High-Level Control. 21

HMI Human Machine Interface. 4

IRI Internationalized Resource Identifier. 64

ISA International Society of Automation. 21

IT Information Technology. 1, 2, 77

KBS Knowledge-Based System. 4, 14, 16–18

LLC Low-Level Control. 21

MaRCO Manufacturing Resource Capability Ontology. 36, 47, 54, 55, 57–59, 81, 82

MQTT Message Queuing Telemetry Transport. 23

MSDL Manufacturing Service Description Language. 49, 50

OBDM Ontology-Based Data Management. 23

OPC UA Open Platform Communications Unified Architecture. xi, xiii, 1, 23, 28, 31, 36, 50, 51, 54–56, 59, 60, 63, 64, 68, 73, 74, 76, 78, 79, 81, 82, 85

ORA Ontologies for Robotics and Automation. 38

OT Operational Technology. 1, 2, 77

OWL Web Ontology Language. 15, 51, 54, 63, 78

PFH$_d$ Probability of dangerous Failure per Hour. 10

PL Performance Level. 10, 13

PPR Product-Process-Resource. 22

PSM Problem-Solving Method. 14, 16–18

RDF Resource Description Framework. 15

RMS Reconfigurable Manufacturing System. 3, 4, 6–9, 21–23, 30, 77

RSS Reconfigurable Safety System. 4, 6, 28, 29, 31, 32, 53, 54, 56, 58, 63, 79, 81, 82

RSSO Reconfigurable Safety System Ontology. 58, 82

SCS Safety-related Control System. 12, 13

SDU Service Data Unit. 72

SHACL Shapes Constraint Language. 80

SIL Safety Integrity Level. 10, 12, 66, 85

SLA Service Level Agreement. 41, 42, 53

SPARQL SPARQL Protocol and RDF Query Language. 15, 24, 28, 63–66, 78, 85

SQL Structured Query Language. 64

SRP/CS Safety-Related Parts of Control System. 13

SUMO Suggested Upper Merged Ontology. 38, 39

SWRL Semantic Web Rule Language. 24

TOVE Toronto Virtual Enterprise. 20

TSN Time Sensitive Networking. xi, xiii, 1, 28, 30, 31, 63, 68–70, 72, 78, 79, 82

VLAN Virtual Local Area Network. 71, 72, 85

W3C World Wide Web Consortium. 53

XML Extensible Markup Language. 50

Bibliography

[1] PROFIsafe Systembeschreibung Technologie und Anwendung. https://www.profibus.com/index.php?eID=dumpFile&t=f&f=51718&token=0f17bd72b13f705ad7cdd03db6df92a225812127. Accessed: 2022-08-20.

[2] Protégé. https://protege.stanford.edu/. Accessed: 2022-08-20.

[3] Umrechnung von Performance Level PL auf Sicherheits-Integritätslevel SIL. http://www.maschinen-sicherheit.net/07-seiten/3450-von-pl-auf-sil.php. Accessed: 2022-08-20.

[4] Allemang, Dean and Hendler, James. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[5] Ameri, Farhad and Dutta, Debasish. An upper ontology for manufacturing service description. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 42578, pages 651–661, 2006.

[6] Angele, Jürgen and Fensel, Dieter and Landes, Dieter and Studer, Rudi. Developing knowledge-based systems with MIKE. In *Domain Modelling for Interactive Systems Design*, pages 9–38. Springer, 1998.

[7] Belden Inc. Moving from the automation pyramid to the automation pillar. https://www.belden.com/Solutions/TSN-Technology?tab=1, 2022. Accessed: 2022-08-20.

[8] Benjamins, V Richard and Fensel, Dieter. Problem-solving methods. *Int. J. Hum. Comput. Stud.*, 49(4):305–313, 1998.

[9] Birmingham, William and Klinker, Georg. Knowledge-acquisition tools with explicit problem-solving models. *The Knowledge Engineering Review*, 8(1):5–25, 1993.

[10] Brecher, Christian and Buchsbaum, Melanie and Ziegler, Frances and Storms, Simon. Ontology-based data management for adaptable safety functions in cyber-physical production systems. *Procedia CIRP*, 104:194–199, 2021.

[11] Chandrasekaran, Balakrishnan. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Intelligent Systems*, 1(03):23–30, 1986.

[12] Chandrasekaran, Balakrishnan and Johnson, Todd R and Smith, Jack W. Task-structure analysis for knowledge modeling. *Communications of the ACM*, 35(9):124–137, 1992.

[13] Chandrasekaran, Balakrishnan and Josephson, John R and Benjamins, V Richard. What are ontologies, and why do we need them? *IEEE Intelligent Systems and their applications*, 14(1):20–26, 1999.

[14] Clancey, William J. Heuristic classification. *Artificial intelligence*, 27(3):289–350, 1985.

[15] Clancey, William J. The knowledge level reinterpreted: Modeling how systems interact. In *Knowledge Acquisition: Selected Research and Commentary*, pages 39–45. Springer, 1989.

[16] Cristani, Matteo and Cuel, Roberta. A survey on ontology creation methodologies. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 1(2):49–69, 2005.

[17] DARPA Agent Markup Language. Semantic Web Services. `http://www.daml.org/services/`, 2004. Accessed: 2022-07-11.

[18] Dave Reynolds. The Organization Ontology. `https://www.w3.org/TR/2014/REC-vocab-org-20140116/`, 2014. Accessed: 2022-07-05.

[19] De Paola, Alessandra and Gatani, Luca and Re, Giuseppe Lo and Pizzitola, Alessia and Urso, Alfonso. A network ontology for computer network management. *Tech. Rep. 22*, 2003.

[20] Directive, Machinery. Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006. *Official Journal of the European Union—09.06*, page L157, 2006.

[21] Eriksson, Henrik and Shahar, Yuval and Tu, Samson W and Puerta, Angel R and Musen, Mark A. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, 1995.

[22] Etz, Dieter and Frühwirth, Thomas and Denzler, Patrick and Kastner, Wolfgang. Functional Safety Use Cases in the Context of Reconfigurable Manufacturing Systems. 2022. in press.

[23] Etz, Dieter and Frühwirth, Thomas and Kastner, Wolfgang. Flexible safety systems for smart manufacturing. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1123–1126. IEEE, 2020.

[24] Fernández-López, Mariano and Gómez-Pérez, Asunción and Juristo, Natalia. Methontology: from ontological art towards ontological engineering. 1997.

[25] Freitas, Jorge Manuel and Correia, Anacleto and e Abreu, Fernando Brito. An Ontology for IT Services. In *JISBD*, pages 367–372, 2008.

[26] Frühwirth, Thomas and Kastner, Wolfgang and Krammer, Lukas. A methodology for creating reusable ontologies. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 65–70. IEEE, 2018.

[27] Gehlen, P. *Funktionale Sicherheit von Maschinen und Anlagen: Umsetzung der Europäischen Maschinenrichtlinie in der Praxis*. Wiley, 2010.

[28] Gennari, John H and Tu, Samson W and Rothenfluh, Thomas E and Musen, Mark A. Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, 41(3):399–424, 1994.

[29] Gil, Yolanda and Paris, Cécile. Towards method-independent knowledge acquisition. *Knowledge acquisition*, 6(2):163–178, 1994.

[30] Grüninger, Michael and Fox, Mark S. Methodology for the design and evaluation of ontologies. 1995.

[31] Hitomi, Katsundo. *Manufacturing Systems Engineering: A unified approach to manufacturing technology, productionmanagement, and industrial economics*. Routledge, 2017.

[32] Horrocks, Ian. Ontologies and the semantic web. *Communications of the ACM*, 51(12):58–67, 2008.

[33] Hummen, René and Kehrer, Stephan and Kleineberg, Oliver. TSN–Time Sensitive Networking. *Hirschmann, USA, WP00027*, 2016.

[34] IEC. Functional safety of electrical/electronic/programmable electronic safety-related systems. Technical Report IEC 61508, The International Electrotechnical Commission, 2005.

[35] IEC. Enterprise-control system integration - Part 1: Models and terminology. Technical Report IEC 62264-1, The International Electrotechnical Commission, 2013.

[36] IEC. Safety of machinery - Functional safety of safety-related control systems. Technical Report IEC 62061, The International Electrotechnical Commission, 2021.

[37] ISO. Safety of machinery — General principles for design — Risk assessment and risk reduction. Technical Report ISO 12100:2010, The International Organization for Standardization, 2010.

[38] ISO. Safety of machinery - Safety-related parts of control systems - Part 2: Validation. Technical Report ISO 13849-2, The International Organization for Standardization, 2012.

[39] ISO. Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design. Technical Report ISO 13849-1, The International Organization for Standardization, 2015.

[40] Eeva Järvenpää, Niko Siltala, Otto Hylli, and Minna Lanz. The development of an ontology for describing the capabilities of manufacturing resources.

[41] Kifer, Michael and Lausen, Georg and Wu, James. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995.

[42] Koch, Thomas. Approach for an automated safety configuration for robot applications. *Procedia CIRP*, 84:896–901, 2019.

[43] Koo, CH and Vorderer, M and Junker, S and Schröck, S and Verl, A. Challenges and requirements for the safety compliant operation of reconfigurable manufacturing systems. *Procedia CIRP*, 72:1100–1105, 2018.

[44] Koren, Yoram and Gu, Xi and Guo, Weihong. Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering*, 13(2):121–136, 2018.

[45] Koren, Yoram and Heisel, Uwe and Jovane, Francesco and Moriwaki, Toshimichi and Pritschow, Gumter and Ulsoy, Galip and Van Brussel, Hendrik. Reconfigurable manufacturing systems. *CIRP annals*, 48(2):527–540, 1999.

[46] Kovalenko, Olga and Grangel-González, Irlán and Sabou, Marta and Lüder, Arndt and Biffl, Stefan and Auer, Sören and Vidal, Maria-Esther. AutomationML ontology: modeling cyber-physical systems for industry 4.0. *IOS Press Journal*, 1, 2018.

[47] Lai, Lien F. A knowledge engineering approach to knowledge management. *Information Sciences*, 177(19):4072–4094, 2007.

[48] Lasi, Heiner and Fettke, Peter and Kemper, Hans-Georg and Feld, Thomas and Hoffmann, Michael. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014.

[49] Leane Clifton. Evolution von Fertigungsparadigmen. `https://new.siemens.com/de/de/unternehmen/stories/industrie/the-factory-of-the-future.html`, 2018. Accessed: 2022-06-09.

[50] Lepuschitz, Wilfried and Zoitl, Alois and Vallée, Mathieu and Merdan, Munir. Toward self-reconfiguration of manufacturing systems using automation agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):52–69, 2010.

[51] Mahnke, Wolfgang and Leitner, Stefan-Helmut and Damm, Matthias. *OPC unified architecture*. Springer Science & Business Media, 2009.

[52] McDermott, John. Preliminary steps toward a taxonomy of problem-solving methods. In *Automating knowledge acquisition for expert systems*, pages 225–256. Springer, 1988.

[53] Melik-Merkumians, Martin and Moser, Thomas and Schatten, Alexander and Zoitl, Alois and Biffl, Stefan. Knowledge-based Runtime Failure Detection for Industrial Automation Systems. In *Models@ run. time*, pages 108–119. Citeseer, 2010.

[54] Morik, Katharina. Underlying assumptions of knowledge acquisition and machine learning. *Knowledge Acquisition*, 3(2):137–156, 1991.

[55] Musen, Mark A. An overview of knowledge acquisition. *Second generation expert systems*, pages 405–427, 1993.

[56] Newell, Allen. The knowledge level. *Artificial intelligence*, 18(1):87–127, 1982.

[57] Noy, N and McGuinness, Deborah L and others. Ontology development 101. *Knowledge Systems Laboratory, Stanford University*, 2001, 2001.

[58] OPC Foundation and PROFIBUS Nutzerorganisation e.V. *OPC Unified Architecture Part 15: Safety*, Release 1.05.00 edition, 11 2021.

[59] Owen, Robert. *Observations on the Effect of the Manufacturing System: With Hints for the Improvement of Those Parts of the System which are the Most Injurious*. R. and A. Taylor, 1815.

[60] Pérez, Jorge and Arenas, Marcelo and Gutierrez, Claudio. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.

[61] Perzylo, Alexander and Profanter, Stefan and Rickert, Markus and Knoll, Alois. Opc ua nodeset ontologies as a pillar of representing semantic digital twins of manufacturing resources. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1085–1092. IEEE, 2019.

[62] Perzylo, Alexander and Somani, Nikhil and Rickert, Markus and Knoll, Alois. An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions. In *2015 ieee/rsj international conference on intelligent robots and systems (iros)*, pages 4197–4203. IEEE, 2015.

[63] Poeck, Karsten and Gappa, Ute. Making role-limiting shells more flexible. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 103–122. Springer, 1993.

[64] Prestes, Edson and Carbonera, Joel Luis and Fiorini, Sandro Rama and Jorge, Vitor AM and Abel, Mara and Madhavan, Raj and Locoro, Angela and Goncalves, Paulo and Barreto, Marcos E and Habib, Maki and others. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.

[65] Puerta, Angel R and Egar, John W and Tu, Samson W and Musen, Mark A. A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge acquisition*, 4(2):171–196, 1992.

[66] Schreiber, Guus and Wielinga, Bob and Breuker, Joost. *KADS: A principled approach to knowledge-based system development*, volume 11. Academic Press, 1993.

[67] Schreiber, Guus and Wielinga, Bob and de Hoog, Robert and Akkermans, Hans and Van de Velde, Walter. CommonKADS: A comprehensive methodology for KBS development. *IEEE expert*, 9(6):28–37, 1994.

[68] Seyedamir, Ahmadi and Ferrer, Borja Ramis and Lastra, Jose L Martinez. An ISA-95 based ontology for manufacturing systems knowledge description extended with semantic rules. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 374–380. IEEE, 2018.

[69] Shadbolt, Nigel and Motta, Enrico and Rouge, Alain. Constructing knowledge-based systems. *IEEE Software*, 10(6):34–38, 1993.

[70] Sowa, John F. *Knowledge representation: logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., 1999.

[71] Steels, Luc. The componential framework and its role in reusability. In *Second generation expert systems*, pages 273–298. Springer, 1993.

[72] Studer, Rudi and Benjamins, V Richard and Fensel, Dieter. Knowledge engineering: Principles and methods. *Data & knowledge engineering*, 25(1-2):161–197, 1998.

[73] The Apache Software Foundation. Apache Jena Fuseki. `https://jena.apache.org/documentation/fuseki2/index.html`, 2022. Accessed: 2022-07-25.

[74] Uschold, Mike and Gruninger, Michael. Ontologies: Principles, methods and applications. *The knowledge engineering review*, 11(2):93–136, 1996.

[75] Zainzinger, Daniel. Configuration Deployment for Reconfigurable Safety Systems. Master's thesis, Technical University of Vienna, 2022. in press.

[76] Zhang, G and Liu, R and Gong, L and Huang, Q. An analytical comparison on cost and performance among DMS, AMS, FMS and RMS. In *Reconfigurable manufacturing systems and transformable factories*, pages 659–673. Springer, 2006.

[77] Zhang, Sijie and Boukamp, Frank and Teizer, Jochen. Ontology-based semantic modeling of construction safety knowledge: Towards automated safety planning for job hazard analysis (JHA). *Automation in Construction*, 52:29–41, 2015.