TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science

DEVICE INFORMATION MODELING IN AUTOMATION - A COMPUTER-SCIENTIFIC APPROACH

Doctoral Thesis by Andreas Gössling

born February 13th 1982 in Duisburg, Germany

Thesis handed in at the Faculty of Computer Science of Technische Universität Dresden

to gain the title of Doktoringenieur

Advisor: Prof. Dr.-Ing. habil. Martin Wollschlaeger, Technische Universität Dresden

Subject Specialist: Prof. Dr.-Ing. habil. Klaus Kabitzsch, Technische Universität Dresden

Second Reporter: Dr. Dimitris Kiritsis, Ecole polytechnique federale de Lausanne

Chairman of the commission: Prof. Dr.-Ing. habil. Rainer G. Spallek, Technische Universität Dresden

Member of the commission: Prof. Dr. rer. nat. habil. Uwe Aßmann, Technische Universität Dresden

Date of the defense talk:

February 27, 2014

CONTENTS

1 INTRODUCTION

The history of computer science is a young one, compared to other, longer established sciences. However, since the rise of computer science from the works of Babbage, Zuse, Turing, and von Neumann, it has driven the course of engineering and natural sciences ever onwards, catalysing developments unthinkable without the support of information processing systems.

The application of computer science has reached almost every part of human life. One such part is work life. The spread of IT (Information Technology) infrastructure and sophisticated communication technologies is still in the process of revolutionising the office workplace. But other workplaces are affected almost as intensely. While logistics have changed under the impact of the Global Positioning System and ubiquitious internet connectivity, one sector is less prominent in public perception, while being as important for the economy as a whole. The sector addressed is the producing industry, divided into manufacturing industry, process industry, semiconductor industry and several other distinct specialisations.

The producing industry in developed countries today is characterised by a high level of automation. This has led to a whole industry, the automation industry, catering to the needs of factory and plant owners. The automation industry is driven by technology (as well as cost pressure). Yet it has shown great care about adapting new technologies. Due to the high risk levels involved when operating machinery in people's working places, as well as the potential financial impact of failures in the production process, only technologies deemed proven and safe are adapted. Nevertheless, the automation industry is adapting new features of computer science with increasing speed, driven by the importance of competitiveness in a global market.

Early automation systems had hard-wired electrical control and were very inflexible. This was a robust approach, but it was hard to engineer and maintain these systems in big plants or factories. In order to make these tasks easier, the automation industry developed PLCs

(Programmable Logic Controllers) for managing and controlling devices on the shop floor, and field buses to ensure reliable and real-time capable communication between PLCs and devices. In this way, sensors and actors became easily exchangeable, if the configuration used was retained. Changing the tasks performed by machines was also easier, because now one could load a new program into a PLC rather than change settings on the hardware side.

The more complex automation systems grew, the harder it was to actually configure them. Modern plants have a large number of PLCs, each containing a PLC program that influences the production process. The engineering and implementation of a plant like this is a task that cannot be performed without efficient information management. However, currently the information management for the engineering process is still largely handled by paper interfaces, i.e. handwritten notes and data exchange based on analogue media. This poses a significant obstacle for a swift and efficient engineering process as desired by plant owners. Different approaches to overcome the analogue status quo in automation engineering are discussed in this thesis, namely in chapter 2. The doctoral thesis of Mathias Mühlhause [60] discusses the situation for engineering models in automation in depth.

Part of the engineering models to be handled by the industry are device descriptions (DDs). A device description is a data entity that models attributes of a specific device type with model elements defined by a device description language (DDL). The device description language defines a metamodel for expressing device attributes in accordance with an underlying device model. Device models, as Mühlhause discussed, are engineering models focusing on devices as composites in the overall engineering tasks. In figure 1.1, the DD labeled X is an instantiation of the DDL 1. DDL 1 in turn is based on the device model A, which as part of a system's device models, is also part of its engineering models. It should be noted that a DD is the description for a type of device. Several individual devices may be described by the same DD.

Device descriptions are needed in order to allow engineers to address a device's functions correctly. Many devices have different operation modes or sometimes offer different functions. These functions can be called by addressing the device in a way that is specific to the field bus or similar technology that connects the device to its controller, usually a PLC. The control is usually handled by a set of commands complying with the IEC standard 61131 [7]. In order to unify the addressing of functions in a field device, the standard ISO 15745 [30] offers a framework for application integration, with a special emphasis put on a submodel for device integration. However, despite these standardisation efforts, there still exist several device description specifications.

The automation industry is faced with the problem that all the specifications in existence represent a dedicated market, to which they want to sell their device. Therefore, a device vendor is inclined to offer a device description corresponding to every communication technology he

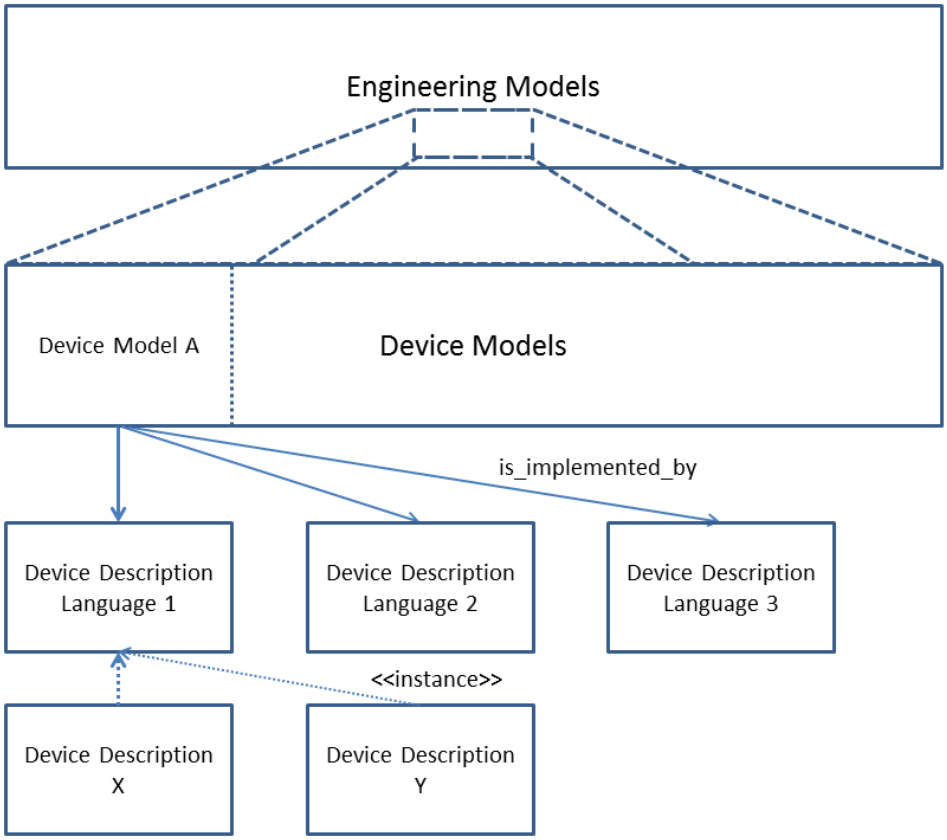


Figure 1.1: An illustration of the Device Description as part of the Engineering Models

supports, in order to ensure the marketability of his products to the corresponding markets. Of course, each device usually needs its own device description. In that way, the size of a device vendor's portfolio multiplies the number of device descriptions that must be generated. For example, the author can personally attest from his work experience that even a specialised automation supplier, that only addresses the discrete manufacturing market, has to provide thirteen different device descriptions for the devices he wants to bring to the market.

The question addressed in this thesis is to what extent the process of generating these device descriptions can be automated. This is a demand that has already been identified by Riedl et. al. [65] in the year 2002. Currently, more than ten years after the demand has been identified, there are proprietary work-arounds that assist engineers in generating these descriptions. The effort on the part of the engineer is nevertheless high and time-consuming. A solution to this would be to find a way of assisting the generation of device descriptions in a way that strips it of repetitive tasks. Taking these considerations into account, this question becomes

Question 1: To what extent can an engineer be supported in the generation of several device descriptions for a single device?

Device descriptions have interdependencies with other information and tools needed to engineer a production plant. The connecting points where a device description ties into other engineering models differ. The communication interface of a device is the connection point with the communication network model that is engineered. The device's function is the main contribution to the automated process. Once the generation of device descriptions is achieved, the topic of using the device descriptions becomes an issue. Device descriptions are used to engineer functions to be fulfilled utilising a communication network. The addressing of functions over industrial communication technologies has created a demand for profiles.

Profiles as defined by the standard IEC 62390 [10]

> define a common set of functionality for a class of devices in a given industrial domain, thus allowing system designers, system integrators and maintenance staff to handle profile-based devices without special tool configuration.

It should be noted that system designers are addressed by this question, as far as the design of a device is concerned. System integrators utilise all engineering models available to create a process instance. This topic is discussed in the thesis by Mühlhause, e.g. [60]. The maintenance staff is another stakeholder that is not discussed as thoroughly as engineering in the automation context so far. It should be noted that works towards product lifecycle management take this task into account, however, e.g. Matsokis in his thesis [59].

The utilisation of profiles for devices with regards to maintenance tasks is often handled under the topic of EAM (Enterprise Asset Management). EAM is an approach to marshal and handle all of a company's assets. Amongst the activities in EAM, one of the most device-centric is the maintenance of an operational system. Not only are sensor devices responsible for acquiring most of the data that can be utilised for state of the art maintenance strategies like CM (Condition Monitoring) and PM (Predictive Maintenance), they also act as the object of maintenance. Faulty parts that need to be replaced in the course of maintenance are often devices, because of the wear involved in their direct exposure to the production process.

The Monitoring and Exchange of a device are both situations where the device descriptions and device profiles can facilitate necessary activities. Condition Monitoring involves access to sensor data and condition information stored in devices. The Exchange of a device calls for the engineering of the replacement part so that it fulfills exactly the same function the old device has had. While replacement simply calls for retaining the engineering information, a device substitution, if necessary, is an engineering task in itself and can benefit from enhanced IT support. The distinction between different maintenance scenarios are discussed in depth by the German central association of the electrical industry in a guideline document [4]. The scenario of substitution promises some leverage for heightened productivity to be had from considering device descriptions and device profiles in the context of CM. Consequently, a second question can be formulated, which will be answered with focusing on the example of CM.

Question 2: How can an enhanced modeling approach for device descriptions and profiles be utilised to improve the handling tasks in the later phases of a device's life time?

In order to answer the two questions given above, this thesis addresses current developments in the automation domain and its overlap with computer science. Special focus is put on technologies for information representation and processing. As is discussed in chapter 4, the application of sophisticated computer-scientific methods is getting a hold of the automation domain in several current approaches. In finding an answer to the questions posed above, an in-depth discussion of current approaches of information processing with regards to field device descriptions is performed. To that end chapter 2 first has a look at the Automation domain, before discussing some technological foundations in chapter 3. Work first described in this thesis is discussed regarding models for certain automation concepts in chapter 5 and regarding an answer and solution to the questions formulated above in chapter 6. Another chapter (chapter 7) evaluates the findings. A conclusion with the discussion of future opportunities can be found in chapter 8.

2 AUTOMATION SYSTEMS AND DEVICES

Automation is a discipline that combines three major branches of science. Mechanical Engineering is used to influence an automated process with technical means. Electrical Engineering is used to make the technical means run without physical interaction by a human being. Computer Science is utilised to plan the running of the automated process according to overlying requirements.

An example for the aforementioned cooperation of technologies may be a discrete manufacturing process of drilling holes into pieces of wood. A mechanical engineer needs to define the specifications of a drill capable of the task and then build a drill that fulfills them. An electrical engineer needs to define the control that moves the drill for drilling holes in pieces of wood. (As well as the transportation of the pieces of wood, which is abstracted away for the sake of our example.) A control engineer needs to define a sequence of hole drillings, possibly switching between drilling programs to move the hole on the workpiece, or order for a switch of drills to produce differently sized holes, according to the orders that have to be fulfilled by the drilling machine. A computer scientist then integrates the control programs of the machine into the overall order processing system of the company.

This simple example illustrates that automation is not so much an insular discipline with clearly defined views on a problem. It is rather a crossing point of different disciplines. Consequently, an important task for an automation engineer is integration. This encompasses the integration of models from different disciplines, each with its own history of modeling techniques.

To facilitate further discussion of automation, what exactly an automation system is needs to be clarified. In his dissertation Lorenz Däubler defines an automation system as follows:

"An Automation system is a technical arrangement that has been developed for the sole purpose of letting an overlying process run towards a set goal without direct human interaction." [21]

A major application area of these automation systems is the manufacturing and the processing of goods for industrial purposes. As this chapter illustrates, a large number of technological advancements has been devised and is employed in order to allow production equipment to let it run towards a set goal without direct human interaction.

The result of the technological advancements is that an automated production plant is, from a computer-scientific point of view, a network of numerous computers, each having very special properties, and each intended to be in use much longer than the average computer system. In addition, complex automated processes need a range of specialised solutions to run efficiently. These specialised solutions demand an integration even between each other.

Some terms will need to be clarified in order to allow an in-depth discussion of the state of the art in automation and the usage of semantic models in this application domain. The first term that needs to be defined beyond the aforementioned automation system is "equipment".

> *equipment*
>
> entity that is stand alone, or interfaces to an automation system, and that performs material processing, material transport, or material storage functions [30]

This equipment can be divided into different categories, one of which is "devices". Since they are important for further discussion, we define devices as

> *device*
>
> entity that performs control, actuating and/or sensing functions and interfaces to other such entities within an automation system [30]

Automation devices are, as recent developments have proven, technical arrangements of a growing complexity. With increasing complexity and functionality of devices, the demand has arisen to better describe their structure. This has been attempted in different places, most notably in ISO 15745 [30]. This international standard defines the inner structure of a device, as will be discussed in more detail in section 2.3.

Despite the approach taken by ISO 15745 of providing a single reference for device integration, this attempt has been regarded as not covering the device manufacturer's needs [35]. To fully

understand where ISO 15745 falls short, it is mandatory to know the term "fieldbus" and the efforts and difficulties to be found in contemporary field bus systems.

Modern fieldbuses are networks that are tailored towards deterministic communication, in order to allow reliable production processes. This is the reason why standard Ethernet and TCP/IP with its non-deterministic nature was not instantly adopted as a fieldbus. Ethernet-based solutions are still in the minority, though their numbers are increasing in comparison to other fieldbuses, but when used for manufacturing they often run protocols more or less different from TCP/IP. Existing fieldbuses are heterogeneous in how they access devices. This leads to the situation that device manufacturers need to develop specific hardware and software if they want to provide devices with connectability to different fieldbus systems. This is no different from the development of standard network interfaces, but with several fieldbuses in use, and a smaller market, the task is a significant effort for device manufacturers. Figure 2.1 shows an example of a single device being sold with two different fieldbus connectors, derived from an actual hardware setup.

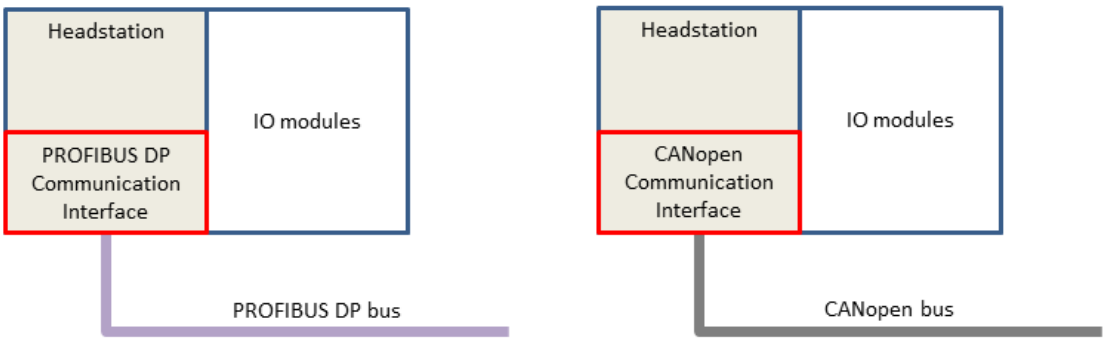


Figure 2.1: Schematic representation of IO Devices with different fieldbus connectors

Not only the hardware of the two devices illustrated differs. The underlying drivers and even the device descriptions provided by the device manufacturer have different formats. Nevertheless, the automation function, carried by the pluggable IO modules, is the same. A quote from the industry shows where the situation is nowadays:

> Quotes by Christoph-Albrecht Winter, head of Tools and Infrastructure division of Custom Solutions department, Festo AG & Co. KG, Esslingen, Germany, 09.12.2011
>
> "Mechanical Engineering Information and Communication Interface Information are increasingly influencing each other."
>
> "The effort for the development of digital information for devices and its accessibility is increasing and has by now reached 50% of the total development cost of a device."

According to the quote, device information management is a non-negligible cost factor for device manufacturers. Other authors, too, state the heterogeneity in automation being a challenge [72]. To fully understand the challenges inherent in this task, some preconditions found in the automation domain need to be analysed. The following sections go into the details of specific automation technologies that are state of the art today.

2.1 THE HIERARCHY OF AUTOMATION

For some time, automation systems have shown a distinct hierarchy. Due to their adherence to either electrical engineering methods or computer scientific methods, a level close to the production process and a level close to what is commonly referred to as the office world can be distinguished. An example of the pyramid shape often used for this hierarchy can be found in Polke's book on process control engineering [64], second german edition, page 536, figure 8.2-1, e.g. A slightly more recent adaptation can be found in the definitions of IEC 62264. The IEC standard 62264 [9], also known as ISA S95, structures automation systems by defining a reference information model for various manufacturing plant operations. The full name of IEC 62264 is "Enterprise-control system integration" and three parts of the standard have been released so far, with three more parts under development at the time of writing of this thesis.

In IEC 62264, automation is classically structured in different levels. These levels follow an abstraction from the concrete production process.The lowest level, level 0, is the production process itself. Level 1 describes the devices, which are close to the process in that they have a direct connection to it. Typically this means that a device on level 1 is either a sensor, an actor or both. Level 2, in contrast, is detached from the process and covers the control layer. Apart from communication infrastructure this also encompasses Programmable Logic Controllers (PLCs), which handle the actual process control in state-of-the-art automation systems. The next level of abstraction, level 3, covers what is typically performed by Manufacturing Execution Systems (MES). Production scheduling, the sequencing of production orders, is performed on this level. Level 4 is the enterprise level, where Enterprise Resource Planning (ERP) systems are allocated. Tasks such as resource logistics and financial administration are handled on this level.

A classical automation pyramid can be seen in figure 2.2. The pyramid layout was chosen by early authors because typically, there are more heterogeneous devices on the lower levels on the pyramid, and the ERP level was handled by a monolithic solution. The figure shows exemplary systems to be found on each level. Each industry has a slightly different setup and utilises differently named systems for each level below the MES layer. Level 0, the production process, has been omitted in the figure because a generalised system for this layer cannot be easily defined.

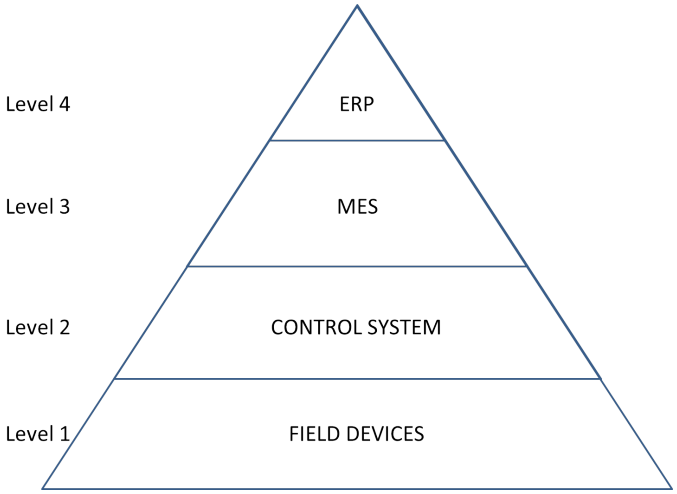


Figure 2.2: Systems on the levels of the classical automation pyramid

The numbering of levels discussed earlier is derived from IEC 62264 [9]. However, the standard does not use a pyramid for illustration purposes. Due to the focus of defining functional interfaces between the MES and ERP levels, a different kind of illustration is utilised, as shown in figure 2.3. IEC 62264 defines activity categories which are then allocated to either level 3 or level 4 of the hierarchy. It defines ways to build a production schedule and to determine the resources available for a production task. For example, scheduled production equipment availability is calculated by taking into account production capability information, capacity scheduling information and maintenance information, because for a certain task, a piece of equipment must be capable of performing the task, must be unscheduled regarding other tasks in the relevant time and must not be taken down for maintenance.

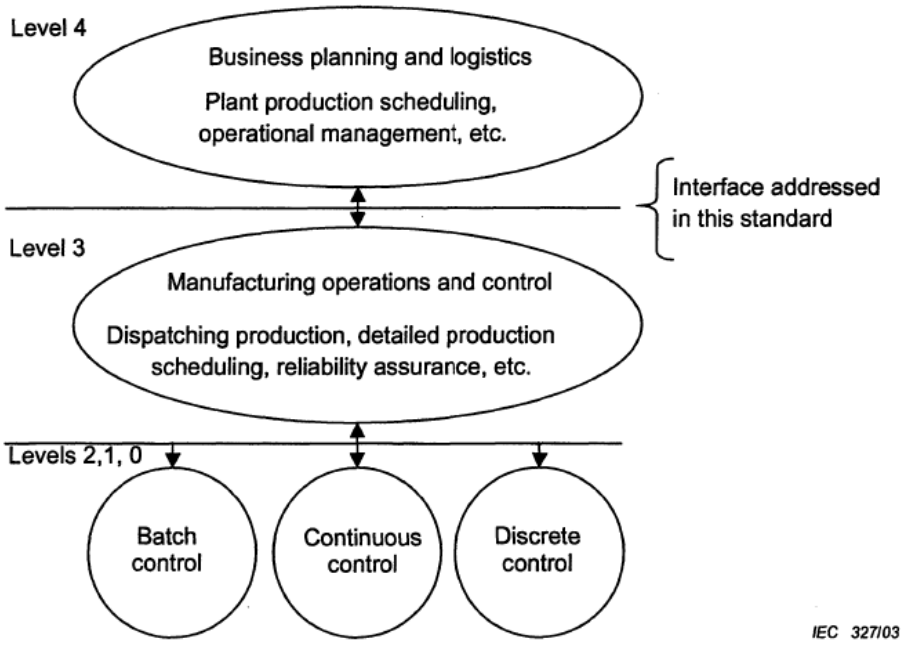


Figure 2.3: Hierarchy of Automation according to IEC 62264 [9]

In addition to defining the calculations already discussed, part 1 of IEC 62264 also defines several models that are meant to facilitate the processing of information on production-relevant entities. Extensive definitions for data exchange between different production activities is given in its appendix. Part 2 of IEC 62264 lists the attributes of the model elements of the models given in part 1. Part 3 provides activity definitions that follow a fixed schema of information exchange and are meant to reduce the risk in implementing these activities in concrete systems.

2.2 LIFE CYCLE OF AUTOMATION ENTITIES

Each entity that can be observed exists in a certain period of time. The duration of time is so universal, that Matsokis decided to take it as a starting point for model generation [59]. When technical systems are considered, the duration of time in which their existence is acknowledged is called the life time of the system. When a system is seen as a type of product, it will evolve over time, new revisions and improvements replacing faults or shortcomings in older instances of the type. This cycle of time spans of types is called the life cycle. The life cycle of a system is a view on the system classifying each point in time that the system is in viewed as belonging to a fixed set of life cycle phases. In general, these life cycle phases when applied to an instance are considered mutually exclusive and sequential, but there do exist overlaps as well as interleaving phases.

Different domains have produced different life cycle phase models. An important commonality of the models is that the life cycle of an entity typically starts before its physical existence, because it does not come into being spontaneously, but rather is the product of intention and forethought. Therefore, the design and the planning of a system are typical phases early in or even at the beginning of its life cycle. Once designed, an entity can be instantiated. In complex technical setups, solutions often need a certain degree of engineering. This process of customisation is another phase often found in life cycle considerations. Afterwards a system needs to be set up for the beginning of operation, usually through a phase of installation and/or commissioning. The fulfillment of the functional purpose of a system is then performed under the phase of operation or a denominator with a similar meaning. In sophisticated systems, operation needs to be interleaved with phases of maintenance and/or optimisation in order to ensure a time of operation as long and efficient as possible. At the end of a system's life time, the life cycle phases of decommissiong and/or recycling can be found.

The abstract view of the life cycle given above becomes easier to understand once the systems whose life cycle is considered are narrowed down. One example of the application of life cycle management with various scientific achievements in recent years is the field of a production plant's life cycle. Especially the process and power industry is faced with the challenge of

running and maintaining systems for decades after their initial installation [4] [60]. This has lead to approaches that try to improve the planning and engineering process as well as maintenance in process industry plants. Doherr and Urbas have devised a system for cost estimation for fieldbus installations [26] [71]. Strube et. al. try to decrease the engineering effort in modernisation scenarios [72]. Frank et. al. propose a workflow for the design of distributed automation systems [33]. Mühlhause et. al. have discussed an approach for utilising semantic models for plant engineering [61]. Vogel-Heuser et. al. have proposed a modeling approach for time-behaviour verification of networked automation systems [78].

Another application field of life cycle management is product life cycle management (PLM). The main difference to other application areas is that PLM considers the responsibilities not only of the user, but especially of the producer of a system. From this point of view, the consistency and maintaining of information over the whole life cycle becomes an issue. The thesis by Matsokis [59] is an example on the effort being put into this field. Dafeng et. al. have discussed a closed loop approach for the product information [82]. Anke et. al. presented a middleware for accessing product information that is accessible through information stored on the product itself [2].

Compared to the two aforementioned application areas of life cycle management, the life cycle of a device shows considerable intersections. From the plant owner's point of view a device is an asset that is part of the plant, and therefore plays a part in the plant's life cycle. From the device manufacturer's point of view the device is a product and therefore subject to PLM. The combination of these two viewpoints does not fully cover the challenges of device life cycle management, however. For example, device engineering differs from plant engineering in that a device's function can still be influenced by its eventual application. While a plant's application is usually defined before engineering, a device needs to be flexible enough to allow a certain degree of customisation by design. When a plant is subject to maintenance, parts may need to be exchanged. When a device is subject to maintenance, it can be exchanged itself. The device differs from the generic view on any product by the common attributes all devices share that go beyond PLM aspects. A discussion of opportunities in device design can be found in Hahn et. al. [42]. A discussion of current maintenance strategies for devices is included in Gössling et. al. [36].

2.3 INTEGRATION OF AUTOMATION DEVICES

ISO 15745 [30] is an international standard fully named "Industrial automation systems and integration - Open systems application integration framework". It consists of several parts, of which the first, "Generic reference description", defines a framework on which the other parts

rely. The premise of the standard is that for automation systems, application specifications are generated. The goal of the standard is then to provide an "application integration framework (AIF)", which facilitates the structuring and further development of application specifications. Especially the interfaces of an automation solution should, according to the standard, be defined as "application interoperability profiles (AIPs)", which follow the rules and profile structure defined in ISO 15745. Subsequent parts of ISO 15745 specialise the profile structure and rules given in part 1 and define "technology specific extensions" to the AIF.

The main concern of ISO 15745 is the profile, which is defined as a means to structure and communicate requirements regarding all aspects of an automation system. In the standard, the main classes of profiles are Resources Profiles, Process Profiles, Information Exchange Profiles, and Application Interoperability Profiles. Resources Profiles can either be Communication Network Profiles, Device Profiles, Human Profiles, Material Profiles, or Equipment Profiles. The standard illustrates its context as seen in figure 2.4. As can be seen in the figure, profiles are understood as a conglomerate of rules set by standards, and contents set by the "real world application system", which is meant to denominate the application context of the concrete system to be implemented.

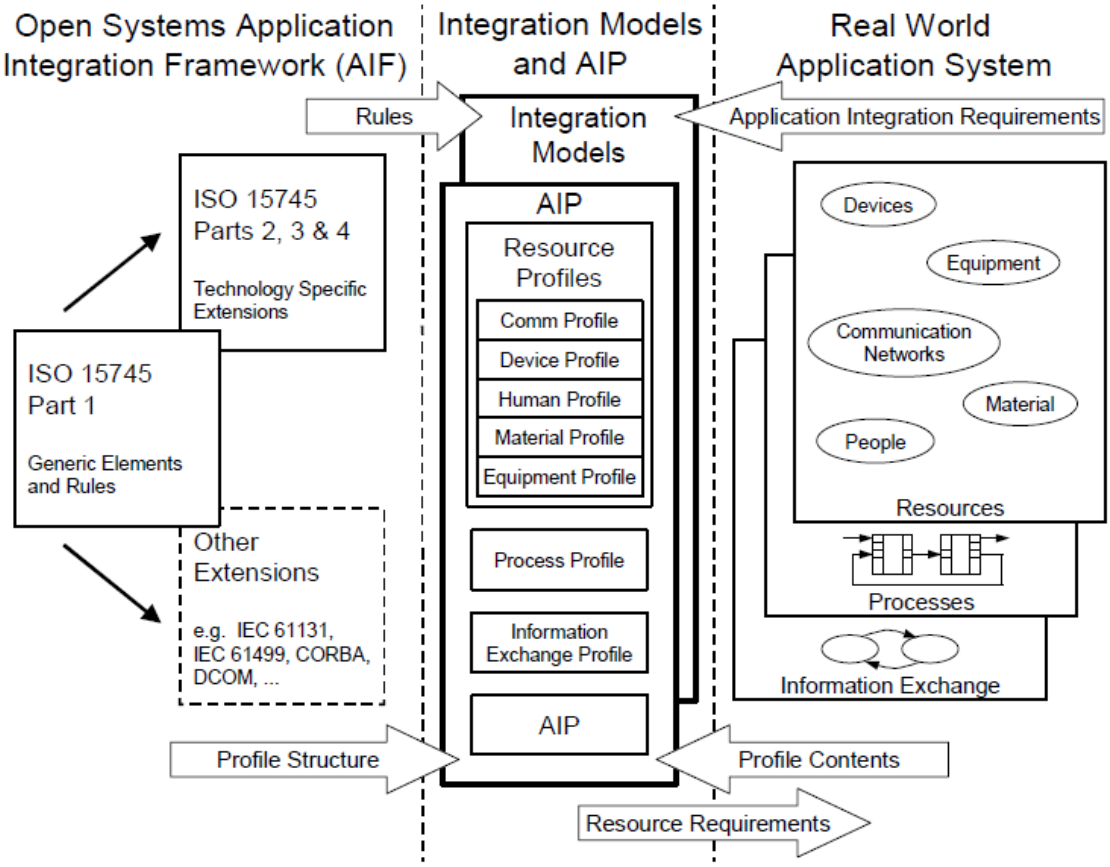


Figure 2.4: Context of ISO15745 according to the standard [30]

ISO 15745 standardises an approach for using the AIF in the generation of AIPs, who are meant to facilitate the implementation of the system that solves the given application context. There are five domains that are meant to be handled by the approach, namely

- Device
- Communication Network
- Equipment
- Human
- Material

For all these domains, profiles are set to be the result of joining together the demands of the according integration model and the specifications of existing solutions. Based on the information gathered in the approach, three models are to be generated:

- Process integration model
- Information exchange integration model
- Resource integration model

The last of these is split into five specialised models for the domains listed before.

An important part of the definitions of ISO 15745 is the definition of profile templates and types. Different profile templates are derived from a master profile template. This relationship is defined as an inheritance similar to object oriented programming, and also illustrated in an UML manner of a specialisation relationship. The profile templates speciales from the master profile template are called generic. These generic profile templates are further specialised with technology specific profile templates.

The master profile template defines two components of a profile, the header section and the body section. The master profile template defines attributes for the header section that contain meta information on the profile. The body section is left to the information payload the profile is built for. An XML representation for the information on the profile is given as well. The different generic profile templates listed in the standard are:

- Generic AIP Template

- Generic Process Profile Template
- Generic Resource Profile Template
- Generic Information Exchange Profile Template
- Generic Device Profile Template
- Generic Comm Network Profile Template
- Generic Equipment Profile Template
- Generic Human Profile Template
- Generic Material Profile Template

The resource profile is of special note, because it is a composite of a number of the latter five types of profiles. Only a device profile is mandatory in this composite. Of special concern for this thesis is the device profile template. The overall composition of a device profile according to ISO 15745 is reproduced in figure 2.5. As can be seen, a device profile contains at least one device function element. One device identity and one device manager are optional, as is any number of application processes.

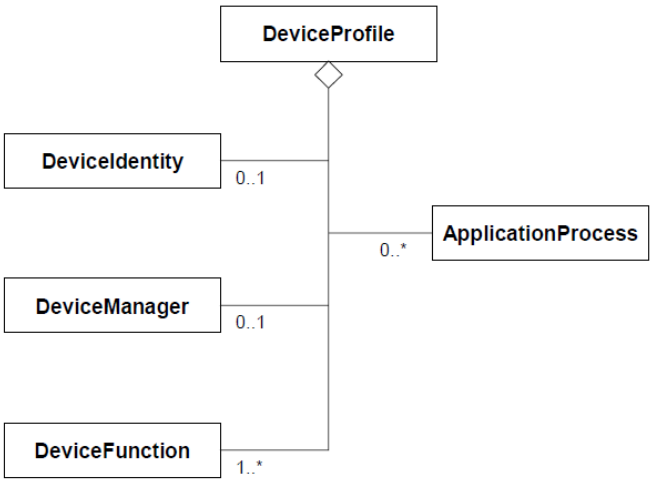


Figure 2.5: Reproduction of the Composition of the Generic Device Profile according to ISO 15745 [30]

Examples given for content of the device identity, which should help to uniquely identify the device, are:

- manufacturer's identification
- part number

- revision
- location of storage of additional information
- indication of the number and type of additional objects within the device

It should be noted that this list of examples is not meant to be exhaustive. The device manager contains attributes that are used to control the device's operation modes. The device function deals with the specific purpose the device is integrated into the automation process for. It aggregates attributes that are related to this automative purpose of the device. Finally, the application process object aggregates attributes and services that fulfill requirements imposed by the application process the device is used for.

The technology specific parts of ISO 15745 offer more details on the information that has to be stored in device profiles for these technologies. Part 2 of the standard defines a profile template for the technologies *DeviceNet* and *CANopen*. For CANopen, the composite elements of the device identity defined in Part 2 of ISO 15745 are reproduced in figure 2.6.

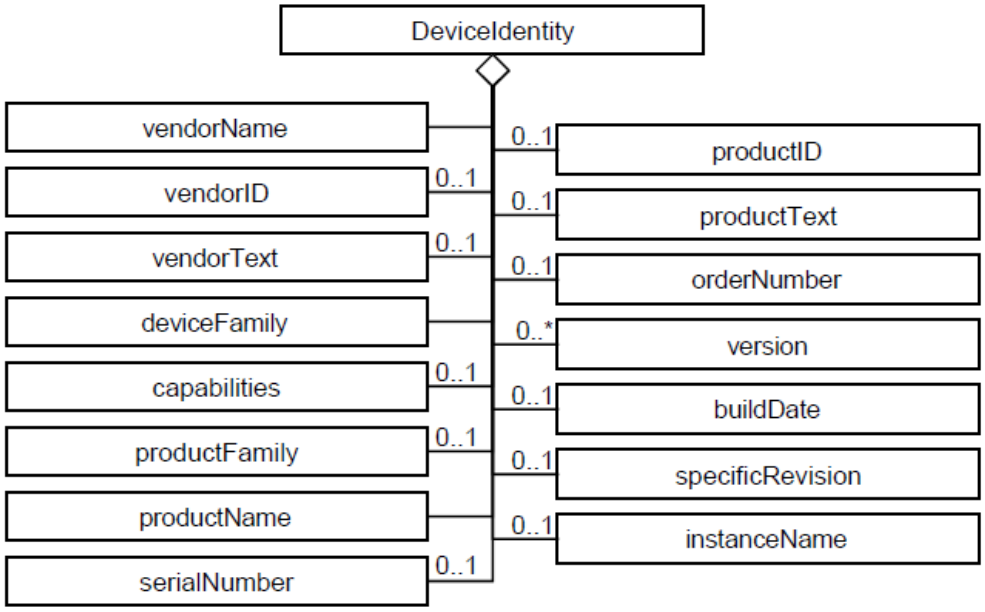


Figure 2.6: Reproduction of the Composition of the Device Identity for CANopen Devices according to ISO 15745 [30]

In Part 3 of the standard, GSD and EDDL are identified as PROFIBUS specific legacy standards for device profiles. This assessment of the legacy state of these technologies is not warranted by the state of the art in the automation industry.

Based on ISO 15745, the IEC technical report 62390 [10] defines a template for device profiles to be used in the overall engineering approach of the ISO standard. The report elaborates on the

responsibilities in device profile generation and goes on to define a methodology, with which device profiles should be generated. To this end, a device is decomposed into resources, functional elements and parameters, which in nested composition make up the device's structure. The resultant class diagram is reproduced in figure 2.7.

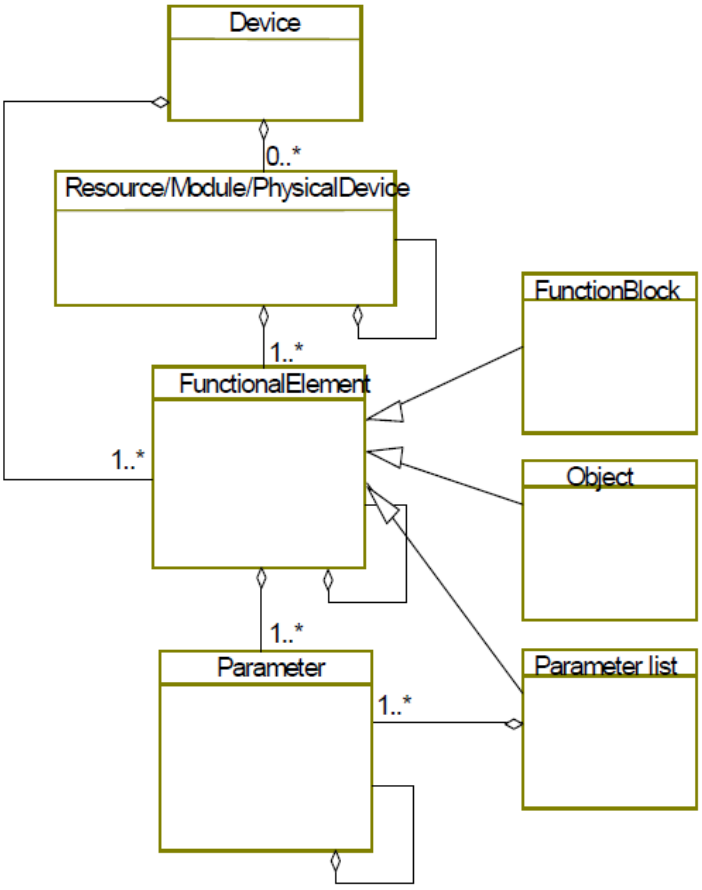
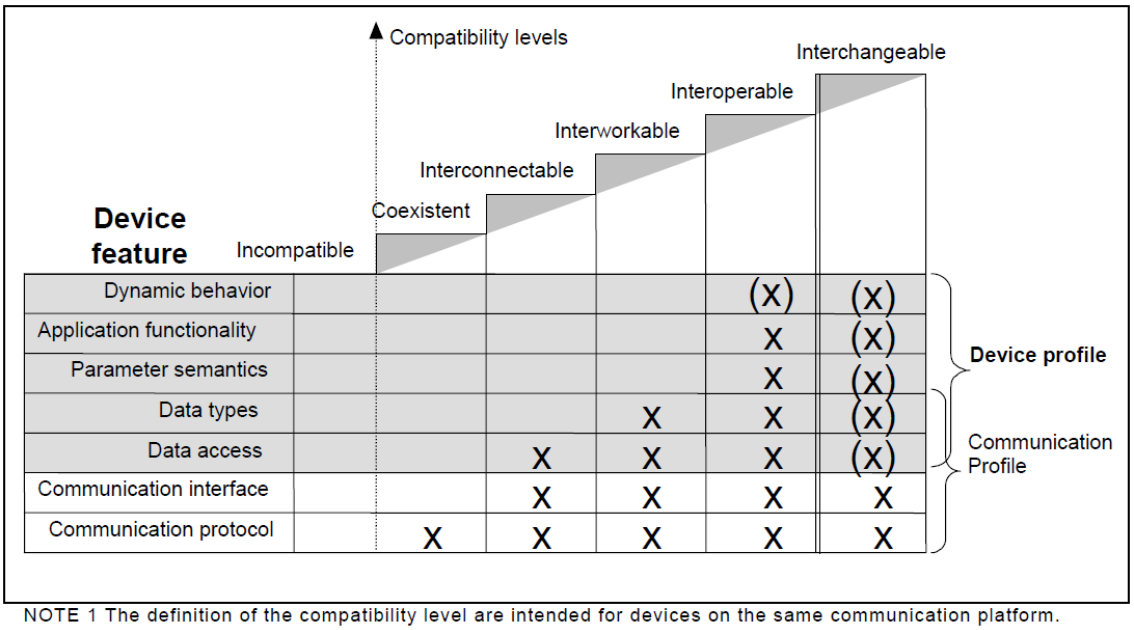


Figure 2.7: Reproduction of the Exemplary Device Structure Class Diagram from IEC TR 62390 [10]

The standard also takes a look at the composition of a typical automation system. It then defines the steps that have to be taken for profile generation according to the report. The steps defined in the report are:

1. Scope, compatibility levels and device classification
2. Definition of device functions and their relations
3. Parameter list definition
4. Grouping of functions to functional elements
5. Device behaviour description
6. Extensions of existing profiles

The first step is intended to define what the profile is meant to express. The compatibility levels mentioned are defined, for functional compatibility, in a table that has been reproduced in figure 2.8. Different features that are available between two devices are classified into different levels like interworkable, interoperable, etc. Each level is described in detail in the report, but here a reproduction of the overview table will suffice. It should be noted, however, that interchangability differs from the other classes of compatibility. Interchangeability does not focus on the coexistence of two devices, but rather on the question if a device could be replaced by another device. The concept thereby breaks with the hierarchy given by the other concepts, as indicated by the double line.



NOTE 1 The definition of the compatibility level are intended for devices on the same communication platform.

NOTE 2 The communication profile is not in the scope of this guideline

Figure 2.8: Reproduction of the Functional Compatibility Levels from IEC TR 62390 [10]

The second step of defining functions and relations between them is handled in a top down approach. No list of device functions is standardized. The third step is defining those parameters of the device, that should be accessible from outside the device. Different methodologies for the definition of the parameters are proposed, amongst them use case considerations and derivation from the device functions. For the fourth step, the ustilisation of models well proven for the automation domain is proposed. For the fifth step, different methods for expressing behaviour, like mathematical algorithms or a state machine, e.g., are given. The sixth step is optional and involves existing profiles that need rework.

In addition to the profile generation process, a profile template structure is given. The recommended sections for a profile are the header section, the parameter list section and the device functional structure section. It should be noted that no details for the header section are given. However, an annex of the report lists recommended features of a profile header for future harmonisation attempts. Another annex discusses the aspects of a device's life cycle for profile generation, comparable to section 2.2.

2.4 SPECIFIC DEVICE DESCRIPTION STANDARDS AND SPECIFICATIONS

ISO 15745 provides an accepted framework for device descriptions in the automation domain. However, all specific technologies supplement the definitions given in the standard with specific device description languages. In order to practically use a specific fieldbus technology, a device manufacturer needs to work with these device description languages.

The technologies described in this section do by a wide margin not cover all available device description languages in the automation domain. Due to the great number of different formats and languages available for different domains, a subset has been chosen that by the author's personal experience is confirmed to be in current use on the side of device manufacturers.

PROFINET, that is present through the GSDML language, had an installation rate of 4.3 million devices in 2011 [50]. It is stated that

> PROFIBUS is the world's most successful fieldbus with more than 40 million devices installed by the end of 2011. [49]

This has to do directly with the GSD language, that is used as a device description for all PROFIBUS devices.

Sources for application numbers of CAN are hard to come by, because CAN is used in a wide variety of application fields. Having originated in the automotive sector, CAN is applied in many different fields of industry. It is also used within capsulated systems, such as between controllers and converters, e.g. The wide application range of CAN can illustrate why considering a CAN relevant technology such as EDS is relevant.

2.4.1 Electronic Device Description (EDD)

The Electronic Device Description Language (EDDL) is a technology that has been developed to facilitate integration of field devices in field bus systems. Particular focus has been laid on the visualisation of process data during run time. The structure of the language has been modeled closely to look alike to the syntax of the programming language C. Therefore, definitions of variables and data structures are a core component of the EDDL. Dynamic effects such as the information exchange between a device and an engineering station (usually a computer) take another big portion of the language. It should be noted that EDDL is the name of the device

description language. A specific file written *in* EDDL is called an Electronic Device Description (EDD). A comprehensive overview of EDDL is given in the book by Riedl et. al. [65]. Only a few key elements will be discussed here.

Like the programming language C, EDDL allows the usage of preprocessor commands. Constants and makros can be defined in order to ease the definition of variables and structures. Other files can also be included to use the definitions made there. One thing that clearly distinguishes EDDL from most other device descriptions is the ability to define conditional interpretation of parts of the file. This can be utilised to change information about the device dependent on information that is read from the field bus, e.g. That way, one file can be used for different possible configurations of a basic device.

Apart from basic identification information, EDDL has few keywords that have a set semantic. The language is flexible and is based on the definition of variables and structs. In addition, EDDL offers mathematical operators just like a programming language does. The structs that can be defined in an EDD are arrays, collections, item arrays, records and variable lists. Other structures exist for the communication to and from the field device from the perspective of an engineering tool that interprets the EDD. This engineering tool is also in the focus for the menu struct that aggregates elements, who all have to have a label for display in a menu structure of the tool.

EDD is also used in the FDCML (Field Device Configuration Markup Language) specification, that aims to provide a meta format for device descriptions based on exisiting standards. [40]

2.4.2 GSD/GSDML

Amongst the fieldbus systems that are used in automation are the PROFIBUS and PROFINET bus technologies. Both of these technologies have application specific subtechnologies. They are maintained by the consortium PROFIBUS & PROFINET International (PI, in Germany represented by PROFIBUS Nutzerorganisation) [45]. This organisation has published specifications for DDLs that are to be used for the configuration and engineering of PROFIBUS as well as PROFINET networks. The technologies for the DD in these two fieldbus technologies are named GSD (General Station Description) and GSDML (GSD Markup Language), respectively. Specifications for both technologies are available through the PROFIBUS & PROFINET International organisation.

GSD

The General Station Description is a text based data format that allows the storage of vendor and identification information of a device as well as capabilities and addressing parameters [46].

It follows the general layout of Windows INI files. A keyword is used to identify pieces of information semantically. The symbol "=" shows that a keyword is over and an assigned value follows. The assigned value has a data type specified for each keyword. A ";" symbol is used to end a value and declare everything else written in the same line to be a comment not to be evaluated when parsing the file. A new line allows the specification of the next keyword.

With this comparatively simple base syntax, a wide array of keywords is defined that allows storing device information. Each keyword can be mandatory, optional, default (meaning if it isn't specified, a default value is used) or group (which means that of all the keywords belonging to one group, at least one needs to be present). The keywords are then classified as being either general specifications, master-related specifications or slave-related specifications. The differentiation of masters and slaves is an attribute of the PROFIBUS fieldbus paradigm.

Listing all the keywords available for GSD in this thesis would be of little use to the reader, since this information can be gathered from the referenced sources quite easily. Therefore, some examples will be presented in order to explain the usage of a GSD file. Every GSD file has to specify the GSD Revision it adheres to, the name of the vendor and of the device, identification numbers, station types and general information like that which is necessary to assign a file to a device as well as get fundamental information on the device's intended purpose. Depending on whether the device is a master or a slave, several support tokens are stored as boolean in order to tell if the device supports the functionality in question. For usage with engineering tools, graphical files can be linked in order to display the device as intended by the manufacturer. Maxima of telegram lengths or plugged submodules can be stored as well as descriptions for the bit coding of diagnostic data telegrams.

Via the keyword Module a list of possible modules with their IDs can be given. This allows engineering tools, e.g., to identify a module properly despite the fact that it is not know which module will be present at the time of the generation of the GSD. GSD files are meant to be parsed by software tools in order to acquire information about devices and their capabilities where needed. To this end, the specification contains a formal deduction rules list for the GSD language. GSD files are shipped with the device, and while they usually contain information about how to read process data from a device, they do not contain process information, which needs to be accessed from the fieldbus directly.

GSDML

The GSDML format is actually called PN-GSD by PROFIBUS & PROFINET International [48]. It fulfills the same function as a standard GSD, but is intended for PROFINET devices. There are more differences, however. The most obvious one is that GSDML is an XML dialect. Instead of

keywords, that are used in a GSD, a GSDML is structured by XML tags. Due to the nature of XML [79], this leads to encapsulated information transportation. The other defining factor of GSDML is that it follows the definitions given for device profile structures in ISO 15745 (see section 2.3).

In accord with ISO 15745, a GSDML consists of a profile head, profile body and possibly a signature. The profile head is defined in the GSDML specification, while the profile body is defined to be an aggregate of the data structures called *DeviceIdentity*, *DeviceFunction* and *ApplicationProcess*, the first of which can be seen in figure 2.9. Each of these further decomposes into data structures that become attributes at the leafs of the XML structure tree. The substructures of a GSDML are based on the concepts of lists and items. Lists are contained in data structures to aggregate groups of similar data elements, which themselves are stored in items.

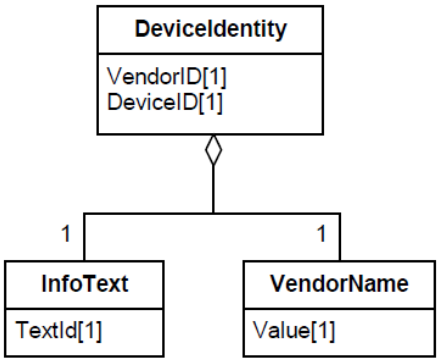


Figure 2.9: Excerpt from the class structure for Device Identity from the GSDML specification [48]

There are several types of lists and items in a GSDML. Device access points, modules, submodules, interface submodules, port submodules and parameter record data (supplemented by F_parameter record data), channel diagnosis and unit diagnosis type are the most complex structures that are given a list as well as an item in the GSDML file structure. (The F_parameter record data is meant for safety purposes and is not focused on further.) Module info are also stored as complex aggregations of other data structures. Data elements are defined with a set of XML-specific data types, partly defined by the W3C (the world wide web consortium, who standardized XML), partly defined for GSDML. The identification of devices, modules and submodules in a GSDML should adhere to the regulations defined for the PROFINET specification that is used.

2.4.3 CANopen

CAN (Controller Area Network) is a technology used in real time communication systems such as vehicles, medical equipment and also factory and process plant automation. The consortium

CAN in Automation (CiA) has released specifications and profiles under the name of CANopen, in order to facilitate and promote the usage of CAN in the automation domain. CANopen communication is handled through Process Data Objects (PDOs) and Service Data Objects (SDOs). PDOs are typically identical with cyclically exchanged data, whereas SDOs have to be requested to be communicated. These objects are linked in an object dictionary, which for each device gives the addresses of the objects that can be accessed.

The object dictionaries of CAN devices are typically defined by a device profile, several of which have been defined for the different application domains of CAN. The concrete device is described by an EDS (Electronic Data Sheet). The EDS fulfills the role of a device description for CANopen devices. The EDS for a CAN device can be provided either along with the device on a separate storage device or can be provided as part of the information stored in the device itself.

As a complement for the EDS, which only stores static information about a device, CANopen expects a Device Configuration File (DCF) to hold all instance-related information, such as the values with which the variables defined in the EDS are set for an individual device. A node ID for an individual device is expected to be stored in the device's DCF.

EDS files are ASCII files, much like classic GSD (see section 2.4.2). Different sections of the file are separated by a line giving the section name, enclosed by the bracket symbols "[" and "]". Afterwards, key value pairs are listed, divided by the "=" symbol. EDS only allows full lines as comments, which have to start with the semicolon symbol ";".

EDS files contain a section named [FileInfo], which reflects on the properties of the EDS itself. In this section, keywords are defined that give information on the file name, revision, EDS version used, creation and last modification info and the like. Of higher importance for later usage in this thesis is the section named [DeviceInfo]. Here information about the device vendor and the device itself is stored. Following are three sections that contain the object dictionary of the device. The section names specified for these sections are [MandatoryObjects], [OptionalObjects] and [ManufacturerObjects]. Each of these sections is to be started with the key "SupportedObjects" which is to give the number of objects defined in the respective section. Those objects are then to be given their own sections with according section names.

The DCF differs from its template EDS insofar as it contains "ParameterValue" entries for the defined objects that specify the conrete value set for an object in question. The additional section [DeviceCommissioning] contains information from the commissioning of the device, as the name indicates. Instance information becomes especially relevant for devices that support pluggable submodules. The EDS of such devices can list a number of possbile pluggable submodules. These extension modules get their own description. For the DCF, the real setup of the extension modules is written into the DCF, adapting the EDS to the individual device.

In addition to the classical EDS format, there is an XML format defined for CANopen device descriptions. The XML specification [44] for CANopen defines an XML structure that is based on ISO 15745 [30]. The device identification structure for the specification is shown exemplarily in figure 2.10.

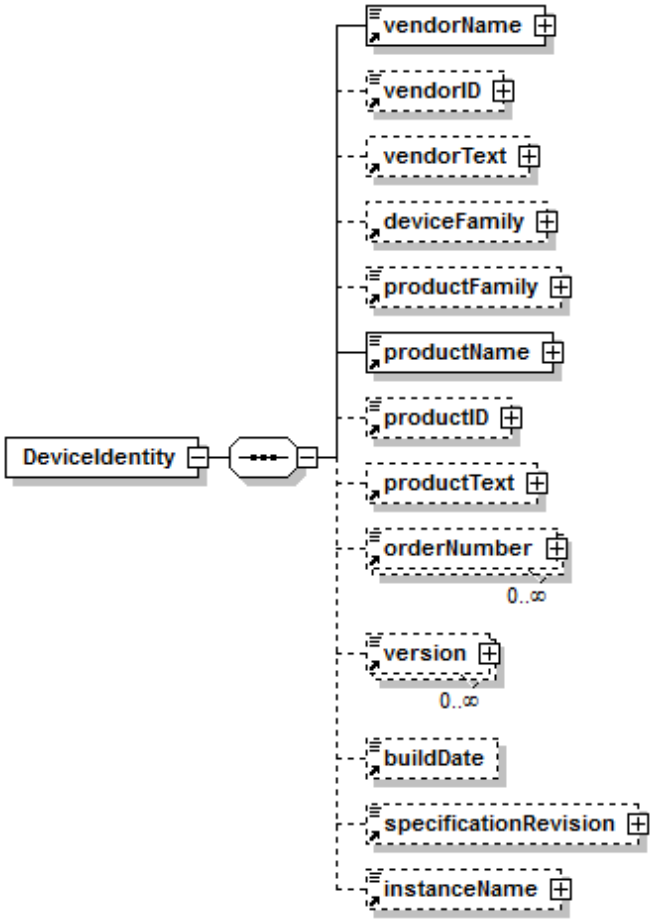


Figure 2.10: Reproduction of the Device Identification Element for CANopen XML [44]

While the CANopen XML specification is easier to map onto other ISO 15745-compatible XML device description languages, it is not as widely used as EDS at the time of the writing of this thesis. The lack of tool support is one major reason for this phenomenon. Therefore, the focus with regards to CANopen in later chapters will be laid on EDS.

2.4.4 OPC Unified Architecture (OPC UA)

OPC stands for the attempt by partners of the IT and automation domains to establish a shop floor connectivity standard. The original meaning of OPC was OLE for Process Control. The acronym indicated the usage of OLE and hence the proximity of the specification to the Microsoft operating system technology. The Microsoft focus was lessened over time and today the acronym is transscribed as Openness, Productivity and Collaboration. The first OPC

specification, later known as OPC DA for Data Access, used the COM interface of the Microsoft operating systems to establish a connection to automation systems that were running a special OPC server [51]. After two more versions of the OPC DA specification and the extension with various technologies (OPC XML, OPC Alarms and Events, e.g.) the OPC foundation, an independent user organisation that develops, distributes and promotes the OPC specification, put significant effort into uniting the different specifications in one flexible but powerful approach. Out of this effort, OPC Unified Architecture (OPC UA) was created.

OPC UA heralded a broader approach to shop floor connectivity. Other than the OPC DA specification, which allowed servers, groups and items as modeling elements, OPC UA allows users to specifiy the information model of their servers flexibly [58]. The expansion of the basic information model allows for any shop floor system to be modelled in the OPC server by user demands. This flexibility has led to several approaches where established standards are transferred to OPC UA server information models.

An example of the application possibilities for OPC UA is the utilisation of OPC servers that are already given in the MES and the shop floor domain and to employ an OPC client to send data from one server to the other [77]. In addition, work is underway to define a MES reference model for OPC UA servers to use as an information model for the OPC UA server from a working group inside the OPC foundation.

OPC UA is also the base technology for other approaches such as FDI (Field Device Integration), which is a joint effort of several automation specification bodies to formulate a single approach towards device integration [19].

2.4.5 PROFIBUS PA profile

While it is not a device description language per se, but rather a set of device profiles, a few words are given here to discuss the PROFIBUS PA profile for devices [47]. The device profile defines an application process for devices that allows addressing an application function on a level above the purely transmission oriented fieldbus protocol. The relationship is illustrated in figure 2.11.

The profile defines Blocks for structuring a device's functionalities. These blocks are further specialised into Function Blocks, Transducer Blocks and Physical Blocks. These represent the according aspects of a device, structured into blocks of variables and parameters. Parameters can be differentiated into contained, input and output parameters, depending on their accessability outside their block. Parameters can also be grouped into view objects to allow

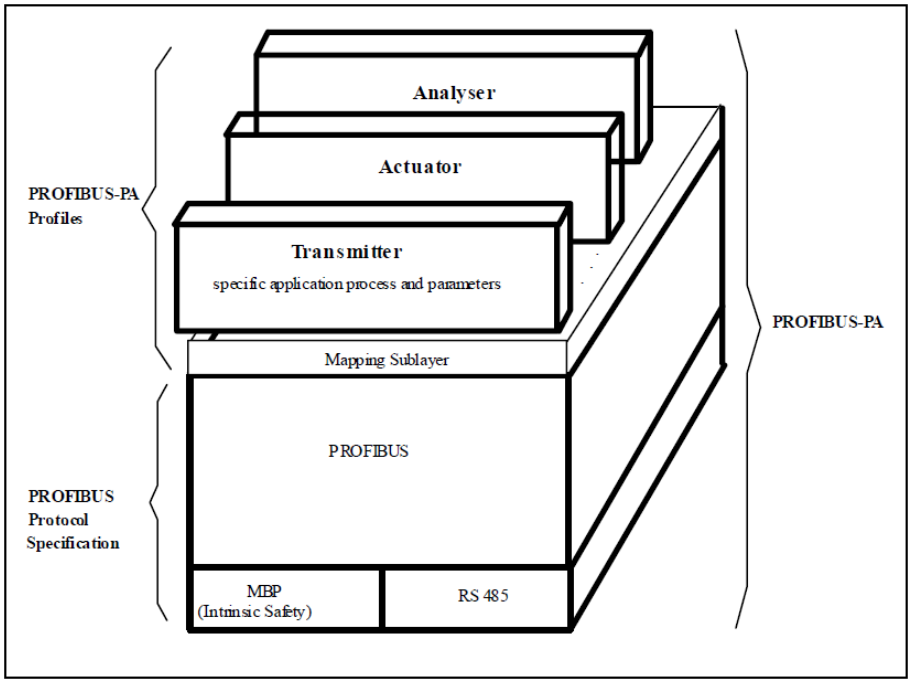


Figure 2.11: Reproduction of the Overview of device profiles for PROFIBUS PA [47]

accessing them as a group. Parameters contain a set of parameter attributes associated with each of them via a parameter attribute table.

The profile defines a set of standard parameters as well as block mapping tables. Listing all the qualities described in the profile would go beyond the scope of this consideration. Automation engineers can use the profile to define the communcation behaviour of a PROFIBUS PA device. This can be a great help when tasked with integration for such devices. The usage of the profile does leave the challenge to fill the mapping tables, however.

2.5 INTEGRATION TOPICS

The overview given in this chapter is just a small excerpt of the different formats, languages and technologies that are in use in the automation domain. While availability of data was an issue in the first place, the number of approaches to provide information has spawned a heterogeneity that goes beyond what most companies involved in the automation domain can handle. Even big companies usually focus on a subset of approaches or try to support the unification of different approaches.

The overall situation in the automation domain described here makes it apparent why integration efforts are a constant topic considered for improvements in automation technologies. In this situation, automation technology providers such as automation device manufacturers have to put

a lot of effort into addressing all technologies that are relevant for their target markets. Lowering the heterogeneity by utilising integration approaches not only decreases costs, but offers the possibility to access new markets.

Easing the integration of different technologies can also help to let technology providers utilise more sophisticated approaches. For example, condition monitoring has been an issue in the automation domain for years [39], [54]. Device manufacturers, however, had the problem of transporting the information gained from condition monitoring through different technologies. Here, too, integration approaches promise to leverage the business outlook of automation technology providers [36].

For performing the integration tasks identified here, sophisticated approaches from the computer science domain are often investigated. For this reason, the following chapter looks at a particular subset of these models, whereas chapter 4 will discuss the application of these models in the automation domain in particular.

3 SEMANTIC MODELS

Computer Science is reliant on the abstraction of existing concepts, in order to make them computable. Modeling of complex technical systems and processes has been investigated thoroughly [70]. Regarding automation, model theory has been applied for example specific to the process industry domain [43]. This chapter provides an overview of the kind of models that can be used for solving complex tasks that stem from the situation described in chapter 2.

Modeling always bears an abstraction of the entity modelled. There are elaborate discussions available regarding the relationship between a thing, its model and the meta models of the model [1]. Here it shall suffice to keep in mind that every model is based either on a meta model or on unexpressed but known associations. Either base for a model assigns *semantics* to the model in question. The semantics of a model give meaning to it, that is the model is only understandable when the semantics are known in one way or the other.

With the importance of semantics in mind, it becomes apparent why scientists, especially computer scientists, have developed methods to express the semantics of a model inside its model. This allowed for the term *semantic model* to gain usage. This chapter deals with the common approaches found in the field of semantic models, especially with regards to computer science. Special focus is put on the technologies that were originally developed in the attempt of Tim Berners Lee and others to establish a *semantic web* under the guidance of the World Wide Web Consortium (W3C) [3].

In order to define terms which are relevant for the discussion in this thesis, the term semantics should be considered in more depth, which is covered in section 3.1. A discussion of the technologies used for the semantic web and a look at reasoners end this chapter.

3.1 SEMANTICS

Semantics are often defined complementary to syntax (e.g. [21]). While the syntax defines a formal correctness of expressions, the semantics of an expression give it meaning. In this sense, semantics go beyond syntax on a language-theoretical level. The syntax defines the mere formal structure of the language. With a correct syntax, expressions in a language can be created that carry no useful semantics. For a simple example once can take the sentence: The weather opens in traffic light. Syntactically, this is proper English. However, one is hard-pressed to devise proper semantics from the expression.

Semantics can also be defined a very formal, mathematical way (e.g. [28]). This is only useful when one wants to expand or redefine formal aspects for semantics. This formalisation of semantics is comparatively new. While syntax can be defined in a formal manner quite easily, semantics are an inherent part of human perception and therefore are more difficult to formalise. This is behind phenomena such as the fact that programming languages can be checked for syntactical correctness on the compiler level but can still be incorrect on a semantic level. To go back to the natural language example from above: The language structure with subject, verb, adverb and object can be easily checked to see that the above example sentence is syntactically correct. The reader will have a harder time to define why the semantics of the sentence are not correct in a formal manner.

Computer scientists have realised that formal definition of semantics is a prerequisite of processing semantics with computers. For that reason, the semantic web technologies were defined that are described in the rest of this chapter.

Before the technologies used in computer-calculated semantics are discussed, however, some terms with regards to computable semantics need to be defined.

ontology

in computer science means a set of formalised statements that assign semantics to *concepts*.

concept

is an entity in an ontological context. Especially concepts can be *classes*.

class

set of entities that follow a common set of attributes which define the class. Classes can be instantiated by *individuals*.

individual

entity in an ontology that symbolises an entity in the modeled context. An individual can be set into relation with other individuals, can instantiate classes and can be set in a relationship with a value.

The relationships can have cardinalities. This is especially prevalent with relationships existing above the indivdual level between classes. An ontology can be defined in a way that some attributes of classes or individuals are not explicitly formalised, but have to be true for logical consistency. If this is the case, then the not expressed knowledge can be inferred. The inferring of knowledge is a key factor in the computer scientific use of semantics described by the semantic web technologies.

3.2 SEMANTIC WEB TECHNOLOGIES

In order to make semantics storeable, computable and thereby automatable, the World Wide Web Consortium under the guidance of Tim Berners-Lee has defined a set of technologies that is tailored towards realising the semantic web [3]. The general idea is a layered model of technologies, that in concert allow to perform semantic web searches, e.g. The structure proposed by the W3C is reproduced in figure 3.1. As can be seen in the figure, several technologies build on top of each other for the realisation of the semantic web architecture.

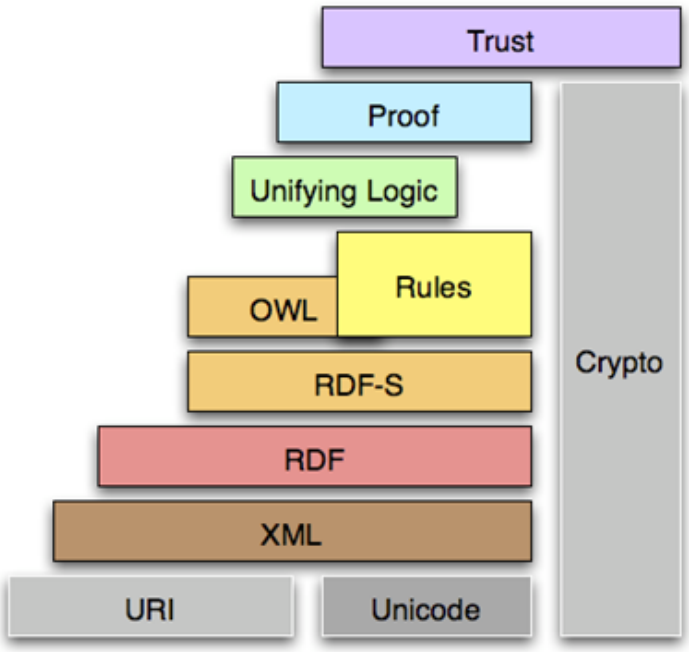


Figure 3.1: Layered Structure of Semantic Web Technologies [3]

The core technology that is best known for computer scientists is the XML standard. XML as an easily extendable data storage format can be read by humans as well as machines, and therefore is considered an important stepping stone of the semantic web. Based on the syntactical framework given by XML, the Resource Description Framework (RDF) has been developed. RDF defines Triplets that allow the expression of relationships between concepts. This concept is discussed in more detail in section 3.2.1. RDF is then, in turn, the basis of OWL, the Web Ontology Language. OWL defines a set of basic semantic relationships between concepts. This is discussed in more detail in section 3.2.2.

3.2.1 Resource Description Framework (RDF)

The Resource Description Framework or RDF is a set of specifications published and maintained by the World Wide Web Consortium (W3C) [18]. It was published in order to provide a language in which to define metamodels in a uniform way. As a web standard, RDF is based on URIs. Other technologies, OWL in particular, build on the so called recommendation the W3C put forward for RDF. RDF defines triplets that define a relationship between two entities. Hence, not unlike natural language, an RDF triplet has a subject - predicate - object structure.

In addition to the triplet structure, RDF defines a number of standard classes and properties. These serve as the primitives of the RDF language, from which user- or technology-specific classes and properties can be derived via specialisation. Hence the main class can be considered to be `rdfs:Class` and the main property for specialisation `rdfs:subClassOf`. Another important property for the use with other semantic technologies is `rdfs:isDefinedBy`. A comprehensive list of classes and properties can be found in the W3C specifications.

RDF itself thereby offers a syntax for expressing relationships between concepts. On this basis, the Web Ontology Language (OWL) allows the definition of complex concepts, as described in the following section.

3.2.2 Web Ontology Language (OWL)

The web ontology language (OWL) is a specification by the W3C that allows the formalised storage of semantic information. As of early 2013, there are the classical OWL [17] and OWL 2 [16] released by the W3C. Most tools and workflows are still based on the classical OWL, so this section focuses on the old specification.

Inside the specification for OWL 1, there are different complexity levels of OWL, called species by the W3C. These species have an increasing degree of complexity, both in the expressability

as in the computability. The simplest version of OWL is OWL Lite. This version of OWL simply provides modeling elements for taxonomies, and minimal cardinality support. More complex is OWL DL, which aims for maximal expressiveness while maintaining computer-verifiability. OWL Full at last grants the maximum freedom and expressability. The downside is that other than OWL DL, an OWL Full ontology cannot be reasoned over by a reasoner (see section 3.3).

The OWL species all base on RDF. On the syntax concept of RDF, a set of OWL-specific elements is defined. For example, the `owl:Class` element can be supplanted by the `rdf:subClassOf` element of RDF mentioned in the previous section. All classes that are defined in OWL base on the base class `owl:Thing`.

The most interesting technology for computability is OWL DL. It offers maximum expressability while maintaining full computability. This computability is employed by special software that can perform so-called reasonings on the ontologies. These reasoners are the topic of the following section.

3.3 REASONERS

As discussed in the previous section, reasoners are specialised pieces of software that are made for performing calculations on formalised ontologies. A reasoner as discussed in this thesis receives an OWL/RDF ontology as input and then processes the input for two results. One result is the identification of inconsistencies in the ontology. The second result is inferred knowledge. Inconsistencies are elements in the formalised ontology that on a logical level contradict each other. This could be because cardinalities are unfulfillable or because an individual does not follow the restrictions defined. Inferred knowledge is the expansion of the formalised ontology by elements that logically follow from the definitions made in the ontology.

For the Web Ontology Language (OWL), there are several reasoners that have been created to check the consistency and infer knowledge on the ontologies formalised in that language. Listing all reasoners available is beyond the scope of this thesis. Specialised papers exist that deal with the performance and reliability of reasoners [5].

4 INFORMATION MODELING IN AUTOMATION

In chapter 2, the state of the art in automation technology with a special focus on automation devices and the corresponding models has been given. In chapter 3, the purpose of semantic models and the semantic web have been discussed. At the intersection of both domains lies the problem of information modeling in automation. Information models for automation entities, just like in any other field, need to convey semantics of automation data. Therefore, it is the application of semantic modeling to the automation domain.

As can be guessed, the problems presented in chapter 1 have led to efforts put forth in this field. Despite these approaches, though, none of the efforts was successful in eliminating the problems of device manufacturers up to this point. This is partly due to existing approaches covering other use cases for the automation domain or related fields. This chapter discusses the most relevant approaches taken or currently being under investigation. An overview is also given by Wollschlaeger [81].

4.1 INTEGRATION AS A THREE-DIMENSIONAL ISSUE

Considering the classical hierarchy of automation as discussed in section 2.1, a general structure for approaches in the automation domain can be derived. In addition, the life cycle of automation entities as discussed in section 2.2 provides an additional dimension for information management considerations. The classical automation pyramid with the vertical and horizontal dimensions thus gets spread into a third dimension, the life cycle. This concept has been described and illustrated [73], as can be seen in figure 4.1.

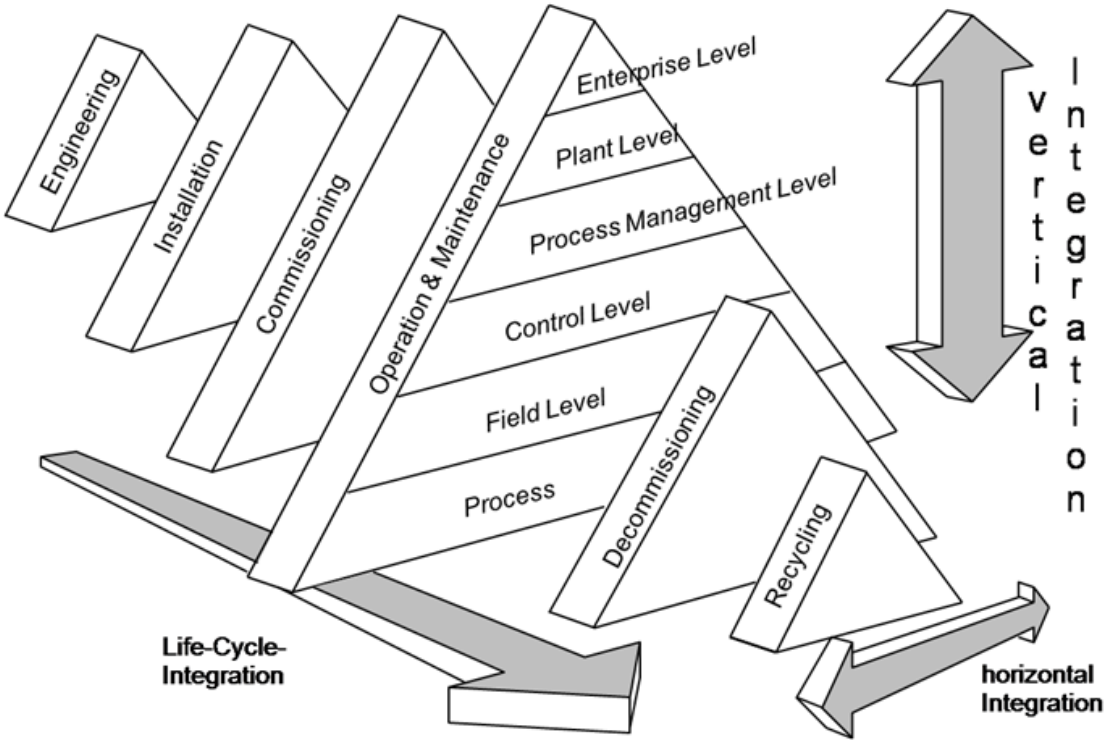


Figure 4.1: Three-dimensional perspective of the Classical Automation Pyramid considering the Life Cycle [73]

In order to illustrate what integration issues are covered by an integration approach, a three-dimensioned matrix has been developed for the research of this thesis. It can be seen in figure 4.2. An integration approach can be placed as a plane covering the areas that are being addressed in the approach. The matrix illustrates different qualities for each dimension by which they can be distinguished. The vertical dimension differs by the information integration as seen in the automation pyramid [9]. The different system layers represent information aggregation as well as a control hierarchy. The life cycle follows the phases of the automation life cycle [4]. The horizontal integration is divided by the compatibility levels used for classifying system interoperability [10].

Categorisation of integration approaches in these three dimensions can help distinguish them. However, similar areas covered do not necessarily mean that the approaches have a common foundation. The reason for this is the complexity of the automation domain. Whether the focus is a single device, a plant, a logistics network or a communication network cannot be differed from these dimensions. A closer look at the approaches is therefore advised. Nevertheless, the following sections use the dimensions as a means of structuring the state of the art of information modeling in automation.

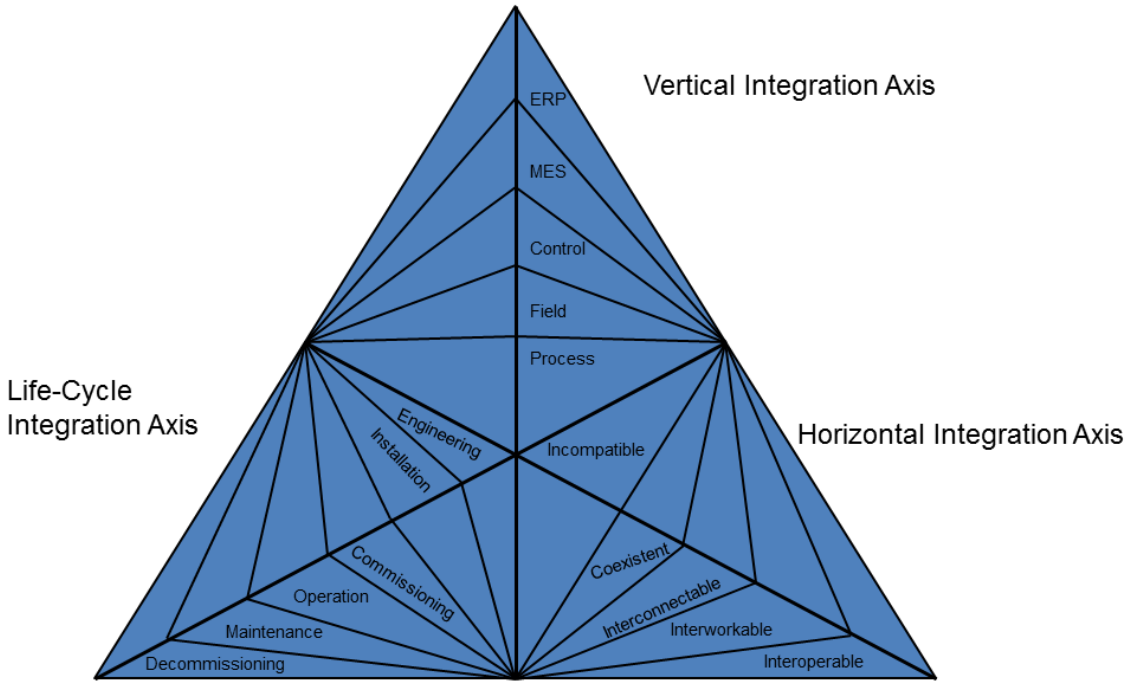


Figure 4.2: Placing Integration Approaches on a Three-dimensional Matrix

4.2 INTEGRATION IN THE HORIZONTAL DIMENSION

Integration in the horizontal dimension of the automation pyramid is a long-standing issue and several approaches have been formulated. Of course, these efforts differ by the level on which they are undertaken. When in the times of simple 4-20 mA signals a connection diagram was considered state of the art, the advent of fieldbus technologies has made horizontal integration an issue for considerations of technology interoperability. On the ERP level, in contrast, many issues were solved with the progress in the integration of office software solutions. These technologies, which are not specific for the automation domain, are not considered in this overview.

An dated overview of the horizontal integration issue for fieldbus technologies can be found in a work by Sauter and Wollschlaeger [67]. The approach taken by the IEC for making all fieldbus technologies work in the same foundations is IEC 61158 [8]. The attempt to unify device profiles can be found in IEC 62390 [10]. Programming of the device controllers is standardised in IEC 61131 [7]. The overall integration is meant to be covered by ISO 15745 [30]. Despite this number of standards dealing with the integration issues, the need for more efforts has been identified and addressed.

The group of Kalogeras has proposed the application of agent-based integration, where each entity is represented by an autonomous agent that then fulfills control tasks on a local level [34].

This approach is interesting, especially considering the applicability of agent-based ontology integration approaches [22]. However, it does not address the fieldbus integration via device descriptions as such. The approach is in principle comparable with the approach proposed by Lastra [55] to create ontologies for automation objects. The approach to expand this to product centric views also exists [74], and will be reviewed further in the section on integration over the life-cycle. Explicit formulation of domain ontologies for the purpose of making informed control decisions can be found in a publication by Uddin [75].

The relevance of the model defined in IEC 62264 [9] for horizontal integration on the MES level (in contrast to the widely seen usability for vertical integration) is also recognised by the OPC Foundation, who has formed a working group that is tasked with defining a specification on how objects and attributes defined in IEC 62264 (also known as ISA 95) can be accessed via the OPC UA technology described in section 2.4.4 [32]. Unfortunately, no results have been published at the time this overview is collected.

The idea of formal verification for a proposed configuration via Petri net models has lead to works by Lobov [56] and the thesis of Däubler [21]. Formal modeling of relevant attributes in the process industry is discussed in the thesis by Heeg [43].

Selig does, in his thesis [69], discuss the formal comparison of device description languages. He does, however, claim that the device models that are the foundation of device description languages cannot be utilised because they are supposedly lost in the creation process of the DDL. The work is based on graph theory, providing visual abstractions of DDLs. A class model for device models is derived, that is then used for functional classification based on graph theory. While the working field, mapping of device description languages, is similar to this thesis, the approach is different. The goal of Selig is to identify functional relations between different devices. This thesis, in contrast, tries to map DDs for one device. Selig only classifies DDLs. This thesis defines a computable mapping methodology. Selig only offers XSLT for the mapping of device description languages, something that he himself admits is not covering the broader basis of relevant DDLs in the market today. His thesis can nevertheless be recommended as a supplementary read for interested professionals who need to compare devices on a functional level.

Overall, it can be surmised that despite the technological basis for horizontal integration being provided by standardisation bodies mostly about a decade ago, there still is demand for a better integration between the different technologies. The issue of redundant information in different device description languages is not yet addressed in a holistic way.

4.3 INTEGRATION IN THE VERTICAL DIMENSION

The issue of integration between different levels of the automation pyramid has been a hot topic for researchers in the last decade. The different development speeds on the different levels along with an overall increased speed of innovation due to the advances of the semiconductor industry have led to a host of different approaches how heterogeneous systems with differing tasks can interoperate on different levels. The classical SCADA did not perform well enough for some applications and so we now have the opportunity to consider a set of paradigms to be used for vertical integration.

One such paradigm is the use of a middleware architecture for integration of heterogeneous information exchange partners. An overview of the criteria and state of the art for middleware in automation has been performed by Mahnke et. al. [57].

An integration between ERP and other industrial software systems was the goal of the Aletheia project [12]. However, the focus of this project is not necessarily the integration of the automation pyramid [68], and therefore does not address some issues that are relevant in this thesis.

The abstraction of fieldbus specific technological issues is the matter of standards such as IEC 61131 [7] and IEC 61499 [11]. Another approach for making applications independent of the low level technological demands of the network layer has been published by Theurich et. al. [73]. Wenbin Dai et. al. propose the usage of ontologies for generation of IEC 61499 code [20].

Performing vertical integration through the use of adapters is the goal of the PLANTCockpit project [14]. The project is aimed at improving the software development process for vertical integration by eliciting examples [23]. For PLANTCockpit, semantic interfaces with adapters for specific target technologies are addressed [24].

In the domain of building automation, ontologies have been utilised to provide information that is necessary when the integration between the different automation systems needs to be facilitated. The works of Dibowski [25], [66] are a state of the art approach for this domain.

The issue of vertical integration in the automation domain is approached by different angles, and no single one has proven to solve all problems that exist. The ongoing efforts to fuel the *next industrial revolution* that is promoted in the popular press at the time of this writing may help come to a common grasp on the matter. Until that happens, full integration in the vertical domain is still a matter of effort that can and should be optimised by all parties involved.

4.4 INTEGRATION OVER THE LIFE-CYCLE

Integration over the life cycle of an automation system has seen rising interest in the last decade. The consideration of total cost of ownership (TCO) of a solution has created a mindset that is not merely concerned with the price of a solution's acquisition, but also with the integration and maintenance efforts necessary. In this spirit, several initiatives were launched that analyse the integration over the life cycle of automation plants and devices as well as propose ways of handling the integration tasks.

A first step can be to erase efforts for upgrading older plants by providing a sophisticated life cycle management. Fay et. al. have discussed a possible solution for this scenario [72]. The usability of diagnostics with reasoner support is discussed by Müller et. al. [62].

The oil and gas industry stepped forward several years ago to propose a data model as a description of their application domain. This has been standardized in ISO 15926 [31] and is a domain description from the application owners' point of view. Since the release of the first part of the standard in 1999, the example has not spread in the same way in other industrial domains. The latter parts of the standard, however, part 8 in particular, utilise the Web Ontology Language (OWL) as a means of providing semantics for the data models.

As has been discussed in section 2.2, the life cycle can be applied to plants, devices as well as products. A sophisticated approach centering on bringing together product models based on the concept of duration of time is presented in Matsokis' thesis [59]. This work has also been discussed by Kiritsis [52], who discusses the possibilities and implications of product embedded information devices.

The research project SemProM [15] had the goal of providing a semantic digital memory for products. The concrete aim for the industrial domain was to provide workflow support for individualised products. A product memory is meant to be kept over the whole life cycle of a product and supply any relevant information wherever it may be needed.

An ontology for product-based manufacturing is a basis of the Pabadis'Promise project [13]. Through the use of RFID tags the goal was to have self-organising, order-based production workflows. The similarity to the approaches for the next industrial revolution, that is currently announced for research projects in the automation domain, is striking.

The problem of a lack of reuse of engineering models is addressed holistically in the PhD thesis of Mathias Mühlhause [60]. This work focuses heavily on the usability of semantic technologies for the plant automation domain, but remains on an engineering level and is not mainly concerned with device description languages.

The industry-motivated specification of AutomationML tries to solve integration issues in manufacturing by defining a common data exchange format [27]. It is based on pre-existing XML formats, specifically CAEX, PLCopen XML and COLLADA. AutomationML offers a class and object hierarchy structure that allows the storage of a whole plant structure and specific information about the machines or devices in the plant. AutomationML is meant to be extended in the future to allow the storage of more information such as communication setup. The focus, however, is on storing information in a unified way. Neither does AutomationML carry formalised semantics, nor does it address the device integration level in particular.

An approach similar to AutomationML is PandIX that instead of automative, focuses on the process industry. The content of the models is the process technical interconnection between devices in the process industry [29].

Based on other technical standards, such as IEC 62264 and OPC, the MIMOSA group tries to promote Open Operations and Maintenance (Open O&M) via the Open Systems Architecture - Enterprise Application Integration (OSA - EAI) [41]. The approach tries to provide a semantic model of operation and maintenance information that is usable by the industry for a common reference framework. Despite collaboration with the ERP provider SAP it has to be noted that these efforts did not reach the European industry as of today.

5 ONTOLOGIES FOR DEVICE INFORMATION MODELLING

So far the existing heterogeneities in the automation domain have been discussed in chapter 2. The possibilities of semantic models have been highlighted in chapter 3. Examples how these two domains are brought together for added value have been discussed in chapter 4. In this chapter, ontology models for concepts from the automation domain are presented. These models have been developed by the author in order to be utilised in an approach presented in chapter 6, which the author proposes to be used for data format transformations in the automation domain. The approach proposed relies heavily on the utilisation of ontology models, so that this chapter does not only present an array of exemplary models, but the following text also defines a modeling heuristic that ensures the usefulness of the models in question.

Modeling an application domain can be done in two ways. Either the whole domain is subjected to a process in which the goal is one general model of the domain, which then should be able to express every concept relevant for that domain, or there are several models that pick out aspects of the domain and only describe these. In general, the latter approach has to be used for the ontology models that we use for transforming data formats. The reason is that in our use cases, engineers need to be able to understand, expand and generate the models easily. A big model is more expressive, but harder to understand. The concept of integration ontologies, as defined by the author [38], e.g., helps overcome the problem of limited applicability of the more specialised ontology models.

As has been illustrated in chapter 3, OWL allows for a high degree of freedom when modeling concepts as an ontology. In order to facilitate later usage of the models developed, a more rigid approach for the modeling of concepts has to be defined. This is especially important because the usage of the models presented here in chapter 6 is based on computer based processing of

the models. An important aspect of utilising ontologies in computed information processing is a unified storage of information payload. Since this thesis discusses the modeling of data formats, a common approach for storing data elements needs to be pursued.

The approach proposed for storing data elements is based on the definition of data properties for OWL ontologies. Every data token defined in a data format should be represented as a data property. This data property should have an appropriate data type. In addition, a naming convention for the data properties is that the name should be a concatenation of two elements. The first element for data properties is always the english word `has`, the second element is to be the name or denominator of the data token in the original data format. For example, a data token named `vendorname` with a string value should be modeled as the data property `hasvendorname` with a range of string values.

Modeling the data tokens is an important stepping stone, but does not allow the classification of individuals by a reasoner. For that, a class hierarchy needs to be defined as well. It is important that each data token, besides the appropriate data property, should be supplied with a class named after it. Therefore, in the above example, there would be a class `Vendorname` in the resulting ontology. Apart from this, little can be defined in general, because each data format is modeled after a more or less different paradigm from all others. However, some guidelines can be given.

If the metamodel of a data format does contain inheritance or specialisation, this should also influence the class hierarchy of the resulting ontology. A concept that is a generalisation of other concepts should be modeled as the superclass of these concepts in the ontology model. A lot of data formats support aggregation. Aggregation should be modeled with object properties. The recommendation is to have the object properties `contains` and `isContainedIn` in the ontology. These object properties should be each others' inverse properties. With these object properties, aggregations can be modeled. With anonymous superclasses, assertions for aggregations can be defined.

The overall approach to build ontologies according to the above rules is put into a work flow overview in figure 5.1.

Additions to a model defined by these guidelines should follow the same guidelines, in order to ensure that software components reliant on the resulting structure keep being consistent with the ontology model. As long as the structure is not perturbed, however, the ontology can be expanded by any further modeling elements that may be desired or useful for other use cases.

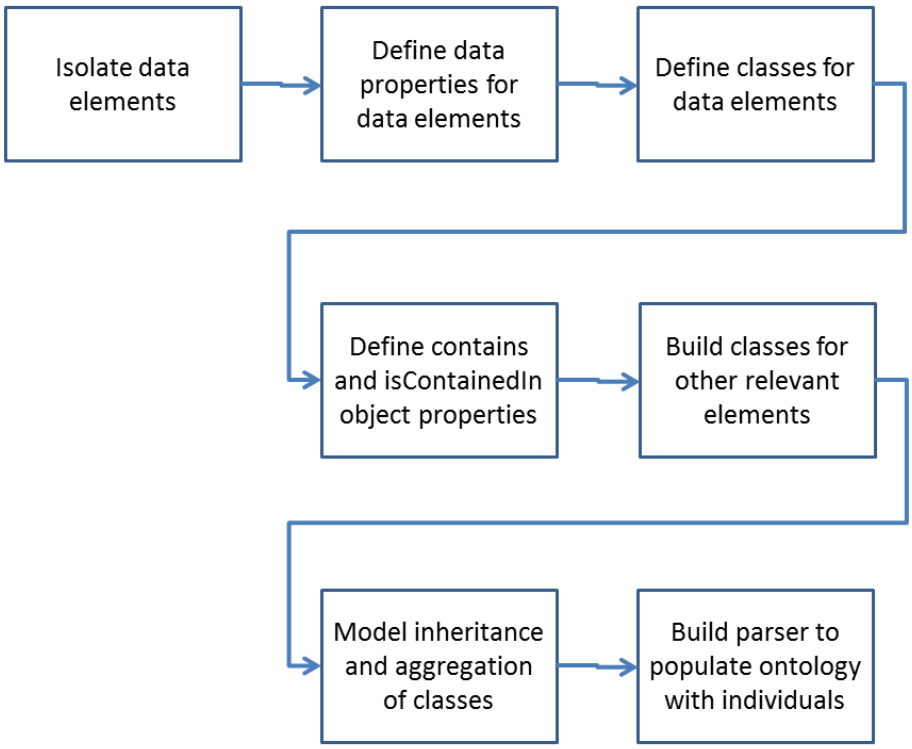


Figure 5.1: Workflow for ontology generation

5.1 ONTOLOGIES FOR DEVICE INFORMATION

As has been discussed in chapter 2, the automation domain offers a plethora of data formats. Even the specific task of storing and communicating device information for field devices is handled by a set of different standards that carry partly redundant information. Before an approach for bringing this information together can be defined, an important tool needs to be provided. This tool is described here in the form of ontology models for device descriptions.

When defining the ontology models, it is useful to discuss the relationships between some concepts in the modeled domain. A device description is an abstract model, which allows the retrieval of information about a device. If an encoding is given that allows the storage of the device description on some kind of information media, the device description is supplemented by a device description language. The language has the role of a data format for storing instances of the device description. In general, an instance of a device description asserts fixed qualities to a specific type of device or group of types of devices. The instance of the device description, given in a device description language, represents the device information of that device (or typically rather that type of devices). A device model is a meta model for a device description. The device model defines which qualities a device can have. The device description then defines a way how these qualities can be modeled. In that way, a device model is the foundation of one or several device descriptions.

In order to make device description languages mappable unto each other, a set of ontology models for device description languages has been defined. The models are created in a way that makes the device information a set of individuals of the classes defined in the ontology model. So a concrete device's device information is modeled by a set of individuals in the ontology model of an appropriate device description language.

The goal will be to make the device description languages compatible to each other, so the modeling approach described at the beginning of this chapter has been applied to all models discussed in this section. ISO 15745 [30] has been chosen as a first example of an ontology model for device descriptions, because it is the basis which all other technologies discussed in this section refer to. Examples for device description languages that are modeled are EDDL, GSD, GSDML and CANopen EDS (see chapter 2 for details on these DDLs). These technologies cover devices connected via the PROFIBUS and the CAN technologies, technologies that are typical for factory automation setups in Germany, as the author can attest from his shop floor observations and work experience.

This section concludes with an integration ontology that defines mapping between the different technologies. The purpose and application of this integration ontology is discussed in chapter 6. In figure 5.2 an overview of the domains modeled in this section is given.

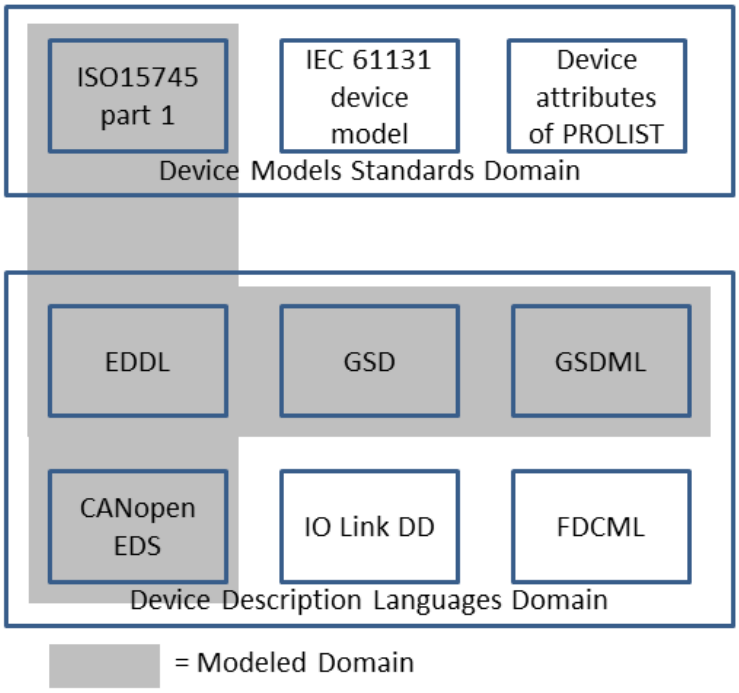


Figure 5.2: Overview of the Focus of the Modeling in the DDL ontologies section

5.1.1 An Ontology for ISO 15745

The ISO standard 15745 provides a powerful approach for integrating enterprise information systems. However, due to its broad scope, not all elements that are modeled in the standard are relevant for every possible task related to it. The generation of an ontology model for ISO15745, at least for the generic part 1 of the standard, can be a valuable stepping stone, however, in that it provides a basic conceptualisation and defines common starting points for use when generating an integration ontology for device descriptions.

The possible usage of ISO 15745 part 1 has lead to an ontology model of the standard being defined. Overall, a whole set of profiles for different resources is given. For the purpose aimed at in this chapter, the device profile and other device related considerations are a priority. However, the ontology model of ISO 15745 Part 1 takes all elements for an application integration into account as they have been discussed in section 2.3. The modeling can easily follow the class structures provided inside the standard, the relationships between the entities are of more complexity, however, and need an understanding of the implied relationships between the classes of the standard. Figure 5.3 shows an excerpt from the defined model highlighting relevant classes. For comparison, figure 5.4 shows again an excerpt from the class structure of ISO 15745 that is modeled in the ontology.

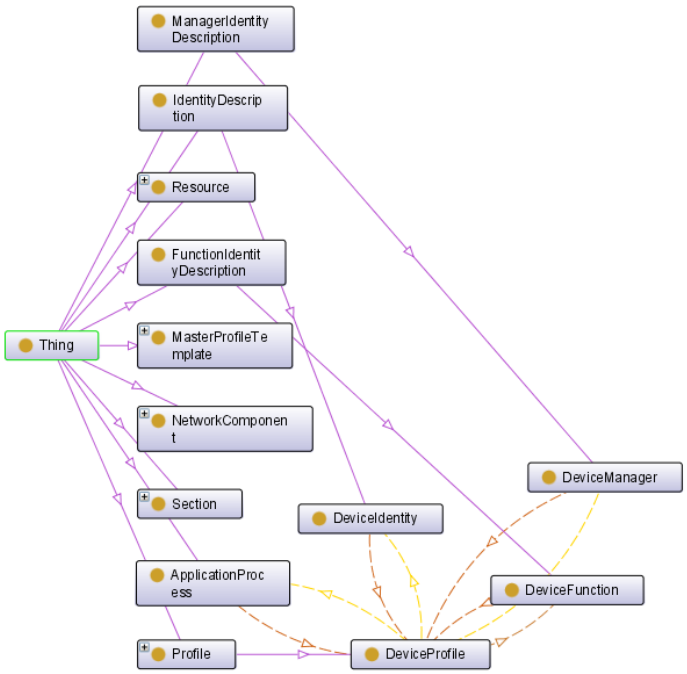


Figure 5.3: Excerpt from the ontology model of ISO 15745 part 1

The containment relationship expressed in the class model is modeled as `contains` relationships between the `DeviceProfile` and the `ApplicationProcess`, `DeviceFunction`, `DeviceManager` and `DeviceIdentity`. The device profile will be the basis for considerations in

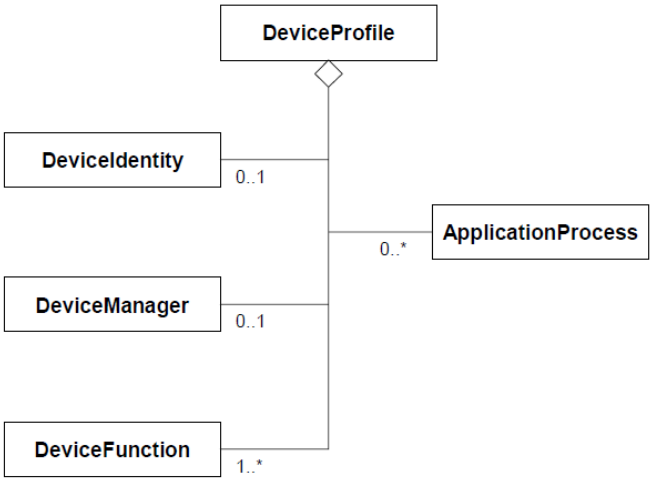


Figure 5.4: Reproduction of the Composition of the Generic Device Profile according to ISO 15745 [30]

section 5.2.1, and the device identity is one thing that is being modeled in all considered DDLs in this section. In chapter 2 an example for the usage of device identity for CANopen was shown as an example. Some things shown, especially the concept of `Section` are not part of the considerations in this work, and are added for completeness of the standard.

5.1.2 An Ontology for EDD

An ontology model for the Electronic Device Description has to differ from the other device descriptions discussed in this thesis. One issue with generating an ontology for EDD is that due to its closeness to the programming language C, there are not only key words and data elements to be considered. The highly variable structure of the language makes it necessary to also model the denominators used for variables, arrays, blocks etc. in the language. Because the keywords of the language are written in capital letters, elements that are added can be distinguished by writing them with a capital first letter followed by all lower case letters, as is typical of other ontologies described in this thesis.

Another speciality of EDDL is that it is tailored to model and define dynamic behaviour. Therefore, some structs and keywords in EDDL are not applicable to static device descriptions. Therefore, only a subset of the keywords in EDDL is to be modeled in the application context given here. (Other aspects may still be modeled, but yield no usage for the use cases discussed in this thesis.) The result is an ontology that focuses on comparatively few classes and data properties when compared to other DD ontologies.

The flexibility of the EDD makes the mapping to an integration ontology challenging. As a result, the integration ontology that the EDD ontology is integrated into will have to take into account

naming conventions for variables and data structs in the EDD. This is most easily done when a GSD ontology or an ontology of the respective bus technology is already integrated, because the definitions of this technology specific ontology can be used for identifying elements in a corresponding EDD. Figure 5.5 shows the language elements relevant for other device descriptions in an EDD, modeled as an ontology. The brown arrows denote containment relationships.

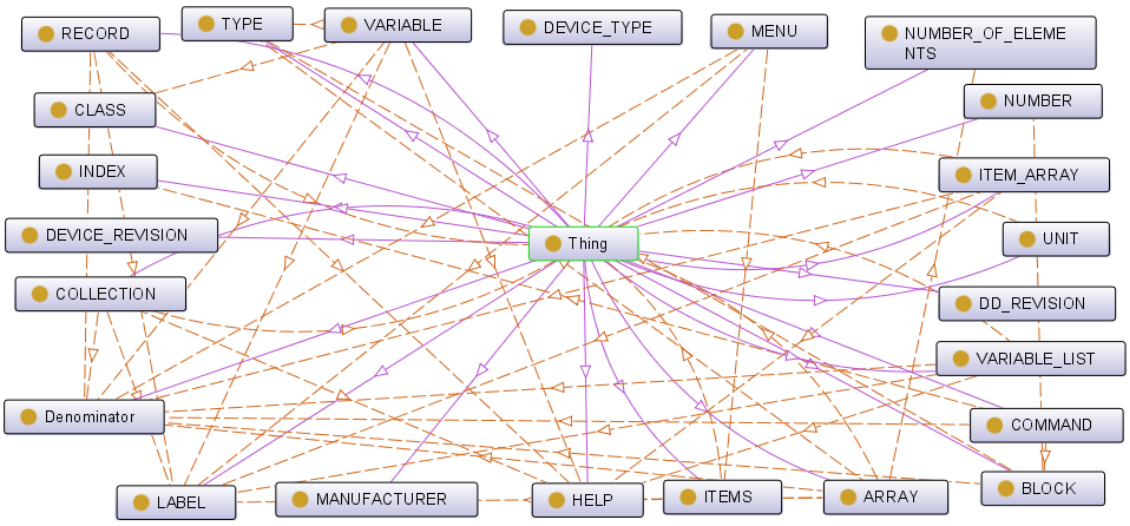


Figure 5.5: Class Structure of a DD-mapping-specific EDD Ontology

5.1.3 An Ontology for GSD

An ontology model for the DDL GSD is presented here. It is based on the GSD technology that has been discussed in section 2.4.2. The model follows the paradigms given above in the generation of data properties and class hierarchy. The keywords from the GSD specification are used as data tokens. Each of the keywords therefore has a data property named after itself. Each keyword also has a class named after itself which is a subclass of the anonymous superclass that has the according data property set with a value of the correct data type.

The class hierarchy of a GSD is comparatively simple. Therefore, starting at the general base class `Thing`, there are only three levels of inheritance. An excerpt from the class hierarchy in the GSD ontology model is given in figure 5.6. A `GSD` contains `GSD Items`, and a `GSD Item` is contained in a `GSD`. The relationship is modeled by the object properties *contains* and *isContainedIn* respectively as indicated by the relations in the class diagram. A `GSD Item` can be a module, a module definition or a unit definition item. Unit definition item is the superclass for all data token classes that are truncated and only shown partly, due to the huge number of classes necessary. The joker block item and the slot item are used in joker blocks and slots, respectively, and are not related to other classes in an inheritance sense.

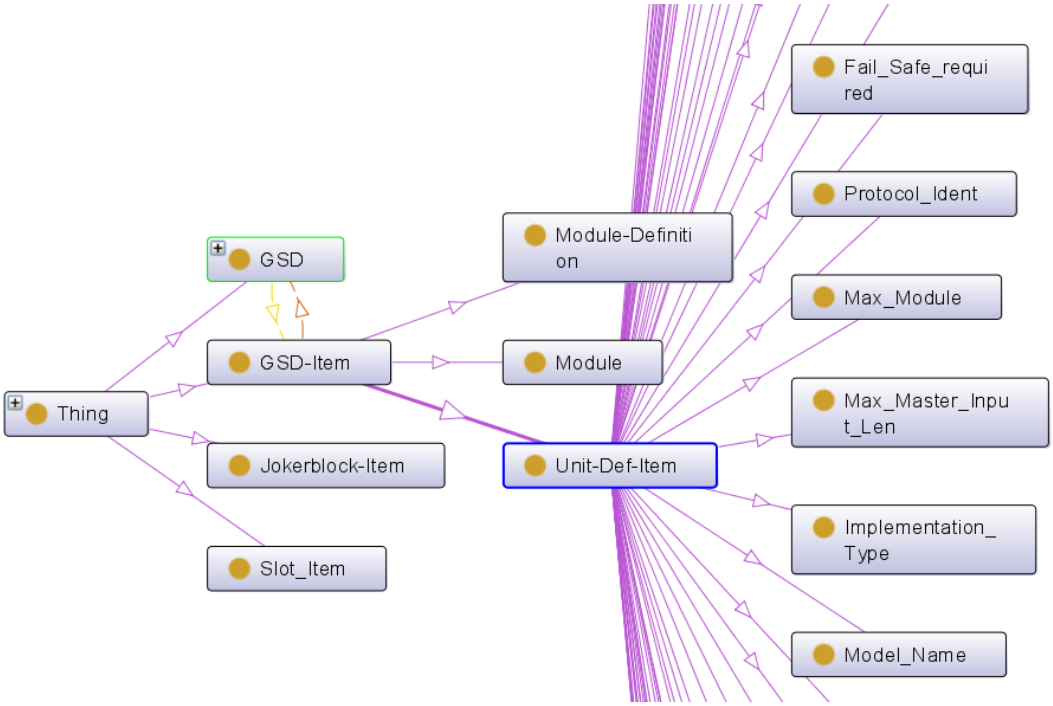


Figure 5.6: Excerpt from the class structure of the developed GSD ontology

5.1.4 An Ontology for GSDML

An ontology model for the GSDML specification (which has been discussed in section 2.4.2) can be built on the fact that the specification itself defines the structure of a GSDML using UML class diagrams. This makes the structure of the resulting ontology quite clear, using `contains` and `isContainedIn` as well as inheritance to model aggregation and inheritance in the UML diagrams, respectively. The attributes defined in the UML diagram can be modeled as classes with corresponding data properties as described above.

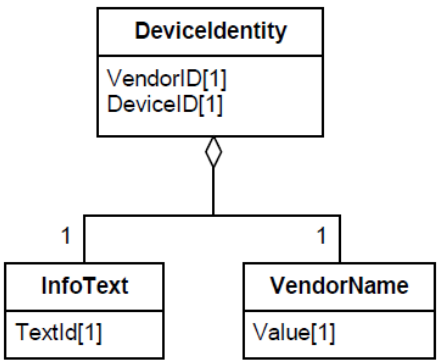


Figure 5.7: Excerpt from the class structure for Device Identity from the GSDML specification [48]

Of note is that attributes in the class structure can be modeled as concepts in the ontology. This is necessary to allow the mapping of attributes/concepts onto each other. For comparison

reasons, the class structure for the device identity from chapter 2 is reproduced here in figure 5.7. When compared to the ontology structure in figure 5.8, it can be seen that both the classes as well as the attribute have been mapped to the ontology as concepts to allow for easy mapping of the concepts to other ontologies.

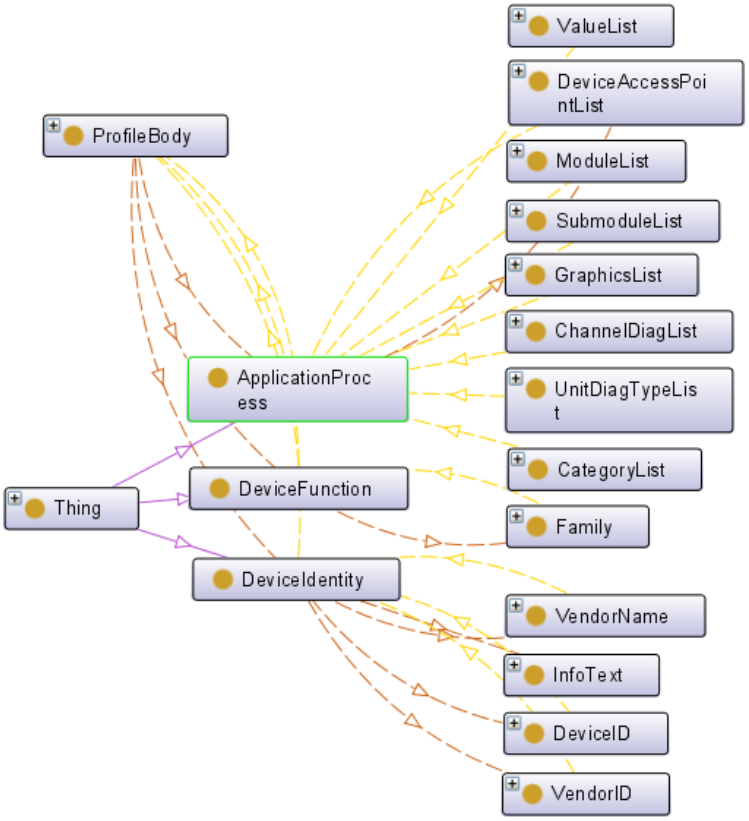


Figure 5.8: Excerpt from the class structure of the developed GSDML ontology

The base structure of the profile body of a GSDML as modeled in the ontology is illustrated in figure 5.8. The brown arrows indicate a `contains` relationship. The yellow arrows indicate a `isContainedIn` relationship. Where there is only one of those relationships defined, one element is not defined by necessarily containing the reverse element on a class level, because the other element is optional to the composition of the data structure.

One speciality noteworthy for the GSDML ontology is based on the reuse of data structures throughout the specification. The structures like `ID`, `TextId`, `InfoText` and even `ModuleInfo` are resued in several places. This leads to the parser for a GSDML having to handle a nested structure for data elements. Concepts from other ontologies will be mapped on higher level elements. It is the task of the parser to build or read the structure of the GSDML accordingly. Since the structure is well-defined, though, this task is fulfillable.

5.1.5 An Ontology for CANopen

An ontology model for the Electronic Data Sheet (EDS) for the CANopen fieldbus technology can be modeled close to the heuristic defined above. CANopen EDS are text files with key words. Therefore, each keyword can be modeled as both a class concept and as a data property. Because EDS parsers have to interpret all entries as strings, the data properties have to use the string data type as well, further easing the modeling process.

Some aspects of the EDS format that serve only for aggregation in the file and do not hold information value about the device have to be handled by the parser. Therefore, arrays are not modeled in the CANopen EDS ontology model. Likewise, the repetitive keywords indicating different baud rates for the communication to and from the device have been shortened to one variable. A parser has to take this simplification into account.

The section names of the EDS file are modeled as generalised concepts, because the information section aspect makes this structuring of the keywords feasible for ontology modeling. Where keywords have different meaning regarding their sections, the section name has been taken into account to ensure a differentiation of class names, which is a mandatory condition for OWL ontologies.

The resulting class structure for the CANopen EDS ontology model can be seen in figure 5.9. Due to the keyword-based approach of the EDS, only one containment relationship is present in the model.

The CANopen EDS is a good example of the straight-forward usage of the modeling approach defined in this thesis. The comparatively simple structure of an EDS allows a swift and robust modeling.

It should be noted that CANopen XML has not been modeled in this thesis. The reason for that is to be found in applicability considerations when deciding between the formats, because the older EDS format is still in heavy use in the industry and has a structure more different from GSDML, so that the approach promised more effect for it. There is no reason not to add an ontology model for CANopen XML in future work, however.

5.1.6 An Integration Ontology for Device Description Languages

The integration ontology that results from merging the aforementioned ontologies is at first glance a huge collection of concepts that in parts double in names. However, due to the

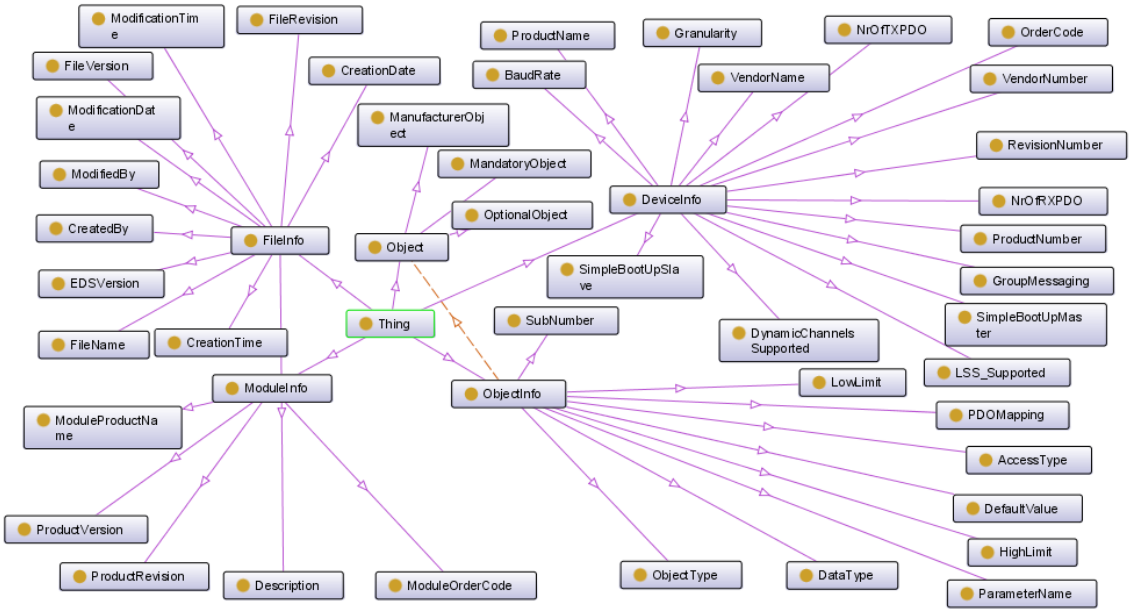


Figure 5.9: Class structure of the CANopen EDS ontology

different source ontologies' URIs, the concepts with same names can still be distinguished. For the mapping it is worth to check the similarly named concepts for possibly describing the same concept. With the OWL property *sameAs*, the reasoner can be informed about concepts that need to be mapped unto each other. This way, a comprehensive mapping of concepts that need to be transformed between the different device description languages can be achieved.

The integration ontology could also use sophisticated structures to calculate values that can not be retrieved from other formats directly, e.g. through lookup-tables that automatically transform vendor IDs for different bus organisation ID schemes. This approach is beyond the scope of this thesis, however, which introduces the mapping approach per se, and is recommended for future work.

The utilisation of the integration ontology will be elaborated in section 6.1 of this thesis.

5.2 ONTOLOGIES FOR DEVICE FUNCTIONALITY

The previous section has explored ontology models for the transformation of device description languages. As has been hinted at in chapter 1, other use cases can be considered once modeling of device information is in the focus. A use case that has increasing relevance is condition monitoring, a task relevant for the maintenance of production facilities [36]. Condition

monitoring encompasses the evaluation of machine status with the purpose of identifying maintenance demand.

Condition Monitoring can increase the performance of maintenance strategies if the condition of a machine can be evaluated through already existing communication channels. This raises the demand for communication profiles that are tailored towards the condition monitoring task. Profiles as far as automation devices are concerned should follow the standard IEC 62390 [10]. This standard defines guidelines for the creation and accordingly the usage of device profiles for automation devices (see chapter 2).

The application of device profiles permeates the automation and fieldbus domain. For profiles, too, heterogeneity exists because of the different fieldbus technologies in use. This is where the situation with device profiles is similar to the one found with device description languages. For the same application and device, different profiles may be necessary because of different technologies in use in different automation environments. Therefore, similar to the work done for device description languages in the previous section, this section discusses ontology models for device profiles for automation devices.

Due to the demand for condition monitoring, this approach for maintenance demand identification has been chosen as an example for device profiles to be modeled. As a base, however, an ontology for IEC 62390 was created first. The condition monitoring ontology discussed thereafter bases on work done for a unified condition monitoring approach by a working group of the German industry with participation of the author [76]. As a supplement, the PROFIBUS PA device profile [47] has been modeled as well.

In figure 5.10 an overview of the domains modeled in this section is given.

5.2.1 An Ontology for Device Profiles

When trying to build an ontology model for device profiles, it is prudent to base it on IEC 62390. As has been shown in section 2.3, the technical report defines an approach for defining device profiles. While little elements are fixed in the report, since it leaves a lot of degrees of freedom, the recommendations given can provide a basic framework for conceptionalisation of device profiles.

The basic makeup of a device profile is the decomposition into header, parameter list and device functional structure. While the header stores identification information that is mappable onto the identification found in other DDs, the parameter list and device functional structure contain a parameter list and a functional element list, respectively.

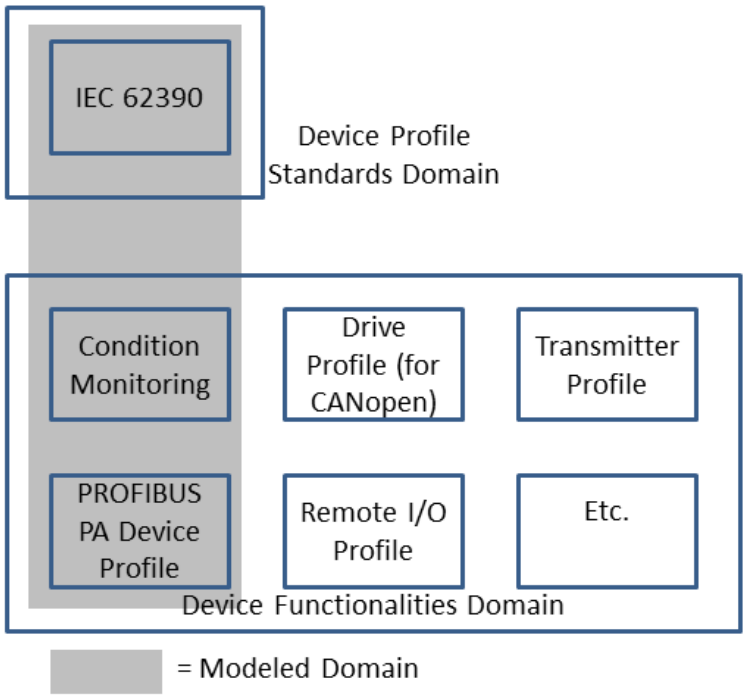


Figure 5.10: Overview of the Focus of the Modeling in the Device Functionality ontologies section

A basic structure for an ontology for device profiles that follows this layout is illustrated in figure 5.11. This ontology is usable as a starting point for integration ontologies that address device profiles.

5.2.2 An Ontology for an Application in Condition Monitoring Profiles

As has been mentioned in section 2.2, there are different maintenance strategies for manufacturing equipment. The strategy of Condition Monitoring has been the focus of an approach by a joint academic and industrial working group [36]. A result of this work has been a guideline by the German Association of Mechanical Engineering Companies [76].

In order to facilitate the possible later mapping of the resulting device profile for Condition Monitoring with other device profiles, an ontology has been created by the author that bases on the device profile ontology described in the above section. In order to facilitate ontology integration, the profile ontology has been directly expanded by the Condition Monitoring parameters.

The parameters added base on a function block concept that is defined in the aforementioned guideline [76]. The function blocks have inputs, outputs, parameters, amongst others, and also identification parameters, that allow the detection of function blocks and their capabilities. Some

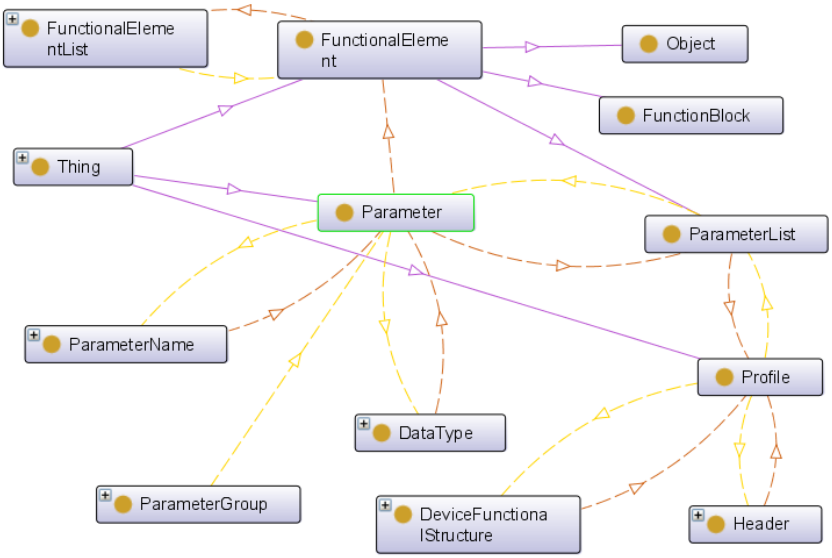


Figure 5.11: Class Structure for a Device Profile Ontology based on IEC 62390

of the parameters are grouped, because they logically have a connection with each other. The details of the guideline cannot be reproduced in this thesis because of copyrights owned by VDMA, the German Association of Mechanical Engineering Companies. However, the details of the communication-relevant paramters can be seen in figure 5.12.

It should be noted that the parameters from the guideline have been added as subclasses of the parameter class found in the device profile ontology. The parameter wrote capitalised has a different meaning, because it refers to calculation parameters of the function block, not device parameters. The aggregate parameters have been added as parameter group subclasses. Further relationships between the parameters follow from the properties of parameters in device profiles in general. As has been mentioned, the concrete application scenario of the guideline has not yet been defined at the time of the writing of this thesis. In order to facilitate the discussion, an ontology for PROFIBUS PA device profiles is presented in the next subsection.

5.2.3 An Ontology for the PROFIBUS PA Device Profile

The PROFIBUS PA device profile [47] has been introduced in chapter 2. In order to show how complementary device profiles may be integrated, an ontology for the base elements of this device profile has been generated and is presented here. The ontology considers the basic elements of a device profile, blocks and parameters, and sets them into an ontologial relationship. An excerpt of the ontology structure devised can be seen in figure 5.13.

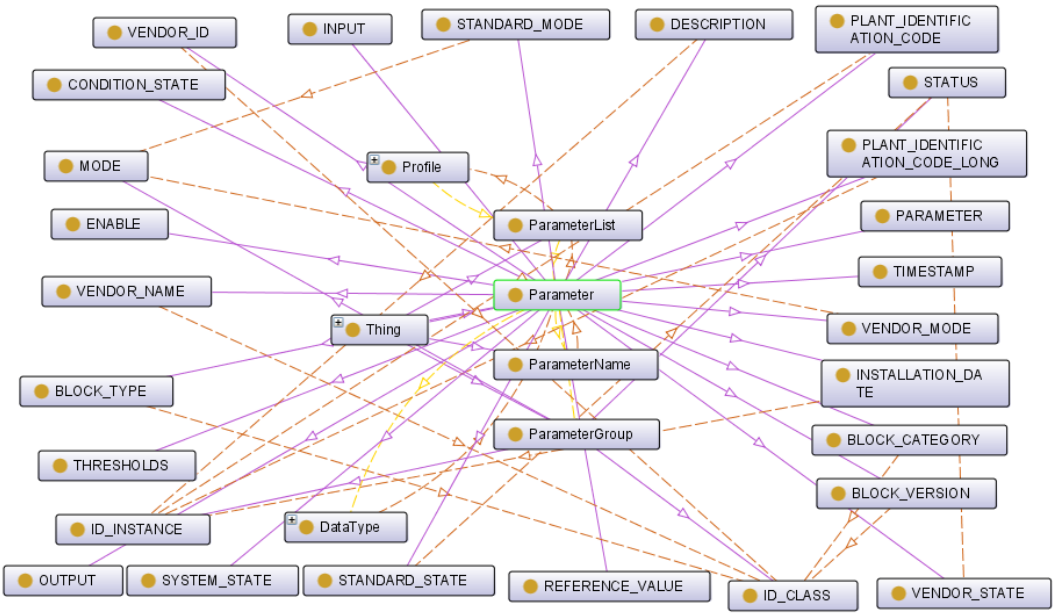


Figure 5.12: Class Structure for an Ontology describing a Device Profile for Condition Monitoring

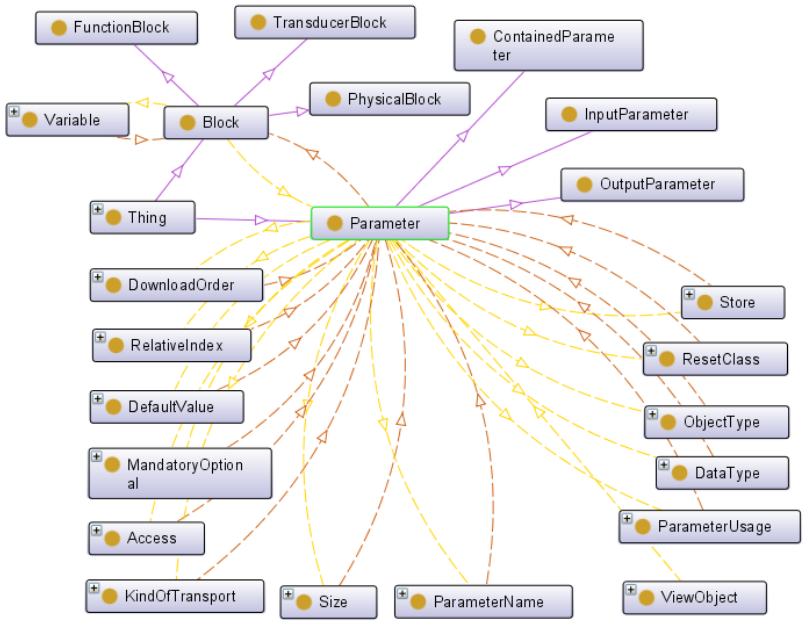


Figure 5.13: Overview of the Ontology for PROFIBUS PA Device Profiles

It can be seen from the figure that a parameter contains a big set of qualities. In comparison, blocks have been modeled rather simple, falling into different classes according to the profile. Parameters are the main quality to map, which is why they have been modelled with more depth. The Condition Monitoring blocks from the previous subsection fall into the category of function blocks according to this profile. The following subsection will show how a mapping between the two profiles might be done.

5.2.4 An Integration Ontology for Device Profiles

When merging the ontologies for Condition Monitoring and for PROFIBUS PA devices, the easiest thing to do next is to look for equally named concepts. Both ontologies feature parameters, and both those parameters contain a data type. When an equivalence between the parameters and the data types is declared, the visualisation gives them a mutual inheritance relationship, as can be seen in figure 5.14.

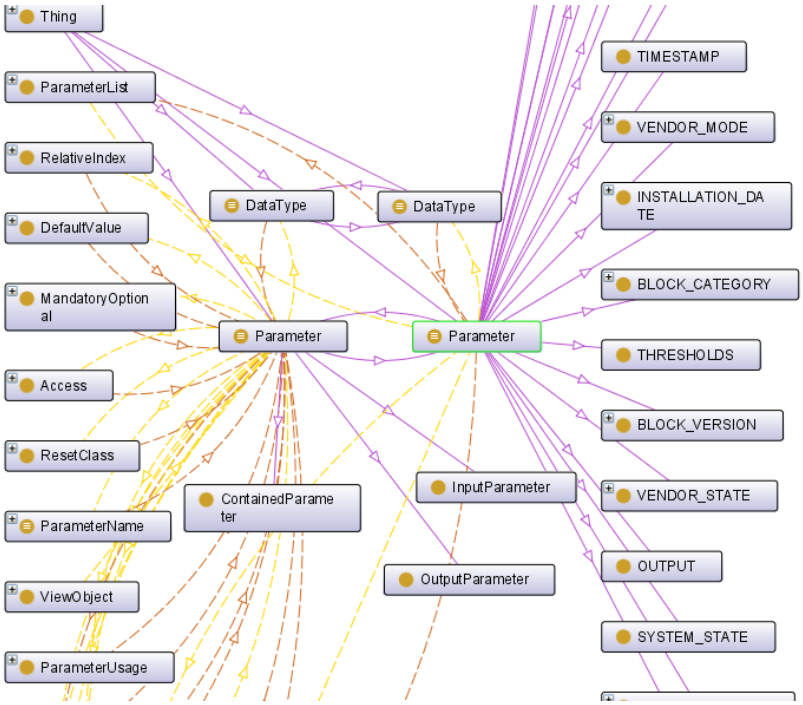


Figure 5.14: Excerpt from the Device Profile Integration Ontology

This merged ontology can then, with the equivalencies expressed, be used to infer a relationship between the data content of either parameter concept. In addition, the internal and external children of the parameters in the PROFIBUS PA profile can be used to further specialise the children of the Condition Monitoring profile. In this way, the quality of each parameter for being either an input or an output parameter can be expressed. As a further step, parameter individuals from a concrete device profile can be parsed into the PROFIBUS PA profile. These can then be mapped onto Condition Monitoring parameters, and the reasoner can perform a

verification of the mapping as well as infer further properties of the parameter according to the integration ontology. More details on parsing and mapping mechanisms are given in chapter 6.

5.3 CONCLUSION ON ONTOLOGIES

This chapter has shown the modeling work performed on device description languages and profiles in the automation domain. In order to evaluate the usefulness of any application that is founded on the models defined here, this section discusses the ease and difficulties of modeling the different device descriptions and profiles.

The device description languages have two broad strains of language structure. The first, and historically older, strain is the key value pair. This is sometimes seen in the form of the INI format, known for example from the Microsoft family of operating systems. Overall, this strain contains all description languages that have segments in an otherwise plain file which contain key value pairs, usually with defined keys and a set value range. The second strain contains a tree-like structure, which makes it suitable for representation in an XML-like structure.

The ease of modeling of the two strains of device description languages is quite different. While the tree-like structure of the latter strain is comparatively easy to map unto the likewise tree-like structure of an ontological class structure, the former strain demands more effort and concentration from the modeller. As an example for this, the GSD and GSDML formats can illustrate the issue. The GSD contains key value pairs. These can be mapped into an ontology model by defining the keys as `Thing` specialisations. However, when this is done, it becomes hard for the modeller to distinguish the different concepts properly. This also makes the mapping to other ontology models an arduous task. Because of the plain class structure, a lot of the relationships between different keys are implicit in the DDL. Expressing these implicit relationships demands a full in-depth understanding of the relationships in the first place. The modeling of DDLs of the first strain can therefore be considered a difficult task.

Where modeling the GSD is a difficult task, the GSDML has a much easier access. Due to the XML structure and class models given in the GSDML specification, inheritance and containment can be easily adapted from the specification into the ontology model. A difference that needs to be considered is that attributes need to be modeled as separate classes rather than as a property of a class. This is necessary for the task discussed in the following chapter, where a reasoner needs to be able to find equivalent classes. However, since this difference is uniformous for all possible class structures to be modeled, it does not raise the difficulty of the task considerably. It can therefore be assessed that the difficulty for modeling the second strain is easy.

As a side note, it should be mentioned that text mining approaches, that have helped ontology generation from scientific articles, in the medical domain e.g. [80], are not easily applicable to the wide range of models that make up the device description languages domain.

As has been mentioned, this modeling has not been performed and evaluated without a purpose. The following chapter utilises the models defined here.

6 A COMPUTER-SCIENTIFIC APPROACH FOR IMPROVING DEVICE INFORMATION MODELING IN AUTOMATION

As has been shown in chapter 4, several approaches for information modeling in the automation domain exist. However, none of these approaches has solved the challenge faced by device manufacturers yet [35].

Most mappings of models for automation devices undertaken today are performed by human experts, who have to define the mapping step by step, as the survey performed by Mühlhause [60] illustrates. This is not only a costly approach, due to important personnel bound up in the task, but it is also error-prone. Reducing the effort of the engineers as well as the likelihood of errors taking place in the process is the goal of an approach developed by the author and presented in this thesis. An important factor in the approach is the combination of different ontologies as discussed in depth by Ehrig [28]. The concept is also based on work published by Hahn et. al. [42] as well as Gössling and Wollschlaeger [38], [37].

Computer science has developed methods for transforming formalised data formats between each other. For any given format, the transformation to another format can be defined in a formalised manner. Taking this for granted, a higher number of formats leads to an exponential growth of the transformations that need to be specified. The phenomenon is illustrated in figure 6.1, where the addition of one format warrants three additional transformations to be defined. The approach discussed here is meant to overcome this growing complexity.

In order to reduce the transformations to be defined, a common ground for the data formats in a specific domain can be utilised. So for example, if a company needs to define different device

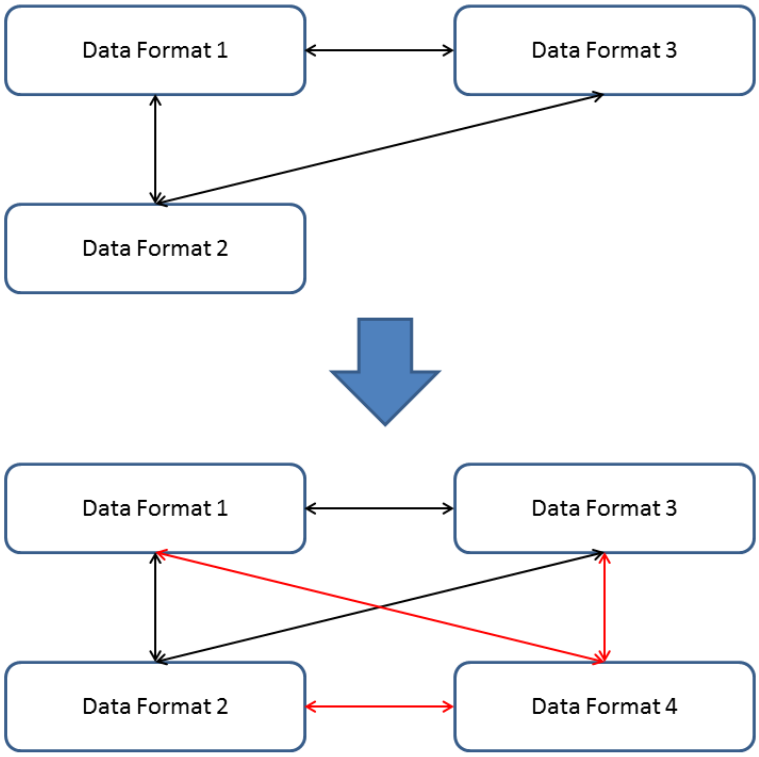


Figure 6.1: Illustration of Quadratical Growth of Transformations between Distinct Formats

descriptions, a generic device model can be employed as a central brokering point for all transformations. Building a generic model, however, can be arduous work. Every target data format has to be considered, something that is not practically feasible. Therefore, the search for an existing common model as a starting point can promise further simplification. Figure 6.2 illustrates the reduced effort when a generic model (given or distilled from different sources) can be used as a mediator between different data formats. When applied to the domain of ontologies, this model that is common for two specialised models can become an integration ontology [38]. This ontology integrates the concepts of its constituent ontologies and sets them into relation with one another.

The use of a generic model is not the only point of leverage to reduce efforts for data format transformations, however. It is also important to make each step as easy as possible and especially to support the personnel involved in minimising errors. Therefore, a high degree of computerisation is advised in both the definition and the performance of transformations. To support the process, a clear workflow for those steps that are not handled automatically is necessary. So if for example a new model of an existing format or entity is to be defined, the approach for generating the model is to be unambigiously defined.

In order to find solutions for two application cases, which are examined in close detail in the following sections 6.1 and 6.2, this thesis presents an approach that can improve the situation regarding data formats in the automation domain. This approach can be a basis for a long term

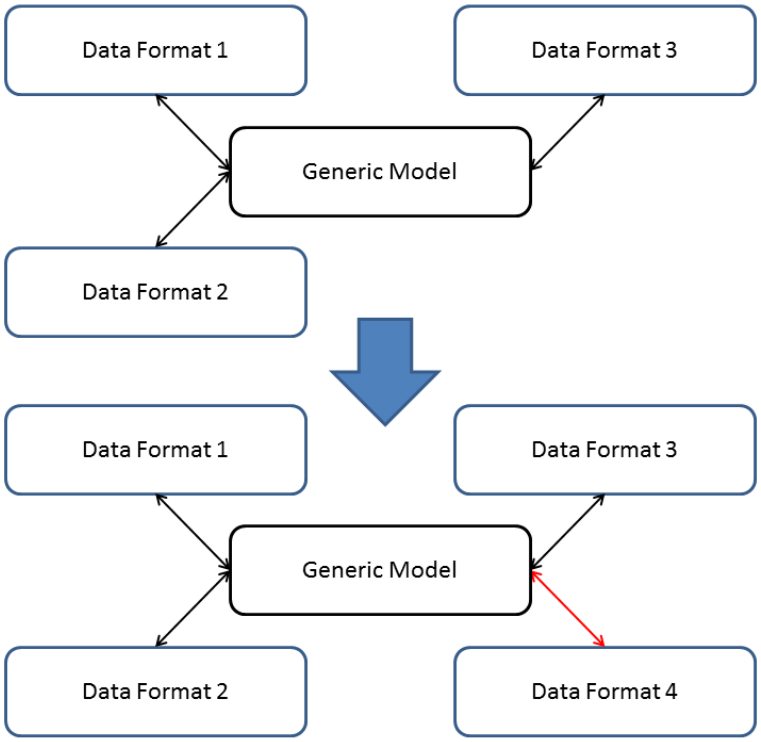


Figure 6.2: Illustration of Reduced Transformations with Utilisation of a Generic Model

improvement of information handling. The core of the approach is the employment of ontologies. For the exemplary use cases the ontologies presented in chapter 5 will be used.

The approach defined here offers a solution to the problem of transforming an instance of data format A into a semantically equivalent instance of data format B. This happens on two levels. On the *language level*, a correlation between the two formats is expressed through models. On the *instance level*, a set of tools employs said models to automatically transfer information from one format into the other. The instance level is dependent on the models from the language level to function. An instance of the data format is mapped onto individuals in the model of the data format. This is illustrated in figure 6.3. In the scenario defined, three models on the language level and two models on the instance level are necessary. Once these models are defined, there are only basic activities required before for all following instances, the transformation can be fully computed.

If a transformation between the data formats A and B were to be defined, the first step is to model each data format separately as an ontology. As has been discussed, the most common practice for doing this is to define an OWL ontology of the data format in question. In order to not open up a magnitude of possible models, these models have to adhere to a defined standard. Once the two formats have each been modeled, their models need to be mapped onto a common platform. This platform serves as an integration ontology, incorporating concepts from both models and defining equivalences between concepts. This can of course

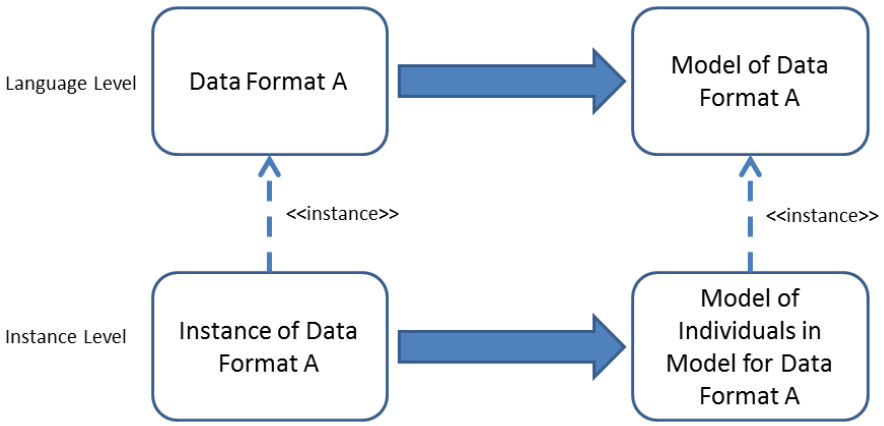


Figure 6.3: Illustration of Language Level and Instance Level

only be defined where information is represented in both data formats, but this limitation is natural in any transformation process. Once the integration ontology is built, the modeling on the language level is done.

For processing input instances for the data formats, a parser needs to be made available. Alternatively, information can be transferred manually by an operator, but this process is error-prone and will not increase the performance of the overall process. A parser for important data formats usually needs to be available for the utilisation of said format in the first place. The author can personally give witness that this is the case for the company he works in, for example. Such a parser needs to be adjusted to produce as output a set of individuals in the OWL format. In this process, the strict modeling paradigms of the language level need to be considered. When the modeling approach of the language model ontology is well defined, a parser can easily produce ontology individuals from parsed data tokens. This process is explained in more detail in the exemplary sections below. It is important to understand, though, that the parser needs to fit to the ontology model used, because the parser needs to have as output individuals that belong to this ontology model. It should also be noted that the reverse process, transforming individuals from an ontology into tokens of the data format, must be implemented as well. For data formats that are based on XML, the parser can be replaced by simply defining an XSLT for the same purpose.

Once the aforementioned tools are available, the work is done. Any instance of data format A can be parsed into individuals for the integration ontology between A and B. A reasoner can classify these individuals into classes originating in the ontology for data format B. The parser for data format B can then transform the individuals into data represented in data format B. Depending on the implementation level of the tool chain, no user interaction is necessary at all. Figure 6.4 gives an overview of the process.

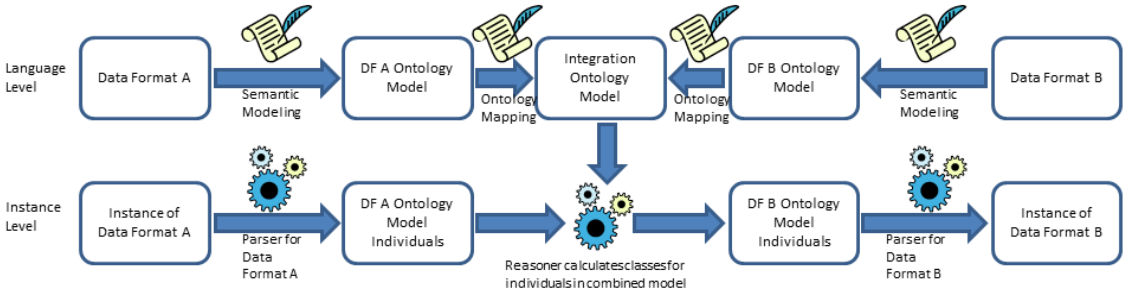


Figure 6.4: Generic Tool Chain for Data Format (DF) Transformation

It should be noted that all the processes on the language level in figure 6.4 involve a certain degree of engineering effort as illustrated by the sheet of paper. After a parser is provided, all processes on the instance level allow automated computing as illustrated by the cogwheel symbols.

The fact that the process on the instance level is able to run without user interaction alone does not justify the modeling effort that has to be spent on the language level. A fixed transformation method for data format A to data format B could be devised using less sophisticated modeling approaches and still perform the task correctly. Added value is gained from this approach, when it is used to integrate a higher number of formats. As has been discussed in figure 6.2, overall transformation work can be reduced in case a negotiating model can be employed. This can be ustilised with the integration ontology approach [38]. When an integration ontology is used, the parsers and models of the already handled data formats can simply be reused on the overall process. Only a model and a parser for the added data format need to be provided. The challenge is the utilisation of the integration ontology. Either a common model is already existant, which then can be utilised, or the integration ontology needs to be generated. For further considerations, we will for now assume that a common model exists.

Figure 6.5 shows what the effect of this is in practice. For the afore mentioned scenario of data formats A and B the tool chain is given as in figure 6.4. Now, data format C is to be transformed into data format B. All that needs to be added to the tool chain are the components used in the red, unfilled arrows. The rest of the components can be reused from the previous use case. Even better, if now data format A is to be transformed into data format C, no additional effort is needed at all. The components from the C to B transformation are now available and can be applied again for this case, as shown in the lower half of figure 6.5 and the filled red arrows.

The system architecture with the software components involved in the approach and the activities they perform for data file processing is shown in figure 6.6. With this approach, efforts for data formats transformation related to field devices can be reduced. The following section 6.1 elaborates a use case, in which the process of device engineering can be enhanced.

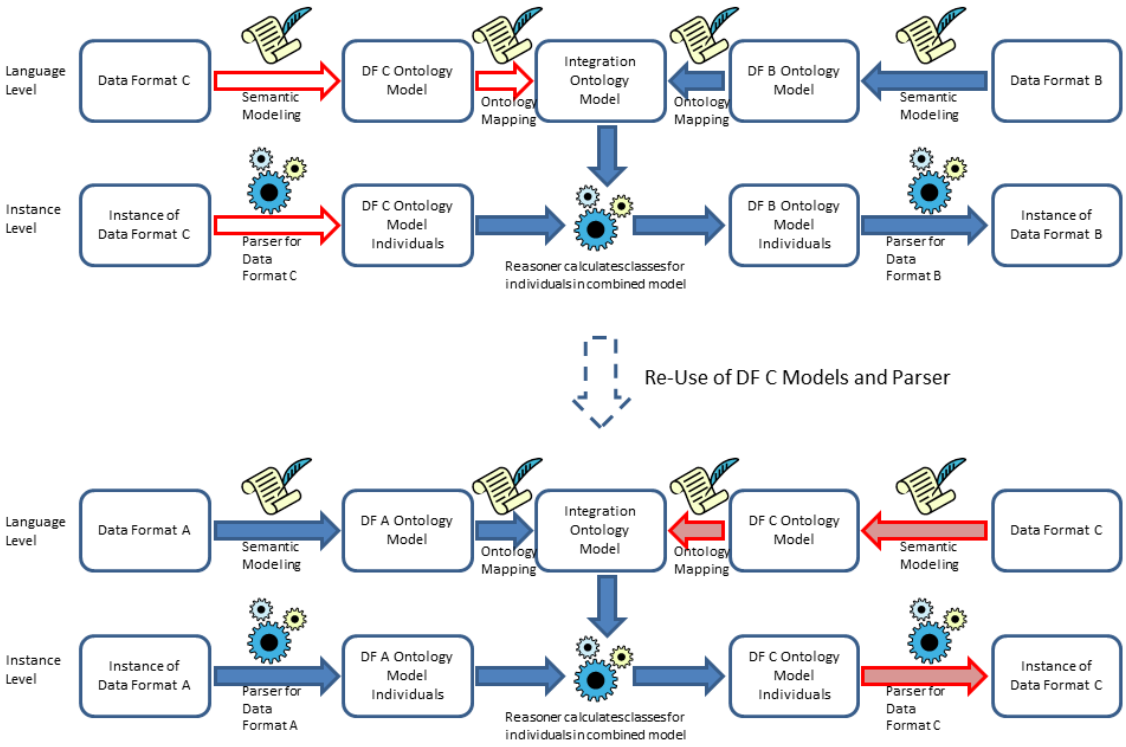


Figure 6.5: Expansion Scenario for the Generic Tool Chain for Data Format Transformation

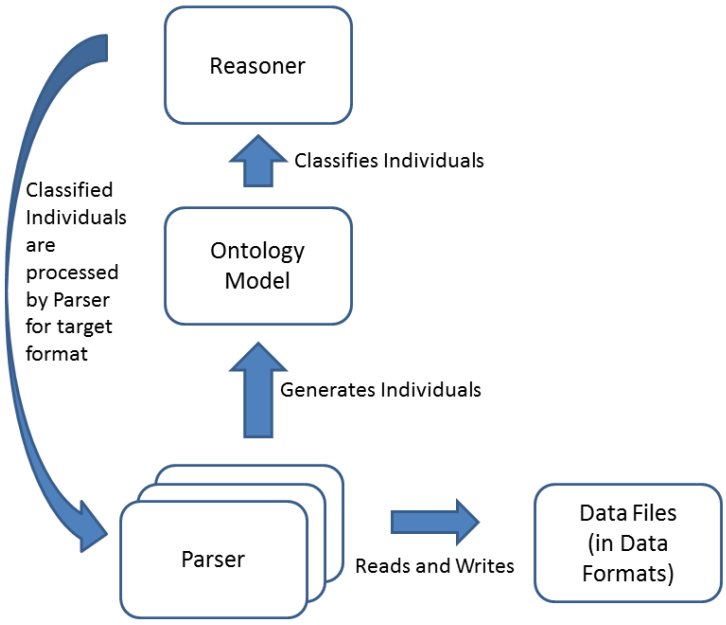


Figure 6.6: System Architecture for the Proposed Approach

6.1 DEVICE ENGINEERING WITH MODEL SUPPORT

The process of device engineering encompasses various tasks, but this section focuses on the data formats that have to be considered during the process. As has been illustrated in chapter 2, a device manufacturer has to address several device descriptions with different application areas if he wants to successfully market his device. Currently, this means that a lot of repetitive tasks need to be performed, as discussed e.g. in Hahn et. al. [42]. Here is discussed how the approach presented in this chapter can ease these tasks.

For the purpose of the application of the approach, a device description can be considered a data format. The generation of device descriptions is mappable on the task of transformation when one considers that the information that are represented in different device descriptions are often equivalent. For example, every device description has the possibility to store a device's manufacturer, as well as a device's identification. In addition, every device offers functionality that needs to be addressed through communication interfaces in some way in order to utilise it. These connection points between different device description formats can be subjected to transformations that reduce the effort of implementing this information into each format separately.

As a result, the transformation approach described in this chapter can be applied to device description languages, who define valid device description formats. Figure 6.7 illustrates how the approach can be applied to the domain of device descriptions. Device description languages take on the role of the data formats to be transformed.

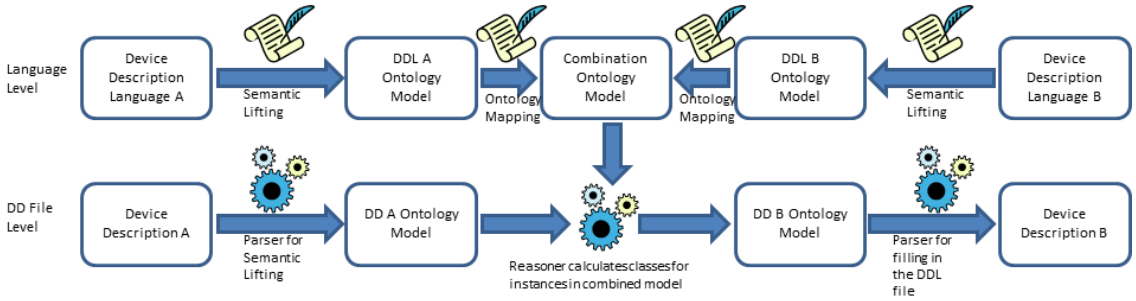


Figure 6.7: The Tool Chain for Device Description Transformation with Ontology Model Support (based on figure 6.4 with Device Description Languages (DDL) as specialised Data Formats (DF))

The application of the approach to device description languages has the advantage that the preconditions for maximum applicability of the transformation approach can be easily met. Device description languages (DDLs) are data formats that have a lot of common information shared between different languages, which allows for transformation to bring results. On the other hand, each DDL has an at least slightly different information content as compared to other languages. This makes the transformation through several languages unfeasible, because it

bears the risk of information being lost that could be transformed by a direct transformation. And finally, a common model that can be utilised for creating an integration ontology for device description languages can easily be created, which makes the transformation approach efficient. The common model can be created by orienting on the ISO 15745 standard that has been discussed in section 2.3, and that has been modeled as an ontology according to section 5.1.1.

With the basis model of ISO 15745, an integration ontology can be created that allows mapping of several device description languages onto each other. While some, like GSD and GSDML, have a lot of similarities regarding the information they model, others, like GSD and CANopen, do not have as much in common. This makes it paramount to have a transformation between every format, a fact that is important for the evaluation of this approach as discussed in chapter 7. If several transformation steps from one format to another were performed, more and more information might be lost with every step, because the intermediate formats may not model information both the source and the target do. Mapping two device description languages onto each other in a transformation according to the approach presented here is a lot of work. However, with the variety of device description languages on the market and in practical use, the reduced effort when transforming between any two of these languages is a robust way to reduce the overall effort. This, of course, makes the approach interesting especially for device manufacturers who address a variety of different DD technologies in their portfolio.

An example of the efforts that need to be spent can illustrate the leverage the approach has. If a device manufacturer wants to transform GSD and GSDML into each other, the effort for this approach is higher than if he would just define a simple transformation between them. However, if he also wants to have a transformation between GSD and CANopen as well as GSDML and CANopen, the effort for these transformations is lessened significantly, because he can reuse the GSD and GSDML models and parsers. All he needs to add are a CANopen model and associated parser, as well as a mapping of the CANopen model to the ISO 15745 based common model he uses as an integration ontology. Adding more DDLs like FDCML is just as easy as adding CANopen, and the transformations between all these languages are given in a way that demands minimum effort for applying them.

A case where the utilisation of a given base model for an integration ontology is not as easy as for the domain of device descriptions is given in the following section 6.2.

6.2 ACCESSIBILITY OF MAINTENANCE INFORMATION THROUGH MODEL SUPPORT

In section 5.2, the existence of a guideline for communicating Condition Monitoring data from field devices has been discussed [76]. For device manufacturers, there has been an increasing

market demand for providing machine health information to overlying information processing systems [36]. In order to fulfill this demand, not only need the devices have sensoric capabilities for measuring condition-relevant data, the devices also need to have the capabilities for communicating this data to a condition monitoring system.

To allow this critical communication task to be fulfilled is the goal of the aforementioned guideline. It defines function blocks for Condition Monitoring that in turn define interfaces for the information flow towards a condition monitoring system. While these interfaces are clear, they still need mapping towards profiles that are specific to the communication technologies that will be used. These profiles still need to be defined by fieldbus organisations or similar interest groups. An example would be the PROFIBUS PA device profile [47]. Since at the time of writing this thesis no target for the mapping has been decided by the responsible parties, this circumstance opens up an opportunity for facilitating this process.

Variables and parameters of field devices need to be mapped to the device profile for Condition Monitoring, in order for the communication to take place. This mapping task is different from the task of mapping device descriptions onto each other. Still, since it is a mapping task, the base approach can be repurposed from the previous section. With the ontology model defined in section 5.2, the inputs and outputs of function blocks for Condition Monitoring become defined as parameters in the context of device profiles. If a concrete profile technology is defined, these parameters can be defined equivalent to device variables in an integration ontology. That way, a reasoner can identify the communication parameters of a field device, modeled as variables in the device description, that is equivalent to an input or output parameter of a condition monitoring function block. This should facilitate the engineering of Condition Monitoring solutions based on the industry guideline paper significantly.

The task of defining the connection between a device's variables and the parameters of a profile is by principle similar to the mapping between different device description languages. The parameters of a profile then replace the entities such as variables from a device description. Therefore, a connection between a device and a communication profile for the device can be drawn from the device description by finding a mapping between the device's variables and the profile's parameters. The principle is illustrated in figure 6.8. When an ontology model for a device description is given, only the profile needs to be semantically annotated. Through the usage of a reasoner, connections between device description languages can be utilised for mappings from different DDLs to a device profile. That there is basically not a difference between mapping between Device Description Languages and Profiles is illustrated in figure 6.9.

The process of what is mapped onto what may not be intuitively understood from figure 6.9. Therefore, in figure 6.10 the figure has been overlaid with concepts that are mapped onto each

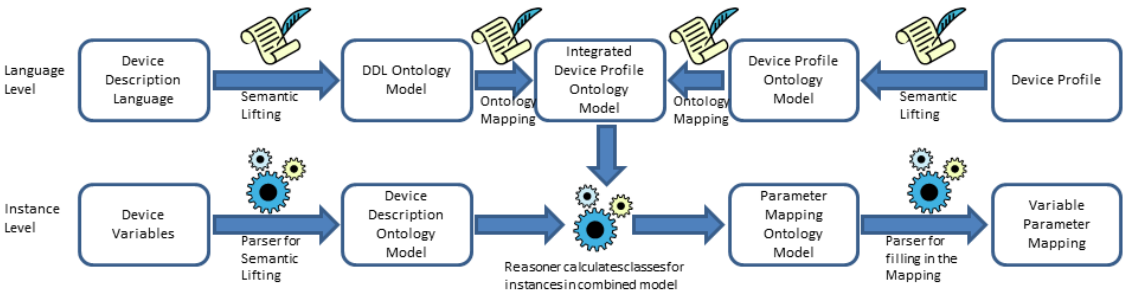


Figure 6.8: The Tool Chain for Mapping a Device Description's variables onto a Device Profile (based on figure 6.4 with Device Profiles and Device Descriptions as specialised Data Formats (DF))

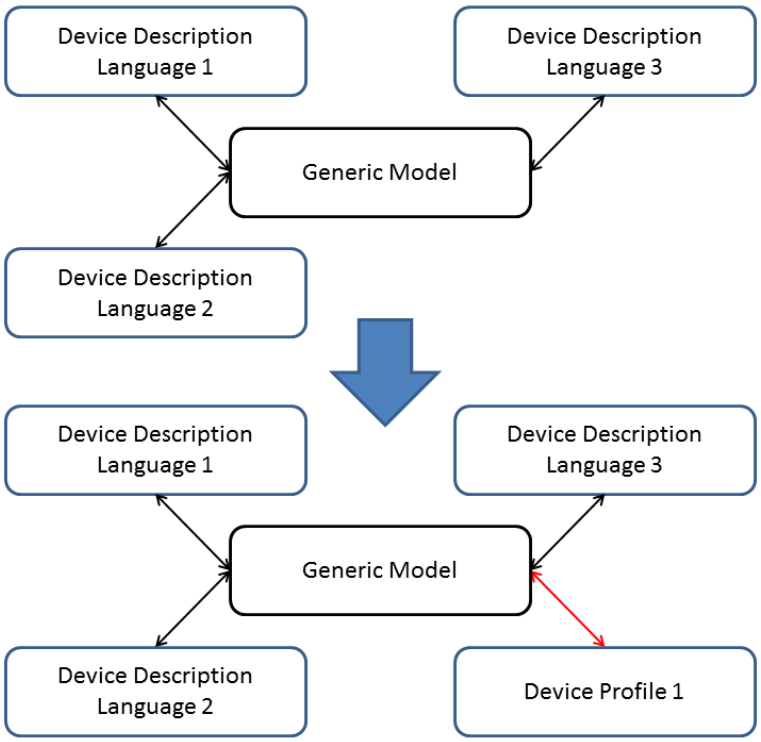


Figure 6.9: Illustration of the Seamless Integration of Profiles into the Eco System of Device Description Languages

other through the Generic Model. Different device inputs, described in the specific ways of the device description languages, are mapped onto one concept in the generic model. If a device profile is added, the concept in the generic model is mapped onto a functional parameter in the profile. This parameter, of course, should be an input parameter as well, in order to allow the mapping to be consistent. That way, in one step a relationship is created between the functional parameter and all the inputs defined in the different device descriptions.

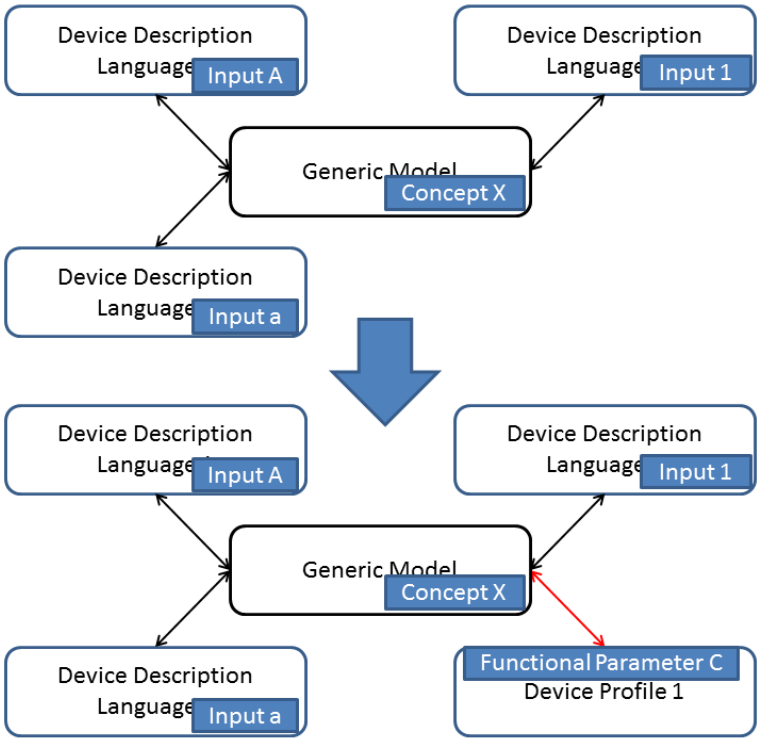


Figure 6.10: Illustration of the Integration of Profiles with Exemplary Mapping

Once a profile is semantically annotated, it can make sense to map similar profiles onto it on a semantic level as well. The instance level of this transformation consists of the mapping of parameters onto a device's variables. The mapping of the device variables to the profile parameters is called a parameter set for our considerations. Figure 6.11 illustrates the generation of a parameter set for one profile from the parameter set for another profile with our tool chain.

A likely use case for the mapping between profile and device description languages can be discussed by the example of the aforementioned Condition Monitoring profile. This new profile is to be adapted by field device manufacturers. The manufacturer knows the profile, by which other solutions will recognise the functionality assigned to device variables. At the same time, the manufacturer provides the device descriptions for his device, which list the accessible variables according to their target fieldbus technology. In order to make sure that the addressing of the profile's functionality is coherent with all device descriptions available for a device, the manufacturer needs to map the profile onto the device descriptions as well as check the

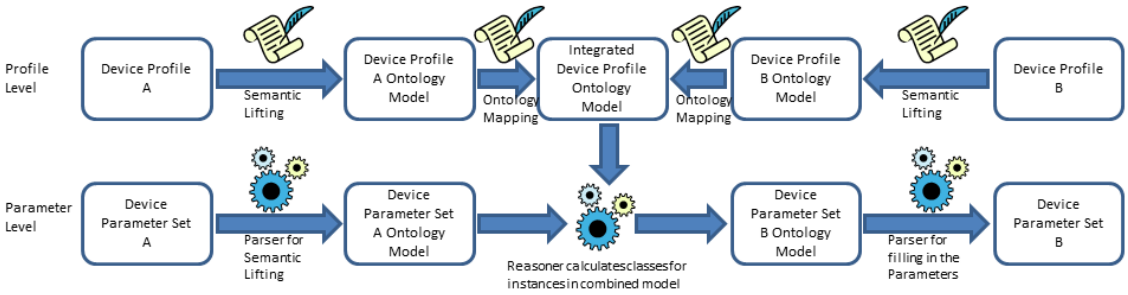


Figure 6.11: The Tool Chain for Device Profile Transformation with Ontology Model Support (based on figure 6.4 with Device Profiles as specialised Data Formats (DF))

consistency of these mappings. The mapping might be even easier in case an ontology for device functions in general were provided. This would demand a methodology to properly model a device's function on an ontological level, a task that is beyond the scope of this thesis.

As an example for the mapping between profiles, the scenario can be expected to be the mapping from the Condition Monitoring profile to the PROFIBUS PA device profile. This scenario will be the basis for an example in section 6.3, and it is illustrated in figure 6.12.

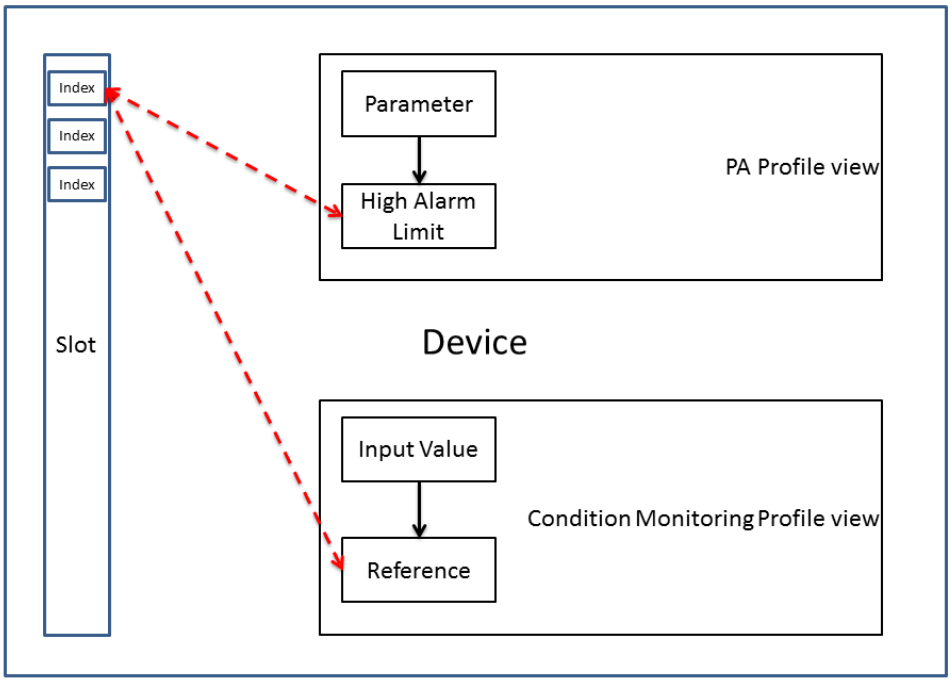


Figure 6.12: Exemplary mapping scenario PROFIBUS PA and Condition Monitoring

The checking of mapping consistency is not a trivial task to perform. Therefore, if a device manufacturer needs to adopt a new profile, he will want to ease the process of bringing together the profile and his device descriptions for the device in question. Of course other

profiles for the device need to be consistent as well. Under these circumstances, it can be feasible to utilise reasoners to facilitate the necessary processes. An evaluation of the approach is discussed in the following chapter.

6.3 EXAMPLE AND DISCUSSION

In order to make the overall approach clearer, this section gives an example of the transformation and mapping process. After the example, the trappings of the approach are discussed.

As an example a closer look is provided at what happens if for a temperature sensor manufactured by the company ABB, for which a GSD is given, information has to be moved into a corresponding GSDML file. For an ABB TF12 temperature sensor, a given GSD contains the following key value pairs.

```
#Profibus_DP

GSD_Revision = 3
Vendor_Name = "ABB Automation";
Model_Name = "TF12 Temperature Transmitter";
Revision = "1.1.1"
Info_Text = "Temperature Device"
OrderNumber = "ABB, TF12"
Ident_Number     = 0x04c4
```

It can be seen that if the vendor name is to be filled in for a corresponding GSDML file, it will also have to be `ABB Automation`. However, it is undesireable to resort to copy-pasting it. In order to employ the models given in chapter 5 and the approach given in this chapter, first a GSD parser must be provided. An example parser has been developed in the course of this thesis. Due to a lack of robustness of the parsing algorithms, device vendors will likely prefer their in-house parsers. However, the parser is able to provide output for the ontology model of the GSD DDL. After parsing the GSD (step illustrated in figure 6.13), the vendor name is then encoded thus. (URIs have been shortened with three dots for enhanced readability.)

The string literal is assigned to the individual for the vendor name for the TF12 input file (the parser allows the definition of such prefixes):

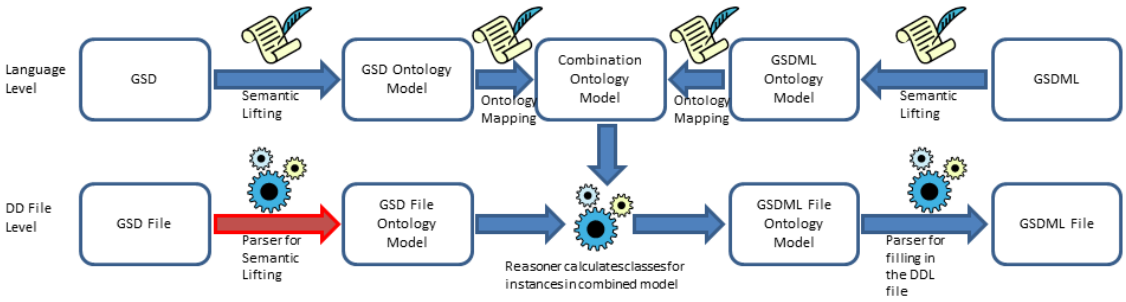


Figure 6.13: Example - Parsing the GSD File

```
<DataPropertyAssertion>
   <DataProperty IRI="http://www.../GSD_ontology#hasVendor_Name"/>
   <NamedIndividual IRI="http://www.../GSD_ontology#TF12_Vendor_Name"/>
   <Literal datatypeIRI="&xsd;string">ABB Automation</Literal>
</DataPropertyAssertion>
```

The TF12-specific vendor name is assigned to the class for vendor names in the GSD:

```
<ClassAssertion>
   <Class IRI="http://www.../GSD_ontology#Vendor_Name"/>
   <NamedIndividual IRI="http://www.../GSD_ontology#TF12_Vendor_Name"/>
</ClassAssertion>
```

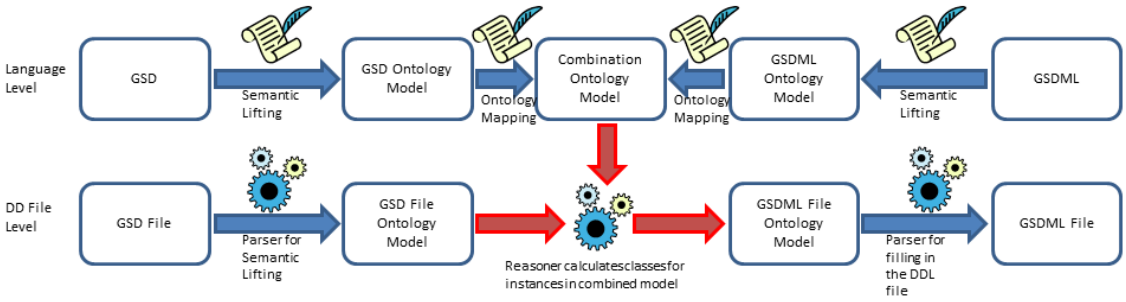


Figure 6.14: Example - Reasoner Application

When now the equivalency between the GSD and the GSDML vendor names is expressed, a reasoner will automatically infer (step illustrated in figure 6.14) the class assertion for the GSDML:

```
<ClassAssertion>
   <Class IRI="http://www.../GSDML_ontology#VendorName"/>
   <NamedIndividual IRI="http://www.../GSD_ontology#TF12_Vendor_Name"/>
</ClassAssertion>
```

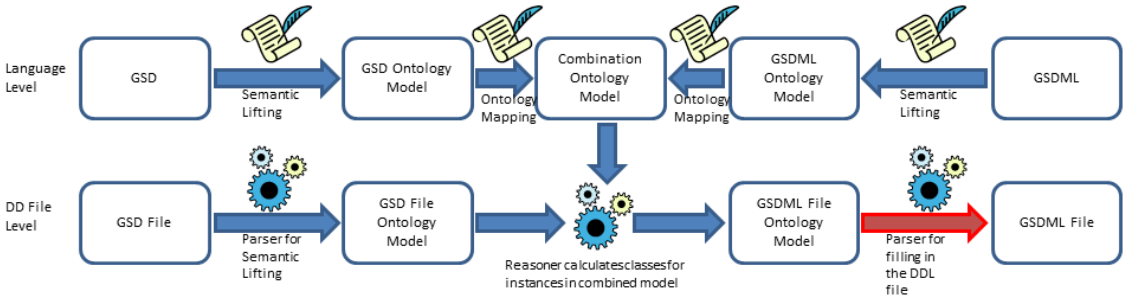


Figure 6.15: Example - Filling in the GSDML

Now the individual with the asserted data property to the string `ABB Automation` is classified as a GSDML vendor name. The parser for the GSDML (step illustrated in figure 6.15) then fills in the name into the GSDML file he creates from the individuals assigned to GSDML-specific classes.

```
<DeviceIdentity VendorID="" DeviceID="0x04c4">
  <InfoText TextId="TF12 Temperature Transmitter" />
  <VendorName Value="ABB Automation" />
</DeviceIdentity>
```

As can be seen, the vendor ID has not been transferred, because in general, the vendor ID is typically different for different DD technologies. This information would have to be hard-wired into the technology-specific parsers of the vendor in question in order to be filled in automatically, an undesireable situation, because changes in the company will demand adaptation.

As a second example, the mapping of profiles is illustrated here. In order to produce an example, a likely use case for the mapping of profiles as discussed in section 6.2 has been identified. The scenario in general is illustrated in figure 6.16.

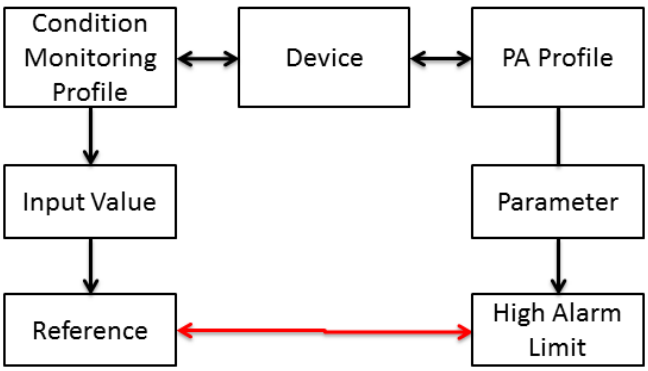


Figure 6.16: Example - Mapping PROFIBUS PA profiles and Condition Monitoring profile

In PROFIBUS PA profiles, sensors can get a limited measurement range through sensor limits. These limits can be used to apply condition monitoring to these sensors. The XML schema file for PROFIBUS PA transmitters defines the following for sensor limits.

```
<xs:element name="LOWER_SENSOR_LIMIT">
  <xs:complexType>
  <xs:simpleContent>
<xs:extension base="xs:float">
  <xs:attribute name="RelativeIndex" type="xs:int" use="required" fixed="18"/>
<xs:attribute name="Readonly" type="xs:boolean" use="required" fixed="false"/>
</xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:element>
```

Our integration ontology then must be adapted to the aspired task by allowing a data property for the identification of thresholds, to be expressed via a boolean data property. If the lower sensor limit is then parsed as an individual into the OWL file for the integration ontology, it is expressed as shown below.

```
<owl:NamedIndividual rdf:about="http://www.../Profibus_PA#LOWER_SENSOR_LIMIT">
  <rdf:type rdf:resource="http://www.../Profibus_PA#Parameter"/>
  <isThreshold rdf:datatype="&xsd;boolean">true</isThreshold>
  <Condition_Monitoring:hasParameterName>LOWER_SENSOR_LIMIT
   </Condition_Monitoring:hasParameterName>
</owl:NamedIndividual>
```

After a reasoner has processed this input, the following assertion has been inferred.

```
<ClassAssertion>
   <Class IRI="http://www.../Condition_Monitoring#THRESHOLDS"/>
   <NamedIndividual IRI="http://www.../Profibus_PA#LOWER_SENSOR_LIMIT"/>
</ClassAssertion>
```

As has been illustrated, the approach presented here can automatise tasks in the generation of device descriptions for device vendors. At the same time, consistency over different technologies is ensured, something the current situation, where different developers provide

different DDs, cannot provide adequately. The approach therefore can be seen as an important step towards a better device engineering process. The application to the domain of profiles shows potential as well. However, it should be noted that due to reasons discussed above, this field has not been investigated as thoroughly. Results for profile mapping will have to be evaluated in practical applications, once they are properly defined by industry demand.

When the application of the approach to profiles is refined, it will have to define a work flow for integrating different sources for filling in information. Sometimes the utilisation of a single integration ontology may not be feasible. In that case, it has to be ensured that the inputs of different integration ontologies are added up without losses, while at the same time preventing contradictions between the different sources. While in a few cases an added consistency check for contradictions can be helpful, in general the input sources for the generation of results need to be complementary.

However, the applicability of the approach for device descriptions has been evaluated. If work can be saved with this approach, which is not effortless due to the model generation and the utilisation of the integration ontology, will be discussed in the following chapter.

7 EVALUATION OF THE PROPOSED APPROACH

Most evaluations of semantic technologies are based on performance measures [5], [63]. This kind of evaluation bears no practical merit for the approach presented in this thesis. While one could consider how many seconds a reasoner needs to classify the tokens from two data formats onto each other, the practical value of knowing it is diminished by other factors. If for two approaches an engineer has a difference in workload measured in hours to days, he cares little about the amount of seconds a computing tool needs to present the results. While user experience can be of import in some use cases in the automation domain, transformation of data formats is not one of them.

Because the computing performance of our models can be neglected for an evaluation, another measure needs to be considered when weighing the approach presented here against the state of the art. What is really relevant to companies is the time their employees need to spend on the task in total, or the *effort* needed. Sadly, automation companies are quite secretive about their internal work flows regarding data format transformation. With what has been published and some small insights (like the citation in chapter 2), little can be said in hard facts regarding transformation efforts. Therefore, this chapter needs to operate on an abstract level and supplements it with reasonable assumptions.

The considerations in this chapter base on the questions that have been postulated in chapter 1. The first question was to what extent can an engineer be supported in the generation of several device descriptions for a single device? This question is the basis for a set of mathematical assumptions presented here. How this chapter shows the validity of the approach presented in this thesis through considering the first question will be discussed in the following stanza. The second question was how can an enhanced modeling approach for device descriptions and

profiles be utilised to improve the handling tasks in the later phases of a device's life time? This question cannot be answered with the same level of quantitative considerations as the first one. However, this question is answered, though not with sufficient proof due to a lack of validation possibilities, in section 6.2 of the previous chapter 6.

Regarding the first question, more thorough considerations have been made about the validity of the approach. The idea of the following exposition is to calculate the effort a device engineer has when he has to generate several device descriptions for a single device. The calculations use variables for the actual effort, so that formulas can be derived whose validity is not diminished by assumptions. These formula can be used to calculate the applicability of the approach by device manufacturers. In addition, assumptions are made about likely effort figures in relative terms. These figures are then used to explain why the approach presented in this thesis is considered to bear advantage for device manufacturers by the author. For that, different scenarios of assumed relative efforts are calculated and illustrated. From this point onward, the formulas discussed will be derived.

The effort needed to generate transformations between each data format is dependent on the completeness of the transformations themselves. If the transformations are symmetrical and transitive, there is no need to define transformations between every single data format in question. (For a mathematically rigorous definition of these terms, see Bosch, Lineare Algebra [6], fourth edition, p. 66, e.g.) However, while commutativeness is common, in most cases there will be no transitive transformation possible, because for example one format does not support the encoding of information that was available in the original format. In those cases, a chain of transformations is not sufficient to transform a maximum of information content, and therefore the situation of figure 6.1 with the quadratically growing complexity is valid.

The formal proof of the quadratical growth of transformations follows the very first example of the mathematical induction found in mathematical text books. (The example used here is Königsberger [53].)

If for a set of n data formats with given transformations, a new data format needs to be added with one transformation to each of the existing formats, there need to be n new transformations. This means the number of transformations *Numtrans* for n data formats is $\frac{1}{2}(n-1)n$

Proof:

$A(1) : \mathit{Numtrans}(1) = \frac{1}{2}(0)(1) = 0$ is true, because with just one format, there needs to be no transformation.

$A(n)$ *to* $A(n+1) : \mathit{Numtrans}(n+1) = \mathit{Numtrans}(n) + n = \frac{1}{2}(n-1)n + n =$
$\frac{1}{2}(n-1)n + \frac{1}{2}2n = \frac{1}{2}n((n-1)+2) = \frac{1}{2}n(n+1)$ q.e.d.

On the other hand, the situation found in figure 6.2 only has a linear growth, because it simply leads to $Numtrans(n) = n$. We can therefore see that the sheer number of necessary transformations can be reduced by one whole order of magnitude. This has little relevance for the praxis as it is, however, since it says nothing about the quality of the transformations, neither in correctness nor in the effort actually put into generating the transformation. To achieve a measure of the work saved by a reduction of transformations to be generated, the effort needed to generate each transformation needs to be taken into account.

In order to do so, *Numtrans*, the number of transformation needed, is supplemented with *Timetrans*, the time times persons that is needed to generate one transformation. A sophisticated approach like the one presented in chapter 6 usually has a higher *Timetrans* than a more simple approach.

If we now assume that for two groups of transformations A and B is valid:
$Timetrans(a) \geq Timetrans(b)\ \forall a \in A \wedge b \in B$ Then we can say with $|A|$ being the number of transformations in A that

$$\sum_{a \in A} Timetrans(a) \geq \sum_{b \in B} Timetrans(b)\ \forall A, B\, |A| = |B|$$

.

This means that for the same number of transformations, if we know the effort for one transformation is higher in one group, the overall effort for that group as compared to the one with lesser effort elements is higher as well. This means in order to be competitive, the higher effort group of transformations will have to have a significantly lower number of transformations. For further considerations, we define the summary and the average time effort needed for a group of transformations. For X as a group of transformations we define

$$Sumtimetrans(X) := \sum_{x \in X} Timetrans(x)$$

and

$$Avgtimetrans(X) := \frac{Sumtimetrans(X)}{|X|}$$

With these terms defined, we can again consider two groups of transformations A and B. We assume that $Avgtimetrans(A) > Avgtimetrans(B)$. If we want to know for which such groups is valid $Sumtimetrans(A) < Sumtimetrans(B)$ we have to take into account the number of elements in each group $|A|$ and $|B|$, because this is the factor by which both measures are related. For further steps we can assume that all variables are bigger than zero, since we only deal with groups of transformations that have at least one member. We can deduct

$$Avgtimetrans(A) > Avgtimetrans(B)$$

$$\Leftrightarrow \frac{Sumtimetrans(A)}{|A|} > \frac{Sumtimetrans(B)}{|B|}$$

$$\Leftrightarrow \frac{Sumtimetrans(A)}{Sumtimetrans(B)} > \frac{|A|}{|B|}$$

$$\Leftrightarrow \frac{Sumtimetrans(A)}{Sumtimetrans(B)} - \frac{|A|}{|B|} > 0$$

If we now want to achieve $Sumtimetrans(A) < Sumtimetrans(B)$, we know that the first fraction in the term will have to be smaller than 1. Therefore, the second fraction needs to be smaller than 1, too, in order to fulfill the condition above. We can therefore formulate the condition for the preferability of A over B:

$$1 > \frac{Sumtimetrans(A)}{Sumtimetrans(B)} > \frac{|A|}{|B|} > 0$$

The advantage of this formula is that it is independent of the concrete measure in which *Sumtimetrans* is given, because any scalar factor is negated by the fraction of two *Sumtimetrans* results.

If we now want to apply this formula to data format transformations, we know the number of transformations $|A|$ and $|B|$ for the groups with the desired properties. We can set in for $|A| = n$, because this is the more complicated approach with a common generic model, and for $|B| = \frac{1}{2}(n-1)n$ as we have proven above for the comparatively easier case of a group of mutually transformable formats. If we use these formulas, based on n being the number of formats to be transformed, we can further transform our condition:

$$1 > \frac{Sumtimetrans(A)}{Sumtimetrans(B)} > \frac{|A|}{|B|} > 0$$

$$\Leftrightarrow 1 > \frac{Sumtimetrans(A)}{Sumtimetrans(B)} > \frac{n}{\frac{1}{2}(n-1)n} > 0$$

$$\Leftrightarrow 1 > \frac{Sumtimetrans(A)}{Sumtimetrans(B)} > \frac{2}{n-1} > 0$$

The resulting condition of $1 > \frac{2}{n-1}$ for our desired properties is fulfillable for $n \geq 4$. Thereby we were able to deduct a minimum number of formats to be transformed into each other until the sophisticated approach we described with group A can possibly be useful. In order to determine a break even point, the real efforts $Timetrans(x) \forall x \in A \vee x \in B$ would need to be known. Hence, we cannot sufficiently conclude this calculation without either using excessive industry experience, which is not published, or at least making some assumptions. Due to the unpublished nature of industry figures, we will calculate a theoretical case with assumed values, in order to demonstrate how our considerations can be utilised to determine the effectiveness of the approach from chapter 6.

Note that the following statements are assumptions. We assume that the effort for a transformation in group A we know an average effort of $Avgtimetrans(A) = 4$ person working days. In contrast, for group B we assume an average effort of $Avgtimetrans(B) = 2$ person working days. Both groups of transformations are defined over the same n data formats. As a result, we can derive

$$Avgtimetrans(A) = \frac{Sumtimetrans(A)}{|A|} = \frac{Sumtimetrans(A)}{n} = 4$$

$$\Rightarrow Sumtimetrans(A) = 4n$$

and

$$Avgtimetrans(B) = \frac{Sumtimetrans(B)}{|B|} = \frac{Sumtimetrans(B)}{\frac{1}{2}(n-1)n} = 2$$

$$\Rightarrow Sumtimetrans(B) = (n-1)n$$

We can now further calculate that for our desired condition

$$Sumtimetrans(A) < Sumtimetrans(B)$$

$$\Leftrightarrow 4n < (n-1)n$$

$$\Leftrightarrow 4 < (n-1)$$

$$\Leftrightarrow 5 < n$$

As a result, for the assumptions we made with the transformations in A taking double the effort to be generated as compared to the ones in B, we can say that as soon as six (6) data formats have to be transformed, the approach presented in chapter 6 offers a more efficient way of generating the transformations than the approach of generating transformations between each format mutually. It should be noted, that double the effort is a very pessimistic assumption, chosen here to have an easily understandable relation. Since only one data format needs to be known for adding it into the system, the others can be taken for granted, this number is too high for most realistic use cases. It also takes into account that in the approach discussed in chapter 6, the integration ontology may need to be expanded in some cases, which raises the effort significantly. In cases where the integration ontology can be generated from the start, the effort per transformation lowers, and an added factor needs to be taken into account for the total effort of the sophisticated approach. This is likely to raise the efficiency of the approach for higher numbers of data formats that need to be transformed onto each other.

The discussion in chapter 5 mentioned a difference in effort for the two main strains of DDLs. Depending on the DDLs considered, a difference between the efforts needed is likely. The personal experience of the author, with a company having to address thirteen different DDLs, provides a mixture of key-value and tree-structured DDLs.

In order to illustrate the relationship between the approach presented in this thesis and the approach of transforming each data format onto each other format, some charts will show the development of the numbers for a growing number of formats to be considered. The first chart in figure 7.1 simply shows what it means that the number of necessary transformations in an assumed group B with the mutual transformation definition approach grows to the square in relation to an assumed group A which has the linear growth of the approach presented in this thesis.

The second chart in figure 7.2 illustrates the scenario which we calculated, with four days needed to generate each transformation on group A and two days needed for each

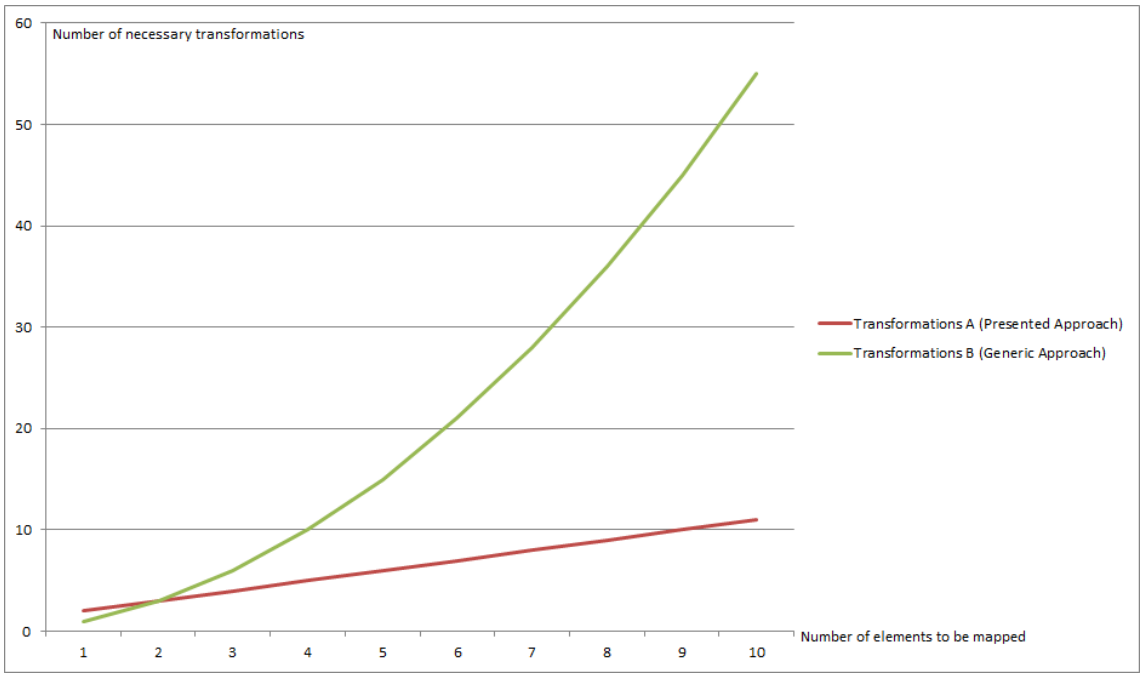


Figure 7.1: Illustration of Quadratical Growth of Transformations as opposed to Linear Growth with the Approach from this Thesis

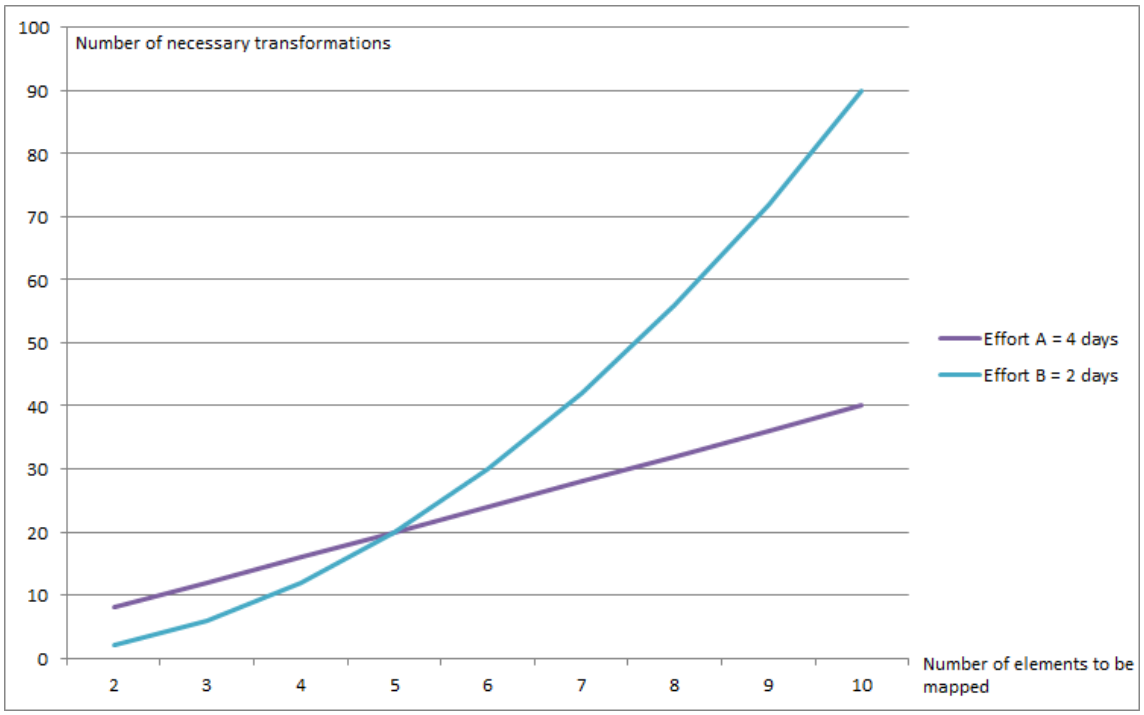


Figure 7.2: Effort of the Different Approaches with an Assumption of Double Effort per Transformation in A

transformation in group B. Note that the break even point for the approach defined in this thesis lies at six formats, as we calculated.

The third chart in figure 7.3 illustrates a scenario in which for a transformation in group A there is only a 50% overhead in comparison to one in group B, namely three days to two days. Note the break even is reached much earlier, as is to be expected.

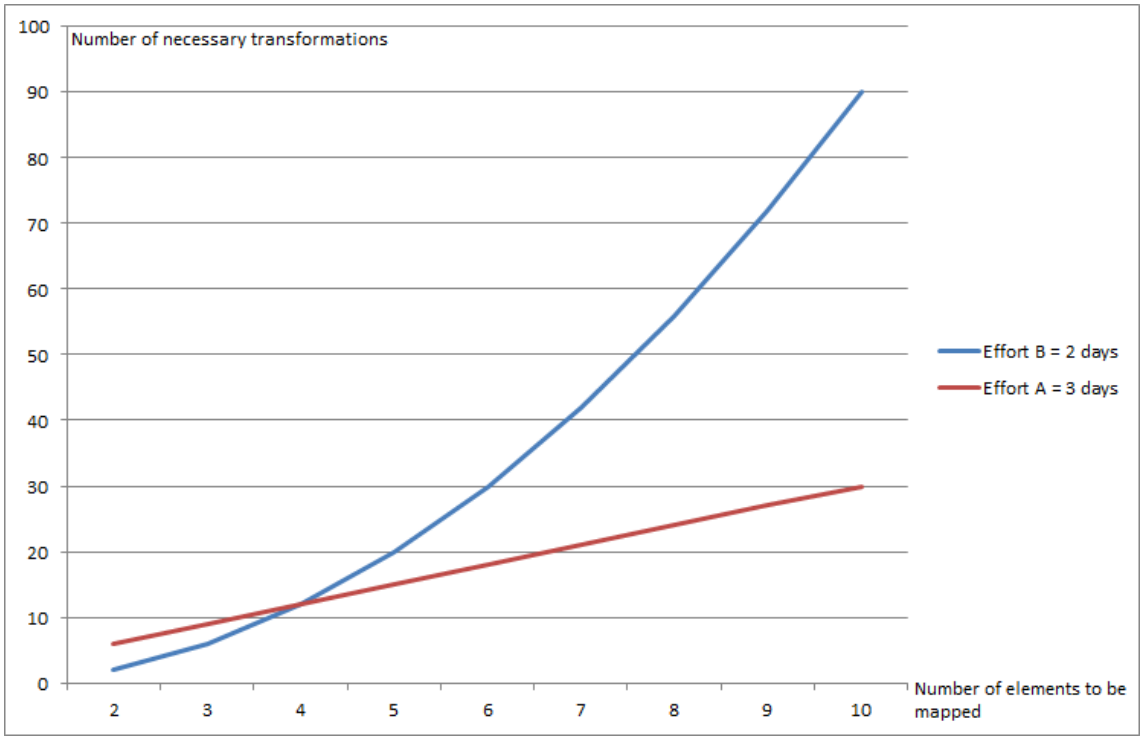


Figure 7.3: Effort of the Different Approaches with an Assumption of 50% Raised Effort per Transformation in A

The final chart in figure 7.4 illustrates a relationship of five days per transformation generation for group A, and three days transformation generation in group B. This distribution is of note because it shows the author's personal assumption of the effort distribution when the maintenance of the generic model for each transformation in group A is taken into account.

As can be seen, whenever a number of data formats needs to be mutually transformed onto each other, it is worth to take a look at the approach presented here for speeding up the process.

In section 6.2 it was shortly discussed that for the purpose of the approach presented in this thesis, a device profile can be compared to any data format. Of course, a profile is not generated by a device manufacturer, as the device descriptions are. Instead, the device manufacturer receives the profile from a specification body. The challenge then becomes to map the profile to the device's variables as well as to check the mapping for consistency with all other device descriptions and profiles relevant for a device in question.

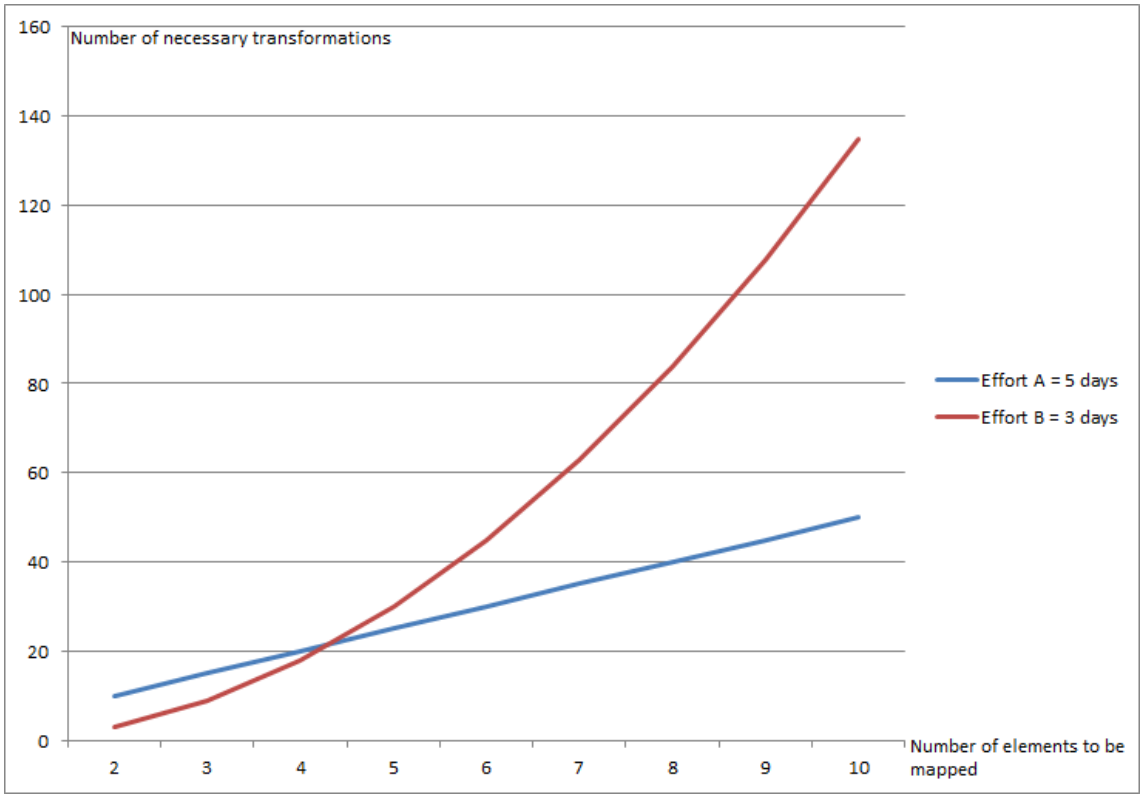


Figure 7.4: Effort of the Different Approaches with an Assumption of a five to three Effort Relationship

The consistency check between device descriptions and profiles thus needs to be considered when trying to evaluate the usefulness of applying the approach of this thesis to device profiles. However, a consistency check between two profiles or a profile and a device description has as unknown an effort as the mapping of two device descriptions onto each other. This leads to the situation that the considerations that have been discussed in this chapter regarding the mapping of two data formats can be applied to consistency checks as well. Added advantage is promised by the fact that a consistency check is a task that the human mind is not best suited for. Therefore, added decrease of effort can be achieved by letting the consistency checks be performed by a reasoner.

8 CONCLUSION

This thesis presents an approach for device information modeling in automation based on the computer-scientific concepts of ontologies and reasoners. As has been motivated in chapters 1 and 2, the problem of generating device descriptions is a task that bestows challenges upon the device manufacturer. Integration of the heterogeneous automation landscape is addressed on many ends, but the device description generation is an unsolved problem. In order to address this problem, chapter 3 introduces the computer scientific handling of semantics, ontologies and the technologies of the semantic web. The application of semantic approaches to the automation domain is exposed in chapter 4, according to the three dimension of integration in the automation domain.

After discussing the state of the art, chapter 5 describes ontologies that the author has modeled for this thesis. These ontologies deal with device models, device description languages and device profiles. These ontologies form the technological basis for a new approach the author devised for the generation of device descriptions. This approach is discussed in chapter 6. Device description language ontologies allow the automated generation of device description models, constituting a set of individuals in the DDL ontology. Through the employment of an integration ontology, a reasoner can automatically map the device description models onto another DDL ontology, thereby transforming the information content into another device description language. The chapter also discusses the widening of the scope of the approach to the generation of device profiles. The approach is motivated by calculating efforts for the approach compared to a full cross-wise mapping of DDLs by handish effort in chapter 7. One finding is that the widening of the scope of the approach is a way to make its utilisation even more efficient.

8.1 DISCUSSION

The main contribution of this thesis is an approach for easing the generation of device descriptions on the device vendor's side. Only after the work on this thesis has begun, the author gained an inside view on how important this issue really is. Device vendors have not solved the DD generation issue, and are striving to streamline and optimise this task. This relevance of the problem means that even small steps can be of help. The approach presented does not rid the device vendor of all effort. Modeling of device descriptions is a task that the standardisation bodies in the fieldbus and automation domains might be reluctant to do. Thus, the device vendor himself would be stuck with the task, giving this approach a high starting investment.

Despite the investment needed to get into this approach, in the long run and especially for continued device development this effort will pay off, both in the effort needed for future devices or device description languages and in the quality of the device descriptions provided. The biggest hindrance can be the utilisation of computer-scientific concepts. Most engineers of a device vendor have training in mechanical or electrical engineering. Since computer science is gaining a bigger role in the automation domain with time, however, this problem is likely to diminish in the near future.

Despite the validation work done in this thesis, a bigger scale application of the approach might be needed to evaluate its applicability on a corporate level. There remains a risk when academic approaches such as the one presented here are brought into the field. The task of tackling this risk is a matter that goes beyond the scope of research and penetrates what the industry considers development. It can therefore not be tackled by this thesis alone.

Mixing different sources always bears the risk of finding that they have very little in common. A mapping between two models have almost no commonalities cannot reduce efforts in model generation. Therefore, the stepping beyond the domain of DDLs and adding profiles bears problems that this thesis, with its set scope, could not solve fully. It will therefore be discussed in the next section as future work.

8.2 FUTURE WORK

The approach presented in this thesis will not immediately solve all the problems the device manufacturers have with fieldbus connectivity and device descriptions. One issue is the necessary modeling of the device description languages. While this task is not overly taxing, a device manufacturer would need to make sure that he has access to experts both for the device,

the description language and the modeling language. While this is not always a given, it still is within the capabilities of most device manufacturers. Another option is to contract experts from outside the organisation to do the task of modeling. It may be thinkable that fieldbus organisations take an interest into modeling their device description languages themselves, to reduce market barriers for their technology on the manufacturer's side. In any case, the potential for deeper research on the approach presented can be identified. Ontology generation on heterogenuous models is a field for future work.

The second research topic identified by the author that promises good results for future work is the widening of the approach that is presented in this thesis. While the application of the approach for the generation of device descriptions and for generation of parameter sets for device profiles is discussed in this thesis, additional data sources and targets warrant further research. For example, bus-connector-relevant information might be extracted from a device manufacturer's database in order to fill the gaps the current state of the approach leaves on the bus-technological level. Because most device manufacturers use standard bus connectors for all their devices, additional reduction of generation effort can be achieved this way. In addition, as has been mentioned in chapter 6, a sufficiently expressive ontology model for a device's function in general is a promising area of research.

The application of the approach to other integration tasks has not been the subject of this thesis. Future work might identify potential for the transposition of the approach. Especially the application of the approach to other levels of the classical automation pyramid could be of interest. This field is likely to yield synergies with the field of product lifecycle ontologies. Vertical integration could be facilitated with ontologies present on different levels of the automation pyramid.

In order to make a large set of ontologies manageable, research towards the management and mapping of ontology models needs to be intensified. For a limited use case, such as the ones described in this thesis, ontologies can be handled by experts. If a broader scope of tasks is to be fulfilled with the help of the ontological models, workflows, classifications and integration patterns need to be defined. These approaches will have to define applicability criteria for ontologies, to make sure that exisiting technologies can be reused in accordance to their modeling focus.

On the foundation of an ontology environment as described above, a proper infrastructure for ontology models in automation can be researched. With an infrastructure for handling and applying ontologies to mapping and integration tasks, the comparatively complex ontology models become useable for tasks such as mapping user roles to device functions, e.g., or cost figures to estimated engineering efforts. Prerequisite for these applications is the existence of an infrastructure that supports the generation, classification, comparison, mapping and

application of ontology models. Whether this infrastructure will be specific for the automation domain or can be taken from other areas of applications is to be evaluated.

Another research topic in the context of infrastructural considerations is the portation of the approach to decentralised infrastructures such as service-based environments. Offering web services is a useful way for making an approach more flexible and to allow decentralised applications that can run on different computational entities. This could help facilitate acceptance of ontology models for automation integration tasks, though research on technical issues such as protection of intellectual property rights will be necessary.

Complexity in automation has been an issue for scientists and engineers for decades. If the current striving for a fourth industrial revolution is to be successful, new ideas to reduce complexity and heterogeneity will be necessary. The acceptance by all parties involved is a critical success factor. The author hopes that this thesis is a stepping stone towards a solution of the integration issue existing in automation today.

Acknowledgements

First and foremost I thank Prof. Martin Wollschlaeger of Technische Universität Dresden for his support and feedback in the process of writing this thesis, as well as for six years of fruitful academic work under his guidance. I also thank Prof. Klaus Kabitzsch of Technische Universität Dresden for his advice and especially for the inspiration he provided for chapter 7. Likewise I thank Dr. Dimitris Kiritsis for his feedback, in particular his suggestions for improving the evaluation chapter. Prof. Leon Urbas of Technische Universität Dresden I thank for support in times of need and continuous academic feedback on my work.

Dr. Jan Bredau of Festo AG & Co. KG I thank for support, encouragement and fruitful collaboration, especially on the topic of Condition Monitoring. Christoph-Albrecht Winter of Schmalz GmbH, formerly Festo AG & Co. KG deserves my thankfulness for the same. I would also like to thank Peter-Michael Synek of the VDMA association and Dr. Heinz Bedenbender of the VDI for providing a great platform for professional exchange in their respective working groups. Dr. Jochen Rode of SAP Research I mention as representative for the people at the SAP Research CEC in Dresden I had the pleasure of working together with for six years.

Special mention goes to the other aspiring academics I had the opportunity of exchanging ideas with over the years. Henning Mersch, Dr. Mathias Mühlhause, Dr. Roberto Gutierrez Gonzalez, Dr. Arne Rost, Stefan Hodek, the team of Martin Wollschlaeger, the team of Leon Urbas, the team of Klaus Kabitzsch, with special mention going to Henrik Dibowski, to name but a few. I also thank the consortial partners in the collaborative research projects I worked in during my time as a researcher at Technische Universität Dresden. Alas, the list would grow too long if I mentioned all of you.

Parts of the research leading up to the writing of this thesis has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°260018.

Lastly, and most importantly, I thank my parents, Wilhelm and Mechtild, whose relentless and untiring support of my academic career in every way possible has enabled me to complete this thesis. Thank you very much.

BIBLIOGRAPHY

[1] Richard Alznauer. *Semantisches Informationsmodell für ein ganzheitliches, rechnergestütztes Engineering am Beispiel der Prozeßleittechnik.* Mainz, 1998.

[2] Juergen Anke, Bernhard Wolf, Mario Neugebauer, Katrin Eisenreich, Hong-Hai Do, and Gregor Hackenbroich. A middleware for real-world aware PLM applications. *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference*, pages 1 –12, 26 2007-march 2 2007.

[3] Tim Berners-Lee. Talk on the future of the semantic web. Technical report, World Wide Web Consortium, 2006.

[4] Rolf Birkhofer, Martin Wollschlaeger, Reinhard Schrieber, Markus Winzenick, Johannes Kalhoff, Claus Kleedörfer, Mathias Mühlhause, Jörg Niemann, and Zentralverband Elektrotechnik und Elektronikindustrie Arbeitskreis Systemaspekte. *Life-Cycle-Management für Produkte und Systeme der Automation: Ein Leitfaden des Arbeitskreises Systemaspekte im ZVEI Fachverband Automation.* Zentralverb. Elektrotechnik- und Elektronikindustrie, Fachverb. Automation, 2010.

[5] Jürgen Bock, Peter Haase, Qiu Ji, and Raphael Volz. Benchmarking OWL reasoners. In *ARea2008 - Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, june 2008.

[6] Siegfried Bosch. *Lineare Algebra.* Springer-Lehrbuch. Springer, 2009.

[7] International Electrotechnical Commission. IEC 61131 - programmable controllers, 2003.

[8] International Electrotechnical Commission. IEC 61158 - digital data communications for measurement and control - fieldbus for use in industrial control systems, 2003.

[9] International Electrotechnical Commission. IEC 62264 - enterprise - control system integration, 2003.

[10] International Electrotechnical Commission. IEC 62390 TR - common automation device - profile guideline, 2005.

[11] International Electrotechnical Commission. IEC 61499 - function blocks, 2012.

[12] Aletheia Consortium. Aletheia project website. website, 2013.

[13] PABADIS'PROMISE consortium. PABADIS'PROMISE project website. website, 2013.

[14] PLANTCockpit consortium. PLANTCockpit project website. website, 2013.

[15] SemProM consortium. Semantic product memory - SemProM project website. website, 2013.

[16] World Wide Web Consortium. OWL 2 web ontology language document overview. website, 2013.

[17] World Wide Web Consortium. OWL web ontology language guide. website, 2013.

[18] World Wide Web Consortium. Resource description framework website. website, 2013.

[19] FDI Cooperation. FDI cooperation website. website, 2013.

[20] Wenbin Dai, Victor Dubinin, and Valeriy Vyatkin. IEC 61499 ontology model for semantic analysis and code generation. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 597 –602, july 2011.

[21] Lorenz Däubler. *Strukturverträgliche Ontologien der Automatisierungstechnik*. PhD thesis, Technische Universität Braunschweig, Institut für Verkehrssicherheit und Automatisierungstechnik, 2006.

[22] S. Misbah Deen and K. Ponnamperuma. Dynamic ontology integration in a multi-agent environment. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 1, page 6 pp., april 2006.

[23] Frederic Demoly, Jessica McCarthy, Connor Upton, and Dimitris Kiritsis. An integrated requirements elicitation approach for the development of data management systems. In *Proceedings of the International Conference on Product Lifecycle Management, PLM2011*, july 2011.

[24] Alexander Dennert, Andreas Gössling, Jakob Krause, Martin Wollschlaeger, and Ana Maria Henao Montoya. Vertical data integration in automation based on IEC 61499. In *Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on*, pages 99 –102, may 2012.

[25] Henrik Dibowski and Klaus Kabitzsch. Ontology-based device descriptions and device repository for building automation devices. *EURASIP Journal on Embedded Systems*, 2011.

[26] Falk Doherr, Thomas Schmidt, and Leon Urbas. Fieldbus material take-off estimation: Towards an automated cost estimation of fieldbus installations. In *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on*, pages 165 –168, may 2010.

[27] Rainer Drath (ed.). *Datenaustausch In Der Anlagenplanung Mit AutomationML: Integration Von CAEX, PLCopen XML Und COLLADA*. VDI-Buch. Springer, 2010.

[28] Marc Ehrig. *Ontology Alignment: Bridging the Semantic Gap*, volume 4 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2007.

[29] Ulrich Epple, Markus Remmel, and Oliver Drumm. PandIX modellbeschreibung (german). 2010.

[30] International Organization for Standardization. ISO 15745 - industrial automation systems and integration - open systems application integration framework, 2003.

[31] International Organization for Standardization. ISO 15926 - industrial automation systems and integration - mulit-part standard, 2004.

[32] OPC Foundation. OPC foundation's ISA-95 OPC UA working group website. website, 2013.

[33] Timo Frank, Karin Eckert, Thomas Hadlich, Alexander Fay, Christian Diedrich, and Birgit Vogel-Heuser. Workflow and decision support for the design of distributed automation systems. In *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, pages 293 –299, july 2012.

[34] Manos Georgoudakis, Athanasios Kalogeras, Christos Alexakos, Konstantinos Charatsis, and Stavros Koubias. A holonic ontology-based multi-agent system for the distributed scheduling and monitoring of industrial processes. In *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, volume 1, pages 6 pp. –920, sept. 2005.

[35] Konrad Gnauck. Entwicklung eines übergreifenden modells zur beschreibung intelligenter feldgeräte. Master's thesis, Technische Universität Dresden, Institut für Angewandte Informatik, 2009.

[36] Andreas Gössling, Jan Bredau, Martin Wollschlaeger, and Roberto Gutierrez Gonzalez. A unified access to condition monitoring and maintenance information in industrial machines and systems. In *24th International Congress on Condition Monitoring and Diagnostics Engineering Management (COMADEM 2011)*, pages 387–394, may 2011.

[37] Andreas Gössling and Martin Wollschlaeger. Integrating information over the life-cycle of manufacturing equipment by assigning semantics. In *Factory Communication Systems, 2008. WFCS 2008. IEEE International Workshop on*, pages 357 –364, may 2008.

[38] Andreas Gössling and Martin Wollschlaeger. On working with the concept of integration ontologies. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 709 –712, sept. 2008.

[39] Cristin Groba, Sebastian Cech, Frank Rosenthal, and Andreas Gössling. Architecture of a predictive maintenance framework. In *Computer Information Systems and Industrial Management Applications, 2007. CISIM '07. 6th International Conference on*, pages 59 –64, june 2007.

[40] IDA GROUP. FDCML website. website, 2013.

[41] MIMOSA group. MIMOSA website. website, 2013.

[42] Christian Hahn, Andreas Gössling, Roman Frenzel, and Martin Wollschlaeger. Towards an automated generation of device firmware components for intelligent field devices. In *IEEE International Conference on Industrial Technology ICIT 2009*, february 2009.

[43] Michael Heeg. *Ein Beitrag zur Modellierung von Merkmalen im Umfeld der Prozessleittechnik*. Fortschrittberichte der VDI-Zeitschriften: Reihe 8, Mess-, Steuerungs- und Regelungstechnik. VDI-Verl., 2005.

[44] CAN in Automation (CiA) e.V. CANopen XML specification, 2006.

[45] PROFIBUS & PROFINET International. PI website. website.

[46] PROFIBUS & PROFINET International. *Specification for PROFIBUS Device Description and Device Integration: GSD V 5.1*. Number Vol. 1. PROFIBUS & PROFINET International, july 2008.

[47] PROFIBUS & PROFINET International. *PROFIBUS - Profile for Process Control Devices V 3.02*. Number Vol. 1. PROFIBUS & PROFINET International, april 2009.

[48] PROFIBUS & PROFINET International. *GSDML - Technical Specification for PROFINET IO V 2.25*. Number Vol. 1. PROFIBUS & PROFINET International, april 2010.

[49] PROFIBUS & PROFINET International. PROFIBUS homepage. website, 2013.

[50] PROFIBUS & PROFINET International. PROFINET - the leading industrial ethernet standard homepage. website, 2013.

[51] Frank Iwanitz and Jürgen Lange. *OPC Fundamentals, Implementation and Application*. Laxmi Publications Pvt Limited, 2010.

[52] Dimitris Kiritsis. Closed-loop PLM for intelligent products in the era of the internet of things. *Computer-Aided Design*, pages 479–501, 2011.

[53] Konrad Königsberger. *Analysis 1*. Springer-Lehrbuch. Springer, 2001.

[54] Jakob Krause, Sebastian Cech, Frank Rosenthal, Andreas Gössling, Cristin Groba, and Volodymyr Vasyutynskyy. Factory-wide predictive maintenance in heterogeneous environments. In *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on*, pages 153 –156, may 2010.

[55] Jose L. Martinez Lastra and Omar J. Lopez Orozco. Semantic extension for automation objects. In *Industrial Informatics, 2006 IEEE International Conference on*, pages 892 –897, aug. 2006.

[56] Andrei Lobov, Jose L. Martinez Lastra, and Reijo Tuokko. On controller and plant modeling for model-based formal verification. In *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, volume 1, pages 8 pp. –128, sept. 2005.

[57] Wolfgang Mahnke, Andreas Gössling, Markus Graube, and Leon Urbas. Information modeling for middleware in automation. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1 –7, sept. 2011.

[58] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC Unified Architecture*. Springer, 2009.

[59] Aristeidis Matsokis. *An Ontology-Based Approach for Closed-Loop Product Lifecycle Management*. EPFL, 2010.

[60] Mathias Mühlhause. *Konzept zur durchgängigen Nutzung von Engineeringmodellen der Automation*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, Fakultät für Elektrotechnik und Informationstechnik, 2012.

[61] Mathias Mühlhause, Nico Suchold, and Christian Diedrich. Application of semantic technologies in engineering processes for manufacturing systems. In *10th IFAC Workshop on Intelligent Manufacturing Systems*, july 2010.

[62] Andreas Müller, Ingmar Hofmann, Ralf Klöß inger, Michael Pirker, and Mikhail Roshchin. Enhancing plant lifecycle management systems with diagnostic functionality based on automated reasoning. In *Automation Congress 2011, Baden-Baden*, 2011.

[63] A. Cemal Oezluek, Henrik Dibowski, and Klaus Kabitzsch. Automated design of room automation systems by using an evolutionary optimization method. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1 –8, sept. 2009.

[64] Martin Polke. *Prozeßleittechnik*. Oldenbourg, 1994.

[65] Matthias Riedl, René Simon, and Mario Thron. *EDDL - Electronic Device Description Language*. Oldenbourg, 2002.

[66] Stefan Runde, Henrik Dibowski, Alexander Fay, and Klaus Kabitzsch. Integrated automated design approach for building automation systems. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 1488 –1495, sept. 2008.

[67] Thilo Sauter and Martin Wollschlaeger. Feldbussysteme - historie, eigenschaften und entwicklungstrends (fieldbus systems - history, properties, and trends). *it - Information Technology*, april 2000.

[68] Daniel Schuster, Till M. Juchheim, and Alexander Schill. Finding and classifying product relationships using information from the public web. In Joaquim Filipe and Jose; Cordeiro, editors, *ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems, Volume 1, DISI, Funchal, Madeira, Portugal, June 8 - 12, 2010*, pages 300–309. SciTePress, 2010.

[69] Andreas Selig. *Informationsmodell zur funktionalen Typisierung von Automatisierungsgeräten*. PhD thesis, Universität Stuttgart, 2011.

[70] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973.

[71] Markus Stoess, Falk Doherr, and Leon Urbas. Automated network layout for the industrial communication engineering system NetGen:X. In *Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on*, pages 281 –290, may 2012.

[72] Martin Strube, Stefan Runde, Helmut Figalist, and Alexander Fay. Risk minimization in modernization projects of plant automation - a knowledge-based approach by means of semantic web technologies. In *In proceedings of IEEE ETFA 2011*, 2011.

[73] Stefan Theurich, Christian Hahn, Roman Frenzel, and Martin Wollschlaeger. Field bus abstraction as a means to enable network-independent applications. In *8th IFAC International Conference on Fieldbuses and Networks in Industrial and Embedded Systems FeT 2009*, pages 131–138, Ansan, Republic of Korea, may 2009.

[74] Angela Tursi, Herve Panetto, Gerard Morel, and Michele Dassisti. Ontological approach for products-centric information system interoperability in networked manufacturing enterprises. *Annual Reviews in Control*, 33(2):238 – 245, 2009.

[75] M. Kamal Uddin, Aleksandra Dvoryanchikova, Andrei Lobov, and Jose L. Martinez Lastra. An ontology-based semantic foundation for flexible manufacturing systems. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pages 340 –345, nov. 2011.

[76] Verband Deutscher Maschinen und Anlagenbauer (German Association of Mechanical Engineering Companies). Feldbusneutrale Referenzarchitektur für Condition Monitoring in der Fabrikautomation (field-bus-neutral reference architecture for condition monitoring in manufacturing automation). VDMA Einheitsblatt 24582 (draft for review), 2013.

[77] Jouko Virta, Ilkka Seilonen, Antti Tuomi, and Kari Koskinen. SOA-based integration for batch process management with OPC UA and ISA-88/95. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1 –8, sept. 2010.

[78] Birgit Vogel-Heuser, Jens Folmer, Georg Frey, Liu Liu, Holger Hermanns, and Arnd Hartmanns. Modeling of networked automation systems for simulation and model checking of time behavior. In *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pages 1 –5, march 2012.

[79] World Wide Web Consortium (W3C). XML core web page.

[80] Thomas Wächter and Michael Schroeder. Semi-automated ontology generation within OBO-Edit. *Bioinformatics*, 26(12):i88–i96, 2010.

[81] Martin Wollschlaeger. Semantische vernetzung im lebenszyklus der automatisierung. In *Karlsruher leittechnisches Kolloquium 2010*, pages 115–125, june 2010.

[82] Dafeng Xu, Qing Li, Hong-Bae Jun, Yuliu Chen, Jim Browne, and Dimitris Kiritsis. Modeling for closed-loop product information tracking and feedback using wireless technology. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 2800 –2805, oct. 2007.

A GLOSSARY

A.1 ABBREVIATIONS

CAN	Controller Area Network
CM	Condition Monitoring
DD	Device Description
DDL	Device Description Language
DF	Data Format
EAM	Enterprise Asset Management
EDD	Electronic Device Description
EDDL	Electronic Device Description Language
EDS	Electronic Data Sheet
ERP	Enterprise Resource Planning
GSD	Generic Station Description
GSDML	GSD Markup Language
IT	Information Technology
MES	Manufacturing Execution System
OWL	Web Ontology Language
PLC	Programmable Logic Controller
PLM	Product Life Cycle Management
PM	Predictive Maintenance
RDF	Resource Description Framework
SCADA	Supervisory Control and Data Acquisition
TCO	Total Cost of Ownership
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

Table A.1: Abbreviations

A.2 VOCABULARY

Actor	A person or a system that performs actions influencing a target system
Automation	A part of the sciences that deals with technical processes that are supposed to reach a goal without direct human interaction
Data Format	A syntactically stringent system for representing information
Field Bus	A computational network that is specifically designed to support automation functions
Field Device	A technical entity in an automation system that can be installed or exchanged as a whole
Information Modeling	A field of research defining techniques or approaches for mapping of abstract information onto elements of a formalised model
Instance	A distinguishable entity that typically can be classified as belonging to a type
Mapping	The act of assigning formalised relationships between entities in a model or in the physical world
Ontology	A computer-scientific model describing and formalising concepts and their relationships
Profile	A technical document that, in a formalised manner, describes qualities or capabilities of a technical entity
Semantics	The meaning of an entity or concept as interpreted by humans or computers
System	A technical composite that has defined boundaries, possibly inputs and/or outputs, as well as an intended purpose
System Designer	An actor tasked with defining the intended function of a system as well as the means to fulfill said function
System Engineer	An actor tasked with defining the implementation of a system's function specific to a use case
Type	A class of entities that can be characterised by the attributes that distinguish the type from other types

Table A.2: Vocabulary of the Thesis

LIST OF FIGURES