Masterarbeit

PISCAS - A Pisciculture Automation System Product Line

Christopher Preschern

Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Begutachter und Betreuer: Dipl.-Ing. Dr. techn. Christian Kreiner

Graz, im Mai 2011

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit auf Software Product Line gestützen Methoden zur Entwicklung von Automatisierungssoftware in der Domäne Fischzucht. Um den Prozess der Softwareentwicklung effizienter zu gestalten und die Softwarequalität zu steigern wurde ein Codegeneratorsystem erarbeitet. Mit Hilfe dieses Systems ist es möglich, mit geringem Aufwand qualitativ hochwertige Automatisierungsanlagen zu erstellen. Weitere Arbeitsschritte für die Inbetriebnahme der Anlage werden optimiert. Durch eine automatisch generierte Dokumentation, welche einen Elektoinstallationsplan der Anlage beinhaltet, ist die Installation der Fischzucht Hardware für eine Drittfirma (Elektroinstallateur) einfacher durchführbar. Die generierte Software beinhaltet einen ausführlichen Test-Modus. Durch diesen werden den Elektroinstallateur und Vertreiber der Automatisierungssoftware bei der Inbetriebnahme und im Langzeitbetrieb unterstützt. Durch die beschriebene effizientere Erstellung eines Fischzuchtautomatisierungsprojekts können sehr viel Arbeitszeit und Kosten eingespart werden.

In der Arbeit wird die umgesetzte Software Product Line für die Domäne Fischzucht beschrieben. Ein spezieller Fokus liegt auf Requirements Engineering während der Domain Engineering und der Application Engineering Phase. Es wird speziell auf Codegenerator Patterns eingegangen. Das Metamodell und die Codegeneratoren für die Fischzucht Product Line werden beschrieben und der Prozess des Erstellens eines neuen Fischzucht-Automatisierungssystems wird erklärt. Es wird ein Vergleich von verschiedenen Softwareentwicklungsmethoden anhand von repräsentativen Daten, die während der Installation von zwei Fischzuchtautomatisierungssystemen gesammelt wurden, gezeigt.

Abstract

This thesis describes different methods for the development of domain specific automation systems for fish farms. To enhance the software development process a software product line was adopted. Fish farm automation systems can be produced more efficiently by using a code generating approach. Systems are modeled with a high level domain specific language, that allows to produce high-quality automation systems in less time. The installation of fish farm automation systems is significantly simplified. Generated documentation, including an electrical installation plan, provides a good basis for the hardware installation of the system. Costs for an electrician who installs the hardware are therefore reduced due to less work time. The generated software includes an extensive test mode which allows for easier detection of errors occurring during the installation and operation. With the proposed software product line costs in the fish farm automation caused by system installation and maintenance work can be saved.

The thesis describes the developed software product line. Focus is put on requirements engineering, as it can be used during the domain engineering and application engineering phase. Code generator patterns which can be used for software product line engineering are covered in detail. The developed metamodel and the code generators for the fish farm product line, as well as the process of generating a fish farm automation system are described. Two fish farm systems were developed with the proposed product line approach. Different development methods were used, evaluated and compared.

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....
date (signature)

Credits

This master thesis was carried out at the Institute for Technical Informatics, Graz University of Technology.

At this point I want to thank my friends and my family for enduring me during the time of my study. I am sure it was not easy all the time - special thanks for that.

Christian Kreiner helped me with my master thesis with many good advices. Without him it would have been a lot more difficult to achieve the aims of this thesis. I also want to thank Michael Hofer for the possibility to carry out this thesis for his firm. He helped me with several organizational issues. For providing me the necessary hardware for my project I would like to thank the company Bernecker&Rainer and the Graz University of Technology. Special thanks for a lot of advice and input about real-life projects to EVA GmbH and Elektro Tisch.

Graz, May 2011 Christopher Preschern

Contents

1 Introduction **10**
- 1.1 Motivation and goal . . . 10
- 1.2 Outline . . . 11

2 Related work **12**
- 2.1 Comparison of the different automation systems . . . 12
  - 2.1.1 Types of automation systems . . . 12
  - 2.1.2 Observed commonalities for automation systems projects . . . 13
- 2.2 Requirements Engineering . . . 14
  - 2.2.1 Requirements engineering process . . . 14
- 2.3 Software Product Line Engineering . . . 16
  - 2.3.1 SPL management . . . 16
  - 2.3.2 Domain engineering . . . 16
  - 2.3.3 Application engineering . . . 21
  - 2.3.4 Program generation patterns . . . 21
  - 2.3.5 Software Product Line development methods . . . 25
- 2.4 Model driven development . . . 28
  - 2.4.1 MDD key attributes . . . 28
  - 2.4.2 Domain specific modeling . . . 30
  - 2.4.3 DSM patterns . . . 32
  - 2.4.4 Tools . . . 33
- 2.5 The PISCAS approach compared . . . 34

3 Design and implementation of PISCAS **36**
- 3.1 Domain engineering . . . 36
  - 3.1.1 4+1 viewpoint model . . . 36
  - 3.1.2 Business model . . . 40
  - 3.1.3 Product scope . . . 42
  - 3.1.4 Requirements . . . 43
  - 3.1.5 Modeling language . . . 49
  - 3.1.6 Metamodel concept . . . 49
  - 3.1.7 Variability . . . 51
  - 3.1.8 Metamodel . . . 52
  - 3.1.9 Generators . . . 52
  - 3.1.10 Use cases . . . 53

3.1.11 PLC software template 56
3.2 Application engineering 63
3.2.1 PISCAS system instantiation 63
3.2.2 Use cases 64

4 Project evaluation **65**
4.1 Project implementation 65
4.2 Software quality 67
4.3 Time to market 68
4.4 Development approach evaluation 68
4.4.1 The cost model 69
4.4.2 Cost analysis 71
4.4.3 Break even point estimation 71
4.5 Achieved goals 73

5 Conclusion and future work **74**

A Evaluation of metamodeling tools **76**

B Questionnaires **80**
B.1 Questionnaire for fish farm owners 80
B.2 Questionnaire for electrician companies 81

Biblography **83**

List of Figures

2.1 SPL work process overview[PBL05] . . . 17
2.2 FORM feature diagram [KCH+90] . . . 18
2.3 What to build pattern for SPLE [CN01] . . . 20
2.4 Classification of code generator transformation processes [vL04] . . . 21
2.5 Templates and filtering program generation pattern [Voe03] . . . 22
2.6 Template and metamodel pattern [Voe03] . . . 23
2.7 Frame processing program generation pattern [Voe03] . . . 23
2.8 API-based generation pattern [Voe03] . . . 24
2.9 Inline code generation pattern [Voe03] . . . 25
2.10 Code attributes pattern [Voe03] . . . 25
2.11 Code weaving pattern [Voe03] . . . 26
2.12 Hierarchical plant structure model [SSV08] . . . 27
2.13 DSM costs vs. general-purpose modeling costs [Siv08] . . . 30
2.14 DSM concept enabling easier product development [Met09] . . . 31

3.1 4+1 viewpoint model [Kru95] . . . 37
3.2 PISCAS main stakeholders . . . 38
3.3 4+1 viewpoint model for PISCAS . . . 40
3.4 PISCAS cost model structure . . . 42
3.5 PISCAS Product scope . . . 43
3.6 PISCAS Meta-Model . . . 52
3.7 Mapping of the PISCAS model to the generated artifacts . . . 53
3.8 Packages of the PISCAS software . . . 57
3.9 State diagram of the oxygen supervision . . . 58
3.10 Calculation of the necessary fodder amount . . . 58
3.11 Pseudocode for the scheduling . . . 60
3.12 State diagram for the power supply . . . 61
3.13 Config File for the feeder information . . . 61
3.14 Data structure for the feeding information . . . 62
3.15 State diagram for switches . . . 62
3.16 Instantiation of a new PISCAS system . . . 63

4.1 Model of the fish farm in Radenthein . . . 66
4.2 Fish farm in Feld am See . . . 67
4.3 Model of the fish farm system Feld am See . . . 67
4.4 Overview part of the documentation of the fish farm system Feld am See . . . 68

4.5 Hardware part of the documentation of the fish farm system Feld am See . 69
4.6 Visualization of the PLC software for the fish farm system Feld am See . . 69
4.7 Work hours for different development approaches 72

List of Tables

2.1 Results of the tool evaluation 35

3.1 Full list of PISCAS project stakeholders 39

4.1 Comparison of fish farm systems Radenthein and Feld am See 70
4.2 Summary of work hours for different development approaches for PISCAS systems 71
4.3 Overview of the work hours for different development methods 72
4.4 Goal analysis 73

A.1 Criteria to rate SPL tools 79

Chapter 1

Introduction

Domain specific modeling of programmable logic controllers (PLC) is a recent field of research. This thesis compares different methods to realize automatic project generation for systems in the fish farm domain.

1.1 Motivation and goal

Until now the available automation systems in the domain of fish farms do not satisfy the customer needs. Advanced approaches like models of growth of the fish are not considered by PLC systems. This increases the service work for the customer, because the fodder amount has to be adjusted very often. Systems including oxygen supervision and feeding just exist for open sea piscicultures [PBL05]. Compareable systems for smaller fish farms are not available and can just be offered custom made. Those systems are therefore very expensive.

The aim of this master thesis is to provide a system, which generates the entire automation system project for a graphically modeled pisciculture. The automatic code generation process leads to higher quality software. This production method is more economical compared to custom made projects as it is described in [Fro03]. The project does not just include the PLC software. Detailed documentation is also provided. A main goal of the thesis is to develop a system which minimizes the work for the implementation and installation. This includes precise documentation of the system developed, especially of the electrical installation plan. This installation might be done by another company, so it is crucial that the electrical documentation is well-defined and that the system is easy to handle for the other company. A key factor for efficient production of PLC systems is producing highly reliable and correct code in an effective way. PISCAS goes one step further. Not just the code generation is optimized. The whole inter-organizational information flow is also improved by automatically generating all the necessary information for the installation of the system to the involved companies.

The goal of PISCAS is to present a framework which can generate a whole pisciculture project with low effort. This enables the vendor of the fish farm system to offer a high quality PLC system for low charge. Maintenance costs are minimized by providing an easy approach to change a delivered system. The possibility to maintain the PLC remotely also reduces maintenance costs.

1.2 Outline

In the next section, related work is presented. First a comparison of different kinds of automation systems is shown. Section 2.2 guides through the process of requirements engineering and shows several methods for the different stages of the process. The main research part of this master thesis is covered in Section 2.3: software product line engineering. The basic approach with special focus on domain engineering and program generation patterns is covered. Section 2.4 presents model driven development including model driven architecture and domain specific modeling. In Section 2.5 the PISCAS approach is compared. Section 3 handles the PISCAS product line design and implementation. The section is divided into domain engineering and application engineering processes. Detailed information about the process of establishing a product line for the fish farm automation domain is given. Section 4 covers an evaluation of the different methods that were used to develop the pisciculture automation systems and gives information about two real fish farm systems which were automated. Also, a goal analysis is presented. Section 5 concludes this thesis and presents topics for future work.

In Appendix A information about an evaluation of different metamodeling tools is shown. Appendix B contains questionnaires for fish farm owners and electric installation companies. These questionnaires were conducted during the gathering of project requirements.

Chapter 2

Related work

The related work section of this thesis presents several well studied topics covered by literature which relate to PISCAS. Their advantages and disadvantages for the proposed SPL are discussed. The main topics covered are basic considerations about the type of automation system, software product line engineering and model driven development. Also special focus is put on requirements engineering as it is needed during the domain engineering and application engineering process of the software product line approach.

2.1 Comparison of the different automation systems

In this section the question which kind of automation system is used for PISCAS is discussed. In the automation sector there are several ways to assemble an automation system. The most common methods are using a programmable logic controller, using a microcontroller, or using an industrial computer. The border between these solutions is continuous. This means that the following given characteristics for the systems are typical, but not necessary.

2.1.1 Types of automation systems

Industrial Computers

Industrial computers, also called remote terminal units (RTU), are often used to control large systems. Such Supervisory Control and Data Acquisition (SCADA) systems often require many complex calculation. Therefore a RTU is better suited than other systems, because it provides the most processing power. RTU systems also provide more programming flexibility and communication support. RTUs are best suited for large SCADA systems which need a lot of processing power and a high bandwidth for communication [Mot07].

Programmable Logic Controllers (PLC)

[Wat01] states that PLCs are cheaper than RTU solutions to a given application which does not exceed a complexity which can easily be handled by the PLC. PLCs offer a cheap standardized solution for automation domain problems.

Microcontrollers

The advantage of customized microcontroller systems is that the hardware of the system is very cheap. Compared to PLC hardware, the microcontroller costs are just a fraction of the PLC price.

The development of a customized microcontroller system requires a lot of work. Compared to PLCs these systems are not standardized and adjusted to the automation domain. Microcontroller systems development is rather expensive. These systems can just be applied to systems with a high quantity.

2.1.2 Observed commonalities for automation systems projects

The following observations were made by interviewing two companies in the automation system sector (Appendix B.1) and during the development of the first version of PISCAS.

In the automation sector a common situation can often be observed:

- The automation system development and its installation are often carried out by two different companies or at least by two different departments of a company. This quite often leads to problems due to lack of communication or non-standardized communication between those two stakeholders. Increased costs for the installation of the system are a consequence.
- A company developing automation systems often specializes on a specific kind of automation system (e.g. automation of gravel pits, automation of fish farms).
- Different products of the same automation system family are often developed by the same employees. They have a lot of implicit knowledge about the domain and about the automation system family.
- In many cases an automation system is very domain specific. The domain knowledge has to be gained from the customer. Getting the necessary knowledge is a difficult and time consuming task.
- It is very common that automation systems contain software errors and that during the first few weeks of operation the system has to be updated quite often.
- The software often has to be changed due to new components which are installed.
- To create a software which is quite similar to an existing project usually the clone&own method is used. This means that the old software is copied and adapted. This leads to many errors because it happens quite often one forgets to correct some parameters or to update some settings for the new project.

Many problems arise from these observations. Without a systematic approach a lot of work hours are wasted due to lack of communication. Especially during the installation of the system many errors could be avoided. As usually products for an automation system family are developed just by few employees, just those people have the necessary knowledge to maintain those domain specific automation systems or to develop new systems in the domain in a reasonable time. This leads to the fact that the possibility of loosing such an employee is a major threat for a company.

2.2 Requirements Engineering

For software product lines it can be devastating if later changes of the system or the domain were not considered as it does not just affect one product, but a whole series of products. Requirements engineering aims to reduce this risk. A key aspect is to determine variability features of the software product line very soon. According to [KE02] and [Mac96] requirements engineering can be divided into five stages:

- Requirements capturing
- Requirements analysis
- Requirements specification
- Requirements verification
- Requirements management

2.2.1 Requirements engineering process

Requirements capturing

Requirements capturing deals with the process to obtain as many requirements information as possible. According to [CY08] there are two phases of gathering requirements. In the early requirements phase the intentions of the customer and the purpose of the system are discussed. In the late requirements phase alternative systems are analyzed. Functional and non-functional requirements are fixed.

Interviews with the customer can be conducted. These interviews can be structured or unstructured. A unstructured discussion between a requirements engineer and a customer can lead to new views of the product and to innovative ideas. Structured interviews often deliver more precise results.

Questionnaires can be handed out to the stakeholders. This provides the opportunity to get the opinion of many different people. As they might have a different view of the product, many for software engineers not obvious requirements can be discovered.

The soft system methodology includes organizational and domain factors to the requirements capturing process. A textual description of the challenges for the product is developed. This description combined with the domain knowledge yield the domain requirements [KE02].

Case studies of competitor products and market analysis gives an impression of the current state of the art. It shows which features existing products include and gives an overview of the needed functionalities for a software product in the given domain.

Designer as apprentice. The software designer can work as an apprentice in the target domain. This provides a lot of experience and knowledge about the problem domain. The disadvantage of this method is that it is very time consuming. It should only be conducted for huge product line projects where design errors are crucial.

Requirements analysis

The in the previous stage gathered information is structured and analyzed. The common attributes and the variability of the products of a software product line can be elaborated in this stage.

Structured system analysis is an function-oriented approach to work out the variabilities of a product based on the given requirements. Structured System Analysis does not support component based design.

Object oriented analysis. These techniques use the object-oriented programming concept for modeling requirements. Compared to the structural models they provide better encapsulation and abstraction of the requirements. Variability can be modeled by inheritance. According to [Mac96] these approaches are very good in supporting reuse. Object-oriented requirements analysis approaches are used in the KobrA (Section 2.3.5) and the Sherlock (Section 2.3.5) model.

Participatory design is a concept where designers and customers work closely together to analyze the requirements. This method ensures that the customer's needs are really all covered.

Quality function deployment translates requirements described by customers into appropriate technical requirements. The method also handles customer priorities.

Requirements specification

The elaborated requirements are specified and documented. This specification is worked out for the product family and for every single product. Common and variable requirements for the different systems have to be documented.

Requirements verification

The verification of the requirements for software product lines does not differ to the methods used during single system development.

Requirements reviews. As the requirements are not static and might change during the development of the software product line, they have to be maintained. The reviews ensure that the requirements are up-to-date and that they really satisfy the customers needs.

Prototyping can be used to show that the prototype of a software product fulfills the requirements.

Requirements testing. All requirements are tested against defined test cases. Each product is tested against each requirement. [PL05] describes requirements verification in single software systems. This can as well be applied to software product lines.

Requirements management

Requirements management ensures that all the requirements are up-to-date and documented. It also handles requirements changes.

Non-functional requirements

To achieve business goals fast non-functional requirements are often not considered.

If a design error occurs and an implicit non-functional requirement cannot be maintained, the cost for fixing this error are often very high as it is discovered late in the development process. Non-functional goals and functional goals often influence each other strong. It is useful to set up a table with functional goals as rows and non-functional goals as columns. In the table the influence between the goals is marked [Ngu09]. With the help of this table the decision which requirements are fulfilled to which degree can be made.

2.3 Software Product Line Engineering

Software product line engineering (SPLE) provides the opportunity to produce customized mass products. Software product lines (SPL) use software engineering methods and tools for creating similar software systems. SPLs apply techniques for reusing core assets for customized products in a domain. SPLs are often used in security critical domains where software development costs for individual products are very high [WHG+09].

There are many reasons for introducing a SPL. SPLE includes up-front investments. If many software systems are sold, then this investment is feasible, because the production cost of a system in a SPL is significantly lower compared to an individual developed system. In a SPL it is easier to maintain high quality [Ham08]. If the tools and the process for generating the software system are sufficiently tested, the produced products can be created with no new errors. If errors occur than they can be corrected in one central component. Individual developed systems would have to be tested more thoroughly as they might contain more errors than a system which is generated by a tested SPL. The time to market is also significantly decreased for SPL systems. For customers SPLs provide cheap software with few errors [PBL05].

Fig. 2.1 gives an overview of the main parts of a SPL. The development process is divided in the domain engineering and the application engineering part.

2.3.1 SPL management

Compared to management for single products, the management of a software product line engineering process implies additional complexity. The management of a SPLE project is positioned closer to a strategic level then to the management of an individual project. The business goal is to produce high quality end-products and to develop a framework which enables the production of the products. The aim of the SPLE management is to develop universal reusable components which can be used to assemble the product in an effective way.

Usually, a project is a temporary venture with a defined input and output. SPLE differs from that. It does not have a defined end point. The SPLE last as long as new products are produced. An additional challenge is that considerations of the project environment (e.g. technologies, competitors) have to be taken over much longer lifetimes [CJNM05].

2.3.2 Domain engineering

The domain engineering process develops core assets for a domain which can be reused. Domain engineering is an iterative process and has to be done during the whole lifetime

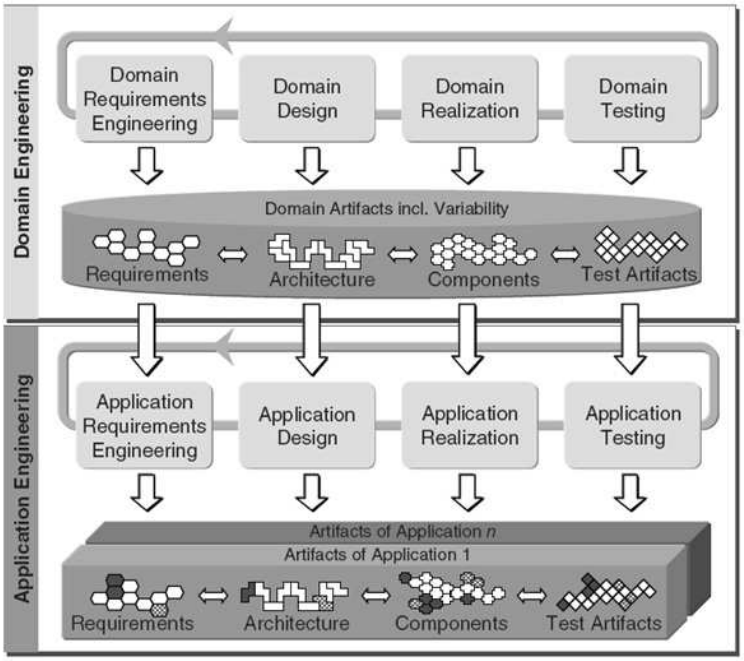


Figure 2.1: SPL work process overview[PBL05]

of the SPL. Its main elements are requirements engineering, development of domain architecture and components and testing of the domain assets. With the information of the documented requirements a domain model is built. This model provides an architecture and other reusable components. To obtain the necessary reusable components, the variability of the domain has to be analyzed. The domain model should support the most important variants. During the domain engineering the granularity of the variants has to be determined and the scope of the product line has to be set. It is very important to elaborate a precise and complete domain model. If some features are not modeled and have to be added at a later development phase of the process then the whole domain has to be re-engineered which is a lot of work [vL04].

Variability management

Variability Management handles differences between the resulting products in a software product line. During the requirements analysis, commonalities and variabilities were identified. It is very important for SPLE to elaborate the variabilities of a system very early. To define the variabilities a lot of communication between the software designer and the stakeholders during the requirements analysis is important. Commonalities describe features of the products which do not differ. Variabilities are the features which differ between products of the SPL [MP07]. Variation points locate a variability and define possible variants. [Tri03] distinguishes between several types of variability.

- Variability in function. A software functionality exists for a product, but not for other products of the product line.

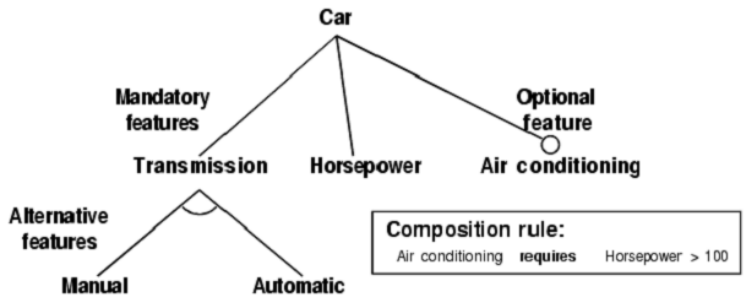


Figure 2.2: FORM feature diagram [KCH+90]

- Variability in data. Some systems use different data structures in parts of the software
- Variability in control flow. Parts of the software behave different for different products.
- Variability in product environment. Systems are used in different environments and have to act according to the environment.

Modeling variability

Feature Diagrams

'A feature is a sum of requirements that defines a products characteristics and qualities' [Tri03].

Feature diagram present the possible features for a product in a tree. Variations are modeled in this hierarchy. Several different notations for feature diagrams are shown in [Tri03] and [KCH+90]. An example of a feature diagram is used by the FODA product line architecture design method (Section 2.3.5). The following slightly altered text from [Tri03] describes the syntax of this feature diagram

Feature diagrams are trees capturing the relationships among features.

- The root of the tree represents the concept being described and the remaining nodes denote features and their sub-features.
- A feature is *mandatory* unless an empty circle is attached above its name, indicating an optional feature.
- *Alternative features* in FODA are considered as specializations of a more general feature. Indeed, alternative features, which are children of the same parent feature and connected by an arc, define the alternative feature set from which almost one feature can be selected.
- More dependencies between features like *require* or *exclude* can be expressed with composition rules to avoid graphical overload.

Fig. 2.2 shows an example of a feature diagram.

Domain specific language (DSL) Variability can easily be modeled with the aid of domain specific languages. Further information regarding DSLs is provided in 2.4.2.

Modeling variability with use cases

Use cases are very effective for single software systems. They are extended in functionality to be usable for SPL. Classical use cases describe what a system should do in a given situation. This descriptions is in a way that also non-technical people can contribute to use cases. To utilize use cases for SPL, they have to be extended. An alternative flow of the use case depending on the specific product is allowed. With these alternative flows the variability is designed. Various methods for extended use case text and diagrams exist [Tri03], [BFG+02].

Other approaches

Thomas von der Maen describes a model where variability is present as a combination of use-cases and feature diagrams [vdML02]. Variability can also be modeled with UML class diagrams [Cla01] .

Granularity

An essential task in software product line engineering is to specify the granularity of the variability. Coarse grained software product lines allow to change modules, while fine grained SPLs facilitate changing several code lines in a module. Fine grained variability allows to develop system with a lot of little differences, where coarse grained systems just allow the products to differ in the modules [KAK08].

[Jun08] discusses two types of code generator systems.

Specialized code generator systems allow easy code generation. With the generator the project can be designed very easily. Domain specific code can be generated and often needs no reworking. A disadvantage of such a system is that it is hard to reuse the generator for another software product line.

Multi-purpose code generators enable the generation of source codes for different systems. Modeling of the domain can be done by a template. Often, variability of the system is modeled for system production. This means that the engineer of a specific product needs a lot of knowledge about the code generator. The advantage of multipurpose systems is that the same generator can be used for several software product lines.

Design and implementation of variability

All the variation points of the system an their constraints are collected and a metamodel of the system is developed. This metaemodel describes the possible systems which can be modeled for the given domain. The information stored in such a system model is used to create an actual application with the aid of code generators.

We can distinguish between two basic concepts for expressing variability in software product lines.

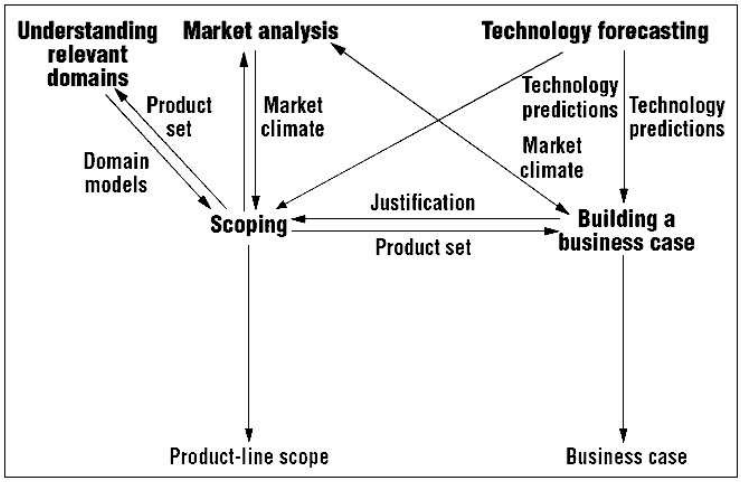


Figure 2.3: What to build pattern for SPLE [CN01]

The first possibility to generate different products is to design modules and compose the product of these modules. This approach can be compared to object-oriented programming. The software is organized in modules (objects) which can be put together. Variability can be achieved by varying the amount of the objects or by applying design patterns like a decorator to the modules. Usually, with this compositional method, just coarse grained variability (Section 2.3.2) is modeled.

The other approach to design variability in SPL is called the annotative approach. With this approach, source code is programmed in a way that decisions about the components are made at deploy or compile time. An example for this are `#ifdef` statements in C/C++ code. This can be compared to structured programming as no objects are involved [KAK08].

Scoping

During the domain engineering the scope of the product line has to be set. It defines the family of products which is covered by the SPL. If the scope is too narrow, then just very few products can be developed with the aid of the SPL and it is unlikely that many products will be developed. If, however, the scope is set too wide, then the management of the SPL gets more difficult and it is probably hard to establish a domain model which suits all variants. The 'What to build' pattern (Fig. 2.3) provides a good start for the scoping process. It explains which components should be included in the domain and gives reasons for the decision [CN01].

Code generators

A code generator is a tool that translates a product model to another model or to corresponding source code. In software product lines, generators are often parameterized with templates. These templates hold information about the domain. The tasks of generators

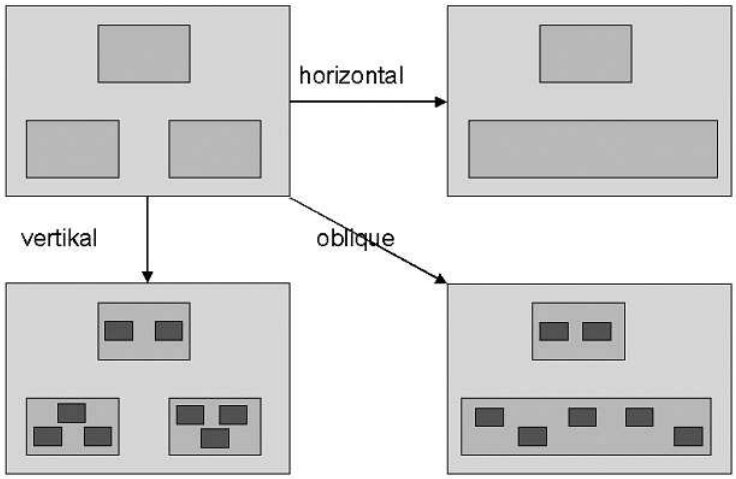


Figure 2.4: Classification of code generator transformation processes [vL04]

are to check and optimize the input and perhaps add information to the input which is missing. With the obtained valid input description the output code is generated [Ben03].

There are several different kinds of transformation processes (Fig. 2.4). A generator can be applied between two models on the same abstraction level (horizontal) or it can transform models or code on different abstraction levels (vertical). Horizontal generators change the structure of the model, while vertical generators change the abstraction level. If those two kinds are combined a so-called oblique model transformation is performed [vL04].

2.3.3 Application engineering

This procedure handles the actual creation of a product with the help of the framework developed during the domain engineering process. The products are matched to the customer requirements, which can be obtained by a system analysis. With the information of the requirements a system can be modeled with the aid of the metamodel for the domain. With the developed code generators the model can then be transformed to a working system. Depending on the software product line and the given domain the produced product might have to be altered. If too many parts of the code have to be changed it might be considered to change the domain model by doing the domain engineering again. This can be the case if the domain was not well understood while the engineering phase [Ben03].

2.3.4 Program generation patterns

As in software design also in software product line engineering patterns can be applied. They provide a solution to standard problems during the development of SPL systems. The information in the following section is mostly based in [Voe03]. The patterns covered in this section are patterns which describe the overall type of the code generation process. More detailed patterns can be found in [VB04].

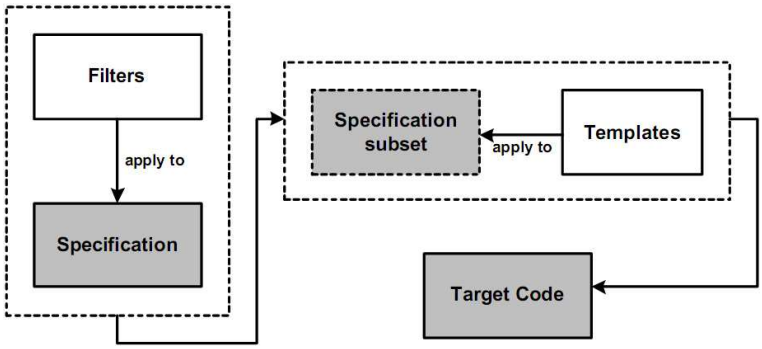


Figure 2.5: Templates and filtering program generation pattern [Voe03]

Templates and filtering

DESCRIPTION - Code should be generated from a higher specification out of a complicated model (e.g. XML-model). The generation of the source code from a UML software model is an example for that. Most of the information in the UML model is not needed for the source code generation.

SOLUTION - Most of the higher specification level information is not necessary and has to be filtered. On the filtered specification templates are applied to generate the target code. Fig. 2.5 illustrates this pattern.

DISCUSSION - Templates and filtering can be applied if the specification is well defined and the templates are well structured. The templates are tightly bound to the specification, which is a disadvantage of this template. However if the product line is very specific and does not change a lot, template and filtering provides an easy and fast way to develop a program generation system.

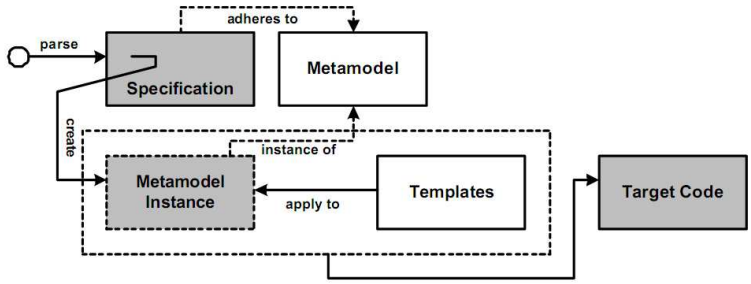


Figure 2.6: Template and metamodel pattern [Voe03]

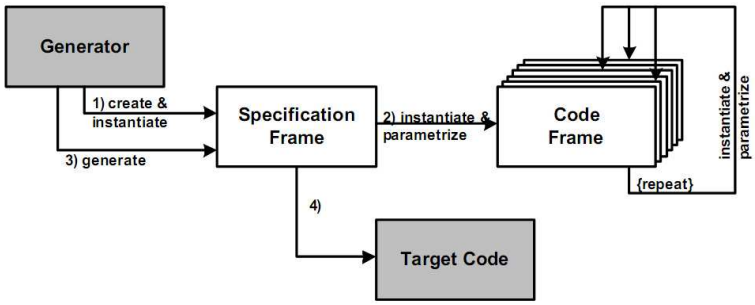


Figure 2.7: Frame processing program generation pattern [Voe03]

Templates and metamodel

DESCRIPTION - Architectural components representing the problem domain exist. These components have a clearly defined mapping to the implementation platform. Code generation templates should be specified platform independent.

SOLUTION - A clear defined metamodel for modeling the specific software products should be provided. Out of the model developed by the user and the metamodel, meta-code is generated. Templates are applied to this code to obtain the target code. Fig. 2.6 shows the structure of this pattern.

DISCUSSION - This pattern can be applied to product families. The model specification is completely independent from the templates. As the implementation information is in the templates, the metamodel can be platform independent and is therefore easier to maintain.

Frame processing

DESCRIPTION - Individual products of the same family with different features should be generated. The features can be clearly structured in modules or they can be interwoven in the specific system.

SOLUTION - Adaptive Templates (so-called frames) are used. A frame is some kind of function that generates code when it is executed. Each frame has slots where other frames or code snippets can be placed (Fig. 2.7). With this approach, the target code can be highly flexible.

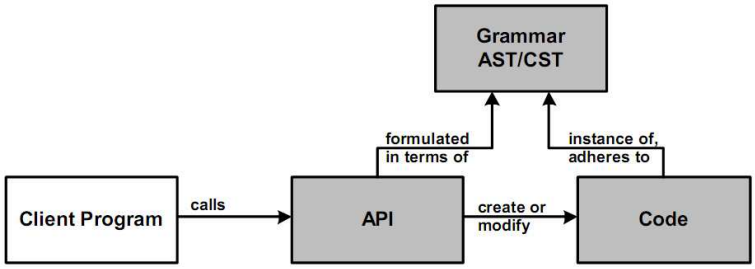


Figure 2.8: API-based generation pattern [Voe03]

Discussion - Frame processing is a well suited approach if a very specific small family of products which are closely related to each other is developed. For bigger SPLs this pattern is too sophisticated.

API-based generation

Description - Small pieces of code have to be generated on demand. The variability of the code is very small. The specification of the code to generate is not given.

Solution - An API is provided to allow parametrized code generation. This model does not use any templates. A third party program calls the API to generate specific code. Fig. 2.8 gives an overview of this pattern.

Discussion - This approach can just be used it the target model is not very complex. API-based generators often serve as basis for other generator types.

Inline Code Generation

Description - A software should be developed in one source project. The system has to be executable on many different platforms. For optimized execution on these platforms the code has to be slightly modified.

Solution - A preprocessor is introduced. At compile time, code is generated according to the parameters for the preprocessor. Fig. 2.9 shows the working principle of this pattern.

Discussion - Generally, this pattern is well known, as it is used in C++ and several other programming languages. Often, the preprocessor is built into the compiler, so no extra tools are necessary.

Code Attributes

Description - A specific source code should be generated out of a prototype. Some pieces of code have to be slightly modified depending on the specific system to generate. These changes can not be expressed in the programming language.

Solution - Additional information is hidden in the comments in the prototype. A code generator software parses the comments and generates code according to these instructions. Fig. 2.10 shows how this principle works.

Discussion - An advantage of this pattern is that code can be annotated with any kind of information. The disadvantage is that the new defined language for the

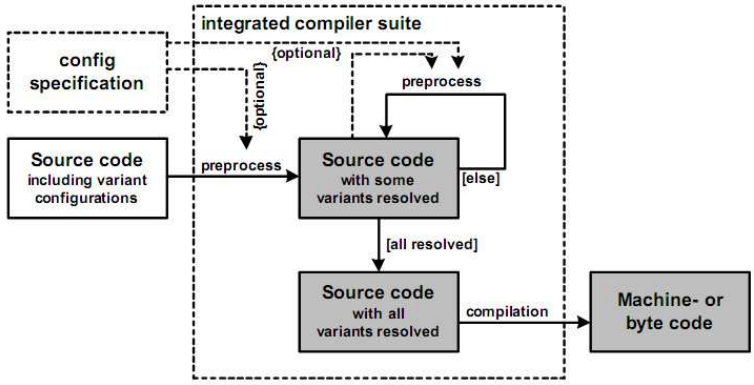


Figure 2.9: Inline code generation pattern [Voe03]

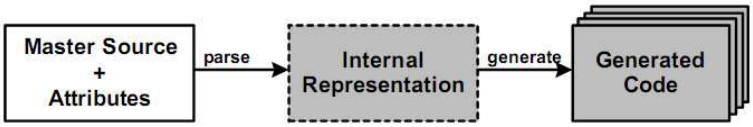


Figure 2.10: Code attributes pattern [Voe03]

information in the comments can get very sophisticated if this pattern is carried to the extremes. JavaDoc is a well known example for this pattern.

Code weaver

DESCRIPTION - Different source code modules should be joined in a program. These modules can be isolated or interact with each other.

SOLUTION - The code weaving pattern suggests the construction of a meta-module. The interaction with this meta-module is specified and the program is then constructed with these modules. The real source artifacts keep the constraints of the meta-modules. The code weaver then assembles the program out of the given components (Fig. 2.11).

DISCUSSION - The approach of this pattern is leaned on the concept of aspect oriented programming and on the model-view-controller pattern. The code weaving pattern is very complex to implement and just chosen huge product lines.

2.3.5 Software Product Line development methods

In this section several software product line architecture design methods are compared regarding their context, the view of the user and their structure. the information in this section is structured according to [Mat04].

MEDEIA MEDEIA [SSV08] is a project of the European Community's Seventh Framework Program (FP7). The aim is to provide a method for modeling automation product lines. The developer describes a project at a level of abstraction where he has to state

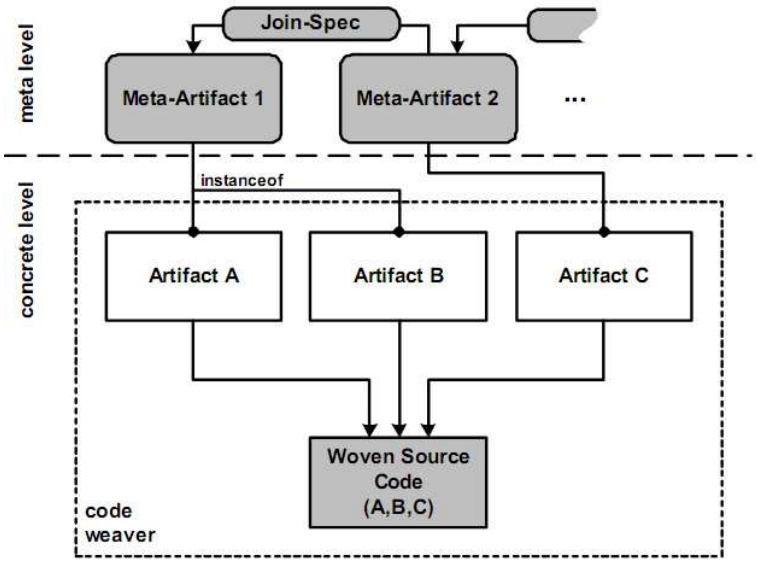


Figure 2.11: Code weaving pattern [Voe03]

what he wants and not how he can achieve that. It should be possible to combine different domain specific systems and approaches. This means that the system is designed to handle many domain specific views. The system architecture and components should not depend on the used languages for the system [SSV08].

MEDEIA provides a formal framework for model driven design with an integrated diagnostic system. The core of the framework are automation components (AC). They are hard- or software systems with defined interfaces. The AC's internal behavior and their hierarchical structure is described. Those components are not associated which the level of abstraction which is used. An example for a project structured with ACs is shown in Fig. 2.12.

COPA Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products (COPA) [OMA+00] has the goal to find the best balance between component based and architecture centric approaches. COPA also addresses business and organizational aspects. The aim of COPA is to use all-purpose software components very efficient.

The COPA methods starts by analyzing the customer needs. Stakeholder expectations and existing architectures are taken into consideration. The method produces a guideline for a software product line architecture. COPA also considers software maintenance and reusability. COPA does not just provide a software architecture, it addresses the whole software project.

FAST Family-Oriented Abstraction, Specification, and Translation process (FAST) [KE02] is a method to design software product lines which share common attributes (e.g. common behavior, common interfaces). The goal of FAST is to make the software production process more efficient by reducing multiple tasks.

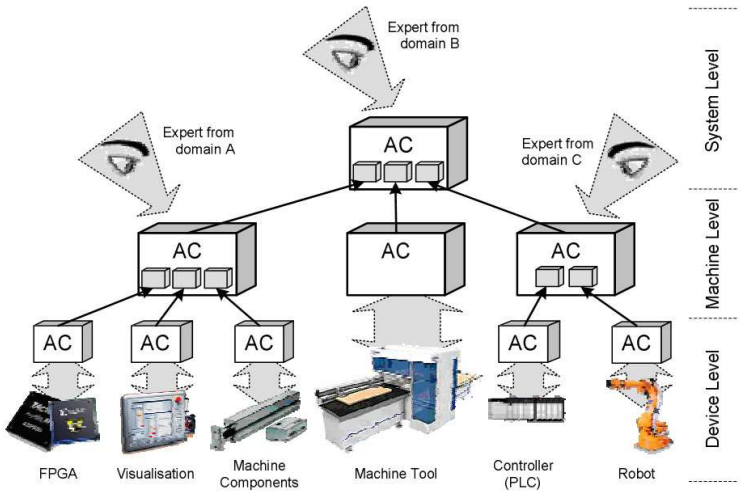


Figure 2.12: Hierarchical plant structure model [SSV08]

FAST provides a full product line process. The decision whether a domain specific product line should be build is made by an estimation of the sold products and their production cost. The software line engineering process is divided into two parts. The first part is the domain engineering part, which models the product family. Domain engineering also contains a requirements analysis and reusability assets are are developed. In the application engineering part the individual products are developed using the reusability assets.

FAST is well suited to develop applications quickly. FAST models variability of the system during the requirements engineering. The method uses no tools. It suggests guidelines to organize the architecture of the software product line.

FORM Feature-Oriented Reuse Method (FORM) [KKL+98] is an extension to the Feature Oriented Domain Analysis (FODA). FODA concentrates on the development of reusable assets. FORM extends this model to the software design and implementation phase.

By feature modeling and a context analysis the variability aspects of the system are captured. With this information a reference architecture is build.

KobrA Komponentenbasierte Anwendungsentwicklung (KobrA) [Mat04] is German and stands for 'component-based application development'. With the aid of UML this method aims for reusability of components and specifications. KobrA clearly states how UML should be used to achieve this goal. KobrA also defines implementation and testing aspects of the software product.

QADA Quality-driven Architecture Design and quality Analysis (QADA) [Mat04] is a quality driven design method. The design of the architecture is checked with a quality analysis which states whether the architecture is suitable. The methods delivers a design

and an analysis of the architecture. The analysis checks whether the quality requirements are met.

PuLSE Product Line Software Engineering (PuLSE) consists of three main elements: Deployment phase, technical components and support components. The deployment phase contains three stages. In the initialization stage the domain is defined. The infrastructure construction stage the product line characteristics are modeled and an architecture is developed. In the infrastructure usage stage the software product is developed. Technical components describe domain related technical issues. Support components handle organizational issues.

KobrA is an object-oriented approach of PuLSE. KobrA focuses more in the design and implementation of the domain framework [KE02].

Sherlock Sherlock is a method to develop software product lines which focus on market analysis during the requirements gathering. During the architecture development several tools are used. The method does not handle the specification of requirements. The architecture modeling process is separated in five stages where the architecture is defined, characterized, modeled and developed [KE02].

2.4 Model driven development

System engineers often build models of a software project for better understanding. With the help of this model the best solution for elaborating the software can be found. A key idea of MDD is that programs are automatically generated from their models. A model is a representation of the program at a higher level of abstraction. It is a formal specification of the function, structure and behavior of the system from a specific point of view [Tru06]. Details which are not necessary for a certain point of view are hidden. The model should be intuitive and understandable. The abstraction of the model is not defined. It can be high level (e.g. UML Diagrams model a software architecture) or low level (e.g. C++ code models the assembler code) [Sla06]. As it is shown by these two examples it is also not defined whether a model has a graphical or a textual representation. Model Driven Development (MDD) approaches software development by iterative building of increasingly detailed models. Early models emphasize requirements where more detailed models focus on software design and implementation [Sel03].

An advantage of MDD is that the models express the structure of the system in different levels of abstraction very clearly. Customers benefit from MDD by getting a better overview of the product structure. They can therefore easier understand the product features and intervene early if their requirements are not fully met. Due to the expressiveness of the models interorganizational workflows are also made easier, as it is possible for other companies to understand the features of the product at different abstractions [BCC+96].

2.4.1 MDD key attributes

The following attributes are listed according to [Sel03].

Code efficiency - When developing code with a model the code efficiency it is often a major concern. Codegeneration has the bad reputation that the software is not efficient. When the first compilers were build they had the same problem. Today's compilers can cope with that problem and produce a code that is sufficiently fast. There are just few applications where programmers write the source code in assembler because of the bad performance of a compiler. [Sel03] states that it will be the same with model driven design. The generation of the code from the models will be optimized and the usage of MDD will get common.

Model executability - For trend setting MDD approaches it is crucial that the models can be executed. Without the possibility to execute the model it is hard to debug errors.

Model-level observability - For compilers it is normal that, if an error occours, the position of the error is shown in the source code. This complex task should also be performed when using MDD. If an error in the source code exists, it should be automatically traced back to the model and shown to the developer.

Scalability - For huge software projects it must be possible that several people work on the same model. The tools and models must scale up to such a scenario.

Model driven architecture - Model Driven Architecture (MDA) is a concept which was introduced by the Object Management Group (OMG). It uses UML to design a platform independent model of a system. This model is then transformed to several other models with lower abstraction. MDA promises platform independent high quality software which is easy to maintain. Production cost should be lowered, testing and simulation is made easier [Tru06]. According to [NCP09] and [Wim05] MDA consists of the following phases:

- Defining a computation independent model (CIM). This can be models like a business model. or a domain model. It states the requirements and the environment of the system [NK04].
- Development of a platform independent model (PIM) with the aid of UML. This model represents the overall architecture of the software. Changes in the software technology does not influence the PIM.
- Transformation of the PIM to a platform specific model (PSM). Platform specific means that several technologies like interface functionality are defined. The PSM reflects the detailed design of the product. As the PIM, the PSM can also be elaborated with the help of UML.
- Transformation of PSM to implementation and runtime models. These models reflect the specific software for a platform. XML can be used to represent the model.

Each of the models is designed according to the Meta Object Facility (MOF). The MOF is a standard for the definition of metamodels. Elaborating MOF conform models

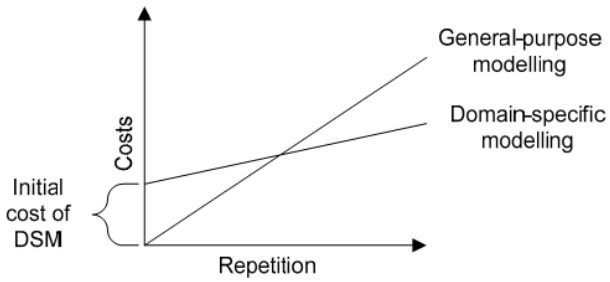


Figure 2.13: DSM costs vs. general-purpose modeling costs [Siv08]

makes the model transformation process easier [Bru07]. Through this transformations the development time of a product is decreased and human errors are reduced [Siv08].

2.4.2 Domain specific modeling

Just like MDA, domain specific modeling (DSM) aims at reducing the development costs and reducing human made errors. A big advantage of MDD is that the code is generated out of the model. This means that the model always is up to date. Compared to general-purpose modeling, DSM offers the advantage that models can be constructed, as current studies state, up to 10 times faster [Siv08]. This is the case because with DSM the gap between the domain model and the implementation has to be bridged just once. This work is done by a domain expert. Further modeling of products can be done by less experienced workers [Met09]. DSM take a lot more of initial work and therefore need upfront investments. For product lines with a lot of slightly different software products, DSM can achieve a significant reduction of development costs. Fig. 2.13 shows a comparison of the costs of domain specific models and general-purpose models.

Fig. 2.14 shows the concept of domain specific modeling. Usually the domain expert designs a framework which can be used by less experienced users to generate a domain specific code. It has to be decided whether the generated code is the finished product or whether is is part of a framework. It is often an advantage to generate code for a framework in complex projects, because then the code generation gets easier. Such frameworks exist in the most cases from earlier projects in the given domain.

Domain specific languages

A Domain Specific Language (DSL) is an abstract programming language which is customized for the given domain. DSLs can be either textual or graphical. Textual models are often described by their grammar, while graphical models are developed using a metamodel. Metamodels are models of models [Siv08]. A DSL is considered as good if it can express the problems in the domain very precisely in an uncomplicated way. A goal of DSM using DSL is that the domain expert and not the software engineer develops the model of the system. This is a big advantage, because the software engineer does often not have sufficient domain knowledge to elaborate a system-model [Sla06]. According to [BGK+06] and [Zdu05] a DSL is composed of:

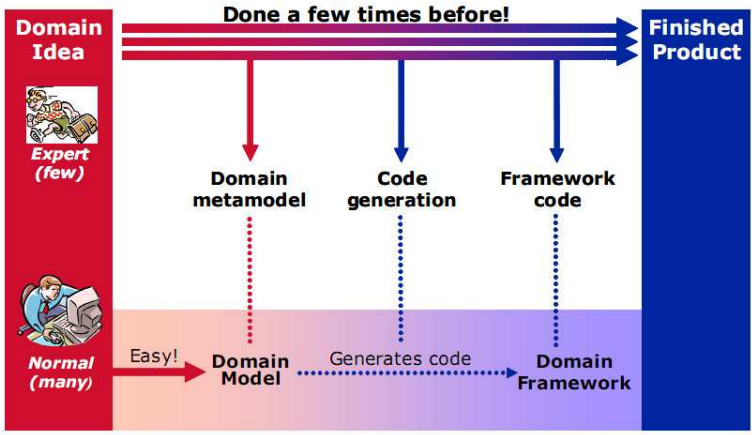


Figure 2.14: DSM concept enabling easier product development [Met09]

- Abstract Syntax - defines the concepts and relationships in the language. This syntax is often also called language model. It can be manually designed or derived by a metamodel like a UML model.
- Concrete Syntax - textual or graphical notation. This notation is then used by the developers to elaborate the specific software program.
- Semantic Domain - models the domain
- Semantic Mapping - relates the semantic domain to the syntactic concepts
- Syntax Mapping - relation between the abstract and concrete syntax

When using a general purpose language like UML for modeling a domain model, several problems occur. The language is often not precise enough to model the domain adequate. They often describe the system in some kind of natural language and not in a formal syntax. This means that the this language can not be used for automatic code generation.

For some domains it can be an advantage to create a domain specific model. This can be done by extending the syntax of a general purpose language line UML or by creating a new language.

Advantages of DSLs are that they model the domain at a very high level of abstraction. Still these models can be transformed to lower abstraction models or direct to source code. For domain experts it is a lot easier to understand domain specific models.

Disadvantages of DSLs are that not as many tools for code generation are available as for a general purpose language like UML. The process of generating the code is also often more complex.

Developing a DSL

An individual created language has to model the features of the domain precisely. It should also recognize whether a constructed model is valid or invalid. The interaction

of the user with the DSL must be specified [Bru07]. [Loh07] describes how to develop a domain specific language. This process is divided into three phases:

Decision phase The decision whether a DSL should be used is made. As help so called 'Decision Patterns' can be used. The base of this decision is the business model. It shows whether the development of a DSL is economic.

Analysis phase The specific domain on which the DSL should be applied is analyzed. The needed domain specific technologies and semantics are explored. For this phase a method like FORM (Section 2.3.5) can be used.

Design and implementation phase A domain specific language can use other software tools like meta-languages which could generate the DSL. Another approach is to implement the entire DSL by oneself.

2.4.3 DSM patterns

The following patters are guidelines for building frameworks [RJ96]. These patterns can also be applied for the design of domain specific languages.

Three examples

Description - It is hard to implement a framework for a domain specific language without knowing some examples for which the language is used. This pattern gives a solution for starting to implement such a framework.

Solution - Develop at least three prototypes for your application to get an idea about the variability of the system.

Discussion - The framework or the DSL will change in time. It is not developed once and never again altered. The first version does not have to be perfect. It should just give an idea which features are covered. As the framework will change it is no problem to focus just on some examples and model their features for the first version of the DSM-software.

Object granularity

Description - The granularity of the templates which are used for translating code with the help of a DSL has to be decided. If it is too fine-grained it is difficult to understand the language. Course-grained elements may not allow the needed variability.

Solution - Adapt the level of abstraction to the knowledge of the persons modeling the system. If the person has detailed domain knowledge it is possible to provide a DSL with many detailed elements. Do not lower abstraction if it is not necessary.

Discussion - To obtain the information about the knowledge of the modeler it is essential to have a detailed understanding of the specific domain. The aim is to keep the abstraction as high as possible.

2.4.4 Tools

Tools for generating a DSL on the basis of the language specification can be used. Those tools generate programs like compilers, editors with syntax highlighting or code generators. The advantage of the use of these tools is that common operations for the generation of a specific DSL do not have to be implemented. The usage of an appropriate tool can reduce the development costs for a DSL significantly.

MetaEdit+ is a collection of different tools for modeling, documentation and code generation. For developing a DSL first a metamodel of the domain is created. This model is build with the Graph-Object-Property-Port-Role-Relationship (GOPPRR) concept. MetaEdit+ generator definition language (MERL) is used to define the code generation from the graphical domain model [Loh07]. The software includes a debugging tool and an API to access all of the functions via SOAP. More about MetaEdit+ can be found in [TK09].

PROGRES (PROgrammed Graph REwriting Systems) is a tool for developing graphical DSL. PROGRES offers a tool for class diagram modeling, graph analysis, interpreter and compiler and a generator for graphical editors.

For DSL development the domain is modeled as a graph. When the DSL is completely modeled it can be compiled. With the language a graphical editor is generated. This editor can be used to model specific systems for the domain with the DSL [Loh07].

Telelogic Tau G2 is a model driven development environment. Normally UML 2.0 is used to elaborate metamodels. For more specific models the language can be extended via UML 2.0 profiles (see [OMG04]). For this toolset advanced knowledge of UML 2.0 is needed. The graphical DSL editors support some nice features like saving and loding a model. The graphical representation is limited and can not be customized [AFR06].

Rational Software Architect is a tool developed by IBM and is built on top the the Eclipse platform. It does not support as many UML 2.0 features as Telelogic Tau G2 (Section 2.4.4) and is therefore easier to handle. For DSL development a UML model has to be designed. Additionally, the generated editor can be customized by providing images and icons for the editor components [AFR06].

XMF-Mosaic is a Eclipse-based development environment. It supports the use of textual and graphical models for the generation of a domain specific language. The generated editor can be customized, but does not include basic functions like saving or loading a model [AFR06].

Eclipse EMF+GEF is a framework for building tools via code generation. The Graphical Editing Framework (GEF) can easily be combined with EMF metamodels and support the development of a graphical editor. It takes much effort to get to know EMF and GEF quite well, but a lot of documentation is available [AFR06].

DSL tools for Microsoft Visual Studio os a plug-in for Microsoft Visual Studio and allows the development of domain specific languages. With a graphical user interface the domain is modeled. Domain constraints have to be stated in an XML syntax. Finally the relations between the graphical model and the XML model have to be elaborated. These task can be done with the aid of a wizard. Microsoft provides online documentation for this tool set [Siv08].

Generic Modeling Environment (GME) is a free open source tool which can be used to develop domain specific languages. It is based on UML and provides visual modeling of the domain concepts. The implementation of the DSL can be customized by configuring the used bitmaps and icons. GME focuses on the construction of a graphical modeling tool. The model interpretation is not done by GME. This can be achieved by implementing it via a COM-enabled programming language like C++ or Visual Basic [Siv08].

B&R model embedded in Matlab Simulink. Bernecker&Rainer provides a code generation tool embedded in the Matlab Simulink framework [Wal10]. This tool allows to model a system with Matlab Simulink and generate B&R PLC source code directly out of the model.

2.5 The PISCAS approach compared

For the fish farm automation systems a centralized control system as presented in Section 2.1.1 is sufficient. PISCAS is based an a Bernecker and Rainer PLC system. Bernecker and Rainer was chosen because it provides a VNC remote visualization of the fish farm. A PC is therefore not needed at the fish farm. The visualization can be accessed over the Internet. This means that the system can be checked with a mobile device and service work for the operator of the fish farm is made easier.

For PISCAS several of the mentioned requirements engineering approaches from Section 2.2 were considered. For requirements capturing interviews and questionnaires provide the advantage, that the customer can directly influence the developed system. Case studies of competitors provide limited application, because PISCAS is a rather novel approach for fish farm automation. To verify the requirements continuous reviews with the customer were conducted. The requirements engineering method could be used for the domain engineering and for the application engineering of the PISCAS project.

To model a fish farm system a domain specific language (Section 2.4.2) was chosen which brings the advantage that systems can be modeled on a higher level of abstraction and tacit domain knowledge is expressed in the DSL. The selected approach adheres to the principles of MDD and brings the advantages mentioned in Section 2.4 like observability and scalability of the model. The presented patterns in Section 2.4.3 and 2.3.4 support the development of the product line. For the code generation the templates and metamodel pattern promises good results as it allows to implement existing PLC templates in the code generation process.

For the selection of a suitable tool for DSL development an evaluation framework proposed by [Lei09] was used. The results as presented in Tab. 2.1 shows that the software

MetaEdit+ is most suited for the given project. The used metrics are explained in the Appendix A.

	Tools	pure:variants	Gears	Feature Modeling Plugin	XFeature	MetaEdit +
Criterion	Weight	Rating	Rating	Rating	Rating	Rating
Attributes Management	8	5	7	5	5	2
Feature and Variability Modeling	10	10	9	10	7	7
Feature Metamodel Maturity	5	3	1	8	6	8
Constraint Checking and Propagation	8	10	8	8	8	9
Product Derivation	7	10	9	7	7	9
Domain Engineering Management	7	7	8	5	5	5
Repository	5	6	9	6	6	6
Traceability Management	1	4	4	1	4	5
Impact Analysis	6	4	8	2	2	9
Reporting	10	8	9	2	2	9
Access Mode	5	2	1	2	2	9
Technical Environment	10	8	4	5	5	9
Usability	10	6	8	3	4	7
Automatic Filters	3	5	3	3	3	6
Tool Configuration	10	7	2	3	10	9
Extensibility	10	10	5	6	4	5
Flexibility	10	8	6	6	6	10
AOB	10	9	5	6	7	7
Benchmark	1000	739	619	519	543	**747**

Table 2.1: Results of the tool evaluation

Chapter 3

Design and implementation of the PISCAS product line

In this section the fish farm product line is presented. With the theoretical background from Section 2 domain engineering and application engineering for the SPL have been conducted. Information about the SPL and PLC architecture is given. The metamodel and the generated PLC code parts are discussed in more detail. A cost model is proposed and an evaluation of the costs for fish farm automation systems was conducted. Additionally information about the installation of PISCAS for two real-life systems is given.

3.1 Domain engineering

3.1.1 4+1 viewpoint model

This section describes the basics about an architecture model which can be applied to the domain.

Philippe Kruchten's 4+1 viewpoint model [Kru95] describes the architecture of software-intensive systems. It provides multiple views of the product for different stakeholders. The design decisions for a system can be represented with these views. The model contains redundant information about a system. The architecture is presented in different ways, so that different stakeholders can easily access the information they are interested in from one of the views. The model is shown in Fig. 3.1.

Logical architecture - The logical view provides a look at the product from the customer's point of view. Most part of it can be described by the domain specific functional requirements for the system.
Process architecture - The process architecture describes non-functional requirements like performance and availability. It also gives an overview of different independent sets of tasks for a project. These sets are called processes. The interaction between processes is described.
Development architecture - This architecture describes the system from the point of view of a developer. In general the software design is presented. Internal requirements like reuse of commonality and software management are presented in this view.

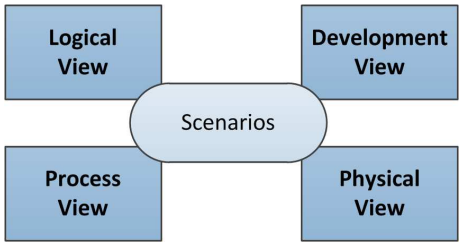


Figure 3.1: 4+1 viewpoint model [Kru95]

Physical architecture - The physical view describes the mapping of the software on the hardware. Different hardware configurations are considered. The aim is to minimize the effort of changing the software according to different hardware.
Scenarios - Scenarios are the '+1' in the '4+1 viewpoint model'. They give no additional information, but they provide a link between the different views. Scenarios can as an example be use cases representing the most important requirements.

Proposed approach

The main problems are that it is very likely that an automation system software contains many errors and that the domain knowledge of the programmers is implicit and not accessible by others. The lack of communication between the two different companies also is a problem.

To overcome those problems software product line engineering integrated in the 4+1 viewpoint model is suggested. With the SPLE approach many of the stated problems can be solved:

- An automation system is modeled with a domain specific language and the software is then generated from this model using a code generator. If a software error occurs then the code generator is corrected and the same error should not occur twice.
- Domain knowledge is contained in the metamodel and therefore not just accessible by the person who developed the software.
- Changes to the system (e.g. new components) can easily be realized as just the model of the system has to be changed and a new code has to be generated. This is a lot easier than programming the new modules by hand.
- Together with the software an electrical installation plan can be generated. This can be a good basis for communication between the automation system providing company and the electrician.

Project stakeholders

To apply the 4+1 viewpoint model the relevant stakeholder for the project have to be known. The analysis of the stakeholders has been conducted with the aid of the Volere Requirements Specification Template [RR]. The overview of the stakeholders shown in

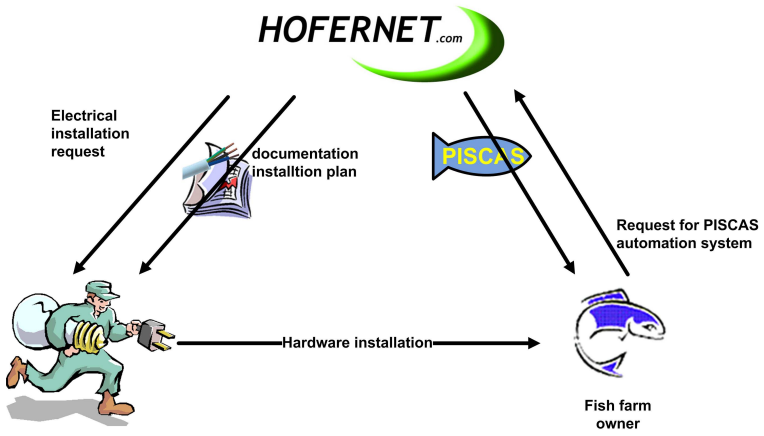


Figure 3.2: PISCAS main stakeholders

Tab. 3.1 gives a good understanding of the parties involved in the project and of their contribution. The most important stakeholders for the project and their relationships are shown in Fig. 3.2.

4+1 viewpoint model applied to PISCAS

With the information about the stakeholders the specific 4+1 viewpoint model for PISCAS can be elaborated. The logical view represents the view from the customer who is the fish farm owner for the PISCAS project. It includes the requirements of the customer which are stated in Section 3.1.4.

The electrician is a stakeholder of the physical view. He is interested in hardware and the link between software and hardware. This link could be a hardware test provided from the automation software as it is the case with PISCAS. Therefore the automation system software is also part of of the physical view. The software design ensures that changes from the physical view (e.g. different hardware modules) do not cause big changes on the software side. Also the fish farm owner is stakeholder of the physical view as he is interested in the feeders and aeration systems which are used for the fish farm. The generated documentation can give the fish farm owner an overview of the hardware.

The development view is part of the model where the proposed solution can be seen. This is the part where the software product line engineering takes place. The stakeholder involved in this view is the company Hofernet which develops the automation system. In this view included is the metamodel, the code generator and the template of the PISCAS software. The aims of stakeholders of this view is to develop a high quality system very efficiently. To achieve these aims a SPLE approach is chosen.

The most interesting architecture is the process view. Here also coordination between the two participating companies is described. This part should be improved with the proposed method because of the automatic generation of an electrical installation plan. With the help of this plan the two involved companies have a defined basis for the rest of the necessary communication. The test mode of PISCAS also allows better cooperation between the companies Hofernet and the electrician as hardware errors can be detected

Table 3.1: Full list of PISCAS project stakeholders

Role	Name	Rationale (why does he have to be involved)	Involvement	Knowledge	Communication	Deliverables from the stakeholder	Deliverables for the stakeholder
Project sponsor	Michael Hofer	Project principal, he pays for the project	Micheal Hofer pays the TU Graz for conducting the PISCAS software generation project. He is the most important stakeholder for the project. He specifies most of the requirements. He determines the maximum project cost and decides about expenses.	He has moderate fish farm domain knowledge and very good project hardware knowledge and very good knowledge about business constraints for the project	direct	B&R SPS System from the company Bernecker & Rainer, fish farm hardware	PISCAS code generation system
Fish farm owner	Andreas Hofer	Fish farm owner, specifies some requirements	A recent fish farm automation system (without the PISCAS code generator) is currently developed for Andreas Hofer	He has very detailed fish farm domain knowledge	via Michael Hofer	Knowledge, possible fish farm structures	PISCAS system
University	TU Graz	Allows the graduand to work on the PISCAS code generator project as a master thesis	The university provides the graduand necessary tools for developing the pisciculture automation system as a master thesis	Software Product Line Engineering knowledge	direct, project advisor: Christian Kreiner	wokplace, tools, for the project necessary software	master thesis, publishable paper
Financial Support	FFG	The 'Österreichische Forschungsförderungsgesellschaft' (FFG) granted Michael Hofer the 'Innovationsscheck' which allows him to make use of research and consulting services of an educational institution.	The 'Innovationsscheck' has to be redeemed until 27.01.2011. The master thesis therefor has to be finished at latest at this date	no necessary knowledge	E-Mail	'Innovationsscheck' for Michael Hofer	confirmation that Michael Hofer redeemed the 'Innovationsscheck' at the TU Graz and that he therefore got research aid from the university in form of a master thesis
Automation System manufacturer	Bernecker & Rainer	Provides necessary hardware	helps with decisions regarding hardware and software development	very good B&R Automation Studio knowledge, some code generation knowledge, very good B&R hardware knowledge	Code generation: Philipp Wallner Automation Studio development: B&R office Graz	B&R SPS System for Michael Hofer	none
Electrician	Elektro Tisch	Installs the electrics for Andreas Hofer's fish farm	Needs information about the automation system to install the hardware	no necessary knowledge	via Michael Hofer	installed hardware for the fish farm of Andreas Hofer	electrical installation plan for the fish farm
Project engineer	Christopher Preschern	Develops the PISCAS code generator	The graduand Christopher Preschern works on the PISCAS master thesis for the TU Graz on behalf of Michael Hofer. Christopher Preschern is responsible for the research, design, implementation and testing of the system.	moderate fish farm domain knowledge, moderate code generation knowledge, good B&R Automation studio knowledge, some fish farm hardware knowledge	direct	PISCAS code generation system, master thesis paper, publishable paper	B&R SPS System, diploma

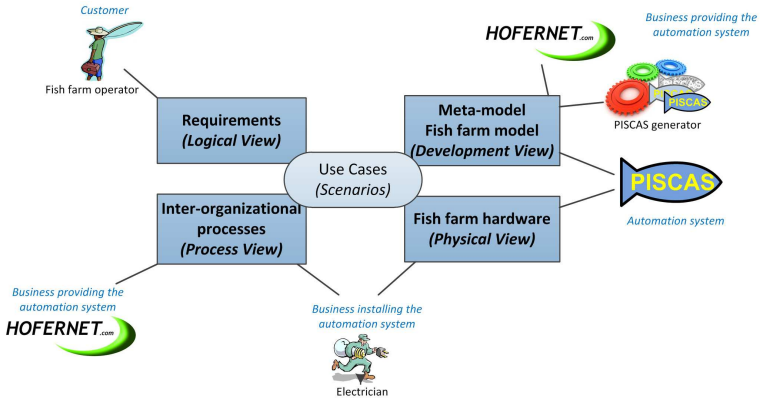


Figure 3.3: 4+1 viewpoint model for PISCAS

easier.

The scenarios are use cases which were worked out of the collected requirements (Section 3.2.2). Fig. 3.3 shows how SPLE can be used in combination with the 4+1 viewpoint model. Even though the specific case for PISCAS is shown, this is a general approach and can be applied for most automation system projects. This concept allows to produce automation systems more efficiently if more systems of the same family are developed. A disadvantage of the approach is that it needs an upfront investment. More details about the business model can be found in Section 3.1.2.

3.1.2 Business model

The described business model is structured according to [Sta02]. It is the basis for the business model used for this project.

Value proposition

The in course of this thesis developed system enables the possibility to provide an individual offer for a pisciculture automation system to a much lower price compared to business competitors. A comparable framework does not exist at the time. Pisciculture automation system (PISCAS) is the first PLC system which introduces a fish growth model to this domain. This reduces the service workload for the operator of the pisciculture a lot, because the food amount need not be adjusted so often. Information about the current amount of fish is also provided due to this growth model. This saves the necessary work to figure out the fish amount by weighing the fish in a pond. Additionally the whole fish farm is rationalized by saving resources like current or fodder as these resources are utilized more effective. Critical states are reported and the operator of the fish farm is alerted. This can prevent high damage costs. PISCAS provides a cheep way to automate a pisciculture.

Value chain structure

Due to the automatic code generation a project can be constructed and changed with very low effort. The framework is designed modular. Changes in the domain do not require

costly changes of the framework.

The design of the software already considers future maintenance aspects. To keep the maintenance work low it is inevitable to allow changes in existing PISCAS systems. For changes in a facility the fish farm configuration can be saved and a new PLC system can be generated. Afterwards the old configuration can be loaded again. Changes in the domain like new kinds of fodder or new feeding automates have to be configured. This is done by the vendor by providing these configurations. They can be applied to the PLC system via remote access.

The maintenance tasks are not linked to high effort of work. This means these tasks can be accomplished very effective.

To reduce issues regarding inter-organizational communication problems PISCAS generates detailed documentation for companies involved in the development process of a fish farm automation system. The installation plan for the electrician is generated. Also the design of the hardware is held easy to ensure that the installation can be done without problems.

Revenue generation model

Most of the work time for constructing a new fish farm automation system has already been spent by developing the code generator system. This means that the development costs have to be divided among the expected sold products. A detailed cost model of code generator based products can be found in [BBMY04]. Due to the low work effort to install a new automation system with PISCAS a margin of profit can be drawn. This means that just few fish farm automation systems have to be sold to justify the use of a project generating software. More specific data can be found in the Appendix 4.4. Maintenance efforts are very low therefore result in increased profits.

Business model innovation

Innovations in business models can lead to a significant competitive advantage [DLRS09]. In this section the innovations in the presented fish farm system are summarized.

In the fish farm domain PISCAS introduces the innovative approach to calculate a stochastic fish growth model. This model reduces service work amount for the pisciculture operator.

According to [FVLD03] innovations can be aided by systematic inter-organizational relations. The electrical installation of the fish farm is done by another firm. To simplify this process the electrical installation plan for the pisciculture is already provided by PISCAS. Additionally the hardware is designed in a way that is simple to assemble. Therefore the inter-organizational workflow is handled in an effective way. For further information about the advantages of structured inter-organizational workflows see [Sch97].

PISCAS cost model

The proposed business model is applied to the fish farm project. This model consists of an upfront investment which is needed to develop the code generator for the domain. Developing new fish farm automation systems just include modeling the fish farm on a high level of abstraction. This means that the costs for new projects are very low compared to

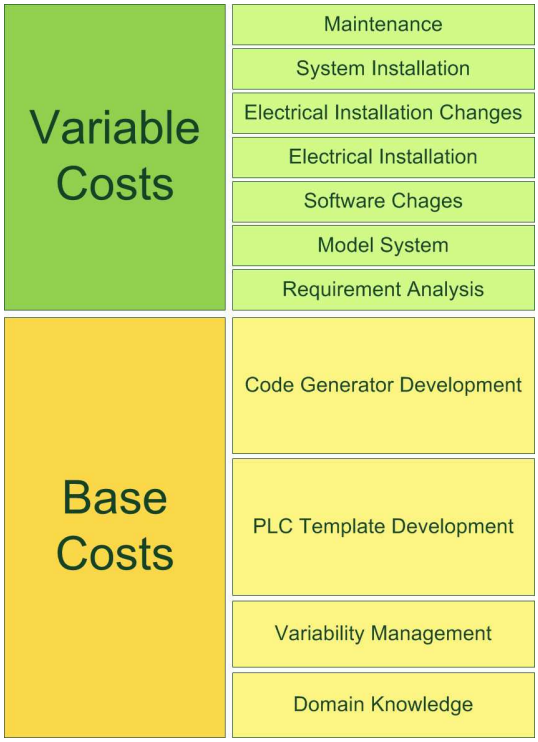


Figure 3.4: PISCAS cost model structure

elaboration of an automation systems without the aid of a code generator software. The general structure of the business model can be seen in Fig. 3.4. The costs are divided in base costs, which are basically of the development cost of the code generator, and variable costs. These are the costs which have to be covered for each new automation system.

3.1.3 Product scope

The product scope describes the rough components of a new PISCAS project. The involved parties and their interaction with parts of the hard- and software are described.

System installation

The vendor of the fish farm automation system is responsible for organizing the system installation process. He has to gather the requirements for a new project and model the fish farm with the aid of the code generator tool. The code generator software generates a full PLC software and a documentation of the project. Hofernet then commissions an electrician to install the project hardware. The electrician gets a generated documentation of the project. The orange arrows in Fig. 3.5 represent the interactions between different parties and components of the project during the system installation.

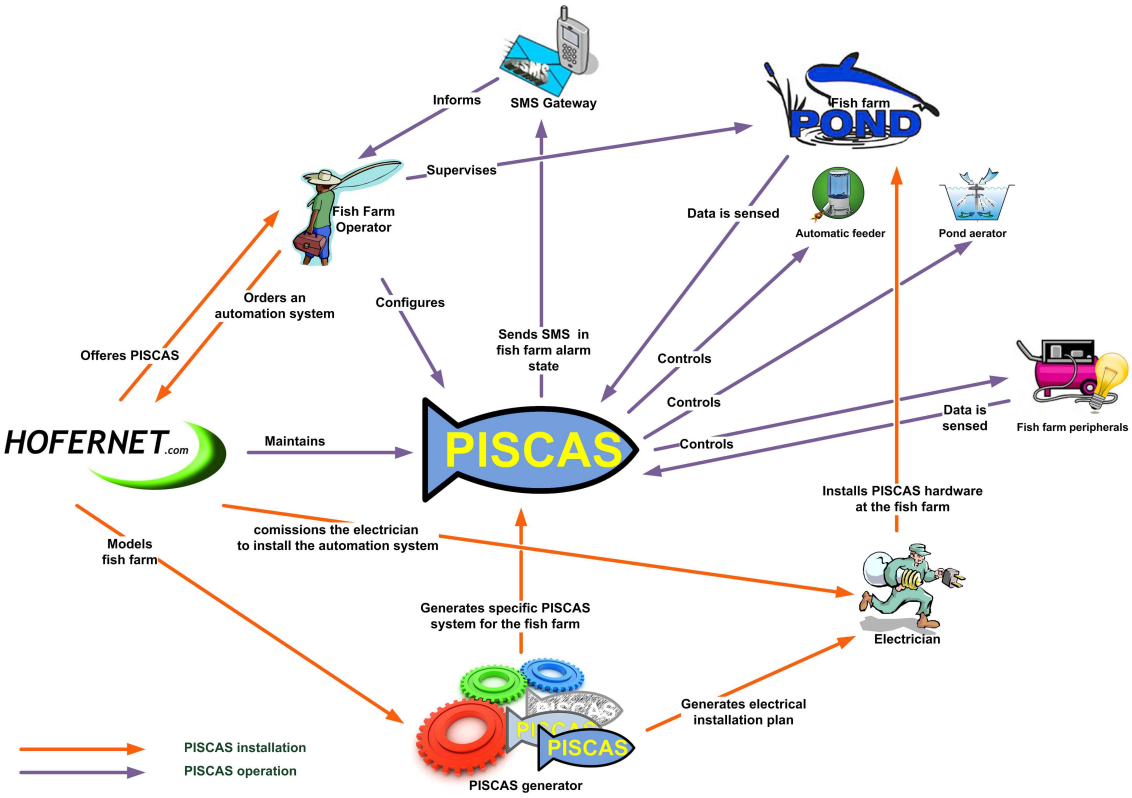


Figure 3.5: PISCAS Product scope

System operation

During operation of the automated fish farm the company Hofernet maintains the software. The fish farm owner configures the software and is able to supervise the current state of the fish farm. Interactions during operation of the automated fish farm are illustrated as purple arrows in Fig. 3.5.

3.1.4 Requirements

The requirements for the project were gathered using the Volere Requirements Specification Template (see [RR]). Additionally to that two questionnaires were conducted. One concerned necessary features of an automated fish farm. The questionnaire was directed of fish farm owners. The results can be seen in Appendix B.1.

The other questionnaire concerned inter-organizational processes between automation system providing companies and electrician companies. Out of this questionnaire several requirements regarding the automatic generation of a installation plan arose. The results of the questionnaire can be found in Appendix B.2.

General requirements

GR1 - Reduction of errors in the software

Description and motivation - Code generation systems can provide software which usually contains fewer bugs than software which is programmed by hand. The use of an error-free PISCAS is more convenient for the customer. The fish farm operator can easier trust in the software if it does not fail often.
Fit criterion - Maintenance work due to errors decreases.

GR2 - Reducing the production time

Description and motivation - Developing an automation system takes a lot of time for the software. With the code generator system this time should be reduced drastically. The motivation is that the profit of selling an automation system can be significantly increased because of the few hours of work needed. For the customer the system is still a lot cheaper than comparable automation systems from other companies.
Fit criterion - Constructing a fish farm automation system with the help of the code generator decreases the development time significantly.

GR3 - Generating software which does not have to be adapted afterwards

Description and motivation - The aim of an automation system code generator is that someone without a lot of software knowledge is able to create a project. This means that the source code should be completely generated and does not have to be altered, which leads to a shorter production time of the automation system.
Fit criterion - The code can be applied to a PLC system without changing it.

GR4 - Standardized hardware

Description and motivation - Hardware for the automation system should be standardized as far as possible. If a replacement part is needed in some years it should be no problem to get it.
Fit criterion - This requirement can be assured, by checking the future availability of hardware parts of the PLC company.

GR5 - Integrity

Description and motivation - The template of an automation project for the code generator must be configured in a way that it checks if a modeled system cannot be generated.
Fit criterion - Before generating a project the model is checked for its integrity.

Functional fish farm requirements

FR1 - Risk reduction by alarm state information

Description and motivation - For a fish farm operator it can be a tremendous incidence if for example the oxygen value of a pond gets too low. In the worst case all the fish in the

pond die, which can be a huge financial loss. PISCAS should provide an interface which allows checking the status of the fish farm online. If a critical state occurs, the operator is informed via an SMS.
Variability - Systems can be delivered with or without an SMS-Gateway module.
Fit criterion - A reduction in the loss of fish due to system malfunctions.

FR2 - Variability in hardware modules

Description and motivation - The hardware and the software have to be flexible. It is very unlikely that a fish farm operator will buy new automatic feeders or new oxygen suppliers for the ponds. These systems are already present at fish farms in most cases. PISCAS has to be compatible with a variety of oxygen and feeder modules.
Variability - The system has to provide functionality a variety of feeders and oxygen sources.
Fit criterion - Code generation software does not have to be changed for equipping a new fish farm using different hardware.

FR3 - Power supply

Description and motivation - Many fish farms possess a power generator. PISCAS should be able to handle it if a blackout occurs. The generator is often needed to ensure a sufficiently high oxygen level for the ponds.
Variability - The software has to be able to handle variants of power generators.
Fit criterion - Variants of power generators can be used with PISCAS.

FR4 - Test mode

Description and motivation - PISCAS should provide a test mode. In this mode the outputs of the PLC can be switched on and off in the software without any interference of the PISCAS program. This mode is needed to test whether the hardware works correct.
Functionality - Feeders, aerators and peripherals can be switched on and off manually.
Fit criterion - The system can be tested with this special mode.

FR5 - Logging

Description and motivation - Essential data, like data about the oxygen value and the fish fodder should be logged. This data helps debugging and also can serve as an evidence for the correct functioning of the system.
Functionality - Files should be stored on the PLC and a backup should exist.
Fit criterion - Data is being logged.

FR6 - Load/save settings for a fish farm system

Description and motivation - The settings stored for a fish farm project (kind of feeder for a pond, fodder size, ...) can be stored in a file and backed up. If a PISCAS has to be changed (e.g. new module) this data can be saved, a new system can be generated and the old data can be restored. This leads to less time needed for maintenance due to changes for a system.

Functionality - System settings are stored on the PLC. Software updates do not reset these settings.
Fit criterion - Effort to change a fish farm system is significantly decreased.

FR7 - Internet maintenance

Description and motivation - Maintenance of the system should be able via the Internet. This reduces effort in case of a problem with the automation system.
Fit criterion - The vendor of PISCAS can access all necessary data and programs over the Internet.

FR8 - Fish growth model

Description and motivation - After configuring the amount of fish in a pond, PISCAS automatically calculates the current amount of fishes by considering the fed fodder and the temperature. The system adapts the amount of fodder to the amount of fishes. The operator of the pisciculture does not have to adjust the fodder amount.
Fit criterion - Corrections of the fodder amount have to be done less often.

FR9 - Oxygen level control

Description and motivation - This feature is very important to the project. The level of oxygen is essential for the fish in a pond to survive. If the oxygen level sinks, countermeasures have to be taken. PISCAS should be able to control different kinds of oxygen supplies.
Fit criterion - Different oxygen systems can be used with PISCAS.

FR10 - Fair feeding

Description and motivation - In many fish farms the automatic feeders are not used all the time. To reduce the costs of PISCAS the used feeders use just few power supply units. As the current of the power supply units is limited, the feeders are not allowed to feed all at once. A fair scheduling for the feedings has to be applied if more than one feeder wants to feed at a time. Feedings should also be spread across the whole day while there is light. This scheduling provides well-directed use of fish fodder.
Fit criterion - The scheduling for many feeders works fair and correct.

FR11 - Documentation

Description and motivation - For easier maintenance it is necessary to provide a consistent documentation for a fish farm project. This documentation is generated with the PLC source code. The documentation includes the PLC pin-assignment, a project overview and maintenance relevant data.
Fit criterion - A consistent documentation is provided with each generated product.

FR12 - Measure water level

Description and motivation - The water level of a pond is measured. It is a critical state if the water level is too low. In that case an alarm message should be send.
Fit criterion - The water level is sensed and an alarm is sent if a critical state occurs.

FR13 - Selectable fish species

Description and motivation - Different fish species need different amount of fodder. To calculate the right amount the species has to be known.
Fit criterion - The system feeds according to the configuration of the fish species.

Non-functional fish farm requirements

NR1 - User-friendly fish farm modeling interface

Description and motivation - As the software might just be used occasionally it is important that the user interface for modeling the fish farm is intuitive and easy to handle. The benefit of this goal is that a new user of the software does not have to be trained a lot.
Fit criterion - Inexperienced users are able to model a fish farm.

NR2 - Easier operation of the automated fish farm

Description and motivation - The operator of the fish farm should have less work with handling the fish farm with the automation system. This leads to a high customer satisfaction.
Fit criterion - Positive feedback of fish farm operators.

NR3 - Low maintenance effort

Description and motivation - The work for changing a fish farm (e.g. adding a module) should be very low. With low maintenance effort it is easy to get a high profit for maintenance work.
Fit criterion - Systems can be generated and altered with low effort.

NR4 - Resource efficiency

Description and motivation - Fish fodder and energy should be used more efficient with PISCAS. This increases the customer satisfaction because his costs for these resources decrease.
Fit criterion - A comparison of the fish farm energy and fodder cost before and after the usage of PISCAS shows that the costs are reduced.

NR5 - Scalability

Description and motivation - It should be no problem (e.g. due to processing power) to scale the system in a given range.
Fit criterion - A system with the maximum number of 255 ponds can be generated and works.

NR6 - User interface usability

Description and motivation - The visualization of the fish farm should provide a good overview of the current fish farm state. Configurations should be intuitive. A new user should be able to operate the system.
Fit criterion - Several untrained users can use PISCAS without any problems.

NR7 - User interface appearance

Description and motivation - The appearance of the visualization might have to be changed according to the customer (e.g. corporate design has to be used).
Fit criterion - The colors and the design can be easily configured.

NR8 - Reliability and availability

Description and motivation - It is essential that the product does not fail undetected. If the system discovers a state which it cannot handle, then an alarm (SMS) has to be sent.
Fit criterion - The system does never fail without sending an alarm.

NR9 - Robustness and fault tolerance

Description and motivation - If the system discovers an error, it should issue a warning and continue its work. If as an example a oxygen sensor fails, then the pond has to be provided with oxygen (to prevent a too low oxygen level) and an SMS is sent to the fish farm operator.
Fit criterion - Fault states like broken hardware (sensors, power generator) are successfully tested.

NR10 - Capacity, storage

Description and motivation - Data about the system are logged. This data is stored on the PLC. The PLC must provide enough storage for this data. If the storage is full the PLC should overwrite the old data by itself. No extra maintenance should be needed for that.
Fit criterion - Data is logged and no maintenance has to be done for ensuring enough storage.

NR11 - Software access

Description and motivation - The fish farm operator should be able to access the fish farm overview via the Internet. He can see the state of the pisciculture and adjust several settings. The vendor of PISCAS can access additional configuration settings.
Fit criterion - PISCAS provides two login modes. A user can access normal functions and an administrator can access everything.

NR12 - Easier cooperation with other firms for installing PISCAS

Description and motivation - It is often a problem that due to a lack of communication between different firms, which are working on the same project, misunderstandings occur. This can lead to problems for installing PISCAS as the hardware might be installed by an electrician and not by the company providing the automation system. To bridge this gap, PISCAS provides automated generation of electrical installation plans.
Fit criterion - The cooperation with an electrics firm is eased, which can be measured by comparing the problems because of inter-organizational misunderstandings with and without the code generation system.

3.1.5 Modeling language

Choosing language for modeling the metamodel for the software product line is an important task. Basically it can be differentiated between standardized modeling languages (e.g. UML) and domain specific languages.

UML brings the advantage that it is well known and many tools for this language exist. A disadvantage is that the metamodel of the product line might be unnecessarily complicated if it is modeled with a language that is too general. [CRR09] states that general models are harder to maintain than domain specific models.

Domain specific language tool support is not as good as it is for UML. Models with a domain specific language can be elaborated a lot faster than with UML, because UML is very generals and often does just support the features of the domain. Domain specific models can also be understood easier and need not be provided by experts [Jun08].

The decision for the kind of modeling language is often linked with the level of granularity (Section 2.3.2) the software product line should have.

MDA will not be used, because this standard involves the use of UML. For PISCAS a domain specific language is used, because it can model the domain more specific in an easy way [CRR09]. The tool used for creating a DSL is MetaEdit+ from MetaCase. This software satisfies all needs for creating a fish farm metamodel and performed best in an evaluation of different metamodeling softwares. The evaluation can be found in Appendix A.

3.1.6 Metamodel concept

The metamodel allows to model several different fish farm automation systems.

The metamodel allows to model the logical software constraints and the physical constraints (which are needed for generating a installation plan) in one graph. The objects of the system are dragged into the model and connected as their physical connection will be.

The alignment of the objects to each other represent their position in the real fish farm. Out of the information of this graph the PLC software and a documentation including an overview of the pin assignment of the system are generated.

The decision that the physical pin assignment has to be made manually was made because this makes modeling existing systems easier. If an existing automation system is changed, then the existing parameters (like the pin assignment) have to be taken info consideration for the model. This is the reason why the pin assignment has to be explicitly modeled.

3.1.7 Variability

Ponds

- Amount - The number of ponds for each PISCAS project can be variable. It can be adjusted by the amount of pond objects in the fish farm model.
- Type of oxygen supply - The type of oxygen supply can be modified. Each pond provides an option where the supply type can be defined. A pond can also be connected to an external module. This provides the possibility that more ponds share an oxygen supply.
- Type of Feeder - The feeder type can be defined for each pond.
- Feeding Power Supply - This option allocates the ponds to their power supply for feeding. This information is needed to provide the scheduling of the feedings.

Power generator

- Type - The standby set type can be defined in the model. The type changes the start behavior of the device.

PLC I/O modules

- Type - States which kind of I/O module is used. The I/O modules are connected to other objects in the model (e.g. ponds). This connection defines the hardware connection between the modules and the fish farm devices. The reason why this mapping is modeled manually is that that makes maintenance work easier if an existing system has to be changed. Automatic mapping of the I/O modules would not be able to cope with that.

Switches

- Type - The type of switch defines it's behavior. Push buttons, normal switches or time triggered switches can be used.
- Output - An output is a representative for a universal device. The combination of switches and outputs lets the modeler develop individual part of the system which low dependence on the fish farm domain.

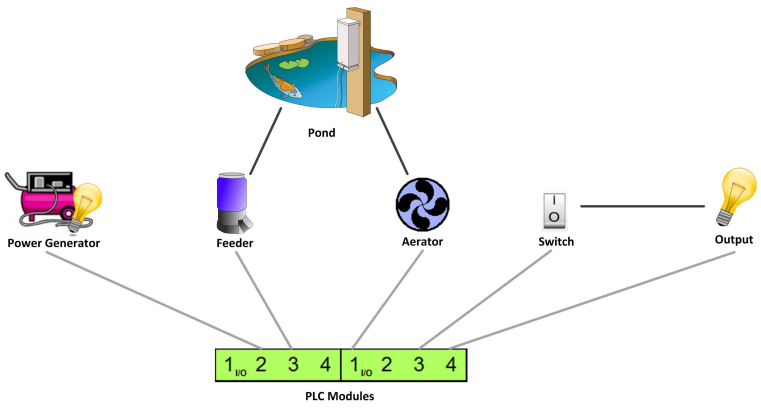


Figure 3.6: PISCAS Meta-Model

3.1.8 Metamodel

The presented metamodel is a result of the gathered requirements for the fish farm domain. With this metamodel it is possible to model all of the during the design process given fish farm systems. These include three different real life fish farms.

An overview of the metamodel elements and their relations can be seen in Fig. 3.6. The number of the given elements can be variable and the connections are optional. With this metamodel it is possible to model a whole fish farm system.

The used B&R I/O modules are modeled explicitly. The reason for not generating this mapping automatically is that it should easily be possible to extend an existing fish farm automation system. This is just possible if one can configure the mapping. Otherwise existing fish farm systems could not be modeled with the code generator tool, because their real I/O mapping would not accord to the mapping generated by the tool. Another reason for explicitly modeling the hardware is that different kinds of I/O modules can be used.

3.1.9 Generators

Fig. 3.7 presents an example for a simple modeled fish farm system. The mapping of the model components to the specific parts of the PLC code and the documentation is shown. The connections in the model between objects like ponds and feeders contain information which is used to configure the cyclic PLC programs. More specifically, these connections establish internal mappings between PLC software parts. This connection information in the model is also used to establish the connections between the objects in the automatically generated visualization and in the documentation. The information in the model about the hardware mapping is needed to map the corresponding variables of the PLC program to the inputs and outputs of the PLC.

Consistency check

A model is restricted to severals constraints. The output of a feeder for example cannot be connected to an input module. To assure that such constraints are maintained, a

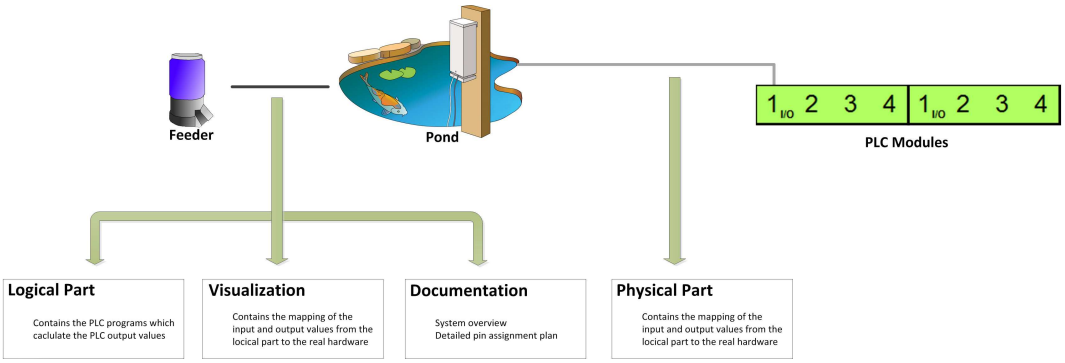


Figure 3.7: Mapping of the PISCAS model to the generated artifacts

consistency check of the model is performed before the code is generated.

Documentation

From the model for the PLC software also the documentation for the project is generated. The documentation consists of a graphical overview of the system and detailed information about the pin assignment. The information for generating the pin connection part of the documentation is stored in the direct connections between the PLC modules and a corresponding object in the model. To generate the overview the information about the positions and connections of the ponds in the model are used. of The documentation makes the hardware installation of the system a lot easier. The generated files are Scalable Vector Graphic (SVG) files and Rich Text Format (RTF) files.

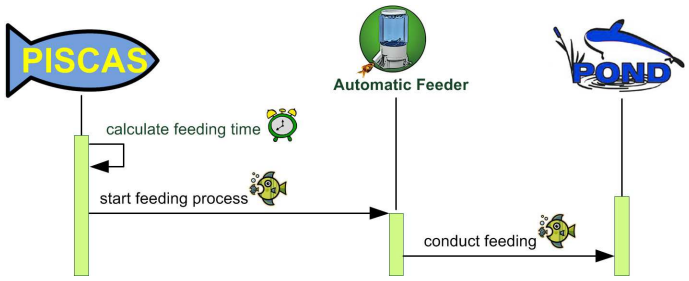
PLC software

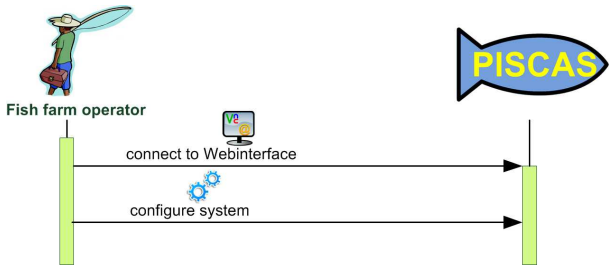
The code for the PLC including the visualization of the PLC software is generated out of a template as suggested by the template&metamodel code generation pattern presented in Section 2.3.4. This template has been developed with the aim to provide a source code which can easily be used with a code generator. This means that the variability of the domain can easily be configured with the PLC software. The code generator copies the PLC project and configures the software according to the given model. For generating the internal connections between the cyclic PLC programs, the information of the connections between the objects like ponds in the model is needed. To map the variables of the PLC software to physical inputs and outputs the model connections between PLC modules and objects line ponds are used. For the PLC programs files programmed in the language structure text and variable definition files for B&R PLCs are generated. For the visualization XML files for B&R PLCs are generated. XML files containing configuration information of the PLC, like the IP address, are generated as well.

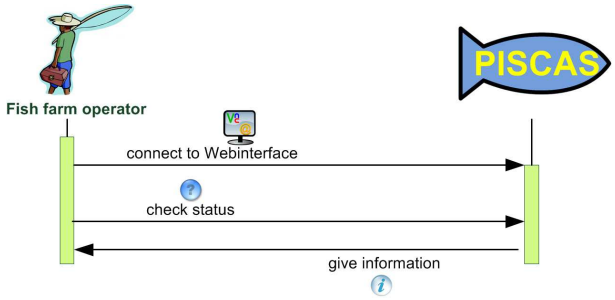
3.1.10 Use cases

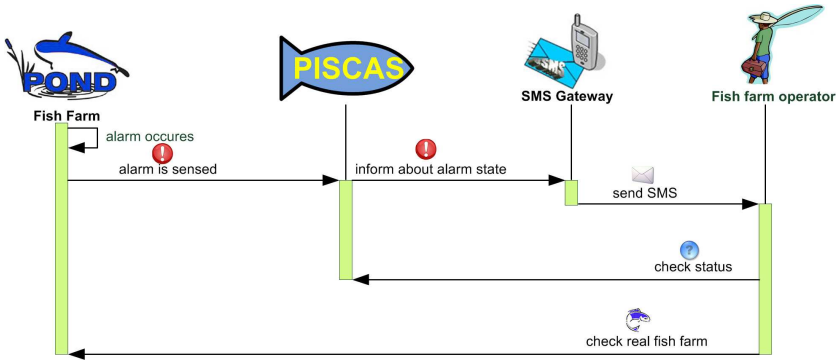
This section contains use cases worked out from the requirements gathered in Section 3.1.4. The use cases describe scenarios valid for all fish farm systems and provide a basis for the

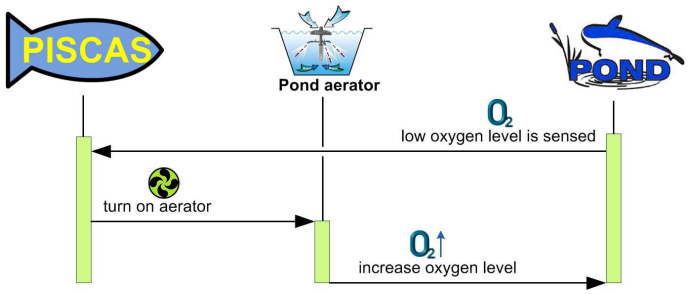
development of the generic PLC template which is described in the next section.

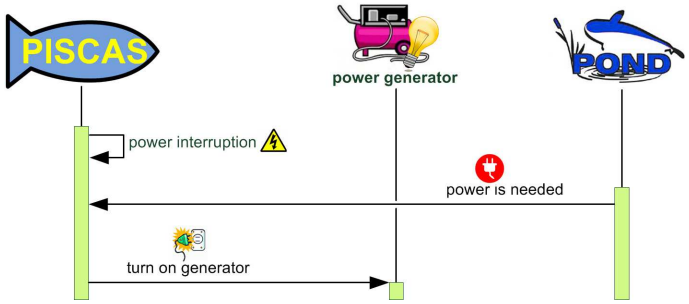
Name	**Feeding**
Description	The pond has to be fed according to the scheduler.
Scenario	A configured pond does all the feeding automatically. The scheduler of PISCAS calculates the optimal feeding times and conducts the feeding.
Sequence	

Name	**Configuration**
Description	The fish farm operator configures specific data for the fish farm.
Scenario	To configure the fish farm system the operator has to log in to the VNC webinterface. With this interface several configuration can be made. The feeding and oxygen supervision of a pond can be activated/deactivated. The amount of fish in a pond can be changed to alter the fodder amount for the pond. The type of fodder which is used can also be configured.
Sequence	

Name	Information
Description	The fish farm operator wants to get information about the current fish farm status.
Scenario	The operator connects to the webinterface and gets information about the fish farm.
Sequence	

Name	Alarm
Description	An alarm at the pisciculture system occurs.
Scenario	PISCAS sends an SMS via the SMS Gateway to the fish farm operator. The operator checks the status of the pisciculture system online and checks the fish farm if necessary.
Sequence	

Name	**Low Oxygen level**
Description	The oxygen level of a pond is low.
Scenario	PISCAS senses the oxygen level and activates the aerator if the level is too low.
Sequence	

Name	**Blackout**
Description	The power supply fails.
Scenario	If a blackout occurs the power generator is activated if power is needed (feeders or aerators have to be operated).
Sequence	

3.1.11 PLC software template

The PLC software was designed in a way that it was easy to generate different code depending on the variability of the model. This means that the software is split up in independent modules. Their behavior or amount can be changed very easily by adjusting some parameters which is done by the code generator.

The main part of the PLC software is the part for feeding and controlling the oxygen level of a pond. The feedings are scheduled in a way that all the feedings of one day

are spread across the whole day. The number of simultaneous feedings is constrained by the maximum current the power supply for the feeders can provide. For the oxygen supervision a output pin is controlled depending on the current oxygen value of a pond.

If the system reaches a critical state (e.g. low oxygen level in a pond) an SMS is sent to the fish farm operator. The PLC communicates via TCP with an SMS gateway. It is also possible to request a specific status from the PLC by sending it an SMS.

Important events are logged on the PLC. These log-files can be accessed over the Internet with an FTP-client.

Information about the used feeders like their flowrate are read from a configuration file. This file can be updated via FTP.

The software provides a visualization which allows to adjust fish farm settings and to get information about the current status. The visualization can be accessed with a VNC-Viewer.

If a blackout occurs the system checks whether power is needed (e.g. for feedings) and starts a power generator if it is necessary.

For testing the hardware of the system a test mode is available. This mode also allows to adjust settings like passwords or the maximum current for the feeding modules.

Overview

This section gives an overview of the PISCAS software design. The PLC software is divided in independent packets which can be configured by elaborating a model for a given fish farm. Fig. 3.8 gives an overview of the packages.

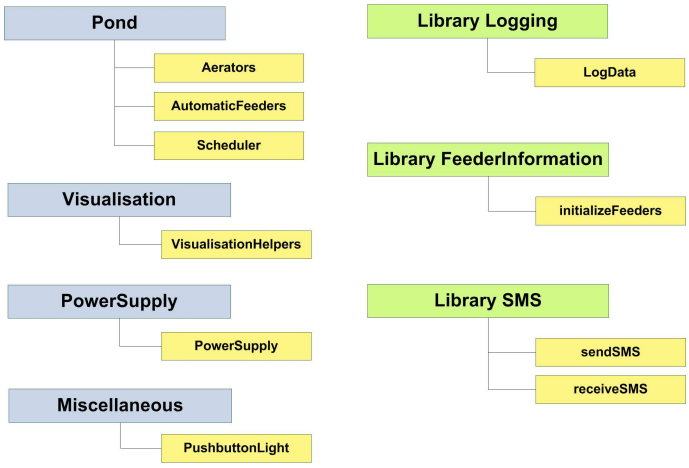


Figure 3.8: Packages of the PISCAS software

Package pond

Oxygen supervision - For each pond the oxygen level has to be supervised and an aerator has to be switched on if the oxygen level is too low. A state diagram of the according software is shown in Fig. 3.9.

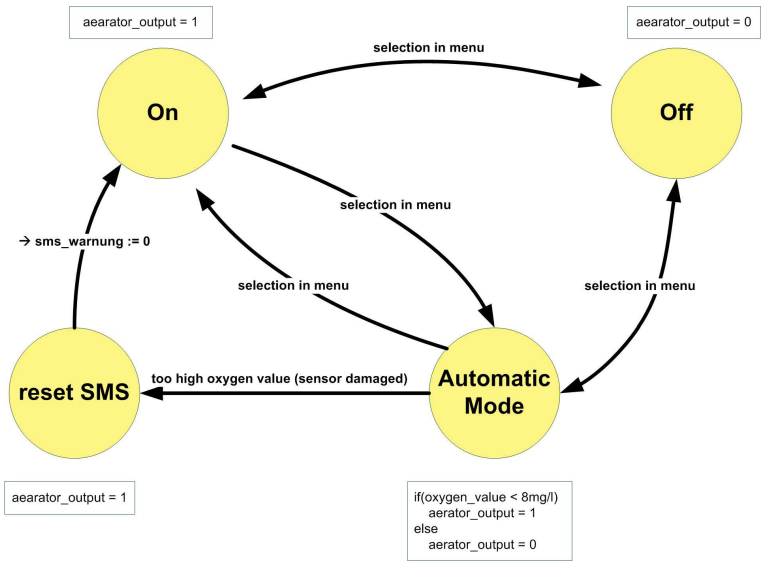


Figure 3.9: State diagram of the oxygen supervision

Feeder - Each feeder can be configured. It calculated the necessary amount of fodder according to this configuration as it is shown in Fig. 3.10.

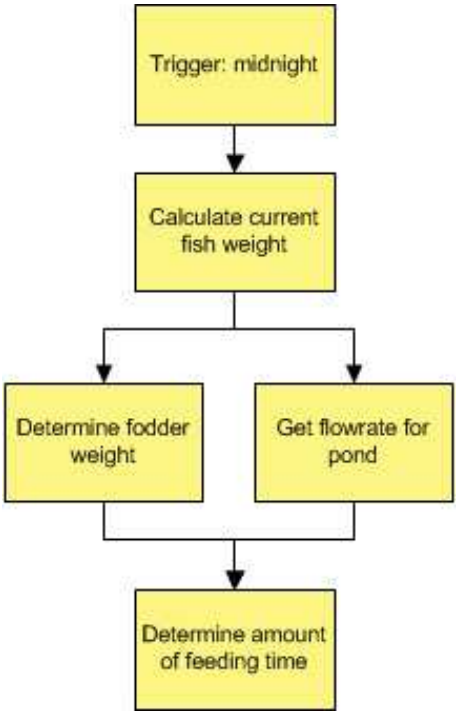


Figure 3.10: Calculation of the necessary fodder amount

Scheduler - Pseudocode for the feeding schedule is shown in Fig. 3.11. The scheduler is the most sophisticated PLC software part of the project. It assures that the feedings for each pond are spread across the whole day and that every pond feeds the needed fodder amount.

The flow rate of the feeder for each pond is determined. According to the information in a lookup table the needed fodder as percentage of the current fish weight is obtained. The lookup table depends of the fish species, the used fodder and the water temperature. Out of this data the amount of fodder per day fore each pond can be calculated. With the information of the fed fodder the fish weight of the pond can be updated and therefore data about the current fish weight in a pond is available.

Feeding-Datastructure

```
feeding_list
    feeding_units_per_day
    rest_of_feedings
    already_fed     [%]

feeding_list[0] ……..idle feeder
```

Calculate for each pond at midnight

```
feeding_list[POND_NUMBER].feeding_units_per_day := #Feeding units for Pond
feeding_list[POND_NUMBER].rest_of_feedings := feeding_list[POND_NUMBER].feeding_units_per_day
feeding_list[POND_NUMBER].already_fed := 0;
```

1min after Midnight

```
total_feedings := #All feeding units
possible_feedings := time from sunrise until sunset
IF (total_feedings > possible_feedings) THEN
  sms_warning := TRUE
END_IF
```

Scheduling (each minute during the day)

Set outputs to zero

```
FOR (i:=1; i < NUMBER_OF_PONDS; i++)
  output[i] := FALSE;
END_FOR
```

Calculate rest of feeding time

```
total_rest_of_feedings := Σ(feeding_list[POND_NUMBER].rest_of_feedings)
possible_feedings := time until sunset
idle_time := possible_feedings - total_rest_of_feedings
IF (idle_time <= 0) THEN
  sms_warning := TRUE
  feeding_list[0].already_fed := 100; (*no more feeding idle time*)
  feeding_list[0].feeding_units_per_day := 0;
ELSE
  feeding_list[0].feeding_units_per_day := idle_time;
END_IF
```

Determine next feeder

```
current_minimum := 100;
pond_to_feed := 0;
(*choose automat which had the least feeding time in %*)
FOR (i:=0; i <= NUMBER_OF_PONDS; i++)
  IF (feeding_list[i].feeding_units_per_day > 0)
    IF (feeding_list[i].already_fed < current_minimun) THEN
      current_minimum := feeding_list[i].already_fed;
      pond_to_feed := i;
    END_IF
  END_IF
END_FOR

(*conduct feeding*)
IF (current_minimum < 100) THEN
  IF (pond_to_feed > 0) THEN
    output[pond_to_feed] = TRUE;
  END_IF
  feedling_list[pond_to_feed].already_fed :=       feedling_list[pond_to_feed].already_fed +
                                                  (1 / feedling_list[pond_to_feed].feeding_units_per_day) *100;
  feedling_list[pond_to_feed].rest_of_feedings := feedling_list[pond_to_feed].rest_of_feedings – 1;
END_IF
```

Figure 3.11: Pseudocode for the scheduling

Package power supply

In case of a blackout the system is backed up by a power generator. A state diagram of this device is shown in Fig. 3.12. The power generator is just switched on of a long blackout occurs and if power is needed at the time (due to feedings or low oxygen level for a pond).

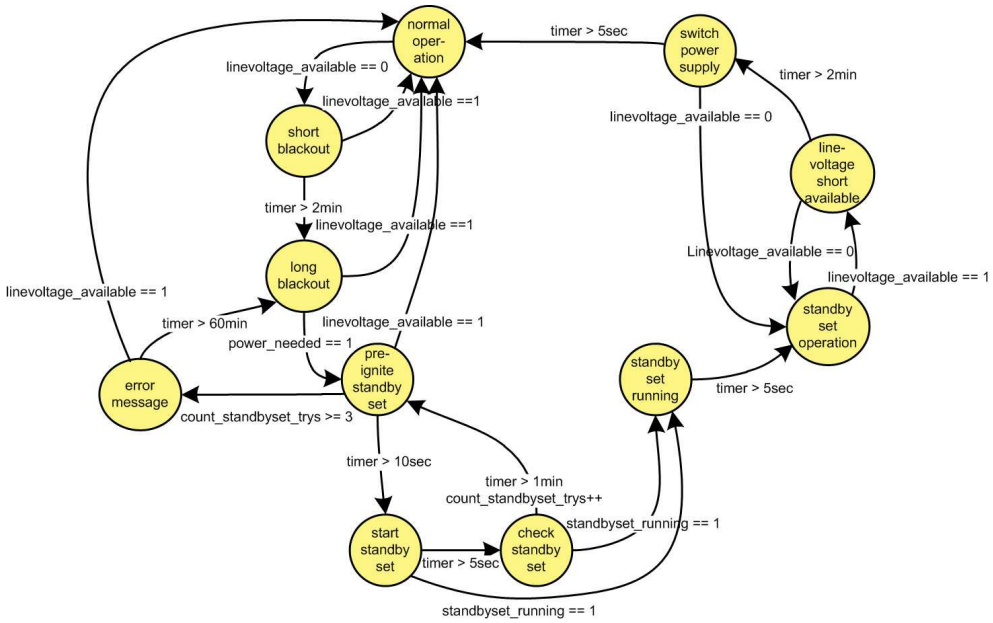


Figure 3.12: State diagram for the power supply

Package feeder information

This package reads the information for the feeders (e.g. flow rate) from a CSV file. Such a CSV file can be produced which an Excel sheet structured like the one shown in Fig. 3.13. The data is used as a lookup table to calculate the needed feeding times according to the used feeder and fodder. This calculation is done by the scheduler.

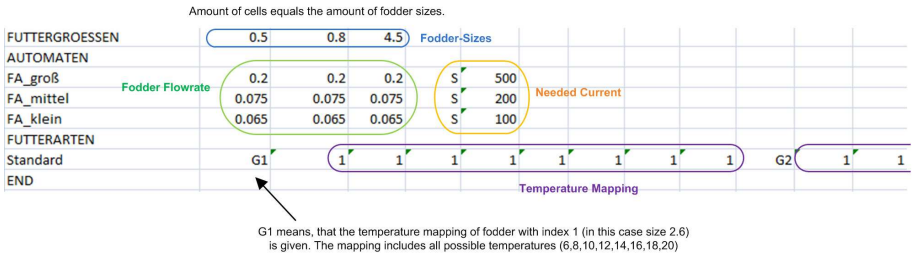


Figure 3.13: Config File for the feeder information

The values of this config file are stored in a specific feeder information data structure as it is shown in Fig. 3.14.

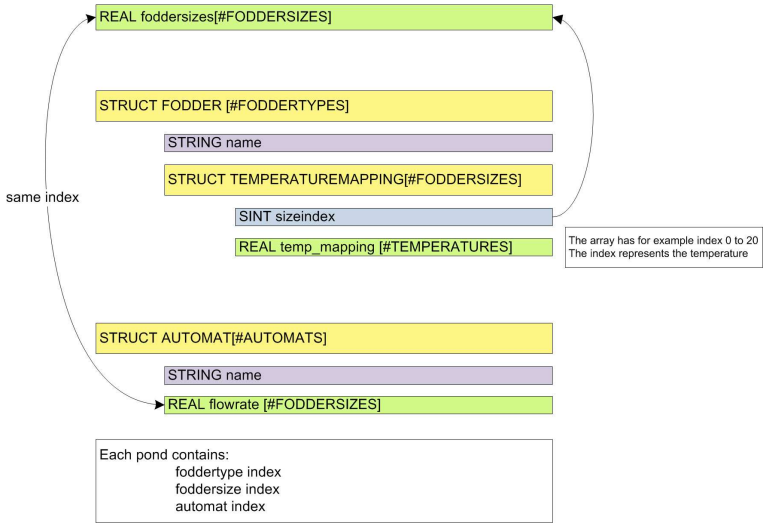


Figure 3.14: Data structure for the feeding information

Miscellaneous

This package contains little helper programs and the program for controlling switches and outputs. The state diagram for a switch is shown in Fig. 3.15. As it is possible to combine switches and outputs in the fish farm model, this section allows to model basic domain independent features.

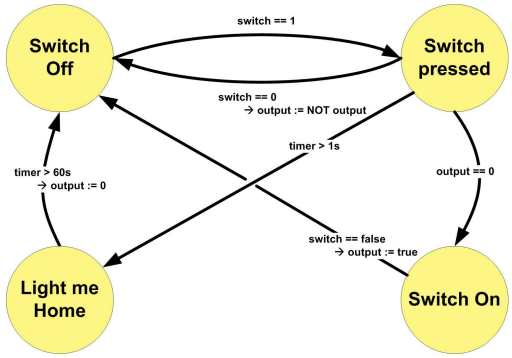


Figure 3.15: State diagram for switches

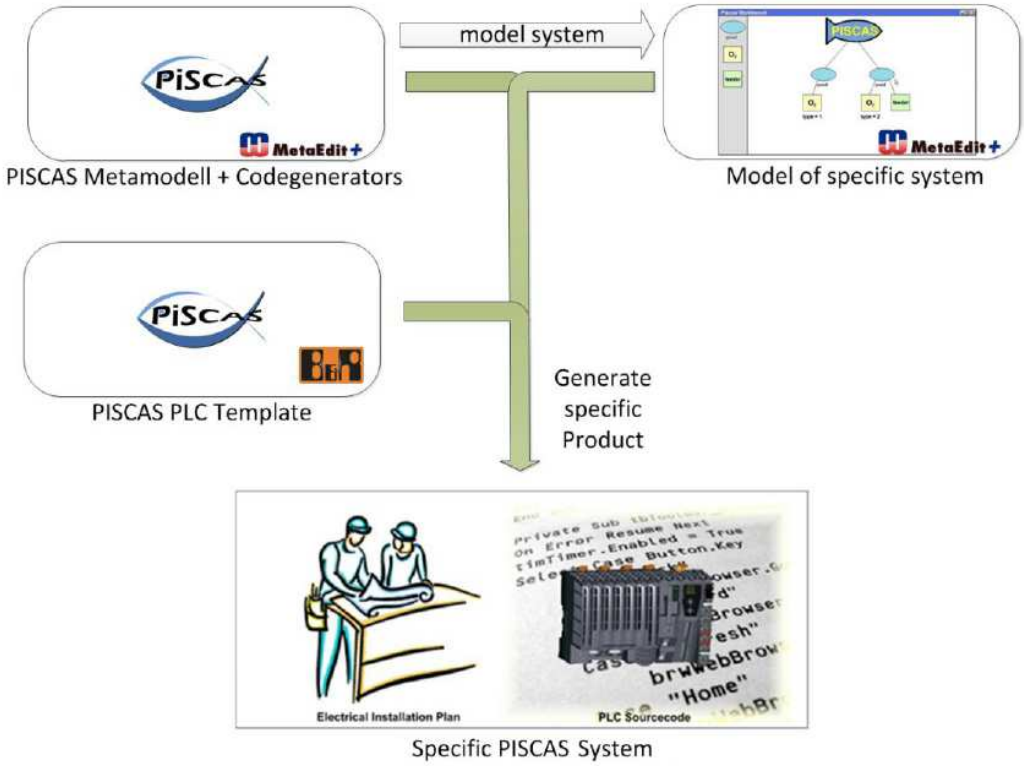


Figure 3.16: Instantiation of a new PISCAS system

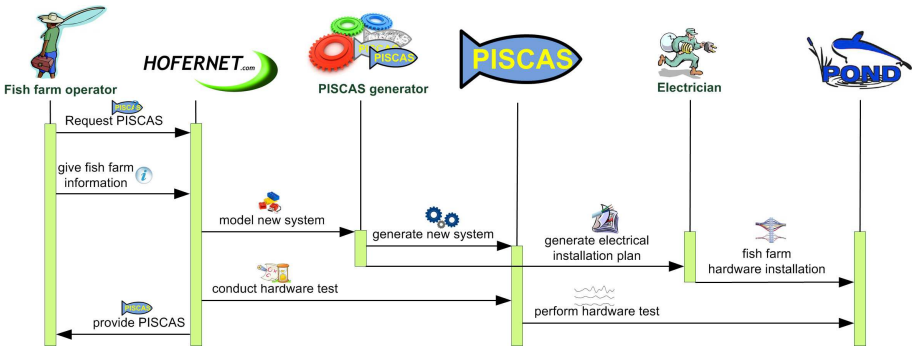
3.2 Application engineering

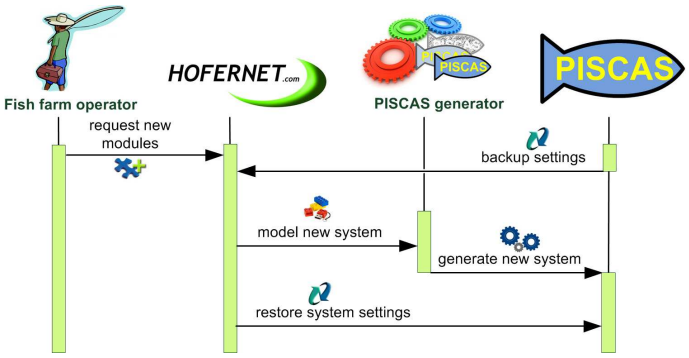
This section describes what has to be done to create a fish farm automation system with the SPL which was set up. Guided by requirements engineering from Section 2.2, the following use cases for fish farm projects were set up. These use cases cover processes that have to be done during application engineering. The following section gives an overview how a new PISCAS system is created.

3.2.1 PISCAS system instantiation

To generate a new PISCAS application the system has to be modeled with the graphical MetaEdit+ edior. This model is constrained by the designed metamdel for the fish farm domain. The code generators generate the PLC software and the documentation using the information in the model and a domain specific template. This template is the generic PLC software presented in Section 3.1.11 which includes functionality to cover the domain requirements. The process of instantiating a new system is shown in Fig. 3.16.

3.2.2 Use cases

Name	**Install a new system**
Description	A new fish farm automation system is requested
Scenario	A fish farm owner wants PISCAS as fish farm automation system. The company providing PISCAS gathers information about the fish farm and models the system. The source code and the documentation is generated out of this model. The electrician installs the hardware of the system with the aid of the generated plans.
Sequence	

Name	**New system components**
Description	New system modules have to be installed
Scenario	The system is expanded due to a request of the fish farm owner. The configuration data for the current PISCAS project is stored and backed up. A new PISCAS project with the additional modules is generated and the settings from the old project are loaded.
Sequence	

Chapter 4

Project evaluation

4.1 Project implementation

This section describes the process of installing PISCAS at two different fish farms. The systems are located at Radenthein/Austria and Feld am See/Austria.

PISCAS Prototype

A software template for a fish farm automation system was developed. The software included basic features like oxygen supervision, feeding, emergency power supply and an alarm system to satisfy the functional requirements FR1, FR3, FR8, FR9 and FR10 from 3.1.4. The prototype was developed in a way that is should be easy to adapt the software for other fish farms with different amount of ponds and slightly different functionalities.

Clone&own Radenthein

The prototype was used and changed to the fish farm system in Radenthein. The clone&own method was used to achieve that. Adapting the prototype to be used in Radenthein took much more time than expected. A detailed listing of the spent work time can be found in section 4.4. Many errors occurred during the first operation of the system and problems with the electrical installation turned up because of a lack of communication between the company Hofernet and the electrician who installed the hardware. The requirements GR1, GR2, GR3 from Section 3.1.4 were clearly missed.

PISCAS Software Product Line

To reduce the amount of work it takes to adapt a prototype to a specific fish farm system a software product line was developed. Fish farms can be modeled with a graphical editor and the source code for the automation system is generated out of this model. Additionally an installation plan for the electrician is provided by the code generator. The amount of work time for the installation of a system using SPLE is also expected to be less compared to the previous approach.

The SPL enables full code generation. 100% of the PLC software is generated and the code does not have to be altered. With the PLC code a visualization including a seperate visualization for test purposes is fully generated. This test visualization is and the fully

generated documentation of the PISCAS system bring advantage for installing PISCAS as the correctness of the hardware installation and its functionality can easliy be checked. The prodcut line was designed to meet the requirements presented in Section 3.1.4.

PISCAS Radenthein

The current software in Radenthein is generated out of a PISCAS model shown in Fig. 4.1. The software works well so far, no major problems occurred and all the requirements from Section 3.1.4 are met.

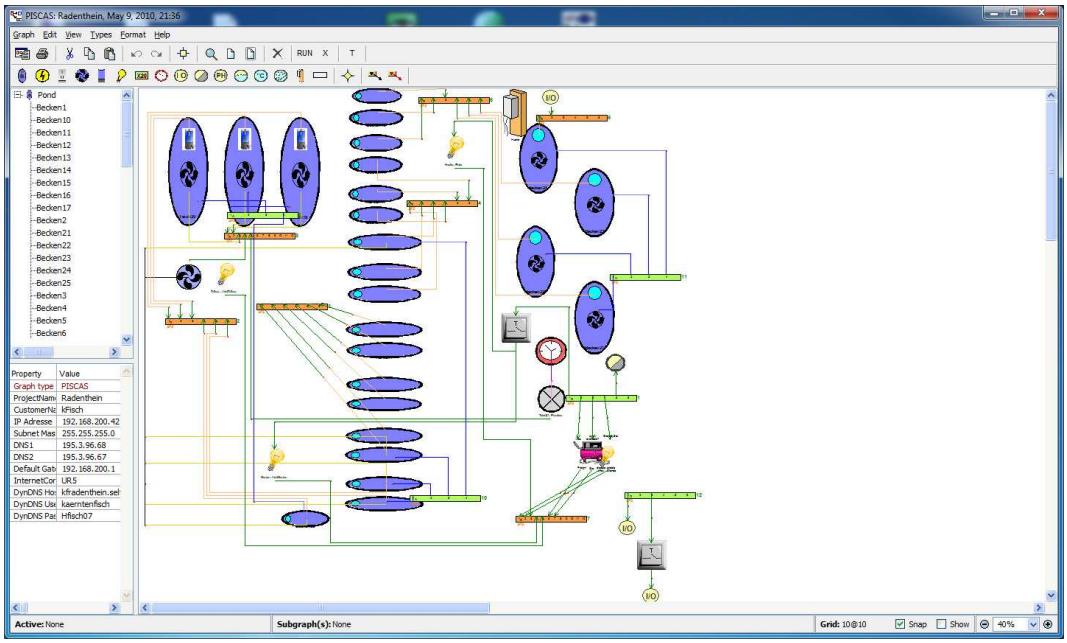


Figure 4.1: Model of the fish farm in Radenthein

PISCAS Feld am See

The fish farm in Feld am See (Fig. 4.2) has been automated using PISCAS. Modeling the fish farm with the graphical editor worked very well and did not require many changes in the software product line. The model is shown in Fig. 4.3. The automation system for this fish farm was taken into operation without big problems. The installation was conducted a lot faster than the first operation of the system in Radenthein. Detailed results can be found in Section 4.4. The requirements from Section 3.1.4 are fully met. The generated documentation is shown in Fig. 4.4 and Fig. 4.5. The visualization software is shown in Fig. 4.6.

Comparison of the two fish farm automation systems

This section shows an overview of the two fish farm systems. In Tab. 4.1 several metrics are considered to compare the size of the two systems.

Figure 4.2: Fish farm in Feld am See

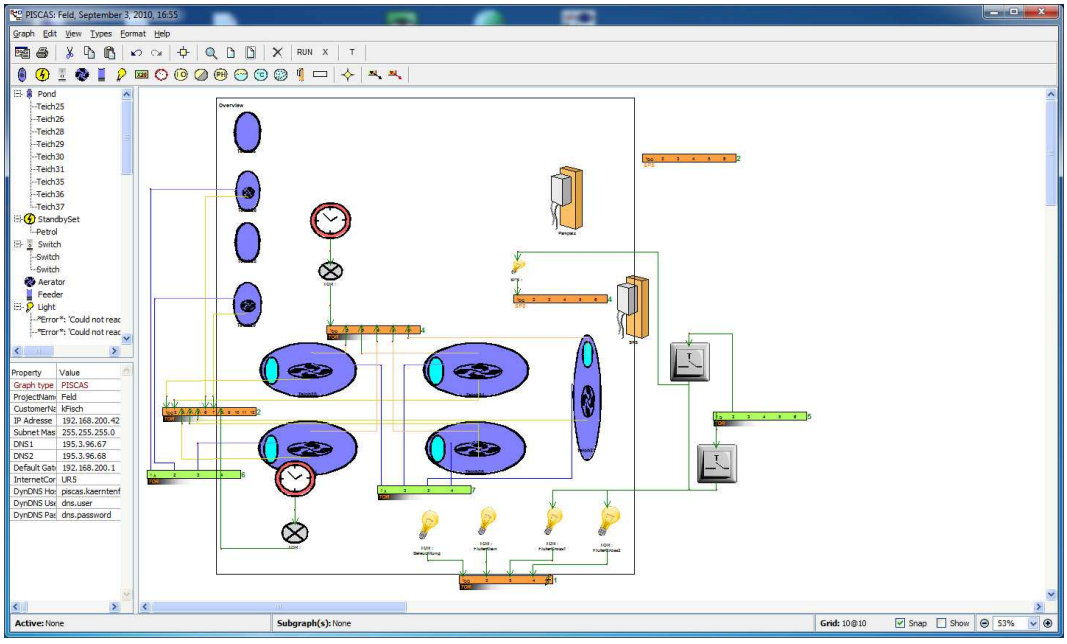


Figure 4.3: Model of the fish farm system Feld am See

4.2 Software quality

Due to the SPL approach an improvement in software quality is expected. This should be the case because the code generator is reviewed each time an error for a PLC software occurs. In practice it is often the case that several errors occur when existing software is adapted for a similar project. This method is called clone&own. Such errors do not emerge if a code generator tool is used.

As higher software quality is expected, also lower maintenance times and costs are expected. The PLC software was designed to be easily maintainable and to keep mainte-

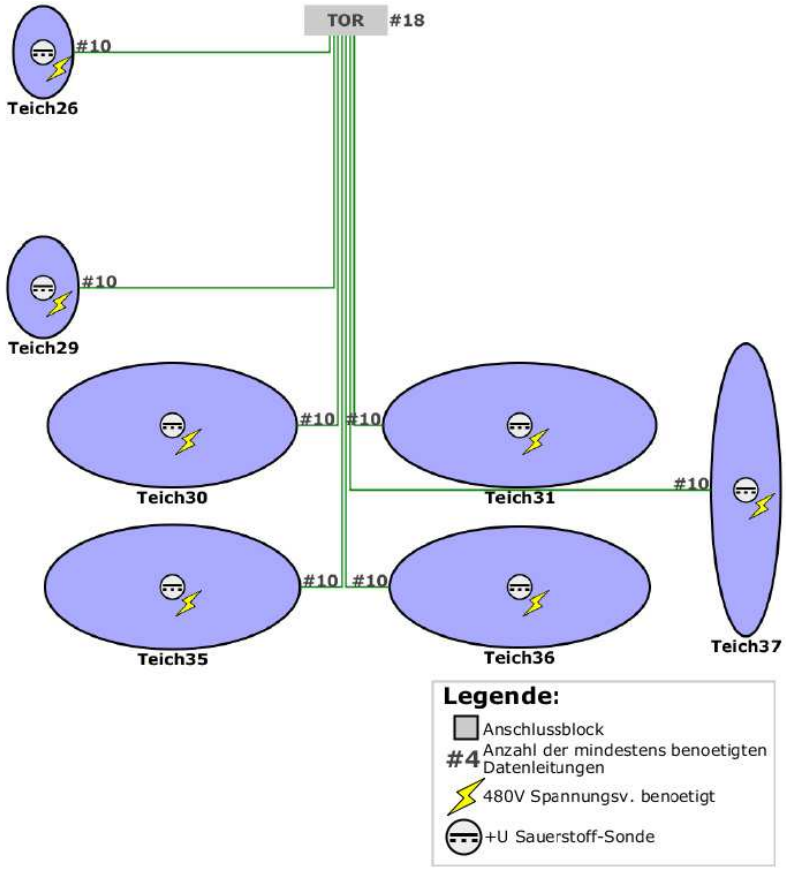


Figure 4.4: Overview part of the documentation of the fish farm system Feld am See

nance costs low. Section 4.4 shows that the expected maintenance time of the SPL system are lower than the times for a standard PLC software system.

4.3 Time to market

The production time for a fish farm automation system is decreased significantly. The data in Section 4.4 shows that developing a PLC project with respect to SPLE allows to produce these systems a lot faster if enough systems of the same family are installed.

4.4 Development approach evaluation

The data for the evaluation was acquired during the development the two fish farm systems in Radenthein and Feld am See. The systems are comparable in size and were partly developed with different methods as stated in 4.1. Therefore the clone%own and the SPL approach can be evaluated.

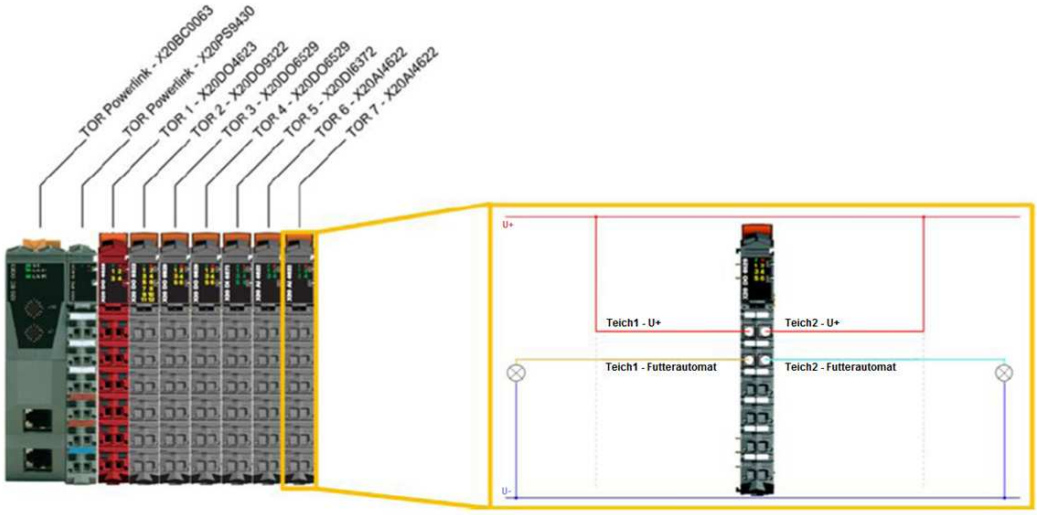


Figure 4.5: Hardware part of the documentation of the fish farm system Feld am See

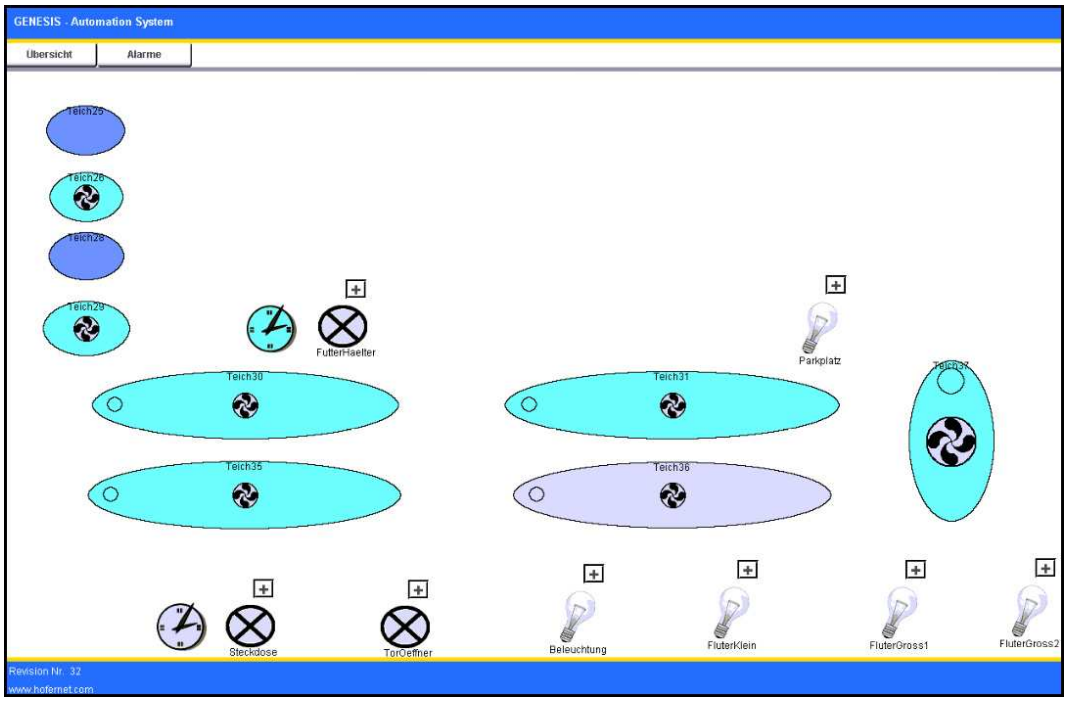


Figure 4.6: Visualization of the PLC software for the fish farm system Feld am See

4.4.1 The cost model

The cost model is based on the model proposed in [CMC05] and adapted to the needs of automation systems. Three scenarios are important for our estimation:

	Radenthein	Feld
PLC software LOC	4711	4394
Number of documentation pages	18	19
Number of ponds	25	9
Number of peripheral components (e.g. lights)	6	9

Table 4.1: Comparison of fish farm systems Radenthein and Feld am See

1. Building n products with individual project development

$$\sum_{i=1}^{n} C_{unique}(p_i) + C_{prod}(p_i) + C_{install}(p_i) + C_{maint}(p_i)$$

2. Building n products with clone&own

$$C_{cab}() + \sum_{i=1}^{n} C_{unique}(p_i) + C_{prod}(p_i) + C_{install}(p_i) + C_{maint}(p_i)$$

3. Building a SPL with n products

$$Cost_{DE} = C_{org}() + C_{cab}()$$

$$Cost_{AE} = (C_{unique}(p_i) + C_{reuse}(p_i) + C_{install}(p_i) + C_{maint}(p_i))$$

$$CostsSPL = Cost_{DE} + \sum_{i=1}^{n} Cost_{AE}(i)$$

For our project we can assume the following:

- C_{org} = Code generator development
- C_{cab} = DSL and PLC template development
- C_{unique} = Requirements analysis
- C_{reuse} = Creation of a product specific model
- C_{prod} = Manual development (without PL)
  This includes PLC Project SW development, SW changes and electrical installation changes.

  Additional to these costs, we introduce two more types of costs which are relevant for automation system development:

- $C_{install}$ = Electrical and on-site installation
- C_{maint} = Maintenance

Work hours						
Tasks	**Ind. Projects**		**Clone&Own**		**SPL**	
	base	**variable**	**base**	**variable**	**base**	**variable**
Domain engineering (DSL dev.)					22.5	
PLC project SW dev.		196				
PLC template development			180		196	
Code generator dev.					100	
Requirements analysis		20		20		20
Product specific model						2
Software adaptations				16		
Electrical installation		40		40		40
Electrical installation changes		5		5		
On-site installation		40		40		18
Maintenance		8.5		8.5		2
Sum	**0**	**309.5**	**180**	**129.5**	**331.5**	**87**

Table 4.2: Summary of work hours for different development approaches for PISCAS systems

4.4.2 Cost analysis

For the three different development methods data has been gathered during the development of PISCAS. Basically the costs can be divided into base costs and variable costs. The base costs accrue just for the first system developed in the domain. Further systems only cause variable costs. Table 4.2 shows the costs for the different methods, based on the cost model illustrated in Fig. 3.4.

In the following discussion the work hours for several systems of the same product family are examined. The costs are calculated according to the formulas given in Section 4.4.1. The costs for the different development approaches are shown in Fig. 4.7. Table 4.3 shows detailed data about the costs for different development methods subject to the number of elaborated automation system projects.

4.4.3 Break even point estimation

Using the cost values in Table 4.3, a break even point for the different approaches can be identified. We are interested in the break even point of the SPL approach and the clone&own method. It can be calculated by setting the Clone&Own costs $C_{C\&O}$ equal to the SPL costs C_{SPL}. The break even point is then given as

$$n = \frac{C_{Clone\&Own-base} - C_{SPL-base}}{C_{SPL-variable} - C_{Clone\&Own-variable}} = 3.14$$

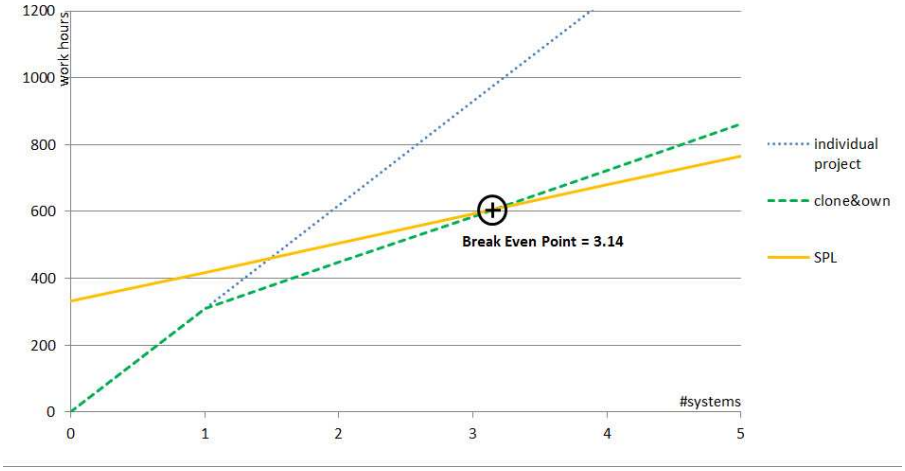


Figure 4.7: Work hours for different development approaches

#Pr.	Ind. Projects			Clone&Own			SPL		
	base	var.	sum	base	var.	sum	base	var.	sum
0	0	0	**0**	0	0	**0**	331.5	0	**331.5**
1	0	309.5	**309.5**	0	309.5	**309.5**	331.5	87	**418.5**
2	0	619	**619**	309.5	138	**447.5**	331.5	174	**505.5**
3	0	928.5	**928.5**	309.5	276	**585.5**	331.5	261	**592.5**
4	0	1238	**1238**	309.5	414	**723.5**	331.5	348	**679.5**
5	0	1547.5	**1547.5**	309.5	552	**861.5**	331.5	435	**766.5**

Table 4.3: Overview of the work hours for different development methods

This result is better than expected. The reason is, that the variable amount of work hours for AS usually are very high. A lot of variable costs during the on-site and electrical installation can be saved with the SPL approach.

4.5 Achieved goals

To test whether the goals of the project are achieved, all the requirements were tested. The weight and the rating which states how good this goal is fulfilled were discussed with the software vendor of the automation system and a fish farm owner who uses the system. As it can be seen, all of the important goal are fulfilled to a satisfying degree.

	PISCAS
GR1 - Reduction of PLC software errors	8
GR2 - Reducing time-to-market	10
GR3 - Generate software which does not have to be adapted	3
GR4 - Standardized hardware	10
GR5 - Model integrity checks	10
FR1 - SMS for alarm states	10
FR2 - Variability in hardware modules	8
FR3 - Power supply	10
FR4 - Test mode	10
FR5 - Logging	10
FR6 - Backup settings functionality	10
FR7 - Internet maintenance	10
FR8 - Fish growth model	7
FR9 - Oxygen level control	10
FR10 - Fair feeding (scheduling)	10
FR11 - Documentation	10
FR12 - Measure Water Level	10
FR13 - Selectable fish species	0
NR1 - User-friendly modeling interface	4
NR2 - Easier operation of the fish farm	10
NR3 - Lower maintenance effort	10
NR4 - Resource efficiency	10
NR5 - Scalability	10
NR6 - User interface usability	8
NR7 - User interface appearances	0
NR8 - Reliability and availability	10
NR9 - Robustness and fault tolerance	6
NR10 - Capacity, storage	10
NR11 - Software access	10
NR12 - Easier cooperation with other firms	7
Benchmark	**84%**

Table 4.4: Goal analysis

Chapter 5

Conclusion and future work

This master thesis described the basics for software product line engineering. A specific SPLE project for the fish farm domain was conducted and presented. The goals of the master thesis were analyzed and compared to the requirements of the project.

Constructing the code generator took less time than expected. The work for refactoring the template, creating a metamodel and developing the code generators was less than the work for the development of the first prototype of the system. Also the results of this work regarding saving costs and time for fish farm automation systems constructed with the aid of the code generator were better than expected. The evaluation showed that the break even point when using the SPL approach instead of the well known clone&own method is 3.14 (Section 4.4). These results are a lot better than results of similar projects like [Has09]. The low break even point can be explained because of the few additional effort that has to be put in if a code generator is constructed and because of the huge savings of variable costs for the SPL approach. These savings follow from the installation of the automation system. The time for error tracing during this phase is reduced a lot and therefore many hours of work can be saved.

The main aims of this project were decreasing the costs for the development of fish farm projects and reducing the maintenance time. The set goals for this project were satisfactory met.

For future work the following aspects would be of interest:

Maintaining the code generator It is planned that the code generator tool will be used in future for several other fish farm systems. It is not expected that the generator can handle all of the system, but it should not be a lot of work to change the tool according to new requirements. Maintaining the code generator and adding new features and correcting errors of the PISCAS template are expected to be the biggest part of future work.

Wireless nodes The local activities for a pond like supervising the oxygen level are critical. This could be achieved by a local device for each pond. The data for configuration and the data for the visualization could be sent via a wireless link to a central station and perhaps could be integrated to an existing PLC project. Advantages would be that no cables have to be installed between the PLC and each pond. A disadvantage is that this solution includes additional individual hardware. This

hardware would be more error-prone because it is not as well tested as standardized industrial hardware.

Generating further fish farm projects At the time fish farm projects have been generated for the systems in Feld am See and Radenthein. The data gathered for the system in Radenthein is not very representative as the code generator was developed with respect to this specific fish farm systems. It is clear that it was possible to model all the features of this system. The system in Feld am See could easily be modeled with the code generator with few additional pieces of software. Data collected during the development of this system is representative, but as the size of the fish farm in Feld am See is not exactly the same as the size of the fish farm in Radenthein, it would be of interest to install more fish farms to get better and more detailed results regarding the development costs.

Developing a PLC software generator for another domain It would be interesting to get data about the development time for a similar project if the developer already has the needed knowledge to create code generators. These times could be compared to the current data shown in Section 4.4

Porting the system to be usable with other PLCs The project is designed to work with Bernecker&Rainer PLCs. Data about the needed time to change the PISCAS template and the code generator so that it can be used with another PLC would be of interest. Theoretically the template does not need a lot of changes as the source-code is written in structured text which is not vendor-specific. The code generator would have to be changes as it generates some B&R specific files concerning the hardware mapping of the PLC.

Appendix A

Evaluation of metamodeling tools

The criteria were taken from existing papers [DSF07, DRGN07, LCP+00, Lei09, Has09].

Nr.	Critertion	Definition
Product Line Engineering criteria		
1	Attribute management	• Differentiate between SPL requirements and product requirements • Manage requirements attributes (identifier, description, justification, cost,...) • Ability to capture future requirements • Ability to capture new requirements during derivation • Autobuild with given specifications (mining)
2	Feature and variability modeling	• Help to model FODA-like concepts (feature decomposition, feature type, cardinalities, dependency links,...) • Support different abstraction levels • Support global constraints
3	Feature Metamodel maturity	• Allow to define a PL metamodel • The tool should be unambiguous • Support product line evolution

Nr.	Critertion	Definition
4	Constraint checking and propagation	• Support validation checking for the PL model and metamodel • Check consistency of product model and PL model • Check consistency of model and artefact base • Support constraint propagation • Compare artefacts to a 'standard' • Rule-checking
5	Product derivation	• Help to derive specific products with guidance and visualization
6	Domain engineering management	• Support the creation of domain artefacts • Support the management of domain artefacts • Map domain artefacts to corresponding features • Search functions, to find a suitable artefacts
7	Repository	• Version-management of artefacts, documents or possibility to integrate such a tool • Re-create any version of a product • Compare different versions of a product
Management criteria		
8	Traceability management	• Support requirements traceability with external documents • Support traceability management of inter-requirements links • Support metamodel traceability • Traceability between and within assets (linkage ...)

Nr.	Critertion	Definition
9	Impact analysis	• Perform impact analysis when changing requirements or models • Perform impact analysis when changing interlink requirements
10	Reporting	• Ability to generate reports
Technical criteria		
11	Access mode	• Allow multi-user access • Allow access with profiles (define the metamodel / use it)
12	Technical environment	• Support synchronization • Interoperability: support import and export from other tools (APIs, neutral format files, etc.)
13	Usability	• Intuitive usage • Stability and efficient support • Offer high accessibility of functions, zoom, views, ... • Ability to handle great amount of artefacts
14	Automatic filters	• Automatic filters on requirements presentations and report generation
15	Tool configuration	• The tool should be configurable for specific user needs • Adaption to current organisation
16	Extensibility	• Should be extensible to integrate existing platforms into the PL
17	Flexibility	• Changes should be possible on each stage of development (also in derived products).

Nr.	Critertion	Definition
18	AOB	• Tool costs and training costs /amortisation time • Light charge of installation, maintenance and migration cost • Flexible licensing service

Table A.1: Criteria to rate SPL tools

Appendix B

Questionnaires

B.1 Questionnaire for fish farm owners

The following questionnaire was given to the fish farm owner *Andreas Hofer* who owns the company *Kärnten Fisch*. The prototype of PISCAS was developed for him and several other systems for his fish farms will be developed. Answers are marked with blue color. The questionnaire was conducted before PISCAS was installed.

How reliable do current fish farm automation systems (feeding automates, oxygen supervision) work?

- very reliable
- reliable
- little reliable
- unreliable

How efficient do current fish farm automation systems work (fish food efficiency, low-energy use)?

- very efficient
- efficient
- little efficient
- not efficient

Are fish farm systems often changed or extended (e.g. new feeding automates)?

- very often
- often
- seldom
- very seldom

How good is the support of fish farm automation system vendors?

- very good

good
poor
- very poor

How important are the following functionalities?

	oxygen supervision	water level supervision	efficient feeding	emergency power unit
very important	x	x	x	x
important				
little important				
unimportant				

How important are the following criteria for an automation system?

	cheap system costs	cheap maintenance costs	quick installation	quick correction of errors
very important	x	x	x	x
important				
little important				
unimportant				

B.2 Questionnaire for electrician companies

The following questionnaire was given to two companies working in the automation and electrical installation sector. *EVA GmbH* is a firm which focuses on industrial automation systems. The company installs the hardware and programs the software for those systems. *Elektro Tisch* focuses on the electrical installation of systems.

The answers of the companies are marked with colored bullets or colored text. Green color represents the answers of the firm *EVA GmbH*, orange bullets represent the answers of the company *Elektro Tisch*.

How often do problems arise due to lack of communication with the customer?

very often
- ● often
seldom
very seldom

Is the communication between the customer and the company standardized?

- Yes
- No

The customer provides rudimentary installation plans

- very often
- often
- seldom
- very seldom

A rudimentary installation plan provided by the customer is

- very desirable
- desirable
- little desirable
- undesired

An detailed installation plan provided by the customer is

- very desirable
- desirable
- little desirable
- undesired

Following methods are used to obtain the requirements of the customer and to get the necessary information for the electrical installations

- Regular meetings with the customer
- Regular meetings with the customer
- List of components which have to be installed provided by the customer

The possibility to test a hardware installation with the automation system software is

- very desirable
- desirable
- little desirable
- undesired

Bibliography

[AFR06] Daniel Amyot, Hanna Farah, and Jean-François Roy. Evaluation of development tools for domain-specific modeling languages. In *SAM*, pages 183–197, 2006.

[BBMY04] Barry Boehm, A. Winsor Brown, Ray Madachy, and Ye Yang. A software product line life cycle cost estimation model. *Empirical Software Engineering, International Symposium on*, 0:156–164, 2004.

[BCC+96] L. Baker, P. Clemente, B. Cohen, L. Permenter, B. Purves, and P. Salmon. Foundational concepts for model driven system design. INCOSE Model Driven System Design Interest Group, 1996.

[Ben03] Michael Benz. Einführung in die generative Programmierung (GP), 2003.

[BFG+02] A. Bertolino, A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Use case description of requirements for product lines. In *Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 - REPL 02. Technical Report: ALR2002-033, AVAYA*, pages 12–18, 2002.

[BGK+06] Krishnakumar Balasubramanian, Aniruddha Gokhale, Gabor Karsai, Janos Sztipanovits, and Sandeep Neema. Developing applications using model-driven design environments. *Computer*, 39:33–40, 2006.

[Bru07] Nicolas Brugger. Vergleich von Modellierungssprachen für MDA, 2007.

[CJNM05] Paul C. Clements, Lawrence G. Jones, Linda M. Northrop, and John D. McGregor. Project management in a software product line organization. *IEEE Softw.*, 22(5):54–62, 2005.

[Cla01] M. Clauss. Modeling variability with UML. In *GCSE 2001 Young*, 2001.

[CMC05] Paul C. Clements, John D. McGregor, and Sholom G. Cohen. The structured intuitive model for product line economics (SIMPLE). Technical report, Software Engineering Institute at Carnegie Mellon University, February 2005.

[CN01] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[CRR09] Lan Cao, Balasubramaniam Ramesh, and Matti Rossi. Are domain-specific models easier to maintain than UML models? *IEEE Softw.*, 26(4):19–21, 2009.

[CY08] Jordi Cabot and Eric Yu. Improving requirements specifications in model-driven development processes, 2008.

[DLRS09] Mike Deimler, Zhenya Lindgardt, Martin Reeves, and George Stalk. Business model innovation: When the game gets tough, change the game. Technical report, The Boston Consulting Group, 2009.

[DRGN07] Deepak Dhungana, Rick Rabiser, Paul Grünbacher, and Thomas Neumayer. Integrated tool support for software product line engineering. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 533–534, New York, NY, USA, 2007. ACM.

[DSF07] O. Djebbi, C. Salinesi, and G. Fanmuy. Industry survey of product lines management tools: Requirements, qualities and open issues. *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 301–306, Oct. 2007.

[Fro03] Geoff Frost. Automatic code generation for safety critical systems. Tarragon Embedded Technology, 2003.

[FVLD03] Dries Faems, Bart Van Looy, and Koenraad Debackere. The role of inter-organizational collaboration within innovation strategies: Towards a portfolio approach. Open access publications from katholieke universiteit leuven, Katholieke Universiteit Leuven, 2003.

[Ham08] James L. Hammond. Improving productivity and quality with domain-specific modeling. *Embedded Systems Design Europe*, pages 20–23, 2008.

[Has09] Andreas Haselsberger. Design and implementation of a domain specific architecture for programmable logic controllers. Master's thesis, Graz University of Technology, 2009.

[Jun08] Martin Jung. Codegeneratoren: Domänenspezifische Automatisierung in der Praxis industrieller Softwareentwicklung. Develop Group, 2008.

[KAK08] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 311–320, New York, NY, USA, 2008. ACM.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[KE02] Chethana Kuloor and Armin Eberlein. Aspect-oriented requirements engineering for software product lines. In *10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, page 98, 2002.

[KKL+98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Gerard Jounghyun Kim, and Euiseob Shin. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[Kru95] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12:42–50, 1995.

[LCP+00] Bass L., Clements, P., Donohoe, P., McGregor, J., and Northrop. Fourth product line practice workshop report. Technical Report CMU/SEI-2000-TR-002 (ESC-TR-2000-002), Software Engineering Institute. Carnegie Mellon University, 2000.

[Lei09] Andrea Leitner. A software product line for a business process oriented IT landscape. Master's thesis, Graz University of Technology, 2009.

[Loh07] Christoph Lohe. Entwicklung von domänenspezifischen Modellierungssprachen, 2007.

[Mac96] L. A. Macaulay. *Requirements Engineering*. Springer-Verlag, 1996.

[Mat04] Mari Matinlassi. Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 127–136, Washington, DC, USA, 2004. IEEE Computer Society.

[Met09] Metacase. Domain-Specific Modeling whith MetaEdit+: 10 times faster than UML. White Paper, 2009.

[Mot07] Motorola. SCADA systems. White Paper, 2007.

[MP07] Andreas Metzger and Klaus Pohl. Variability management in software product line engineering. In *ICSE Companion*, pages 186–187, 2007.

[NCP09] Oksana Nikiforova, Antons Cernickins, and Natalja Pavlova. Discussing the difference between model driven architecture and model driven development in the context of supporting tools. *Software Engineering Advances, International Conference on*, 0:446–451, 2009.

[Ngu09] Quyen L. Nguyen. Non-functional requirements analysis modeling for software product lines. In *MISE '09: Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*, pages 56–61, Washington, DC, USA, 2009. IEEE Computer Society.

[NK04] Oksana Nikiforova and Marite Kirikova. Two-hemisphere model driven approach: Engineering based software development. In *CAiSE*, pages 219–233, 2004.

[OMA+00] Henk Obbink, Jürgen Müller, Pierre America, Rob van Ommering, Gerrit Muller, William Van Der Sterren, and Jan Gerben Wijnstra. A component-oriented platform architecting method for families of software-intensive electronic products, 2000.

[OMG04] OMG. Unified modeling language (UML) 2.0, 2004.

[PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[PL05] Prasanna Padmanabhan and Robyn R. Lutz. Tool-supported verification of product line requirements. *Automated Software Engg.*, 12(4):447–465, 2005.

[RJ96] Don Roberts and Ralph Johnson. Evolve frameworks into domain-specific languages. In *Proc. 3rd Intl Conf.*, 1996.

[RR] James Robertson and Suzanne Robertson. Volere requirements specification template, http://www.volere.co.uk/ (21.06.2010).

[Sch97] Christian Schwarz. Zwischenbetriebliches Workflowmanagement im World Wide Web. Technical report, Europa-Universität Viadrina, 1997.

[Sel03] Bran Selic. The pragmatics of model-driven development. *IEEE Softw.*, 20(5):19–25, 2003.

[Siv08] Sanna Sivonen. Domain-specific modelling language and code generator for developing repository-based Eclipse plug-ins. *VTT publication 680*, 2008.

[Sla06] Stefan Slapeta. Ein Vergleich zwischen Visual Studio 2005 und Eclipse Graphical Modeling Framework zur Unterstützung von modellgetriebener Softwareentwicklung. Master's thesis, TU Wien, 2006.

[SSV08] Thomas Strasser, Christoph Sünder, and Antonio Valentini. Model-driven embedded systems design environment for the industrial automation sector. *IEEE international conference on industrial informatics 2008*, 2008.

[Sta02] Patrick Stahler. *Geschäftsmodelle in der digitalen Ökonomie.* Josef Eul Verlag, 2002.

[TK09] Juha-Pekka Tolvanen and Steven Kelly. Metaedit+: defining and using integrated domain-specific modeling languages. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 819–820, New York, NY, USA, 2009. ACM.

[Tri03] Heymans Trigaux. Modeling variability requirements in software product lines: A comparative survey. Technical report, Institut d'Informatique FUNDP, 2003.

[Tru06] Frank Truyen. The fast guide to model driven architecture. White Paper, 2006.

[VB04] Markus Voelter and Jorn Bettin. Patterns for model-driven software-development, 2004.

[vdML02] T. von der Maen and H. Lichter. Modeling variability by UML use case diagrams. In *Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002*, 2002.

[vL04] Tammo van Lessen. Generatives programmieren, 2004.

[Voe03] Markus Voelter. A catalog of patterns for program generation. Technical report, EuroPloP2003, Eighth European Conference on Pattern Languages of Programs, 2003.

[Wal10] Philipp Wallner. Moderne Methoden schaffen strategische Vorteile, http://www.polyscope.ch/dlcenter/ps/2010_1/ps1_2_s24_26.pdf (21.06.2010), 2010.

[Wat01] Bob Waterbury. DCS, PLC, PC, or PAS? *ICONICS Control Magazine*, July 2001.

[WHG+09] Jules White, James H. Hill, Jeff Gray, Sumant Tambe, Aniruddha S. Gokhale, and Douglas C. Schmidt. Improving domain-specific language reuse with software product line techniques. *IEEE Softw.*, 26(4):47–53, 2009.

[Wim05] Manuel Wimmer. Model Driven Architecture in der Praxis - Evaluierung aktueller Entwicklungswerkzeuge und Fallstudie. Master's thesis, TU Wien, 2005.

[Zdu05] U. Zdun. Concepts for model-driven design and evolution of domain-specific languages. In *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, San Diego, CA, USA, January 2005.