Technische Universität München

Department of Mechanical Engineering

Institute of Automation and Information Systems

Prof. Dr.-Ing. Birgit Vogel-Heuser

Bachelor Thesis

**Automated Generation of Modular and Dynamic Industrial Process Plant Visualizations in a Manufacturing Execution System (MES)**



|  |  |
| --- | --- |
| Author: | Miguel Romero Karam |
| Matriculation No.: | 03675217 |
| Advisor: | Dr.-Ing. Daniel Schütz |
| Supervisor: | Emanuel Trunzer, M.Sc. |
| Begin Date: | 15 April 2018 |
| Submission Date: | 15 Oktober 2018 |

Statutory Declaration

I hereby confirm to have written the present dissertation independently and only with the use of the sources and resources I have indicated. Both content and literal content were identified as such. The work has not been available in this or similar form to any other panel of examiners.



Date: Signature:

Abstract

…

Kurzzusammenfassung

…

Index

[Statutory Declaration III](#_Toc523327168)

[Abstract V](#_Toc523327169)

[Index VII](#_Toc523327170)

[1 Introduction 1](#_Toc523327171)

[1.1 Overview and Motivation 1](#_Toc523327172)

[1.2 Problem Definition 2](#_Toc523327173)

[1.3 Initial Situation 3](#_Toc523327174)

[1.4 Goals of the Bachelor Thesis 4](#_Toc523327175)

[1.5 Project Requirements 5](#_Toc523327176)

[1.6 Composition of the Bachelor Thesis 8](#_Toc523327177)

[1.6.1 Project Management 8](#_Toc523327178)

[1.6.2 Project Sprints 8](#_Toc523327179)

[2 Technological Standpoint 2](#_Toc523327180)

[2.1 Industrial Process Control 2](#_Toc523327181)

[2.1.1 Overview 2](#_Toc523327182)

[2.1.2 Plant Hierarchy Model (ISA-95) and ISA-88 3](#_Toc523327183)

[2.1.3 Piping and Instrumentation Diagram (P&ID) 4](#_Toc523327184)

[2.2 Manufacturing Execution Systems 4](#_Toc523327185)

[2.2.1 Functions 4](#_Toc523327186)

[2.2.2 MES in Context of the 4th Industrial Revolution 4](#_Toc523327187)

[2.2.3 Overview of Legato Sapient® 4](#_Toc523327188)

[2.3 Web Applications 5](#_Toc523327189)

[2.3.1 Overview 5](#_Toc523327190)

[2.3.2 List of Technologies 6](#_Toc523327191)

[2.4 Unified Modelling Language (UML) 9](#_Toc523327192)

[2.4.1 Class Diagram 10](#_Toc523327193)

[2.4.2 Object Diagram 10](#_Toc523327194)

[2.4.3 Entity Relationship Diagram 11](#_Toc523327195)

[2.4.4 Activity Diagram 11](#_Toc523327196)

[2.5 Graphing Algorithms 11](#_Toc523327197)

[2.5.1 Graph Theory 11](#_Toc523327198)

[2.5.2 Types of Graphs 12](#_Toc523327199)

[2.5.3 Implementation of Graph Visualizations in Software 13](#_Toc523327200)

[2.5.4 Graph Layout Algorithms 14](#_Toc523327201)

[2.5.5 Graphs and P&IDs 15](#_Toc523327202)

[2.6 Related Works 15](#_Toc523327203)

[2.6.1 Overview of Related Works 15](#_Toc523327204)

[2.6.2 Comparison of Related Works 16](#_Toc523327205)

[2.6.3 17](#_Toc523327206)

[3 P&ID Shapes Library 19](#_Toc523327207)

[3.1 Introduction 19](#_Toc523327208)

[3.2 Graph Theory 20](#_Toc523327209)

[3.3 mxGraph API 20](#_Toc523327210)

[3.3.1 Core Architecture 20](#_Toc523327211)

[3.3.2 Technologies 21](#_Toc523327212)

[3.3.3 Schema 21](#_Toc523327213)

[4 Legato Sapient® Boardlet 23](#_Toc523327214)

[4.1 Overview of Software Architecture 23](#_Toc523327215)

[4.2 Design 24](#_Toc523327216)

[4.2.1 User Interface (UI) 25](#_Toc523327217)

[4.2.2 Presentation Logic 25](#_Toc523327218)

[4.3 Business Logic 25](#_Toc523327219)

[4.3.1 Overview 25](#_Toc523327220)

[4.3.2 Database Queries 26](#_Toc523327221)

[4.3.3 Object Relational Data Mapping 26](#_Toc523327222)

[4.3.4 Graph Layout Algorithm 27](#_Toc523327223)

[SEE NEXT CHAPTER 27](#_Toc523327224)

[4.3.5 Generation of the XML File 27](#_Toc523327225)

[5 P&ID Graphing Algorithm 29](#_Toc523327226)

[5.1 Build Hierarchy 29](#_Toc523327227)

[5.2 Hierarchy Traversal 29](#_Toc523327228)

[5.3 Simplification of Edges 29](#_Toc523327229)

[5.4 Vertex Placement 29](#_Toc523327230)

[5.4.1 Simplifying connections in model 30](#_Toc523327231)

[6 Verification, Validation and Evaluation 36](#_Toc523327232)

[6.1 Definitions 36](#_Toc523327233)

[6.1.1 Verification 36](#_Toc523327235)

[6.1.2 Validation 36](#_Toc523327236)

[6.1.3 Evaluation 36](#_Toc523327237)

[6.2 Prototypical Implementation in an Industrial Context 36](#_Toc523327238)

[7 Synopsis XXXVIII](#_Toc523327239)

[List of Figures XL](#_Toc523327240)

[List of Tables XLII](#_Toc523327241)

[Abbreviations 44](#_Toc523327242)

[8 Bibliography 46](#_Toc523327243)

1. Introduction
   1. Overview and Motivation

Software has become indispensable in modern industrial contexts. Nevertheless, the increasing complexity of the industrial contexts themselves makes its implementation in an efficient, economical and advantageous manner more challenging now than ever, (Evoke/incite/induce)

The development of control software and visualization interfaces for the operation of smaller process engineering systems is a major cost driver in process engineering automation projects. Additional to the high up-front implementation cost for the connection and configuration of a Manufacturing Execution System (MES), operational costs rise rapidly during the MES product life cycle; Creating and later modifying plant-specific visualization interfaces represents a significant technical effort, which translates to these continued increments in operational expenditure. The MES software architecture is often deeply intertwined; Adjustments in any area usually have consequences in others, even rather simple modifications can propagate and lead to important sources of errors, imposing constant software adjustments. A slight shop floor modification, be it a physical change in the plant like the disabling of a temperature sensor or a change to the order of production, might result in significant number of adjustments for the MES. Process visualizations in the Graphical User Interface (GUI) are similarly influenced; being virtual representations of the physical process facility, they demand frequent adjustments which result in significant overhead for its implementation.

The ProcAppCom (Process Application Composer) research project behind this bachelor thesis represents a cooperation between multiple industrial partners, namely 3S-Smart Software Solutions GmbH, Gefasoft GmbH, Johann Albrecht Brautechnik GmbH and APE Engineering GmbH with the Technical University of Munich. The main objective of the ProcAppCom research project is the automated configuration and generation of control code and visualizations for production plants in the field of process engineering.

Gefasoft GmbH is a leading and innovative provider of production-related software solutions. With the product Legato Sapient® Gefasoft offers a web-based MES for the distributed control of production. Main functions of a MES are production management, supervisory control, maintenance management and the real-time data acquisition, storage and integration to other information systems. These include Enterprise Resource Planning (ERP) and Supervisory control and data acquisition (SCADA) systems as well as programmable logic controllers (PLC). A MES therefore typically spans from the operational management level, where it is implemented, through the process management (SCADA) and the control levels (PLC), unto the field layer or shop floor. Key functionalities of a MES for the process engineering industry include the cross-plant evaluation of messages, alarms, process variables ​​and key figures.

Motivation of this bachelor thesis is the development of a system for the automated generation of dynamic Piping and Instrumentation Diagram (P&ID) visualizations for industrial plants with the goal of reducing implementation and operation expenditure for a MES, so that any enterprises can profit from these software solutions.

* 1. Problem Definition

The present trends in automation technology lead to a permanent increase in the complexity of industrial process facilities and to permanent technical changes. These changes propagate through the documentation, maintenance and operation of mentioned facilities, which represents a major engineering challenge. This leads to the need for the frequent and often manual reconfiguration and adjustment of such systems during its life cycle. Plant-specific visualizations in the graphical user interface (GUI) demand significant efforts for their creation and modification upon any technical change. As virtual representations of the process facility and with the imminent increase in changes, they are subject to constant manual updating. Moreover, this constant change leads to deviation from the industry standards for visualizations, like those for P&ID visualizations. As a result, different companies from different sectors end up each with different standards, which further increases the engineering complexity and results in counterproductive GUIs. For these reasons, production software requires adapting to these demanding trends in the process engineering industry. With respect to process visualizations in the process engineering industry, a system must be developed for the automated generation of modular and dynamic plant P&ID visualizations with minimal user configuration and integration to the MES software at hand.

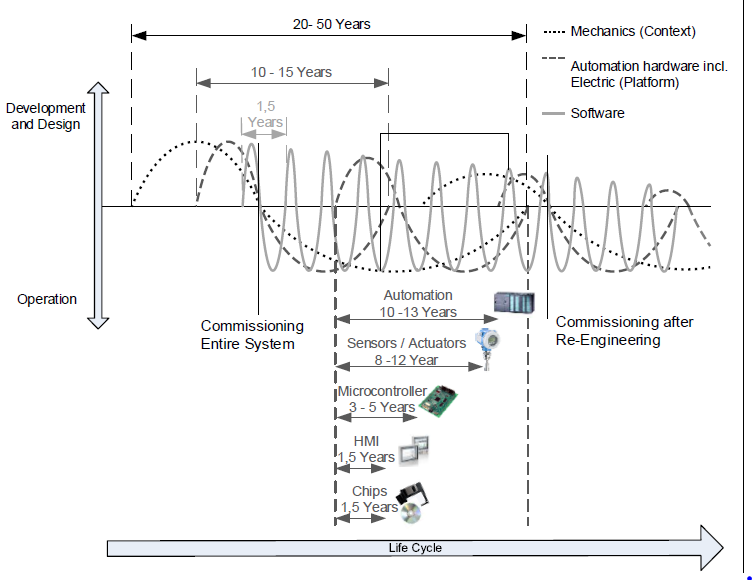


Figure 1 Software life cycles in comparison with mechanics and automation hardware including electronics and the rise in its importance for present day industrial plants.

* 1. Initial Situation

The foundations of this project were already set by previous projects in the context of the ProcAppCom research project at Gefasoft. A general description model for process engineering plants was initially developed. Before the start of this project, it was also possible for plant models to be read and directly transcribed to database tables of the MES Legato Sapient®. This enables the automated connection of the MES to the plant’s control and field levels via the factory edge gateways. A system for the automated generation of P&ID visualizations for the user selected site, area, production unit, process cell or equipment module of the modelled process engineering plant is the culmination of this research project.

The Legato Graphic Designer boardlet of Legato Sapient® was developed for the dynamic rendering of visualizations in the form of a single, static xml file. In favor of preventing repetition and to seamlessly integrate to the Legato Sapient Environment, existing code should be leveraged as much as possible. The final product is a dashboard for the creation and rendering of the user selected process engineering plant instance.

<INSERT GLOBAL PROCAPPCOM PROJECT OVERVIEW WITH ZOOM IN TO BOARDLET >

* 1. Goals of the Bachelor Thesis

The goals of the bachelor thesis give an overview of what is ultimately intended. They enabled a plan to be defined initially in terms of the desired outcomes of the project. The specifics for the fulfillment of the goals came later with the definition of technical and conceptual requirements. The following goals were set to define what was to be ultimately achieved by means of the developed solution:

* Reduce technical effort and accelerate the generation and modification of Piping and Instrumentation Diagram (P&ID) visualizations for industrial process plants.
* Normalization of the visualization components according to industrial standards and best practices for generation of consistent P&ID visualizations.
* Prototypal implementation of the software solution in the web-based MES Legato Sapient® in the form of a user-friendly GUI dashboard for the generation and viewing of P&ID visualizations with abstraction of configurations for the user.
* Verification, validation and evaluation of the solution for the implementation in an industrial grade manufacturing execution system.
  1. Project Requirements

Specific requirements where set apart from the project goals for the technical and conceptual aspects of the project. It was intended for all requirements to be met by the end of the project, but the development cycle brought slight variations to some during the course of the project. Technical requirements define the intended outcome of the practical implementation of the project at Gefasoft® Gmbh. Conceptual requirements define the core concepts which the proposed technical solution addresses and on which the functionality is based. Furthermore, the development of new methodologies to achieve the project goals outlined in section 1.4 represents the research part of the project.

R1: Library of modular P&ID Visualization Components According to Industry Standards

* Object-oriented abstraction of industrial process engineering elements into UML 2.0 classes based on their inherent geometrical and functional traits (class diagram).
* Conception of a library of modular and composable P&ID visualization components in SVG format according to the current industry standards.
* Static geometrical definition of visualization components in a P&ID shapes library implemented as a JSON file for efficiency, portability and ease of personalization and maintainability. Each shape corresponds to a JSON object with a defined set of properties with predefined values (defined by library version), or a default value if left blank.
* Shape instances inherit object properties from their parent P&ID class and category based on the data model defined as a UML 2.0 class diagram. This enables automatic propagation of inherited and composed styles throughout library for ease of user personalization.
* Shapes display real-time plant process values if a data binding to process variable exists.
* Semi-colon separated string of styles destructured into styles object for targeted configuration of individual styles. Styles object then concatenated back to string on XML generation.

R2: User Friendly Graphical User Interface (GUI) Boardlet for Creation of P&ID Visualizations

* User friendly boardlet with a simple interface for generating new or updating previously generated P&ID visualizations.
* No configurations needed and abstraction of inner workings for the user.
* File input for the selection of the desired version of the P&ID shapes library.
* Buttons for generating the XML of the visualization, for downloading the visualization in both JSON and XML format, and to upload the XML visualization file to Sapient Engine® file system for the Legato Graphic Designer® boardlet to import and render.
* Visual feedback: progress bar for script run and viewer for the generated XML as text.
* Dashboard with boardlets: P&ID Creator, Node Tree Selector, P&ID Viewer.

R3: Client-Side Script for the Automated P&ID Visualization Generation as an XML File

* Required from user is only the selection of the desired P&ID shapes library version for the visualization and no additional configurations.
* Script encapsulates all required business logic in a single modular and composable, well-documented JavaScript file.
* Separation of primary concerns: presentation logic, database queries, data mappings, graphing algorithm and xml generation are all separate and inter-independent from each other.

R4: Prototypal Implementation in the Infrastructure of a MES (Legato Sapient®) and Documentation

* Component-driven, modular design of boardlet implemented with the Ember.js framework used in Legato Sapient®.
* Evaluation of the system: functionality, performance and scalability.
* Clear documentation of code.
* Document with next steps in case of interest on continuing development.

R5: Mapping of Physical Plant Instances to Corresponding Visualization Component

* Mapping to work with minimal changes to the original data model for an unobtrusive implementation of the automated P&ID visualization generator and to avoid the need for new tables and fields in database.
* Minimum database requirement is a shapeName attribute to be property set in the model and thus in the database.
* Connections don’t require changes in model. The shapeName attribute for each line shape is determined via logic.

R6: Automatic Type Detection and Simplification of Connections

* Logic for setting the corresponding *shapeName* property to all connections: differentiate between data, process, connection and signal lines. Because of this, no need to specify a *shapeName* for connections in the data model.
* Connections defined in plant model, and thus also in database, in a logical instance to instance way, but suboptimal manner for the application of P&ID line shapes.
* Connections with multiple waypoints simplified by skipping intermediate ports, until a shape to shape connection (from start source to end target) is reached.
* Orthogonal line shapes optimized for minimal crossings and shortest routing between source and target.

R7: Declarative specification of Graphing Constraints in Form of Tags

* Declarative approach of tags which allow targeting specific shapes to be positioned according to specific set of positioning rules.
* Tags are loosely coupled so they don’t intervene with the algorithm, rather define the algorithm to be run.
* Separates vertex placement logic for shapes to be positioned with distinct positioning rules.
* Vertex placement algorithm can be easily progressively enhanced through the addition of more and more tags.

R8: P&ID Graphing Algorithm

* Research and analysis of state-of-the-art graphing algorithms.
* Simplicity over efficiency of the algorithm as to allow later improvements and since the creation of P&ID visualizations is not time critical.
* Depth-first post-order instance hierarchy traversal to get nodes in graphing order.
* Block-packing algorithm for the positioning of groups in groups to minimize the area.
* Algorithm concept for P&ID visualizations works no matter the complexity of the modelled process engineering plant.
* Ability of progressively enhancing the algorithm for creation of better and more complex visualizations without change in concept.
* Implementation of the algorithm for the example Aida Brewery plant.

R9: Dynamic Real-Time Display of Process Variables in the P&ID Visualization

* P&ID visualization with real-time updating shapes and shape labels.
* Different types of display of process values depending on data type of process variable and on shape category (for example: Boolean values set fill color for valves, but not for tanks).
* Components encapsulate a uniform set of data bindings to the actual process values and display values in real time.
* Default settings to override labels for shapes with data bindings to empty values.
* One-way data bindings that update automatically on the client-side instead of on the server-side for optimizing performance.
* Data bindings implemented with the sapient-bind property of the shape’s XML user-defined object which uses the mxGraph API already (placeholders).
  1. Composition of the Bachelor Thesis
     1. Project Management

In favor of lightweight and flexible project management, the GIST methodology was preferred over more traditional agile methods. GIST is called after its main building blocks: Goals, Ideas, Sprints and Tasks, each with distinct planning perspective and frequency of change [1]. Instead of initially declaring tasks, goals where set, which enabled a plan to be defined in terms of the desired project outcomes. The goals stated in section 1.4 lead the decisions from beginning to end of the project and where maintained for the most part. Ideas where tracked during the entire project’s life cycled and reconsidered for implementation or discarded at the beginning of each sprint. Sprints where executed until all tasks where completed, though tasks of previous and future sprints where sometimes worked upon outside of the corresponding sprint. Tasks themselves where reconsidered weekly for the current sprint and tracked with a Kanban board.

* + 1. Project Sprints

S1: Research and Choice of Tools and Technologies

Before any work was done, the tools and technologies with which the goals would be achieved had to be at least preliminary decided. Though many diagramming frameworks and libraries exist, not all where optimal for the task at hand, therefore comparisons where done between the available technologies after meticulous analysis of the projects technical and conceptual requirements. The open-source mxGraph JavaScript diagramming library was chosen due to its light weight, robustness between distinct web browsers and compatibility with the diagramming tool draw.io, built with mxGraph. Furthermore, the Legato Graphic Designer Boardlet in which the visualization is to be viewed is implemented with the mxGraph library.

S2: Creation of a P&ID Shapes Library

The second project sprint was the conception of an object-oriented library of modular shapes conforming to the industry standards for P&ID visualizations. This task was further divided into subtasks. first of which was the analysis of the mxGraph application programming interface (API), with which the visualizations were to be implemented in the browser. These consisted of the manual creation and analysis of example visualizations as well as a thorough reading to the API’s documentation. mxGraph is a fully client-side JavaScript diagramming library that uses SVG and HTML for rendering. The predefined process engineering shape library was used as a base for the next step: the creation of a statically defined, modular and composable object-oriented shapes library for P&ID elements. It was decided that this library was to be defined in JSON format, to facilitate user modification and tuning of the geometries, rather than in XML format like the visualization file itself.

S3: Requirements and Design of Software Architecture

After the creation of the statically defined P&ID shapes library file in JSON format came the conceptual elaboration of a preliminary software architecture for the project’s technical implementation in the MES Legato Sapient®. This task preceded the commencement of the agile development life cycle.

S4: Boardlet Design

Aligning to the Legato Sapient® design and coding principles implemented in the component based Ember.js framework, the start of the development phase consisted in setting up boilerplate code for the P&ID Creator Boardlet. After the creation of both a JavaScript (.js) and handlebars (.hbs) templating file, the preliminary wireframe design for the boardlet was made and coded. Attaining to principles of component- based design, the handlebars template was designed and developed modularly with both new and reused ember components. After having the boardlet up and running on the Sapient Engine® it became possible to start the progressive development of the business logic for the automated P&ID visualization generation.

S5: Generation of the XML file of the P&ID Visualization

The first development sprint for the generation of the XML file of the visualization where made before establishing a database connection for fetching of the plant instances on a separate testing boardlet. This testing boardlet allowed for constant modification and experimentation of the algorithms with pure JavaScript, HTML5 and CSS. These allowed for rapid coding without needing to be connected to the full sapient architecture. The plant hierarchy was first modelled statically in form of JSON files in place of the database queries which return equivalent JSON responses. The file input component for the uploading of the P&ID shapes library was recycled to directly load the needed files in the client, thus enabling faster trials and testing of the script in development until XML of unplaced, overlapping vertices was correctly generated.

S6: Connection and Fetching of Plant Instances from Database

After the script successfully and automatically generated an XML file of the P&ID from static JSON files of plant instances, the connection to the database and registering of database tables in the Legato Configuration Center® (LC2) followed. This allowed to test the XML generation script now with actual plant data queried from the database. This required a global data map of all required tables and fields. With the data map, name mappings where done and a function to fetch the data with custom filters implemented via the available Legato getRecords() function. Modification of the database queries could now be done only by modification of the passed parameters for the query. The result of the XML generation algorithm with the actual plant data corresponded now with what was originally modelled for the plant. By now, all vertices and edges where correctly instantiated in the diagram, but vertices where placed one on top of the other. This lead to the start of the graphing algorithm for the placement of these vertices.

S7: P&ID Graphing Algorithm

The main task during this sprint was the development of a vertex placement algorithm to set the x and y properties of each vertex according to a defined set of positioning rules. Both a declarative, rule-based approach and an algorithmic approach for the optimization of area where used. First part of the algorithm consisted in the declaration of constraints in the form of tags based on shape attributes. This way, the positioning logic could be later better targeted at the distinct tags individually, since distinct shapes are to be positioned based on a distinct set of rules. The loosely-coupled tags where specified first and apart from the positioning logic, as this part was based on algorithmic optimization rather than classification. Afterwards, distinct sets of positioning rules where defined for each of the tags. The shapes would thus be iteratively positioned by the algorithm in a distinct and independent way. Furthermore, a block-packing algorithm was to be implemented for the positioning of groups in groups to minimize the area. Though much progress was made in a short time, a standpoint was later reached. Though the algorithm could still be made better, the time invested was too much compared to the progress, and because of the lack of time, the algorithm was left as is in order to continue with the last sprint.

S8: Dynamic Real-Time Display of Process Variables in the P&ID Visualization

Although the output of the P&ID Creator boardlet and the input for the P&ID Viewer is a static XML file of the P&ID Visualization, it must contain the data bindings for the dynamic display of the process variables. The data binding should be independently set based on the data type of the process variable via the sapient-bind attribute in each XML object. This logic had to be set before the string generation so that the distinct data types of the process values result in distinct labels or colors for each shape instance. The Legato Graphic Designer requires the ID (primary key) of the value in the database and automatically fetches the value in the background whenever it changes. This functionality is implemented as a mxGraph placeholders and allows for the data bindings to be also included in the static XML file of the P&ID visualization.

S9: Prototypal Implementation and Evaluation in the Infrastructure of MES Legato Sapient®

Although plant instances where already modelled and available from the database during the project, no connection to the gateway of the example Aida Brewery plant existed. This

1. Technological Standpoint
   1. Industrial Process Control
      1. Overview

Control engineering is an engineering subfield which applies automatic control theory for designing systems with certain behaviors. FIGURE XXX shows some applications of control engineering in different contexts. This thesis will focus on the application of control theory in industrial production contexts, more specifically, in industrial processes.

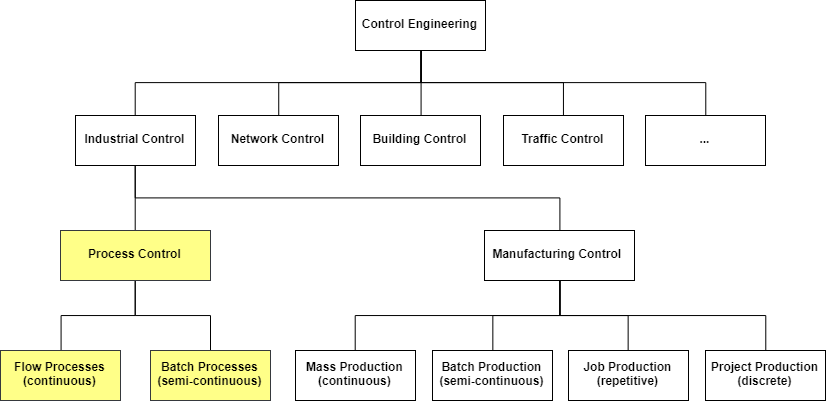


Figure 2 Scope of the bachelor thesis within the applications of control engineering. Photo from Automatisierungstechnik 1 Lehrstuhl AIS TUM

A process is defined as a series of actions taken to achieve an objective. More specifically, an industrial process is the systematic application of mechanical and/or chemical operations for the production or manufacture of something.

**Industrial process control** is thus the systematic optimization of consistency, economy and safety of continuous production industrial processes [TODAS LAS DEFINICIONES SON DE WIKIPEDIA]. To achieve this, process control incorporates the fields of control and chemical engineering to automate production of continuous or batch processes. In contrast to manufacturing control systems which typically favor flexibility because of the heterogeneous and usually discontinuous nature of manufacturing, process control systems favor robustness, real-time reactivity and safety above flexibility. This way, process control systems are implemented to run non-stop for decades. The size and complexity of industrial process plants generally represents a significant engineering challenge, but its long-lasting nature has historically taken the process industry to the vanguard in industrial production control.

Process control systems have traditionally been hierarchically structured. Nonetheless, the high degree of networking and the availability of substantial amounts of data has lead the trend for their decentralization. Enabled by an interconnected network of Cyber Physical Systems (CPSs), modern process control systems now allow for adaptive and self-configuring production. FIGURE XXX shows the trend for the decentralization of traditional automation into Cyber Physical Systems (CPS) for flexible, data-driven and efficient production.

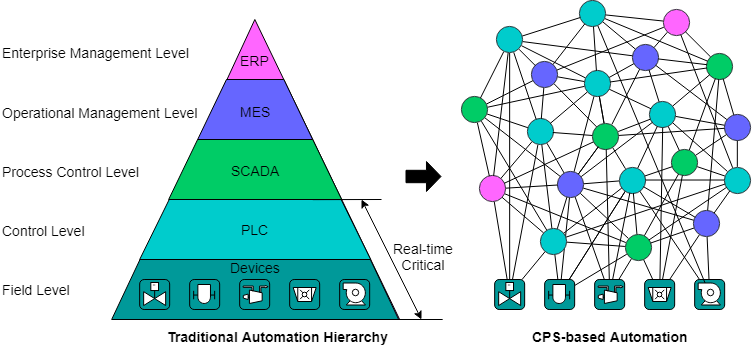
* 

Figure 3 Decentralization of traditional automation into Cyber Physical Systems (CPS)

* + 1. Plant Hierarchy Model (ISA-95) and ISA-88

The plant hierarchy model or [ISA-95](https://en.wikipedia.org/wiki/ISA-95), as it is more commonly referred to, is an international standard for developing an automated interface between enterprise and control systems [<https://en.wikipedia.org/wiki/Enterprise_control#ISA95_.E2.80.9Clevels.E2.80.9D_for_enterprise_integration>]. The model organizes the enterprise in key layers so that end users, integrators and vendors can share a common standard for integrating applications and can be divided in the following levels:

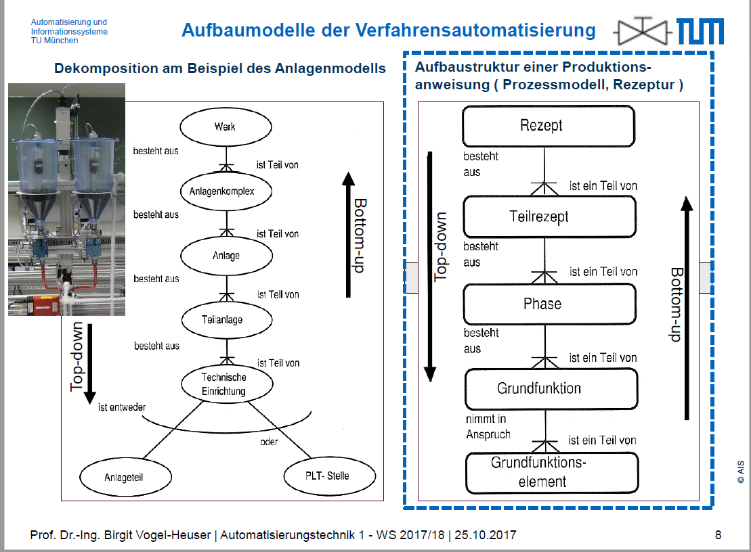
Level 0: The physical process

Level 1: Intelligent devices

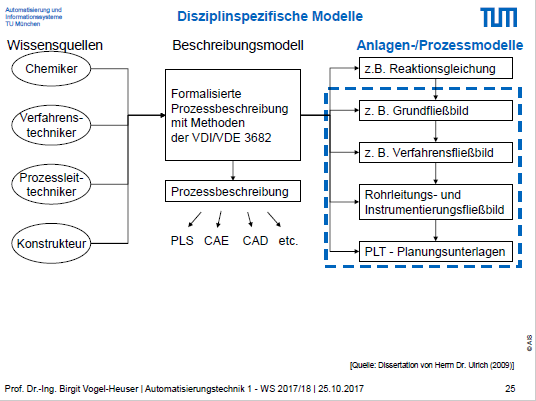
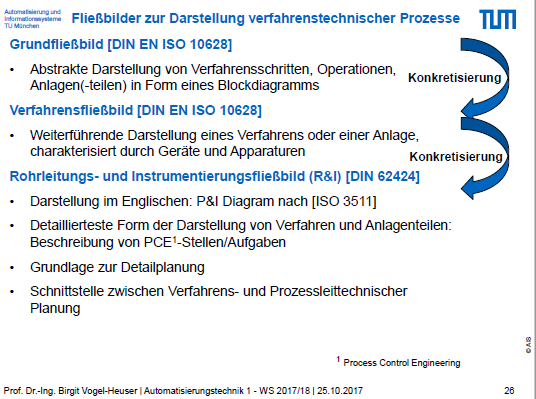
Level 2:

Level 3:

Level 4:



* ISA-95 and ISA-88

* + 1. Piping and Instrumentation Diagram (P&ID)
* Definition
* Functions
* Advantages
* Disadvantages
  1. Manufacturing Execution Systems
     1. Functions
     2. MES in Context of the 4th Industrial Revolution
     3. Overview of Legato Sapient®

Entirely Web-based architecture and modular and customizable to the core to keep of with requirements.

* Design

Component based, modular design of dashboards (easy creation by adding boardlets)

Dashboard > Boardlet > Ember Components

* Features
* Software Architecture
  1. Web Applications
     1. Overview

The World Wide Web was originally conceived as an information space for the distribution of static documents and resources interlinked by hypertext links and accessible via the internet [https://en.wikipedia.org/wiki/World\_Wide\_Web]. The web has since then been critical for the development of the information age. Initially, web content was limited to being static, simple text documents navigable via embedded hyperlinks. The web browser was nothing more than a search engine, hence the name “browser”. Nevertheless, with the later development of scripting languages like JavaScript, code could be programmed to run directly in the client’s browser. This empowered web browser to dynamically generate and update content in the client-side, what lead the shift to the development of web applications [<https://en.wikipedia.org/wiki/World_Wide_Web> ].

A web application is a full client-server computer programs with a user interface and client-logic running in the web browser. Although initially limited by the web browsers capabilities, new Application Programming Interfaces (APIs) have been developed which empower the browser with capabilities that were previously exclusive to native applications. These APIs allow web apps to leverage and consume services like Bluetooth, the camera, the local file system, sensors and accelerometers, GPS (Global Positioning System), amongst many others. Web and native are thus at par regarding many key functionalities for application development. Furthermore, the cross-platform nature of the web platform is a major advantage of developing web and not native applications. While web applications require only to be programmed once to run in all devices with a web browser and an internet connection, the code for native applications is usually platform-specific. By virtue of the recent development of the web platform, more and more applications are being advantageously developed for the web. The Legato Sapient® MES exemplifies how companies today leverage these benefits to build robust, industrial grade web applications.

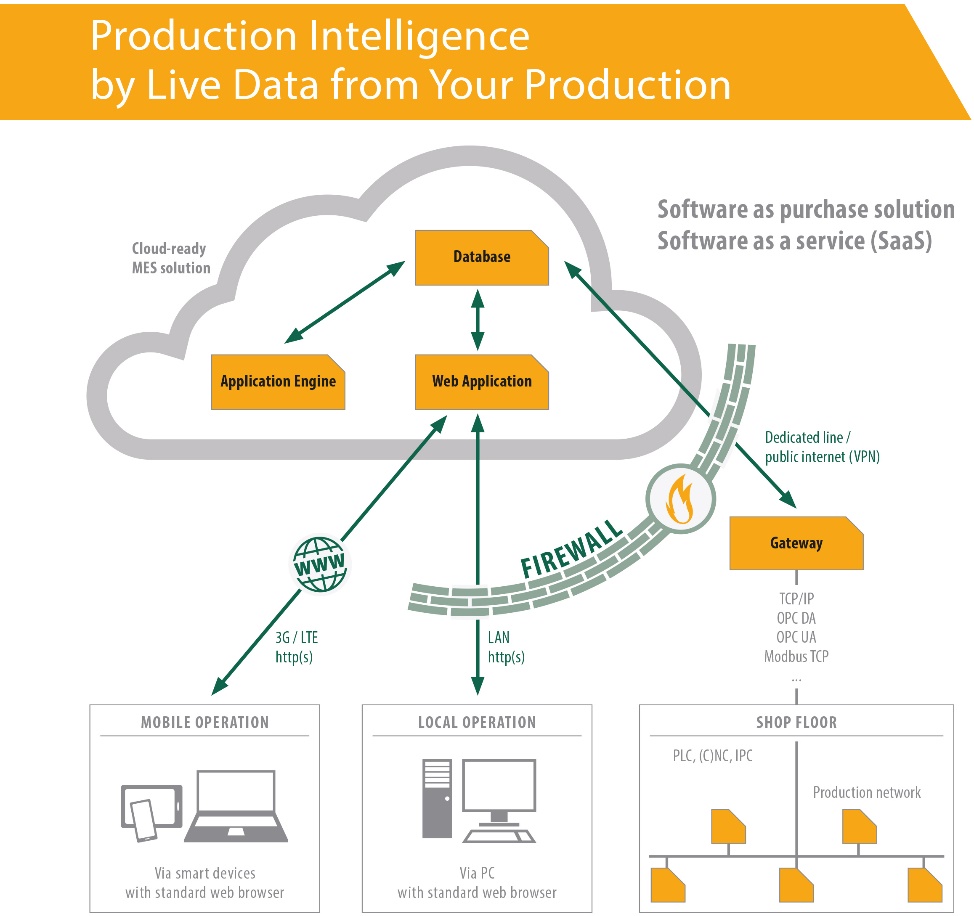


Figure 4 Legato Sapient® Software Architechture leverages the web platform to offer a fully-fledged MES Software as a Service (SaaS) solution. Photo from GEFASOFT

* + 1. List of Technologies

The accelerated technological evolution in recent years has brought significant changes to the web platform. Meanwhile, web application technologies have had to swiftly adapt to not lose ground against native application technologies. Driven by the constant evolution of the web platform, the amount of web application technologies has dramatically increased in parallel. The following list covers the concepts and technologies employed for the technical implementation of this project.

**HTML**

Hypertext Markup Language (HTML) is the standard markup language for the creation of web content. HTML elements or tags are the building block of the web and are used to declaratively define the structure and content of the web page.

**CSS**

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentational aspect (layout, colors, fonts, etc.) of content defined in a markup language like HTML. CSS enables the separation of style from content, and allows for sharing of styles throughout multiple web pages. CSS has selectors to target specific HTML elements and style rules which follow a priority scheme to determine the styles to be applied for the selected elements.

**JavaScript**

JavaScript (JS) is a dynamic, weakly-typed, high-level interpreted programming language. Along with HTML and CSS, JavaScript is one of the three core technologies of the world wide web. JavaScript enables web pages to be interactive and is thus essential for web applications. Being a multi-paradigm language, JavaScript allows for event-driven, functional, imperative and object-oriented programming. Although it was initially developed for the front-end as a client-side implementation for the web browser, it has now been extended to support back-end implementation for web servers and database development.

**XML**

Extensible Markup Language (XML) is a textual data format and markup language for encoding documents in a human- and machine-readable format. Although XML was specifically designed for documents, the language is also widely used for the representation of arbitrary data structures such as those used in the web [ <https://en.wikipedia.org/wiki/XML> ]. XML documents use user defined tags and implementations can be specified with the use of any of the available schema systems. Moreover, many APIs exist for distinct programming languages for working with XML. In this project, the XML format is used to define the generated P&ID visualization, or more specifically the data structures that define it.

**JSON**

JavaScript Object Notation (JSON) is a lightweight data interchange format used to represents arbitrary data structures as data objects consisting of attribute-value-pairs and array data types [ <https://en.wikipedia.org/wiki/JSON> ]. The JSON format is commonly used for asynchronous browser-server communication and has replaced XML in many applications. JSON is a language independent data format, nevertheless, it was derived from JavaScript and thus is syntactically similar to a common JavaScript object. This, its high readability, amongst other advantages facilitate working with data structures in JSON, reason for which it is the chosen data format for the P&ID shapes library of this project.

**Scalable Vector Graphics (SVG)**

Scalable Vector Graphics (SVG) are an open-standard vector-image format developed by the web to represent two-dimensional interactive and animable graphics in an XML-based format. SVG allows for three types of graphics: vector graphic shapes such as outlines and paths, text and bitmap images. Vector-graphics enable geometries to be defined in mathematical terms, instead of as a bitmap of pixels, which in turn keep the image quality when scaling and zooming. The standard supports the grouping of graphical objects, as well as linking, animation, font-selection, metadata, filter-addition, amongst others, all of which can be directly modified in standard text editors. All major browsers support SVG making it a preferred format for high-quality graphics in the web.

**mxGraph and Draw.io**

mxGraph is an open-source diagramming library for the rapid creation of interactive charts and diagrams that run natively in web browsers. The mxGraph API exposes a number of functionalities for the implementation of the library and is available in various programming languages, including JavaScript, Java, PHP and .NET. The API provides a robust package of high-level methods and services for integration in web applications. Built using the mxGraph library, draw.io is a visual diagramming tool with rich functionality (that of mxGraph) for the creation of fully integrated charts and diagrams in other popular software platforms like Google GSuite, Confluence and Jira.

**Web Development Frameworks: Ember JS**

Ember JS is a front-end JavaScript framework that facilitates building websites with rich and complex user interactions. Ember JS provides developers with features to manage complexity in modern web applications, as well as an integrated development toolkit for rapid development and iteration. Features of the framework include: routing to drive the application state via common URLs, a data layer to manage application state and provide a consistent way for external API communication, a Handlebars based templating engine, a robust Command Line Interface (CLI) toolkit to create, develop and build ember applications, the Ember component model, which facilitates the creation of web components, amongst others.

**Relational Database: PostgreSQL and SQL**

A relational database is a collection of data structured based on a relational model of the data [ <https://en.wikipedia.org/wiki/Relational_database> ]. The relational model organizes the data into one or more tables of columns (or attributes) and rows (or records), with a unique key to identify each row. Normally, each table represents a distinct type of entity, like a student or an exam, and rows represent individual instances of that entity, like “Mike” or “Math”. In relation to object-oriented programming, it can be said that tables are analogous to classes, rows to objects and columns to the object’s attributes, with fields being the specific values of those attributes for the corresponding object.

Normally, Structured Query Language (SQL) is used to manage a relational database. SQL is a domain-specific language designed manage data through Relational Database Management Systems (RDBMS). Features of the language include data definition, data manipulation (update, insert and delete), data query and data access control. SQL defines certain statements for working with the data at hand using relational algebra and tuple relational calculus. A typical SQL statement or query consist of clauses like “SELECT”, “UPDATE”, “SET”, “WHERE”, expressions and predicates and allow to retrieve and persist changes to data.

The database at hand for this research product is implemented in PostgreSQL. PostgreSQL is an open source object-relational database system with an extensive and powerful set of features for developers. The PostgreSQL database and an available Legato-specific database API were available before the start of the project. More on the database implementation for the context of this project will be discussed later.

**Data transmission: AJAX and JSON API**

The Fetch API provides a simple interface for the fetching of resources, including across the network [ <https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API> ]. The Legato Engine implements its own high-end functionalities to query the database via the Fetch API in the form of Asynchronous JavaScript And XML (AJAX) requests that return JSON objects. These return objects are specified by the JSON API to ensure best practices. The JSON API is responsible for the definition of standards and best practices for data exchange in JSON format [following from <http://jsonapi.org/format/> ]. It specififies how a client should request to fetch or modifiy resources, and how a server should respond to those requests. The correct use of JSON API can minimize the number of requests and the amount of data transmitted between clients and servers.

* 1. Unified Modelling Language (UML)

The Unified Modelling Language (UML) is a graphical description language for specification, visualization, construction and documentation of systems. The use of standardized graphical modelling languages like UML increase the clarity and thus understanding of the model even across different engineering disciplines. The different types of UML diagrams can be classified into two categories: structure models and behavior models and further classified as FIGURE XXX shows.

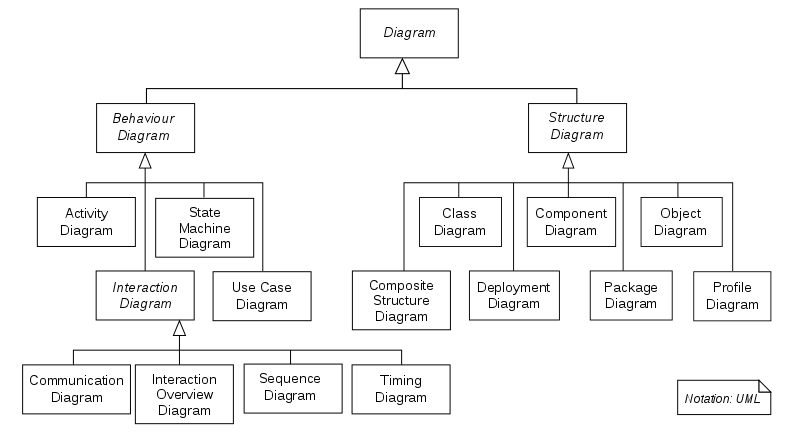


Figure 5 UML 2.0 diagram hierarchy shown as a class diagram

Apart from the standard UML 2.0 implementation, the standard allows for its extension by means of new element declarations using stereotypes and constraints. The extensions thus facilitates domain specific implementations. In production automation systems (both process and manufacturing), extension profiles like SysML (Systems Modeling Language) and UML-PA (UML for Process Automation) serve to model the system description. The system description can be further broken down as follows [LEHRSTUHL AIS SEFI1]:

Plant structure: component hierarchy, topology, component relationships

Plant components: Mechatronic structure, properties/parameters, economic data

Networks: electrical construction, communication systems

Behavior: component behavior, control system design, robot processes

Geometry and Kinematics: mechanical construction, motion planning, electrical construction

This enables UML to directly address the specific modelling requirements for systems of distinct nature. Throughout this document, following UML 2.0 and extension diagram types will be used, some of which with slight modifications and simplifications:

* + 1. Class Diagram

Class Diagrams are a type of static structure diagrams for the description of systems in the form of classes, their attributes, methods, and the relationships to other classes. A class diagram can be considered the main block of object-oriented programming, as it abstracts systems into detailed conceptual models which can be used to generate programming code. Class diagrams can also be used for data modelling and are thus crucial for the structuring of a system into its corresponding data model.

* + 1. Object Diagram

An object diagram is a graph of instances, including objects and data values. A static object diagram is thus an actual representation of the state of the system at any point in time, encompassing both objects and relationships at that moment in the system. Although similar to class diagrams in appearance, an object diagram further serves software developers to examine specific iterations of a more general system and consequently, to achieve a detailed overview of the system at the object level, rather than the more general class level. Object Relational Mapping (ORM) is facilitated by virtue of this diagramming method. ORM systems serve to convert data between incompatible type systems using object-oriented code. In this project for instance, the data in the relational database tables had to be mapped to object instances for its manipulation with code. An object diagram was used correspondingly to define the data model, which would later be mapped using JavaScript at runtime to JavaScript-compatible object instances.

* + 1. Entity Relationship Diagram

Entity Relationship Diagrams (ERD) are a type of flowchart used to map how entities relate to each other within a system. In software engineering, ERDs are most commonly used to design and debug relational databases. This type of diagram defines a set of graphical symbols to depict entities with their attributes and relationships in which entities are usually expressed nouns and relationships as verbs. Several notations of such graphical symbols exist; In this writing, a simplified version of the Crow Foot style for Information Engineering will be used.

* + 1. Activity Diagram

Activity Diagrams are a subset of UML behavior diagrams used to describe the dynamic aspects of a system’s workflow as a set of step-by-step activities or actions. As such, activity diagrams are key in the representation of algorithm logic, to model software architecture elements like methods and functions, and in clarifying complicated use cases for the simplification and improvement of processes. They consist of actions, decision nodes for conditional divergence of flow, control flows, and a start and end node.

* 1. Graphing Algorithms
     1. Graph Theory

Graph theory stems from discrete mathematics and is the study of graphs, mathematical structures used to model pairwise relations between objects [<https://en.wikipedia.org/wiki/Graph_theory> ]. Graphs have many applications, most notably in modeling of social, biological, physical and information systems [Mashaghi, A.; et al. (2004). "Investigation of a protein complex network". European Physical Journal B. 41 (1): 113–121].

A graph can be mathematically described as an ordered pair G=(V,E) composed of a set of *nodes* or *vertices* V and a set of *lines* or *edges* E, which are two-element subsets of V since each edge is associated to two vertices. These associations can be summed up in an adjacency matrix, which is a square matrix used to represent finite graphs mathematically. Graphs can therefore be entirely mathematically defined, which permits the formalized nature of representations in graph form and their complexity analysis. Graphs can therefore be summed up in three elements, nodes or vertices, lines or edges, and adjacency, all of which can be described in various formats: visually, with mathematical notation and in matrix form (adjacency matrix).

* + 1. Types of Graphs

The flexible nature of graphs results in the possibility of breaking them down into countless different types, some of which are trivial for most cases, and most of which attain to no particular standard in their nomenclature. Rather than defining all graph type possibilities, only those relevant to this project’s scope and context are considered. The categorization is split into two subcategories based on either their relationship characteristics or layout heuristics.

Based on Relations

Based on their mathematical categories, or more specifically the nature of their edges, graphs are broken into:

* **Undirected Graph**: a graph with edges with no direction between vertices
* **Directed Graph**: a graph with directed edges, which connect vertices in a specific way
* **Weighted Graph:** a graph in which each edge is given a numerical value or weight to represent distinct things like costs, lengths or capacities. Although usually numerical, weights can be nominal, ordinal or quantitative

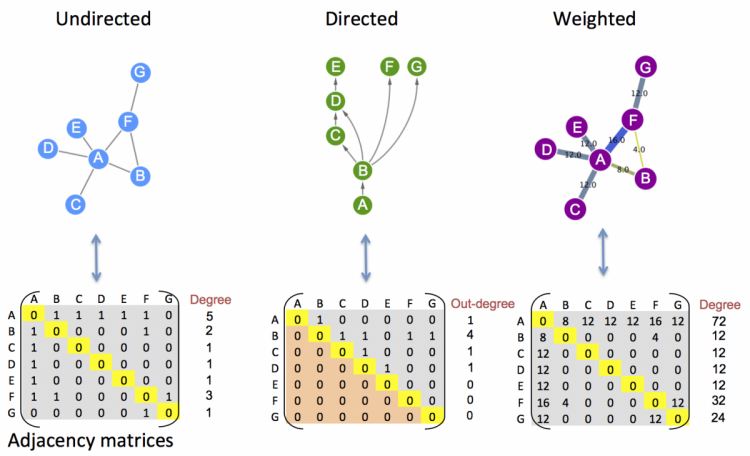


Figure 6 Types of graphs relevant to this project and their corresponding adjacency matrices. Source: EMBL-EBI

Based on Layout Heuristics

Based on their layout heuristics, which significantly influence the visual appearance, following graph types are relevant for the scope of this project:

* Planar Graph:
* Grid-based Graph:
* Orthogonal Graph:
* Hierarchical Graph:
* Horizontal/Vertical Flow Graph:
* Horizontal/Vertical Tree Graph:
* Circular Graph:
* Organic Graph:

**<INSERT A SMALL PICTURE OF ALL GRAPH LAYOUTS IN A BIGGER PICTURE OF 1O IN TOTAL>**

* + 1. Implementation of Graph Visualizations in Software

Although many algorithms exist for the efficient generation of graph visualizations, a number of aspects directly proportional to the size of the graph must be considered and several challenges remain as stated in [ <https://slideplayer.com/slide/12488750/> ]. This include:

* **Graph layout and positioning:** consists in making a concrete render of an abstract graph. Complex since vertices can have any degree (any number of connected edges). Further sub-challenges include:
  + **Rank Assignment:** compute which nodes have larger degree to place them at center of clusters
  + **Crossing Minimization:** swap nodes to rearrange edges
  + **Subgraph Extraction:** Pull out clusters of nodes
  + **Planarization:** pull out a set of nodes that can lay out on plane
* **Scaling**: challenging for large graphs which cannot fit vertices and edges into screen space. Scaling can also significantly slow down an algorithm
* **Navigation and Interaction:** how to support user moving around graph and changing focus without entailing a new render
* **Vertex issues:** defining the shape, color, size, location and label for vertices of different types, with different positioning, etc.
* **Edge issues**: defining the shape, color, size, label, form, if polyline, straight line, orthogonal, etc.

Additionally, the following complexity considerations:

* **Crossings**: minimize crossings to strive for a planar graph
* **Total Edge Length:** minimize towards proper scaling
* **Area:** minimize for efficient use of space
* **Maximum Edge Length**: minimize the longest edge
* **Uniform Edge Length**: minimize variances in lengths of edges
* **Total Bends:** minimize orthogonal bends to favor straighter lines
  + 1. Graph Layout Algorithms

Likewise, many graph layout algorithms for different graph types exist. With respect to this project, the following are relevant and where used as reference in development of this project’s P&ID graph layout algorithm:

* **Grid Layout**: consists in placing vertices on a constrained two-dimensional grid
* **Tree Layout**: traverses the tree hierarchy in order (for example by using a depth-first or breadth-first algorithm) and sets vertices in that order
* **Force Directed Layout**: models graph as a set of masses connected by springs, and simulates this mass-spring system for the layout or separation of the nodes
* **Planar Layout:** detects parts of the graph which can be laid out without edge crossings and lay them out



Figure 7 DELETE??? Overview of types of Layout Algorithms. Source: SIAT.SFU.CA

<INSERT A 2x2 DIAGRAM OF EACH GRAPH LAYOUT ALGORITHM TYPES>

* + 1. Graphs and P&IDs

The P&ID visualizations required for this project borrow characteristics from all graph types mentioned, nevertheless, tree structures are the best match. Trees represent a subset of a general graph with no cycles and a single, designated root vertex, typically only with directed edges. Trees are analogous to the plant instance hierarchy modelled by the SysML model and translated to the relational database in form of flat tables (REFER TO SECTION XXX). Despite this translation, the intrinsic hierarchical structure of the model remains in the flat database tables in the form of parent and children attributes. Being a tree structure what best embodies the structure of the P&ID visualization to be generated, the tree layout algorithm heavily influenced the developed algorithm.

* 1. Related Works
     1. Overview of Related Works

The topic of automatic visualization generation is much investigated, and many concepts exist for different purposes, nevertheless, not many exclusively targeted at industrial process applications. Consequently, no one research nor practical project was identified, which coincides with the majority of this project’s requirements. Following is a list of related works which do indeed assimilate some of this project’s requirements outlined in section 1.5 and listed again below in figure <XXX>.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Work** |  | **CR1** | **CR6** | **CR7** |
| W1 | Prat, 2017 |  |  |  |
| W2 | Oppelt, Wolf, Drumm, Lutz, 2014 |  |  |  |
| W3 | Yang, Prasad, Xie, 2013 |  |  |  |
| W4 |  |  |  |  |
| W5 |  |  |  |  |
| W6 |  |  |  |  |
| W7 | Romero Karam, 2018 |  |  |  |
|  |  |  |  |  |

Figure 8 Overview of Related Works

|  |  |
| --- | --- |
| **Key** | Title |
| W1 | An Automated Generation Approach of Simulation Models for Checking Control/Monitoring System [1] |
| W2 | Automatic Model Generation for Virtual Commissioning based on Plant Engineering Data [2] |
| W3 | A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications [3] |
| W4 |  |
| W5 |  |
| W6 |  |
|  |  |

* + 1. Comparison of Related Works

The conceptual requirements to be addressed for this project compare as follows to the previously listed set of related works:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Work** | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** |
| W1 | ✓ |  |  |  |  |  |  |
| W2 | ✗ |  |  |  |  |  |  |
| W3 | ○ |  |  |  |  |  |  |
| W4 |  |  |  |  |  |  |  |
| W5 |  |  |  |  |  |  |  |
| W6 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Figure 9 Related works summary based on the adressed conceptual requirements.

✓✓ – Addressed and implemented

✓ – Addressed and partially implemented

○ – Not addressed

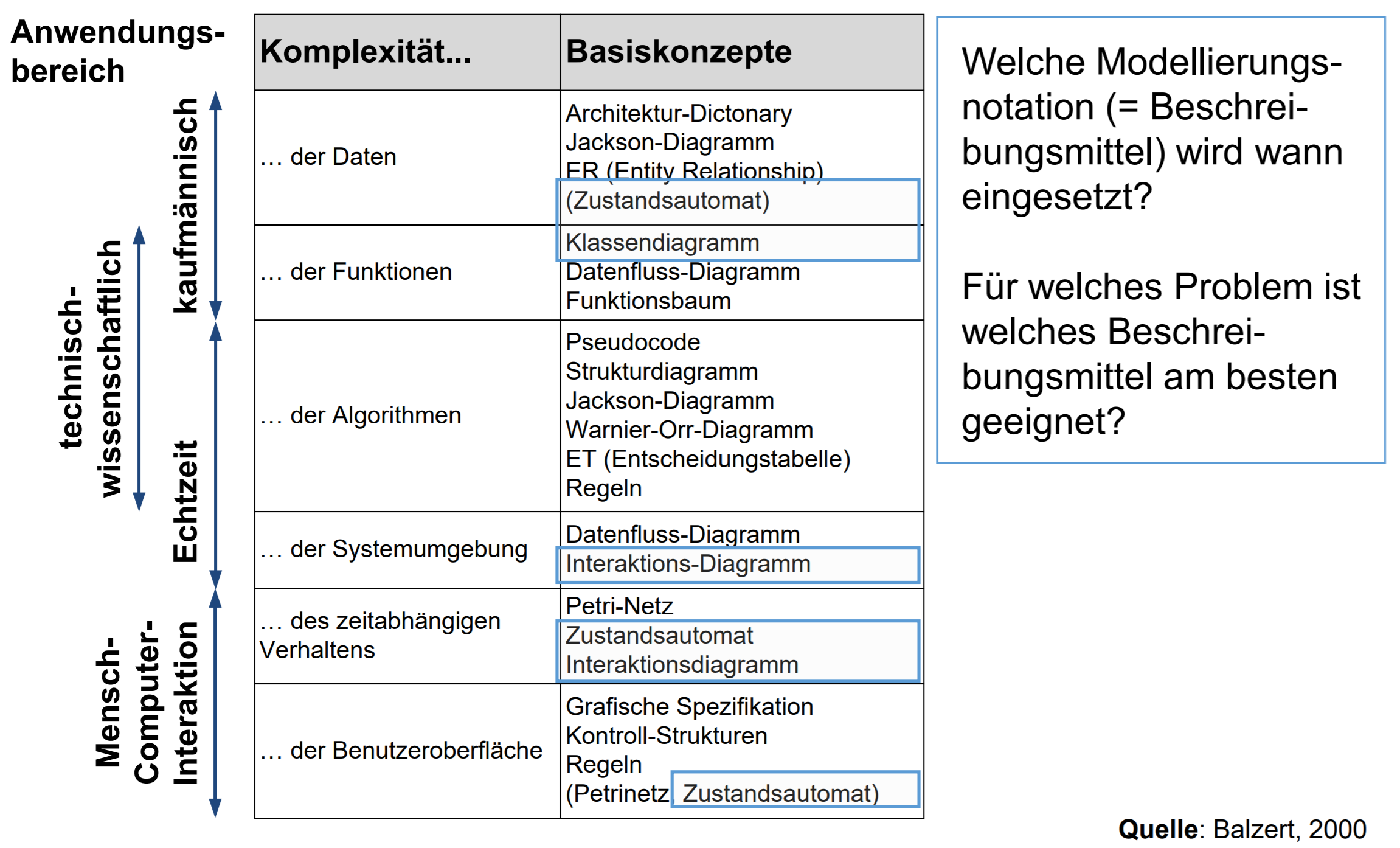
✗ – Not implemented

|  |  |
| --- | --- |
| R1 | Library of modular P&ID Visualization Components According to Industry Standards |
| R2 | User Friendly Graphical User Interface (GUI) Boardlet for Creation of P&ID Visualizations |
| R3 | Client-Side Script for the Automated P&ID Visualization Generation as an XML File |
| R4 | Prototypal Implementation in the Infrastructure of a MES (Legato Sapient®) and Documentation |
| R5 | Mapping of Physical Plant Instances to Corresponding Visualization Component |
| R6 | Automatic Type Detection and Simplification of Connections |
| R7 | Declarative specification of Graphing Constraints in Form of Tags |
| R8 | P&ID Graphing Algorithm |
| R9 | Dynamic Real-Time Display of Process Variables in the P&ID Visualization |



1. P&ID Shapes Library

PARA SELECCIONAR EL TIPO ADECUADO DE MODELL PARA LA INFORMACION QUE BUSQUE PRESENTAR, YA SEAN DATOS, FUNCIONES, ALGORITHMOS, ETC



* 1. Introduction

The Legato Graphic Designer boardlet where the generated P&ID visualization in form of a single, static XML file is to be uploaded for rendering, implements the mxGraph API already, from which the draw.io diagramming software tool is built upon. This heavily influenced the decision for the implementation of the mxGraph API for the project. Still, other alternative libraries and frameworks where considered and compared. Nonetheless, it was concluded that the intended functionalities could indeed be implemented via the mxGraph library and that it was the best option. Moreover, this mature, open-source API has many implementations from which to choose from. The mxGraph JavaScript library was selected for this project and implemented for the object-oriented abstraction of the geometries of the graphical elements. These graphical elements or shapes where abstracted in terms of the parameters specified by the mxGraph library for later compatibility. Apart from the adoption of the existing mxGraph parameters that define the shape’s geometries, most which are directly derived from the SVG format, no other mxGraph methods or services where used; The geometrical parameter specification was soley analyzed and structured into a general class diagram, from which a static shapes library was assembled to be later manipulated with pure JavaScript. This allows for the developed solution to be virtually library-independent.

* 1. mxGraph API

<INSERT Overview FROM 1.1 and Basic Licenscing Info FROM 1.5 and javascript client side working principles FROM 2.2 FROM mxGraph Javascript User Manual>

* + 1. Core Architecture

<INSERT mxGraph API Diagrams globales (class diagram)>

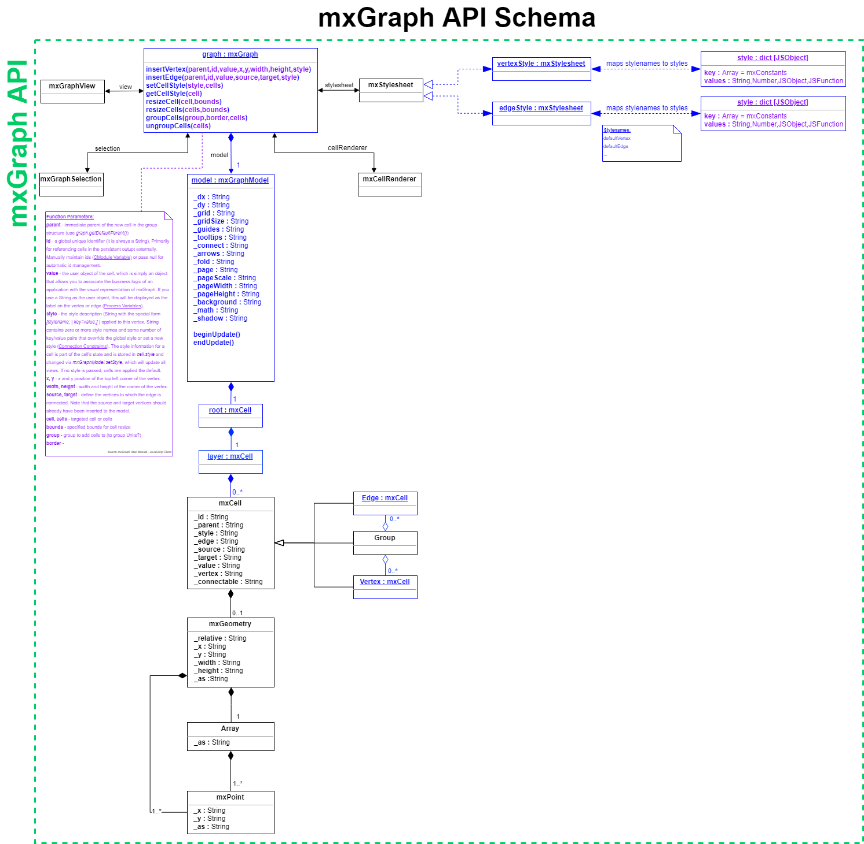
* mxGraph Model
* Transaction Model
* mxCell
* Styles
* Geometry
* User Objects
* Cell Types
* Group Structure
  + 1. Technologies
* Deployment

<INSERT HIGHLIGHTS FROM 1.3 mxGraph JavaScript User Manual>

* mxGraph Technologies

<INSERT HIGHLIGHTS FROM 1.4 mxGraph JavaScript User Manual>

* + 1. Schema



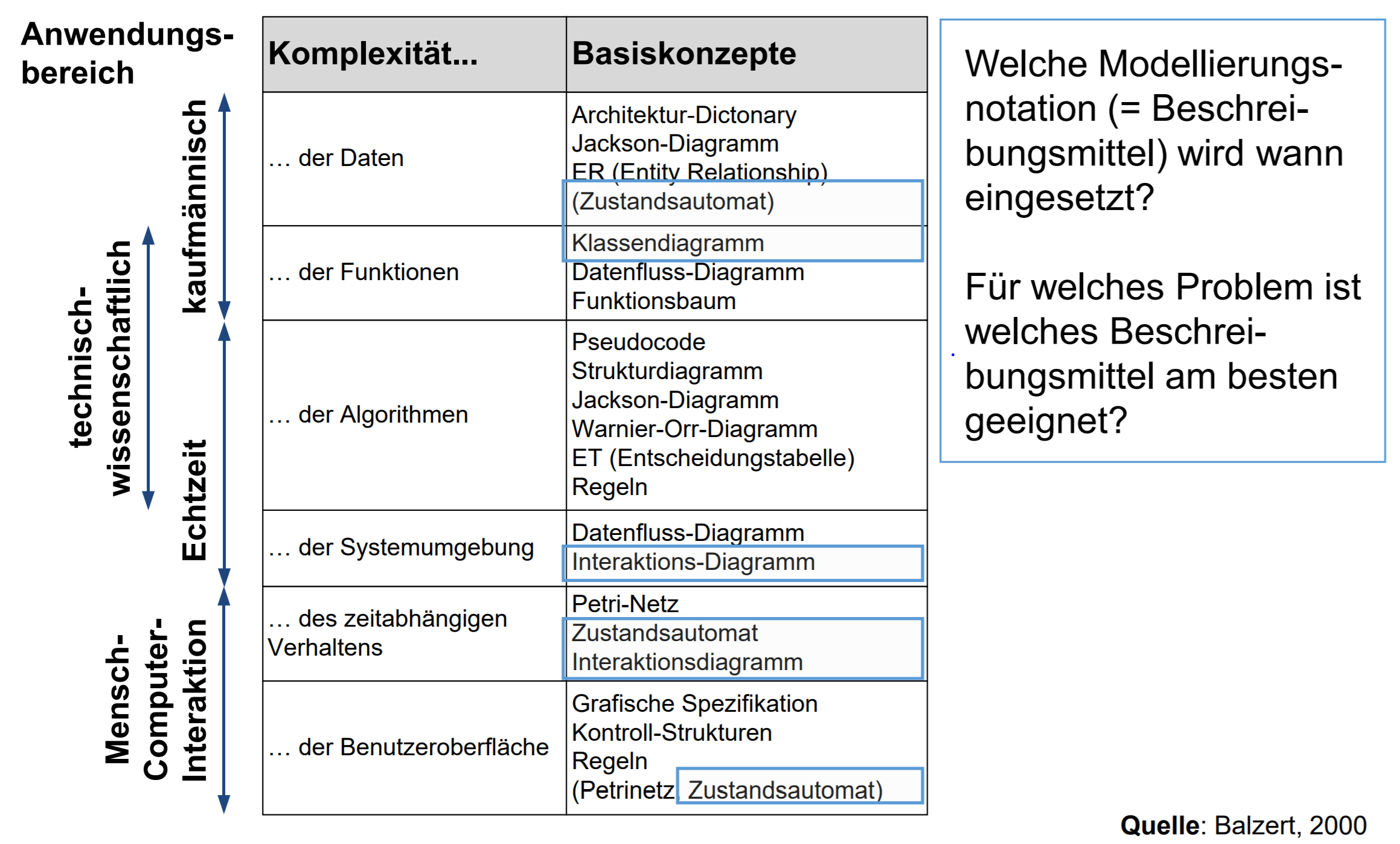
* 1. Object-oriented library of P&ID Shapes

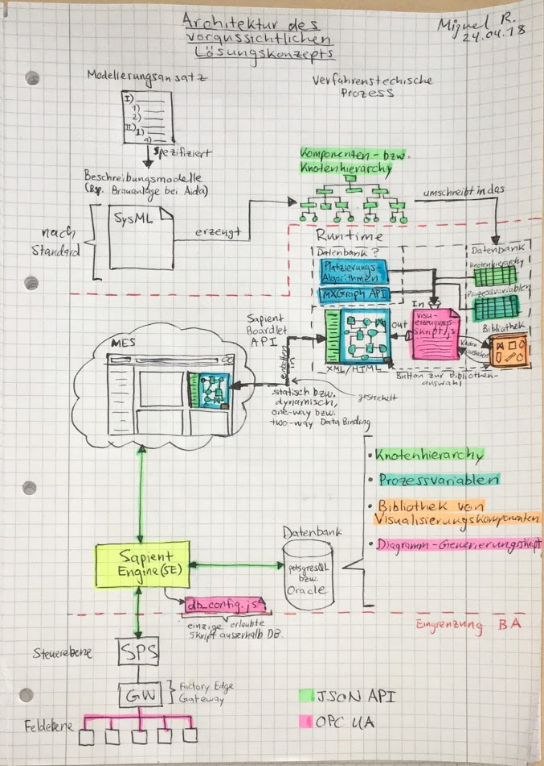
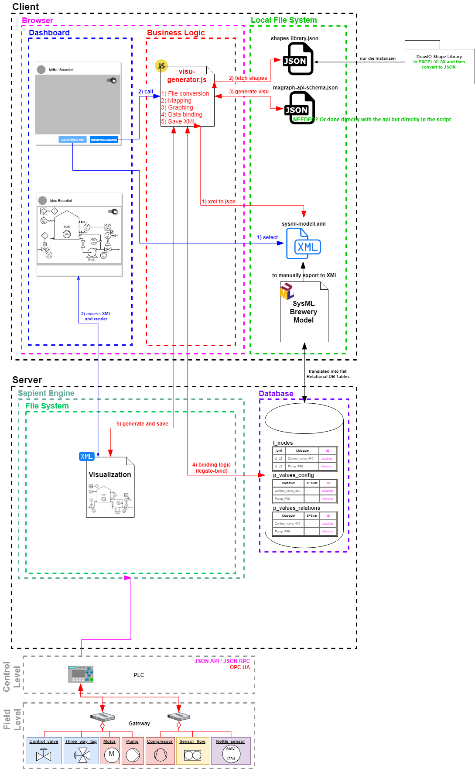
1. Legato Sapient® Boardlet



* 1. Overview of Software Architecture

CHECAR SEFI K1 FOLIE 46-47 PARA DETERMINAR EL MEJOR TIPO DE DIAGRAMA (PARA GROBDESIGN Y FINEDESIGN Y IMPLEMENTIERUNG ETC) O ESTO MEJOR:





Agregar datenbank y SE y conectar modelierungsansatz y knotenbaum y reestrucRequirements

* 1. Design

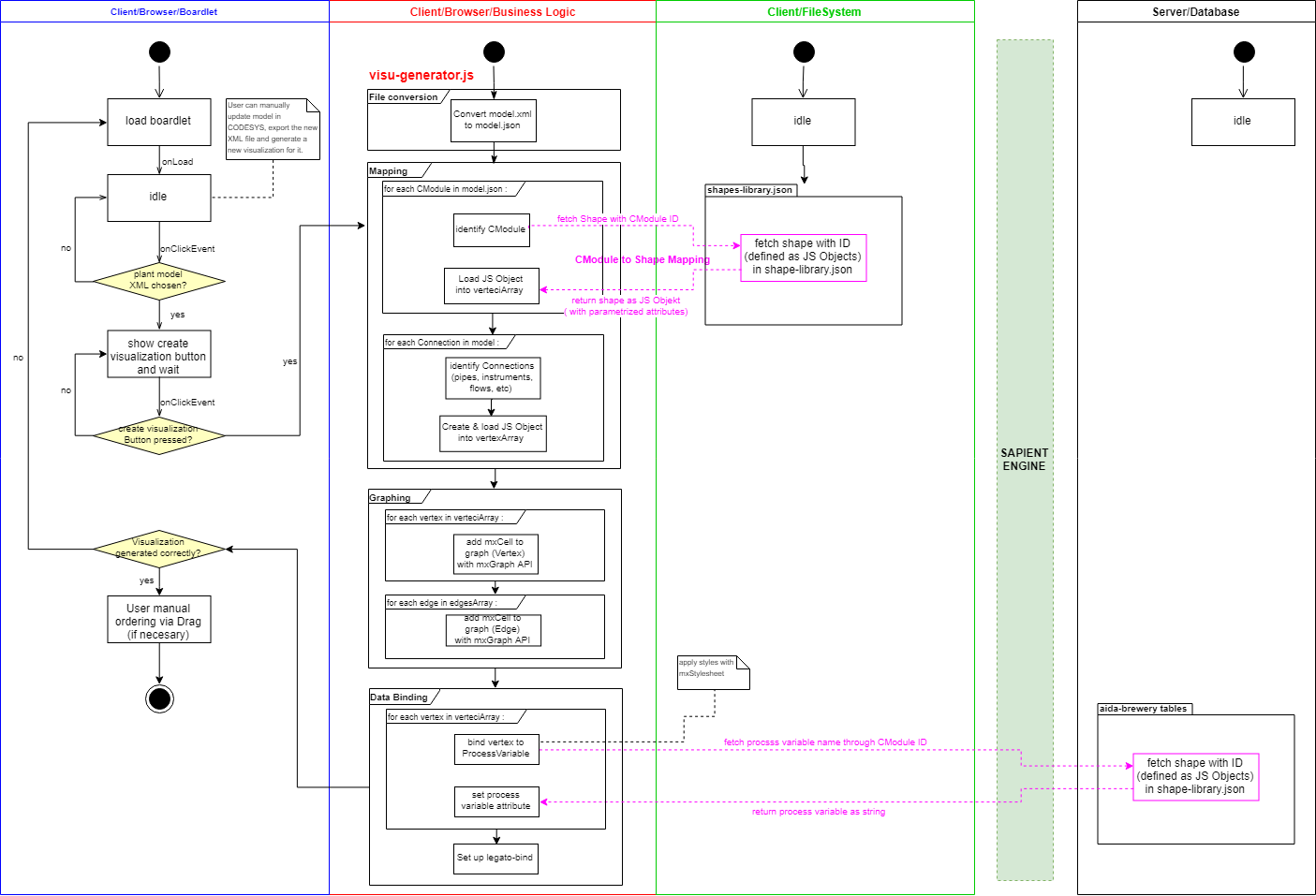
<INSERT PHOT OF P&ID VISUALIZER DASHBOARD WITH ALL BOARDLETS, SQUARE ON THE P&ID CREATOR BOARDLET AND SQUARE ON INDIVIDUAL EMBER COMPONENTS? OR TOO UNUBERSICHTLICH >

Modular Component-based solution. All included, up and ready boardlet.

* + 1. User Interface (UI)
    2. Presentation Logic
  1. Business Logic

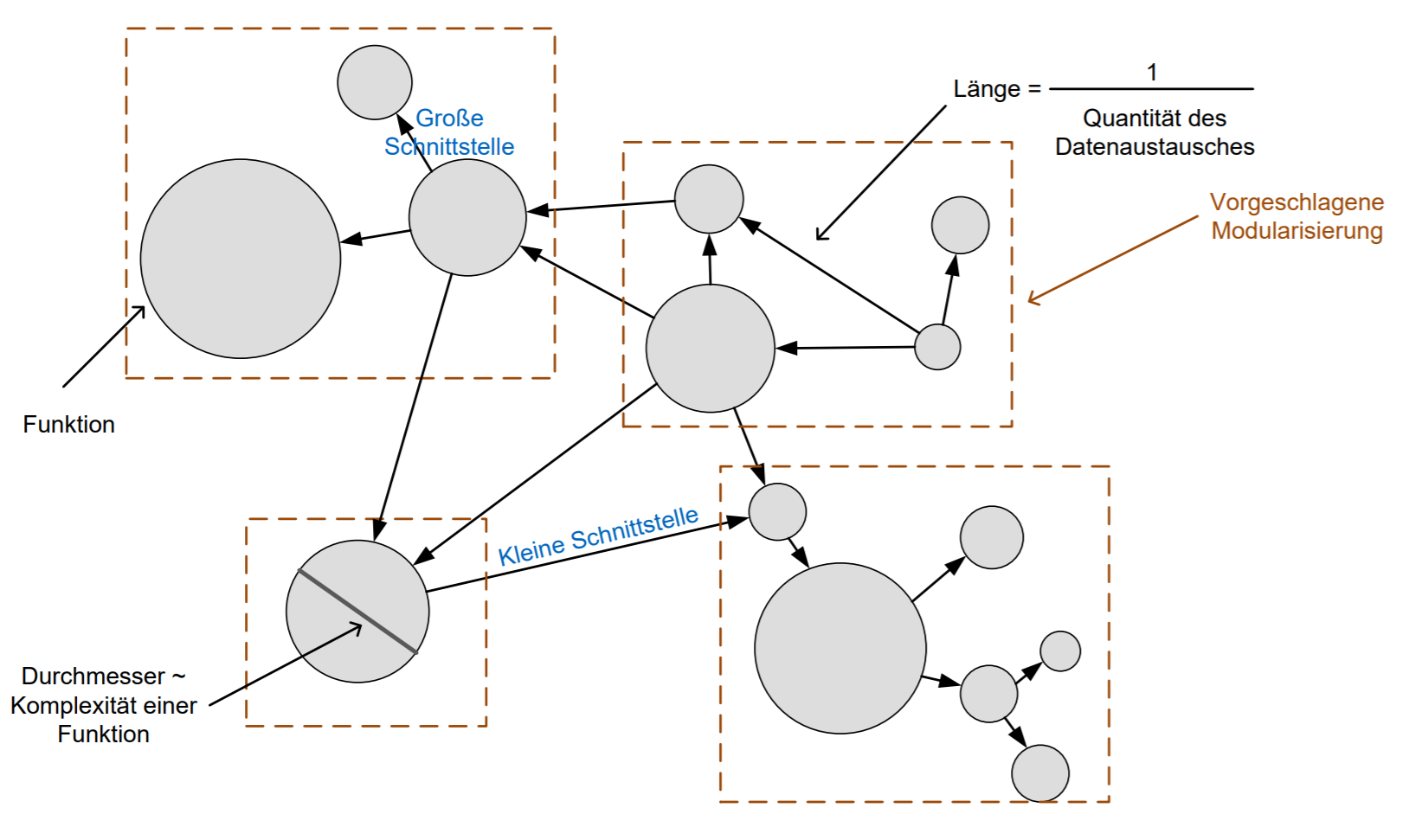
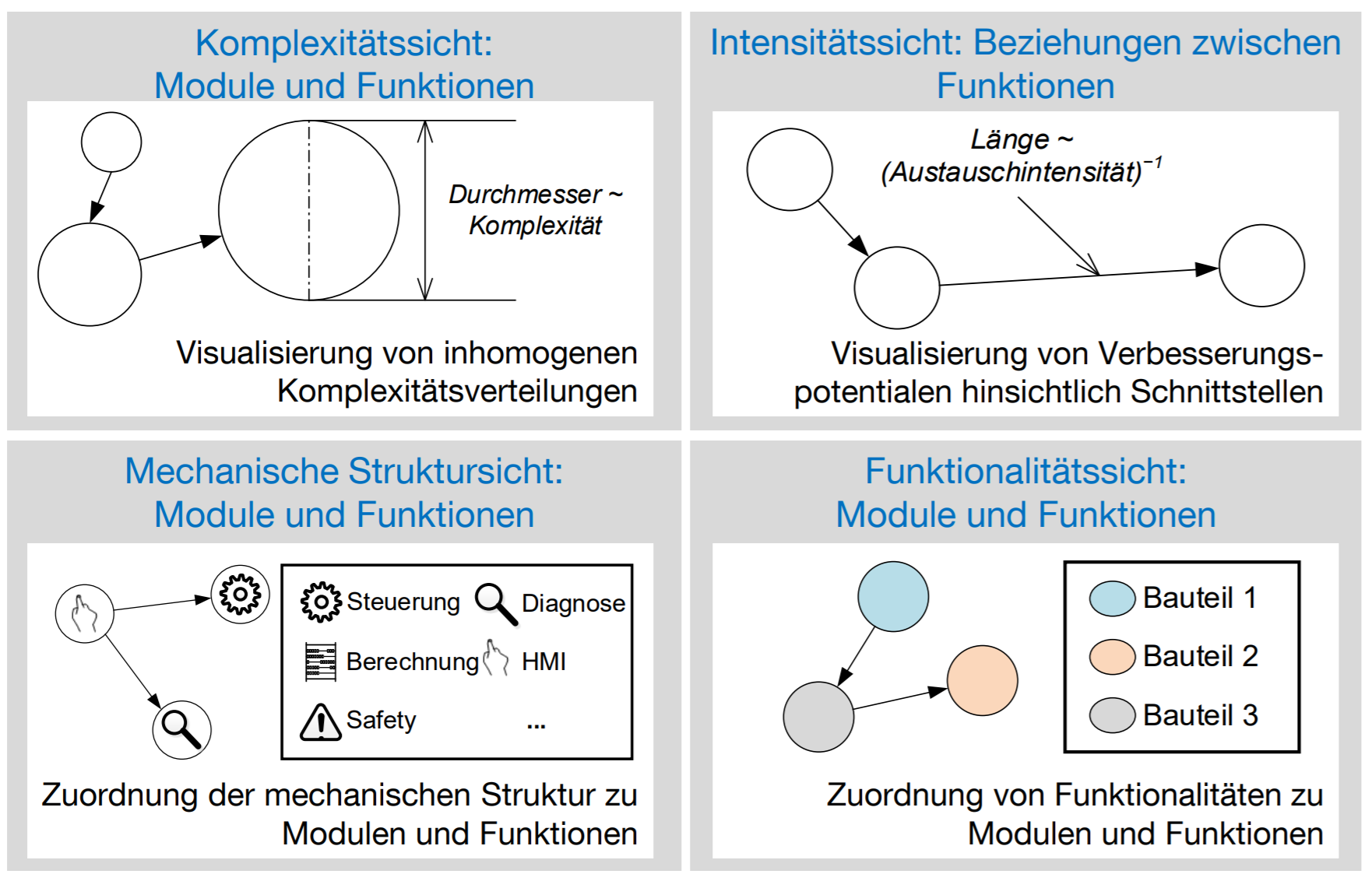
<INSERT Table with ALL FUNCTIONS AND VARIABLES LIKE API DOCUMENTATION>

* + 1. Overview



Software Metriken: Stukturelle Vergleiche (SEFI K.5.4 Folie 55)

HACER UNO DE ESTOS GRAFICOS PARA EXPLICAR COMO ES QUE MODULARICÉ EL CODIGO EN BASE A LA COMPLEJIDAD DE LAS FUNCIONES Y EN EL DATENTAUSCH ENTRE ELLAS (ADAPTAR EL GRAPHICO A EL CODIGO QUE YA ESTA MEDIO MODULARIZADO) IMPLEMENTAR TAMBIEN LOS SYMBOLITOS (ENGRANE ETC) PARA INDICAR EL TIPO DE FUNCTION. POR EJEMPLO LOS EVENT LISTENERS TENDRIAN EL SIMBOLITO DE LA MANITA PARA SIMBOLIZAR HMI



* + 1. Database Queries
* PostgreSQL Queries
* Get Data Generic Function via getRecords()
* Waiting for Asynchronous Requests to Complete

Asynchronous

* + 1. Object Relational Data Mapping
* Nodes to Vertex Shapes (E, I, A, G)
* Connections to Edge Shapes (L)

Only process\_flows are modelled in model, so business logic to determine the line shapes accordingly.

ROBUST SET OF RULES

P&ID Line Shapes:

P - pipe\_line

C – connection\_line

S – signal\_line

D – data\_line

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Source\Target** | **Equipment** | **Instrument** | **Group** | **Arrow** | **Line** |
| **Equipment** | P | P (\*) | P (\*\*) |  | - |
| **Instrument** | P (\*) | D | D (\*\*) |  | (\*) |
| **Group** | P (\*\*) | D (\*\*) | P (\*\*\*) |  | - |
| **Arrow** |  |  |  |  |  |
| **Line** | - | (\*) | - |  | - |

Table 1[QUITAR POR QUE EL LINE TIPE EN REALIDAD SE DEBE ESPECIFICAR EN EL MODEL, NO TENGO QUE HABLAR DE ESTE FALLBACK POR QUE YA VA MAS ALLA DE MI BA]

Special Cases:

\* if Equipment to Instrument to Equipment (Instrument between 2 equipment, short circuit Equipment to Equipment with one single pipe\_line and connect instrument to that pipe\_line with a connection\_line.

\*\* if group to anything or anything to group, connect to group border, but if outermost group, then create a new arrow and connect to this arrow (attention to arrow direction).

\*\*\* use ports so that lines are continuous and don’t appear to break on group borders

* + 1. Graph Layout Algorithm

SEE NEXT CHAPTER

* + 1. Generation of the XML File
* Structure of XML File

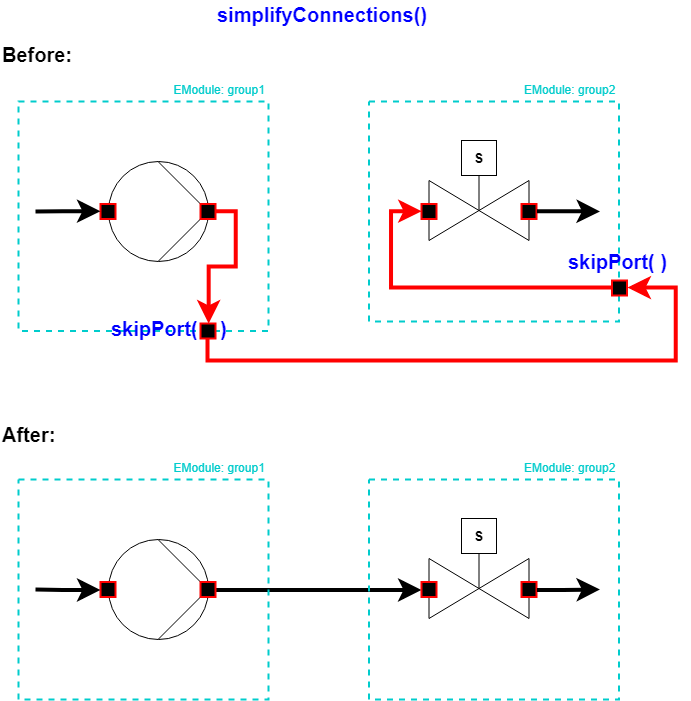
<INSERT CLASS DIAGRAM WITH VERERBUNGEN HIER THAT MODELLS THE STRUCTURE (FOR EXAMPLE MXGRAPH <|-----MXMODEL…

* Recursive Instantiation

1. P&ID Graphing Algorithm
   1. Build Hierarchy
   2. Hierarchy Traversal

Pathfinder in form of post-order depth-first search to find ordered path of node visited while traversing hierarchy.

* 1. Simplification of Edges



* 1. Vertex Placement

Algorithm can be PROGRESSIVELY ENHANCED: allows for incremental

* Overview
* Settings implemented as parameters allow for fine tuning of the algorithm.
* Specification of constraints as tags (loosely coupled to positioning logic)
* Vertex positioning based on constraints/rules
  + 1. Simplifying connections in model

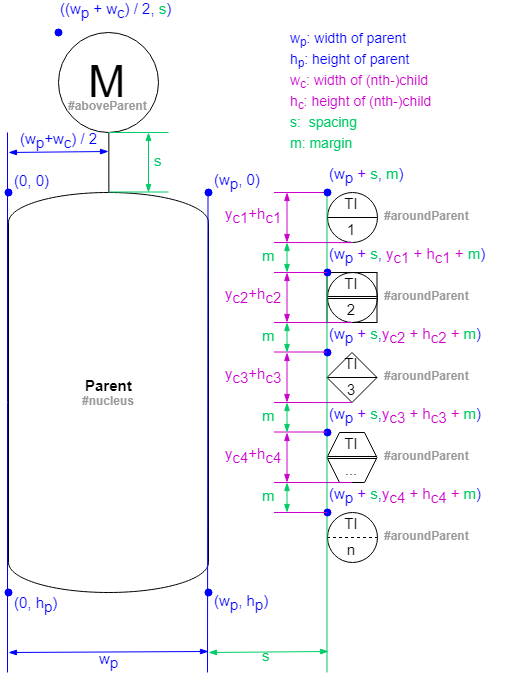


Figure 10 [INSERT CAPTION HERE]

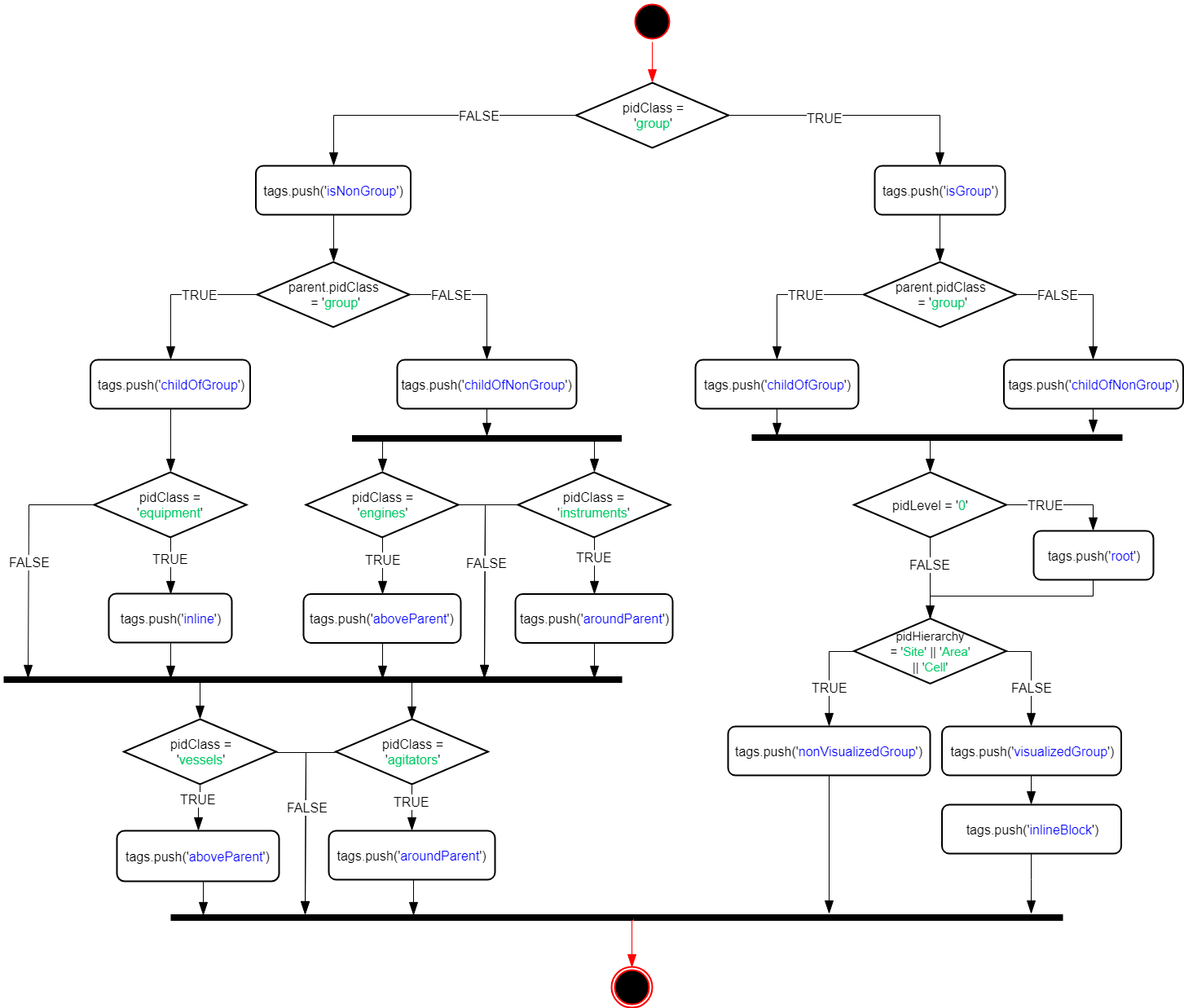


Figure 11 [ACTIVITY DIAGRAM OF SPECIFICATION OF CONSTRAINTS]

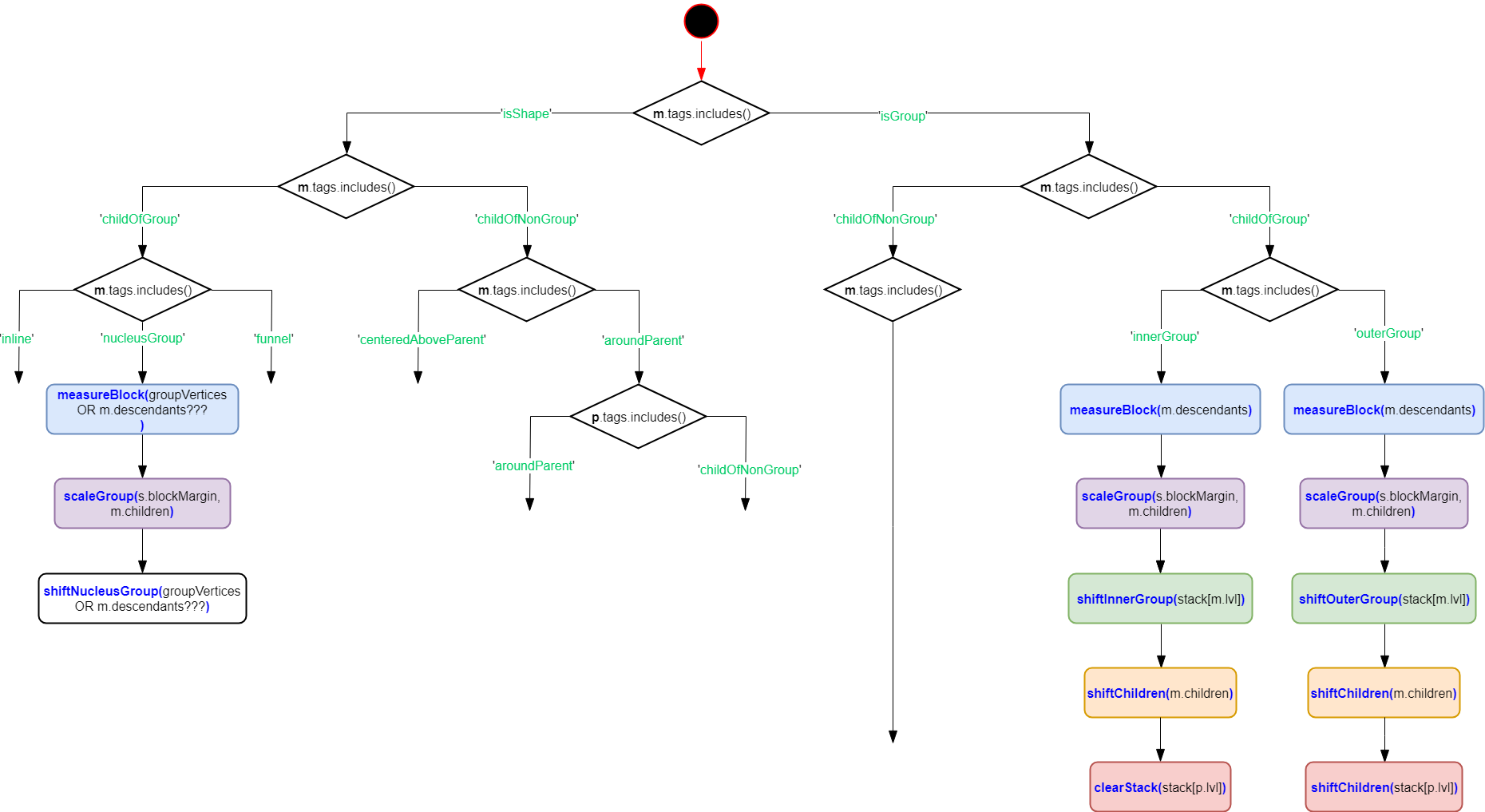
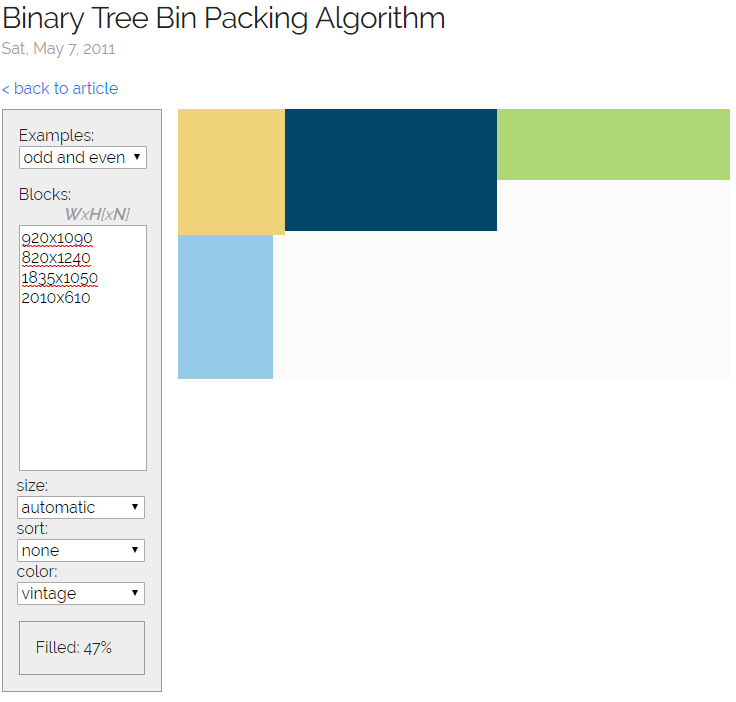
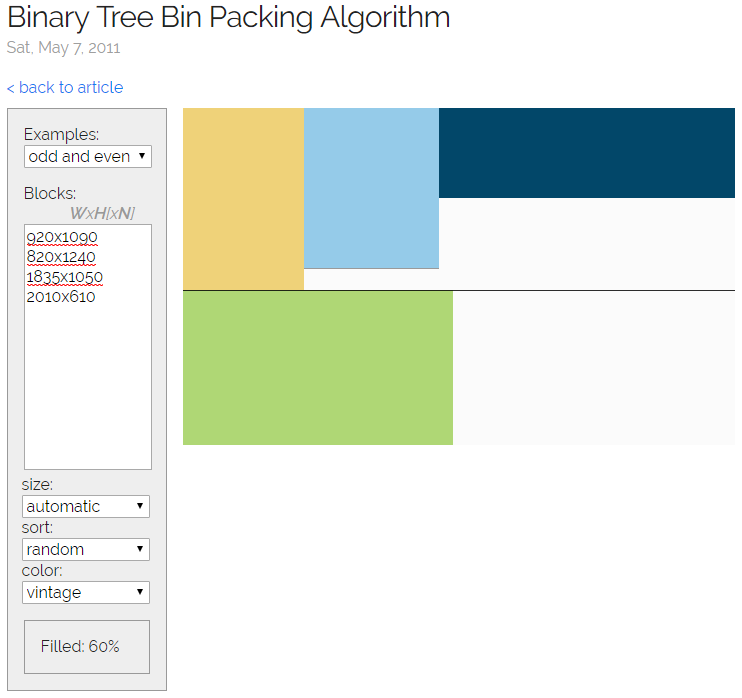
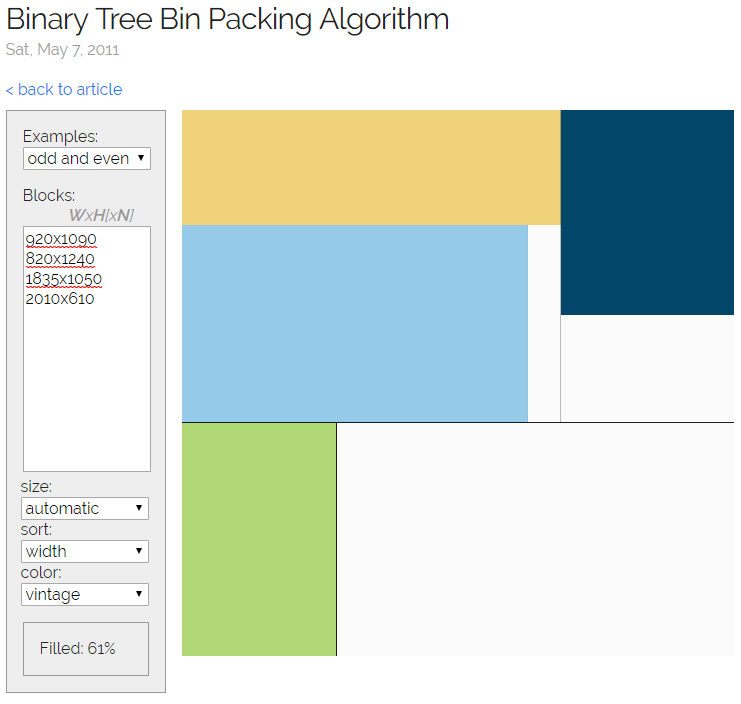
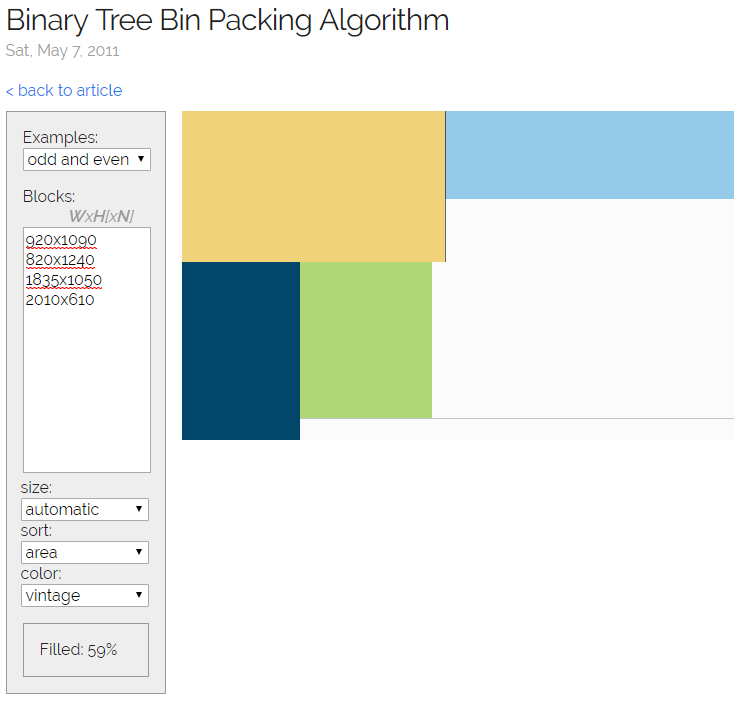
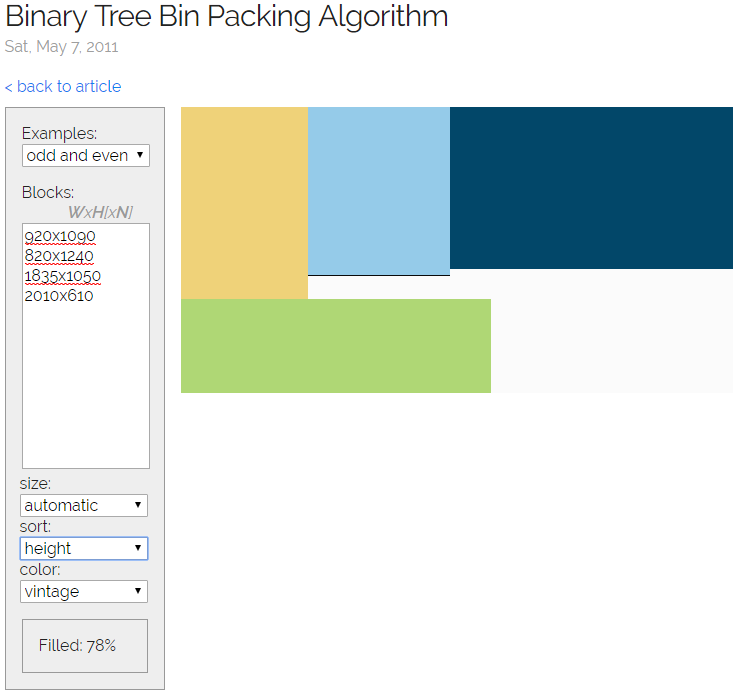
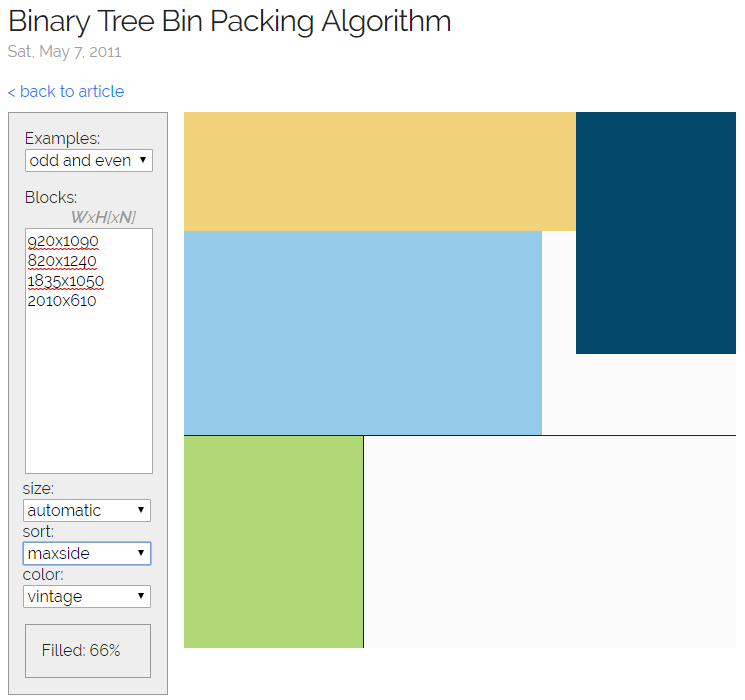


Figure 12[ACTIVITY DIAGRAM OF VERTEX PLACEMENT ALGORITHM WITH CONSTRAINTS]

* Fine Tuning of Parameters
* Group Placemente (#innerGroup, #OuterGroup) algorithm: <https://codeincomplete.com/posts/bin-packing/>
* Orthogonal packing of rectangles in auto-scaling containing group.
* TRIALS WITH THE DEMO WITH ALL SORT SETTINGS: (best results with maxside) RECREATE THE GRAPHICS FROM BELLOW IN DRAW IO
* 

1. Verification, Validation and Evaluation
   1. Definitions

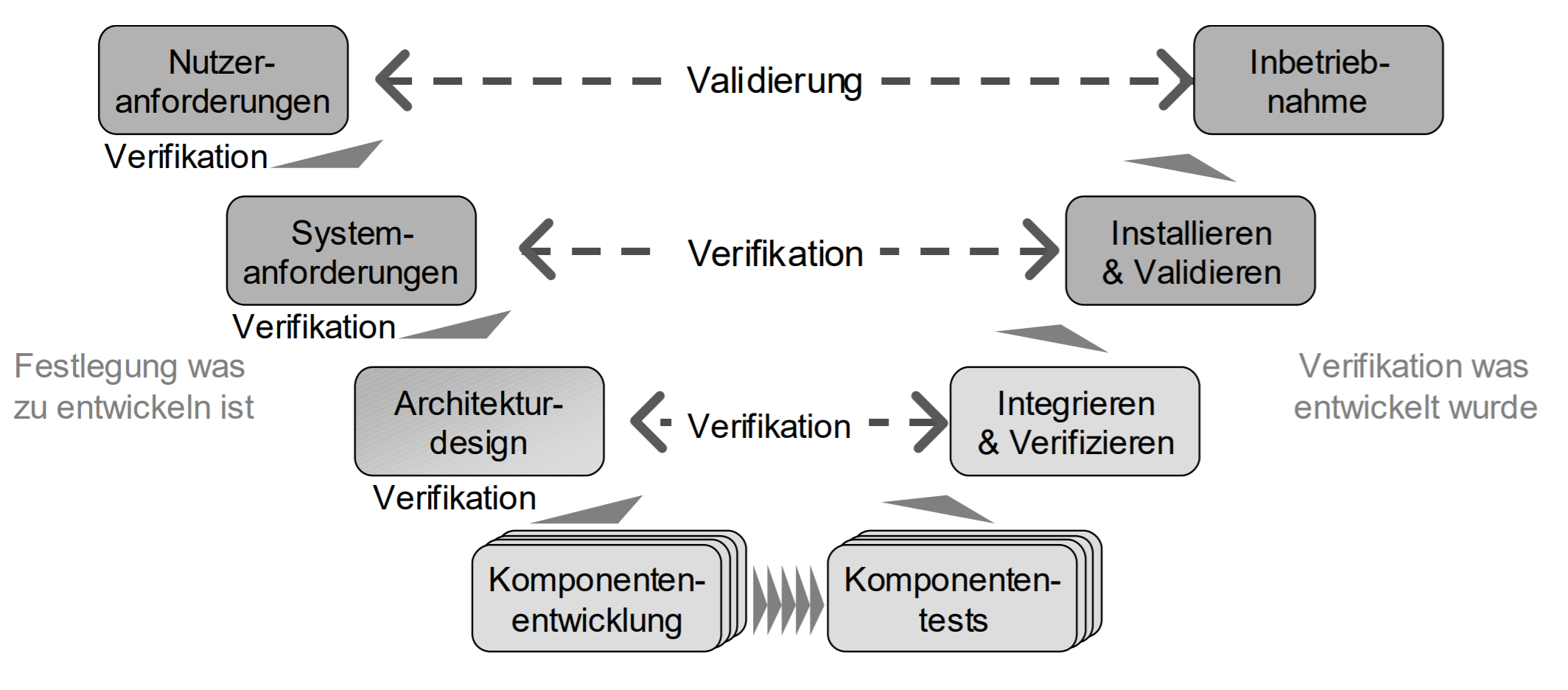


Figure 13 V-Model for Software Verifikation and Validation

* + 1. Verification
    2. Validation
    3. Evaluation
  1. Prototypical Implementation in an Industrial Context

For Comercial Deployment and Industrial Application

1. Synopsis

List of Figures

[Figure 1 [VORLAUFIGE ZIEL DER VISUALISIERUNG] 2](#_Toc519595442)

[Figure 2 [INSERT CAPTION HERE] 18](#_Toc519595443)

[Figure 3 [ACTIVITY DIAGRAM OF SPECIFICATION OF CONSTRAINTS] 19](#_Toc519595444)

[Figure 4[ACTIVITY DIAGRAM OF VERTEX PLACEMENT ALGORITHM WITH CONSTRAINTS] 19](#_Toc519595445)

List of Tables

[Table 1[QUITAR POR QUE EL LINE TIPE EN REALIDAD SE DEBE ESPECIFICAR EN EL MODEL, NO TENGO QUE HABLAR DE ESTE FALLBACK POR QUE YA VA MAS ALLA DE MI BA] 14](#_Toc519595446)

Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Description |
|  |  |
| API | Application Programming Interface |
| GUI | Graphical User Interface |
| SVG | Scalable Vector Graphics |
| MES | Manufacturing Execution System |
| ProcAppCom | Process Application Composer |
| PCS | Process Control System |
| UML | Unfied Modelling Language |
| SysML | Systems Modelling Language |
| UML-PA | Unfied Modelling Language for Process Automation |
| JSON | JavaScript Object Notation |
| XML | Extensible Markup Language |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. Bibliography

[1]

[2]

[3]<https://ac.els-cdn.com/S2405896317314921/1-s2.0-S2405896317314921-main.pdf?_tid=6de94709-7dc6-4269-beb0-fac1338b7c82&acdnat=1532269678_c4b89c7b4e187d7595218b0a1fe60b22>

[4]<https://ac.els-cdn.com/S1474667016434671/1-s2.0-S1474667016434671-main.pdf?_tid=f5714d82-fd23-4d7d-901b-89b13e70dd5b&acdnat=1532269602_0022f42b4de6b59dd533fb07b008d28e>

[5]<https://patentimages.storage.googleapis.com/d2/51/b7/d2fff3397660d9/US20040230328A1.pdf>

[6] <https://pubs.acs.org/doi/pdfplus/10.1021/ie020644a>

[7]

[8]<https://ac.els-cdn.com/S0951832013002433/1-s2.0-S0951832013002433-main.pdf?_tid=86339899-bf4d-4070-b990-5b5512119412&acdnat=1532289234_12ee529d873799f27b7de9a90910e9fb>

[9] <https://link.springer.com/content/pdf/10.1007%2F978-3-642-37057-1_19.pdf>

[10]<https://patentimages.storage.googleapis.com/a9/d3/e4/754ea4055b3871/US20070150821A1.pdf>

[11] <https://slideplayer.com/slide/12488750/> USADO EN EL CAPITULO 2.5.3 2.5.3 (Software Implementation of Graph Visualizations)