HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

KISTLER
measure. analyze. innovate.

Development of a Modelling Procedure for the Description of Test Set-Ups with Measuring Instruments

Project Thesis 1

University of Applied Sciences Rapperswil

Autumn Semester 2016/17

Author:	Mario Meili
Advisor:	Prof. Dr. Farhad D. Mehta
Industry Partner:	Kistler Instrumente AG

Abstract

Measurement techniques are a highly complex field in which small technical details decide over the accuracy and therefore the success or failure of a measurement. What sensors can be used? How can noise be handled properly? These and many more are questions one has to answer when wanting to capture the measurand of interest.

Modelling a test set-up requires experience and is therefore mostly done by experts in the field. The lack of uniform modelling approaches makes this an even bigger problem, because discussing models can lead to confusion when a common language is missing. Furthermore, the validation of a model on different abstraction levels has to be done manually in the absence of the necessary tooling, which is error-prone.

This thesis provides a common formal language for describing test set-ups. It does so by proposing a modelling approach for test set-ups which allows the definition of different abstraction levels on top of a general set of rules. These rules allow the validation of the set-up. For example, it can be validated if components are configured correctly. This was achieved by defining a set of validity constraints in first-order logic. These constraints are then applied to a graph representing the test set-up.

A formal specification of the model is provided and backed by an implementation in the language Prolog.

Acknowledgements

Firstly, I would like to thank Kistler Instrumente AG. In particular, I would like to extend my gratitude to Viola Ehrensperger, David Peter Mueller, Matthias Gwerder and Marco Angliker. Their professional inputs have been invaluable and their critical observation of the project's progress helped me to keep on track.

Secondly, I would like to thank my thesis advisor Prof. Dr. Farhad D. Mehta, who was always willing to lend a helping hand when I ran into problems. Whenever I felt lost, he pushed me in the right direction.

Finally, I would like to express my gratitude to my girlfriend for providing me with unfailing support and continuous encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

AL1 Abstraction Level 1

AL2 Abstraction Level 2

AL3 Abstraction Level 3

UID Unique Identifier

SI International System of Units

ADC Analogue/Digital Converter

JSON JavaScript Object Notation

XML Extensible Markup Language

1. Introduction

Measurement techniques play a very important role in engineering and research. Emerging from the need to compare goods during trading, developing new techniques and instrumentation has become indispensable to, for example, modern production techniques [9]. Therefore, it does not come as a surprise that the field of measurement techniques and instrumentation is highly complex. It requires a lot of technical expertise to take part in the development and design of test set-ups that satisfy the needs of the industry.
Some of the current challenges in the design of test set-ups have been elaborated with Kistler Instrumente AG. These problems and the goals of this thesis are stated in the following sections.

1.1. Problems

Numerous industrial sectors require a high degree of precision and the resulting assurance of quality of their products. This is mainly achieved by the use of measuring systems during or after the production process. The measurement and analysis of physical processes and the monitoring of industrial processes does not only help to improve products, but also supports reducing costs for production, service and maintenance.
The continuously growing palette of sensors and measuring systems, as well as the requirements coming from the industry, result in an increasing complexity of test set-up modelling. The two main problems that were identified and tackled during the period of this thesis are the following:

- **P1: Abstraction Levels:** A complete description of a test set-up contains an extensive amount of information, even for the smallest applications. The larger a set-up becomes, the more difficult it gets to keep an overview of what is important.
  In addition, not every party involved in developing and designing a test set-up brings the same level of expertise to the table, which can complicate the elaboration of requirements the set-up should be able to fulfil. To counter this problem, it should be made possible to model a test set-up on different abstraction levels, revealing the exact amount of information the developer

needs.
The problem hereby lies in the verification of the equivalence of different representations of the same test set-up on different abstraction levels.

- **P2: Model validation:** Having modelled a test set-up is mostly not enough to guarantee the success of the planned measurement. Different sorts of validation have to be run through to ensure that a model fulfils the requirements defined.
  The simplest, but at the same time most important form of validation is to check whether all defined components of a model fit together and fulfil their purpose. Other validations are imaginable, for example, to check if the propagated error through a test set-up does not overstep a certain boundary. Since the sum of all validation problems could probably not even be covered in multiple theses, the scope in this thesis will only be set on the validation of single components and the connections between them.
  The following three examples illustrate, what kind of questions must be answerable:

  1. Are the signal properties for the connection interface in Figure 1.1 valid? Is the custom signal correctly defined?

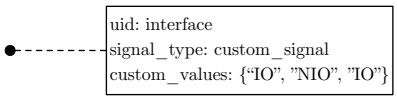

Figure 1.1.: Interface example

  2. Can the connection interfaces shown in Figure 1.2 be linked? Do the interfaces match in terms of their signal properties?

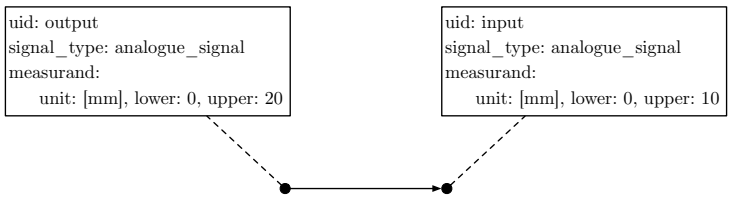

Figure 1.2.: Link example

  3. How can be made sure that the sensor in Figure 1.3 is modelled correctly? Does the incoming signal match the outgoing signal?

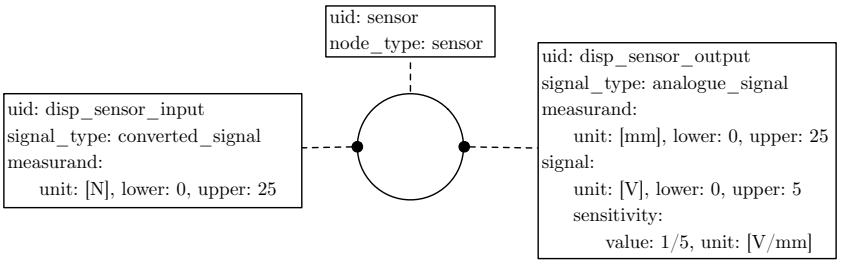

Figure 1.3.: Sensor example

It is not important to fully understand the notation of the examples yet. It will be introduced in 4.2. The importance lies on getting a feeling for validity problems of interest.
It is important to realise that it is not enough to simply specify constraints on model validation, but also to show the feasibility of implementation. Otherwise, automatically validating a given model would not be possible.

1.2. Goals

The goals for this thesis were defined with the intention of tackling the two problems stated in section 1.1 directly. The goals listed in the following are considered to be mandatory:

- **G1:** A modelling system for test set-ups has to be designed and formally specified.
- **G2:** It must be possible to validate models defined with the formally specified modelling system. A means to achieve that has to be part of the modelling system's specification.
- **G3:** The modelling system must be evaluated according to the following criteria:
  - Fulfilment of elaborated requirements (see chapter 3)
  - Extensibility
- **G4:** A functional prototype of the formally specified modelling system has to be implemented.

The last goal shown above is considered optional, since the time period in which this thesis was conducted is limited. However, this does not mean that the importance of achieving this goal has less of an impact than previously stated goals.

1.3. Document Structure

The subsequent chapter 2 will provide necessary background knowledge and discuss related work, especially existing tooling that allows the modelling of test set-ups. Chapter 3 gives a quick overview over the requirements elaborated for the modelling system. A formal specification of the modelling system can be found in chapter 4. In chapter 5, the modelling system is evaluated.
The results of the work conducted are presented in chapter 6. Some concluding thoughts by the author in chapter 7 round off the contents of this thesis.

2. Background & Related Work

The intention behind this chapter is to equip the reader with the knowledge required to fully understand the provided content of this thesis. In order to achieve that, it will firstly be stated how much technical expertise on measurement and instrumentation is necessary. Secondly, a simple but all-encompassing test set-up will be introduced which will periodically be referenced to in ongoing chapters. Lastly, a brief section on related work will depict why the work conducted during this thesis is relevant to solving the stated problems.

2.1. Measurement & Instrumentation

It is important to emphasise at this point that readers of this thesis are not required to be experts in the field of measurement and instrumentation. In fact, the author himself comes from the field of computer science with strong tendencies towards software engineering.
Readers with only little or no expertise, a crash course in measurement and instrumentation is strongly recommended. The majority of technical knowledge in this thesis is based on [9], [10] and [3].
The model specified does therefore not guarantee completeness or even the absence of errors regarding technical aspects of measuring and instrumentation. However, this has no negative impact on the concept of how the model is set up and how it is operating. The focus lies on a general approach of how to solve the problems stated in section 1.1.

2.2. Exemplary Test Set-Up

The test set-up described in this section represents a well known solution to a real world problem. It was selected by the author for its specific characteristics which will be further elaborated in the following description. The set-up is part of a show room at Kistler Instrumente AG and therefore all the specific parts and devices are manufactured by Kistler Instrumente AG. However, it is important to clarify that the set-up could have been built using any other brand of instruments. Note that specific property values can vary in upcoming examples.

In measurement techniques it is quite common to set two or more measurands in relation to each other. In doing so, it is possible to analyse and evaluate physical processes, for example by studying a X-Y-diagram. The test set-up shown in Figure 2.1 does exactly this. It measures the press force introduced via a hand press to different items. In addition to the press force, the displacement of the hand press gets measured.

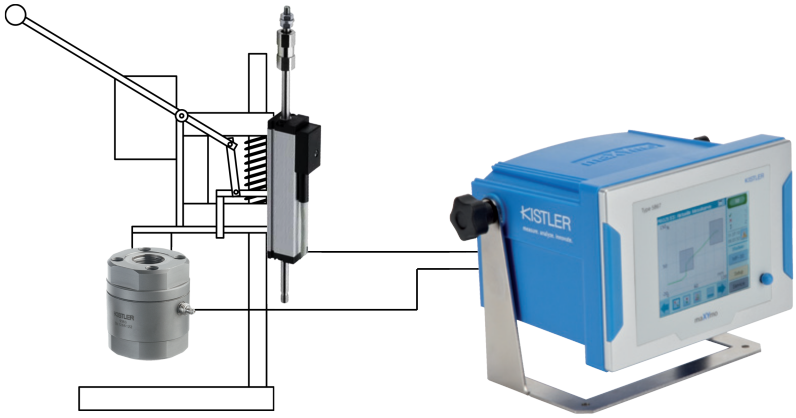

Figure 2.1.: Press force over distance test set-up

Each cycle is visualised on an X-Y-diagram, where X is the measured displacement in [mm] and Y represents the press force in [N]. Note that this is always a closed cycle since both the press force and the displacement return to their initial state once the lever is completely released.
This set-up is also able to use a predefined acceptance area on the diagram with which it can decide if a measured result is acceptable or not. A possible use case is therefore manufacturing processes where the measurement can automatically detect parts of insufficient quality and throw these parts out of the assembly line as early as possible.

2.2.1. The Press Force Sensor

For measuring the press force, a Type 9323AA sensor was used. The following technical data stems from the sensor's data sheet which is publicly accessible on the homepage of Kistler Instrumente AG [6]. Further referencing is omitted.
The sensor generates an electric charge that is proportional to the force acting on its quartz element. This electrical charge can be measured in [pC]. To be able to translate this into [N], the sensitivity of the sensor has to be known, which is -9.6 $[\frac{\text{pC}}{\text{N}}]$. The maximum measuring range spans from 0 to 10 [kN]. Moreover, there

are three measuring ranges calibrated: 100% which represents the maximum range, 10% from 0 to 1 [kN] and 1% from 0 to 0.1 [kN].
The sensor's characteristics regarding its form, dimension and components materials are regarded as details that are not relevant for the modelling process. Other characteristics such as the sensor's behaviour according to changes in its environment (temperature, pressure, ...), although important for a satisfactory outcome of a measurement, are beyond the scope of this thesis.

2.2.2. The Position Sensor

For measuring the displacement of the hand press, a Type 2120A sensor—specifically a model T25—was used. Again, the technical data stems from the sensors data sheet [5].
The electrical signal generated by the sensor is proportional to the displacement of the sensor to its resting state. This is achieved by applying an electric current to the sensor and measuring the changing resistance when the distance between input and output increases or decreases. The nominal resistance of this specific sensor is 1 [kΩ]. The defined electrical range spans up a measurand range from 0 to 25 [mm].

2.2.3. Signal Analysis

Both sensors in this set-up are connected to a maXYmos Type 5867. This device is used to monitor and evaluate XY curves of two measurands that are in a certain relationship to each other [4]. The following data and Figures 2.2 and 2.3 stem from the device's data sheet [4].
Figure 2.2 shows the internals of maXYmos. Channel-X as well as Channel-Y support ±10 [V]-signals as inputs instead of the piezo- or potentioelectric ones. However, this feature is not relevant for this example.
Before the signals are converted into an analogue form, they usually get amplified first. This is necessary when the signal output level of a sensor is considered to be too low [9].
In addition, the maximum and minimum amplitude of the incoming signals need to be known so that the outgoing amplitude range can be calculated and used to check compatibility with further signal processing elements.
At some point, the so far analogue signal has to be translated into a digital signal in order for it to be displayed on a digital screen. An Analogue/Digital Converter (ADC) is necessary to perform this task. The output signal of the ADC depends strongly on the input signal's characteristics, but also on the sampling rate and the resolution used to represent the outgoing signal.

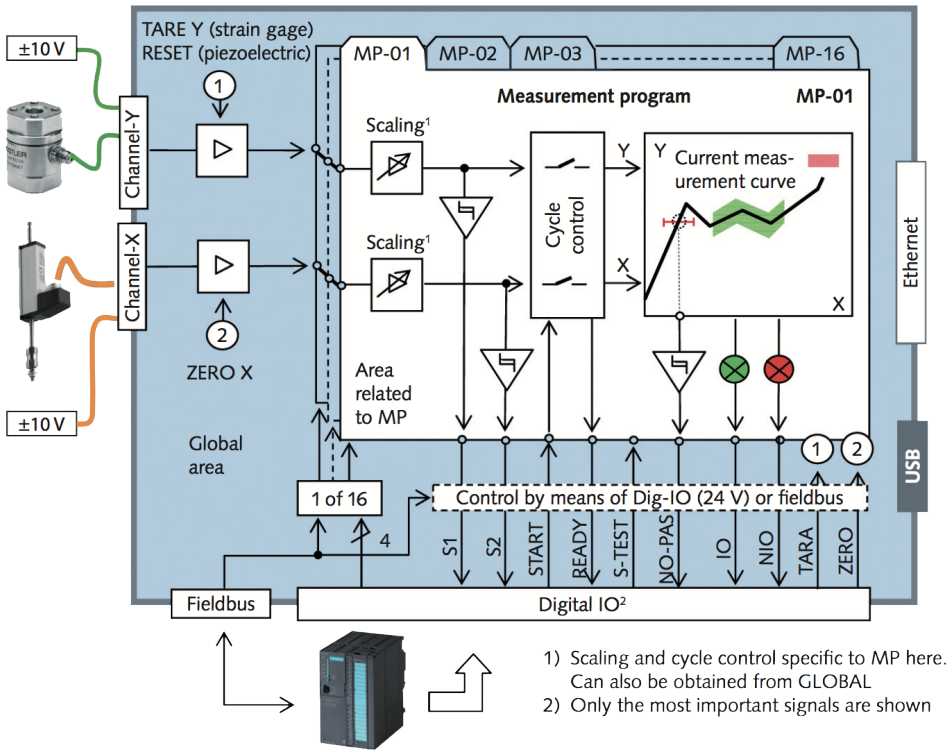

Figure 2.2.: Internals of maXYmos

The display output shows the two converted signals on the `X`-axis and the `Y`-axis of a `X-Y`-diagram respectively. This is shown in Figure 2.3.

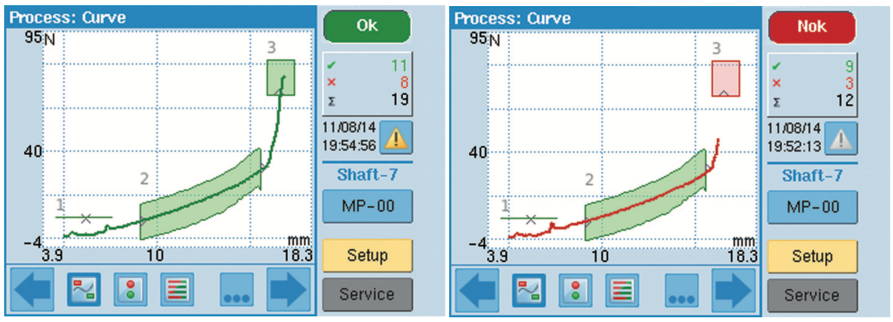

Figure 2.3.: `X-Y`-diagram examples

Next to the bare output, the maXYmos device also performs an analysis on the measured signals. The user is able to define an area in which the curve has to lay in. If successful, the result can be distributed over a digital output. In the case of maXYmos, the output is simply `IO` or `NIO`.

2.2.4. What Is Important Here?

As stated earlier in this section, the reader is not expected to fully understand every detail of the described example. The purpose of introducing the set-up this early on in the documentation is solely to give a brief overview and to shape a picture of what is to be discussed in later chapters.

2.3. Related Work & Modelling Tools

This section lists related work and existing modelling tools, mainly to emphasise the justification of the problems stated in section 1.1. Because of the limited time and the vast amount of literature about measurement and instrumentation, the author does not claim completeness on the following listings.

2.3.1. Related Work

During the course of this thesis, a number of attempts to find relating literature have been conducted. Most publications regarding measurement and instrumentation cover very technical proposals for the improvement of instrumentation and controlling of measurements. However, not a single publication covering the modelling of test set-ups could be found.
It might appear as if the problems stated in section 1.1 are not relevant, but in the author's opinion and the opinion of Kistler Instrumente AG this is not the case. The speculation is that the focus in the field of measurement techniques still lies mostly on the improvement of instrumentation and this is why the development of software and tooling for the modelling process of test set-ups does not currently receive a lot of attention.

2.3.2. Existing Modelling Tools

There are several tools available which allow the modelling of test set-ups, at least to some extent. These tools are listed below along with an evaluation of how good they solve the problems stated in section 1.1:

- LabView [2]
- DASYLab [1]
- ATEasy [7]

All of these tools allow the acquisition, analysis and manipulation of measurement data, the controlling of measurement instrumentation and automated testing

systems. The primary use case of these software solutions is to be integrated in actual set-ups, grouping a lot of signal processing and analysis functionality. Since all the tools are working on the level of signal processing, it takes a lot of time and technical expertise to become familiar with them. None of these tools support modelling test set-ups.

3. Requirements

In the process of developing the requirements for the modelling system to be built, three levels of abstraction have been defined. Each of these abstraction levels describes the test set-up to be modelled in an increasing amount of detail. The levels have been numbered from 1 to 3, where Abstraction Level 1 (AL1) is the most general and Abstraction Level 3 (AL3) is the most detailed abstraction level. It is noteworthy that each level depends on all levels with numbers smaller than their own number, for example, Abstraction Level 2 (AL2) depends on AL1 while AL3 depends on AL2 and AL1 respectively.
The following sections describe each level by briefly summarising their corresponding requirements. The complete set of requirements can be found in appendix A. To improve readability, objects defined on AL2 are further called components and objects defined on AL3 are called parts respectively. In multiple requirements, conditions for determining model validity are described. Where this is not the case, validity is given if all attributes have been defined according to the requirement description.

3.1. AL1 Requirements

This highest level of abstraction concentrates on the bare minimum that is necessary to describe a test set-up. It leaves out all details about components, devices and any kind of processes and instead focuses solely on the given measurands and desired outputs.
Three different kinds of inputs are required to model a test set-up. Next to the standard inputs described in Table A.1, the requirements for audio and video inputs can be found in Table A.2 and A.3. The main difference between these inputs is that both the audio and video input cannot be described by a measurand and a range, whereas standard inputs are defined exactly that way. Hence, the attributes for a measurand and its corresponding range are omitted for audio and video inputs. The validity requirement on these different input types is another important difference worth mentioning. For audio and video inputs it suffices to assign a Unique Identifier (UID), whereas the standard input requires that the measurand and range attributes properly match in addition.
The requirement on a standard output is shown in Table A.4. In some cases, it

is desirable to define a custom output instead of just a measurand and its range. Table A.5 shows how these outputs can be described. This type of output can represent digitally computed outputs like yes or no decisions or rankings of quality (for example {"bad", "ok", "good"} or {1,2,3,4,5}) and so on. To be able to model visual outputs like charts or diagrams that can be displayed on a screen, it must be possible to group outputs and assign special meaning to them. Table A.6 describes the requirement to achieve this behaviour.
For a model to be valid on AL1, all defined inputs and outputs must be valid themselves. This is stated in Table A.10. The complete listing of requirements regarding AL1 can be found in section A.1.

3.2. AL2 Requirements

This intermediary level of abstraction divides the test set-up into three process oriented parts: sensors needed to match the inputs defined in AL1, measuring modules and analysis components. Although this level of abstraction allows to model on a device level, it is important to emphasise that the components defined do not have to represent existing physical devices.
Components on AL2 can be sensors, audio and video recorders, measuring modules and analysis components. Mandatory properties and validity constraints for these components are defined in Tables A.11 to A.15. While for sensors, the validity depends on the matching between its measurand and unit property, measuring modules and analysis components must guarantee that their outputs can be derived from the inputs defined on them.
On this abstraction level, a lot of constraints about every valid linking between components have been elaborated. The requirements for these link constraints can be found in Tables A.16 to A.29. Mostly, these constraints include the information, which component can be paired with which other component or ensures that the properties of linked components do match in terms of measurands, units and ranges. In Table A.30, it is defined how many links are allowed between components of AL2 and inputs or outputs of AL1.
For a model to be valid on AL2, the corresponding model on AL1 must also be valid. In addition, all defined components and links must fulfil their validity constraints. This is stated in Table A.31. The complete listing of requirements regarding AL2 can be found in section A.2.

3.3. AL3 Requirements

This lowest level of abstraction reaches one step further and already comes close to actual hardware. It describes the basic parts a measuring device is built with. It connects the measuring problem with the actual technical aspects of the test set-up. Within this level of detail, it is for example possible to determine the compatibility of chosen components on AL2 when existing devices are being modelled or even to define new, so far non-existing devices.
On this level of abstraction, parts of a measuring module can be defined. These include amplifiers, analogue-digital converters, filters and triggers. The mandatory properties for these parts are defined in Tables A.35 to A.38. For example, when modelling an amplifier, the scale factor for the amplification has to be set. An analogue-digital converter requires the definition of a resolution and a sampling rate with which the incoming signal gets sampled.
As was the case for requirements for AL2, constraints for the link validity between parts among themselves and between parts and components have been defined and can be found in Tables A.39 to A.42.
What is new on AL3, is that for each input and output of components and parts, physical signal properties have to be defined. So far, only measurands with their corresponding ranges have been considered. Now the underlying signal representing the measurand has to be defined as well. The properties of analogue and digital signals can be found in Tables A.32 and A.33. How the assignment of signal properties to components of AL2 has to be handled is stated in Table A.34.
Finally, AL3 model validity is described in Table A.43. The complete listing of requirements regarding AL3 can be found in section A.3.

3.4. Overall Model Requirements

The last requirement defined in Table A.44 states that for the overall model to be valid, the representative model defined on each abstraction level has to be valid. The requirement can be found in section A.4.

4. Model Specification

In the scope of this thesis, one model fulfilling the requirements of chapter 3 has been developed and is specified in this chapter. Firstly, an overview of the approach is discussed. Then, some definitions and a formal specification are given.

4.1. Overview

The overall idea of this modelling approach is to represent the test set-up by a graph where every component is modelled as a node with ports and every link is modelled as a directed edge between ports. In addition, properties on nodes and ports can be defined. The nature of possible properties is further elaborated in section 4.4. The domain model for this base model is illustrated in Figure 4.1.

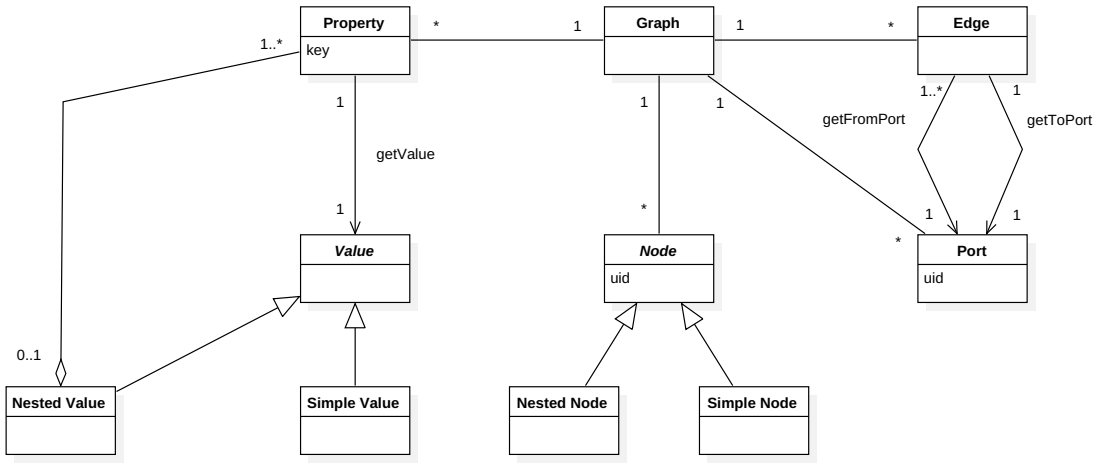

Figure 4.1.: Domain model for the base graph model

Note that every node and every port has a unique identifier, so that they are distinguishable across several graphs representing different abstraction levels of the same test set-up. Edges can be uniquely identified by their tuple of ports, one of which represents the origin and the other the destination of the edge. Properties can be thought of as key-value pairs, where the values themselves can be either

simple values or properties again. The difference between nested and simple nodes will become clear in section 4.4.
Specific components and parts of the different abstraction levels are modelled as specialisations of simple and nested nodes as shown in Figure 4.2. Note that the two domain models are actually just one that has been separated for the sake of readability. The abstract class *`Node`* and the classes `Simple Node` and `Nested Node` are the same in both models.

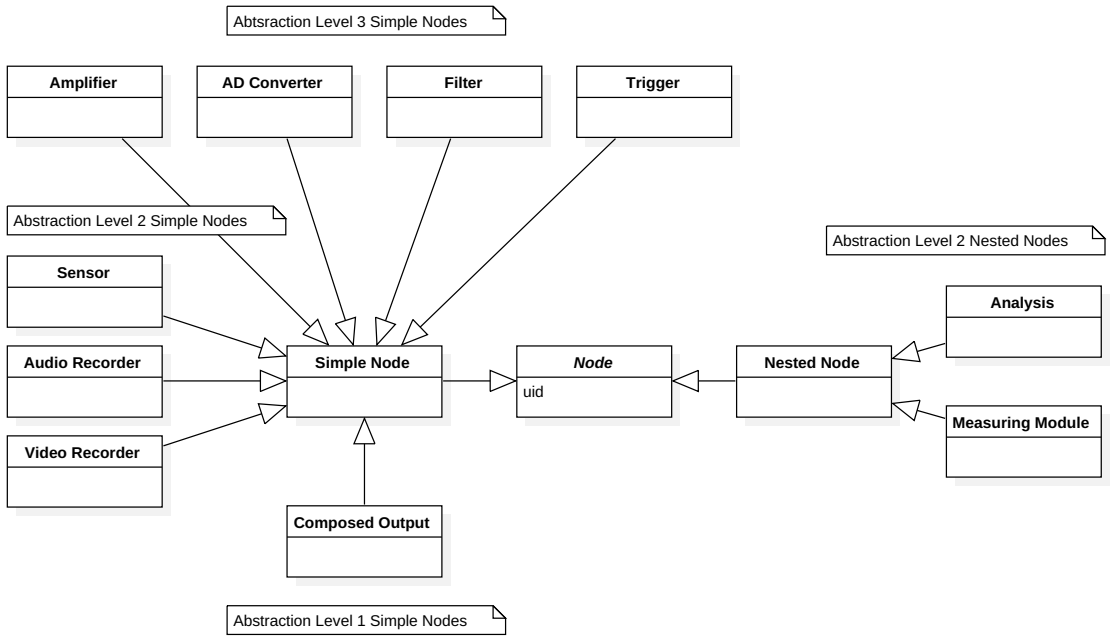

Figure 4.2.: Domain model for specific nodes of abstraction levels

For the sake of clarity, the different nodes have been arranged in groups according to their level of abstraction. This is also marked with a comment.
It is important to point out that the graphs defined for each abstraction level have to represent the same test set-up. Graphs of more detailed abstraction levels can therefore only enrich their corresponding graphs on higher abstraction levels with more detailed information. This is formally specified in subsection 4.5.6.

4.2. Visualisation

It is possible to describe a graph by listing all of its components and properties. The result could then for example be formatted as is done with JavaScript Object Notation (JSON) or Extensible Markup Language (XML) code. The readability of this representation diminishes with increasing size of the graph.

In order to aid readability, the visual elements defined in Figure 4.3 will be used instead.

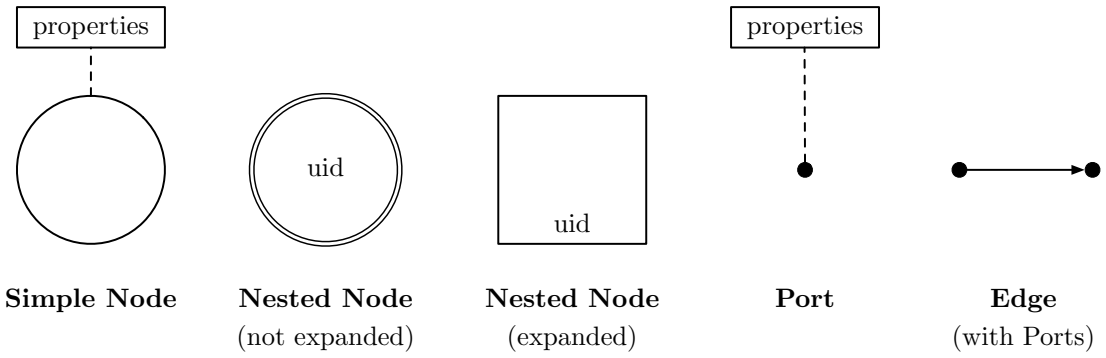

Figure 4.3.: Visualisation elements

In upcoming examples, graphs will be visualised following this definition. Figure 4.4 shows how the elements are composed.

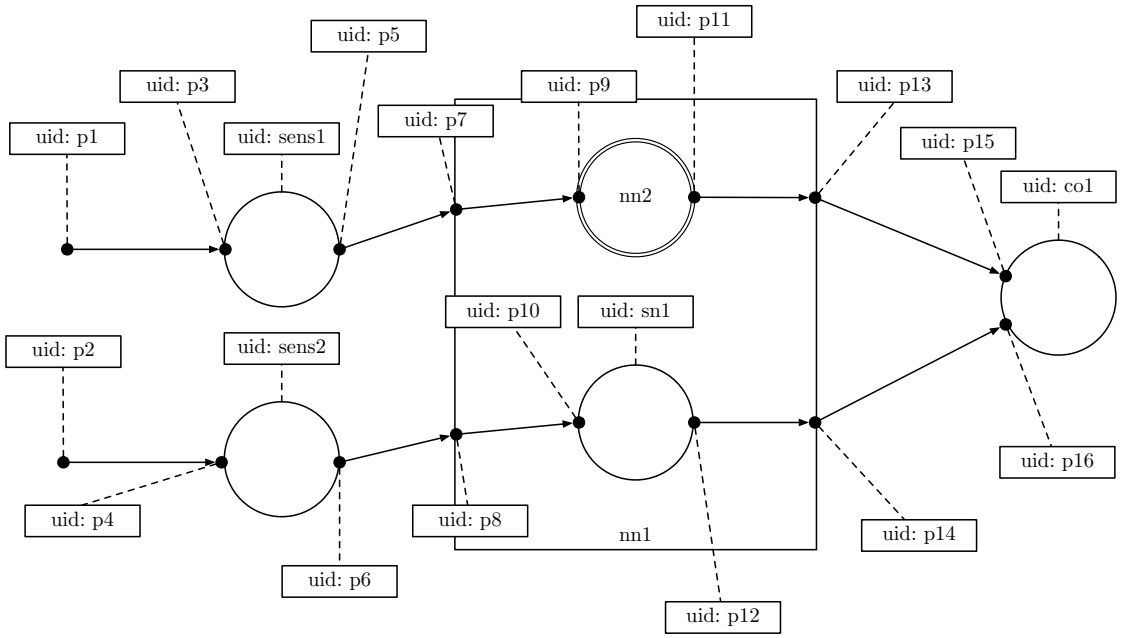

Figure 4.4.: Visualisation example

Ports can either stand for themselves or be part of a node's input or output port set. Edges do always connect ports with each other, never directly to nodes. An expanded nested node contains a graph, that is only connected to the node's input and output ports. Properties other than the unique identifiers of elements are left out in this example for simplicity.

4.3. Definitions & Syntax

For the sake of simplicity and readability, the following definitions are used in the specifications described in the subsequent sections:

- G: $\{g \mid g := (N, E, Po, Pr)\}$
  The set of possible graphs defined by a 4-tuple (N, E, Po, Pr). g_i denotes a graph defined on abstraction level i.
- N: Set of nodes n.
- Po: Set of ports p.
- E: Set of edges $e := (p_i, p_j)$, where $p_i, p_j \in Po$ and $E \subset Po \times Po$.
- Pr: Set of properties $prop$.

Most properties are only accessible via the set of properties Pr. The following functions are defined to ease the use of properties:

- $getKey(prop) : Property \rightarrow String$
  Access the key of a property $prop$.
- $getValue(prop) : Property \rightarrow String$
  Access the value of a property $prop$.

However, there are a few exceptions, where properties are defined directly on the corresponding element of the graph. The following functions are defined to denote the access of these properties:

- $getUid(n) : Node \rightarrow String$
  Access the unique identifier of a node n.
- $getUid(p) : Port \rightarrow String$
  Access the unique identifier of a port p.
- $getFromPort(e) : Edge \rightarrow Port$
  Access the port p_f of an edge (p_f, p_t).
- $getToPort(e) : Edge \rightarrow Port$
  Access the port p_t of an edge (p_f, p_t).

4.4. Properties

In the previous sections of this chapter it became clear that some sort of properties have to be defined and that the combination thereof forms the set Pr which is needed to define a graph. This section lists all mandatory and optional properties that are needed to define the validity constraints of section 4.5.

4.4.1. Port Properties

For a port p, the following nested structures of properties can be defined. The '_' symbol denotes a placeholder for any value fitting the key of the property.

- $(getUid(p), \{(\text{"}signal\_type\text{"}, \_)\})$
- $(getUid(p), \{(\text{"}signal\_type\text{"}, \_),$
  $(\text{"}measurand\text{"}, \{(\text{"}lower\text{"}, \_), (\text{"}upper\text{"}, \_), (\text{"}unit\text{"}, \_)\})\})$
- $(getUid(p), \{(\text{"}signal\_type\text{"}, \_),$
  $(\text{"}measurand\text{"}, \{(\text{"}lower\text{"}, \_), (\text{"}upper\text{"}, \_), (\text{"}unit\text{"}, \_)\}),$
  $(\text{"}signal\text{"}, \{(\text{"}lower\text{"}, \_), (\text{"}upper\text{"}, \_), (\text{"}unit\text{"}, \_),$
  $(\text{"}sensitivity\text{"}, \{(\text{"}value\text{"}, \_), (\text{"}unit\text{"}, \_)\})\})\})$
- $(getUid(p), \{(\text{"}signal\_type\text{"}, \_),$
  $(\text{"}measurand\text{"}, \{(\text{"}lower\text{"}, \_), (\text{"}upper\text{"}, \_), (\text{"}unit\text{"}, \_)\}),$
  $(\text{"}signal\text{"}, \{(\text{"}lower\text{"}, \_), (\text{"}upper\text{"}, \_), (\text{"}unit\text{"}, \_),$
  $(\text{"}sensitivity\text{"}, \{(\text{"}value\text{"}, \_), (\text{"}unit\text{"}, \_)\}),$
  $(\text{"}resolution\text{"}, \_), (\text{"}sampling\_rate\text{"}, \_)\})\})$

What properties can be defined and combined depends on the port's signal type, but also on the abstraction level the port is defined on.

4.4.2. Node Properties

For a simple node n, the following properties can be defined. Again, the '_' symbol denotes a placeholder for values.

- $(getUid(n), \{(\text{"}node\_type\text{"}, \_), (\text{"}input\_ports\text{"}, \_), (\text{"}output\_prots\text{"}, \_)\})$
- $(getUid(n), \{(\text{"}node\_type\text{"}, \_), (\text{"}input\_ports\text{"}, \_), (\text{"}output\_prots\text{"}, \_),$
  $(\text{"}scale\_factor\text{"}, \_)\})$
- $(getUid(n), \{(\text{"}node\_type\text{"}, \_), (\text{"}input\_ports\text{"}, \_), (\text{"}output\_prots\text{"}, \_),$
  $(\text{"}resolution\text{"}, \_), (\text{"}sampling\_rate\text{"}, \_)\})$

- $(getUid(n), \{(\text{"}node\_type\text{"}, \_), (\text{"}input\_ports\text{"}, \_), (\text{"}output\_prots\text{"}, \_),$
  $(\text{"}range\text{"}, \{(\text{"}lower\text{"}, \_), (\text{"}upper\text{"}, \_)\})\})$
- $(getUid(n), \{(\text{"}node\_type\text{"}, \_), (\text{"}input\_ports\text{"}, \_), (\text{"}output\_prots\text{"}, \_),$
  $(\text{"}main\_port\text{"}, \_)\})$

For nested nodes and the specialisations measuring module and analysis, the nested properties must be of the following form:

- $(getUid(n), \{(\text{"}node\_type\text{"}, \_), (\text{"}input\_ports\text{"}, \_), (\text{"}output\_prots\text{"}, \_),$
  $(\text{"}graph\text{"}, \_)\})$

4.5. Validity Constraints

This section describes the base validity constraints on modelled graphs of every abstraction level. This means that these constraints have to be valid for every defined graph and at all times. The constraints will be defined using formulae in first-order logic. The numbering of the constraints follows the corresponding numbering of the complete set of constraints in appendix B.
The validity of a graph depends on the validity of its ports, edges, nodes and properties. This can be defined using the *ValidGraph* relational predicate which is defined using the following predicate:

$$\begin{aligned} \mathit{ValidGraph}((N, E, Po, Pr)) \Leftrightarrow & \mathit{ValidPorts}(Po, Pr, E) \\ & \wedge \mathit{ValidEdges}(E, Pr) \\ & \wedge \mathit{ValidNodes}(N, Pr, E) \\ & \wedge \mathit{ValidProperties}(Pr) \end{aligned} \tag{1}$$

It is important to note that the sets N, E, Po, Pr can not be checked separately. For example, the set Pr is needed for every one of the newly introduced predicates *ValidPorts*, *ValidEdges*, *ValidNodes* and *ValidProperties*. *ValidPorts* and *ValidNodes* do even need the set E to be able to check validity.
In the following subsections, the four new predicates will be formally defined.

4.5.1. Port Validity

The *ValidPorts* predicate is true if and only if the validity constraints for every single port are fulfilled. The definition of the predicate is given in the following:

$$ValidPorts(Po, Pr, E) \Leftrightarrow \forall p \in Po \ \cdot ValidPort(p, Pr, E) \tag{2}$$

The *ValidPort* predicate is a conjunction of structural statements and the *ValidPort-Properties* predicate. The structural statements left out in the following definition concern for example the number of incoming and outgoing edges to the port p.

$$\begin{aligned} ValidPort(p, Pr, E) \Leftrightarrow & \\ & \vdots \\ & /*\text{structural constraints}*/ \\ & \vdots \\ & \wedge\ ValidPortProperties(getValue(prop_t), prop_p) \end{aligned} \tag{3}$$

What can be seen is that a port's validity depends on the validity of its properties. The predicate *ValidPortProperties* is defined differently depending on the signal type assigned to the port. This is the point where the constraints become specific for specific properties and no more port validities can be generalised. The complete definition *ValidPort* can be found in appendix B.
In the following, one definition of *ValidPortProperties* is given as a representative:

$$\begin{aligned} ValidPortProperties(\text{"}custom\_signal\text{"}, props) \Leftrightarrow\ & (|props| = 2) \\ & \wedge\ ((\text{"}custom\_values\text{"}, values) \in props) \\ & \wedge\ \forall value_1 \in values \cdot \\ & \quad \neg(\exists value_2 \in values \cdot value_1 = value_2) \\ & \wedge\ (|values| \geq 1) \end{aligned} \tag{10}$$

This definition of *ValidPortProperties* must be true for all ports with "*custom_signal*" as their assigned signal type. Looking back at the first example given in section 1.1, it can now be stated that the port defined in Figure 1.1 is not valid. Figure 4.5 highlights in red, what violates the predicate.

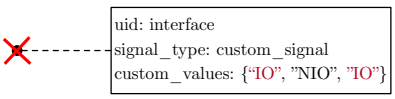

Figure 4.5.: Interface example resolution

The set of custom values contains twice the value "*IO*", which is not allowed. The predicate fails when trying to evaluate lines 3 and 4.
For the sake of brevity, all other definitions of *ValidPortProperties* have been moved to appendix B.

4.5.2. Edge Validity

The *ValidEdges* predicate summarises the validity constraints for every edge. The definition of the predicate is therefore straightforward:

$$ValidEdges(E, Pr) \Leftrightarrow \forall e \in E \cdot ValidEdge(e, Pr) \tag{13}$$

The *ValidEdge* predicate checks if for each property of the input-port there exists a matching property on the output-port.

$$\begin{aligned} ValidEdge((p_f, p_t), Pr) \Leftrightarrow\ & (\exists(key_f, value_f) \in Pr \cdot key_f = getUid(p_f)) \\ & \wedge\ (\exists(key_t, value_t) \in Pr \cdot key_t = getUid(p_t)) \\ & \wedge\ MatchingProperties(value_f, value_t) \end{aligned} \tag{14}$$

$$\begin{aligned} MatchingProperties((props_f, props_t)) \Leftrightarrow\ & \forall(key_f, value_f) \in props_f \cdot \\ & \exists(key_t, value_t) \in props_t \cdot \\ & MatchingProperty((key_f, value_f), (key_t, value_t)) \end{aligned} \tag{15}$$

As was the case with *ValidPortProperties* in subsection 4.5.1, *MatchingProperty* has more than one definition depending on the property given as arguments to the predicate. As representative, the definition for the property "*upper*" was selected, whose value represents the upper boundary of a measurand or underlying signal.

$$\begin{aligned} MatchingProperty(("upper", value_f), (key_t, value_t)) \Leftrightarrow\ & (key_t = "upper") \\ & \wedge\ (value_f \leq value_t) \end{aligned} \tag{20}$$

When looking back at the second example given in section 1.1, it can now be determined that the edge defined in Figure 1.2 does not fulfil the *MatchingProperty* predicate defined above. The upper boundaries of the measurand ranges on both

ports at the end of the edge are highlighted in red in Figure 4.6. Clearly, 20 is larger than 10, which violates line 3 of the predicate.

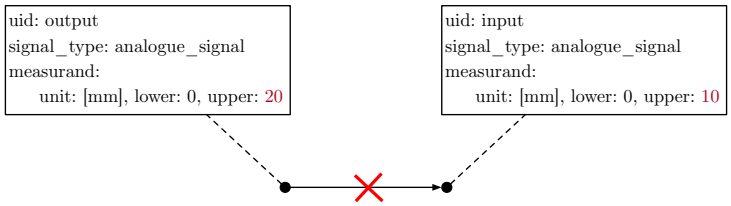

Figure 4.6.: Link example resolution

Again, the complete set definitions for all *MatchingProperty* predicates can be found in appendix B.

4.5.3. Node Validity

The *ValidNodes* predicate's definition is quite similar to *ValidPorts* and *ValidEdges* defined in previous subsections. It is given in the following:

$$ValidNodes(N, Pr, E) \Leftrightarrow \forall n \in N \cdot ValidNode(n, N, Pr, E) \tag{21}$$

The *ValidNode* predicate is composed of multiple structural statements, all conjuncted with the *MatchingPorts* predicate. The complete definition can be found in appendix B.

$$\begin{aligned} &ValidNode(n, N, Pr, E) \Leftrightarrow \\ &\quad \vdots \\ &\quad \text{/*structural constraints*/} \\ &\quad \vdots \\ &\wedge\ MatchingPorts(n, type, inputs, outputs, Pr) \end{aligned} \tag{22}$$

The structural constraints mainly concern connection constraints for input and output ports, but also verify that a node's ports are distinct from the ports of any other nodes. The definition of *MatchingPorts* depends on the type of the node under inspection. That is why there are multiple definitions of this predicate. One exemplary definition of *MatchingPorts* is given in the following. It represents the type specific validity constraint for sensor nodes.

$$
\begin{aligned}
&MatchingPorts(\_, "sensor", \{input\}, \{output\}, Pr) \Leftrightarrow \\
&\quad ((getUid(input), values_i) \in Pr) \\
&\wedge\ (("signal\_type", "analogue\_signal") \in values_i) \\
&\wedge\ (("measurand", \_) \in values_i) \\
&\wedge\ (|values_i| = 2) \\
&\wedge\ ((getUid(output), values_o) \in Pr) \\
&\wedge\ (("signal\_type", "analogue\_signal") \in values_o) \\
&\wedge\ (values_i \subset values_o)
\end{aligned}
\tag{26}
$$

The third example given in section 1.1 can now be validated. The sensor node defined in Figure 1.3 is not valid. In Figure 4.7, the signal type of the input port of the sensor is highlighted in red. That this is not a valid signal type for a sensor input is determined by the *MacthingPorts* predicate above on line 3.

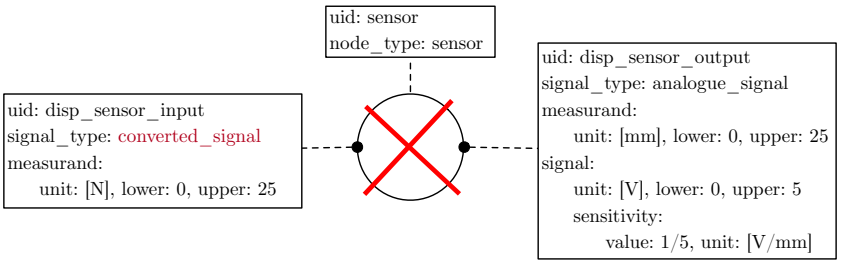

Figure 4.7.: Sensor example resolution

Other definitions of *MatchingPorts* can be found in B.

4.5.4. Property Validity

The set of properties Pr is valid, if and only if every key of the top level key-value pairs occurs at most once. This prevents the definition of multiple properties for the same element. An example is not necessary here because the predicate is regarded as self-explaining.

$$
\begin{aligned}
&ValidProperties(Pr) \Leftrightarrow \forall (key_1, value_1) \in Pr \cdot \\
&\qquad \neg(\exists (key_2, value_2) \in Pr \cdot key_1 = key_2)
\end{aligned}
\tag{44}
$$

This concludes the base validity constraints on modelled graphs of every abstraction level.

4.5.5. Abstraction Level Validity

So far, the validity constraints defined in this chapter concerned the base graph model. Graphs defined on every abstraction level have to comply with these constraints. In this subsection however, validity constraints will be defined for all three abstraction levels required in chapter 3.
An abstraction level can be thought of as the general model, but with additional constraints. Because the base constraints are defined as a conjunction of formulae in first-order logic, these additional constraints can just be added, again with a conjunction. And this is exactly what has to be done to check for abstraction level validity.

AL1 Validity

How this looks like is shown in the following definition of the *ValidAL1Graph* predicate:

$$
\begin{aligned}
\mathit{ValidAL1Graph}((N, E, Po, Pr) \Leftrightarrow\ & \forall n \in N \cdot \\
& (\exists (n, values_n) \in Pr \cdot \\
& (("node\_type", "nested\_node") \in values_n \\
& \wedge ("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n \\
& \wedge (|N_n| = |E_n| = |Po_n| = |Pr_n| = 0)) \\
& \vee (("node\_type", "composed\_output") \in values_n)) \\
& \wedge (|\{n_n \mid (n_n, values_{n_n}) \in Pr \\
& \quad \wedge ("node\_type", "nested\_node") \in values_{n_n}\}| = 1) \\
& \wedge\ \mathit{ValidGraph}(N, E, Po, Pr)
\end{aligned}
\tag{65}
$$

What happens here is quite simple. It is checked, if all the nodes are of type *"composed_output"* or *"nested_node"*, which are the only two node types allowed according to the requirements for AL1. In addition, it is assured that there is exactly one nested node and that this node's inner graph is empty. The nested node on AL1 must not be expanded. If these constraints are fulfilled, then the graph model can be checked for general validity, using *ValidGraph*.
In Figure 4.8, the test set-up introduced in section 2.2 has been modelled on AL1. The signal sources are modelled as ports which are not assigned to any node. The custom output values `IO` and `NIO` calculated by the maXYmos device are modelled as a simple node with type *"custom_output"*, whereas the `X-Y`-diagram is represented by a simple node of type *"composed_output"*. The nested node with uid *"connector"* abstracts away all components that connect the signal sources with these outputs.

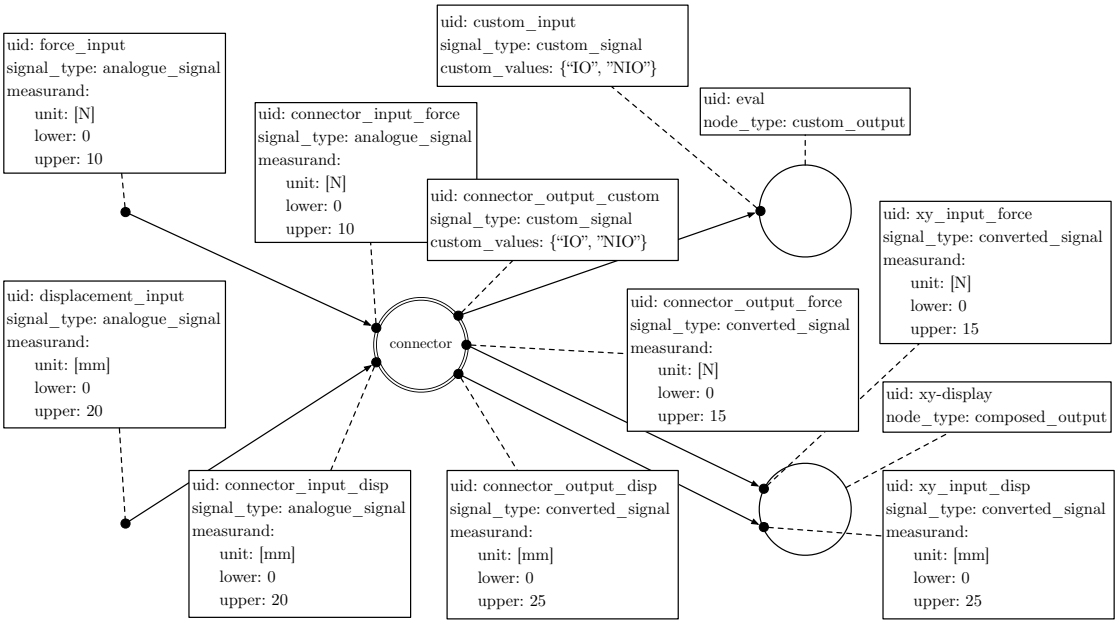

Figure 4.8.: Valid AL1 graph for exemplary test set-up

Therefore, a model graph defined on AL1 shows only which measurands shall be measured in which range, and what the output after a successful test shall look like.

AL2 Validity

AL2 validity can be checked similarly. The predicate *ValidAL2Graph* is defined in the following:

$$
\begin{aligned}
&\mathit{ValidAL2Graph}((N, E, Po, Pr) \Leftrightarrow\ \forall n \in N \cdot \\
&\qquad (\exists (n, values_n) \in Pr \cdot \\
&\qquad ((\text{"}node\_type\text{"}, \text{"}nested\_node\text{"}) \in values_n \\
&\qquad \wedge\ (\text{"}graph\text{"}, (N_n, E_n, Po_n, Pr_n)) \in values_n \\
&\qquad\ \wedge \forall n_n \in N_n \cdot \\
&\qquad (\exists (n_n, values_{n_n}) \in Pr \cdot \\
&\qquad ((\text{"}node\_type\text{"}, type) \in values_{n_n} \\
&\qquad\ \wedge\ (type \in \{\text{"}measuring\_module\text{"}, \text{"}analysis\_module\text{"}\} \\
&\qquad\ \wedge\ (\text{"}graph\text{"}, (N_{n_n}, E_{n_n}, Po_{n_n}, Pr_{n_n})) \in values_{n_n} \\
&\qquad\ \wedge\ (|N_{n_n}| = |E_{n_n}| = |Po_{n_n}| = |Pr_{n_n}| = 0))
\end{aligned}
\tag{66}
$$

$$\vee\ (type \in \{\text{"}sensor\text{"}, \text{"}audio\_recorder\text{"}, \text{"}video\_recorder\text{"}\}))$$
$$\vee\ ((\text{"}node\_type\text{"}, \text{"}composed\_output\text{"}) \in values_n))$$
$$\wedge\ (|\{n_n \mid (n_n, values_{n_n}) \in Pr$$
$$\wedge\ (\text{"}node\_type\text{"}, \text{"}nested\_node\text{"}) \in values_{n_n}\}| = 1)$$
$$\wedge\ ValidGraph(N, E, Po, Pr)$$

The constraints of *ValidAL1Graph* are almost fully contained in this AL2 validity predicate. What changed is that the one nested node that is allowed on AL1 now has to be expanded. The nodes of the expanded node's inner graph must be of type *"measuring_module"*, *"analysis_module"*, *"sensor"*, *"audio_recorder"* or *"video_recorder"*. In addition, measuring module and analysis nodes, which are specialisations of nested node, must have an empty inner graph.

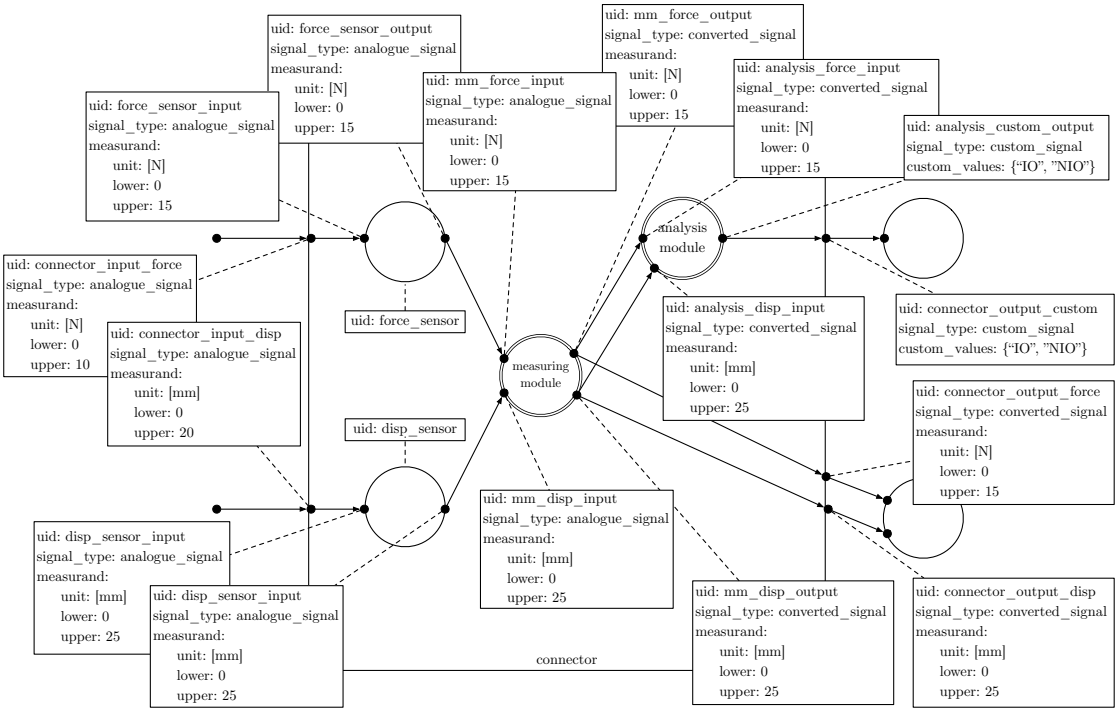

Figure 4.9.: Valid AL2 graph for exemplary test set-up

Figure 4.9 shows the same graph from Figure 4.8, but with the expanded nested node *"connector"*. The properties outside of this node are the same as before and are therefore omitted. It is important to convince oneself that both models represent the same test set-up.
For an example as small as this, to verify whether the AL2 graph is a refinement of the AL1 graph, can be done manually. However, with an increasing model size, the

verification gets significantly harder. How refinement can be defined as a predicate will be shown in section 4.5.6.

AL3 Validity

This last example of how abstraction levels can be defined shows the constraints on AL3. The predicate *ValidAL3Graph* is defined as follows:

$$
\begin{aligned}
&\mathit{ValidAL3Graph}((N, E, Po, Pr) \Leftrightarrow\ \forall n \in N\ \cdot \\
&\quad (\exists (n, values_n) \in Pr\ \cdot \\
&\quad (\text{"}node\_type\text{"}, \text{"}nested\_node\text{"}) \in values_n \\
&\wedge\ (\text{"}graph\text{"}, (N_n, E_n, Po_n, Pr_n)) \in values_n \\
&\quad \wedge \forall n_n \in N_n\ \cdot \\
&\quad (\exists (n_n, values_{n_n}) \in Pr\ \cdot \\
&\quad ((\text{"}node\_type\text{"}, type) \in values_{n_n} \\
&\quad \wedge\ (type \in \{\text{"}measuring\_module\text{"}, \text{"}analysis\_module\text{"}\} \\
&\quad \wedge\ (\text{"}graph\text{"}, (N_{n_n}, E_{n_n}, Po_{n_n}, Pr_{n_n})) \in values_{n_n} \\
&\quad \wedge \forall n_{n_n} \in N_{n_n}\ \cdot \\
&\quad (\exists (n_{n_n}, values_{n_{n_n}}) \in Pr\ \cdot \\
&\quad ((\text{"}node\_type\text{"}, type) \in values_{n_{n_n}} \\
&\quad \wedge\ (type \in \{\text{"}amplifier\text{"}, \text{"}analogue\_digital\_converter\text{"}, \\
&\quad \text{"}filter\text{"}, \text{"}trigger\text{"}\} \\
&\quad \wedge\ (|N_{n_n}| = |E_{n_n}| = |Po_{n_n}| = |Pr_{n_n}| = 0)) \\
&\quad \vee (type \in \{\text{"}sensor\text{"}, \text{"}audio\_recorder\text{"}, \text{"}video\_recorder\text{"}\})) \\
&\vee\ ((\text{"}node\_type\text{"}, \text{"}composed\_output\text{"}) \in values_n)) \\
&\wedge\ (|\{n_n \mid (n_n, values_{n_n}) \in Pr \\
&\quad \wedge\ (\text{"}node\_type\text{"}, \text{"}nested\_node\text{"}) \in values_{n_n}\}| = 1) \\
&\wedge\ \mathit{ValidGraph}(N, E, Po, Pr)
\end{aligned}
\tag{67}
$$

The modelled graph for the exemplary test set-up is given in Figure 4.10. For the sake of completeness, every property that has to be defined to achieve AL3 validity is shown. Some of the properties are highlighted in bold. These properties were added to the properties of already existing elements of the graph. Because none of the old properties have been changed, this can be seen as another form of refinement.

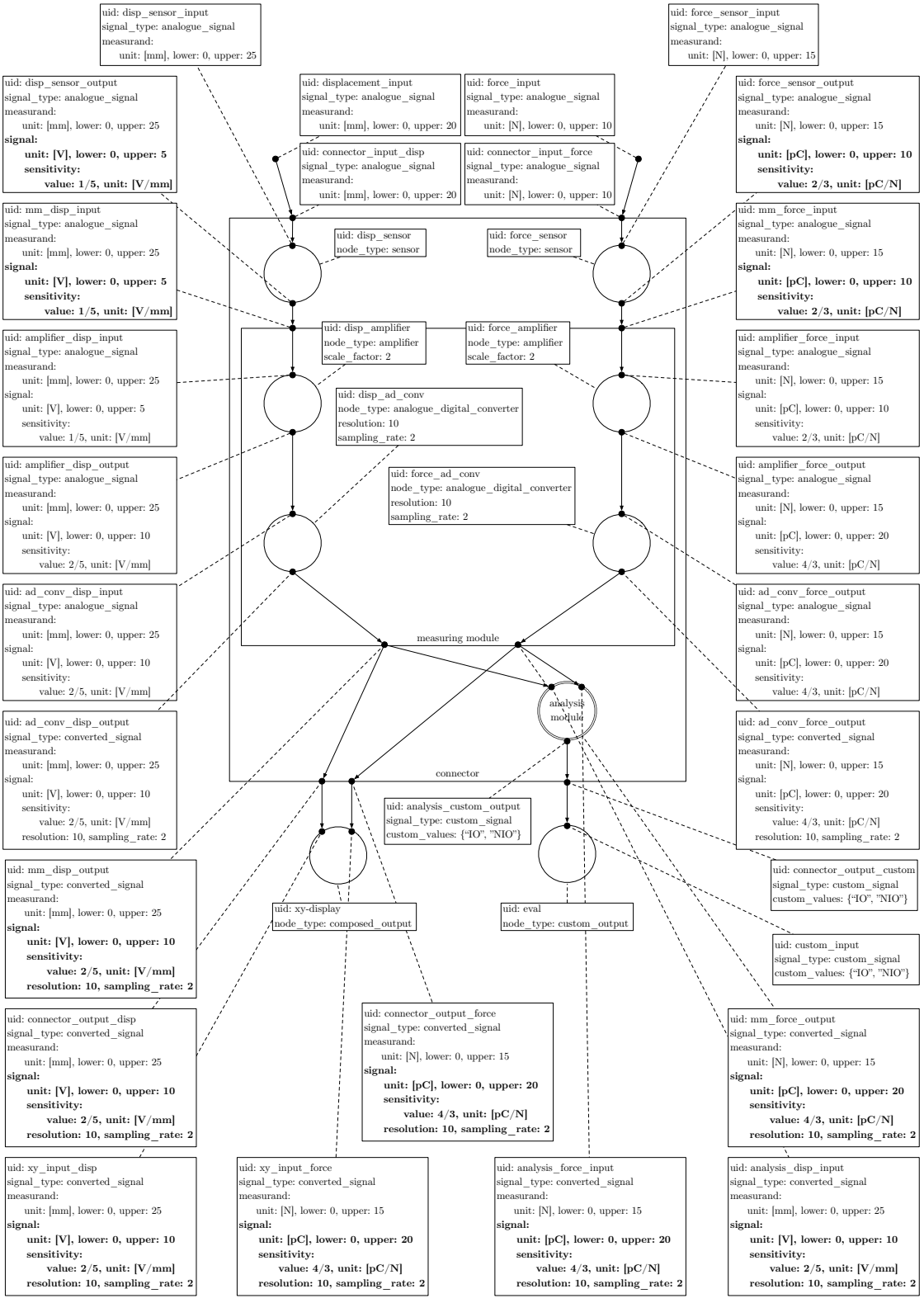

Figure 4.10.: Valid AL3 graph for exemplary test set-up

4.5.6. Refinement

So far, it is possible to determine if a model graph has been defined correctly in terms of the model requirements and, if checked with the corresponding predicate, if a graph model is valid on one of the abstraction levels AL1, AL2 or AL3. But how can it be verified, that two graphs are actually representing the same test set-up? For this purpose a *RefinedGraph* relational predicate has been defined which in turn is defined using the following predicate:

$$\begin{aligned} \mathit{RefinedGraph}((N_2, E_2, Po_2, Pr_2), (N_1, E_1, Po_1, Pr_1)) \Leftrightarrow\ & (N_1 = N_2) \\ & \wedge\ (E_1 = E_2) \\ & \wedge\ (Po_1 = Po_2) \\ & \wedge\ \forall prop_1 \in Pr_1 \cdot \mathit{EnrichedProperty}(prop_1, Pr_2, N_1, Po_1) \end{aligned} \tag{68}$$

EnrichedProperty is a predicate that checks, if all properties defined on a higher level of abstraction are still present in the lower abstraction level. As was mentioned in subsection 4.5.5, it is allowed to add additional properties to Pr, as long as existing properties are not overwritten.
Note that *EnrichedProperty* is defined more than once, covering different cases that can occur. In terms of first-order logic, these definitions must be regarded as being connected by a logical $\vee$. Also, a new syntactic element has been introduced, the '_', which stands for any value that is not needed for the predicate's definition.

$$\begin{aligned} \mathit{EnrichedProperty}((key, values), Pr, \_, Po)) \Leftrightarrow\ & (\exists p \in Po \cdot key = getUid(p)) \\ & \wedge\ ((key, values_2) \in Pr) \\ & \wedge\ (values \subset values_2) \end{aligned} \tag{69}$$

$$\begin{aligned} \mathit{EnrichedProperty}((key, values), Pr, N, \_)) \Leftrightarrow\ & (\exists n \in N \cdot key = getUid(n)) \\ & \wedge\ ((key, values_2) \in Pr) \\ & \wedge\ (("node\_type", type) \in values) \\ & \wedge\ \mathit{EnrichedNodeProperty}(type, values, values_2) \end{aligned} \tag{70}$$

For port properties, *EnrichedProperty* was a straightforward definition. It suffices to check whether the values of properties are a subset of the values on a lower abstraction level. For nodes however, an additional predicate *EnrichedNodeProperty* has been defined, which deals —among other cases— with the expansion of nested

nodes.

$$\begin{aligned} &\mathit{EnrichedNodeProperty}(type, values_1, values_2) \Leftrightarrow \\ &\quad (type \in \{\text{"}nested\_node\text{"}, \text{"}measuring\_module\text{"}, \\ &\quad \text{"}analysis\_module\text{"}\}) \\ &\quad \wedge ((\text{"}graph\text{"}, (N, E, Po, Pr)) \in values_1) \\ &\quad \wedge (|N| = |E| = |Po| = |Pr| = 0) \\ &\quad \wedge (values_1 \setminus (\text{"}graph\text{"}, (N, E, Po, Pr)) \subset values_2) \end{aligned} \tag{71}$$

$$\begin{aligned} &\mathit{EnrichedNodeProperty}(type, values_1, values_2) \Leftrightarrow \\ &\quad (type \in \{\text{"}nested\_node\text{"}, \text{"}measuring\_module\text{"}, \\ &\quad \text{"}analysis\_module\text{"}\}) \\ &\quad \wedge ((\text{"}graph\text{"}, (N_1, E_1, Po_1, Pr_1)) \in values_1) \\ &\quad \wedge \neg(|N_1| = |E_1| = |Po_1| = |Pr_1| = 0) \\ &\quad \wedge ((\text{"}graph\text{"}, (N_2, E_2, Po_2, Pr_2)) \in values_2) \\ &\quad \wedge \neg(|N_2| = |E_2| = |Po_2| = |Pr_2| = 0) \\ &\quad \wedge (values_1 \setminus (\text{"}graph\text{"}, (N_1, E_1, Po_1, Pr_1)) \subset \\ &\quad values_2 \setminus (\text{"}graph\text{"}, (N_2, E_2, Po_2, Pr_2))) \\ &\quad \wedge \mathit{RefinedGraph}((N_2, E_2, Po_2, Pr_2), (N_1, E_1, Po_1, Pr_1)) \end{aligned} \tag{72}$$

$$\begin{aligned} &\mathit{EnrichedNodeProperty}(type, values_1, values_2) \Leftrightarrow \\ &\quad \neg(type \in \{\text{"}nested\_node\text{"}, \text{"}measuring\_module\text{"}, \\ &\quad \text{"}analysis\_module\text{"}\}) \\ &\quad \wedge (values_1 \subset values_2) \end{aligned} \tag{73}$$

The three model graphs of subsection 4.5.5 have been unit tested regarding refinement validity. Unit tests for the whole model specification are shown in [8].

5. Model Evaluation

Subsections 4.5.5 and 4.5.6 already showed promising results by illustrating that the exemplary test set-up of section 2.2 can be modelled on all three abstraction levels. The intention behind this chapter is to strengthen the reader's confidence, that the described modelling system really solves the problems stated in section 1.1.
To achieve that, it will firstly be verified that the modelling system fulfils the requirements listed in chapter 3. Secondly, the author's thoughts about extensibility will be provided.

5.1. Model Verification

To visualise the fulfilment of requirements, a table layout has been worked out which demonstrates the couplings between requirements, domain classes and, of course, the predicates defined to solve validity constraints. As was the case for the requirements, there are tables for the requirements of every abstraction level and one table for the overall model requirement. Table 5.1 shows the couplings for requirements of AL1.

Req. ID (Table)	**Domain Class**	**Predicates**
AL1.1 (A.1)	`Port`	(3), (4)
AL1.2 (A.2)	`Port`	(3), (8)
AL1.3 (A.3)	`Port`	(3), (9)
AL1.4 (A.4)	`Port`	(3), (4), (5), (6), (7), (22), (23), (24), (42), (43)
AL1.5 (A.5)	`Port`	(3), (10)
AL1.6 (A.6)	`Custom Output`	(22), (25)
AL1.7 (A.7)	`Port`	(3), (8)
AL1.8 (A.8)	`Port`	(3), (9)
AL1.9 (A.9)		(3), (4), (5), (11), (12)
AL1.10 (A.10)	`Graph`	(1), (21), (65)

Table 5.1.: AL1 model verification

It stands out that almost all requirements are met with the aid of the domain class `Port`. When looking at the domain model shown in Figure 4.2, the same observation can be made. Only custom outputs are modelled using a specification

of class *Node*. But in fact, when looking at the requirements more closely, it becomes obvious that the properties of ports are sufficient to model all other inputs and outputs defined for AL1. The predicates *ValidPort* (3) and the corresponding *ValidPortProperties* can therefore be used to fulfil almost all of the requirements of AL1.
Further, Table 5.2 illustrates the couplings for requirements of AL2, again with domain classes and predicates.

Req. ID (Table)	**Domain Class**	**Predicates**
AL2.1 (A.11)	`Sensor`	(22), (26)
AL2.2 (A.12)	`Audio Recorder`	(22), (27)
AL2.3 (A.13)	`Video Recorder`	(22), (28)
AL2.4 (A.14)	`Measuring Module`	(22), (29), (30)
AL2.5 (A.15)	`Analysis`	(22), (31), (32), (42), (43)
AL2.6 (A.16)	`Edge`	(14), (15)
AL2.7 (A.17)	`Edge`	(14), (15)
AL2.8 (A.18)	`Edge`	(14), (15)
AL2.9 (A.19)	`Edge`	(14), (15)
AL2.10 (A.20)	`Edge`	(14), (15)
AL2.11 (A.21)	`Edge`	(14), (15)
AL2.12 (A.22)	`Edge`	(14), (15)
AL2.13 (A.23)	`Edge`	(14), (15)
AL2.14 (A.24)	`Edge`	(14), (15)
AL2.15 (A.25)	`Edge`	(14), (15)
AL2.16 (A.26)	`Edge`	(14), (15)
AL2.17 (A.27)	`Edge`	(14), (15)
AL2.18 (A.28)	`Edge`	(14), (15)
AL2.19 (A.29)	`Edge`	(14), (15)
AL2.20 (A.30)	*Node*, `Port`	(3), (22), (25), (26), (27), (28), (40)
AL2.21 (A.31)	`Graph`	(1), (65), (66), (68)

Table 5.2.: AL2 model verification

A few observations can be made at this point. A lot of requirements of AL2 are fulfilled using the domain class `Edge` along with predicates (14) and (15), which represent the *ValidEdge* and *MatchingProperties* predicates. When looking at the requirements again, it stands out that all the requirements under investigation concern the validities of links between two specific components. The way the modelling system was specified, allowed to group these validity constraints into the base graph model, which leads to the result visible above.
The more complex components required for AL2 are realised using their corresponding domain classes and the *ValidNode* (22) and corresponding *MatchingPorts* predicates.
AL2.21 does not only include *ValidAL1Graph* (65) and *ValidAL2Graph* (66) predicates, but is also partly fulfilled with the help of *RefinedGraph* (68).

What is missing in terms of abstraction levels are the couplings for requirements of AL3. These are listed in Table 5.3.

Req. ID (Table)	**Domain Class**	**Predicates**
AL3.1 (A.32)	`Port`	(3), (4), (5), (11)
AL3.2 (A.33)	`Port`	(3), (6), (7), (12)
AL3.3 (A.34)	`Port`	(3), (5), (7)
AL3.4 (A.35)	`Amplifier`	(22), (33)
AL3.5 (A.36)	`AD Converter`	(22), (34)
AL3.6 (A.37)	`Filter`	(22), (35)
AL3.7 (A.38)	`Trigger`	(22), (36)
AL3.8 (A.39)	`Edge`	(14), (15)
AL3.9 (A.40)	`Edge`	(14), (15)
AL3.10 (A.41)	`Edge`	(14), (15)
AL3.11 (A.42)	`Edge`, *`Node`*, `Port`	(3), (14), (15), (22)
AL3.12 (A.43)	`Graph`	(1), (66), (67), (68)

Table 5.3.: AL3 model verification

The coupling for the one requirement concerning the overall modelling system is shown in Table 5.4.

Req. ID (Table)	**Domain Class**	**Predicates**
OM1.1 (A.44)	`Graph`	(1), (65), (66), (67), (68)

Table 5.4.: Overall model verification

When looking at all couplings it can be seen that the entry points for all validity constraints of every requirement are either *ValidPort* (3), *ValidEdge* (14), *ValidNode* (22) or *ValidGraph* (1).

5.2. Extensibility

The modelling system has limitations on what can be modelled in its current state. Therefore, it is important to address the complexity of extending the system. This section tries to estimate the investment necessary, to implement various extensions, by listing all potential changes that would have to be undertaken.
Possible extensions are the introduction of new abstraction levels, new components and additional signal properties on ports. These discussed in the following subsections.

5.2.1. Adding Abstraction Levels

The validity of models on specific abstraction levels has been discussed in subsection 4.5.5. For every abstraction level, a new predicate was defined, taking the graph as its parameters. The abstraction level specific constraints were then connected with the *ValidGraph* (1) predicate by conjunction. So, to define the new abstraction level X (ALX), the following modifications to the existing modelling system have to be made:

- Definition of a new predicate *ValidALXGraph*:
  The argument of this predicate is the 4-tuple representing a graph model.
- Conjunction of all abstraction level specific constraints:
  This will probably include checks for the occurrence or absence of specific components and the nesting of nested nodes and specialisations thereof. Other constraints are imaginable. It is important that the constraints are less restrictive than the ones for higher abstraction levels and more restrictive in comparison with lower abstraction levels. At least, graphs modelled on this new abstraction level have to be a refinement of all corresponding graphs modelled on levels above.
- Conjunction with the *ValidGraph* predicate:
  The argument can be handed through from *ValidALXGraph*.

By following the above steps, the definition of a new abstraction level is straightforward and not coupled with any changes to existing predicates of the model. To check the definition for correctness, one can use the *RefinedGraph* (68) predicate and check if graphs modelled on the abstraction levels represent the same model as the ones defined on existing abstraction levels.

5.2.2. Adding Components

Adding a component to the modelling system is equivalent to adding a new node to the base graph model. Subsection 4.5.3 has illustrated how validity constraints for nodes are defined. The following step has to be followed to introduce a new node of type "*my_type*":

- Definition of an additional *MatchingPorts* predicate:
  The second argument must be the newly defined node type "*my_type*".
- Conjunction of all node constraints:
  Newly introduced node properties can be accessed through Pr, which is an argument *MatchingPorts*. The constraints defined here are usually checks

whether the input ports and output ports are defined according to the nodes intended behaviour.

The addition of components does not change existing predicates of the current modelling system.

5.2.3. Adding Signal Properties

It is likely that at some point the current signal properties assigned to ports will be extended. For example, audio and video signals have no other properties than "*signal_type*" in the current modelling system. It is also imaginable to define completely new signal types. In the following, the steps needed to introduce a new signal type "*my_signal_type*" are listed:

- Definition of an additional *ValidPortProperties* predicate:
  The first argument must be "*my_signal_type*".
- Conjunction of all property constraints:
  The constraints concern the nested properties of the port under inspection.
- Definition of additional *MatchingProperties* predicates:
  The *MatchingProperties* predicates are defined for checking edge validity on port property level. The introduction of a new signal type would mean to define new predicates for every property and nested properties of this signal type.
- Adjustment of the *ValidPort* predicate: *ValidPort* (3) has constraints that check the number of outgoing edge from the port under inspection. It could be that the new signal type requires another amount of connections and a change to the predicate would be inevitable.
- Adjustment of *MatchingPorts* properties:
  If, for example, existing nodes should not be able to have the the new signal type assigned to their input and output ports, the validity constraints in their corresponding *MatchingPorts* predicate would have to be changed.

Summarising the findings above, it can be said, that adding signal properties can result in necessary adjustments to the existing node and port validity constraints. However, the adjustments are only necessary, if the current constraints are influenced by the new properties. This must not be the case. Because adding properties is the only extension where changes might occur, it is recommended to start an extension here.

6. Results

The model specified in chapter 4 can be used to model test set-ups. But does it solve the problems stated in section 1.1? That this is the case will be shown in the following sections which are named after the problem under inspection.

6.1. P1: Abstraction Levels

In subsection 4.5.5, it was shown how the base model can be extended with additional constraints to define different levels of abstraction. Of course, the three examples (65), (66) and (67) were exactly that, examples. As was shown in subsection 5.2.1 defining a new abstraction level is simply defining a new predicate which combines all constraints that the abstraction level must fulfil. With the new predicate it can then be tested if a model is valid and therefore, if the model actually represents a set-up on the defined abstraction level. For example, one could define a lowest abstraction level, where every nested node has been fully expanded and every property of every port has been given a value.
But just defining abstraction levels is not enough. It still needs to be shown that models on different abstraction levels represent the same test set-up. This can be achieved by applying the *RefinedGraph* (68) on the two models in question. If the predicate evaluates to true, it can be stated that both models represent the same set-up.
Problem **P1** has been solved successfully. However, the model only provides the means to define abstraction levels and test for refinement. The definition of useful abstraction level constraints still poses a complex challenge which requires further research.

6.2. P2: Model Validation

Because validity constraints have been defined on every component of the model and all constraints are concatenated in the *ValidGraph* (1) predicate, every modelling error is detected by simply applying the predicate to the model. The validation of the examples in chapter 4 showed just a glimpse of what the model is capable to detect. As stated in **P2**, the validity constraints specified only consider validity

constraints on components and the connections between them. Of course, the same strategy of defining validation constraints and concatenating them in one predicate can be used to define all sorts of validity checks.
To verify that the specified validity constraints solve the problem in practice, a prototype has been implemented in Prolog. As a result of extensive testing, it can safely be assumed, that the intended validation behaviour was achieved. The feasibility of implementation is therefore evident. The documentation of the code can be found in [8].
This section showed that problem **P2** has been solved successfully. Modelled test set-ups can be validated by applying them to one largely nested predicate containing all desired constraints.

7. Conclusion

Concluding this thesis, the realisation of the goals defined in section 1.2 is discussed and proposals for future work are pointed out in the following sections.

7.1. Realisation of Goals

During the time period in which this thesis was conducted, a modelling system for test set-ups has been designed and formally specified. The specification was presented in chapter 4 of this documentation (**G1**).
A set of constraints, combined into a single predicate has been defined, which allows to assess the validity of modelled set-ups. The constraints were also captured in chapter 4 as part of the models specification (**G2**).
Section 5.1 showed that all requirements on the modelling systems elaborated by the author and Kistler Instrumente AG were implemented. Section 5.2 identified the modelling system to be adequately and sometimes even easily extensible (**G3**).
Finally, a prototype of the modelling system's validation mechanism has been implemented and documented in [8].
Summarising this section, it can be stated that the goals defined in section 1.2 have been achieved completely.

7.2. Future Work

As was stated in section 2.1, the current model specification is not complete. This last section proposes possible future work for improving the modelling system.

- Extension of the property set on different signal types, for example, audio and video signals, different encodings, bitrate and so on.
- Extension of the set of components the modelling systems contains, for example, additional types of sensors, signal processing components and so on.
- Addition of validity constraints for more than just the most general cases.
- Definition of multiple abstraction levels fulfilling specifically defined requirements.

- Definition of additional model checks, for example, checking if the propagated error through a set-up does not overstep a certain boundary and so on.
- Enrichment of the prototype with a graphical user interface and a parser that translates the model drawing into the format Prolog requires for the validity query.
- Development of an integrated software tool that is easy to use for an engineer.

Bibliography

[1] M. C. Corporation. DASYLab®2016 Software. http://www.mccdaq.com/products/dasylab.htm. Accessed: 2017-01-02.

[2] N. I. Corporation. LabVIEW System Design Software. http://www.ni.com/labview. Accessed: 2017-01-02.

[3] R. Felderhoff. *Elektrische und elektronische Messtechnik: Grundlagen, Verfahren, Gerte und Systeme*. Carl Hanser Verlag GmbH Co KG, 2015.

[4] K. Group. maXYmos BL. https://www.kistler.com/?type=669&fid=56082&callee=frontend. Accessed: 2016-10-18.

[5] K. Group. Position Sensor Model T. https://www.kistler.com/?type=669&fid=36012&callee=frontend. Accessed: 2016-10-15.

[6] K. Group. Press Force Sensor. https://www.kistler.com/?type=669&fid=58581&callee=frontend. Accessed: 2016-10-15.

[7] I. Marvin Test Solutions. Test Executive And Development Studio. http://www.marvintest.com/Product.aspx?model=ATEasy. Accessed: 2017-01-02.

[8] M. Meili. Development of a Modelling Procedure for the Description of Test Set-Ups with Measuring Instruments - Prototype Documentation. 2017.

[9] A. S. Morris. *Measurement and instrumentation principles*. Butterworth, Oxford, 2001.

[10] R. B. Northrop. *Introduction to instrumentation and measurements*. CRC Press, 2005.

[11] A. Thompson and B. N. Taylor. Guide for the Use of the International System of Units (SI). 2008.

A. Complete Requirements

This chapter list all requirements elaborated during the work conducted on this thesis.

A.1. AL1 Requirements

In this section, all requirements regarding AL1 are listed without any further explanations.

ID	AL1.1
Last Updated	2016-11-01
Title	Input attributes
Description	The following attributes have to be defined for each input: • UID • Measurand • Range For an input to be valid, the measurand has to match the physical unit of the defined range.
Priority	High

Table A.1.: Requirement: input attributes

ID	AL1.2
Last Updated	2016-11-01
Title	Audio input attributes
Description	The following attributes have to be defined for each audio input: • UID
Priority	High

Table A.2.: Requirement: audio input attributes

ID	AL1.3
Last Updated	2016-11-01
Title	Video input attributes
Description	The following attributes have to be defined for each video input: • UID
Priority	High

Table A.3.: Requirement: video input attributes

ID	AL1.4
Last Updated	2016-11-01
Title	Output attributes
Description	The following attributes have to be defined for each output: • UID • Measurand • Range For an output to be valid, the physical unit of its range must be either • equal to to the physical unit of the range of any input or • the physical unit of its range must be derivable from the physical units of any number of inputs (for example acceleration in $[\frac{m}{s}]$ is derivable from inputs measuring time in `[s]` and displacement in `[m]`). Derivable units also include units of the same measurand family (for example a measurement in `[m]` can also be expressed in `[mm]`.
Priority	High

Table A.4.: Requirement: output attributes

ID	AL1.5
Last Updated	2016-12-31
Title	Custom outputs
Description	The following attributes have to be defined for each custom output: • UID • $\{output\_value\_1, \ldots, output\_value\_t\}$, $\{t \in \mathbb{N} \mid t \geq 1\}$ • Description (optional) For a custom output to be valid, each output value has to be of the same type.
Priority	High

Table A.5.: Requirement: custom outputs

ID	AL1.6
Last Updated	2016-11-01
Title	Composed output attributes
Description	Composed outputs need the following attributes defined, where n is the number of regular outputs that are combined and $\{n \in \mathbb{N} \mid n \geq 2\}$: • UID • $n\times$ (Measurand, Range) • Description (optional) For a composed output to be valid, each of the contained outputs has to be valid.
Priority	Medium

Table A.6.: Requirement: composed output attributes

ID	AL1.7
Last Updated	2016-11-01
Title	Audio output attributes
Description	The following attributes have to be defined for each audio output: • UID
Priority	High

Table A.7.: Requirement: audio output attributes

ID	AL1.8
Last Updated	2016-11-01
Title	Video output attributes
Description	The following attributes have to be defined for each video output: • UID
Priority	High

Table A.8.: Requirement: video output attributes

ID	AL1.9
Last Updated	2016-10-31
Title	Range attritubes
Description	The following attributes have to be defined for each range: • Lower bound • Upper bound • Physical unit Note that the physical unit in each (Measurand, Range) pair must match the measurand.
Priority	High

Table A.9.: Requirement: range attritubes

ID	AL1.10
Last Updated	2016-11-01
Title	AL1 validity
Description	For the model to be valid on AL1 the following conditions have to hold: • Each defined input is valid. • Each defined output is valid.
Priority	High

Table A.10.: Requirement: al1 validity

A.2. AL2 Requirements

In this section, all requirements regarding AL2 are listed without any further explanations.

ID	AL2.1
Last Updated	2016-11-01
Title	Sensor attributes
Description	The following attributes have to be defined for each sensor: • UID • (Measurand, Range) For a sensor to be valid, the measurand has to match the physical unit of the defined range.
Priority	High

Table A.11.: Requirement: sensor attributes

ID	AL2.2
Last Updated	2016-11-01
Title	Audio recorder attributes
Description	The following attributes have to be defined for each audio recorder: • UID
Priority	High

Table A.12.: Requirement: audio recorder attributes

ID	AL2.3
Last Updated	2016-11-01
Title	Video recorder attributes
Description	The following attributes have to be defined for each video recorder: • UID
Priority	High

Table A.13.: Requirement: video recorder attributes

ID	AL2.4
Last Updated	2016-11-22
Title	Measuring module attributes
Description	Measuring modules need the following attributes defined, where n is the number of inputs, m is the number of outputs and $\{n, m \in \mathbb{N} \mid n > 0 \wedge m > 0\}$: • UID • $n\times$ (Measurand, Range) for inputs • $m\times$ (Measurand, Range) for outputs For a measuring module to be valid, there exists an input for every output and • Measurand input = Measurand output • Range input $\subseteq$ Range output
Priority	High

Table A.14.: Requirement: measuring module attributes

ID	AL2.5
Last Updated	2016-11-22
Title	Analysis attributes
Description	Analyses need the following attributes defined, where $n+o+p$ is the number of inputs, $q+r$ is the number of outputs and $\{n, o, p, q, r \in \mathbb{N} \mid n+o+p > 0 \wedge q+r > 0\}$: • UID • Description • $n\times$ (Measurand, Range) for inputs • $o\times$ audio input • $p\times$ video input • $q\times$ $\{output\_value\_1, \ldots, output\_value\_t\}$, $\{t \in \mathbb{N} \mid t \geq 2\}$ • $r\times$ (Measurand, Range) for outputs The description can contain any kind of process description, for example plain text, pseudo code, source code and so on. For an analysis to be valid, each output value has the be of the same type. For an analysis to be valid, the physical unit of each output range must be either • equal to to the physical unit of the range of any input or • the physical unit of each output range must be derivable from the physical units of any number of inputs (for example acceleration in $[\frac{m}{s}]$ is derivable from inputs measuring time in `[s]` and displacement in `[m]`). Derivable units also include units of the same measurand family (for example a measurement in `[m]` can also be expressed in `[mm]`.
Priority	High

Table A.15.: Requirement: analysis attributes

ID	AL2.6
Last Updated	2016-11-01
Title	Input sensor link
Description	Inputs of AL1 can only be linked with sensors. For the link to be valid, the following conditions have to hold: • Measurand input = Measurand sensor • Range input $\subseteq$ Range sensor (The desired input range must be in the sensors range) • Each input is linked to a sensor.
Priority	High

Table A.16.: Requirement: input sensor link

ID	AL2.7
Last Updated	2016-11-01
Title	Audio input audio recorder link
Description	Audio inputs of AL1 can only be linked with audio recorders. For the link to be valid, the following conditions have to hold: • Each audio input is linked to an audio recorder.
Priority	High

Table A.17.: Requirement: audio input audio recorder link

ID	AL2.8
Last Updated	2016-11-01
Title	Video input video recorder link
Description	Video inputs of AL1 can only be linked with video recorders. For the link to be valid, the following conditions have to hold: • Each video input is linked to a video recorder.
Priority	High

Table A.18.: Requirement: video input video recorder link

ID	AL2.9
Last Updated	2016-11-01
Title	Sensor measuring module link
Description	Sensor to measuring module links have to fulfil the following conditions to be valid: • Measurand sensor = Measurand measuring module input • Range sensor $\subseteq$ Range measuring module input • Each measuring module input is linked to a sensor.
Priority	High

Table A.19.: Requirement: sensor measuring module link

ID	AL2.10
Last Updated	2016-11-01
Title	Sensor output link
Description	Sensor to output links have to fulfil the following conditions to be valid: • Measurand sensor = Measurand output • Range sensor $\subseteq$ Range output
Priority	High

Table A.20.: Requirement: sensor output link

ID	AL2.11
Last Updated	2016-11-01
Title	Sensor composed output link
Description	Sensor to composed output links have to fulfil the following conditions to be valid: • Measurand sensor = Measurand composed output • Range sensor $\subseteq$ Range composed output
Priority	High

Table A.21.: Requirement: sensor composed output link

ID	AL2.12
Last Updated	2016-11-01
Title	Audio recorder analysis link
Description	Audio recorder to analysis links have to fulfil the following conditions to be valid: • Each analysis audio input is linked to an audio recorder.
Priority	High

Table A.22.: Requirement: audio recorder analysis link

ID	AL2.13
Last Updated	2016-11-01
Title	Audio recorder audio output link
Description	No special conditions have to hold for a an audio recorder to audio output link to be valid.
Priority	High

Table A.23.: Requirement: audio recorder audio output link

ID	AL2.14
Last Updated	2016-11-01
Title	Video recorder analysis link
Description	Video recorder to analysis links have to fulfil the following conditions to be valid: • Each analysis video input is linked to a video recorder.
Priority	High

Table A.24.: Requirement: video recorder analysis link

ID	AL2.15
Last Updated	2016-11-01
Title	Video recorder video output link
Description	No special conditions have to hold for a a video recorder to video output link to be valid.
Priority	High

Table A.25.: Requirement: video recorder video output link

ID	AL2.16
Last Updated	2016-11-01
Title	Measuring module analysis link
Description	Measuring module to analysis links have to fulfil the following conditions to be valid: • Measurand measuring module output = Measurand analysis input • Range measuring module output $\subseteq$ Range analysis input • Each analysis input is linked to a measuring module output.
Priority	High

Table A.26.: Requirement: measuring module analysis link

ID	AL2.17
Last Updated	2016-11-01
Title	Measuring module output link
Description	Measuring module to output links have to fulfil the following conditions to be valid: • Measurand measuring module output = Measurand output • Range measuring module output $\subseteq$ Range output
Priority	High

Table A.27.: Requirement: measuring module output link

ID	AL2.18
Last Updated	2016-11-01
Title	Measuring module composed output link
Description	Measuring module to composed output links have to fulfil the following conditions to be valid: • Measurand measuring module output = Measurand composed output • Range measuring module output $\subseteq$ Range composed output
Priority	High

Table A.28.: Requirement: measuring module composed output link

ID	AL2.19
Last Updated	2016-11-01
Title	Analysis custom output link
Description	Analysis to custom output links have to fulfil the following conditions to be valid: • $\{output\_value\_1 \ldots, output\_value\_t\}$ analysis output = $\{output\_value\_1 \ldots, output\_value\_t\}$ custom output, $\{t \in \mathbb{N} \mid t \geq 2\}$ • Each custom output is linked to an analysis output.
Priority	High

Table A.29.: Requirement: analysis custom output link

ID	AL2.20
Last Updated	2016-11-22
Title	AL2 link validity
Description	For the model to be link valid on AL2 the following conditions have to hold: • Each input, audio input and video input of AL1 is linked exactly once. • Each output, audio output, video output and custom output of AL1 is linked exactly once. • Each composed output of AL1 is linked exactly n times, where n is the number of outputs the composed output is composed of. • Each sensor, audio recorder and video recorder have exactly one outgoing link. • Each input of every component is linked exactly once.
Priority	High

Table A.30.: Requirement: al2 link validity

ID	AL2.21
Last Updated	2016-11-01
Title	AL2 validity
Description	For the model to be valid on AL2 the following conditions have to hold: • The model is valid on AL1. • AL2 link validity holds. • Each defined component is valid. • Each defined link is valid. or • No components have been defined. • No links have been defined.
Priority	High

Table A.31.: Requirement: al2 validity

A.3. AL3 Requirements

In this section, all requirements regarding AL3 are listed without any further explanations.

ID	AL3.1
Last Updated	2017-01-08
Title	Analog signal attributes
Description	The following attributes have to be defined for each analogue signal: • UID • Range
Priority	Medium

Table A.32.: Requirement: analog signal attributes

ID	AL3.2
Last Updated	2017-01-08
Title	Digital signal attributes
Description	The following attributes have to be defined for each digital signal: • UID • Range • Resolution • Sampling Rate
Priority	Medium

Table A.33.: Requirement: digital signal attributes

ID	AL3.3
Last Updated	2016-11-01
Title	AL3 Signal validity
Description	For signal validity to hold, the following conditions have to hold: • An analogue signal is assigned to every sensor output defined on AL2. • An analogue signal is assigned to every measuring module input defined on AL2. • A digital signal is assigned to every measuring module output defined on AL2. • A digital signal is assigned to every analysis input defined on AL2. • Links between components a and b defined on AL2 where signals are assigned $\Rightarrow$ Output signal component a = Input signal component b.
Priority	Medium

Table A.34.: Requirement: al3 signal validity

ID	AL3.4
Last Updated	2016-11-01
Title	Amplifier attributes
Description	The following attributes have to be defined for each amplifier: • UID • Input Signal • Scale Factor The output signal results directly from the input signal multiplied with the scale factor.
Priority	Medium

Table A.35.: Requirement: amplifier attributes

ID	AL3.5
Last Updated	2016-11-01
Title	Analogue/digital converter attributes
Description	The following attributes have to be defined for each analogue/digital converter: • UID • Input Signal • Resolution • Sampling Rate The output signal is a digital signal gained by sampling the input signal with the given sampling rate and resolution.
Priority	Medium

Table A.36.: Requirement: analogue/digital converter attributes

ID	AL3.6
Last Updated	2016-11-01
Title	Filter attributes
Description	The following attributes have to be defined for each filter: • UID • Input Signal • Range The output signal is the part of the input signal that is in the range of the input signal as well as in the range of the filter.
Priority	Medium

Table A.37.: Requirement: filter attributes

ID	AL3.7
Last Updated	2016-11-01
Title	Trigger attributes
Description	The following attributes have to be defined for each trigger, where n is the number of input signals and $\{n \in \mathbb{N} \mid n > 0\}$ • UID • $n\times$ Input Signal • Type • Description The description can contain any kind of process description, for example plain text, pseudo code, source code and so on. For the trigger to be valid, the output signal has to be equal to one of the input signals.
Priority	Medium

Table A.38.: Requirement: trigger attributes

ID	AL3.8
Last Updated	2016-11-22
Title	Component part link
Description	For the parts defined for a measuring module of AL2, each of the measuring module's inputs has to be linked to the input of one of the enclosed parts. For the link to be valid, the following conditions have to hold: • Input signal component = Input signal part
Priority	Medium

Table A.39.: Requirement: component part link

ID	AL3.9
Last Updated	2016-11-22
Title	Part component link
Description	For the parts defined for a measuring module of AL2, each of the measuring module's outputs has to be linked to the output of one of the enclosed parts. For the link to be valid, the following conditions have to hold: • Output signal part = Output signal component
Priority	Medium

Table A.40.: Requirement: part component link

ID	AL3.10
Last Updated	2016-11-01
Title	Inter part link
Description	For a link between a part a and a part b to be valid, the following conditions have to hold: • Output signal part a = Input signal part b
Priority	Medium

Table A.41.: Requirement: inter part link

ID	AL3.11
Last Updated	2016-11-22
Title	AL3 Link validity
Description	For link validity to hold, the following conditions have to hold: • The input of a component links to multiple inputs of parts. ⇒ The input's signal is a digital signal. • The output of a component links to multiple outputs of a component. ⇒ The output's signal is a digital signal. • The output of a part links to multiple inputs of parts and/or outputs of components. ⇒ The output's signal is a digital signal. • Each component output is linked to only one part output. • Cyclic paths must not be defined.
Priority	Medium

Table A.42.: Requirement: al3 link validity

ID	AL3.12
Last Updated	2016-11-01
Title	AL3 validity
Description	For the model to be valid on AL3 the following conditions have to hold: • The model is valid on AL2. • AL3 Signal validity holds. • AL3 Link validity holds. • Links between components/parts a and b where signals are assigned $\Rightarrow$ An AL3 link is defined. • Each defined part is valid. • Each defined link is valid. or • No parts/links have been defined.
Priority	Medium

Table A.43.: Requirement: al3 validity

A.4. Overall Model Requirements

In this section, all requirements regarding the overall modelling system are listed without any further explanations.

ID	OM1.1
Last Updated	2016-11-01
Title	Model validity
Description	For the model to be valid, the following conditions have to hold: • The model is valid on AL1. • The model is valid on AL2. • The model is valid on AL3.
Priority	High

Table A.44.: Requirement: model validity

B. Complete Validity Constraints

This chapter list all validity constraints in the form of predicates using formulae in first-order logic.

$$\begin{aligned} ValidGraph((N, E, Po, Pr)) \Leftrightarrow & ValidPorts(Po, Pr, E) \\ & \wedge ValidEdges(E, Pr) \\ & \wedge ValidNodes(N, Pr, E) \\ & \wedge ValidProperties(Pr) \end{aligned} \tag{1}$$

B.1. Port Validity

In this section, all validity constraints regarding ports are listed without any further explanations.

$$ValidPorts(Po, Pr, E) \Leftrightarrow \forall p \in Po \ \cdot ValidPort(p, Pr, E) \tag{2}$$

$$\begin{aligned} ValidPort(p, Pr, E) \Leftrightarrow \ & (0 \leq |\{e_t \mid (getToPort(e_t) = p) \wedge (e_t \in E)\}| \leq 1) \\ & \wedge (|\{e_s \mid (getFromPort(e_s) = getToPort(e_s) = p) \\ & \quad \wedge (e_s \in E)\}| = 0) \\ & \wedge (\exists prop_p \in Pr \cdot getKey(prop_p) = getUid(p)) \\ & \wedge (\exists prop_t \in getValue(prop_p) \cdot \\ & \quad getKey(prop_t) = "signal\_type") \\ & \wedge ((getValue(prop_t) \in \{"analogue\_signal", \\ & \quad "audio\_signal", "video\_signal"\} \\ & \quad \wedge (0 \leq |\{e_f \mid (getFromPort(e_f) = p) \wedge (e_f \in E)\}| \leq 1)) \\ & \quad \vee getValue(prop_t) \in \{"converted\_signal", \\ & \quad "custom\_signal"\}) \\ & \wedge ValidPortProperties(getValue(prop_t), prop_p) \end{aligned} \tag{3}$$

$$\begin{aligned} \mathit{ValidPortProperties}(\text{"}\mathit{analogue\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ & (|\mathit{props}| = 2) \\ & \wedge\ ((\text{"}\mathit{measurand}\text{"}, \mathit{values}) \in \mathit{props}) \\ & \wedge\ \mathit{ValidMeasurand}((\text{"}\mathit{measurand}\text{"}, \mathit{values})) \end{aligned} \tag{4}$$

$$\begin{aligned} \mathit{ValidPortProperties}(\text{"}\mathit{analogue\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ & (|\mathit{props}| = 3) \\ & \wedge\ ((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m) \in \mathit{props}) \\ & \wedge\ \mathit{ValidMeasurand}((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m)) \\ & \wedge\ ((\text{"}\mathit{signal}\text{"}, \mathit{values}_s) \in \mathit{props}) \\ & \wedge\ (|\mathit{values}_s| = 4) \\ & \wedge\ \mathit{ValidSignal}((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m), (\text{"}\mathit{signal}\text{"}, \mathit{values}_s)) \end{aligned} \tag{5}$$

$$\begin{aligned} \mathit{ValidPortProperties}(\text{"}\mathit{converted\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ & (|\mathit{props}| = 2) \\ & \wedge\ ((\text{"}\mathit{measurand}\text{"}, \mathit{values}) \in \mathit{props}) \\ & \wedge\ \mathit{ValidMeasurand}((\text{"}\mathit{measurand}\text{"}, \mathit{values})) \end{aligned} \tag{6}$$

$$\begin{aligned} \mathit{ValidPortProperties}(\text{"}\mathit{converted\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ & (|\mathit{props}| = 3) \\ & \wedge\ ((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m) \in \mathit{props}) \\ & \wedge\ \mathit{ValidMeasurand}((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m)) \\ & \wedge\ ((\text{"}\mathit{signal}\text{"}, \mathit{values}_s) \in \mathit{props}) \\ & \wedge\ (|\mathit{values}_s| = 6) \\ & \wedge\ \mathit{ValidSignal}((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m), (\text{"}\mathit{signal}\text{"}, \mathit{values}_s)) \end{aligned} \tag{7}$$

$$\mathit{ValidPortProperties}(\text{"}\mathit{audio\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ (|\mathit{props}| = 1) \tag{8}$$

$$\mathit{ValidPortProperties}(\text{"}\mathit{video\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ (|\mathit{props}| = 1) \tag{9}$$

$$\begin{aligned} \mathit{ValidPortProperties}(\text{"}\mathit{custom\_signal}\text{"}, \mathit{props}) \Leftrightarrow\ & (|\mathit{props}| = 2) \\ & \wedge\ ((\text{"}\mathit{custom\_values}\text{"}, \mathit{values}) \in \mathit{props}) \\ & \wedge\ \forall \mathit{value}_1 \in \mathit{values} \cdot \\ & \quad \neg(\exists \mathit{value}_2 \in \mathit{values} \cdot \mathit{value}_1 = \mathit{value}_2) \\ & \wedge\ (|\mathit{values}| \geq 1) \end{aligned} \tag{10}$$

$$\begin{aligned} \mathit{ValidMeasurand}((\text{"}\mathit{measurand}\text{"}, \mathit{values})) \Leftrightarrow\ & ((\text{"}\mathit{lower}\text{"}, \mathit{lower}) \in \mathit{values}) \\ & \wedge\ ((\text{"}\mathit{upper}\text{"}, \mathit{upper}) \in \mathit{values}) \\ & \wedge\ ((\text{"}\mathit{unit}\text{"}, \mathit{unit}) \in \mathit{values}) \\ & \wedge\ \mathit{ValidUnit}(\mathit{unit}) \\ & \wedge\ (\mathit{lower} \leq \mathit{upper}) \end{aligned} \tag{11}$$

$$\begin{aligned} \mathit{ValidSignal}((\text{"}\mathit{measurand}\text{"}, \mathit{values}_m), (\text{"}\mathit{signal}\text{"}, \mathit{values}_s)) \Leftrightarrow\ & \\ & ((\text{"}\mathit{lower}\text{"}, \mathit{lower}_s) \in \mathit{values}_s) \\ & \wedge\ ((\text{"}\mathit{upper}\text{"}, \mathit{upper}_s) \in \mathit{values}_s) \\ & \wedge\ ((\text{"}\mathit{unit}\text{"}, \mathit{unit}_s) \in \mathit{values}_s) \\ & \wedge\ \mathit{ValidUnit}(\mathit{unit}_s) \\ & \wedge\ \mathit{ExtractPrefixFactor}(\mathit{unit}_s, \mathit{factor}_s) \\ & \wedge\ ((\text{"}\mathit{sensitivity}\text{"}, \mathit{sens}) \in \mathit{values}_s) \\ & \wedge\ ((\text{"}\mathit{value}\text{"}, \mathit{value}_{sens}) \in \mathit{sens}) \\ & \wedge\ ((\text{"}\mathit{unit}\text{"}, \mathit{unit}_{sens}) \in \mathit{sens}) \\ & \wedge\ \mathit{ValidUnit}(\mathit{unit}_{sens}) \\ & \wedge\ \mathit{ExtractPrefixFactor}(\mathit{unit}_{sens}, \mathit{factor}_{sens}) \\ & \wedge\ (\mathit{lower}_s \leq \mathit{upper}_s) \\ & \wedge\ ((\text{"}\mathit{lower}\text{"}, \mathit{lower}_m) \in \mathit{values}_m) \\ & \wedge\ ((\text{"}\mathit{upper}\text{"}, \mathit{upper}_m) \in \mathit{values}_m) \\ & \wedge\ ((\text{"}\mathit{unit}\text{"}, \mathit{unit}_m) \in \mathit{values}_m) \\ & \wedge\ \mathit{ValidUnit}(\mathit{unit}_m) \\ & \wedge\ \mathit{ExtractPrefixFactor}(\mathit{unit}_m, \mathit{factor}_m) \\ & \wedge\ \mathit{MultiplyUnits}(\{\mathit{unit}_m\}, \mathit{unit}_{sens}, \mathit{unit}_s) \end{aligned} \tag{12}$$

$$\wedge\ (lower_m * factor_m * value_{sens} * factor_{sens} = lower_s * factor_s)$$
$$\wedge\ (upper_m * factor_m * value_{sens} * factor_{sens} = upper_s * factor_s)$$

B.2. Edge Validity

In this section, all validity constraints regarding edges are listed without any further explanations.

$$ValidEdges(E, Pr) \Leftrightarrow \forall e \in E \cdot ValidEdge(e, Pr) \tag{13}$$

$$\begin{aligned} ValidEdge((p_f, p_t), Pr) \Leftrightarrow\ & (\exists (key_f, value_f) \in Pr \cdot key_f = getUid(p_f)) \\ & \wedge\ (\exists (key_t, value_t) \in Pr \cdot key_t = getUid(p_t)) \\ & \wedge\ MatchingProperties(value_f, value_t) \end{aligned} \tag{14}$$

$$\begin{aligned} MatchingProperties((props_f, props_t)) \Leftrightarrow\ & \forall (key_f, value_f) \in props_f \cdot \\ & \exists (key_t, value_t) \in props_t \cdot \\ & MatchingProperty((key_f, value_f), (key_t, value_t)) \end{aligned} \tag{15}$$

$$\begin{aligned} MatchingProperty((key_f, value_f), (key_t, value_t)) \Leftrightarrow\ & (key_f \in \{"signal\_type", "sampling\_rate", \\ & "resolution", "custom\_values", "value"\}) \\ & \wedge\ (key_f = key_t) \\ & \wedge\ (value_f = value_t) \end{aligned} \tag{16}$$

$$\begin{aligned} MatchingProperty((key_f, value_f), (key_t, value_t)) \Leftrightarrow\ & (key_f \in \{"measurand", "signal", "sensitivity"\}) \\ & \wedge\ (key_f = key_t) \\ & \wedge\ MatchingProperties(value_f, value_t) \end{aligned} \tag{17}$$

$$\begin{aligned} &MatchingProperty((\text{"}unit\text{"}, value_f), (key_t, value_t)) \Leftrightarrow \\ &\quad (key_t = \text{"}unit\text{"}) \\ &\wedge\ ExtractPrefixFactor(value_f, factor_f) \\ &\wedge\ ExtractPrefixFactor(value_t, factor_t) \\ &\wedge\ (factor_f = factor_t) \\ &\wedge\ EqualUnits(value_f, value_t) \end{aligned} \tag{18}$$

$$\begin{aligned} &MatchingProperty((\text{"}lower\text{"}, value_f), (key_t, value_t)) \Leftrightarrow \\ &\quad (key_t = \text{"}lower\text{"}) \\ &\wedge\ (value_f \geq value_t) \end{aligned} \tag{19}$$

$$\begin{aligned} &MatchingProperty((\text{"}upper\text{"}, value_f), (key_t, value_t)) \Leftrightarrow \\ &\quad (key_t = \text{"}upper\text{"}) \\ &\wedge\ (value_f \leq value_t) \end{aligned} \tag{20}$$

B.3. Node Validity

In this section, all validity constraints regarding nodes are listed without any further explanations.

$$ValidNodes(N, Pr, E) \Leftrightarrow \forall n \in N \cdot ValidNode(n, N, Pr, E) \tag{21}$$

$$\begin{aligned} &ValidNode(n, N, Pr, E) \Leftrightarrow\ (\exists prop_n \in Pr \cdot getKey(prop_n) = getUid(n)) \\ &\quad \wedge\ \exists(\text{"}input\_ports\text{"}, inputs) \in getValue(prop_n) \\ &\quad \wedge\ \exists(\text{"}output\_ports\text{"}, outputs) \in getValue(prop_n) \\ &\quad \wedge\ DisjointPorts(inputs, outputs) \\ &\quad \wedge\ DisjointNode(n, inputs, outputs, N, Pr) \\ &\quad \wedge\ IncomingConnections(inputs, E) \\ &\quad \wedge\ IncomingConnected(inputs, E) \end{aligned} \tag{22}$$

$$\wedge\ OutgoingConnections(outputs, E)$$
$$\wedge\ \exists(\text{"}node\_type\text{"}, type) \in getValue(prop_n)$$
$$\wedge\ MatchingPorts(n, type, inputs, outputs, Pr)$$

$$\begin{aligned} MatchingPorts(n, \text{"}nested\_node\text{"}, inputs, outputs, Pr) \Leftrightarrow \\ (\exists(key_n, values_n) \in Pr \cdot key_n = getUid(n)) \\ \wedge\ ((\text{"}graph\text{"}, (N_n, E_n, Po_n, Pr_n)) \in values_n) \\ \wedge\ (|N_n| = |E_n| = |Po_n| = |Pr_n| = 0) \\ \wedge\ (\forall output \in \{out \mid (out \in outputs) \\ \wedge\ ((getUId(out), values_o) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, type) \in values_o) \\ \wedge\ (type \in \{\text{"}analogue\_signal\text{"}, \text{"}converted\_signal\text{"}\})\} \cdot \\ DerivableUnit(inputs, output, Pr)) \end{aligned} \tag{23}$$

$$\begin{aligned} MatchingPorts(n, \text{"}nested\_node\text{"}, inputs, outputs, Pr) \Leftrightarrow \\ (\exists(key_n, values_n) \in Pr \cdot key_n = getUid(n)) \\ \wedge\ ((\text{"}graph\text{"}, (N_n, E_n, Po_n, Pr_n)) \in values_n) \\ \wedge\ (inputs \subset Po_n) \\ \wedge\ (\forall input \in inputs \cdot ((getUid(input), values_i) \in Pr) \\ \wedge\ ((getUid(input), values_{ni}) \in Pr_n) \wedge (values_i = values_{ni})) \\ \wedge\ OutgoingConnections(inputs, E_n) \\ \wedge\ (outputs \subset Po_n) \\ \wedge\ (\forall output \in outputs \cdot ((getUid(output), values_o) \in Pr) \\ \wedge\ ((getUid(output), values_{no}) \in Pr_n) \wedge (values_o = values_{no})) \\ \wedge\ IncomingConnections(outputs, E_n) \\ \wedge\ ValidGraph((N_n, E_n, Po_n, Pr_n)) \end{aligned} \tag{24}$$

$$\begin{aligned} MatchingPorts(\_, \text{"}composed\_output\text{"}, inputs, \emptyset, Pr) \Leftrightarrow \\ (|inputs| \geq 2) \\ \wedge\ (\forall input \in Inputs \cdot ((getUid(input), values_i) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}analogue\_signal\text{"}) \in values_i) \end{aligned} \tag{25}$$

$$\vee\ (\text{"}signal\_type\text{"}, \text{"}converted\_signal\text{"}) \in values_i))$$

$$\begin{aligned} MatchingPorts(\_, \text{"}sensor\text{"}, \{input\}, \{output\}, Pr) \Leftrightarrow \\ ((getUid(input), values_i) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}analogue\_signal\text{"}) \in values_i) \\ \wedge\ ((\text{"}measurand\text{"}, \_) \in values_i) \\ \wedge\ (|values_i| = 2) \\ \wedge\ ((getUid(output), values_o) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}analogue\_signal\text{"}) \in values_o) \\ \wedge\ (values_i \subset values_o) \end{aligned} \tag{26}$$

$$\begin{aligned} MatchingPorts(\_, \text{"}audio\_recorder\text{"}, \{input\}, \{output\}, Pr) \Leftrightarrow \\ ((getUid(input), values_i) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}audio\_signal\text{"}) \in values_i) \\ \wedge\ ((getUid(output), values_o) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}audio\_signal\text{"}) \in values_o) \\ \wedge\ (values_i \subset values_o) \end{aligned} \tag{27}$$

$$\begin{aligned} MatchingPorts(\_, \text{"}video\_recorder\text{"}, \{input\}, \{output\}, Pr) \Leftrightarrow \\ ((getUid(input), values_i) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}video\_signal\text{"}) \in values_i) \\ \wedge\ ((getUid(output), values_o) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}video\_signal\text{"}) \in values_o) \\ \wedge\ (values_i \subset values_o) \end{aligned} \tag{28}$$

$$\begin{aligned} MatchingPorts(n, \text{"}measuring\_module\text{"}, inputs, outputs, Pr) \Leftrightarrow \\ (\forall input \in inputs \cdot ((getUid(input), values_i) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}analogue\_signal\text{"}) \in values_i)) \\ \wedge\ (\forall output \in outputs \cdot ((getUid(output), values_o) \in Pr) \\ \wedge\ ((\text{"}signal\_type\text{"}, \text{"}converted\_signal\text{"}) \in values_o)) \end{aligned} \tag{29}$$

$$
\begin{aligned}
&\wedge\ ((getUid(n), values_n) \in Pr) \\
&\wedge\ (("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n) \\
&\wedge\ (|N_n| = |E_n| = |Po_n| = |Pr_n| = 0) \\
&\wedge\ (\forall output \in outputs \cdot \\
&\quad DerivableUnit(inputs, output, Pr))
\end{aligned}
$$

$$
\begin{aligned}
&MatchingPorts(n, "measuring\_module", inputs, outputs, Pr) \Leftrightarrow \qquad (30) \\
&\quad (\forall input \in inputs \cdot ((getUid(input), values_i) \in Pr) \\
&\quad\ \wedge (("signal\_type", "analogue\_signal") \in values_i)) \\
&\wedge\ (\forall output \in outputs \cdot ((getUid(output), values_o) \in Pr) \\
&\quad\ \wedge (("signal\_type", "converted\_signal") \in values_o)) \\
&\quad (\exists (key_n, values_n) \in Pr \cdot key_n = getUid(n)) \\
&\wedge\ (("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n) \\
&\wedge\ (inputs \subset Po_n) \\
&\wedge\ (\forall input \in inputs \cdot ((getUid(input), values_i) \in Pr) \\
&\quad\ \wedge ((getUid(input), values_{ni}) \in Pr_n) \wedge (values_i = values_{ni})) \\
&\wedge\ OutgoingConnections(inputs, E_n) \\
&\wedge\ (outputs \subset Po_n) \\
&\wedge\ (\forall output \in outputs \cdot ((getUid(output), values_o) \in Pr) \\
&\quad\ \wedge ((getUid(output), values_{no}) \in Pr_n) \wedge (values_o = values_{no})) \\
&\wedge\ IncomingConnections(outputs, E_n) \\
&\wedge\ ValidGraph((N_n, E_n, Po_n, Pr_n))
\end{aligned}
$$

$$
\begin{aligned}
&MatchingPorts(n, "analysis\_module", inputs, outputs, Pr) \Leftrightarrow \qquad (31) \\
&\quad (\exists (key_n, values_n) \in Pr \cdot key_n = getUid(n)) \\
&\wedge\ (("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n) \\
&\wedge\ (|N_n| = |E_n| = |Po_n| = |Pr_n| = 0) \\
&\wedge\ (\forall output \in \{out \mid (out \in outputs) \\
&\quad\ \wedge ((getUId(out), values_o) \in Pr) \\
&\quad\ \wedge (("signal\_type", type) \in values_o) \\
&\quad\ \wedge (type \in \{"analogue\_signal", "converted\_signal"\})\} \cdot \\
&\quad DerivableUnit(inputs, output, Pr))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{MatchingPorts}(n, \text{"}\mathit{analysis\_module}\text{"}, \mathit{inputs}, \mathit{outputs}, \mathit{Pr}) \Leftrightarrow \\
&\quad (\exists(\mathit{key}_n, \mathit{values}_n) \in \mathit{Pr} \cdot \mathit{key}_n = \mathit{getUid}(n)) \\
&\wedge\ ((\text{"}\mathit{graph}\text{"}, (N_n, E_n, \mathit{Po}_n, \mathit{Pr}_n)) \in \mathit{values}_n) \\
&\wedge\ (\mathit{inputs} \subset \mathit{Po}_n) \\
&\wedge\ (\forall \mathit{input} \in \mathit{inputs} \cdot ((\mathit{getUid}(\mathit{input}), \mathit{values}_i) \in \mathit{Pr}) \\
&\quad \wedge\ ((\mathit{getUid}(\mathit{input}), \mathit{values}_{ni}) \in \mathit{Pr}_n) \wedge (\mathit{values}_i = \mathit{values}_{ni})) \\
&\wedge\ \mathit{OutgoingConnections}(\mathit{inputs}, E_n) \\
&\wedge\ (\mathit{outputs} \subset \mathit{Po}_n) \\
&\wedge\ (\forall \mathit{output} \in \mathit{outputs} \cdot ((\mathit{getUid}(\mathit{output}), \mathit{values}_o) \in \mathit{Pr}) \\
&\quad \wedge\ ((\mathit{getUid}(\mathit{output}), \mathit{values}_{no}) \in \mathit{Pr}_n) \wedge (\mathit{values}_o = \mathit{values}_{no})) \\
&\wedge\ \mathit{IncomingConnections}(\mathit{outputs}, E_n) \\
&\wedge\ \mathit{ValidGraph}((N_n, E_n, \mathit{Po}_n, \mathit{Pr}_n))
\end{aligned}
\tag{32}
$$

$$
\begin{aligned}
&\mathit{MatchingPorts}(n, \text{"}\mathit{amplifier}\text{"}, \{\mathit{input}\}, \{\mathit{output}\}, \mathit{Pr}) \Leftrightarrow \\
&\quad ((\mathit{getUid}(\mathit{input}), \mathit{values}_i) \in \mathit{Pr}) \\
&\wedge\ ((\text{"}\mathit{signal\_type}\text{"}, \mathit{type}_i) \in \mathit{values}_i) \\
&\wedge\ (\mathit{type}_i \in \{\text{"}\mathit{analogue\_signal}\text{"}, \text{"}\mathit{converted\_signal}\text{"}\}) \\
&\wedge\ ((\mathit{getUid}(\mathit{output}), \mathit{values}_o) \in \mathit{Pr}) \\
&\wedge\ ((\text{"}\mathit{signal\_type}\text{"}, \mathit{type}_o) \in \mathit{values}_o) \\
&\wedge\ (\mathit{type}_o \in \{\text{"}\mathit{analogue\_signal}\text{"}, \text{"}\mathit{converted\_signal}\text{"}\}) \\
&\wedge\ (\mathit{type}_i = \mathit{type}_o) \\
&\wedge\ ((\text{"}\mathit{measurand}\text{"}, \mathit{values}_{mi}) \in \mathit{values}_i) \\
&\wedge\ ((\text{"}\mathit{measurand}\text{"}, \mathit{values}_{mo}) \in \mathit{values}_o) \\
&\wedge\ (\mathit{values}_{mi} = \mathit{values}_{mo}) \\
&\wedge\ ((\text{"}\mathit{signal}\text{"}, \mathit{values}_{si}) \in \mathit{values}_i) \\
&\wedge\ ((\text{"}\mathit{lower}\text{"}, \mathit{lower}_{si}) \in \mathit{values}_{si}) \\
&\wedge\ ((\text{"}\mathit{upper}\text{"}, \mathit{upper}_{si}) \in \mathit{values}_{si}) \\
&\wedge\ ((\text{"}\mathit{sensitivity}\text{"}, \mathit{values}_{sensi}) \in \mathit{values}_{si}) \\
&\wedge\ ((\text{"}\mathit{value}\text{"}, \mathit{value}_{sensi}) \in \mathit{values}_{sensi}) \\
&\wedge\ ((\text{"}\mathit{signal}\text{"}, \mathit{values}_{so}) \in \mathit{values}_o) \\
&\wedge\ ((\text{"}\mathit{lower}\text{"}, \mathit{lower}_{so}) \in \mathit{values}_{so})
\end{aligned}
\tag{33}
$$

$$
\begin{aligned}
&\wedge (("upper", upper_{so}) \in values_{so}) \\
&\wedge (("sensitivity", values_{senso}) \in values_{so}) \\
&\wedge (("value", value_{senso}) \in values_{senso}) \\
&\wedge ((getUid(n), values_n) \in Pr) \\
&\wedge (("scale\_factor", factor_n) \in values_n) \\
&\wedge (lower_{si} * factor_n = lower_{so}) \\
&\wedge (upper_{si} * factor_n = upper_{so}) \\
&\wedge (value_{sensi} * factor_n = value_{senso}) \\
&\wedge (values_{si} \setminus \{("lower", \_), ("upper", \_), ("sensitivity", \_)\} \\
&\quad = values_{so} \setminus \{("lower", \_), ("upper", \_), ("sensitivity", \_)\})
\end{aligned}
$$

$$
\begin{aligned}
&MatchingPorts(n, "analogue\_digital\_converter", \{input\}, \{output\}, Pr) \Leftrightarrow \quad (34) \\
&\quad ((getUid(input), values_i) \in Pr) \\
&\wedge (("signal\_type", "analogue\_signal") \in values_i) \\
&\wedge ((getUid(output), values_o) \in Pr) \\
&\wedge (("signal\_type", "converted\_signal") \in values_o) \\
&\wedge (("signal", values_{si}) \in values_i) \\
&\wedge (("signal", values_{so}) \in values_o) \\
&\wedge (values_{si} \subset values_{so}) \\
&\wedge (("resolution", res_{so}) \in values_{so}) \\
&\wedge (("sampling\_rate", samp_{so}) \in values_{so}) \\
&\wedge ((getUid(n), values_n) \in Pr) \\
&\wedge (("resolution", res_n) \in values_n) \\
&\wedge (("sampling\_rate", samp_n) \in values_n) \\
&\wedge (res_n = res_{so}) \\
&\wedge (samp_n = samp_{so})
\end{aligned}
$$

$$
\begin{aligned}
&MatchingPorts(n, "filter", \{input\}, \{output\}, Pr) \Leftrightarrow \quad (35) \\
&\quad ((getUid(input), values_i) \in Pr) \\
&\wedge (("signal\_type", type_i) \in values_i) \\
&\wedge (type_i \in \{"analogue\_signal", "converted\_signal"\}) \\
&\wedge ((getUid(output), values_o) \in Pr)
\end{aligned}
$$

$$
\begin{aligned}
&\wedge (("signal\_type", type_o) \in values_o) \\
&\wedge (type_o \in \{"analogue\_signal", "converted\_signal"\}) \\
&\wedge (type_i = type_o) \\
&\wedge (("measurand", values_{mi}) \in values_i) \\
&\wedge (("measurand", values_{mo}) \in values_o) \\
&\wedge (("unit", unit_{mi}) \in values_{mi}) \\
&\wedge (("unit", unit_{mo}) \in values_{mo}) \\
&\wedge EqualUnits(unit_{mi}, unit_{mo}) \\
&\wedge (("signal", values_{si}) \in values_i) \\
&\wedge (("lower", lower_{si}) \in values_{si}) \\
&\wedge (("upper", upper_{si}) \in values_{si}) \\
&\wedge (("signal", values_{so}) \in values_o) \\
&\wedge (("lower", lower_{so}) \in values_{so}) \\
&\wedge (("upper", upper_{so}) \in values_{so}) \\
&\wedge ((getUid(n), values_n) \in Pr) \\
&\wedge (("range", values_{rn}) \in values_n) \\
&\wedge (("lower", lower_{rn}) \in values_{rn}) \\
&\wedge (("upper", upper_{rn}) \in values_{rn}) \\
&\wedge (((lower_{rn} \geq lower_{si}) \wedge (lower_{so} = lower_{rn})) \\
&\quad \vee ((lower_{rn} \leq lower_{si}) \wedge (lower_{so} = lower_{si}))) \\
&\wedge (((upper_{rn} \geq upper_{si}) \wedge (upper_{so} = upper_{si})) \\
&\quad \vee ((upper_{rn} \leq upper_{si}) \wedge (upper_{so} = upper_{rn}))) \\
&\wedge (values_{si} \setminus \{("lower", \_), ("upper", \_)\} \\
&\quad = values_{so} \setminus \{("lower", \_), ("upper", \_)\})
\end{aligned}
$$

$$
\begin{aligned}
MatchingPorts(n, "trigger", \_, \{output\}, Pr) \Leftrightarrow \quad & (36) \\
((getUid(n), values_n) \in Pr) & \\
\wedge (("main\_port", p) \in values_n) & \\
\wedge ((getUid(p), values_p) \in Pr) & \\
\wedge ((getUid(output), values_o) \in Pr) & \\
\wedge (values_p = values_o) &
\end{aligned}
$$

$$DisjointPorts(inputs, outputs) \Leftrightarrow inputs \cap outputs = \emptyset \tag{37}$$

$$\begin{aligned} &DisjointNode(n, inputs, outputs, N, Pr) \Leftrightarrow \\ &\quad \forall other \in \{N \setminus \{n\}\} \cdot \\ &\quad \wedge ((getUid(other), values_o) \in Pr) \\ &\quad \wedge (("input\_ports", inputs_o) \in values_o) \\ &\quad \wedge (("output\_ports", outputs_o) \in values_o) \\ &\quad \wedge ((inputs_o \cup outputs_o) \cap (inputs \cup outputs) = \emptyset) \end{aligned} \tag{38}$$

$$\begin{aligned} &IncomingConnections(inputs, E) \Leftrightarrow \\ &\quad \forall input \in inputs \cdot \\ &\quad (|\{edge \mid (inputs, \_) \in E\}| = 0) \end{aligned} \tag{39}$$

$$\begin{aligned} &IncomingConnected(inputs, E) \Leftrightarrow \\ &\quad \forall input \in inputs \cdot \\ &\quad (|\{edge \mid (\_, inputs) \in E\}| = 1) \end{aligned} \tag{40}$$

$$\begin{aligned} &OutgoingConnections(outputs, E) \Leftrightarrow \\ &\quad \forall output \in outputs \cdot \\ &\quad (|\{edge \mid (\_, outputs) \in E\}| = 0) \end{aligned} \tag{41}$$

$$\begin{aligned} &DerivableUnit(inputs, output, Pr) \Leftrightarrow \\ &\quad ((getUid(output), values_o) \in Pr) \\ &\wedge (("measurand", values_{mo}) \in values_o) \\ &\wedge (("unit", unit_{mo}) \in values_{mo}) \\ &\wedge (\exists input \in inputs \cdot \\ &\quad ((getUid(input), values_i) \in Pr) \end{aligned} \tag{42}$$

$$\begin{aligned}
&\wedge (("measurand", values_{mi}) \in values_i) \\
&\wedge (("unit", unit_{mi}) \in values_{mi})) \\
&\wedge EqualUnits(unit_{mi}, unit_{mo})
\end{aligned}$$

$$\begin{aligned}
&DerivableUnit(inputs, output, Pr) \Leftrightarrow \\
&\quad ((getUid(output), values_o) \in Pr) \\
&\wedge (("measurand", values_{mo}) \in values_o) \\
&\wedge (("unit", unit_{mo}) \in values_{mo}) \\
&\wedge (selected_inputs \subset inputs) \\
&\wedge (selected_units = \{unit \mid (in \in selected_inputs) \\
&\quad \wedge ((getUid(in), values_{in}) \in Pr) \\
&\quad \wedge (("measurand", values_{min}) \in values_{in}) \\
&\quad \wedge (("unit", unit) \in values_{min})\}) \\
&\wedge (\exists head \in selected_units \cdot \\
&\quad MultiplyUnits(selected_units \setminus head, head, unit_{mo}))
\end{aligned} \tag{43}$$

B.4. Property Validity

In this section, all validity constraints regarding properties are listed without any further explanations.

$$\begin{aligned}
&ValidProperties(Pr) \Leftrightarrow \forall (key_1, value_1) \in Pr \cdot \\
&\qquad \neg(\exists (key_2, value_2) \in Pr \cdot key_1 = key_2)
\end{aligned} \tag{44}$$

B.5. Unit Calculation Helper Predicates

In this section, all validity constraints regarding units are listed. It is important to know that these constraints are far from ideal. The handling of units is a topic that could be covered in a separate thesis. This means, for example, that all units have to be defined in terms of their base unit composition. Derived units are not usable. To better understand the constraints, it is recommended to the reader to have a look at chapter C.
In addition to the syntax defined in section 4.3, some of the predicates in this section make use of the syntax $\{head \mid tail\}$, where *head* denotes the first element

of the list and *tail* denotes the list of all other elements except *head*. It is important to not confuse this with a list comprehension $\{ele \mid constraint(ele)\}$.

$$\begin{aligned} &ValidUnit((numerators, denominators)) \Leftrightarrow \\ &\quad (\forall (prefix_n, unit_n) \in numerators \cdot \\ &\quad SIPrefix(prefix_n) \\ &\quad \wedge (SIBaseUnitunit_n \vee (unit_n = \text{"}none\text{"}))) \\ &\wedge (\forall (prefix_d, unit_d) \in denominators \cdot \\ &\quad SIPrefix(prefix_d) \\ &\quad \wedge (SIBaseUnitunit_d \vee (unit_d = \text{"}none\text{"}))) \end{aligned} \tag{45}$$

$$\begin{aligned} &EqualUnits((num_1, den_1), (num_2, den_2)) \Leftrightarrow \\ &\quad EqualSubUnits(num_1, num_2) \\ &\wedge\ EqualSubUnits(den_1, den_2) \end{aligned} \tag{46}$$

$$\begin{aligned} &MultipyUnits(\{(num, den) \mid tail\}, (num_2, den_2), unit) \Leftrightarrow \\ &\quad ReduceFractionUnits((num \cup num_2), (den \cup den_2), \emptyset, unit_r) \\ &\wedge\ MultiplyUnits(tail, unit_r, unit) \end{aligned} \tag{47}$$

$$\begin{aligned} &MultipyUnits(\{(num, den)\}, (num_2, den_2), unit) \Leftrightarrow \\ &\quad ReduceFractionUnits((num \cup num_2), (den \cup den_2), \emptyset, unit_r) \\ &\wedge\ EqualUnits(unit_r, unit) \end{aligned} \tag{48}$$

$$\begin{aligned} &ExtractPrefixFactor((num, den), factor) \Leftrightarrow \\ &\quad MultiplyFactors(\{prefix_n \mid (prefix_n, \_) \in num\}, 1, factor_n) \\ &\wedge\ MultiplyFactors(\{prefix_d \mid (prefix_d, \_) \in den\}, 1, factor_d) \\ &\wedge\ (factor = factor_n / factor_d) \end{aligned} \tag{49}$$

$$SIBaseUnit(base) \Leftrightarrow base \in \{"m","kg","s","A","K","mol","cd"\} \tag{50}$$

$$\begin{aligned} SIPrefix(prefix) \Leftrightarrow prefix \in \{&"yotta","zetta","exa","peta","tera", \\ &"giga","mega","kilo","hecto","deka","none","deci","centi", \\ &"milli","micro","nano","pico","femto","atto","zepto","yocto"\} \end{aligned} \tag{51}$$

$$\begin{aligned} SIPrefixFactor(prefix, factor) \Leftrightarrow prefix \\ &((prefix = "yotta") \wedge (factor = 10^{24})) \\ \vee\ &((prefix = "zetta") \wedge (factor = 10^{21})) \\ \vee\ &((prefix = "exa") \wedge (factor = 10^{18})) \\ \vee\ &((prefix = "peta") \wedge (factor = 10^{15})) \\ \vee\ &((prefix = "tera") \wedge (factor = 10^{12})) \\ \vee\ &((prefix = "giga") \wedge (factor = 10^{9})) \\ \vee\ &((prefix = "mega") \wedge (factor = 10^{6})) \\ \vee\ &((prefix = "kilo") \wedge (factor = 10^{3})) \\ \vee\ &((prefix = "hecto") \wedge (factor = 10^{2})) \\ \vee\ &((prefix = "deka") \wedge (factor = 10^{1})) \\ \vee\ &((prefix = "none") \wedge (factor = 1)) \\ \vee\ &((prefix = "deci") \wedge (factor = 10^{(-1)})) \\ \vee\ &((prefix = "centi") \wedge (factor = 10^{(-2)})) \\ \vee\ &((prefix = "milli") \wedge (factor = 10^{(-3)})) \\ \vee\ &((prefix = "micro") \wedge (factor = 10^{(-6)})) \\ \vee\ &((prefix = "nano") \wedge (factor = 10^{(-9)})) \\ \vee\ &((prefix = "pico") \wedge (factor = 10^{(-12)})) \\ \vee\ &((prefix = "femto") \wedge (factor = 10^{(-15)})) \\ \vee\ &((prefix = "atto") \wedge (factor = 10^{(-18)})) \\ \vee\ &((prefix = "zepto") \wedge (factor = 10^{(-21)})) \\ \vee\ &((prefix = "yocto") \wedge (factor = 10^{(-24)})) \end{aligned} \tag{52}$$

$$ReduceFractionUnits(\emptyset, den, nom, unit) \Leftrightarrow unit = (nom, den) \tag{53}$$

$$\begin{aligned} ReduceFractionUnits(\{(prefix_c, unit_c) \mid tail\}, den, nom, unit) \Leftrightarrow \\ (unit_c = \text{"}none\text{"}) \\ \wedge\ ReduceFractionUnits(tail, den, \\ (\{(prefix_c, unit_c)\} \cup nom), unit) \end{aligned} \tag{54}$$

$$\begin{aligned} ReduceFractionUnits(\{(prefix_c, unit_c) \mid tail\}, den, nom, unit) \Leftrightarrow \\ SIBaseUnit(unit_c) \\ \wedge\ \neg((\_, unit_c) \in den) \\ \wedge\ ReduceFractionUnits(tail, den, \\ (\{(prefix_c, unit_c)\} \cup nom), unit) \end{aligned} \tag{55}$$

$$\begin{aligned} ReduceFractionUnits(\{(prefix_c, unit_c) \mid tail\}, den, nom, unit) \Leftrightarrow \\ SIBaseUnit(unit_c) \\ \wedge\ ((prefix_d, unit_c) \in den) \\ \wedge\ DeleteFirst((prefix_d, unit_c), den, den_{interim}) \\ \wedge\ ReduceFractionUnits(tail, \\ (\{(prefic_d, \text{"}none\text{"})\} \cup den_{interim}), \\ (\{(prefix_c, \text{"}none\text{"})\} \cup nom), unit) \end{aligned} \tag{56}$$

$$\begin{aligned} EqualSubUnits(\emptyset, subunits_2) \Leftrightarrow \\ \forall(\_, subunit_2) \in subunits_2 \cdot subunit_2 = \text{"}none\text{"} \end{aligned} \tag{57}$$

$$\begin{aligned} EqualSubUnits(\{(\_, subunit_1) \mid tail\}, subunits_2) \Leftrightarrow \\ (subunit_1 = \text{"}none\text{"}) \wedge EqualSubUnits(tail, subunits_2) \end{aligned} \tag{58}$$

$$\begin{aligned} EqualSubUnits(\{(\_, subunit_1) \mid tail\}, subunits_2) \Leftrightarrow \\ SIBaseUnit(subunit_1) \\ \wedge\ ((subprefix_2, subunit_1) \in subunits_2) \\ \wedge\ DeleteFirst((subprefix_2, subunit_1), subunits_2, subunits_{interim}) \\ \wedge\ EqualSubUnits(tail, subunits_{interim}) \end{aligned} \tag{59}$$

$$\begin{aligned} MultiplyFactors(\{prefix_n\}, acc, factor_n) \Leftrightarrow \\ SIPrefixFactor(prefix_n, x) \\ \wedge\ (factor_n = x * acc) \end{aligned} \tag{60}$$

$$\begin{aligned} MultiplyFactors(\{prefix_n \mid tail\}, acc, factor_n) \Leftrightarrow \\ SIPrefixFactor(prefix_n, x) \\ \wedge\ (acc_{interim} = x * acc) \\ \wedge\ MultiplyFactors(tail, acc_{interim}, factor_n) \end{aligned} \tag{61}$$

$$DeleteFirst(\_, \emptyset, \emptyset) \Leftrightarrow true \tag{62}$$

$$DeleteFirst(term, \{term \mid tail\}, tail) \Leftrightarrow true \tag{63}$$

$$\begin{aligned} DeleteFirst(term, \{head \mid tail\}, \{head \mid result\}) \Leftrightarrow \\ DeleteFirst(term, tail, result) \end{aligned} \tag{64}$$

B.6. Abstraction Level Validity

In this section, all validity constraints regarding abstraction levels are listed without any further explanations.

$$
\begin{aligned}
&\mathit{ValidAL1Graph}((N, E, Po, Pr) \Leftrightarrow\ \forall n \in N \cdot \\
&\qquad (\exists (n, values_n) \in Pr \cdot \\
&\qquad (("node\_type", "nested\_node") \in values_n \\
&\quad \wedge ("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n \\
&\quad \wedge (|N_n| = |E_n| = |Po_n| = |Pr_n| = 0)) \\
&\quad \vee (("node\_type", "composed\_output") \in values_n)) \\
&\quad \wedge (|\{n_n \mid (n_n, values_{n_n}) \in Pr \\
&\qquad \wedge ("node\_type", "nested\_node") \in values_{n_n}\}| = 1) \\
&\quad \wedge\ \mathit{ValidGraph}(N, E, Po, Pr)
\end{aligned}
\tag{65}
$$

$$
\begin{aligned}
&\mathit{ValidAL2Graph}((N, E, Po, Pr) \Leftrightarrow\ \forall n \in N \cdot \\
&\qquad (\exists (n, values_n) \in Pr \cdot \\
&\qquad (("node\_type", "nested\_node") \in values_n \\
&\quad \wedge ("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n \\
&\qquad \wedge \forall n_n \in N_n \cdot \\
&\qquad (\exists (n_n, values_{n_n}) \in Pr \cdot \\
&\qquad (("node\_type", type) \in values_{n_n} \\
&\qquad \wedge\ (type \in \{"measuring\_module", "analysis\_module"\} \\
&\qquad \wedge\ ("graph", (N_{n_n}, E_{n_n}, Po_{n_n}, Pr_{n_n})) \in values_{n_n} \\
&\qquad \wedge\ (|N_{n_n}| = |E_{n_n}| = |Po_{n_n}| = |Pr_{n_n}| = 0)) \\
&\qquad \vee (type \in \{"sensor", "audio\_recorder", "video\_recorder"\})) \\
&\quad \vee (("node\_type", "composed\_output") \in values_n)) \\
&\quad \wedge (|\{n_n \mid (n_n, values_{n_n}) \in Pr \\
&\qquad \wedge ("node\_type", "nested\_node") \in values_{n_n}\}| = 1) \\
&\quad \wedge\ \mathit{ValidGraph}(N, E, Po, Pr)
\end{aligned}
\tag{66}
$$

$$
\begin{aligned}
&\mathit{ValidAL3Graph}((N, E, Po, Pr) \Leftrightarrow\ \forall n \in N \cdot \\
&\qquad (\exists (n, values_n) \in Pr \cdot \\
&\qquad (("node\_type", "nested\_node") \in values_n \\
&\quad \wedge ("graph", (N_n, E_n, Po_n, Pr_n)) \in values_n
\end{aligned}
\tag{67}
$$

$$
\begin{aligned}
&\wedge \forall n_n \in N_n \cdot \\
&(\exists (n_n, values_{n_n}) \in Pr \cdot \\
&((\text{"}node\_type\text{"}, type) \in values_{n_n} \\
&\wedge\ (type \in \{\text{"}measuring\_module\text{"}, \text{"}analysis\_module\text{"}\} \\
&\wedge\ (\text{"}graph\text{"}, (N_{n_n}, E_{n_n}, Po_{n_n}, Pr_{n_n})) \in values_{n_n} \\
&\wedge \forall n_{n_n} \in N_{n_n} \cdot \\
&(\exists (n_{n_n}, values_{n_{n_n}}) \in Pr \cdot \\
&((\text{"}node\_type\text{"}, type) \in values_{n_{n_n}} \\
&\wedge\ (type \in \{\text{"}amplifier\text{"}, \text{"}analogue\_digital\_converter\text{"}, \\
&\text{"}filter\text{"}, \text{"}trigger\text{"}\} \\
&\wedge\ (|N_{n_n}| = |E_{n_n}| = |Po_{n_n}| = |Pr_{n_n}| = 0)) \\
&\vee\ (type \in \{\text{"}sensor\text{"}, \text{"}audio\_recorder\text{"}, \text{"}video\_recorder\text{"}\})) \\
\vee\ &((\text{"}node\_type\text{"}, \text{"}composed\_output\text{"}) \in values_n)) \\
\wedge\ &(|\{n_n \mid (n_n, values_{n_n}) \in Pr \\
&\wedge\ (\text{"}node\_type\text{"}, \text{"}nested\_node\text{"}) \in values_{n_n}\}| = 1) \\
\wedge\ &ValidGraph(N, E, Po, Pr)
\end{aligned}
$$

B.7. Refinement

In this section, all validity constraints regarding refinement are listed without any further explanations.

$$
\begin{aligned}
&RefinedGraph((N_2, E_2, Po_2, Pr_2), (N_1, E_1, Po_1, Pr_1)) \Leftrightarrow (N_1 = N_2) \\
&\wedge\ (E_1 = E_2) \\
&\wedge\ (Po_1 = Po_2) \\
&\wedge \forall prop_1 \in Pr_1 \cdot EnrichedProperty(prop_1, Pr_2, N_1, Po_1)
\end{aligned} \tag{68}
$$

$$
\begin{aligned}
&EnrichedProperty((key, values), Pr, \_, Po)) \Leftrightarrow (\exists p \in Po \cdot key = getUid(p)) \\
&\wedge\ ((key, values_2) \in Pr) \\
&\wedge\ (values \subset values_2)
\end{aligned} \tag{69}
$$

$$
EnrichedProperty((key, values), Pr, N, \_)) \Leftrightarrow (\exists n \in N \cdot key = getUid(n)) \tag{70}
$$

$$\wedge\ ((key, values_2) \in Pr)$$
$$\wedge\ (("node\_type", type) \in values)$$
$$\wedge\ EnrichedNodeProperty(type, values, values_2)$$

$$EnrichedNodeProperty(type, values_1, values_2) \Leftrightarrow \tag{71}$$
$$(type \in \{"nested\_node", "measuring\_module", "analysis\_module"\})$$
$$\wedge\ (("graph", (N, E, Po, Pr)) \in values_1)$$
$$\wedge\ (|N| = |E| = |Po| = |Pr| = 0)$$
$$\wedge\ (values_1 \setminus ("graph", (N, E, Po, Pr)) \subset values_2)$$

$$EnrichedNodeProperty(type, values_1, values_2) \Leftrightarrow \tag{72}$$
$$(type \in \{"nested\_node", "measuring\_module", "analysis\_module"\})$$
$$\wedge\ (("graph", (N_1, E_1, Po_1, Pr_1)) \in values_1)$$
$$\wedge\ \neg(|N_1| = |E_1| = |Po_1| = |Pr_1| = 0)$$
$$\wedge\ (("graph", (N_2, E_2, Po_2, Pr_2)) \in values_2)$$
$$\wedge\ \neg(|N_2| = |E_2| = |Po_2| = |Pr_2| = 0)$$
$$\wedge\ (values_1 \setminus ("graph", (N_1, E_1, Po_1, Pr_1)) \subset values_2 \setminus ("graph", (N_2, E_2, Po_2, Pr_2)))$$
$$\wedge\ RefinedGraph((N_2, E_2, Po_2, Pr_2), (N_1, E_1, Po_1, Pr_1))$$

$$EnrichedNodeProperty(type, values_1, values_2) \Leftrightarrow \tag{73}$$
$$\neg(type \in \{"nested\_node", "measuring\_module", "analysis\_module"\})$$
$$\wedge\ (values_1 \subset values_2)$$

C. SI Base and Derived Units

The tables shown in this section are taken from [11]. This paper also describes how to correctly use International System of Units (SI) units.

C.1. SI Base Units

Measurand	**Unit name**	**Symbol**
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd

Table C.1.: SI base units

C.2. SI Derived Units

Measurand	**Unit name**	**Symbol**
area	square meter	m^2
volume	cubic meter	m^3
speed, velocity	meter per second	m/s
acceleration	meter per second squared	m/s^2
wavenumber	reciprocal meter	m^{-1}
density, mass density	kilogram per cubic meter	kg/m^3
specific volume	cubic meter per kilogram	m^3/kg
current density	ampere per square meter	A/m^2
magnetic field strength	ampere per meter	A/m
luminance	candela per square meter	cd/m^2
amount-of-substance concentration	mole per cubic meter	mol/m^3

Table C.2.: Examples of SI derived units

C.3. SI Derived Units with Special Names and Symbols

Measurand	**Special name**	**Special symbol**	**Expression in terms of other SI units**	**Expression in terms of SI base units**
plane angle	radian	rad	1	m/m
solid angle	steradian	sr	1	m^2/m^2
frequency	hertz	Hz		s^{-1}
force	newton	N		$m \cdot kg \cdot s^{-2}$
pressure, stress	pascal	Pa	N/m^2	$m^{-1} \cdot kg \cdot s^{-2}$
energy, work, amount of heat	joule	J	$N \cdot m$	$m^2 \cdot kg \cdot s^{-2}$
power, radiant flux	watt	W	J/s	$m^2 \cdot kg \cdot s^{-3}$
electric charge, amount of electricity	coulomb	C		$s \cdot A$
electric potential difference, electromotive force	volt	V	W/A	$m^2 \cdot kg \cdot s^{-3} \cdot A^{-1}$
capacitance	farad	F	C/V	$m^{-2} \cdot kg^{-1} \cdot s^4 \cdot A^2$
electric resistance	ohm	Ω	V/A	$m^2 \cdot kg \cdot s^{-3} \cdot A^{-2}$
electric conductance	siemens	S	A/V	$m^{-2} \cdot kg^{-1} \cdot s^3 \cdot A^2$
magnetic flux	weber	Wb	$V \cdot s$	$m^2 \cdot kg \cdot s^{-2} \cdot A^{-1}$
magnetic flux density	tesla	T	Wb/m^2	$kg \cdot s^{-2} \cdot A^{-1}$
inductance	henry	H	Wb/A	$m^2 \cdot kg \cdot s^{-2} \cdot A^{-2}$
Celsius temperature	degree Celsius	C		K
luminous flux	lumen	lm	$cd \cdot sr$	Cd
illuminance	lux	lx	lm/m^2	$m^{-2} \cdot cd$
activity referred to a radionuclide	becquerel	Bq		s^{-1}

absorbed dose, specific energy (imparted), kerma	gray	Gy	J/kg	$m^2 \cdot s^{-2}$
dose equivalent, ambient dose equivalent, directional dose equivalent, personal dose equivalent	sievert	Sv	J/kg	$m^2 \cdot s^{-2}$
catalytic activity	katal	kat		$s^{-1} \cdot mol$

Table C.3.: SI derived units with special names

C.4. SI Prefixes

Factor	**Prefix name**	**Symbol**	**Factor**	**Prefix name**	**Symbol**
10^{24}	yotta	Y	10^{-1}	deci	d
10^{21}	zetta	Z	10^{-2}	centi	c
10^{18}	exa	E	10^{-3}	milli	m
10^{15}	peta	P	10^{-6}	micro	μ
10^{12}	tera	T	10^{-9}	nano	n
10^{9}	giga	G	10^{-12}	pico	p
10^{6}	mega	M	10^{-15}	femto	f
10^{3}	kilo	k	10^{-18}	atto	a
10^{2}	hecto	h	10^{-21}	zepto	z
10^{1}	deka	da	10^{-24}	yocto	y

Table C.4.: SI prefixes

D. Agreement for Project Thesis

Student: Mario Meili
Semester: HS 2016/17
Advisor: Prof. Dr. Farhad Mehta
Project Partner: Kistler Instrumente AG
Project Start Date: 19.09.2016
Project End Date: 12.01.2017
ECTS-Credits: 12 ECTS Credits

Project Title

Development of a Modelling Procedure for the Description of Test Set-Ups with Measuring Instruments

Goals and Project Description

Numerous industrial sectors require a high degree of precision and the resulting assurance of quality. Kistler Instrumente AG develops and produces sensors and measuring systems for measuring pressure, force, acceleration and torque. This enables their customers to measure and analyse physical processes, to monitor industrial processes and to optimise their products.
The growing palette of products of Kistler Instrumente AG, as well as the requirements of their customers, results in an increased complexity of test set-up modelling. The main difficulty in doing so is to find a level of abstraction that is most appropriate.
Based on gathered requirements for a generic data acquisition system, different possible abstract descriptions of measuring tasks are to be developed exploratively and to be evaluated. The main goal of this project thesis is therefore the planning, design and —as far as possible within the scope of the project— explorative implementation of a prototype of a modelling system for test set-ups. This shall demonstrate the feasibility of such a modelling environment and —if possible— lay the foundation for future work.

Expected Results

- Technical report
- Poster
- Implementation of a prototype system for the modelling of test set-ups
- Documentation of the prototype
- DVD containing all documents and the source code

Competencies to Be Gained (Professional, Methodological and Self-Competence)

- Modelling of measuring systems
- Planning, design and evaluation of the corresponding implementation
- Structured and independent execution of a project thesis
- Contribute to the state of technology in collaboration with an industrial partner

Description of the Assessment of Performance

According to the module description SWSY_PJ:

1. Overall assessment criteria: originality and innovative character of the projects results, target achievement
2. Organisation and execution of the project task assessment criteria: formulation and updating of the topic and the planning of the project, organisation of tasks according to the project schedule, independent work, commitment, collaboration with employer, team members and advisor
3. Report assessment criteria: content of the report, structure and presentation of the documentation, language of the documentation
4. Presentation: consideration of the target audience, language, content
5. Content assessment criteria: preliminary study, requirement analysis, design (system architecture, user interface), complexity and scope, quality of the code, documentation

Further assessment criteria can be defined by the advisor during the specification of the project topic.