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

Prototype Documentation

University of Applied Sciences Rapperswil

Autumn Semester 2016/17

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

Contents

1 Introduction

This document describes the prototype developed for thesis [2]. The goal of the thesis was to develop a modelling procedure for the description of test set-ups with measuring instruments. One particular goal was the automated validation of modelled test set-ups. To achieve that, a set of validity constraints was defined, using formulae in first-order logic.

These constraints have been implemented in Prolog, a logic based programming language.

1.1 Document Structure

In section 2, the development environment used to implement the prototype is described. This includes the listing of necessary tools, a manual on how to use the code and short overview of the testing process. Section 3 lists the full source code.

2 Development Environment

This section gives a brief insight on how to run the prototype and its unit tests. The necessary tools and a short tutorial are described in the following subsections.

2.1 Tools

The prototype was implemented in Prolog. In particular, the implementation SWI Prolog in version 7.2.3 was used. To download and install SWI Prolog, follow the link `http://www.swi-prolog.org/download/stable`.

In addition, the framework plunit was used to implement and run the unit tests for the prototype. This framework is part of the standard library of SWI Prolog and therefore, no installation is needed. The documentation of plunit can be found by following the link `http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/plunit.html%27)`.

2.2 User Manual

To use the prototype, download the source code from [1] and follow the steps below.

1. Open a command line or terminal window.
2. Navigate to the directory where the source files are located.
3. Run the command `swipl` to start SWI Prolog.
4. Load all predicates by typing the query `[graph].`. Note that there is a colon at the end of the query which is part of the syntax of prolog. This will load the file `graph.pl`, which in turn loads all the other necessary files.
5. Run the query of your choice. For example:

```
validEdge(
    (p1,p2),
    [
      (p1,[(signal_type,audio_signal)]),
      (p2,[(signal_type,audio_signal)])
    ]
  ).
```

Note that in order to see changes made by you to source files, you have to reload `graph.pl` after saving the changes.
The prototype has one feature, that makes its usage much more useful. In addition to tell if a graph is valid or not, it can, in the latter case, return a list of all elements that are wrongly modelled. To test a graph for validity, use the following query:

```
validGraph(graph(...),Errors).
```

`Errors` is a variable. As result of the evaluation, Prolog will return this bound variable and it will be the list of all errors in the model.

2.3 Unit Tests

To run the unit tests for the prototype, download the source code from [1] and follow the steps below.

1. Open a command line or terminal window.
2. Navigate to the directory where the source files are located.
3. Run the command `swipl` to start SWI Prolog.
4. Load all test files by typing the query `[tests].`. Note that there is a colon at the end of the query which is part of the syntax of prolog. This will load the file `tests.pl`, which in turn loads the `graph.pl` and all the other test files.
5. To run all unit tests, execute the query `runValidityTests.`.
6. To run the unit tests for specific files, execute the queries `runGraphValidityTests.`, `runPortValidityTests.`, `runEdgeValidityTests.`, `runNodeValidityTests.`, `runPropertyValidityTests.` or `runUnitValidityTests.`
7. To run a single test, execute the query `run_tests(<filename without extension>:<testname>).`. For example, `run_tests(edge:validSignalTypeEdge).`.

3 Source Code

This section lists the complete source code of the prototype. The structure of subsections represents the file structure of the source files. This makes sense because in Prolog, it is possible to load definitions by file, which is the way it was done for this implementation. The code will be listed without additional comments. The reader of this document is expected to have read [2], where the nature of the validity constraints have been described.

3.1 Graph Validity

The file `graph.pl` gathers validity constraints for the base graph model, abstraction levels and refinement. It can be regarded as the top level entry point for queries regarding graph validity. In Listing 1, the contained source code of `graph.pl` is given.

```
1
2  :- [port]. % validPorts definition
3  :- [edge]. % validEdges definition
4  :- [node]. % validNodes definiton
5  :- [property]. % validProperties definiton
6  :- [unit]. % validUnit definiton
7
8  validGraph(graph(N, E, Po, Pr), Errors) :-
9    is_set(N),
10   is_set(E),
11   is_set(Po),
12   is_set(Pr),
13   validPorts(Po, Pr, E, PortErrors),
14   validEdges(E, Pr, EdgeErrors),
15   validNodes(N, Pr, E, NodeErrors),
16   validProperties(Pr),
17   append(PortErrors, EdgeErrors, InterimErrors),
18   append(NodeErrors, InterimErrors, Errors).
19
20 %*********************************
21 %* ABSTRACTION_LEVEL_PREDICATES **
22 %*********************************
23
24 validAL1Graph(graph(N, E, Po, Pr), Errors) :-
25   forall(member(Node, N), (
26       member((Node, Values), Pr),
27       (
28         member((node_type, nested_node), Values),
29         member((graph, graph([],[],[],[])), Values)
30       );
31       (
32         member((node_type, composed_output), Values)
33       )
34     )
35   ),
36   findall(_, member((_, [(node_type, nested_node) | _]), Pr), NestedNodes),
37   length(NestedNodes, 1),
38   validGraph(graph(N, E, Po, Pr), Errors).
39
```

```
validAL2Graph(graph(N, E, Po, Pr), Errors) :-
  forall(member(Node, N), (
      member((Node, Values), Pr),
      (
        member((node_type, nested_node), Values),
        member((graph, graph(NestedN, _, _, _)), Values),
        forall(member(NestedNode, NestedN), (
            member((NestedNode, NestedValues), Pr),
            (
              member((node_type, Type), NestedValues),
              member(Type, [measuring_module, analysis_module]),
              member((graph, graph([],[],[],[])), NestedValues)
            );
            (
              member((node_type, Type), NestedValues),
              member(Type, [sensor, audio_recorder, video_recorder])
            )
          )
        );
        (
          member((node_type, composed_output), Values)
        )
      )
    )
  ),
  findall(_, member((_, [(node_type, nested_node) | _]), Pr), NestedNodes),
  length(NestedNodes, 1),
  validGraph(graph(N, E, Po, Pr), Errors).

validAL3Graph(graph(N, E, Po, Pr), Errors) :-
  forall(member(Node, N), (
      member((Node, Values), Pr),
      (
        member((node_type, nested_node), Values),
        member((graph, graph(NestedN, _, _, _)), Values),
        forall(member(NestedNode, NestedN), (
            member((NestedNode, NestedValues), Pr),
            (
              member((node_type, Type), NestedValues),
              member(Type, [measuring_module, analysis_module]),
              member((graph, graph(NestedNestedN, _, _, _)), NestedValues),
              forall(member(NestedNestedNode, NestedNestedN), (
                  member((NestedNestedNode, NestedNestedValues), Pr),
                  member((node_type, Type), NestedNestedValues),
                  member(Type, [amplifier, analogue_digital_converter, filter
                      , trigger])
                )
              )
            );
            (
              member((node_type, Type), NestedValues),
              member(Type, [sensor, audio_recorder, video_recorder])
            )
          )
        );
        (
          member((node_type, composed_output), Values)
        )
      )
    )
  ),
  findall(_, member((_, [(node_type, nested_node) | _]), Pr), NestedNodes),
```

```
101   length(NestedNodes, 1),
102   validGraph(graph(N, E, Po, Pr), Errors).
103
104 %*****************************
105 %* GRAPH_REFINEMENT_PREDICATE **
106 %*****************************
107
108 refinedGraph(graph(N2, E2, Po2, Pr2), graph(N1, E1, Po1, Pr1)) :-
109   N1 = N2,
110   E1 = E2,
111   Po1 = Po2,
112   forall(member(Property1, Pr1), (
113       enrichedProperty(Property1, Pr2, N1, Po1)
114     )
115   ).
116
117 %**********************
118 %* HELPER_PREDICATES **
119 %**********************
120
121 enrichedProperty((Name, Values), Properties, _, Ports) :-
122   member(Name, Ports),
123   member((Name, Values2), Properties),
124   subset(Values, Values2).
125
126 enrichedProperty((Name, Values), Properties, Nodes, _) :-
127   member(Name, Nodes),
128   member((Name, Values2), Properties),
129   member((node_type, Type), Values),
130   enrichedNodeProperty(Type, Values, Values2).
131
132 enrichedNodeProperty(Type, Values, Values2) :-
133   member(Type, [nested_node, measuring_module, analysis_module]),
134   member((graph, graph([],[],[],[])), Values),
135   subtract([(graph, graph([],[],[],[]))], Values, ReducedValues),
136   subset(ReducedValues, Values2).
137
138 enrichedNodeProperty(Type, Values, Values2) :-
139   member(Type, [nested_node, measuring_module, analysis_module]),
140   member((graph, graph(NestedN1, NestedE1, NestedPo1, NestedPr1)), Values),
141   member((graph, graph(NestedN2, NestedE2, NestedPo2, NestedPr2)), Values2),
142   not(member((graph, graph([], [], [], [])), Values2)),
143   subtract([(graph, graph(NestedN1, NestedE1, NestedPo1, NestedPr1))], Values
        , ReducedValues),
144   subtract([(graph, graph(NestedN2, NestedE2, NestedPo2, NestedPr2))], Values
        , ReducedValues2),
145   subset(ReducedValues, ReducedValues2),
146   refinedGraph(graph(NestedN2, NestedE2, NestedPo2, NestedPr2), graph(
        NestedN2, NestedE2, NestedPo2, NestedPr2)).
147
148 enrichedNodeProperty(Type, Values, Values2) :-
149   not(member(Type, [nested_node, measuring_module, analysis_module])),
150   subset(Values, Values2).
```

Listing 1: Source code of `graph.pl`

3.2 Port Validity

The predicates to test port validity are gathered in the file `port.pl`. Listing 2 shows the source code defined in this file.

```
1
2  validPorts(Po, Pr, E, Errors) :-
3    findall(Error, (member(Port, Po), invalidPort(Port, Pr, E, Error), nonvar(
         Error)), Errors).
4
5  validPort(Port, Pr, E) :-
6    findall(_, member((_, Port), E), ToEdges),
7    length(ToEdges, Y),
8    (Y = 0 ; Y = 1),
9    findall(_, member((Port, Port), E), SelfEdges),
10   length(SelfEdges, 0),
11   member((Port, Properties), Pr),
12   member((signal_type, Type), Properties),
13   findall(_, member((Port, _), E), FromEdges),
14   (
15     (member(Type, [analogue_signal, audio_signal, video_signal]),
16     length(FromEdges, X),
17     (X = 0 ; X = 1));
18     member(Type, [converted_signal, custom_signal])
19   ),
20   validPortProperties(Type, Properties).
21
22 invalidPort(Port, Pr, E, Error) :-
23   not(validPort(Port, Pr, E)),
24   Error = Port.
25
26 %********************
27 %* ANALOGUE_SIGNAL **
28 %********************
29
30 validPortProperties(analogue_signal, Properties) :-
31   length(Properties, 2), % has signal_type and measurand
32   member((measurand, Values), Properties),
33   validMeasurand((measurand, Values)).
34
35 validPortProperties(analogue_signal, Properties) :-
36   length(Properties, 3), % has signal_type, measurand and signal
37   member((measurand, ValuesM), Properties),
38   validMeasurand((measurand, ValuesM)),
39   member((signal, ValuesS), Properties),
40   length(ValuesS, 4), % has lower, upper, unit and sensitivity
41   validSignal((measurand, ValuesM), (signal, ValuesS)).
42
43 %*********************
44 %* CONVERTED_SIGNAL **
45 %*********************
46
47 validPortProperties(converted_signal, Properties) :-
48   length(Properties, 2), % has signal_type and measurand
49   member((measurand, Values), Properties),
50   validMeasurand((measurand, Values)).
51
52 validPortProperties(converted_signal, Properties) :-
53   length(Properties, 3), % has signal_type, measurand and signal
54   member((measurand, ValuesM), Properties),
55   validMeasurand((measurand, ValuesM)),
```

```
56    member((signal, ValuesS), Properties),
57    length(ValuesS, 6), % has lower, upper, unit, sensitivity, sampling_rate
          and resolution
58    validSignal((measurand, ValuesM), (signal, ValuesS)).
59
60  %*****************
61  %* AUDIO_SIGNAL **
62  %*****************
63
64  validPortProperties(audio_signal, Properties) :-
65    length(Properties, 1). % has no further properties
66
67  %*****************
68  %* VIDEO_SIGNAL **
69  %*****************
70
71  validPortProperties(video_signal, Properties) :-
72    length(Properties, 1). % has no further properties
73
74  %******************
75  %* CUSTOM_SIGNAL **
76  %******************
77
78  validPortProperties(custom_signal, Properties) :-
79    length(Properties, 2), % has signal_type and custom_values
80    member((custom_values, Values), Properties),
81    is_set(Values),
82    length(Values, X),
83    X >= 1.
84
85  %**********************
86  %* HELPER_PREDICATES **
87  %**********************
88
89  validMeasurand((measurand, Values)) :-
90    member((lower, Lower), Values),
91    member((upper, Upper), Values),
92    member((unit, Unit), Values),
93    validUnit(Unit),
94    Lower =< Upper.
95
96  validSignal((measurand, ValuesM), (signal, ValuesS)) :-
97    member((lower, LowerS), ValuesS),
98    member((upper, UpperS), ValuesS),
99    member((unit, UnitS), ValuesS),
100   validUnit(UnitS),
101   extractPrefixFactor(UnitS, FactorS),
102   member((sensitivity, SensitivityS), ValuesS),
103   member((value, ValueSens), SensitivityS),
104   member((unit, UnitSens), SensitivityS),
105   validUnit(UnitSens),
106   extractPrefixFactor(UnitSens, FactorSens),
107   LowerS =< UpperS,
108   member((lower, LowerM), ValuesM),
109   member((upper, UpperM), ValuesM),
110   member((unit, UnitM), ValuesM),
111   validUnit(UnitM),
112   extractPrefixFactor(UnitM, FactorM),
113   multiplyUnits([UnitM], UnitSens, UnitS),
114   LowerM * FactorM * ValueSens * FactorSens =:= LowerS * FactorS,
```

```
115    UpperM * FactorM * ValueSens * FactorSens =:= UpperS * FactorS.
```

Listing 2: Source code of `port.pl`

3.3 Edge Validity

Listing 3 shows the source code of file `edge.pl`, where the predicates to test for edge validity are defined.

```
1
2  validEdges(E, Pr, Errors) :-
3    findall(Error, (member(Edge, E), invalidEdge(Edge, Pr, Error), nonvar(Error
         )), Errors).
4
5  validEdge((Pf, Pt), Pr) :-
6    member((Pf, PropertiesF), Pr),
7    member((Pt, PropertiesT), Pr),
8    matchingProperties(PropertiesF, PropertiesT).
9
10 invalidEdge(E, Pr, Error) :-
11   not(validEdge(E, Pr)),
12   Error = E.
13
14 %**********************
15 %* SIGNAL_PROPERTIES **
16 %**********************
17
18 matchingProperties(PropertiesF, PropertiesT) :-
19   forall(member((NameF, ValueF), PropertiesF),
20     (
21       member(PropT, PropertiesT),
22       matchingProperty((NameF, ValueF), PropT)
23     )
24   ).
25
26 matchingProperty((NameF, ValueF), (NameT, ValueT)) :-
27   member(NameF, [signal_type, sampling_rate, resolution, custom_values, value
         ]),
28   NameT = NameF,
29   ValueF = ValueT.
30
31 matchingProperty((NameF, ValueF), (NameT, ValueT)) :-
32   member(NameF, [measurand, signal, sensitivity]),
33   NameT = NameF,
34   matchingProperties(ValueF, ValueT).
35
36 matchingProperty((unit, ValueF), (NameT, ValueT)) :-
37   NameT = unit,
38   extractPrefixFactor(ValueF, FactorF),
39   extractPrefixFactor(ValueT, FactorT),
40   FactorF = FactorT,
41   equalUnits(ValueF, ValueT).
42
43 matchingProperty((lower, ValueF), (NameT, ValueT)) :-
44   NameT = lower,
45   ValueF >= ValueT.
46
47 matchingProperty((upper, ValueF), (NameT, ValueT)) :-
48   NameT = upper,
```

```
49    ValueF =< ValueT.
```

Listing 3: Source code of `edge.pl`

3.4 Node Validity

Node validity predicates are defined in the file `node.pl`. The source code of this file is given in Listing 4.

```
1
2  validNodes(N, Pr, E, ErrorsList) :-
3    findall(Errors, (member(Node, N), invalidNode(Node, N, Pr, E, Errors),
         nonvar(Errors)), ErrorsList).
4
5  validNode(Node, N, Pr, E, InnerErrors) :-
6    member((Node, PropertiesN), Pr),
7    member((input_ports, Inputs), PropertiesN),
8    member((output_ports, Outputs), PropertiesN),
9    disjointPorts(Inputs, Outputs),
10   disjointNode(Node, Inputs, Outputs, N, Pr),
11   incomingConnections(Inputs, E),
12   incomingConnected(Inputs, E),
13   outgoingConnections(Outputs, E),
14   member((node_type, Type), PropertiesN),
15   matchingPorts(Node, Type, Inputs, Outputs, Pr, InnerErrors).
16
17 invalidNode(Node, N, Pr, E, Errors) :-
18   not(validNode(Node, N, Pr, E, _)),
19   Errors = [Node].
20
21 invalidNode(Node, N, Pr, E, Errors) :-
22   validNode(Node, N, Pr, E, InnerErrors),
23   length(InnerErrors, X),
24   X > 0,
25   Errors = InnerErrors.
26
27 %****************
28 %* NESTED_NODE **
29 %****************
30
31 matchingPorts(Node, nested_node, Inputs, Outputs, Pr, []) :-
32   member((Node, Properties), Pr),
33   member((graph, graph([], [], [], [])), Properties),
34   findall(Output, (
35       member(Output, Outputs),
36       member((Output, PropertiesO), Pr),
37       member((signal_type, Type), PropertiesO),
38       member(Type, [analogue_signal, converted_signal])
39     ),
40     SelectedOutputs
41   ),
42   forall(member(SelectedOutput, SelectedOutputs), derivableUnit(Inputs,
         SelectedOutput, Pr)).
43
44 matchingPorts(Node, nested_node, Inputs, Outputs, Pr, InnerErrors) :-
45   member((Node, Properties), Pr),
46   member((graph, graph(NestedN, NestedE, NestedPo, NestedPr)), Properties),
47   subset(Inputs, NestedPo),
48   forall(member(Input, Inputs), (
```

```
49        member((Input, ValuesI), Pr),
50        member((Input, ValuesNI), NestedPr),
51        ValuesI = ValuesNI
52      )
53    ),
54    outgoingConnections(Inputs, NestedE),
55    subset(Outputs, NestedPo),
56    forall(member(Output, Outputs), (
57        member((Output, ValuesO), Pr),
58        member((Output, ValuesNO), NestedPr),
59        ValuesO = ValuesNO
60      )
61    ),
62    incomingConnections(Outputs, NestedE),
63    validGraph(graph(NestedN, NestedE, NestedPo, NestedPr), InnerErrors).
64
65  %********************
66  %* COMPOSED_OUTPUT **
67  %********************
68
69  matchingPorts(_, composed_output, Inputs, [], Pr, []) :-
70    length(Inputs, X),
71    X >= 2,
72    forall(member(Input, Inputs), (
73        member((Input, Properties), Pr), (
74          member((signal_type, analogue_signal), Properties);
75          member((signal_type, converted_signal), Properties)
76        )
77      )
78    ).
79
80  %************
81  %* SENSOR **
82  %************
83
84  matchingPorts(_, sensor, [Input], [Output], Pr, []) :-
85    member((Input, PropertiesI), Pr),
86    member((signal_type, analogue_signal), PropertiesI),
87    member((measurand, _), PropertiesI),
88    length(PropertiesI, 2), % has signal_type and measurand
89    member((Output, PropertiesO), Pr),
90    member((signal_type, analogue_signal), PropertiesO),
91    subset(PropertiesI, PropertiesO).
92
93  %********************
94  %* AUDIO_RECORDER **
95  %********************
96
97  matchingPorts(_, audio_recorder, [Input], [Output], Pr, []) :-
98    member((Input, PropertiesI), Pr),
99    member((signal_type, audio_signal), PropertiesI),
100   member((Output, PropertiesO), Pr),
101   member((signal_type, audio_signal), PropertiesO),
102   PropertiesI = PropertiesO.
103
104 %********************
105 %* VIDEO_RECORDER **
106 %********************
107
108 matchingPorts(_, video_recorder, [Input], [Output], Pr, []) :-
109   member((Input, PropertiesI), Pr),
110   member((signal_type, video_signal), PropertiesI),
```

```
111   member((Output, PropertiesO), Pr),
112   member((signal_type, video_signal), PropertiesO),
113   PropertiesI = PropertiesO.
114
115 %*********************
116 %* MEASURING_MODULE **
117 %*********************
118
119 matchingPorts(Node, measuring_module, Inputs, Outputs, Pr, []) :-
120   forall(member(Input, Inputs), (
121       member((Input, Properties), Pr), (
122         member((signal_type, analogue_signal), Properties)
123       )
124     )
125   ),
126   forall(member(Output, Outputs), (
127       member((Output, Properties), Pr), (
128         member((signal_type, converted_signal), Properties)
129       )
130     )
131   ),
132   member((Node, Properties), Pr),
133   member((graph, graph([], [], [], [])), Properties),
134   forall(member(Output, Outputs), derivableUnit(Inputs, Output, Pr)).
135
136 matchingPorts(Node, measuring_module, Inputs, Outputs, Pr, InnerErrors) :-
137   forall(member(Input, Inputs), (
138       member((Input, Properties), Pr), (
139         member((signal_type, analogue_signal), Properties)
140       )
141     )
142   ),
143   forall(member(Output, Outputs), (
144       member((Output, Properties), Pr), (
145         member((signal_type, converted_signal), Properties)
146       )
147     )
148   ),
149   member((Node, Properties), Pr),
150   member((graph, graph(NestedN, NestedE, NestedPo, NestedPr)), Properties),
151   subset(Inputs, NestedPo),
152   forall(member(Input, Inputs), (
153       member((Input, ValuesI), Pr),
154       member((Input, ValuesNI), NestedPr),
155       ValuesI = ValuesNI
156     )
157   ),
158   outgoingConnections(Inputs, NestedE),
159   subset(Outputs, NestedPo),
160   forall(member(Output, Outputs), (
161       member((Output, ValuesO), Pr),
162       member((Output, ValuesNO), NestedPr),
163       ValuesO = ValuesNO
164     )
165   ),
166   incomingConnections(Outputs, NestedE),
167   validGraph(graph(NestedN, NestedE, NestedPo, NestedPr), InnerErrors).
168
169 %*******************
170 %* ANALYSIS_MODULE **
171 %*******************
172
```

```
173 matchingPorts(Node, analysis_module, Inputs, Outputs, Pr, []) :-
174   member((Node, Properties), Pr),
175   member((graph, graph([], [], [], [])), Properties),
176   findall(Output, (
177       member(Output, Outputs),
178       member((Output, PropertiesO), Pr),
179       member((signal_type, Type), PropertiesO),
180       member(Type, [analogue_signal, converted_signal])
181     ),
182     SelectedOutputs
183   ),
184   forall(member(SelectedOutput, SelectedOutputs), derivableUnit(Inputs,
        SelectedOutput, Pr)).
185
186 matchingPorts(Node, analysis_module, Inputs, Outputs, Pr, InnerErrors) :-
187   member((Node, Properties), Pr),
188   member((graph, graph(NestedN, NestedE, NestedPo, NestedPr)), Properties),
189   subset(Inputs, NestedPo),
190   forall(member(Input, Inputs), (
191       member((Input, ValuesI), Pr),
192       member((Input, ValuesNI), NestedPr),
193       ValuesI = ValuesNI
194     )
195   ),
196   outgoingConnections(Inputs, NestedE),
197   subset(Outputs, NestedPo),
198   forall(member(Output, Outputs), (
199       member((Output, ValuesO), Pr),
200       member((Output, ValuesNO), NestedPr),
201       ValuesO = ValuesNO
202     )
203   ),
204   incomingConnections(Outputs, NestedE),
205   validGraph(graph(NestedN, NestedE, NestedPo, NestedPr), InnerErrors).
206
207 %**************
208 %* AMPLIFIER **
209 %**************
210
211 matchingPorts(Node, amplifier, [Input], [Output], Pr, []) :-
212   member((Input, PropertiesI), Pr),
213   member((signal_type, TypeI), PropertiesI),
214   (TypeI = analogue_signal ; TypeI = converted_signal),
215   member((Output, PropertiesO), Pr),
216   member((signal_type, TypeO), PropertiesO),
217   (TypeO = analogue_signal ; TypeO = converted_signal),
218   TypeI = TypeO,
219   member((measurand, MeasurandValuesI), PropertiesI),
220   member((measurand, MeasurandValuesO), PropertiesO),
221   MeasurandValuesI = MeasurandValuesO,
222   member((signal, ValuesI), PropertiesI),
223   member((lower, LowerI), ValuesI),
224   member((upper, UpperI), ValuesI),
225   member((sensitivity, SensI), ValuesI),
226   member((value, ValueSensI), SensI),
227   member((signal, ValuesO), PropertiesO),
228   member((lower, LowerO), ValuesO),
229   member((upper, UpperO), ValuesO),
230   member((sensitivity, SensO), ValuesO),
231   member((value, ValueSensO), SensO),
232   member((Node, PropertiesN), Pr),
233   member((scale_factor, ScaleFactor), PropertiesN),
```

```
234   LowerI * ScaleFactor =:= LowerO,
235   UpperI * ScaleFactor =:= UpperO,
236   ValueSensI * ScaleFactor =:= ValueSensO,
237   subtract(ValuesI, [(lower, _), (upper, _), (sensitivity, _)],
          ReducedValuesI),
238   subtract(ValuesO, [(lower, _), (upper, _), (sensitivity, _)],
          ReducedValuesO),
239   ReducedValuesI = ReducedValuesO.
240
241 %*******************************
242 %* ANALOGUE_DIGITAL_CONVERTER **
243 %*******************************
244
245 matchingPorts(Node, analogue_digital_converter, [Input], [Output], Pr, []) :-
246   member((Input, PropertiesI), Pr),
247   member((signal_type, analogue_signal), PropertiesI),
248   member((Output, PropertiesO), Pr),
249   member((signal_type, converted_signal), PropertiesO),
250   member((signal, ValuesI), PropertiesI),
251   member((signal, ValuesO), PropertiesO),
252   subset(ValuesI, ValuesO),
253   member((resolution, ResolutionS), ValuesO),
254   member((sampling_rate, SamplingRateS), ValuesO),
255   member((Node, PropertiesN), Pr),
256   member((resolution, ResolutionN), PropertiesN),
257   member((sampling_rate, SamplingRateN), PropertiesN),
258   ResolutionN = ResolutionS,
259   SamplingRateN = SamplingRateS.
260
261 %************
262 %* FILTER **
263 %************
264
265 matchingPorts(Node, filter, [Input], [Output], Pr, []) :-
266   member((Input, PropertiesI), Pr),
267   member((signal_type, TypeI), PropertiesI),
268   (TypeI = analogue_signal ; TypeI = converted_signal),
269   member((Output, PropertiesO), Pr),
270   member((signal_type, TypeO), PropertiesO),
271   (TypeO = analogue_signal ; TypeO = converted_signal),
272   TypeI = TypeO,
273   member((measurand, MeasurandValuesI), PropertiesI),
274   member((measurand, MeasurandValuesO), PropertiesO),
275   member((unit, UnitI), MeasurandValuesI),
276   member((unit, UnitO), MeasurandValuesO),
277   equalUnits(UnitI, UnitO),
278   member((signal, ValuesI), PropertiesI),
279   member((lower, LowerI), ValuesI),
280   member((upper, UpperI), ValuesI),
281   member((signal, ValuesO), PropertiesO),
282   member((lower, LowerO), ValuesO),
283   member((upper, UpperO), ValuesO),
284   member((Node, PropertiesN), Pr),
285   member((range, ValuesN), PropertiesN),
286   member((lower, LowerN), ValuesN),
287   member((upper, UpperN), ValuesN),
288   ((LowerN >= LowerI, LowerO = LowerN); (LowerN =< LowerI, LowerO = LowerI)),
289   ((UpperN >= UpperI, UpperO = UpperI); (UpperN =< UpperI, UpperO = UpperN)),
290   subtract(ValuesI, [(lower, _), (upper, _)], ReducedValuesI),
291   subtract(ValuesO, [(lower, _), (upper, _)], ReducedValuesO),
292   ReducedValuesI = ReducedValuesO.
293
```

```
294 %************
295 %* TRIGGER **
296 %************
297
298 matchingPorts(Node, trigger, _, [Output], Pr, []) :-
299   member((Node, PropertiesN), Pr),
300   member((main_port, Port), PropertiesN),
301   member((Port, PropertiesP), Pr),
302   member((Output, PropertiesO), Pr),
303   PropertiesP = PropertiesO.
304
305 %**********************
306 %* HELPER_PREDICATES **
307 %**********************
308
309 disjointPorts(Inputs, Outputs) :-
310   intersection(Inputs, Outputs, []).
311
312 disjointNode(Node, Inputs, Outputs, N, Pr) :-
313   subtract(N, [Node], OtherNodes),
314   union(Inputs, Outputs, PortsN),
315   forall(member(Other, OtherNodes), (
316       member((Other, PropertiesO), Pr),
317       member((input_ports, InputsO), PropertiesO),
318       member((output_ports, OutputsO), PropertiesO),
319       union(InputsO, OutputsO, PortsO),
320       intersection(PortsN, PortsO, [])
321     )
322   ).
323
324 incomingConnections(Inputs, E) :-
325   forall(member(Input, Inputs), (
326       findall(_, member((Input, _), E), FromEdges),
327       length(FromEdges, 0)
328     )
329   ).
330
331 incomingConnected(Inputs, E) :-
332   forall(member(Input, Inputs), (
333       findall(_, member((_, Input), E), ConnectionsToInput),
334       length(ConnectionsToInput, 1)
335     )
336   ).
337
338 outgoingConnections(Outputs, E) :-
339   forall(member(Output, Outputs), (
340       findall(_, member((_, Output), E), ToEdges),
341       length(ToEdges, 0)
342     )
343   ).
344
345 derivableUnit(Inputs, Output, Pr) :-
346   member((Output, PropertiesO), Pr),
347   member((measurand, ValuesO), PropertiesO),
348   member((unit, UnitO), ValuesO),
349   member(Input, Inputs),
350   member((Input, PropertiesI), Pr),
351   member((measurand, ValuesI), PropertiesI),
352   member((unit, UnitI), ValuesI),
353   equalUnits(UnitI, UnitO).
354
355 derivableUnit(Inputs, Output, Pr) :-
```

```
356     member((Output, PropertiesO), Pr),
357     member((measurand, ValuesO), PropertiesO),
358     member((unit, UnitO), ValuesO),
359     subset(SelectedInputs, Inputs),
360     findall(UnitSI, (
361         member(SelectedInput, SelectedInputs),
362         member((SelectedInput, PropertiesSI), Pr),
363         member((measurand, ValuesSI), PropertiesSI),
364         member((unit, UnitSI), ValuesSI)
365       ),
366       [HeadUnit|Tail]
367     ),
368     multiplyUnits(Tail, HeadUnit, UnitO).
```

Listing 4: Source code of `node.pl`

3.5 Property Validity

The file `property.pl` contains a single predicate to check the validity of a set of properties. The source code of this file is shown in Listing 5.

```
1
2   validProperties(Pr) :-
3     findall(X, member((X, [_]), Pr), Properties),
4     is_set(Properties).
```

Listing 5: Source code of `property.pl`

3.6 Helper Predicates for Calculations with Units

A lot of predicates defined in previous mentioned files make use of predicates regarding units. These have been defined in a separate file called `unit.pl`. Listing ?? shows the source code of this file.

```
1
2   validUnit((Numerators, Denominators)) :-
3     forall(member((PrefixN, UnitN), Numerators), (
4         siPrefix(PrefixN),
5         (
6           siBaseUnit(UnitN);
7           UnitN = 'none'
8         )
9       )
10    ),
11    forall(member((PrefixD, UnitD), Denominators), (
12        siPrefix(PrefixD),
13        (
14          siBaseUnit(UnitD);
15          UnitD = 'none'
16        )
17      )
18    ).
19
20  equalUnits((Numerators1, Denominators1), (Numerators2, Denominators2)) :-
21    equalSubUnits(Numerators1, Numerators2),
```

```
22    equalSubUnits(Denominators1, Denominators2).
23
24  multiplyUnits([(Numerators, Denominators)|T], (Numerators2, Denominators2),
        Unit) :-
25    append(Numerators, Numerators2, CombinedNumerators),
26    append(Denominators, Denominators2, CombinedDenominators),
27    reduceFractionUnits(CombinedNumerators, CombinedDenominators, [], UnitR),
28    multiplyUnits(T, UnitR, Unit).
29
30  multiplyUnits([(Numerators1, Denominators1)], (Numerators2, Denominators2),
        Unit) :-
31    append(Numerators1, Numerators2, CombinedNumerators),
32    append(Denominators1, Denominators2, CombinedDenominators),
33    reduceFractionUnits(CombinedNumerators, CombinedDenominators, [], UnitR),
34    equalUnits(UnitR, Unit).
35
36  extractPrefixFactor((Numerators, Denominators), Factor) :-
37    findall(NumPrefix, member((NumPrefix, _), Numerators), NumPrefixes),
38    findall(DenPrefix, member((DenPrefix, _), Denominators), DenPrefixes),
39    multiplyFactors(NumPrefixes, 1, NumFactor),
40    multiplyFactors(DenPrefixes, 1, DenFactor),
41    Factor is NumFactor / DenFactor.
42
43  %**********************
44  %* HELPER_PREDICATES **
45  %**********************
46
47  siBaseUnit(X) :-
48    member(X, ['m', 'kg', 's', 'A', 'K', 'mol', 'cd']).
49
50  siPrefix(X) :-
51    member(X, ['yotta', 'zetta', 'exa', 'peta', 'tera', 'giga', 'mega', 'kilo',
          'hecto', 'deka', 'none',
52        'deci', 'centi', 'milli', 'micro', 'nano', 'pico', 'femto', 'atto', '
              zepto', 'yocto']).
53
54  siPrefixFactor(Prefix, Factor) :-
55    (Prefix = 'yotta' , Factor = 10**24);
56    (Prefix = 'zetta' , Factor = 10**21);
57    (Prefix = 'exa' , Factor = 10**18);
58    (Prefix = 'peta' , Factor = 10**15);
59    (Prefix = 'tera' , Factor = 10**12);
60    (Prefix = 'giga' , Factor = 10**9);
61    (Prefix = 'mega' , Factor = 10**6);
62    (Prefix = 'kilo' , Factor = 10**3);
63    (Prefix = 'hecto' , Factor = 10**2);
64    (Prefix = 'deka' , Factor = 10**1);
65    (Prefix = 'none' , Factor = 1);
66    (Prefix = 'deci' , Factor = 10**(-1));
67    (Prefix = 'centi' , Factor = 10**(-2));
68    (Prefix = 'milli' , Factor = 10**(-3));
69    (Prefix = 'micro' , Factor = 10**(-6));
70    (Prefix = 'nano' , Factor = 10**(-9));
71    (Prefix = 'pico' , Factor = 10**(-12));
72    (Prefix = 'femto' , Factor = 10**(-15));
73    (Prefix = 'atto' , Factor = 10**(-18));
74    (Prefix = 'zepto' , Factor = 10**(-21));
75    (Prefix = 'yocto' , Factor = 10**(-24)).
76
77  reduceFractionUnits([], CombinedDenominators, AccNominators, UnitR) :-
78    UnitR = (AccNominators, CombinedDenominators).
79
```

```
80  reduceFractionUnits([(CNumPrefix, CNumBaseUnit)|T], CombinedDenominators,
        AccNominators, UnitR) :-
81    CNumBaseUnit = 'none',
82    append([(CNumPrefix, CNumBaseUnit)], AccNominators, NewAccNominators),
83    reduceFractionUnits(T, CombinedDenominators, NewAccNominators, UnitR).
84
85  reduceFractionUnits([(CNumPrefix, CNumBaseUnit)|T], CombinedDenominators,
        AccNominators, UnitR) :-
86    siBaseUnit(CNumBaseUnit),
87    not(member((_, CNumBaseUnit), CombinedDenominators)),
88    append([(CNumPrefix, CNumBaseUnit)], AccNominators, NewAccNominators),
89    reduceFractionUnits(T, CombinedDenominators, NewAccNominators, UnitR).
90
91  reduceFractionUnits([(CNumPrefix, CNumBaseUnit)|T], CombinedDenominators,
        AccNominators, UnitR) :-
92    siBaseUnit(CNumBaseUnit),
93    member((CDenPrefix, CNumBaseUnit), CombinedDenominators),
94    append([(CNumPrefix, 'none')], AccNominators, NewAccNominators),
95    deleteFirst((CDenPrefix, CNumBaseUnit), CombinedDenominators,
          InterimCombinedDenominators),
96    append([(CDenPrefix, 'none')], InterimCombinedDenominators,
          NewCombinedDenominators),
97    reduceFractionUnits(T, NewCombinedDenominators, NewAccNominators, UnitR).
98
99  equalSubUnits([], SubUnits2) :-
100   forall(member((_, SubUnit2), SubUnits2), (
101       SubUnit2 = 'none'
102     )
103   ).
104
105 equalSubUnits([(_, SubUnit1)|T], SubUnits2) :-
106   SubUnit1 = 'none',
107   equalSubUnits(T, SubUnits2).
108
109 equalSubUnits([(_, SubUnit1)|T], SubUnits2) :-
110   siBaseUnit(SubUnit1),
111   member((SubPrefix2, SubUnit1), SubUnits2),
112   deleteFirst((SubPrefix2, SubUnit1), SubUnits2, NewSubUnits2),
113   equalSubUnits(T, NewSubUnits2).
114
115 multiplyFactors([NumPrefix], Acc, NumFactor) :-
116   siPrefixFactor(NumPrefix, X),
117   NumFactor is X * Acc.
118
119 multiplyFactors([NumPrefix|T], Acc, NumFactor) :-
120   siPrefixFactor(NumPrefix, X),
121   NewAcc is  X * Acc,
122   multiplyFactors(T, NewAcc, NumFactor).
123
124 deleteFirst(_, [], []).
125
126 deleteFirst(Term, [Term|Tail], Tail).
127
128 deleteFirst(Term, [Head|Tail], [Head|Result]) :-
129   deleteFirst(Term, Tail, Result).
```

Listing 6: Source code of `unit.pl`

3.7 Test Files

Because the test files contain a large amount of code and are somewhat repetitive, listing all the test files was neglected. To give an insight of how the test look like, the source code of the top level file `tests.pl` and a single example of a unit test are given in Listings 7 and 8.

```
1
2  :- [graph].
3  :- [graph_tests].
4  :- [port_tests].
5  :- [edge_tests].
6  :- [node_tests].
7  :- [property_tests].
8  :- [unit_tests].
9
10 runValidityTests :-
11   run_tests.
12
13 runGraphValidityTests :-
14   run_tests(graph).
15
16 runEdgeValidityTests :-
17   run_tests(edge).
18
19 runPortValidityTests :-
20   run_tests(port).
21
22 runNodeValidityTests :-
23   run_tests(node).
24
25 runPropertyValidityTests :-
26   run_tests(property).
27
28 runUnitValidityTests :-
29   run_tests(unit).
```

Listing 7: Source code of `tests.pl`

```
1
2  test(validAudioSignalTypeEdge, [nondet]) :-
3    validEdge(
4      (p1,p2),
5      [
6        (p1,[(signal_type,audio_signal)]),
7        (p2,[(signal_type,audio_signal)])
8      ]
9    ).
```

Listing 8: Source code of a single unit test

References

[1] M. Meili. GitHub Repository PA1_Protoype. `https://github.com/pipeaesac/PA1_Prototype`. Accessed: 2017-01-11.

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