

Why semantics matter:

a demonstration on knowledge-based control system design

Wim Pessemier
ICALEPCS 2015
Melbourne

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their relations

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

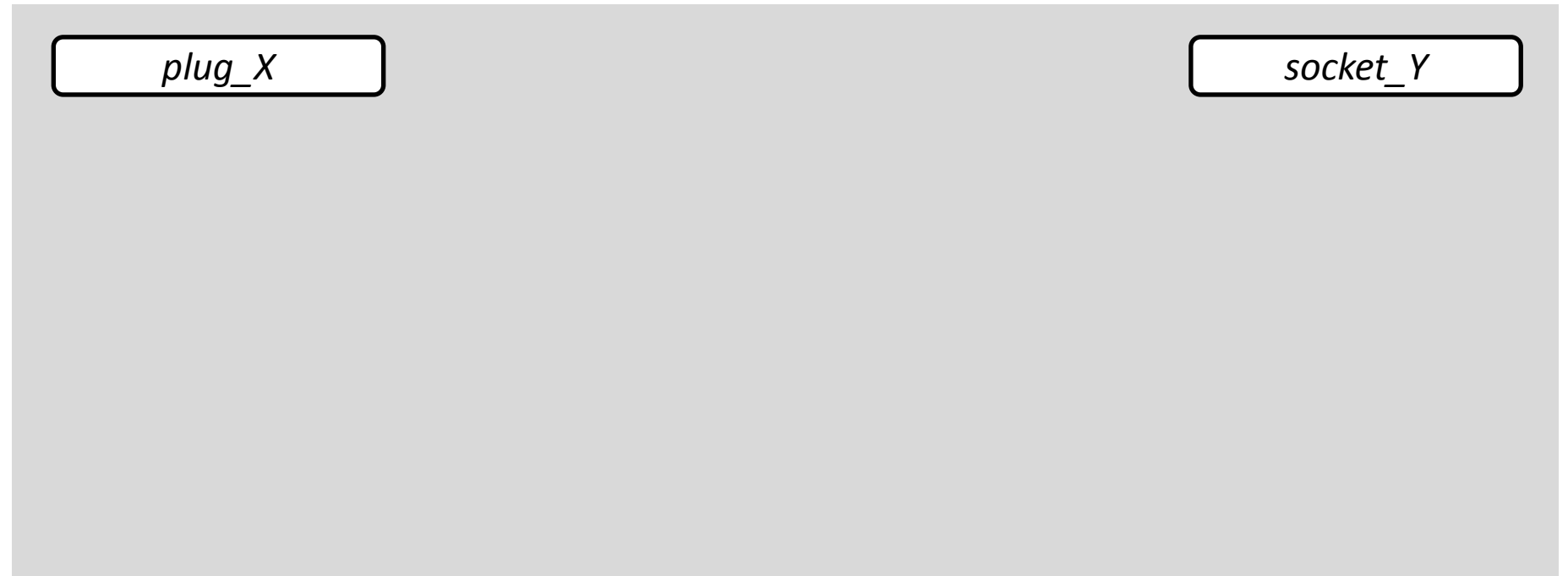
- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**

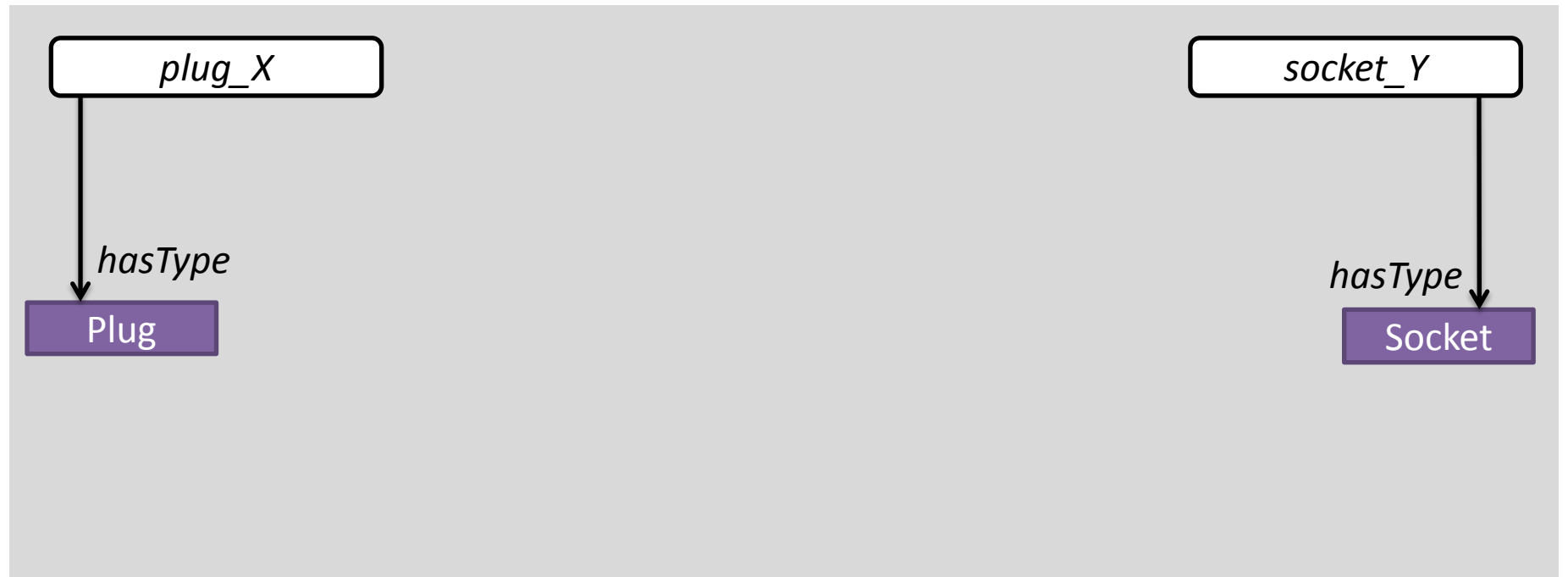


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

hasType

ELECTRIC

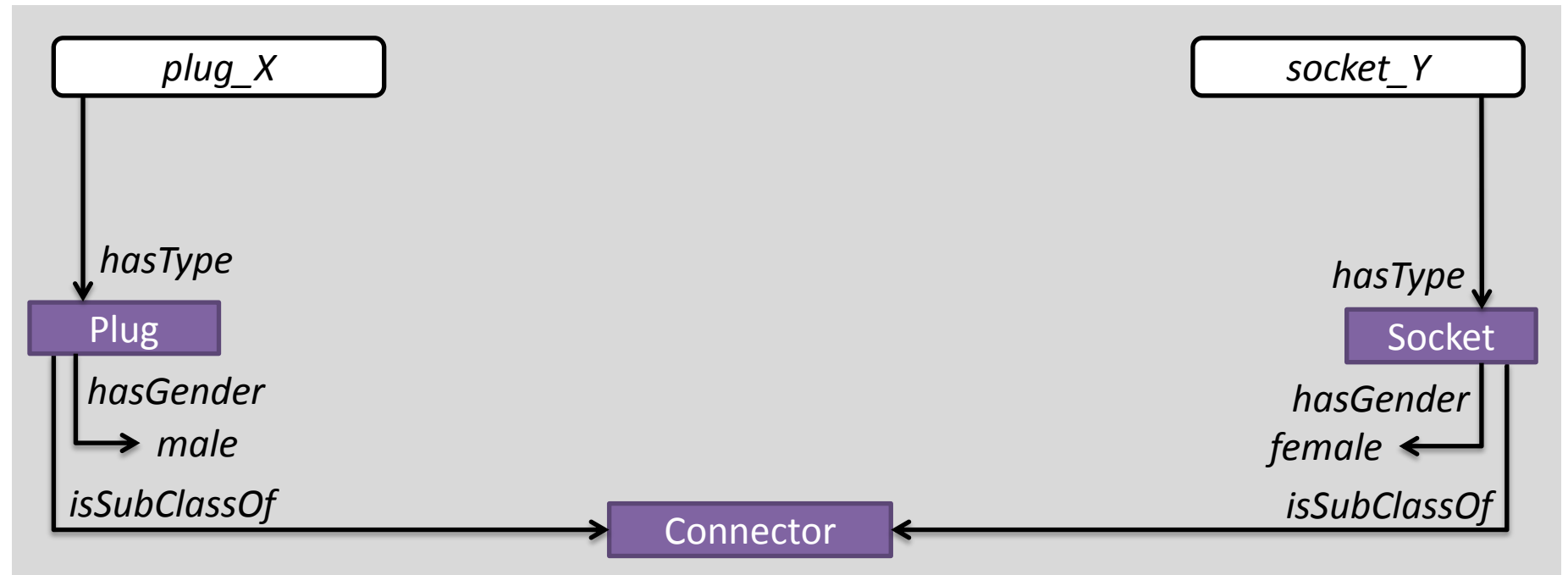
Plug
Socket

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

hasType

ELECTRIC

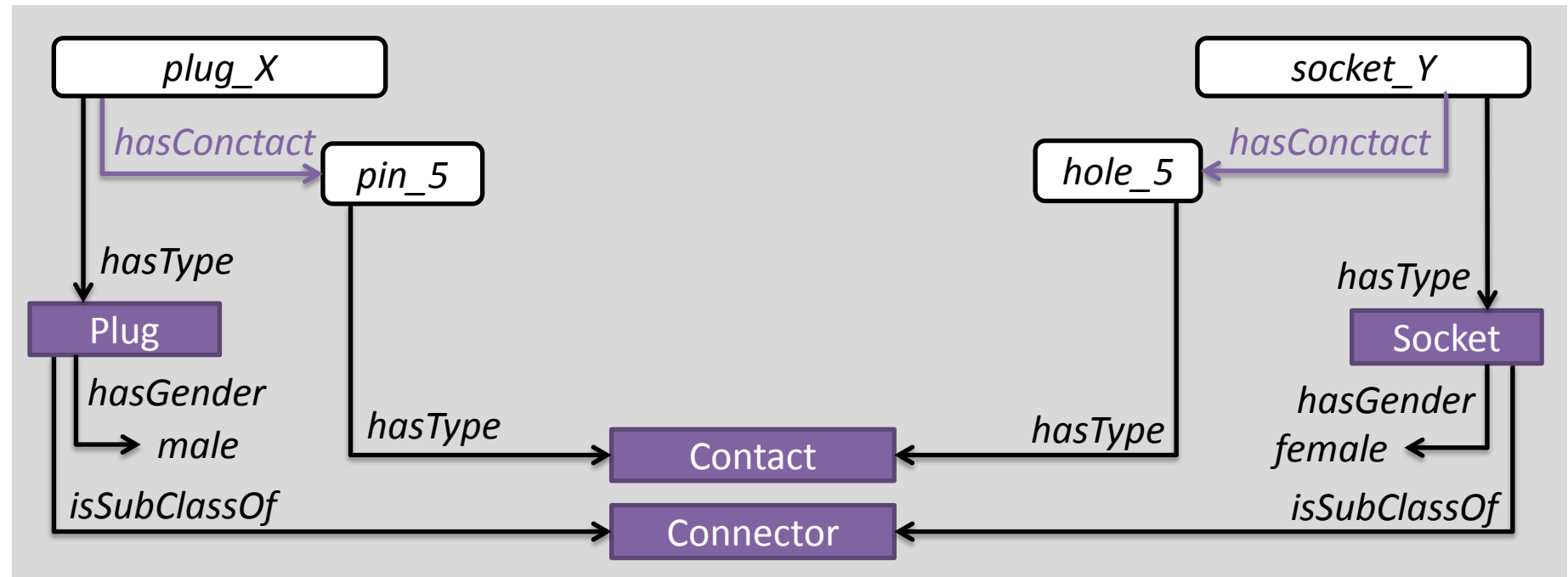
Plug
Socket

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

<code>hasType</code>	<code>isSubClassOf</code>
<code>hasGender</code>	

ELECTRIC

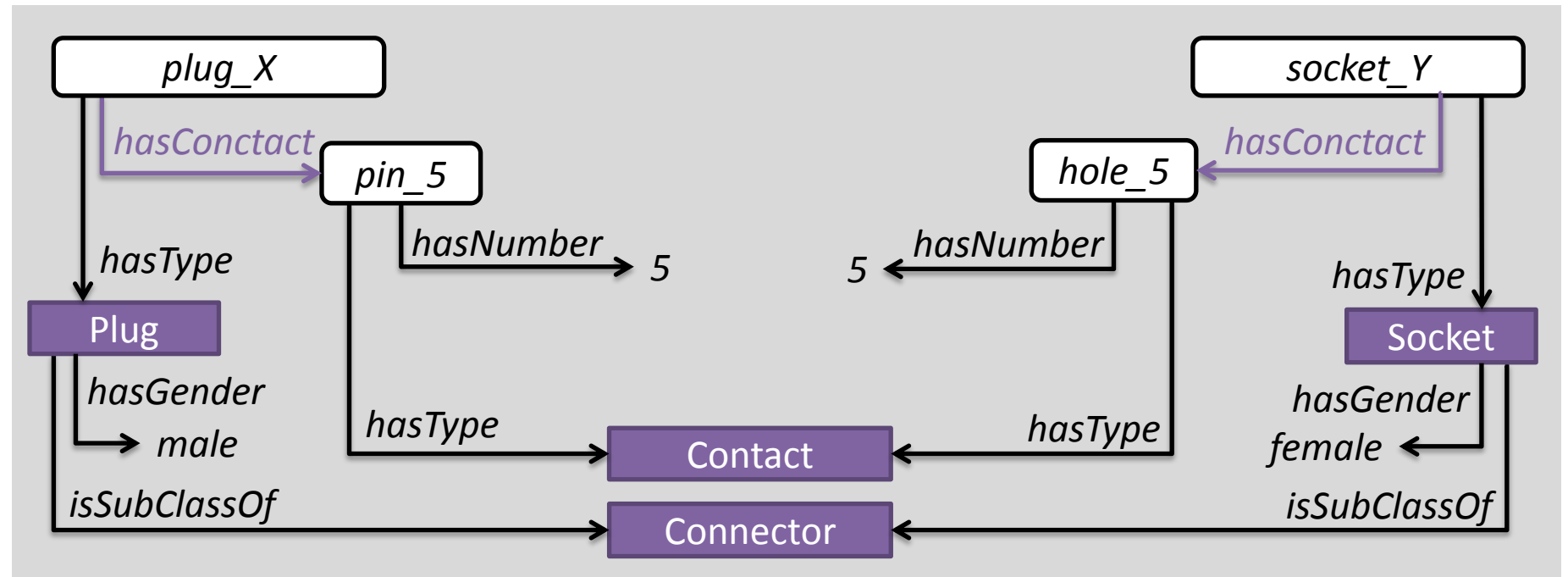
<code>Plug</code>	<code>Connector</code>
<code>Socket</code>	<code>hasContact</code>

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

hasType	isSubClassOf
hasGender	hasNumber

ELECTRIC

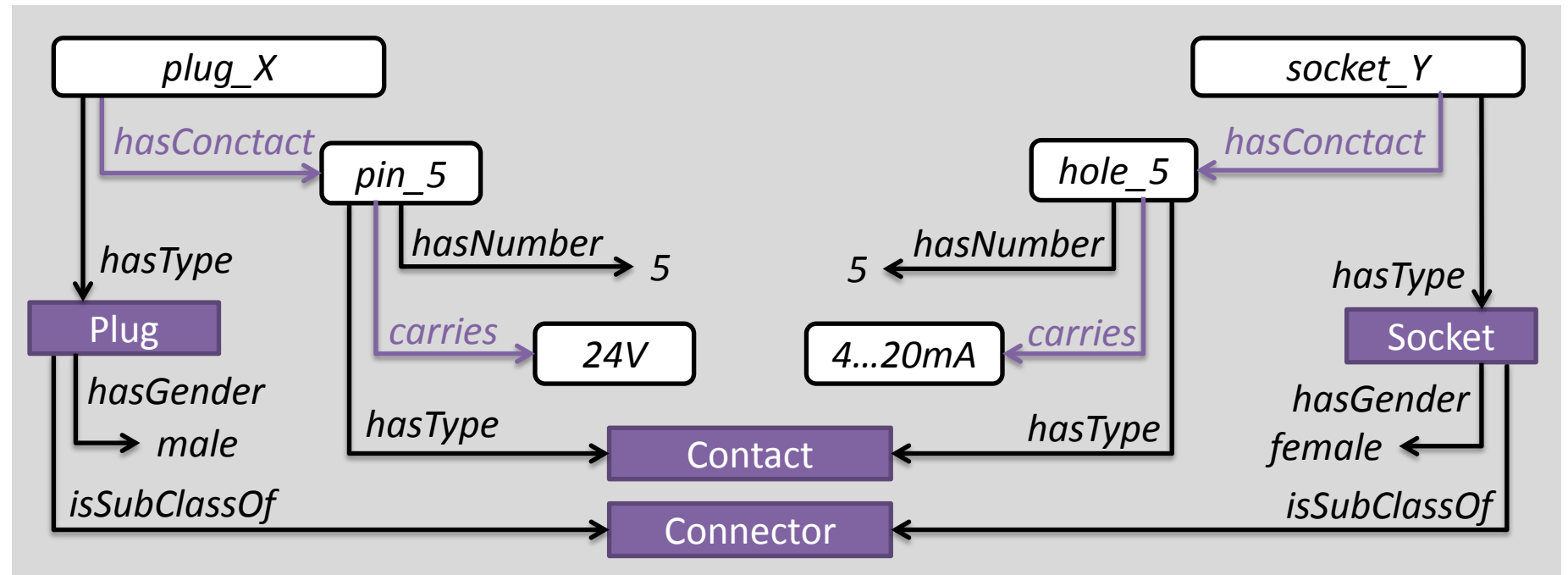
Plug	Connector	Contact
Socket	hasContact	

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

hasType	isSubClassOf
hasGender	hasNumber

ELECTRIC

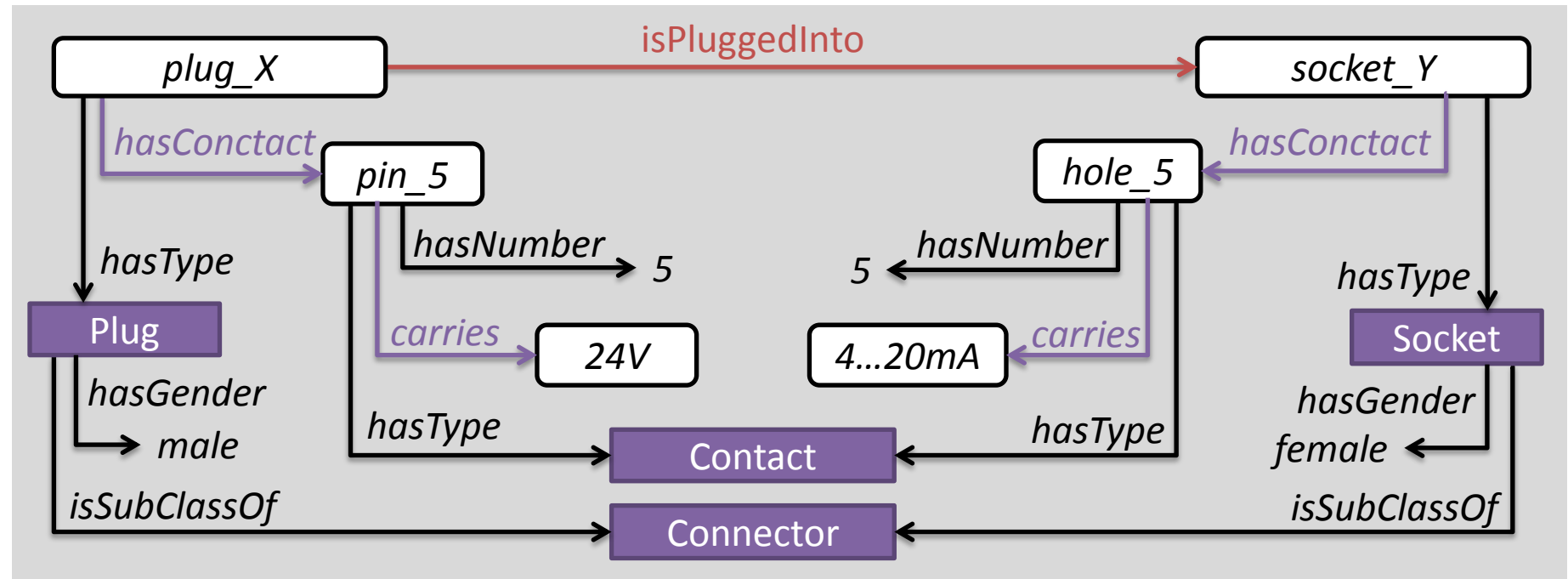
Plug	Connector	Contact
Socket	hasContact	carries

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

hasType *isSubClassOf*
hasGender *hasNumber*

ELECTRIC

Plug Connector Contact
Socket *hasContact* *carries*

MECHANIC

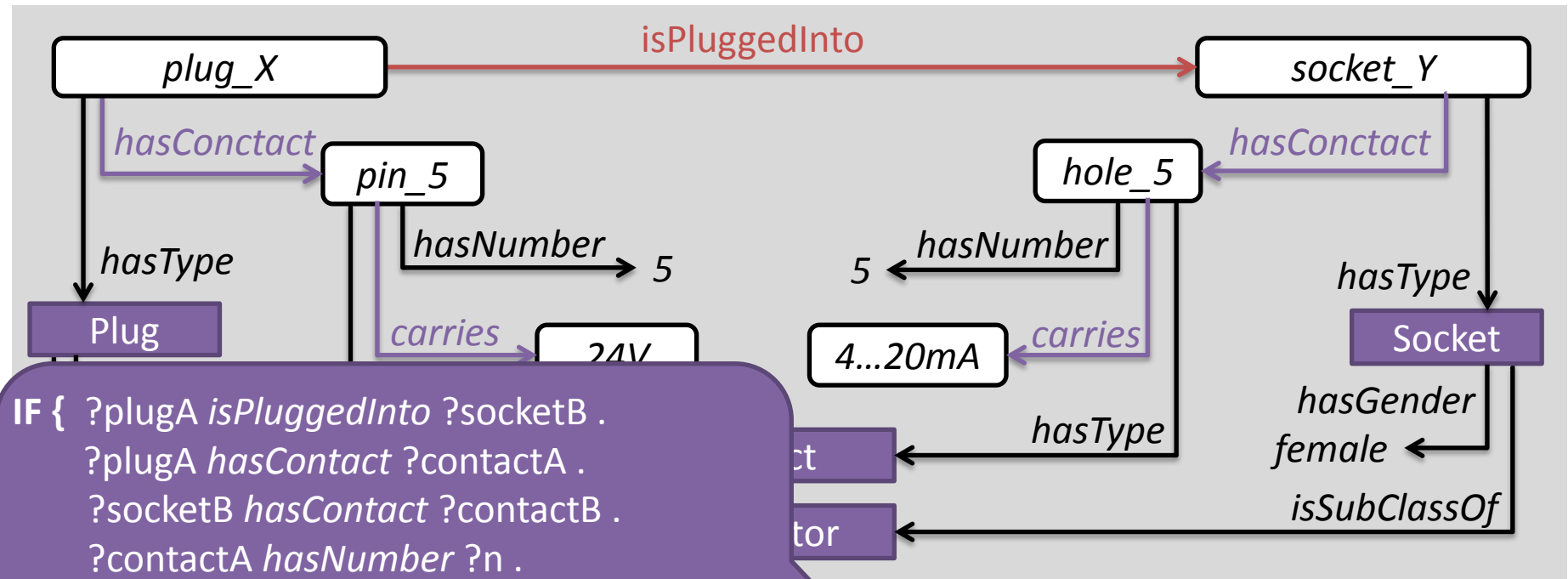
isPluggedInto

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



IF { ?plugA isPluggedInto ?socketB .
?plugA hasContact ?contactA .
?socketB hasContact ?contactB .
?contactA hasNumber ?n .
?contactB hasNumber ?n }
THEN { ?contactA isConnectedTo ?contactB }

hasType	isSubClassOf	Plug	Connector	Contact
hasGender	hasNumber	Socket	hasContact	carries

MECHANIC

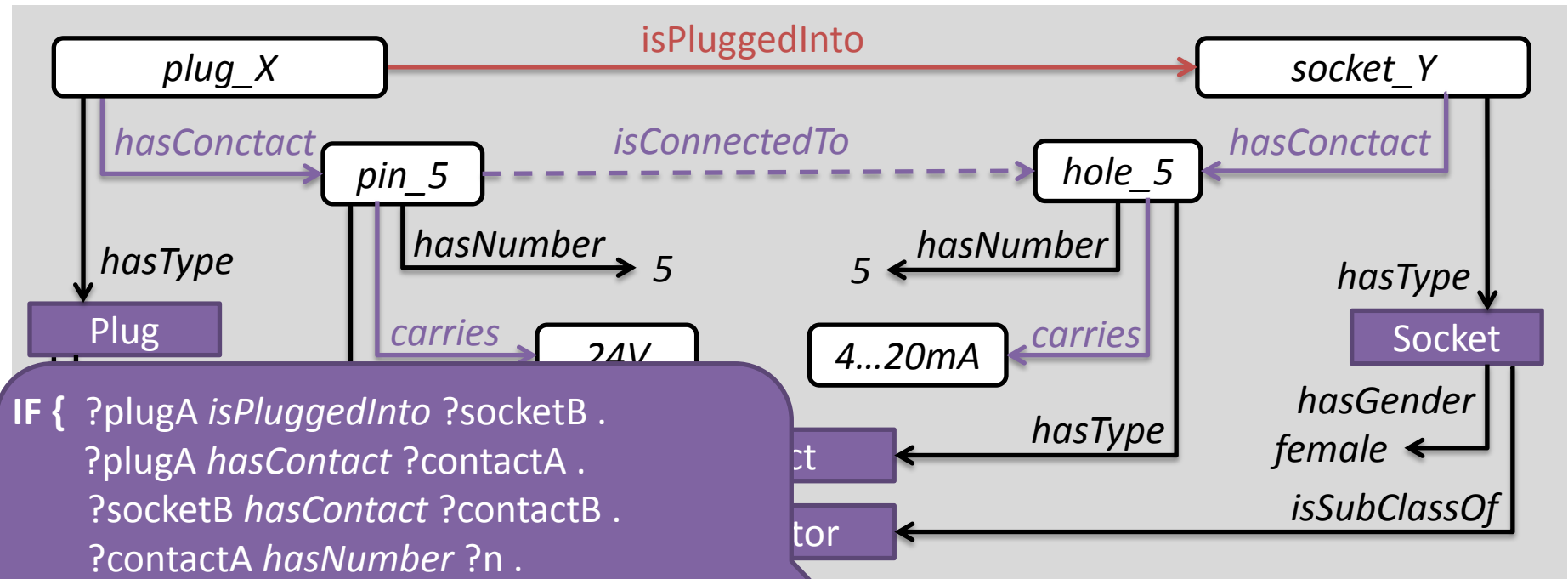
isPluggedInto

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



IF { ?plugA isPluggedInto ?socketB .
?plugA hasContact ?contactA .
?socketB hasContact ?contactB .
?contactA hasNumber ?n .
?contactB hasNumber ?n }
THEN { ?contactA isConnectedTo ?contactB }

hasType	isSubClassOf	Plug	Connector	Contact
hasGender	hasNumber	Socket	hasContact	carries

MECHANIC

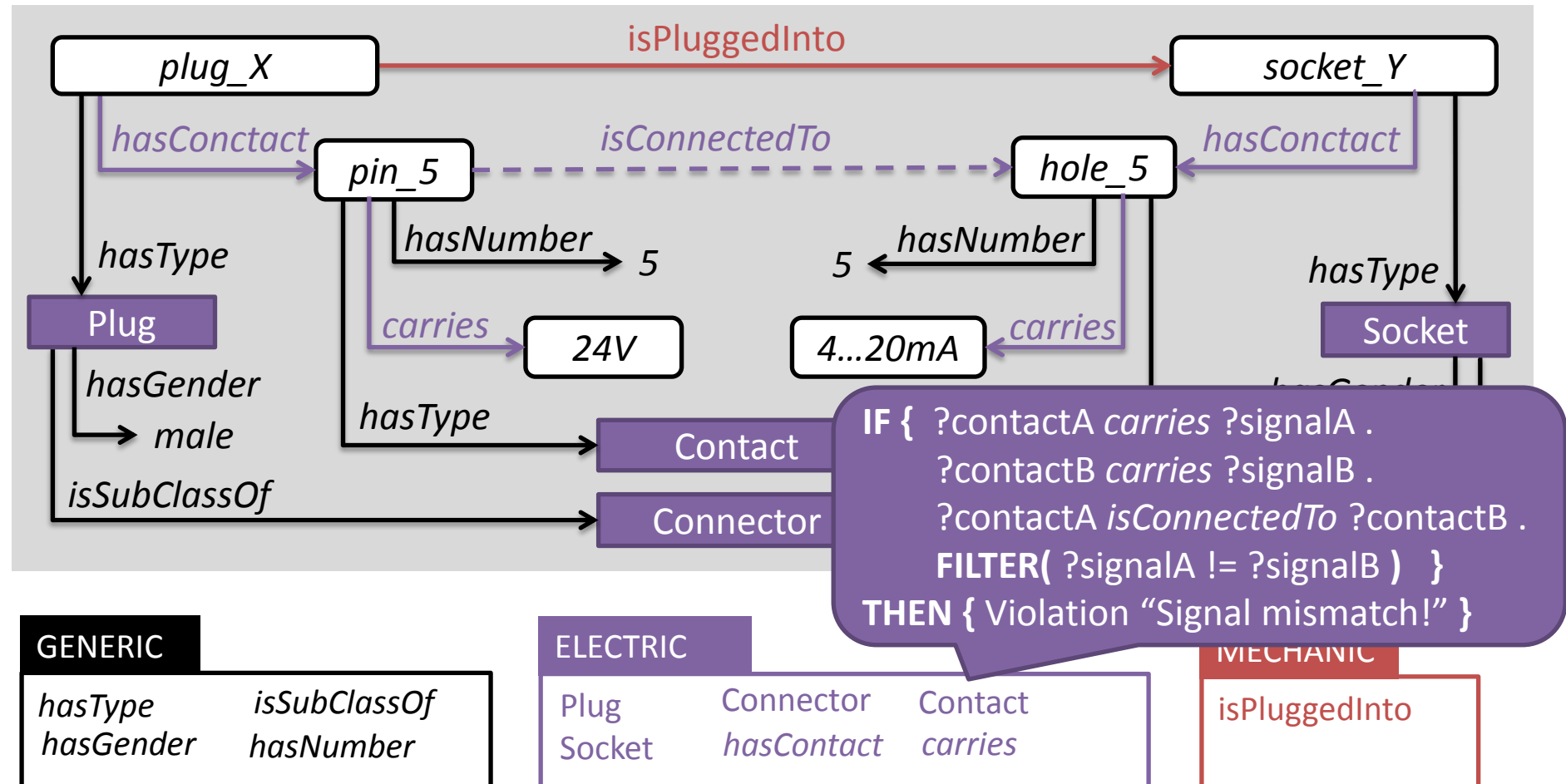
isPluggedInto

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**

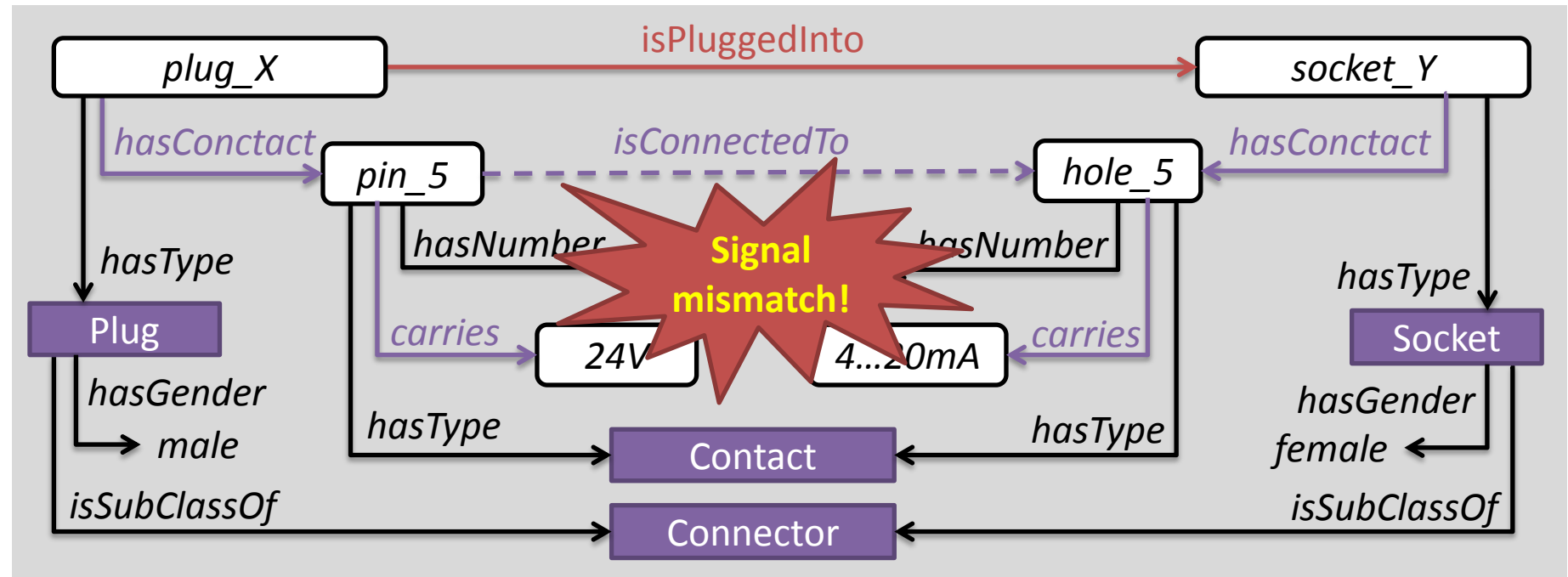


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

What are semantic models?

- Models that describe
 - pieces of information (data, descriptions)
 - their **relations** → **meaning (semantics)**



GENERIC

hasType *isSubClassOf*
hasGender *hasNumber*

ELECTRIC

Plug Connector Contact
Socket *hasContact* *carries*

MECHANIC

isPluggedInto

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Where to apply them?

Why semantics matter?

- What are semantic models?
- **Where to apply them?**
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Where to apply them?

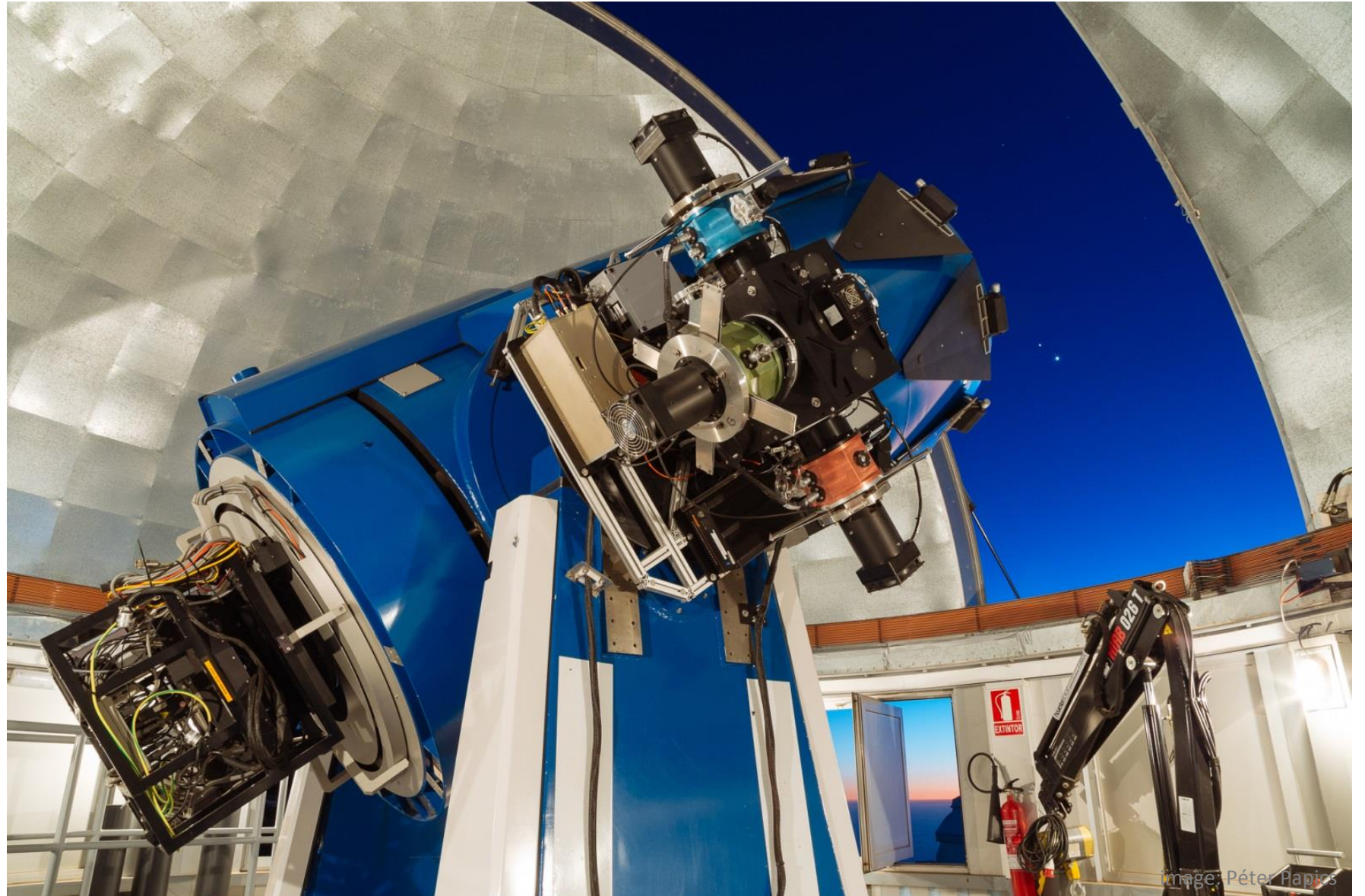
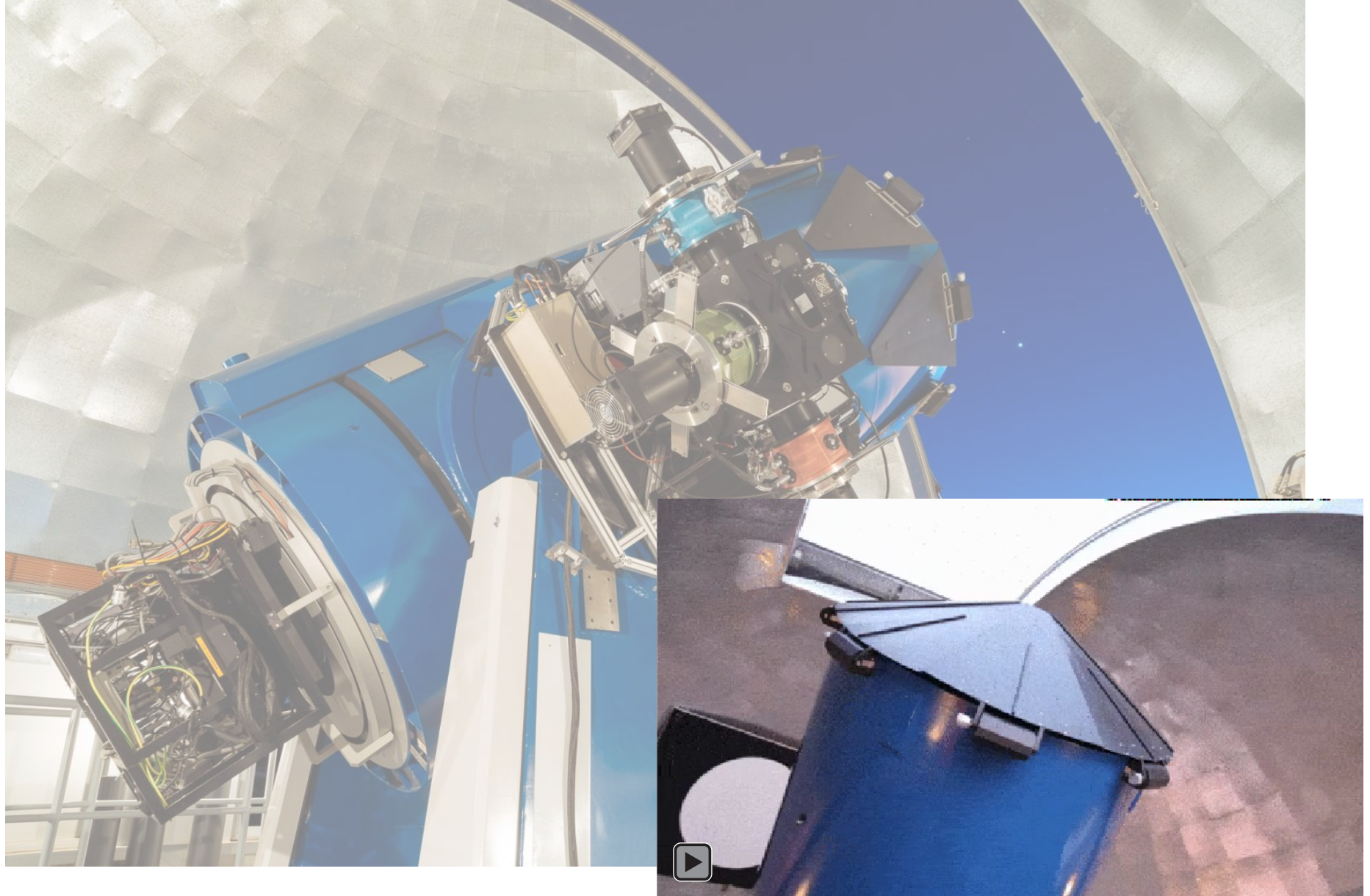


image: Péter Papics

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Where to apply them?



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

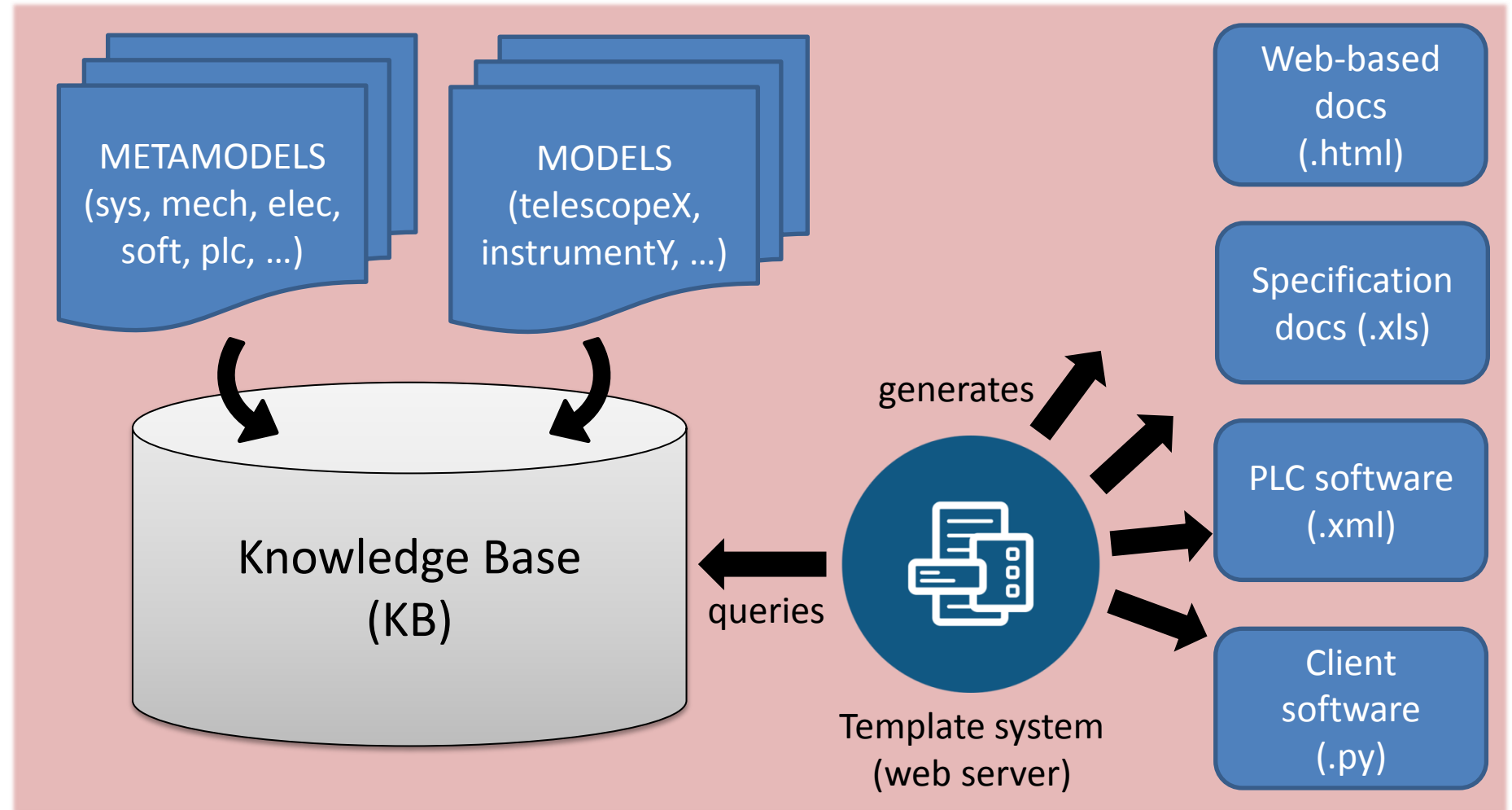
How to apply them?

Why semantics matter?

- What are semantic models?
- Where to apply them?
- **How to apply them?**
- How to build them?
- How to use them?
- Conclusions

How to apply them?

- Put them in a Knowledge Base and extract information!



OntoManager

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

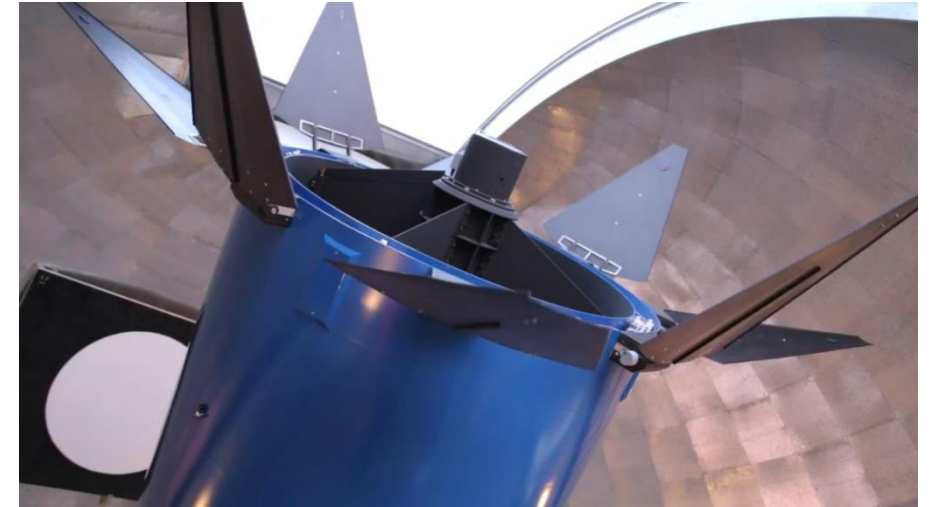
How to build them?

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Using an existing modeling language?
 - UML, SysML, ... : semantics not sufficiently formal
 - Modeling languages have no “programming” capabilities (loops, functions, if-then, ...)



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Using an existing modeling language?

- UML, SysML, ... : semantics not sufficiently formal
- Modeling languages have no “programming” capabilities (loops, functions, if-then, ...)



- Using a Domain Specific Language (DSL)?

- Internal DSL called Ontoscript
- Based on coffeescript (~javascript)
- Idea “adopted” from the Giant Magellan Telescope project [1]

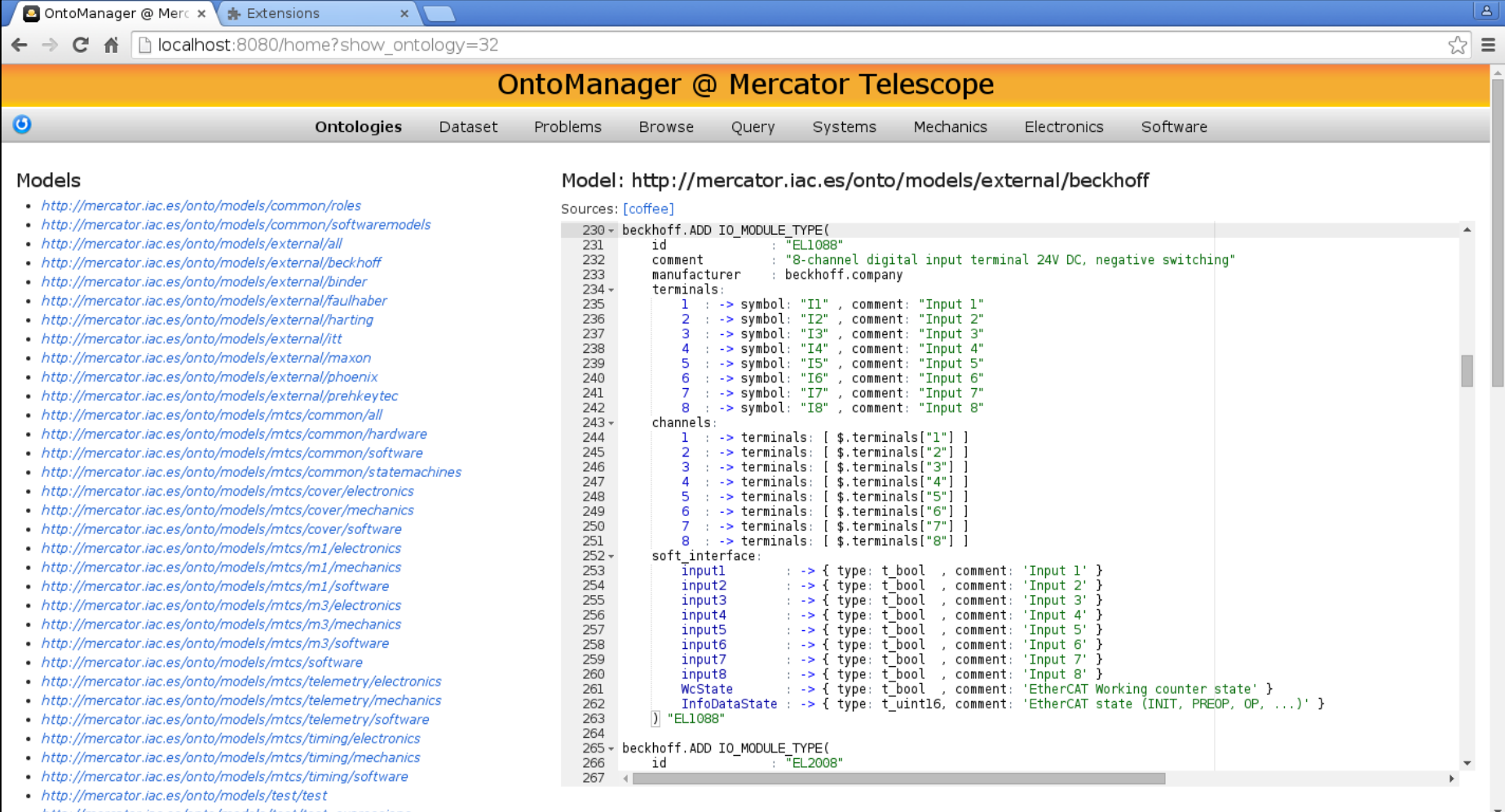
[1] J. M. Filgueira, “GMT software and controls overview”, Proc. SPIE 8451, Amsterdam, July 2012, 845111

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Example: model of an I/O module **type**



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=32`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology URLs, including `http://mercator.iac.es/onto/models/common/roles`, `http://mercator.iac.es/onto/models/common/softwaremodels`, `http://mercator.iac.es/onto/models/external/all`, `http://mercator.iac.es/onto/models/external/beckhoff`, `http://mercator.iac.es/onto/models/external/binder`, `http://mercator.iac.es/onto/models/external/faulhaber`, `http://mercator.iac.es/onto/models/external/harting`, `http://mercator.iac.es/onto/models/external/itt`, `http://mercator.iac.es/onto/models/external/maxon`, `http://mercator.iac.es/onto/models/external/phoenix`, `http://mercator.iac.es/onto/models/external/prehkeytec`, `http://mercator.iac.es/onto/models/mtcs/common/all`, `http://mercator.iac.es/onto/models/mtcs/common/hardware`, `http://mercator.iac.es/onto/models/mtcs/common/software`, `http://mercator.iac.es/onto/models/mtcs/common/statemachines`, `http://mercator.iac.es/onto/models/mtcs/cover/electronics`, `http://mercator.iac.es/onto/models/mtcs/cover/mechanics`, `http://mercator.iac.es/onto/models/mtcs/cover/software`, `http://mercator.iac.es/onto/models/mtcs/m1/electronics`, `http://mercator.iac.es/onto/models/mtcs/m1/mechanics`, `http://mercator.iac.es/onto/models/mtcs/m1/software`, `http://mercator.iac.es/onto/models/mtcs/m3/electronics`, `http://mercator.iac.es/onto/models/mtcs/m3/mechanics`, `http://mercator.iac.es/onto/models/mtcs/m3/software`, `http://mercator.iac.es/onto/models/mtcs/telemetry/electronics`, `http://mercator.iac.es/onto/models/mtcs/telemetry/mechanics`, `http://mercator.iac.es/onto/models/mtcs/telemetry/software`, `http://mercator.iac.es/onto/models/mtcs/timing/electronics`, `http://mercator.iac.es/onto/models/mtcs/timing/mechanics`, `http://mercator.iac.es/onto/models/mtcs/timing/software`, `http://mercator.iac.es/onto/models/test/test`, and `http://mercator.iac.es/onto/models/test/test_expressions`.

The main content area displays the "Model: `http://mercator.iac.es/onto/models/external/beckhoff`". The "Sources: [coffee]" section shows the following code:

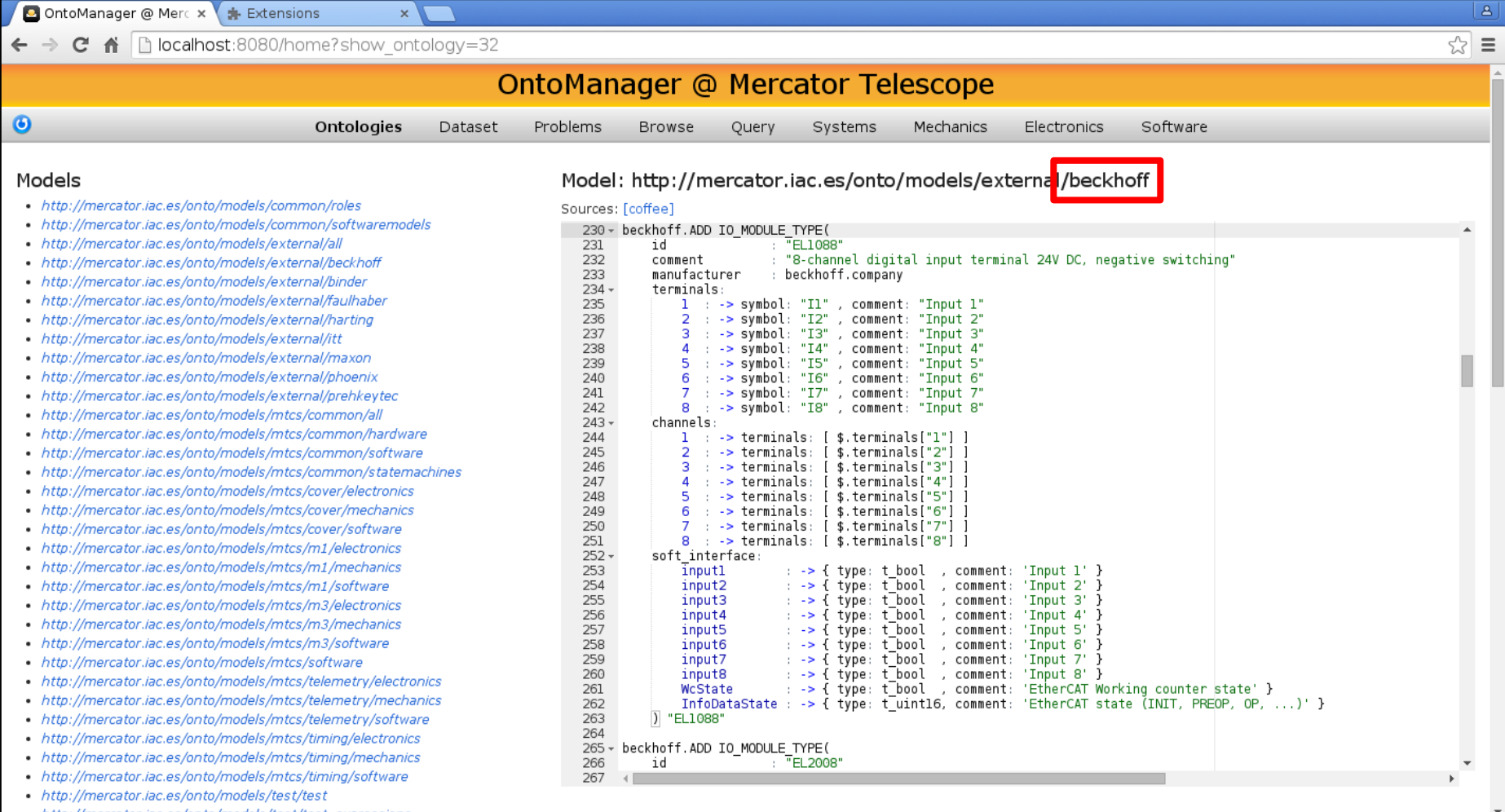
```
230 beckhoff.ADD_IO_MODULE_TYPE(  
231   id          : "EL1088"  
232   comment     : "8-channel digital input terminal 24V DC, negative switching"  
233   manufacturer : bechhoff.company  
234   terminals:  
235     1 : -> symbol: "I1" , comment: "Input 1"  
236     2 : -> symbol: "I2" , comment: "Input 2"  
237     3 : -> symbol: "I3" , comment: "Input 3"  
238     4 : -> symbol: "I4" , comment: "Input 4"  
239     5 : -> symbol: "I5" , comment: "Input 5"  
240     6 : -> symbol: "I6" , comment: "Input 6"  
241     7 : -> symbol: "I7" , comment: "Input 7"  
242     8 : -> symbol: "I8" , comment: "Input 8"  
243   channels:  
244     1 : -> terminals: [ $.terminals["1"] ]  
245     2 : -> terminals: [ $.terminals["2"] ]  
246     3 : -> terminals: [ $.terminals["3"] ]  
247     4 : -> terminals: [ $.terminals["4"] ]  
248     5 : -> terminals: [ $.terminals["5"] ]  
249     6 : -> terminals: [ $.terminals["6"] ]  
250     7 : -> terminals: [ $.terminals["7"] ]  
251     8 : -> terminals: [ $.terminals["8"] ]  
252   soft_interface:  
253     input1 : -> { type: t_bool , comment: 'Input 1' }  
254     input2 : -> { type: t_bool , comment: 'Input 2' }  
255     input3 : -> { type: t_bool , comment: 'Input 3' }  
256     input4 : -> { type: t_bool , comment: 'Input 4' }  
257     input5 : -> { type: t_bool , comment: 'Input 5' }  
258     input6 : -> { type: t_bool , comment: 'Input 6' }  
259     input7 : -> { type: t_bool , comment: 'Input 7' }  
260     input8 : -> { type: t_bool , comment: 'Input 8' }  
261     WcState : -> { type: t_bool , comment: 'EtherCAT Working counter state' }  
262     InfoDataState : -> { type: t_uint16, comment: 'EtherCAT state (INIT, PREOP, OP, ...)' }  
263   ) "EL1088"  
264  
265 beckhoff.ADD_IO_MODULE_TYPE(  
266   id          : "EL2008"  
267
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Example: model of an I/O module **type**



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=32`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology URLs, including `http://mercator.iac.es/onto/models/external/beckhoff`. The main content area displays the model for `http://mercator.iac.es/onto/models/external/beckhoff`, which is highlighted with a red box. The model is a CoffeeScript file defining an I/O module type.

```
Model: http://mercator.iac.es/onto/models/external/beckhoff
Sources: [coffee]
230 beckhoff.ADD IO_MODULE_TYPE(
231   id      : "EL1088"
232   comment : "8-channel digital input terminal 24V DC, negative switching"
233   manufacturer : beckhoff.company
234   terminals:
235     1 : -> symbol: "I1", comment: "Input 1"
236     2 : -> symbol: "I2", comment: "Input 2"
237     3 : -> symbol: "I3", comment: "Input 3"
238     4 : -> symbol: "I4", comment: "Input 4"
239     5 : -> symbol: "I5", comment: "Input 5"
240     6 : -> symbol: "I6", comment: "Input 6"
241     7 : -> symbol: "I7", comment: "Input 7"
242     8 : -> symbol: "I8", comment: "Input 8"
243   channels:
244     1 : -> terminals: [ $.terminals["1"] ]
245     2 : -> terminals: [ $.terminals["2"] ]
246     3 : -> terminals: [ $.terminals["3"] ]
247     4 : -> terminals: [ $.terminals["4"] ]
248     5 : -> terminals: [ $.terminals["5"] ]
249     6 : -> terminals: [ $.terminals["6"] ]
250     7 : -> terminals: [ $.terminals["7"] ]
251     8 : -> terminals: [ $.terminals["8"] ]
252   soft_interface:
253     input1 : -> { type: t_bool, comment: 'Input 1' }
254     input2 : -> { type: t_bool, comment: 'Input 2' }
255     input3 : -> { type: t_bool, comment: 'Input 3' }
256     input4 : -> { type: t_bool, comment: 'Input 4' }
257     input5 : -> { type: t_bool, comment: 'Input 5' }
258     input6 : -> { type: t_bool, comment: 'Input 6' }
259     input7 : -> { type: t_bool, comment: 'Input 7' }
260     input8 : -> { type: t_bool, comment: 'Input 8' }
261     WcState : -> { type: t_bool, comment: 'EtherCAT Working counter state' }
262     InfoDataState : -> { type: t_uint16, comment: 'EtherCAT state (INIT, PREOP, OP, ...)' }
263   ) "EL1088"
264
265 beckhoff.ADD IO_MODULE_TYPE(
266   id      : "EL2008"
267
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Example: model of an I/O module **type**

The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=32`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

On the left, under the "Models" section, a list of URIs is shown, including `http://mercator.iac.es/ontology/common/roles`, `http://mercator.iac.es/ontology/common/softwaremodels`, and `http://mercator.iac.es/ontology/external/all`. A small image of a Beckhoff I/O module is visible.

On the right, the "Model: `http://mercator.iac.es/ontology/models/external/beckhoff`" is displayed. The "Sources: [coffee]" are listed. The main content area shows the definition of the `beckhoff.ADD_IO_MODULE_TYPE` function, which is highlighted with a red box. The definition includes:

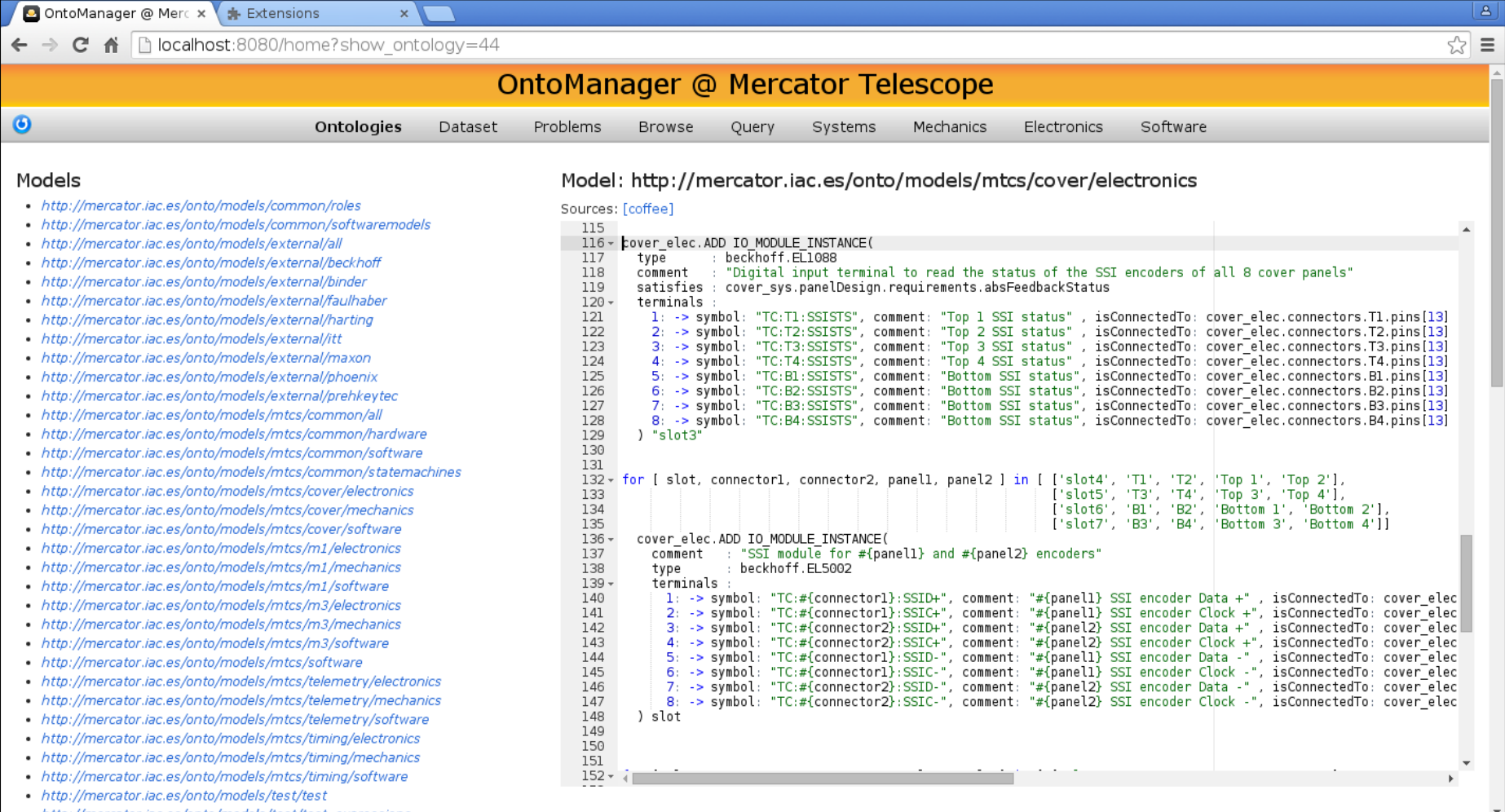
```
230 beckhoff.ADD_IO_MODULE_TYPE(  
231   id          : "EL1088"  
232   comment     : "8-channel digital input terminal 24V DC, negative switching"  
233   manufacturer : beckhoff.company  
234   terminals:  
235     1 : -> symbol: "I1" , comment: "Input 1"  
236     2 : -> symbol: "I2" , comment: "Input 2"  
237     3 : -> symbol: "I3" , comment: "Input 3"  
238     4 : -> symbol: "I4" , comment: "Input 4"  
239     5 : -> symbol: "I5" , comment: "Input 5"  
240     6 : -> symbol: "I6" , comment: "Input 6"  
241     7 : -> symbol: "I7" , comment: "Input 7"  
242     8 : -> symbol: "I8" , comment: "Input 8"  
243   channels:  
244     1 : -> terminals: [ $.terminals["1"] ]  
245     2 : -> terminals: [ $.terminals["2"] ]  
246     3 : -> terminals: [ $.terminals["3"] ]  
247     4 : -> terminals: [ $.terminals["4"] ]  
248     5 : -> terminals: [ $.terminals["5"] ]  
249     6 : -> terminals: [ $.terminals["6"] ]  
250     7 : -> terminals: [ $.terminals["7"] ]  
251     8 : -> terminals: [ $.terminals["8"] ]  
252   soft_interface:  
253     input1 : -> { type: t_bool , comment: 'Input 1' }  
254     input2 : -> { type: t_bool , comment: 'Input 2' }  
255     input3 : -> { type: t_bool , comment: 'Input 3' }  
256     input4 : -> { type: t_bool , comment: 'Input 4' }  
257     input5 : -> { type: t_bool , comment: 'Input 5' }  
258     input6 : -> { type: t_bool , comment: 'Input 6' }  
259     input7 : -> { type: t_bool , comment: 'Input 7' }  
260     input8 : -> { type: t_bool , comment: 'Input 8' }  
261     WcState : -> { type: t_bool , comment: 'EtherCAT Working counter state' }  
262     InfoDataState : -> { type: t_uint16, comment: 'EtherCAT state (INIT, PREOP, OP, ...)' }  
263   ) "EL1088"  
264  
265 + beckhoff.ADD_IO_MODULE_TYPE(  
266   id          : "EL2008"  
267
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Example: model of an I/O module instance



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=44`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

On the left, under the "Models" section, a list of ontology URLs is provided, including:

- <http://mercator.iac.es/onto/models/common/roles>
- <http://mercator.iac.es/onto/models/common/softwaremodels>
- <http://mercator.iac.es/onto/models/external/all>
- <http://mercator.iac.es/onto/models/external/beckhoff>
- <http://mercator.iac.es/onto/models/external/binder>
- <http://mercator.iac.es/onto/models/external/faulhaber>
- <http://mercator.iac.es/onto/models/external/harting>
- <http://mercator.iac.es/onto/models/external/itt>
- <http://mercator.iac.es/onto/models/external/maxon>
- <http://mercator.iac.es/onto/models/external/phoenix>
- <http://mercator.iac.es/onto/models/external/prehkeytec>
- <http://mercator.iac.es/onto/models/mtcs/common/all>
- <http://mercator.iac.es/onto/models/mtcs/common/hardware>
- <http://mercator.iac.es/onto/models/mtcs/common/software>
- <http://mercator.iac.es/onto/models/mtcs/common/statemachines>
- <http://mercator.iac.es/onto/models/mtcs/cover/electronics>
- <http://mercator.iac.es/onto/models/mtcs/cover/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/cover/software>
- <http://mercator.iac.es/onto/models/mtcs/m1/electronics>
- <http://mercator.iac.es/onto/models/mtcs/m1/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/m1/software>
- <http://mercator.iac.es/onto/models/mtcs/m3/electronics>
- <http://mercator.iac.es/onto/models/mtcs/m3/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/m3/software>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/electronics>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/software>
- <http://mercator.iac.es/onto/models/mtcs/timing/electronics>
- <http://mercator.iac.es/onto/models/mtcs/timing/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/timing/software>
- <http://mercator.iac.es/onto/models/test/test>
- http://mercator.iac.es/onto/models/test/test_expressions

The main content area displays the "Model: <http://mercator.iac.es/onto/models/mtcs/cover/electronics>". The source is listed as `[coffee]`. The model content is as follows:

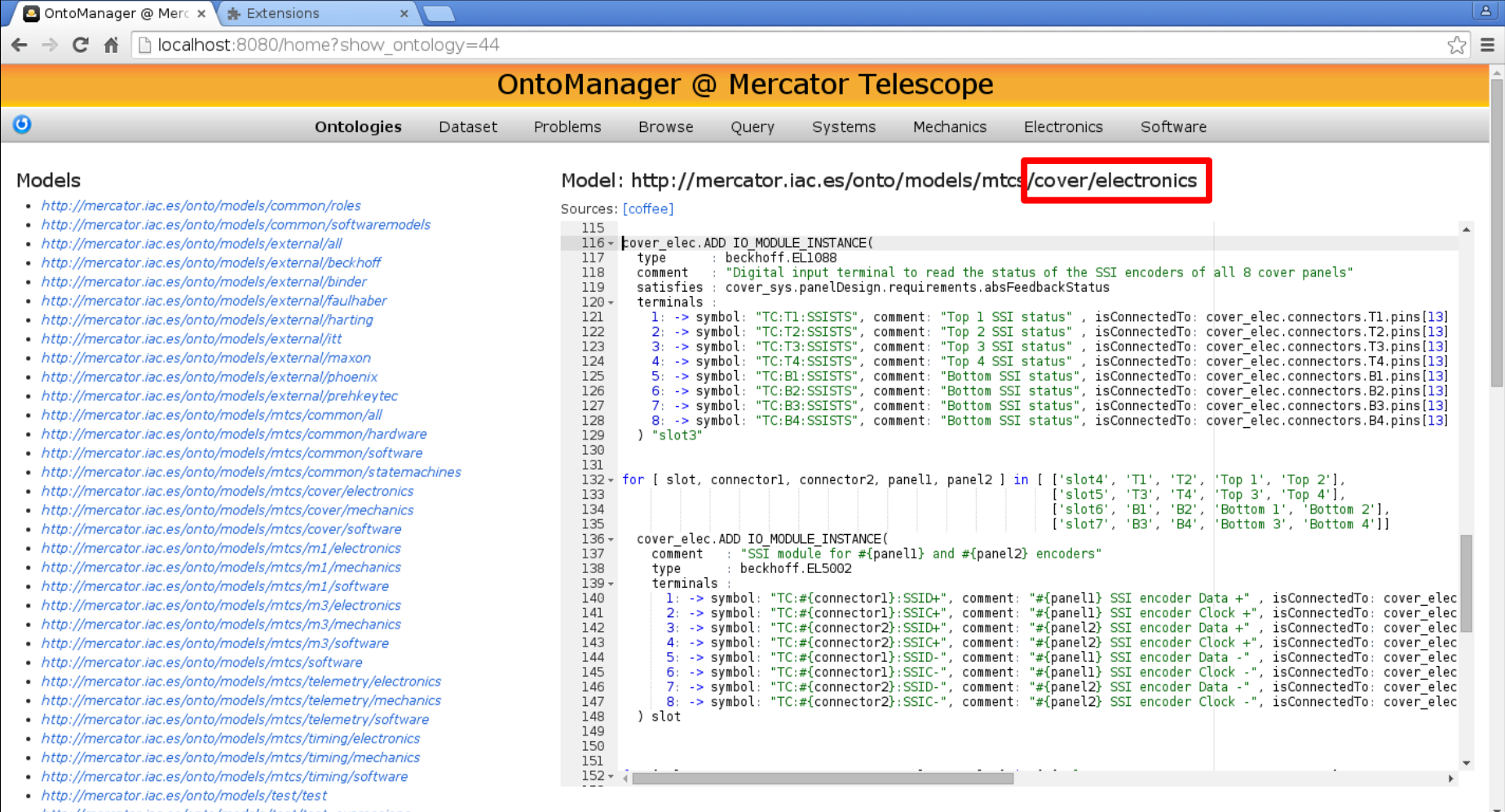
```
115
116 cover_elec.ADD IO_MODULE_INSTANCE(
117     type      : beckhoff.EL1088
118     comment    : "Digital input terminal to read the status of the SSI encoders of all 8 cover panels"
119     satisfies   : cover_sys.panelDesign.requirements.absFeedbackStatus
120     terminals :
121     1: -> symbol: "TC:T1:SSISTS", comment: "Top 1 SSI status", isConnectedTo: cover_elec.connectors.T1.pins[13]
122     2: -> symbol: "TC:T2:SSISTS", comment: "Top 2 SSI status", isConnectedTo: cover_elec.connectors.T2.pins[13]
123     3: -> symbol: "TC:T3:SSISTS", comment: "Top 3 SSI status", isConnectedTo: cover_elec.connectors.T3.pins[13]
124     4: -> symbol: "TC:T4:SSISTS", comment: "Top 4 SSI status", isConnectedTo: cover_elec.connectors.T4.pins[13]
125     5: -> symbol: "TC:B1:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B1.pins[13]
126     6: -> symbol: "TC:B2:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B2.pins[13]
127     7: -> symbol: "TC:B3:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B3.pins[13]
128     8: -> symbol: "TC:B4:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B4.pins[13]
129 ) "slot3"
130
131
132 for [ slot, connector1, connector2, panel1, panel2 ] in [ ['slot4', 'T1', 'T2', 'Top 1', 'Top 2'],
133                                                         ['slot5', 'T3', 'T4', 'Top 3', 'Top 4'],
134                                                         ['slot6', 'B1', 'B2', 'Bottom 1', 'Bottom 2'],
135                                                         ['slot7', 'B3', 'B4', 'Bottom 3', 'Bottom 4']]
136
137 cover_elec.ADD IO_MODULE_INSTANCE(
138     comment    : "SSI module for #{panel1} and #{panel2} encoders"
139     type      : beckhoff.EL5002
140     terminals :
141     1: -> symbol: "TC:#{connector1}:SSID+", comment: "#{panel1} SSI encoder Data +", isConnectedTo: cover_elec
142     2: -> symbol: "TC:#{connector1}:SSIC+", comment: "#{panel1} SSI encoder Clock +", isConnectedTo: cover_elec
143     3: -> symbol: "TC:#{connector2}:SSID+", comment: "#{panel2} SSI encoder Data +", isConnectedTo: cover_elec
144     4: -> symbol: "TC:#{connector2}:SSIC+", comment: "#{panel2} SSI encoder Clock +", isConnectedTo: cover_elec
145     5: -> symbol: "TC:#{connector1}:SSID-", comment: "#{panel1} SSI encoder Data -", isConnectedTo: cover_elec
146     6: -> symbol: "TC:#{connector1}:SSIC-", comment: "#{panel1} SSI encoder Clock -", isConnectedTo: cover_elec
147     7: -> symbol: "TC:#{connector2}:SSID-", comment: "#{panel2} SSI encoder Data -", isConnectedTo: cover_elec
148     8: -> symbol: "TC:#{connector2}:SSIC-", comment: "#{panel2} SSI encoder Clock -", isConnectedTo: cover_elec
149 ) slot
150
151
152
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Example: model of an I/O module **instance**



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows the URL `localhost:8080/home?show_ontology=44`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology models, including:

- <http://mercator.iac.es/onto/models/common/roles>
- <http://mercator.iac.es/onto/models/common/softwaremodels>
- <http://mercator.iac.es/onto/models/external/all>
- <http://mercator.iac.es/onto/models/external/beckhoff>
- <http://mercator.iac.es/onto/models/external/binder>
- <http://mercator.iac.es/onto/models/external/faulhaber>
- <http://mercator.iac.es/onto/models/external/harting>
- <http://mercator.iac.es/onto/models/external/itt>
- <http://mercator.iac.es/onto/models/external/maxon>
- <http://mercator.iac.es/onto/models/external/phoenix>
- <http://mercator.iac.es/onto/models/external/prehkeytec>
- <http://mercator.iac.es/onto/models/mtcs/common/all>
- <http://mercator.iac.es/onto/models/mtcs/common/hardware>
- <http://mercator.iac.es/onto/models/mtcs/common/software>
- <http://mercator.iac.es/onto/models/mtcs/common/statemachines>
- <http://mercator.iac.es/onto/models/mtcs/cover/electronics>
- <http://mercator.iac.es/onto/models/mtcs/cover/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/cover/software>
- <http://mercator.iac.es/onto/models/mtcs/m1/electronics>
- <http://mercator.iac.es/onto/models/mtcs/m1/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/m1/software>
- <http://mercator.iac.es/onto/models/mtcs/m3/electronics>
- <http://mercator.iac.es/onto/models/mtcs/m3/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/m3/software>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/electronics>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/software>
- <http://mercator.iac.es/onto/models/mtcs/timing/electronics>
- <http://mercator.iac.es/onto/models/mtcs/timing/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/timing/software>
- <http://mercator.iac.es/onto/models/test/test>
- http://mercator.iac.es/onto/models/test/test_expressions

The main content area displays the model for the selected ontology: `http://mercator.iac.es/onto/models/mtcs/cover/electronics`. The source is listed as `[coffee]`. The model is defined as follows:

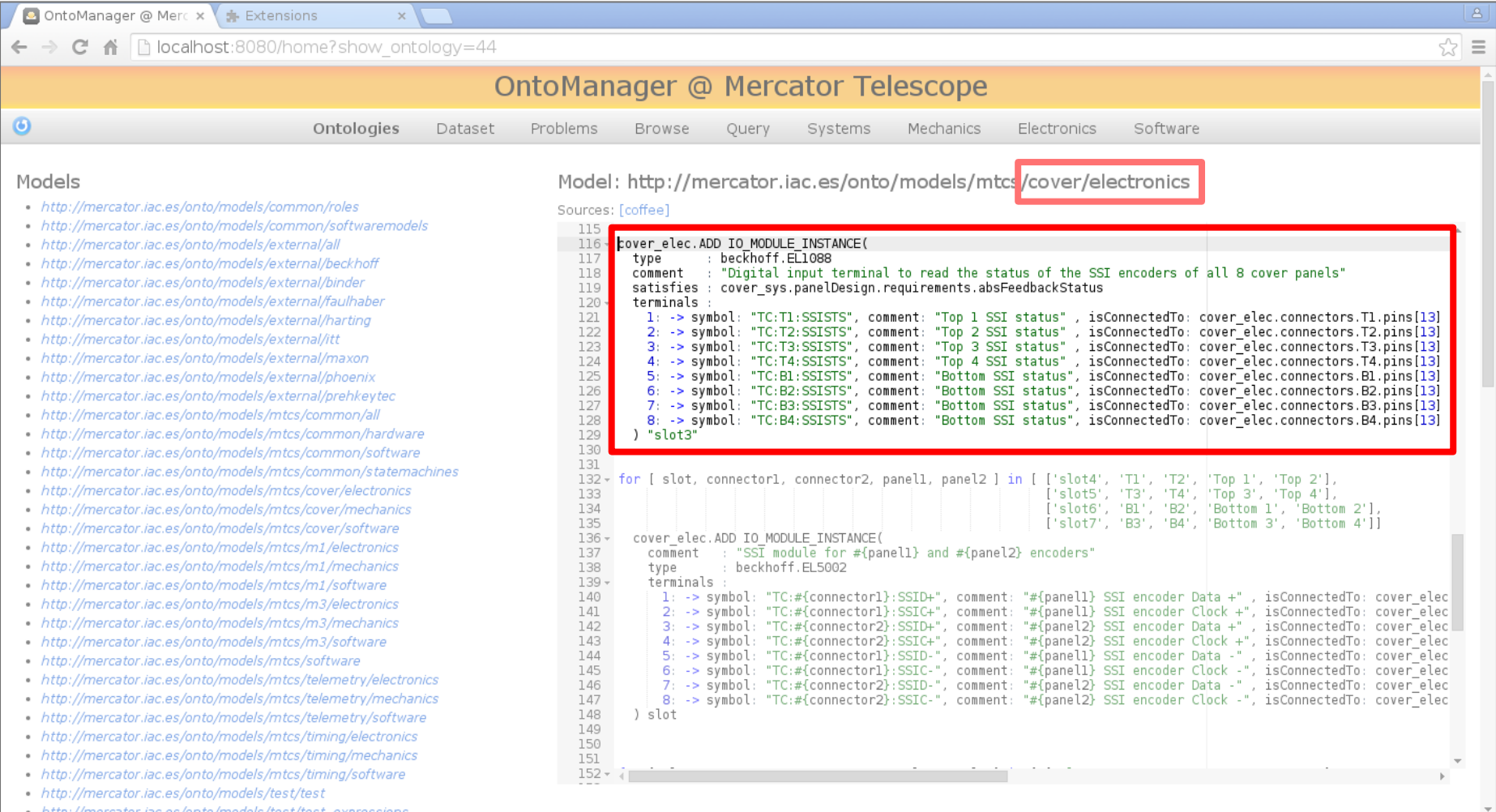
```
115
116 cover_elec.ADD IO_MODULE_INSTANCE(
117   type      : beckhoff.EL1088
118   comment   : "Digital input terminal to read the status of the SSI encoders of all 8 cover panels"
119   satisfies : cover_sys.panelDesign.requirements.absFeedbackStatus
120   terminals :
121     1: -> symbol: "TC:T1:SSISTS", comment: "Top 1 SSI status", isConnectedTo: cover_elec.connectors.T1.pins[13]
122     2: -> symbol: "TC:T2:SSISTS", comment: "Top 2 SSI status", isConnectedTo: cover_elec.connectors.T2.pins[13]
123     3: -> symbol: "TC:T3:SSISTS", comment: "Top 3 SSI status", isConnectedTo: cover_elec.connectors.T3.pins[13]
124     4: -> symbol: "TC:T4:SSISTS", comment: "Top 4 SSI status", isConnectedTo: cover_elec.connectors.T4.pins[13]
125     5: -> symbol: "TC:B1:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B1.pins[13]
126     6: -> symbol: "TC:B2:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B2.pins[13]
127     7: -> symbol: "TC:B3:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B3.pins[13]
128     8: -> symbol: "TC:B4:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B4.pins[13]
129   ) "slot3"
130
131
132 for [ slot, connector1, connector2, panel1, panel2 ] in [ ['slot4', 'T1', 'T2', 'Top 1', 'Top 2'],
133   ['slot5', 'T3', 'T4', 'Top 3', 'Top 4'],
134   ['slot6', 'B1', 'B2', 'Bottom 1', 'Bottom 2'],
135   ['slot7', 'B3', 'B4', 'Bottom 3', 'Bottom 4']]
136
137 cover_elec.ADD IO_MODULE_INSTANCE(
138   comment : "SSI module for #{panel1} and #{panel2} encoders"
139   type    : beckhoff.EL5002
140   terminals :
141     1: -> symbol: "TC:#{connector1}:SSID+", comment: "#{panel1} SSI encoder Data +", isConnectedTo: cover_elec
142     2: -> symbol: "TC:#{connector1}:SSIC+", comment: "#{panel1} SSI encoder Clock +", isConnectedTo: cover_elec
143     3: -> symbol: "TC:#{connector2}:SSID+", comment: "#{panel2} SSI encoder Data +", isConnectedTo: cover_elec
144     4: -> symbol: "TC:#{connector2}:SSIC+", comment: "#{panel2} SSI encoder Clock +", isConnectedTo: cover_elec
145     5: -> symbol: "TC:#{connector1}:SSID-", comment: "#{panel1} SSI encoder Data -", isConnectedTo: cover_elec
146     6: -> symbol: "TC:#{connector1}:SSIC-", comment: "#{panel1} SSI encoder Clock -", isConnectedTo: cover_elec
147     7: -> symbol: "TC:#{connector2}:SSID-", comment: "#{panel2} SSI encoder Data -", isConnectedTo: cover_elec
148     8: -> symbol: "TC:#{connector2}:SSIC-", comment: "#{panel2} SSI encoder Clock -", isConnectedTo: cover_elec
149   ) slot
150
151
152
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

How to build them?

- Example: model of an I/O module **instance**



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=44`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology URLs, including `http://mercator.iac.es/onto/models/mtcs/cover/electronics`. The main content area shows the model `http://mercator.iac.es/onto/models/mtcs/cover/electronics` with sources `[coffee]`.

The model content is displayed in a code editor with line numbers. A red box highlights the `cover_elec.ADD IO_MODULE_INSTANCE` block, which defines a digital input terminal for reading the status of 8 cover panels. The block includes a comment, a `satisfies` constraint, and a list of terminals connected to SSI encoders.

```
115
116 cover_elec.ADD IO_MODULE_INSTANCE(
117   type      : beckhoff.EL1088
118   comment   : "Digital input terminal to read the status of the SSI encoders of all 8 cover panels"
119   satisfies : cover_sys.panelDesign.requirements.absFeedbackStatus
120   terminals :
121     1: -> symbol: "TC:T1:SSISTS", comment: "Top 1 SSI status", isConnectedTo: cover_elec.connectors.T1.pins[13]
122     2: -> symbol: "TC:T2:SSISTS", comment: "Top 2 SSI status", isConnectedTo: cover_elec.connectors.T2.pins[13]
123     3: -> symbol: "TC:T3:SSISTS", comment: "Top 3 SSI status", isConnectedTo: cover_elec.connectors.T3.pins[13]
124     4: -> symbol: "TC:T4:SSISTS", comment: "Top 4 SSI status", isConnectedTo: cover_elec.connectors.T4.pins[13]
125     5: -> symbol: "TC:B1:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B1.pins[13]
126     6: -> symbol: "TC:B2:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B2.pins[13]
127     7: -> symbol: "TC:B3:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B3.pins[13]
128     8: -> symbol: "TC:B4:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B4.pins[13]
129   ) "slot3"
130
131
132 for [ slot, connector1, connector2, panel1, panel2 ] in [ ['slot4', 'T1', 'T2', 'Top 1', 'Top 2'],
133   ['slot5', 'T3', 'T4', 'Top 3', 'Top 4'],
134   ['slot6', 'B1', 'B2', 'Bottom 1', 'Bottom 2'],
135   ['slot7', 'B3', 'B4', 'Bottom 3', 'Bottom 4']]
136
137   cover_elec.ADD IO_MODULE_INSTANCE(
138     comment : "SSI module for #{panel1} and #{panel2} encoders"
139     type    : beckhoff.EL5002
140     terminals :
141       1: -> symbol: "TC:#{connector1}:SSID+", comment: "#{panel1} SSI encoder Data +", isConnectedTo: cover_elec
142       2: -> symbol: "TC:#{connector1}:SSIC+", comment: "#{panel1} SSI encoder Clock +", isConnectedTo: cover_elec
143       3: -> symbol: "TC:#{connector2}:SSID+", comment: "#{panel2} SSI encoder Data +", isConnectedTo: cover_elec
144       4: -> symbol: "TC:#{connector2}:SSIC+", comment: "#{panel2} SSI encoder Clock +", isConnectedTo: cover_elec
145       5: -> symbol: "TC:#{connector1}:SSID-", comment: "#{panel1} SSI encoder Data -", isConnectedTo: cover_elec
146       6: -> symbol: "TC:#{connector1}:SSIC-", comment: "#{panel1} SSI encoder Clock -", isConnectedTo: cover_elec
147       7: -> symbol: "TC:#{connector2}:SSID-", comment: "#{panel2} SSI encoder Data -", isConnectedTo: cover_elec
148       8: -> symbol: "TC:#{connector2}:SSIC-", comment: "#{panel2} SSI encoder Clock -", isConnectedTo: cover_elec
149     ) slot
150
151
152
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

How to use them?

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

The screenshot shows a web browser window with the title "OntoManager @ Merc" and a single tab. The address bar displays the URL "localhost:8080/elec?open=configuration;path=cover_elec:Cover". The page has a yellow header bar with the text "OntoManager @ Mercator Telescope". Below the header is a navigation bar with a blue circular icon and several tabs: "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Electronics" tab is currently selected. The main content area displays a hierarchical tree structure. The root node is "Cover", which is expanded to show three sub-nodes: "I/O modules", "terminals", and "connectors". Each node has a small square icon with a plus sign next to it. Below the "Cover" node, there are four more nodes: "M1", "M3", "Telemetry", and "Timing", each also with a plus icon. A mouse cursor is pointing at the "Cover" node.

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design



The screenshot displays the OntoManager @ Mercator Telescope web application. The browser address bar shows the URL: `localhost:8080/elec?open=configuration;path=cover_elec:Cover::I%2FO%20modules`. The application header features the title "OntoManager @ Mercator Telescope" and a navigation menu with tabs: Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, **Electronics**, and Software. The main content area shows a hierarchical tree structure for the "Cover" ontology. A mouse cursor is pointing at the "I/O modules" node. The tree structure is as follows:

- Cover
 - + I/O modules
 - + slot0
 - + slot1
 - + slot2
 - + slot3
 - + slot4
 - + slot5
 - + slot6
 - + slot7
 - + slot8
 - + slot9
 - + slot10
 - + slot11
 - + slot12
 - + slot13
 - + terminals
 - + connectors
- + M1
- + M3
- + Telemetry
- + Timing

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

The screenshot shows a web browser window with the title "OntoManager @ Merc" and a URL bar containing "localhost:8080/elec?open=configuration;path=cover_elec:Cover::terminals". The page header is "OntoManager @ Mercator Telescope". Below the header is a navigation bar with tabs: "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The main content area displays a tree view of an ontology. The root node is "Cover", which is expanded. Under "Cover", there are three main branches: "I/O modules", "terminals", and "connectors". The "I/O modules" branch contains a list of slots from "slot0" to "slot13". The "terminals" branch contains a list of terminals: "PE", "L", "H", "24V", and "GND". The "connectors" branch contains a list of connectors: "M1", "M3", and "Telemetry". A mouse cursor is pointing at the "terminals" branch.

OntoManager @ Merc

localhost:8080/elec?open=configuration;path=cover_elec:Cover::terminals

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

- Cover
 - I/O modules
 - + slot0
 - + slot1
 - + slot2
 - + slot3
 - + slot4
 - + slot5
 - + slot6
 - + slot7
 - + slot8
 - + slot9
 - + slot10
 - + slot11
 - + slot12
 - + slot13
 - terminals
 - + PE
 - + L
 - + H
 - + 24V
 - + GND
 - + connectors
 - + M1
 - + M3
 - + Telemetry

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

The screenshot displays the OntoManager @ Mercator Telescope web application. The browser address bar shows the URL: `localhost:8080/elec?open=configuration;path=cover_elec:Cover::connectors`. The application's main header is orange and contains the title "OntoManager @ Mercator Telescope". Below the header is a navigation bar with tabs: "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Electronics" tab is currently selected. The main content area shows a hierarchical tree structure under the "Cover" ontology. The tree is expanded to show the "connectors" sub-ontology, which includes the following elements:

- ☐ Cover
 - ☐ I/O modules
 - ☐ slot0
 - ☐ slot1
 - ☐ slot2
 - ☐ slot3
 - ☐ slot4
 - ☐ slot5
 - ☐ slot6
 - ☐ slot7
 - ☐ slot8
 - ☐ slot9
 - ☐ slot10
 - ☐ slot11
 - ☐ slot12
 - ☐ slot13
 - ☐ terminals
 - ☐ PE
 - ☐ L
 - ☐ N
 - ☐ 24V
 - ☐ GND
 - ☐ connectors
 - ☐ ECAT
 - ☐ T1
 - ☐ T2

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot4

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

Cover

- I/O modules
 - slot0
 - slot1
 - slot2
 - slot3
 - slot4**
 - slot5
 - slot6
 - slot7
 - slot8
 - slot9
 - slot10
 - slot11
 - slot12
 - slot13
- terminals
 - PE
 - L
 - II
 - 24V
 - GND
- connectors
 - ECAT
 - T1
 - T2

I/O Module instance slot4

SSI module for Top 1 and Top 2 encoders

Module type summary

ID	EL5002
Manufacturer	Beckhoff Automation
Description	2-channel SSI encoder
Used in	Cover (4)

Run LED1

D1+ D1- CL1+ CL1- D2+ D2- CL2+ CL2-

Power contact +24 V

Power contact 0 V

Data Clock

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot4

Connections

Type (EL5002)				Instance		
Channel	Terminal	Symbol	Description	Symbol	Description	Connected to
1	1	D1+	Channel 1: Data +	TC:T1:SSID+	Top 1 SSI encoder Data +	Connector T1 : pin 6
	2	CL1+	Channel 1: Clock +	TC:T1:SSIC+	Top 1 SSI encoder Clock +	Connector T1 : pin 7
	5	D1-	Channel 1: Data -	TC:T1:SSID-	Top 1 SSI encoder Data -	Connector T1 : pin 14
	6	CL1-	Channel 1: Clock -	TC:T1:SSIC-	Top 1 SSI encoder Clock -	Connector T1 : pin 15
2	3	D2+	Channel 2: Data +	TC:T2:SSID+	Top 2 SSI encoder Data +	Connector T2 : pin 6
	4	CL2+	Channel 2: Clock +	TC:T2:SSIC+	Top 2 SSI encoder Clock +	Connector T2 : pin 7
	7	D2-	Channel 2: Data -	TC:T2:SSID-	Top 2 SSI encoder Data -	Connector T2 : pin 14
	8	CL2-	Channel 2: Clock -	TC:T2:SSIC-	Top 2 SSI encoder Clock -	Connector T2 : pin 15

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=ConnectorInstance;qname=cover_elec:connectors.T1

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

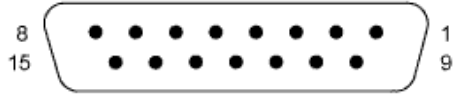
Cover

- I/O modules
 - slot0
 - slot1
 - slot2
 - slot3
 - slot4
 - slot5
 - slot6
 - slot7
 - slot8
 - slot9
 - slot10
 - slot11
 - slot12
 - slot13
- terminals
 - PE
 - L
 - N
 - 24V
 - GND
- connectors
 - ECAT
 - T1
 - T2

Connector instance T1

Connector type summary

ID	D-sub 15 F
Gender	female
Manufacturer	ITT Corporation
Description	D-sub 15 female connector
Fits to	D-sub 15 M
Used in	Cover (8), M1 (1), M3 (2)



DA-15S (Female Socket Front View)

Connections

Type (D-sub 15 F)			Instance		
Pin	Symbol	Description	Symbol	Description	Connected to
1	1	Pin 1	TC:T1:GND HM	Top 1 GND of holding magnet	Cover : terminal GND
2	2	Pin 2	TC:T1:GND MOT	Top 1 GND of motor	Cover : terminal GND
3	3	Pin 3	TC:T1:MMON	Top 1 motor monitor	
4	4	Pin 4	TC:T1:MDIR	Top 1 motor direction	I/O module slot1 : terminal 1
5	5	Pin 5	TC:T1:GND ENC	Top 1 GND of encoder	Cover : terminal GND

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x


localhost:8080/elec?show=ConnectorInstance;qname=cover_elec:connectors.T1

slot5
slot6
slot7
slot8
slot9
slot10
slot11
slot12
slot13

terminals
PE
L
N
24V
GND

connectors
ECAT
T1
T2
T3
T4
B1
B2
B3
B4
M1
M3
Telemetry
Timing

Description	D-sub 15 female connector
Fits to	D-sub 15 M
Used in	Cover (8), M1 (1), M3 (2)



DA-15S (Female Socket Front View)

Connections

Type (D-sub 15 F)			Instance		
Pin	Symbol	Description	Symbol	Description	Connected to
1	1	Pin 1	TC:T1:GND HM	Top 1 GND of holding magnet	Cover : terminal GND
2	2	Pin 2	TC:T1:GND MOT	Top 1 GND of motor	Cover : terminal GND
3	3	Pin 3	TC:T1:MMON	Top 1 motor monitor	
4	4	Pin 4	TC:T1:MDIR	Top 1 motor direction	I/O module slot1 : terminal 1
5	5	Pin 5	TC:T1:GND ENC	Top 1 GND of encoder	Cover : terminal GND
6	6	Pin 6	TC:T1:SSID+	Top 1 SSI Data +	I/O module slot4 : terminal 1
7	7	Pin 7	TC:T1:SSIC+	Top 1 SSI Clock +	I/O module slot4 : terminal 2
8	8	Pin 8	TC:T1:PE	Top 1 Earth	Cover : terminal PE
9	9	Pin 9	TC:T1:+24V HM	Top 1 +24V of holding magnet	I/O module slot12 : terminal 2
10	10	Pin 10	TC:T1:+24V MOT	Top 1 +24V of motor	I/O module slot8 : terminal 2
11	11	Pin 11	TC:T1:MSPEED	Top 1 motor speed	I/O module slot2 : terminal 1
12	12	Pin 12	TC:T1:+24V ENC	Top 1 +24V of encoder	Cover : terminal 24V
13	13	Pin 13	TC:T1:SSISTS	Top 1 SSI status	I/O module slot3 : terminal 1
14	14	Pin 14	TC:T1:SSID-	Top 1 SSI Data -	I/O module slot4 : terminal 5
15	15	Pin 15	TC:T1:SSIC-	Top 1 SSI Clock -	I/O module slot4 : terminal 6

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot3

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

Cover

- I/O modules
 - + slot0
 - + slot1
 - + slot2
 - + slot3
 - + slot4
 - + slot5
 - + slot6
 - + slot7
 - + slot8
 - + slot9
 - + slot10
 - + slot11
 - + slot12
 - + slot13
- terminals
 - + PE
 - + L
 - + N
 - + 24V
 - + GND
- connectors
 - + ECAT
 - + T1
 - + T2

I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

System properties

Satisfies `cover_sys:panelDesign.requirements.absFeedbackStatus`

Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

The diagram illustrates the terminal block and internal circuitry of the I/O module. On the left, terminal labels point to specific pins: Signal LED1, Signal LED3, Signal LED5, Signal LED7, Input 1, Input 3, and Power contact +24 V. On the right, labels point to Signal LED2, Signal LED4, Signal LED6, Signal LED8, Input 2, and Input 4. The internal circuitry shows the connection of these inputs to the module's logic, including a 24V DC supply and negative switching components.

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot3

System properties

Satisfies [cover_sys:panelDesign.requirements.absFeedbackStatus](#)

Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

Diagram: A physical terminal block (EL1088) and its internal wiring diagram. The terminal block has 16 pins labeled I1-I8, I9-I16, and I17-I24. It includes 8 signal LEDs (LED1-LED8) and 8 inputs (Input 1-Input 8). The wiring diagram shows the internal circuitry for each input channel, including a 24V DC supply, a 0V supply, and a switch.

Connections

Left sidebar (Tree view):

- slot3
- slot4
- slot5
- slot6
- slot7
- slot8
- slot9
- slot10
- slot11
- slot12
- slot13
- terminals
 - PE
 - L
 - N
 - 24V
 - GND
- connectors
 - ECAT
 - T1
 - T2
 - T3
 - T4
 - B1
 - B2
 - B3
 - B4
- M1
- M3
- Telemetry

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Systems design

The screenshot shows a web browser window with the URL `localhost:8080/sys?show=requirement;qname=cover_sys:panelDesign.requirements.absFeedbackStatus`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Systems" link is highlighted with a mouse cursor. On the left sidebar, a tree view shows the hierarchy: "Cover" (expanded), "M1", "M3", "Telemetry", and "Timing". The main content area displays the requirement "Requirement `absFeedbackStatus`" with a description: "The status of the absolute feedback shall be known". Below this, a "Properties" table lists the relationships for the requirement.

Properties	
Derives	
Derived from	<ul style="list-style-type: none">← <code>cover_sys:concept.requirements.monitorable</code>← <code>cover_sys:panelDesign.requirements.absFeedback</code>
Satisfied by	<ul style="list-style-type: none">• <code>cover_sys:panelDesign.parts.encoder</code>• <code>cover_elec:slot3</code>
Declared by	<ul style="list-style-type: none">• <code>cover_sys:panelDesign</code>

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Systems design

The screenshot shows a web browser window with the URL `localhost:8080/sys?show=requirement;qname=cover_sys:panelDesign.requirements.absFeedbackStatus`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, **Systems**, Mechanics, Electronics, and Software. On the left sidebar, there is a tree view with expandable items: Cover, M1, M3, Telemetry, and Timing. The main content area displays the "Requirement `absFeedbackStatus`". Below the title, a yellow box contains the text: "The status of the absolute feedback shall be known". Under the "Properties" section, there is a table with the following data:

Derives	
Derived from	<ul style="list-style-type: none">← <code>cover_sys:concept.requirements.monitorable</code>← <code>cover_sys:panelDesign.requirements.absFeedback</code>
Satisfied by	<ul style="list-style-type: none">• <code>cover_sys:panelDesign.parts.encoder</code>• <code>cover_elec:slot3</code>
Declared by	<ul style="list-style-type: none">• <code>cover_sys:panelDesign</code>

A mouse cursor is pointing at the `cover_sys:panelDesign` link in the "Declared by" row.

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Systems design

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query **Systems** Mechanics Electronics Software

Design **panelDesign**

The design of the telescope cover panels

Requirements derivation matrix

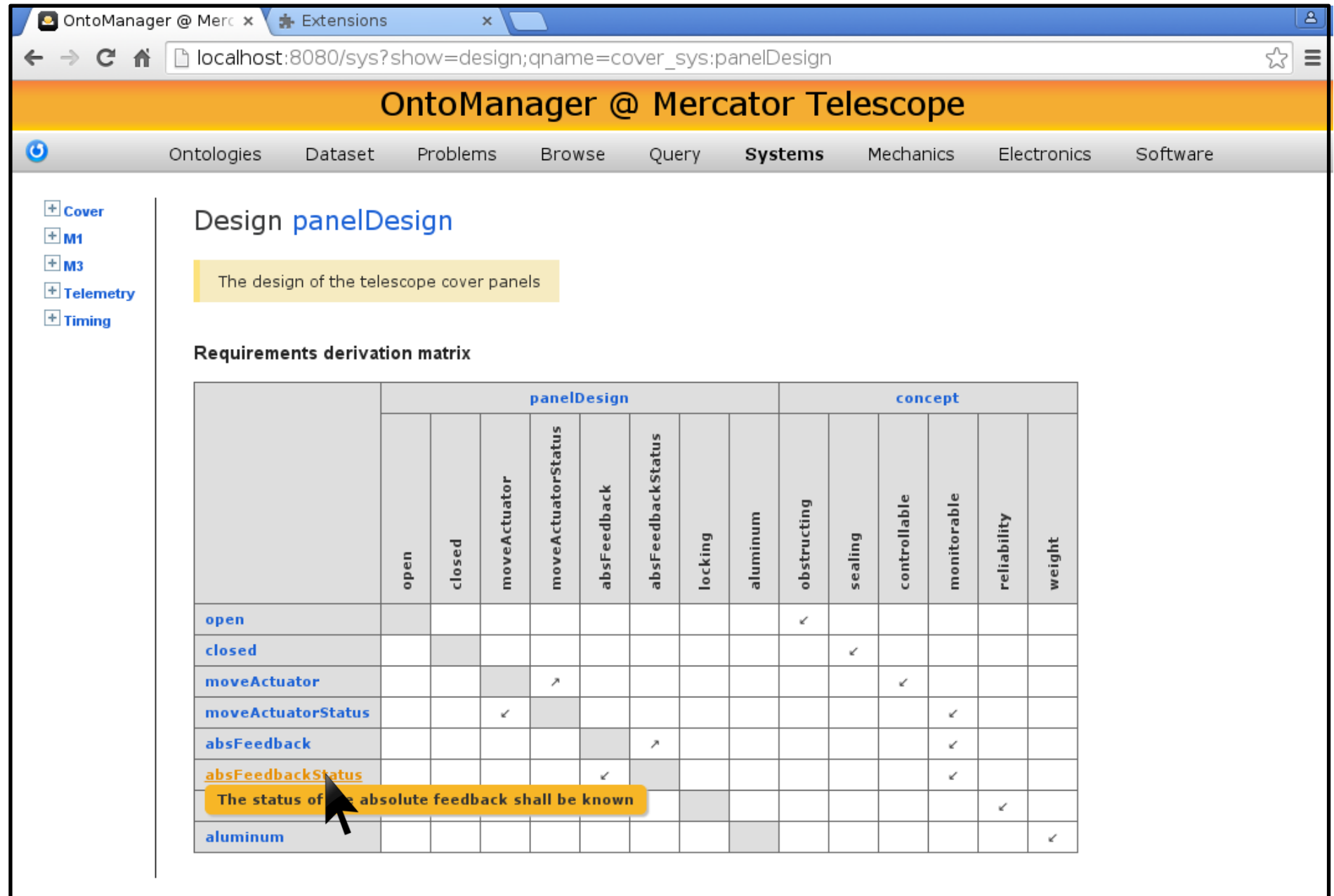
	panelDesign								concept					
	open	closed	moveActuator	moveActuatorStatus	absFeedback	absFeedbackStatus	locking	aluminum	obstructing	sealing	controllable	monitorable	reliability	weight
open									↙					
closed										↙				
moveActuator				→							↙			
moveActuatorStatus			↙									↙		
absFeedback						→							↙	
absFeedbackStatus					↙							↙		
													↙	
aluminum														↙

The status of the absolute feedback shall be known

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Systems design



OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query **Systems** Mechanics Electronics Software

Design **panelDesign**

The design of the telescope cover panels

Requirements derivation matrix

	panelDesign								concept					
	open	closed	moveActuator	moveActuatorStatus	absFeedback	absFeedbackStatus	locking	aluminum	obstructing	sealing	controllable	monitorable	reliability	weight
open									↙					
closed										↙				
moveActuator				↗							↙			
moveActuatorStatus			↙									↙		
absFeedback						↗							↙	
absFeedbackStatus						↙							↙	
													↙	
aluminum														↙

The status of the absolute feedback shall be known

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Systems design

The screenshot shows a web browser window with the title 'OntoManager @ Merc'. The address bar shows the URL 'localhost:8080/sys?show=requirement;qname=cover_sys:panelDesign.requirements.absFeedbackStatus'. The page has a yellow header with the title 'OntoManager @ Mercator Telescope'. Below the header is a navigation bar with tabs: 'Ontologies', 'Dataset', 'Problems', 'Browse', 'Query', 'Systems' (selected), 'Mechanics', 'Electronics', and 'Software'. On the left side, there is a sidebar with a tree view showing a hierarchy: 'Cover' (expanded), 'M1', 'M3', 'Telemetry', and 'Timing'. The main content area displays the 'Requirement absFeedbackStatus'. It includes a yellow box with the text 'The status of the absolute feedback shall be known'. Below this, there is a section titled 'Properties' containing a table with the following data:

Derives	
Derived from	<ul style="list-style-type: none">← cover_sys:concept.requirements.monitorable← cover_sys:panelDesign.requirements.absFeedback
Satisfied by	<ul style="list-style-type: none">• cover_sys:panelDesign.parts.encoder• cover_elec:slot3
Declared by	<ul style="list-style-type: none">• cover_sys:panelDesign

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot3

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

Cover

- I/O modules
 - slot0
 - slot1
 - slot2
 - slot3
 - slot4
 - slot5
 - slot6
 - slot7
 - slot8
 - slot9
 - slot10
 - slot11
 - slot12
 - slot13
- terminals
 - PE
 - L
 - II
 - 24V
 - GND
- connectors
 - ECAT
 - T1
 - T2

I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

System properties

Satisfies [cover_sys:panelDesign.requirements.absFeedbackStatus](#)

Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

The diagram illustrates the terminal block and internal circuitry of the I/O module. On the left, terminal labels point to specific pins: Signal LED1, Signal LED3, Signal LED5, Signal LED7, Input 1, Input 3, and Power contact +24 V. On the right, labels point to Signal LED2, Signal LED4, Signal LED6, Signal LED8, Input 2, and Input 4. The internal circuitry shows the connection of these inputs to the module's logic, including a power supply section with a 24V input and a ground connection.

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Mercator Telescope

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot3

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

System properties

Satisfies `cover_sys:panelDesign.requirements.absFeedbackStatus`

Module type summary

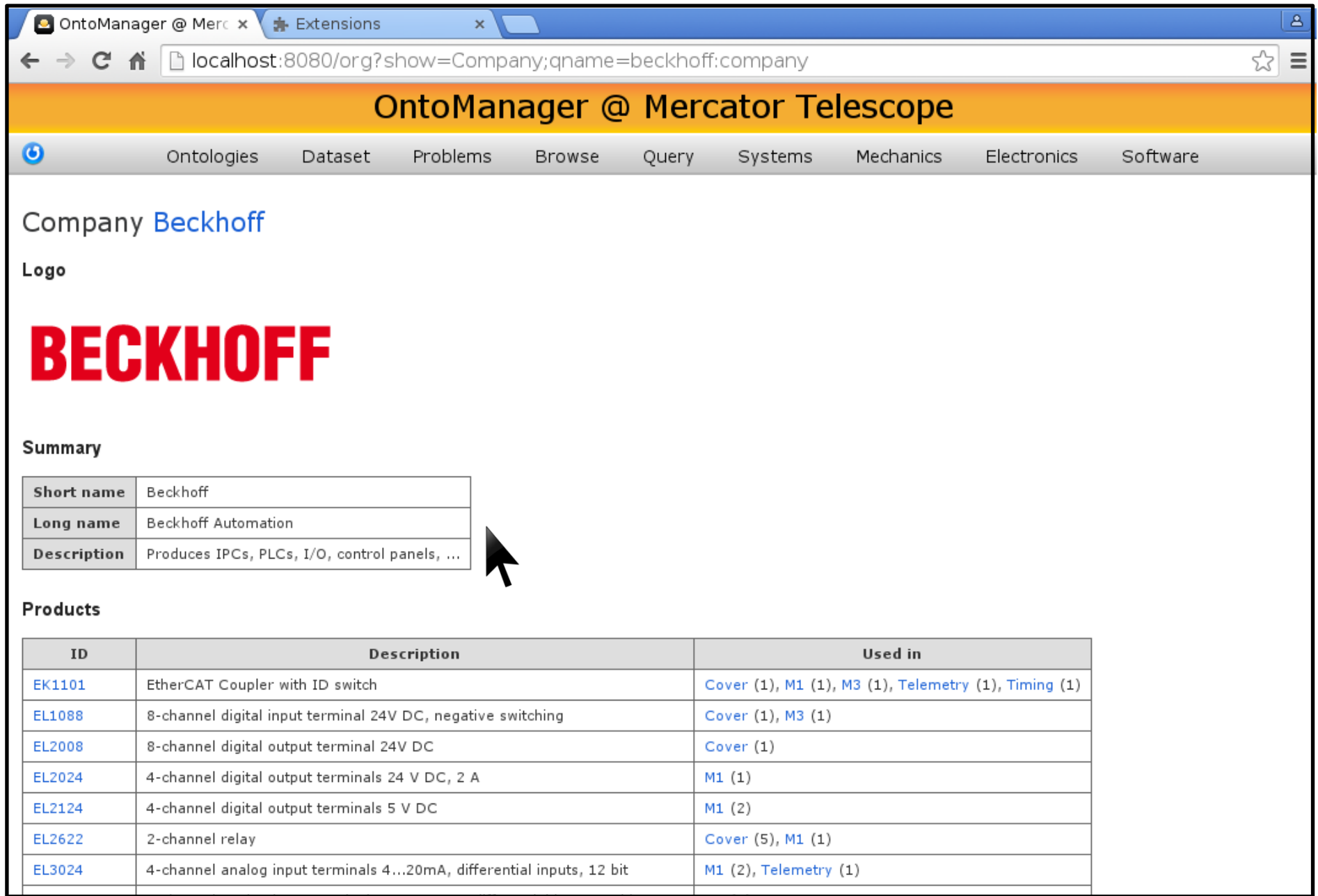
ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

Signal LED1
Signal LED3
Signal LED5
Signal LED7
Signal LED2
Signal LED4
Signal LED6
Signal LED8
Input 1
Input 2
Input 3
Power contact +24 V
Input 4

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design



The screenshot shows a web browser window with the URL `localhost:8080/org?show=Company;qname=beckhoff:company`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. The main content area displays the "Company Beckhoff" profile, including a logo and a summary table. A mouse cursor is pointing at the summary table. Below the summary table is a "Products" section with a table listing various products and their usage.

Company **Beckhoff**

Logo

BECKHOFF

Summary

Short name	Beckhoff
Long name	Beckhoff Automation
Description	Produces IPCs, PLCs, I/O, control panels, ...

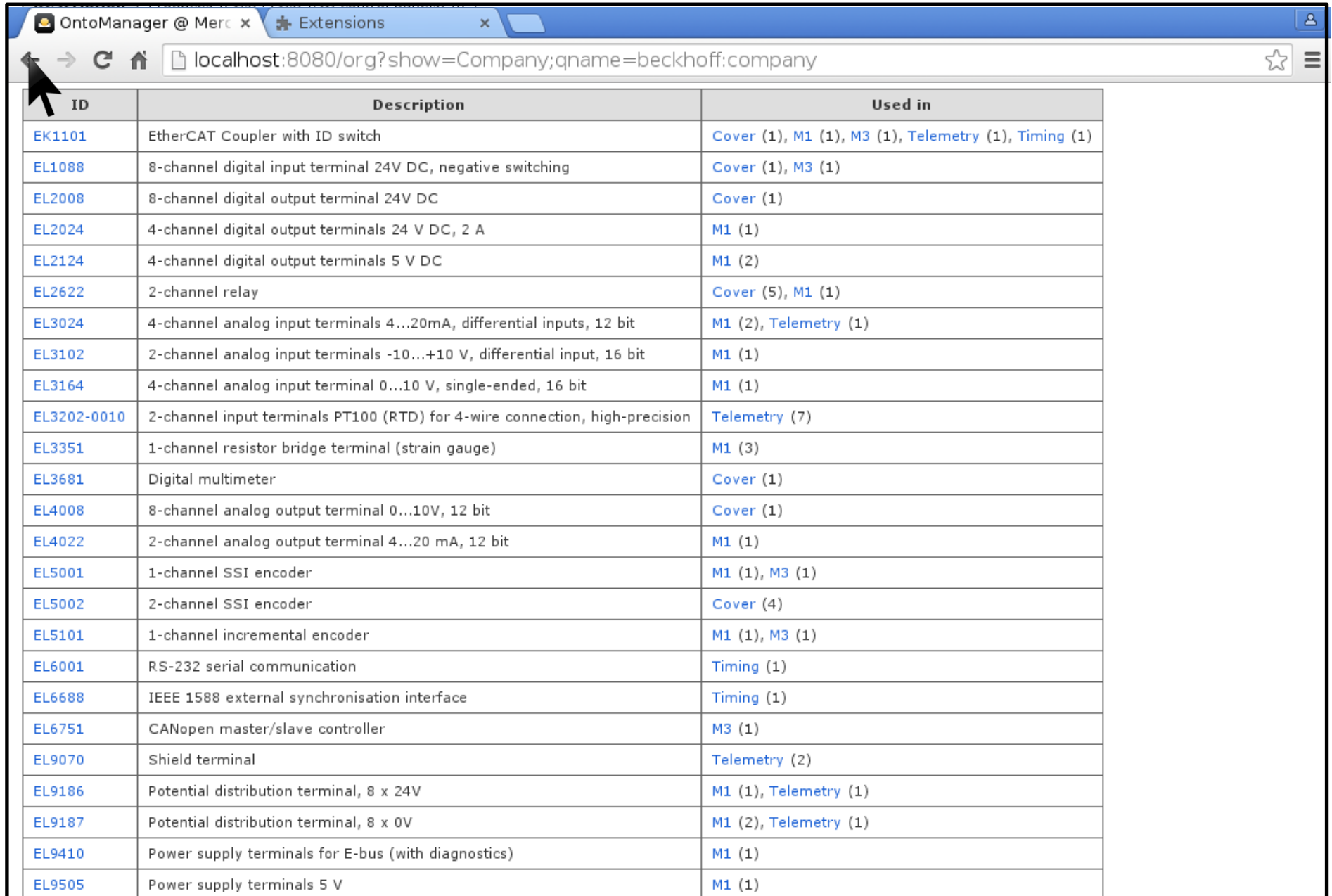
Products

ID	Description	Used in
EK1101	EtherCAT Coupler with ID switch	Cover (1), M1 (1), M3 (1), Telemetry (1), Timing (1)
EL1088	8-channel digital input terminal 24V DC, negative switching	Cover (1), M3 (1)
EL2008	8-channel digital output terminal 24V DC	Cover (1)
EL2024	4-channel digital output terminals 24 V DC, 2 A	M1 (1)
EL2124	4-channel digital output terminals 5 V DC	M1 (2)
EL2622	2-channel relay	Cover (5), M1 (1)
EL3024	4-channel analog input terminals 4...20mA, differential inputs, 12 bit	M1 (2), Telemetry (1)

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design



The screenshot shows a web browser window with the title "OntoManager @ Merc" and a tab for "Extensions". The address bar displays "localhost:8080/org?show=Company;qname=beckhoff:company". A mouse cursor is pointing at the first row of the table. The table has three columns: "ID", "Description", and "Used in". It lists various Beckhoff components and their associated modules.

ID	Description	Used in
EK1101	EtherCAT Coupler with ID switch	Cover (1), M1 (1), M3 (1), Telemetry (1), Timing (1)
EL1088	8-channel digital input terminal 24V DC, negative switching	Cover (1), M3 (1)
EL2008	8-channel digital output terminal 24V DC	Cover (1)
EL2024	4-channel digital output terminals 24 V DC, 2 A	M1 (1)
EL2124	4-channel digital output terminals 5 V DC	M1 (2)
EL2622	2-channel relay	Cover (5), M1 (1)
EL3024	4-channel analog input terminals 4...20mA, differential inputs, 12 bit	M1 (2), Telemetry (1)
EL3102	2-channel analog input terminals -10...+10 V, differential input, 16 bit	M1 (1)
EL3164	4-channel analog input terminal 0...10 V, single-ended, 16 bit	M1 (1)
EL3202-0010	2-channel input terminals PT100 (RTD) for 4-wire connection, high-precision	Telemetry (7)
EL3351	1-channel resistor bridge terminal (strain gauge)	M1 (3)
EL3681	Digital multimeter	Cover (1)
EL4008	8-channel analog output terminal 0...10V, 12 bit	Cover (1)
EL4022	2-channel analog output terminal 4...20 mA, 12 bit	M1 (1)
EL5001	1-channel SSI encoder	M1 (1), M3 (1)
EL5002	2-channel SSI encoder	Cover (4)
EL5101	1-channel incremental encoder	M1 (1), M3 (1)
EL6001	RS-232 serial communication	Timing (1)
EL6688	IEEE 1588 external synchronisation interface	Timing (1)
EL6751	CANopen master/slave controller	M3 (1)
EL9070	Shield terminal	Telemetry (2)
EL9186	Potential distribution terminal, 8 x 24V	M1 (1), Telemetry (1)
EL9187	Potential distribution terminal, 8 x 0V	M1 (2), Telemetry (1)
EL9410	Power supply terminals for E-bus (with diagnostics)	M1 (1)
EL9505	Power supply terminals 5 V	M1 (1)

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot3

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

Cover

- I/O modules
 - slot0
 - slot1
 - slot2
 - slot3
 - slot4
 - slot5
 - slot6
 - slot7
 - slot8
 - slot9
 - slot10
 - slot11
 - slot12
 - slot13
- terminals
 - PE
 - L
 - N
 - 24V
 - GND
- connectors
 - ECAT
 - T1
 - T2

I/O Module instance slot3

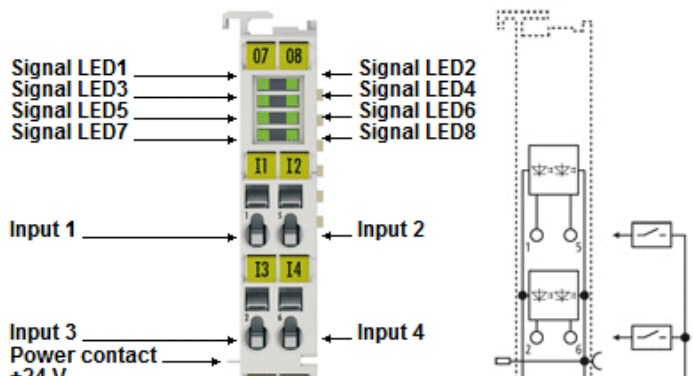
Digital input terminal to read the status of the SSI encoders of all 8 cover panels

System properties

Satisfies `cover_sys:panelDesign.requirements.absFeedbackStatus`

Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover_elec:slot3

Connections

Type (EL1088)				Instance		
Channel	Terminal	Symbol	Description	Symbol	Description	Connected to
1	1	I1	Input 1	TC:T1:SSISTS	Top 1 SSI encoder status	Connector T1 : pin 13
2	2	I2	Input 2	TC:T2:SSISTS	Top 2 SSI encoder status	Connector T2 : pin 13
3	3	I3	Input 3	TC:T3:SSISTS	Top 3 SSI encoder status	Connector T3 : pin 13
4	4	I4	Input 4	TC:T4:SSISTS	Top 4 SSI encoder status	Connector T4 : pin 13
5	5	I5	Input 5	TC:B1:SSISTS	Bottom SSI encoder status	Connector B1 : pin 13
6	6	I6	Input 6	TC:B2:SSISTS	Bottom SSI encoder status	Connector B2 : pin 13
7	7	I7	Input 7	TC:B3:SSISTS	Bottom SSI encoder status	Connector B3 : pin 13
8	8	I8	Input 8	TC:B4:SSISTS	Bottom SSI encoder status	Connector B4 : pin 13

Interface

Variable	Type	Description	Linked variable
input1	BOOL	Input 1	interface.parts.cover.parts.top.parts.p1.encoderErrorSignal
input2	BOOL	Input 2	interface.parts.cover.parts.top.parts.p2.encoderErrorSignal
input3	BOOL	Input 3	interface.parts.cover.parts.top.parts.p3.encoderErrorSignal
input4	BOOL	Input 4	interface.parts.cover.parts.top.parts.p4.encoderErrorSignal
input5	BOOL	Input 5	interface.parts.cover.parts.bottom.parts.p1.encoderErrorSignal
input6	BOOL	Input 6	interface.parts.cover.parts.bottom.parts.p2.encoderErrorSignal
input7	BOOL	Input 7	interface.parts.cover.parts.bottom.parts.p3.encoderErrorSignal
input8	BOOL	Input 8	interface.parts.cover.parts.bottom.parts.p4.encoderErrorSignal
WcState	BOOL	EtherCAT Working counter state	interface.parts.cover.parts.io.parts.slot3.wcState
InfoDataState	UINT	EtherCAT state (INIT, PREOP, OP, ...)	interface.parts.cover.parts.io.parts.slot3.infoData

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

The screenshot shows the OntoManager @ Mercator Telescope web application. The browser address bar displays `localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel`. The application has a navigation bar with tabs: Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. On the left, a tree view shows the project structure: Tc2_MC2, mtcs_common, mtcs_cover, mtcs_m1, mtcs_m3, mtcs_telemetry, mtcs_timing, and mtcs. The main content area displays the FunctionBlock **SM_CoverPanel**. It features a diagram with inputs on the left (encoderErrorSignal, initializationStatus, operatorStatus, operatingStatus, config, coverConfig) and outputs on the right (actualStatus, statuses, parts, processes). At the bottom right of the diagram are two methods: `startOpening()` and `startClosing()`. To the right of the diagram is a 'Jump to:' menu with links: Variables, Methods, Implementation, and PLCopen XML serialization. Below the diagram is a 'Variables' table.

Variable	Name	Type	Initial value	Address	Description	Qualif
VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

OntoManager @ Merc x

localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel

FunctionBlock SM_CoverPanel

The diagram shows the SM_CoverPanel FunctionBlock with the following connections:

- Inputs (left): encoderErrorSignal, initializationStatus ▷, operatorStatus ▷, operatingStatus ▷, config ▷, coverConfig ▷
- Outputs (right): actualStatus, statuses, parts, processes
- Methods (bottom right): startOpening(), startClosing()

Jump to:

- Variables
- Methods
- Implementation
- PLCopen XML serialization

Variables

Variable	Name	Type	Initial value	Address	Description	Qualif
VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC
	statuses	CoverPanelStatuses			Statuses of the state machine	
	parts	CoverPanelParts			Parts of the state machine	
	processes	CoverPanelProcesses			Processes of the state machine	

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

OntoManager @ Merc x

localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel

VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC
	statuses	CoverPanelStatuses			Statuses of the state machine	
	parts	CoverPanelParts			Parts of the state machine	
	processes	CoverPanelProcesses			Processes of the state machine	

Methods

- **startOpening()**

Comment	Start opening the panel						
Return type	RequestResults						
Interface	Variable	Name	Type	Initial value	Address	Description	Qualifiers
Implementation	startOpening := THIS^.processes.startOpening.request();						

- **startClosing()**

Comment	Start closing the panel						
Return type	RequestResults						
Interface	Variable	Name	Type	Initial value	Address	Description	Qualifiers
Implementation	startClosing := THIS^.processes.startClosing.request();						

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Software design

OntoManager @ Merc x

localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel

Implementation startOpening := THIS^.processes.startOpening.request();

- startClosing()

Comment	Start closing the panel						
Return type	RequestResults						
Interface	Variable	Name	Type	Initial value	Address	Description	Qualifiers
Implementation	startClosing := THIS^.processes.startClosing.request();						

Implementation

```
parts.axis(  
  isEnabled := operatorStatus.tech AND (operatingStatus.manual AND initializationStatus.initialized),  
  standstillTolerance := config.standstillTolerance);  
parts.motorRelay( isEnabled := parts.axis.isEnabled );  
statuses.busyStatus( isBusy := parts.axis.statuses.busyStatus.busy OR parts.motorRelay.statuses.busyStatus.busy );  
statuses.apertureStatus(  
  isOpen := (ABS(config.openPosition - parts.axis.actPos.degrees.value)) < config.openTolerance,  
  isClosed := (ABS(config.closedPosition - parts.axis.actPos.degrees.value)) < config.closedTolerance);  
statuses.healthStatus(  
  isGood := parts.axis.statuses.healthStatus.isGood AND (NOT(encoderErrorSignal)),  
  hasWarning := parts.axis.statuses.healthStatus.hasWarning);  
statuses.openingStatus(  
  isOpening := parts.axis.statuses.motionStatus.backward,  
  isClosing := parts.axis.statuses.motionStatus.forward);  
processes.startOpening( isEnabled := operatorStatus.tech AND (operatingStatus.manual AND initializationStatus.initialized) );  
processes.startClosing( isEnabled := operatorStatus.tech AND (operatingStatus.manual AND initializationStatus.initialized) );
```

PLCopen XML serialization

```
1 <pou name="SM_CoverPanel" pouType="functionBlock">  
2   <interface>  
3     <inputVars>  
4       <variable name="encoderErrorSignal" address="%I*">  
5         <type><BOOL /></type>  
6         <addData>
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

The screenshot shows the OntoManager @ Mercator Telescope web application. The browser address bar displays `localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel`. The application has a navigation bar with tabs: Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. On the left, a tree view shows the project structure, with `mtcs_cover` selected. The main content area displays the `FunctionBlock SM_CoverPanel`. It includes a diagram of the function block with inputs/outputs and a table of variables.

FunctionBlock SM_CoverPanel

Diagram of the function block:

- Inputs/Outputs:
 - `encoderErrorSignal` (input)
 - `actualStatus` (output)
 - `initializationStatus` (input/output)
 - `statuses` (output)
 - `operatorStatus` (input/output)
 - `parts` (output)
 - `operatingStatus` (input/output)
 - `processes` (output)
 - `config` (input)
 - `coverConfig` (input)
- Methods:
 - `startOpening()`
 - `startClosing()`

Variables

Variable	Name	Type	Initial value	Address	Description	Qualif
VAR_INPUT	<code>encoderErrorSignal</code>	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	<code>initializationStatus</code>	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	<code>operatorStatus</code>	OperatorStatus			TECH or OBSERVER or ...	
	<code>operatingStatus</code>	OperatingStatus			MANUAL or AUTO or NONE	
	<code>config</code>	CoverPanelConfig			Configuration of the panel	
	<code>coverConfig</code>	CoverConfig			Configuration of the cover	
VAR_OUTPUT	<code>actualStatus</code>	STRING			Current status description	OPC.UA.DA=1, OPC

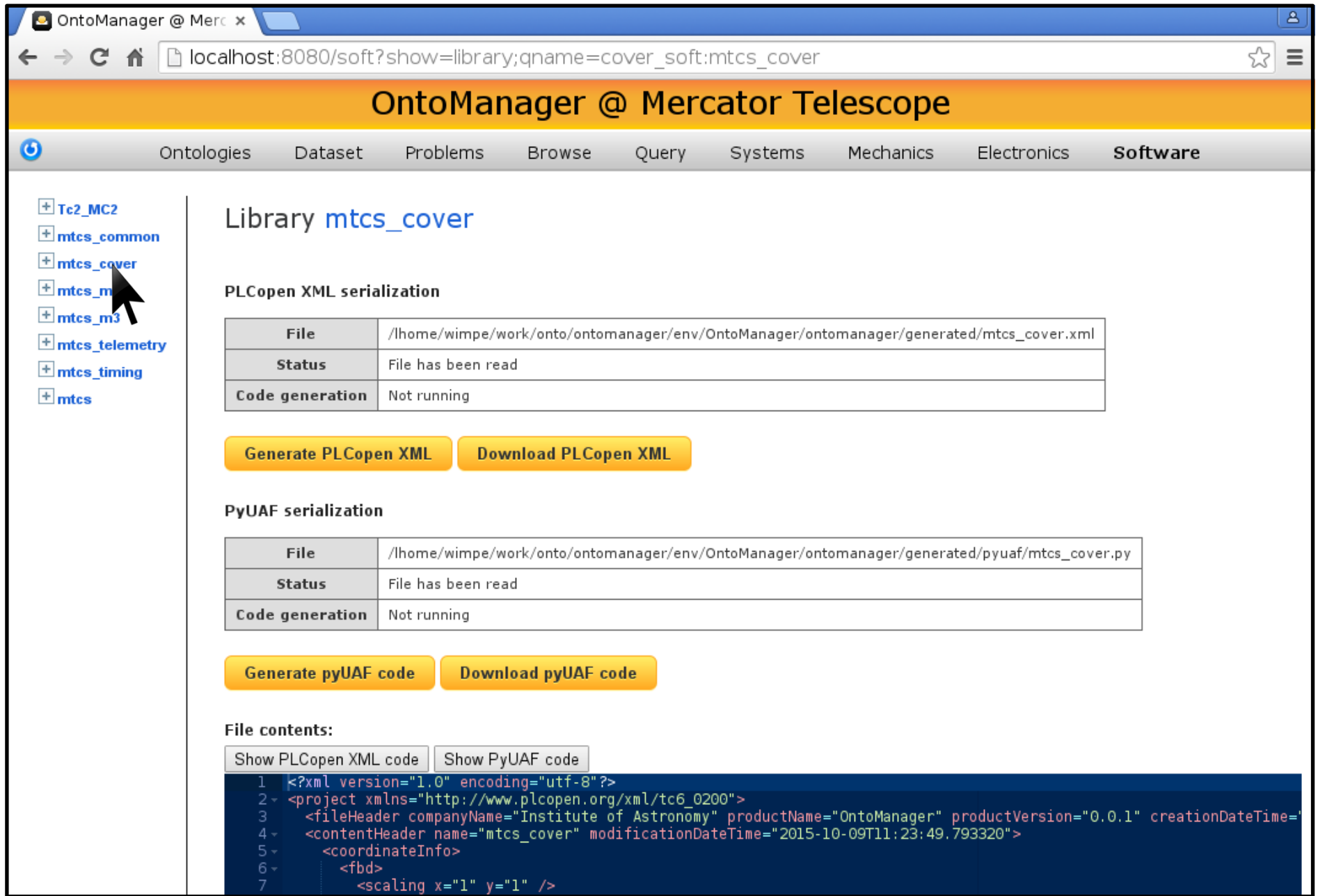
Jump to:

- Variables
- Methods
- Implementation
- PLCopen XML serialization

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design



The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar shows the URL `localhost:8080/soft?show=library;qname=cover_soft:mtcs_cover`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. The left sidebar shows a tree of ontologies, with `mtcs_cover` selected. The main content area displays the "Library mtcs_cover" page. It includes a table for "PLCopen XML serialization" and a table for "PyUAF serialization". Both tables show the file path, status, and code generation status. Below each table are buttons for "Generate" and "Download". At the bottom, there is a "File contents:" section with tabs for "Show PLCopen XML code" and "Show PyUAF code". The "Show PLCopen XML code" tab is active, displaying XML code for the mtcs_cover library.

OntoManager @ Mercator Telescope

Library `mtcs_cover`

PLCopen XML serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml	File has been read	Not running

Generate PLCopen XML Download PLCopen XML

PyUAF serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py	File has been read	Not running

Generate pyUAF code Download pyUAF code

File contents:

Show PLCopen XML code Show PyUAF code

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://www.plcopen.org/xml/tc6_0200">
  <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
  <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
    <coordinateInfo>
      <fbid>
        <scaling x="1" y="1" />
      </fbid>
    </coordinateInfo>
  </contentHeader>
</project>
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics Electronics Software

Library **mtcs_cover**

PLCopen XML serialization

File	/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml
Status	File has been read
Code generation	Not running

Generate PLCopen XML Download PLCopen XML

PyUAF serialization

File	/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py
Status	File has been read
Code generation	Not running

Generate pyUAF code Download pyUAF code

File contents:

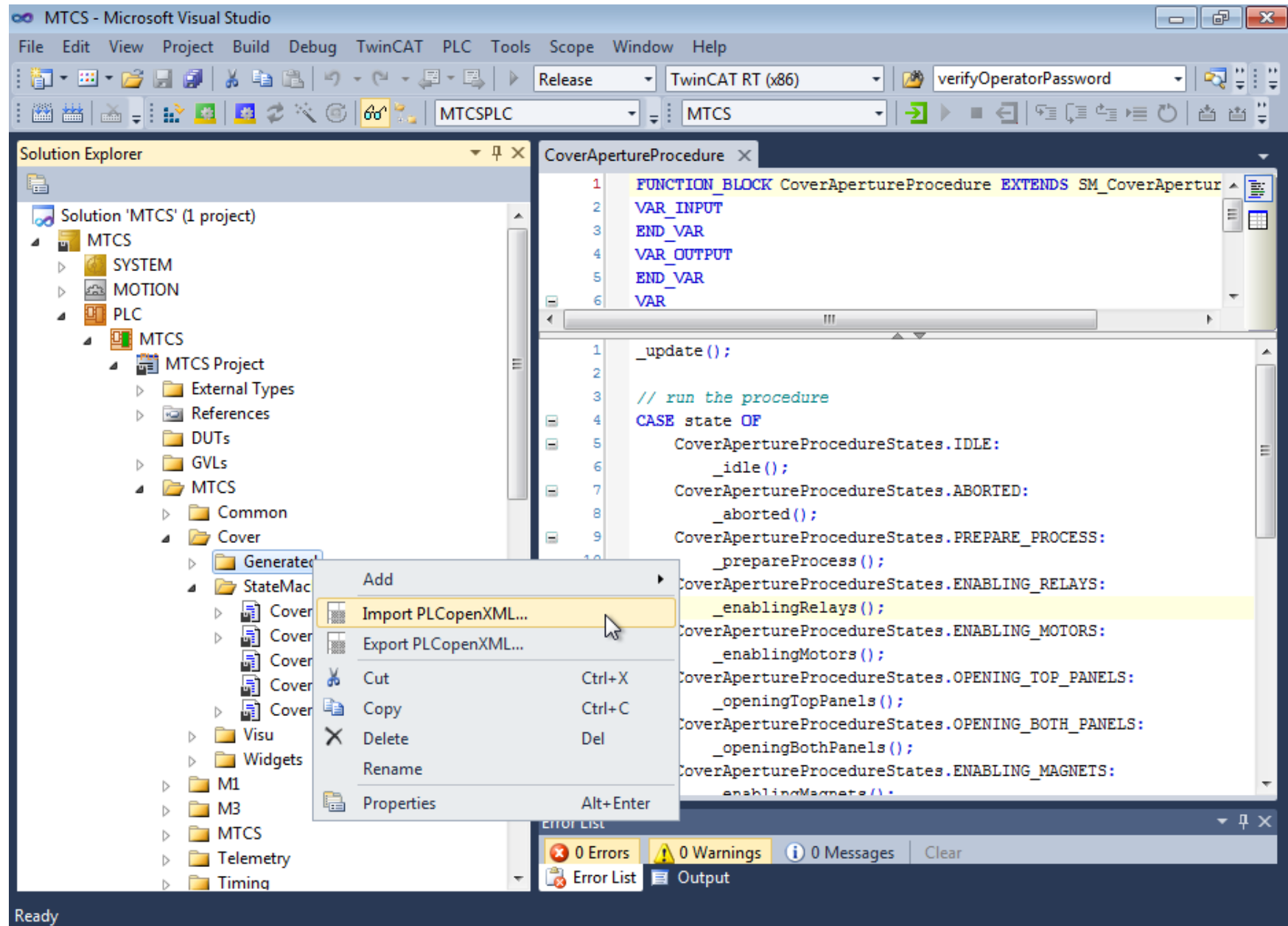
Show PLCopen XML code Show PyUAF code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <project xmlns="http://www.plcopen.org/xml/tc6_0200">
3   <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
4   <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
5     <coordinateInfo>
6       <fbid>
7         <scaling x="1" y="1" />
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

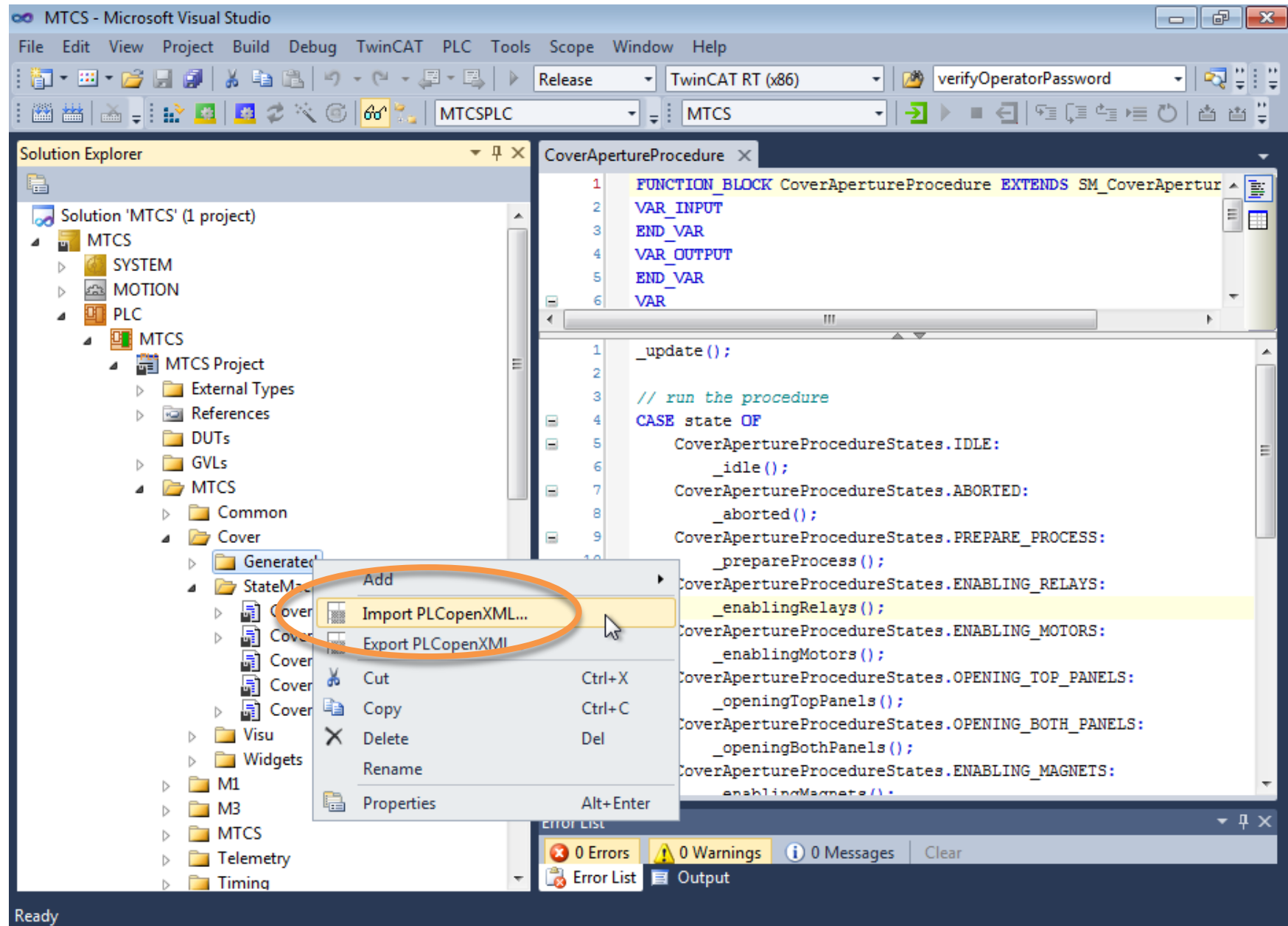
Software design



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

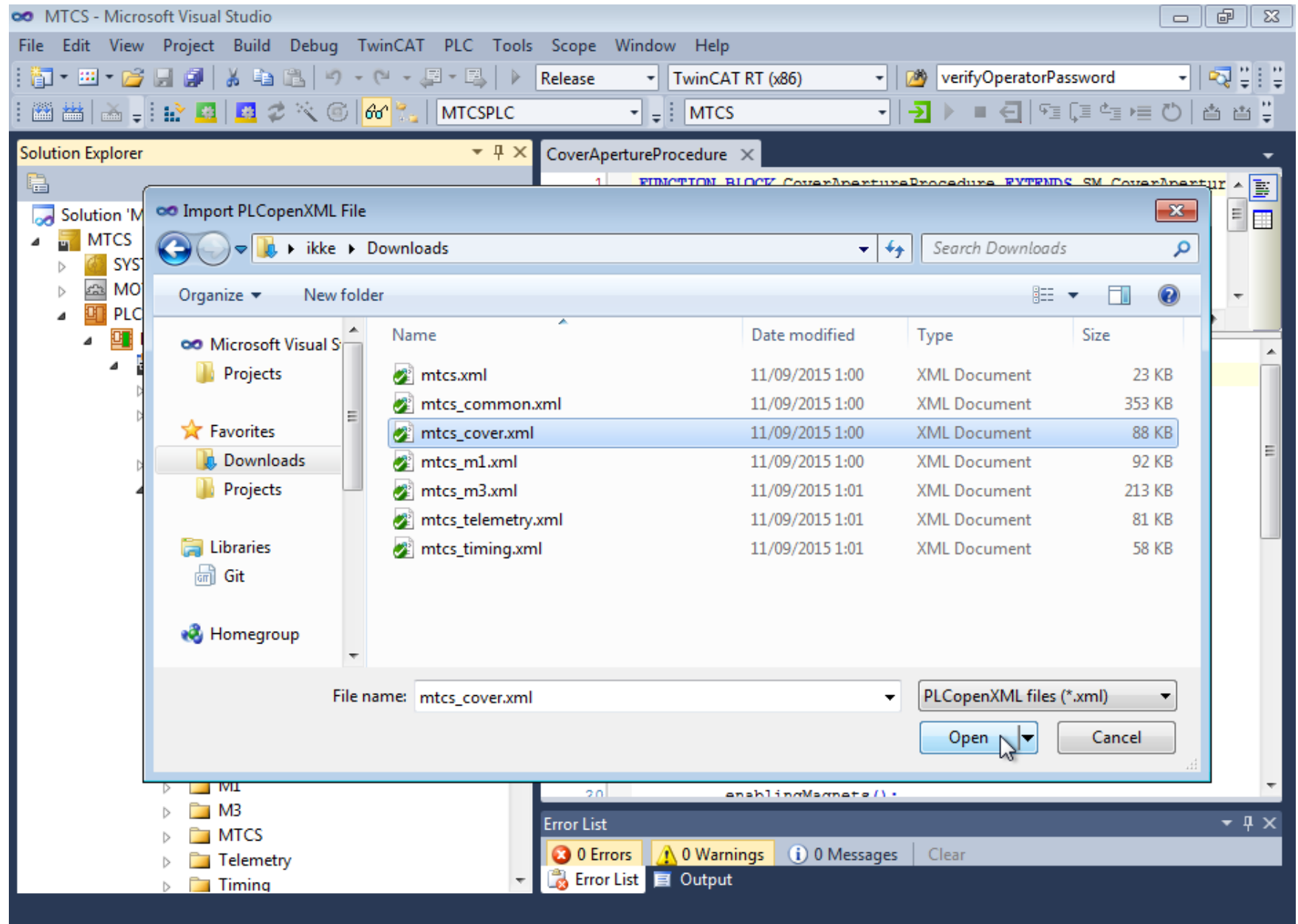
Software design



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

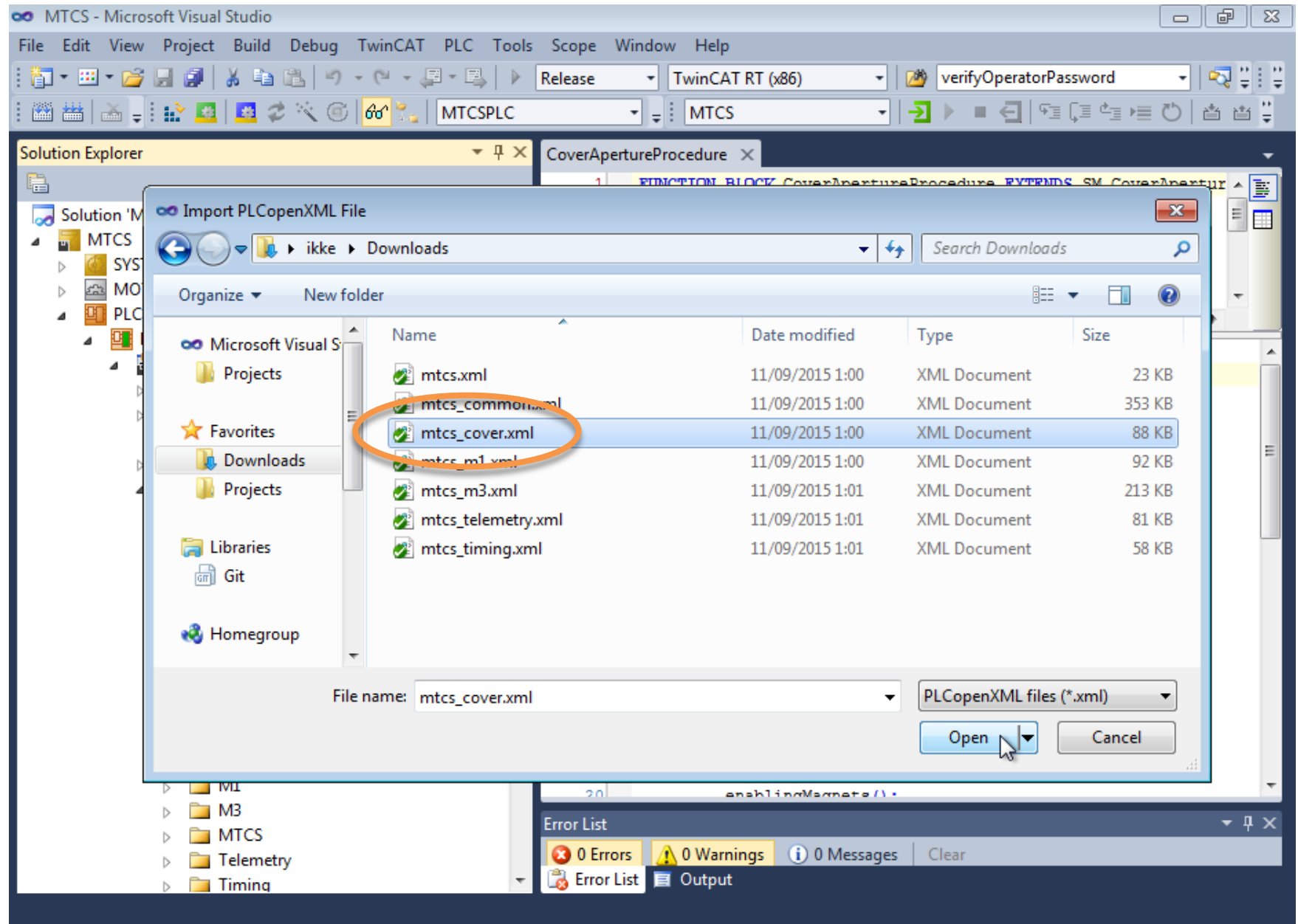
Software design



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

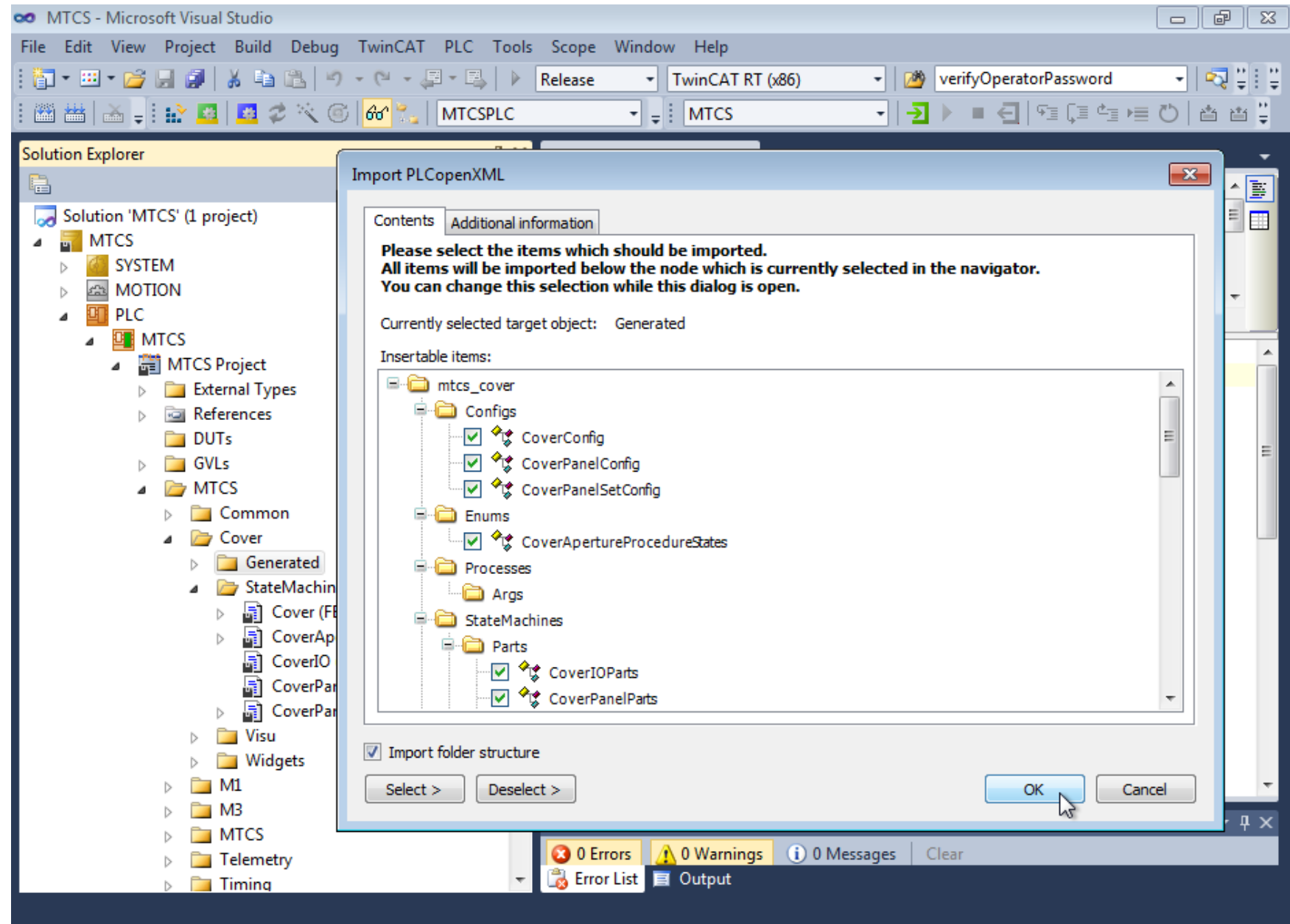
Software design

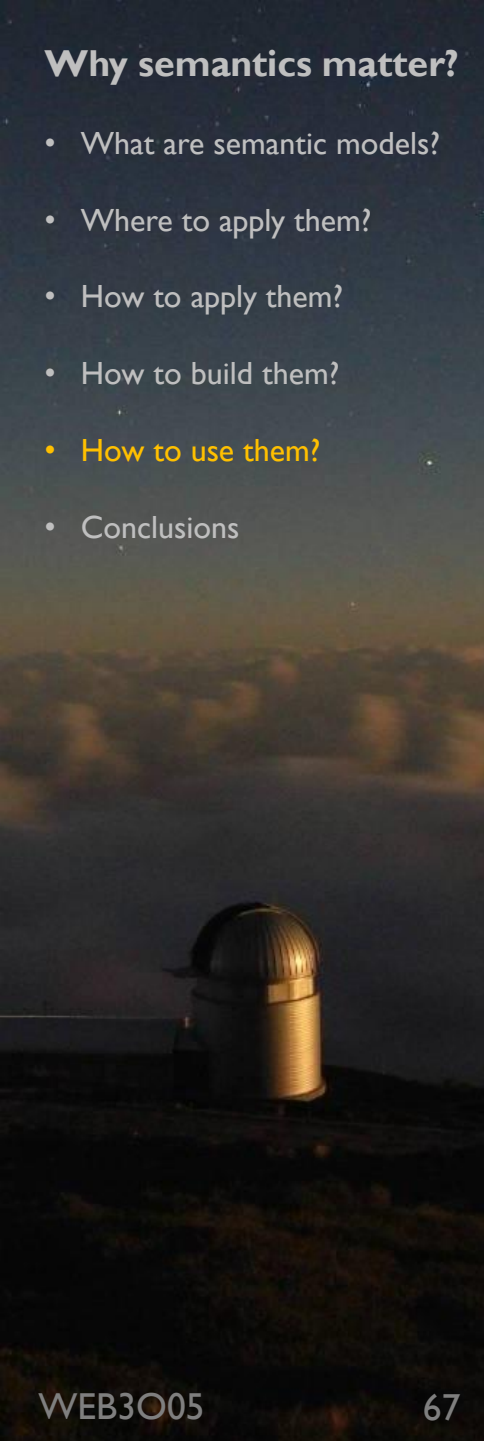


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Software design



A photograph of a telescope dome on a hill, illuminated by the warm light of a low sun, creating a silhouette effect. The sky is filled with soft, golden clouds. The overall mood is contemplative and serene.

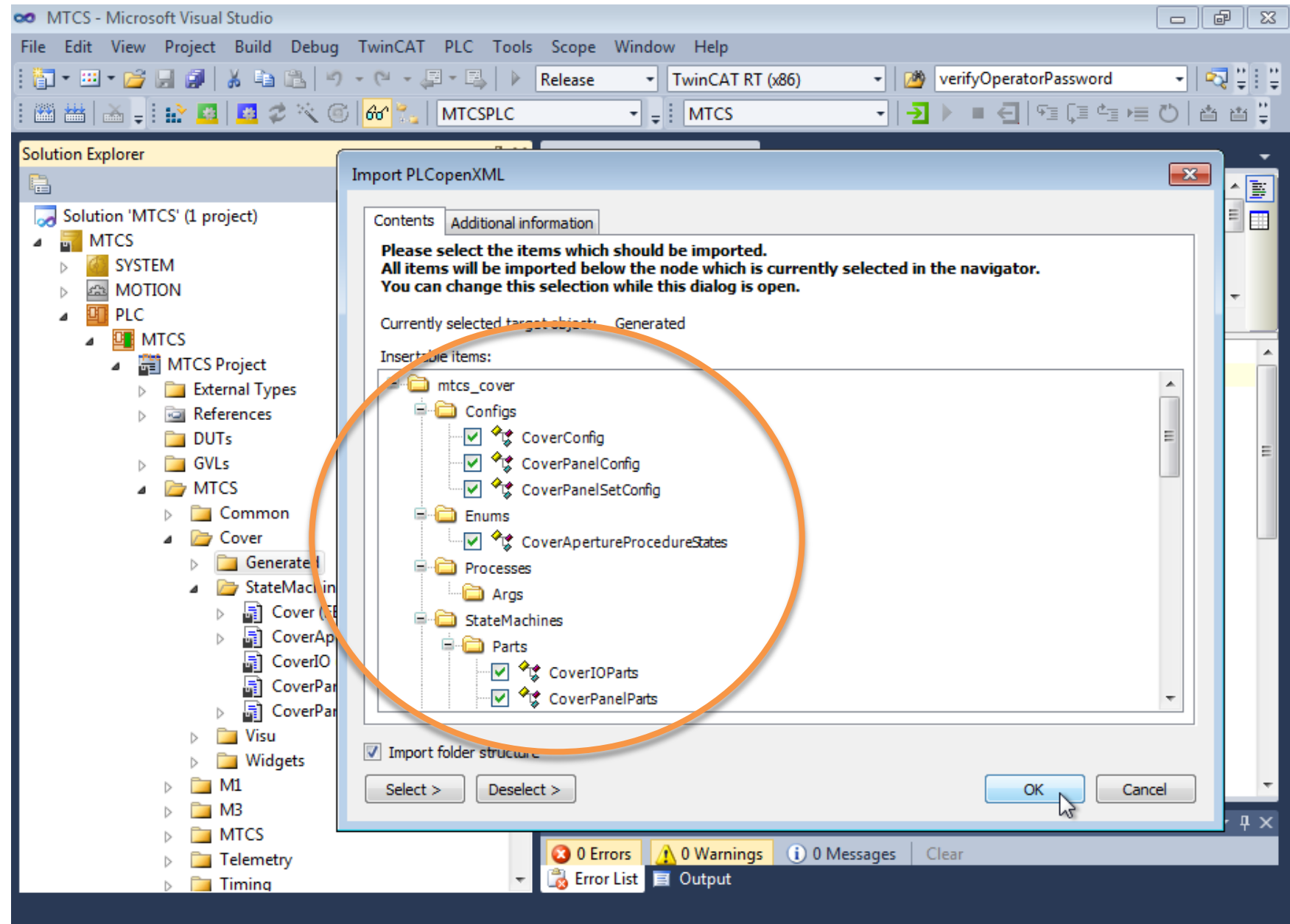
Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

WEB3O05 67

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics Electronics **Software**

Library mtcs_cover

PLCopen XML serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml	File has been read	Not running

[Generate PLCopen XML](#) [Download PLCopen XML](#)

PyUAF serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py	File has been read	Not running

[Generate pyUAF code](#) [Download pyUAF code](#)

File contents:

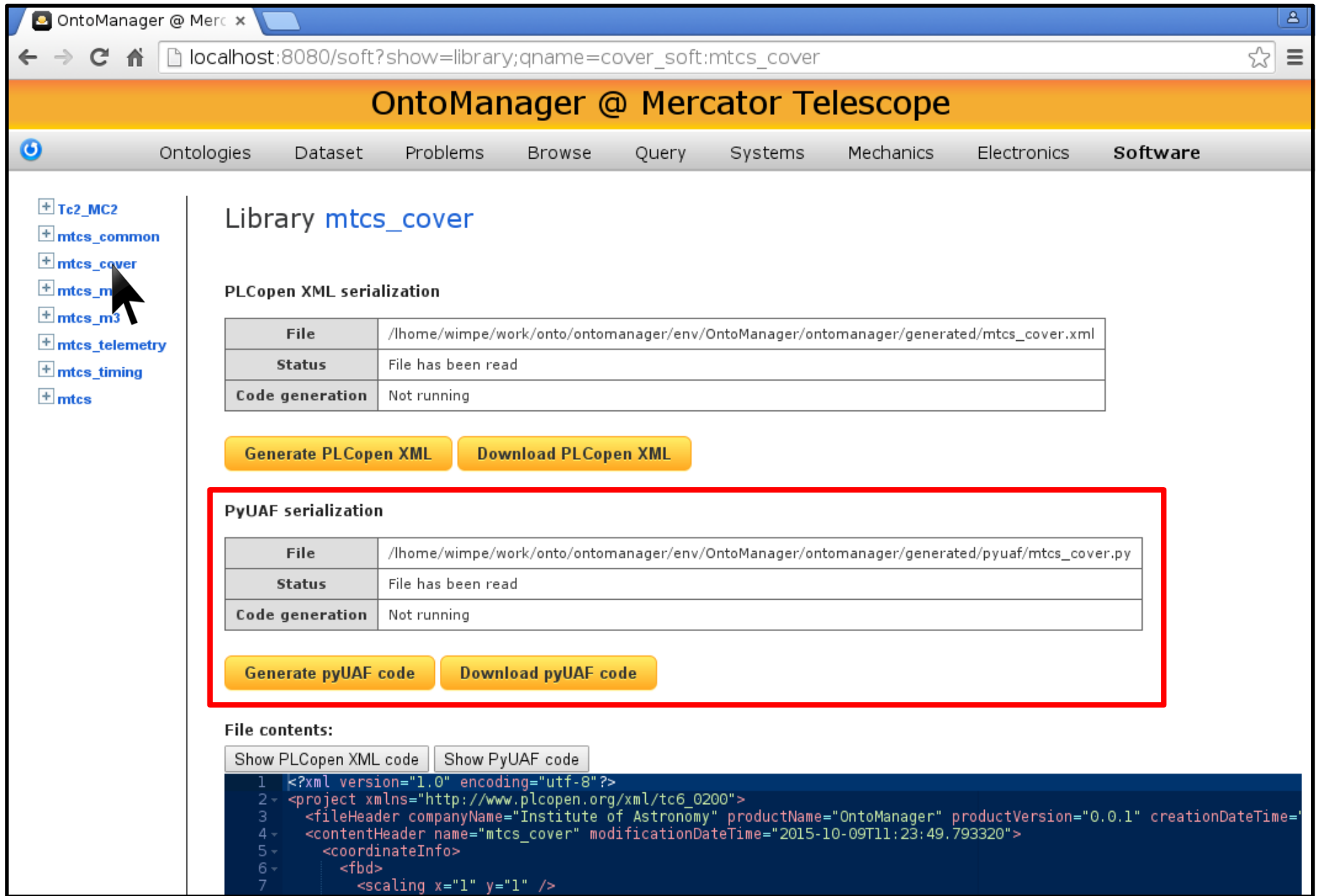
Show PLCopen XML code Show PyUAF code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <project xmlns="http://www.plcopen.org/xml/tc6_0200">
3   <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
4   <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
5     <coordinateInfo>
6       <fbid>
7         <scaling x="1" y="1" />
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design



The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar shows the URL `localhost:8080/soft?show=library;qname=cover_soft:mtcs_cover`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. The left sidebar shows a tree of ontologies, with `mtcs_cover` selected. The main content area displays the "Library mtcs_cover" page. It includes a table for "PLCopen XML serialization" and a table for "PyUAF serialization". Both tables show the file path, status, and code generation status. Below the tables are buttons for "Generate PLCopen XML", "Download PLCopen XML", "Generate pyUAF code", and "Download pyUAF code". The PyUAF section is highlighted with a red border. At the bottom, there is a "File contents:" section with buttons for "Show PLCopen XML code" and "Show PyUAF code". The PyUAF code is displayed in a dark blue box.

OntoManager @ Mercator Telescope

Library `mtcs_cover`

PLCopen XML serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml	File has been read	Not running

Generate PLCopen XML Download PLCopen XML

PyUAF serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py	File has been read	Not running

Generate pyUAF code Download pyUAF code

File contents:

Show PLCopen XML code Show PyUAF code

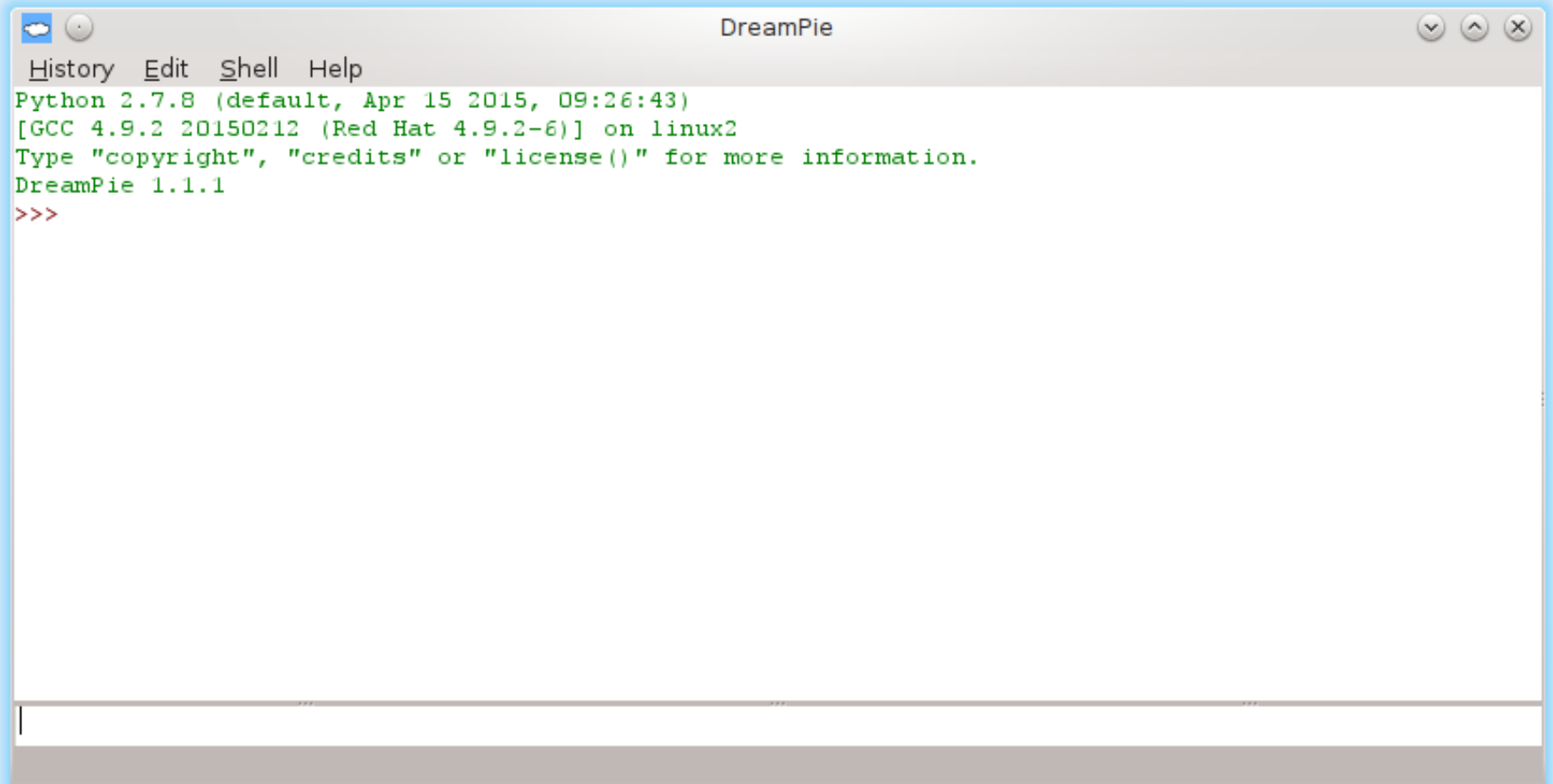
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <project xmlns="http://www.plcopen.org/xml/tc6_0200">
3   <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
4   <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
5     <coordinateInfo>
6       <fbid>
7         <scaling x="1" y="1" />
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



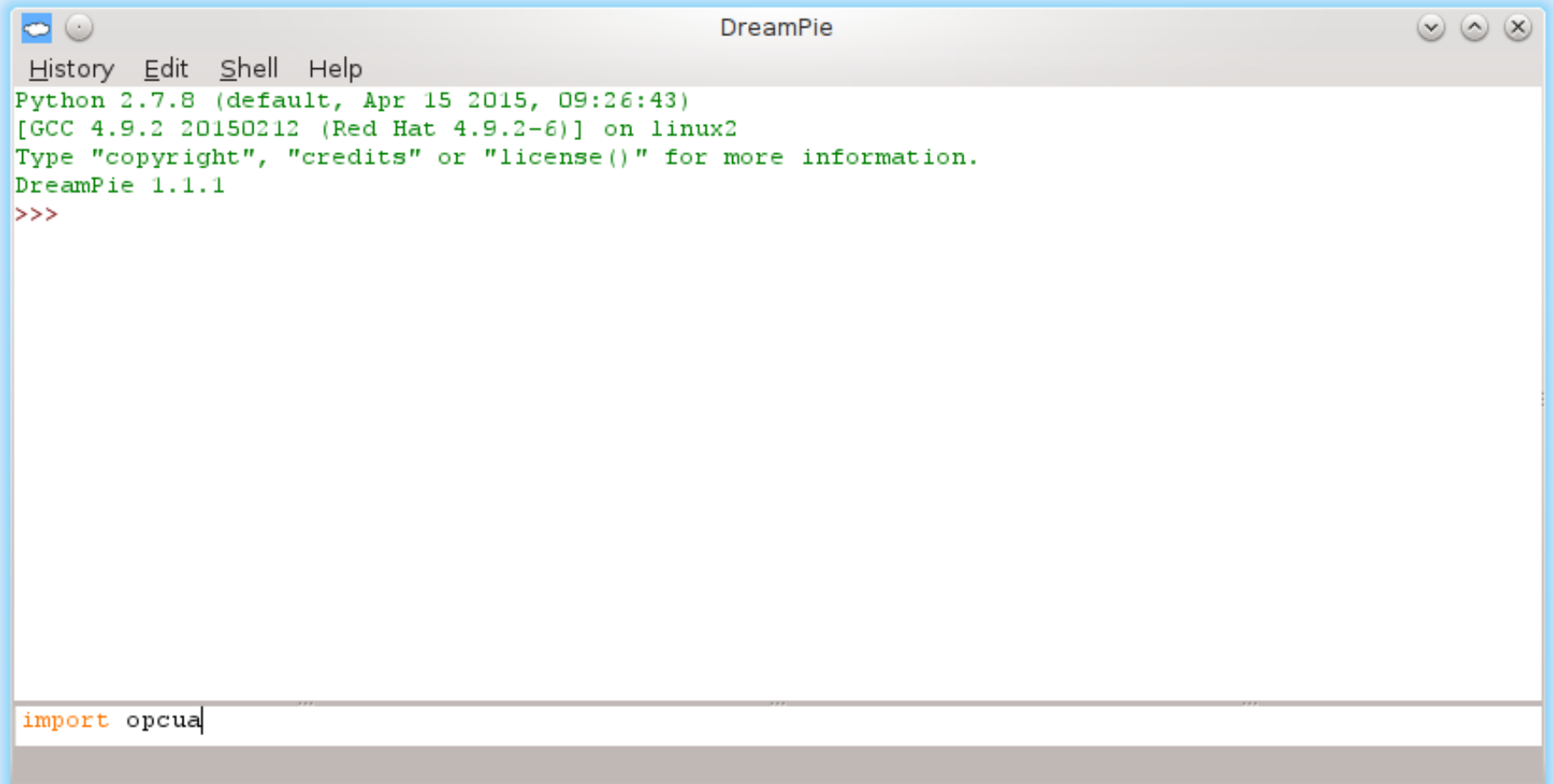
The screenshot shows a window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The main area displays the following text in green: "Python 2.7.8 (default, Apr 15 2015, 09:26:43)", "[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2", "Type \"copyright\", \"credits\" or \"license()\" for more information.", and "DreamPie 1.1.1". At the bottom, there is a red prompt ">>>" indicating a ready state for input.

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>>

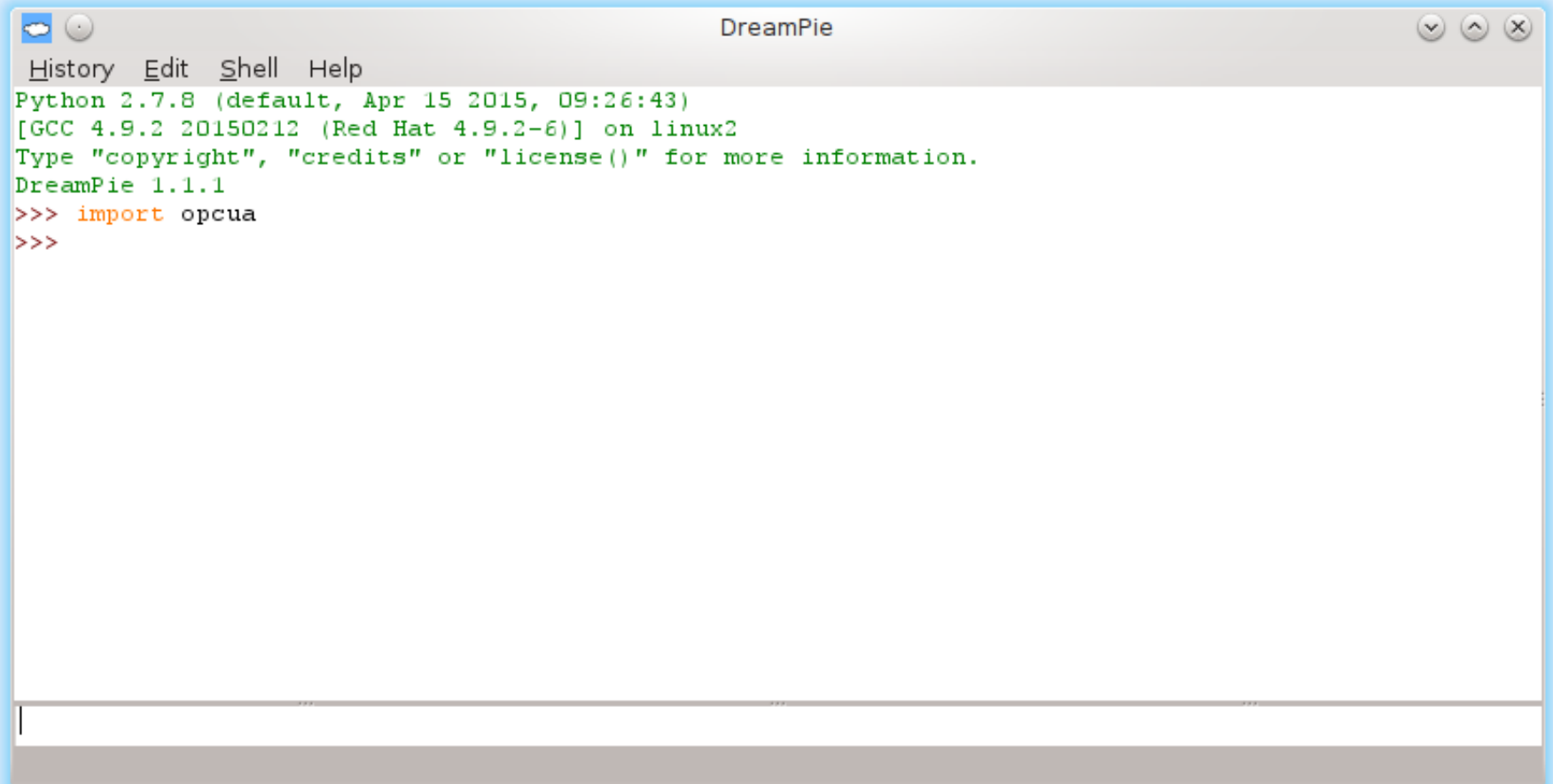
import opcua|
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



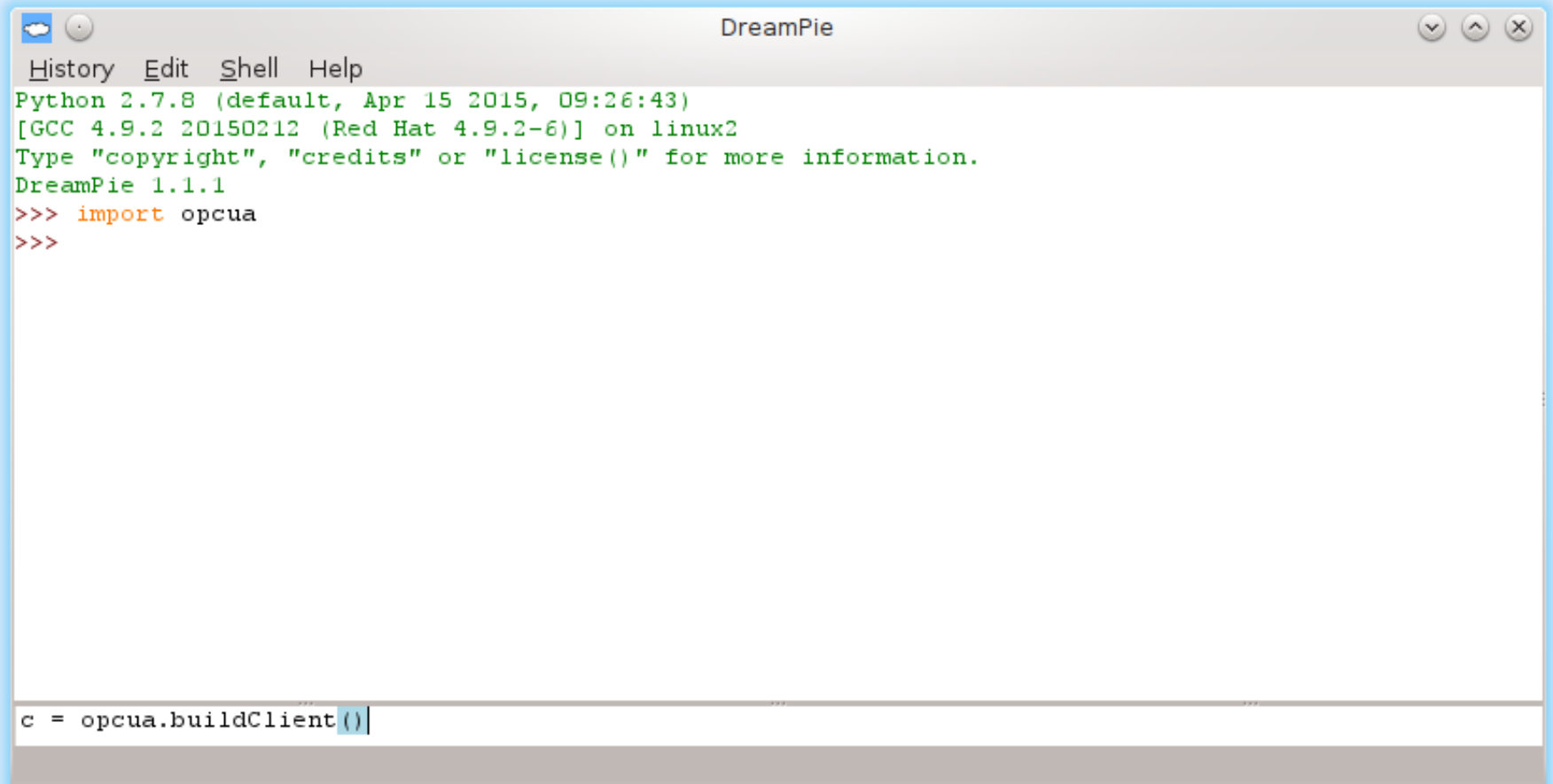
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>>
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output displays the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the command `>>> import opcua` and is now at the prompt `>>>`. The bottom of the window shows the command `c = opcua.buildClient()` being typed.

```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>>

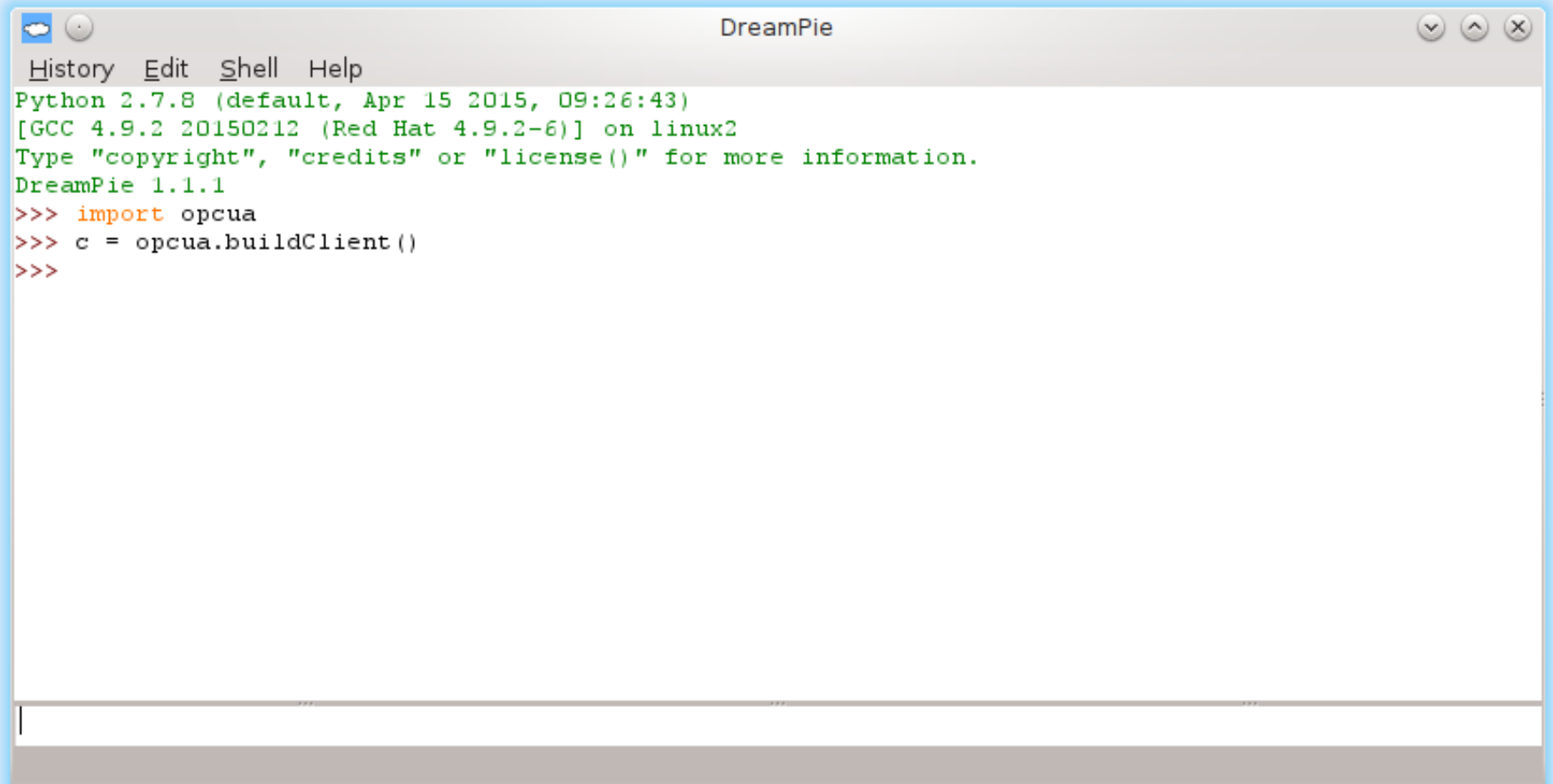
c = opcua.buildClient()
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>

A screenshot of a terminal window titled "DreamPie". The window has a menu bar with "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered three lines of Python code: "import opcu", "c = opcu.buildClient()", and an empty line. The terminal is set against a dark background with a small, glowing, dome-shaped object in the bottom left corner.

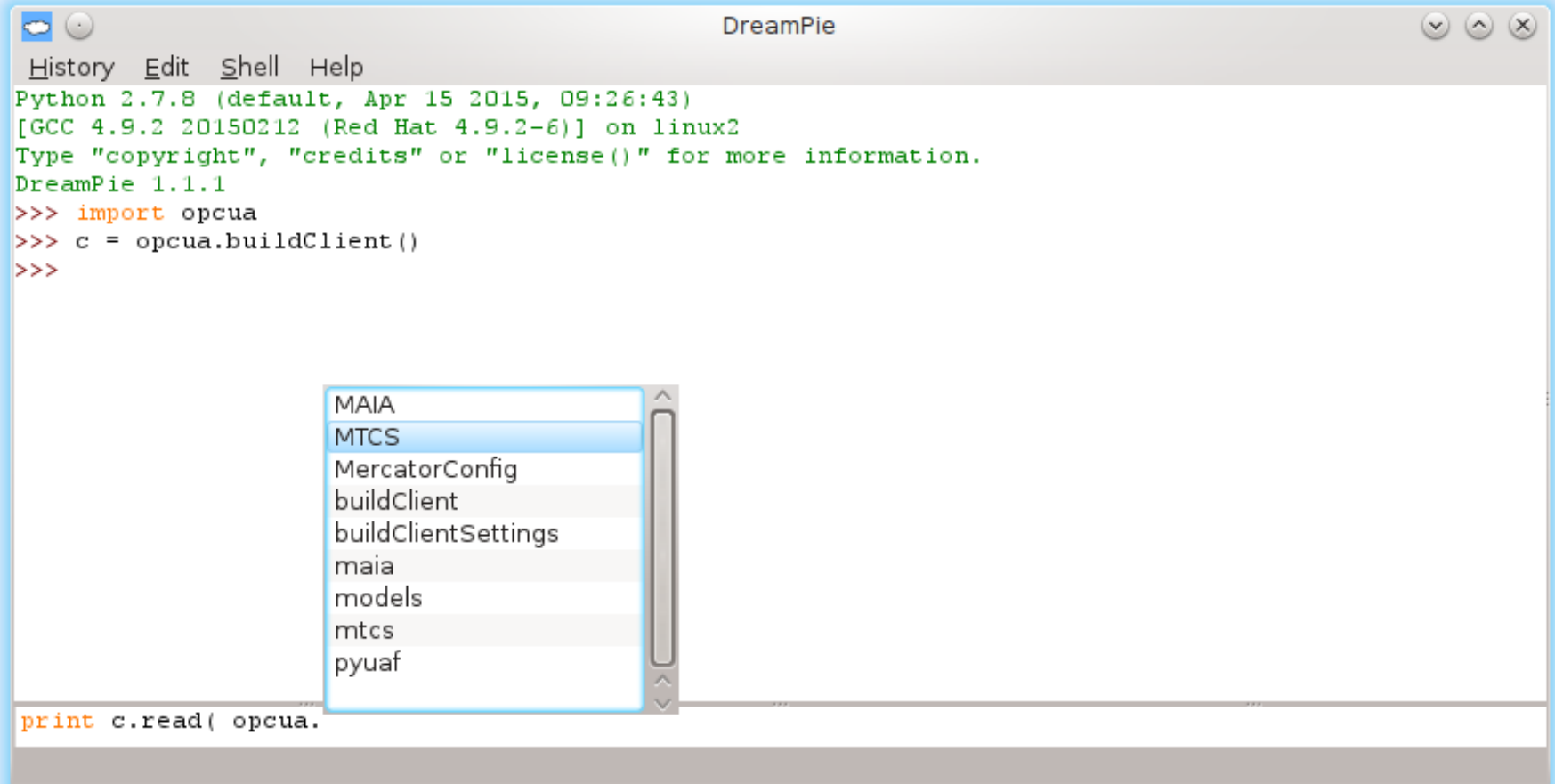
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcu
>>> c = opcu.buildClient()
>>>
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

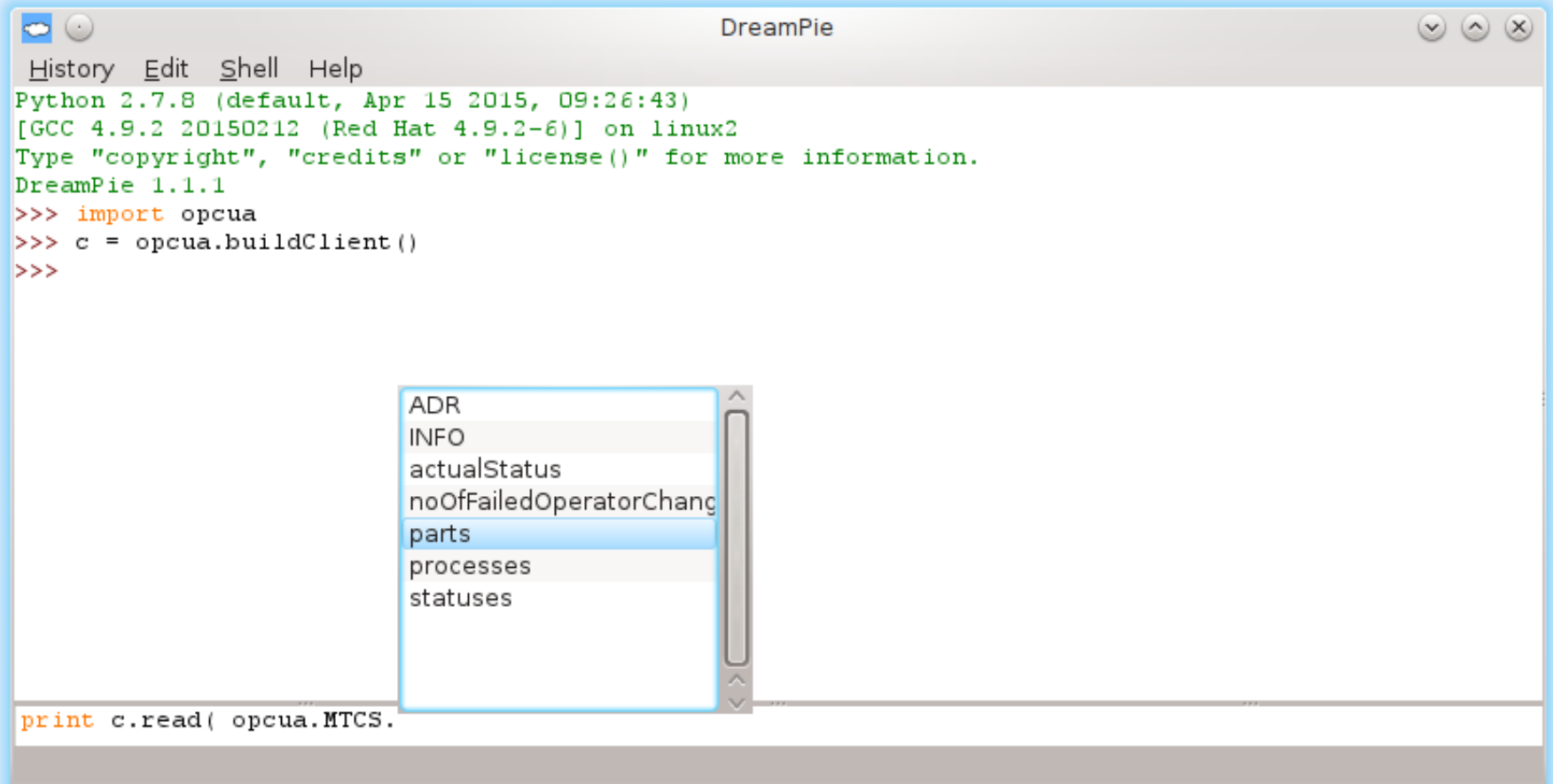
A list of attributes for the 'opcua' module is displayed, including MAIA, MTCS, MercatorConfig, buildClient, buildClientSettings, maia, models, mtcs, and pyuaf. The list is scrollable, and the current selection is 'MAIA'. Below the list, the code `print c.read(opcua.` is visible.

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, showing a list of attributes: ADR, INFO, actualStatus, noOfFailedOperatorChang, parts (highlighted), processes, and statuses. The code at the bottom of the terminal is:

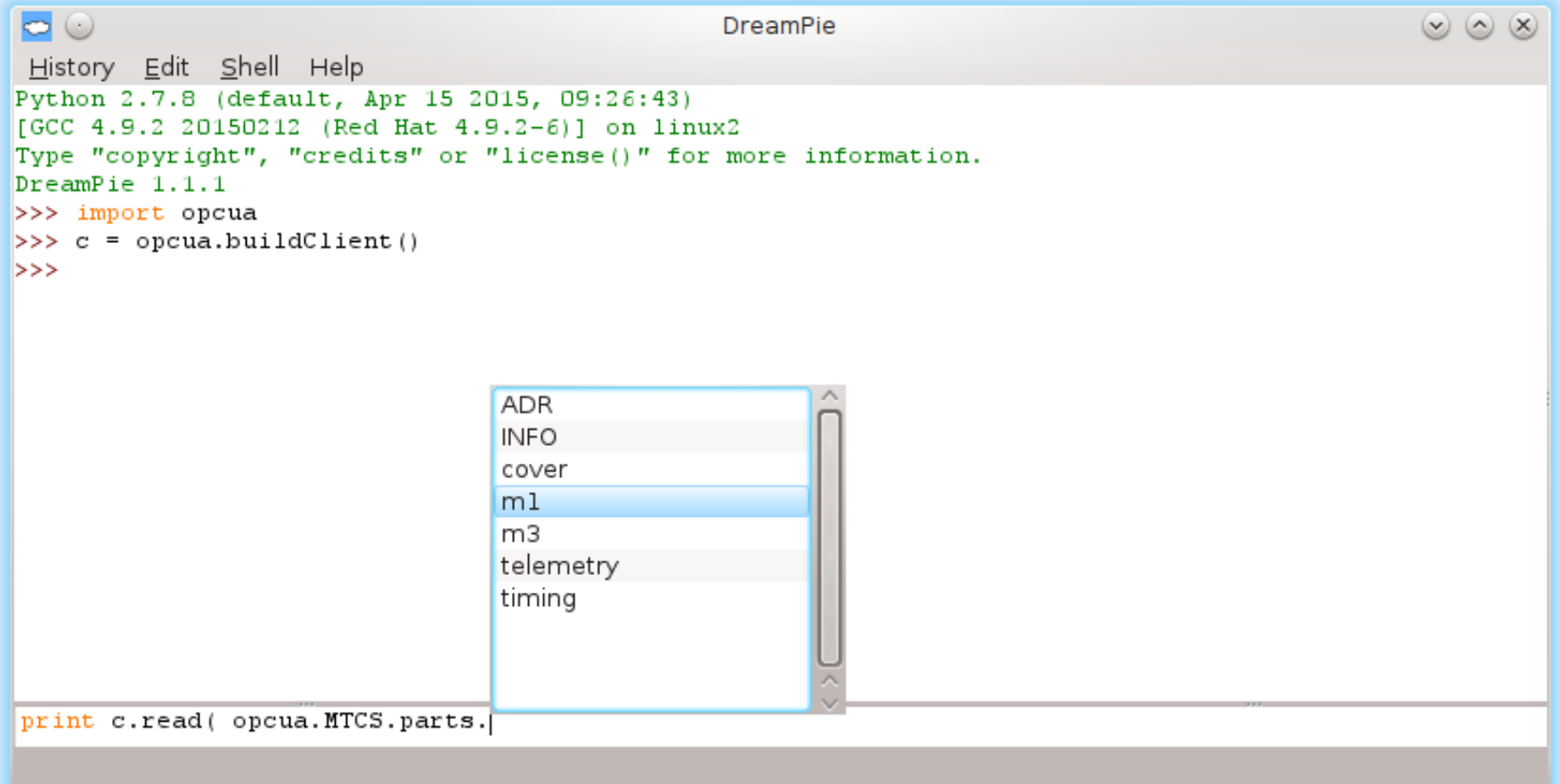
```
print c.read( opcua.MTCS.
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following Python code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, showing a list of items: "ADR", "INFO", "cover", "m1", "m3", "telemetry", and "timing". The item "m1" is currently selected. At the bottom of the terminal, the following code is partially visible:

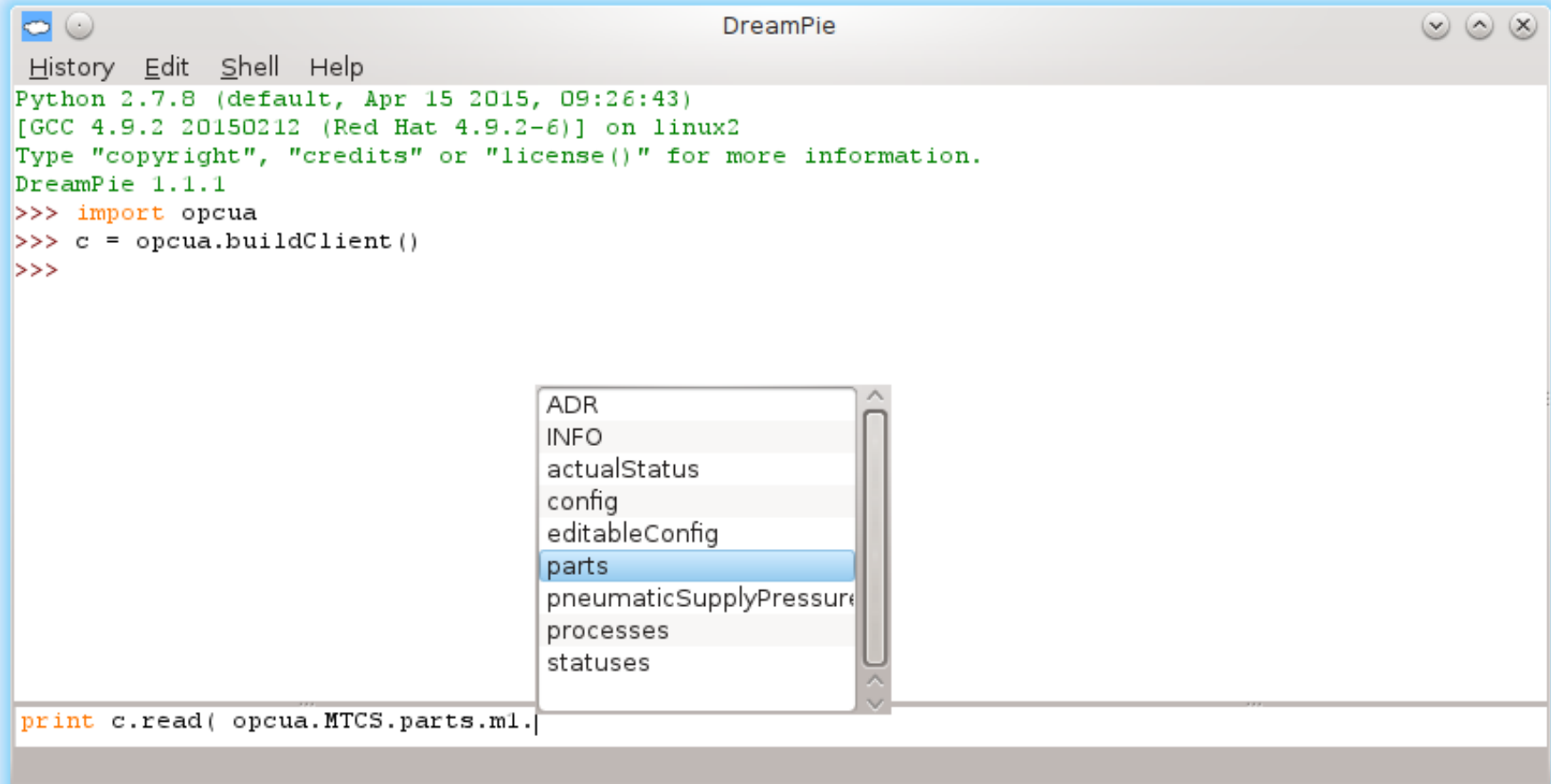
```
print c.read( opcua.MTCS.parts,
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python environment (Python 2.7.8, GCC 4.9.2) and the execution of the following code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, displaying a list of attributes: ADR, INFO, actualStatus, config, editableConfig, **parts** (highlighted), pneumaticSupplyPressure, processes, and statuses.

At the bottom of the terminal, the following code is being typed:

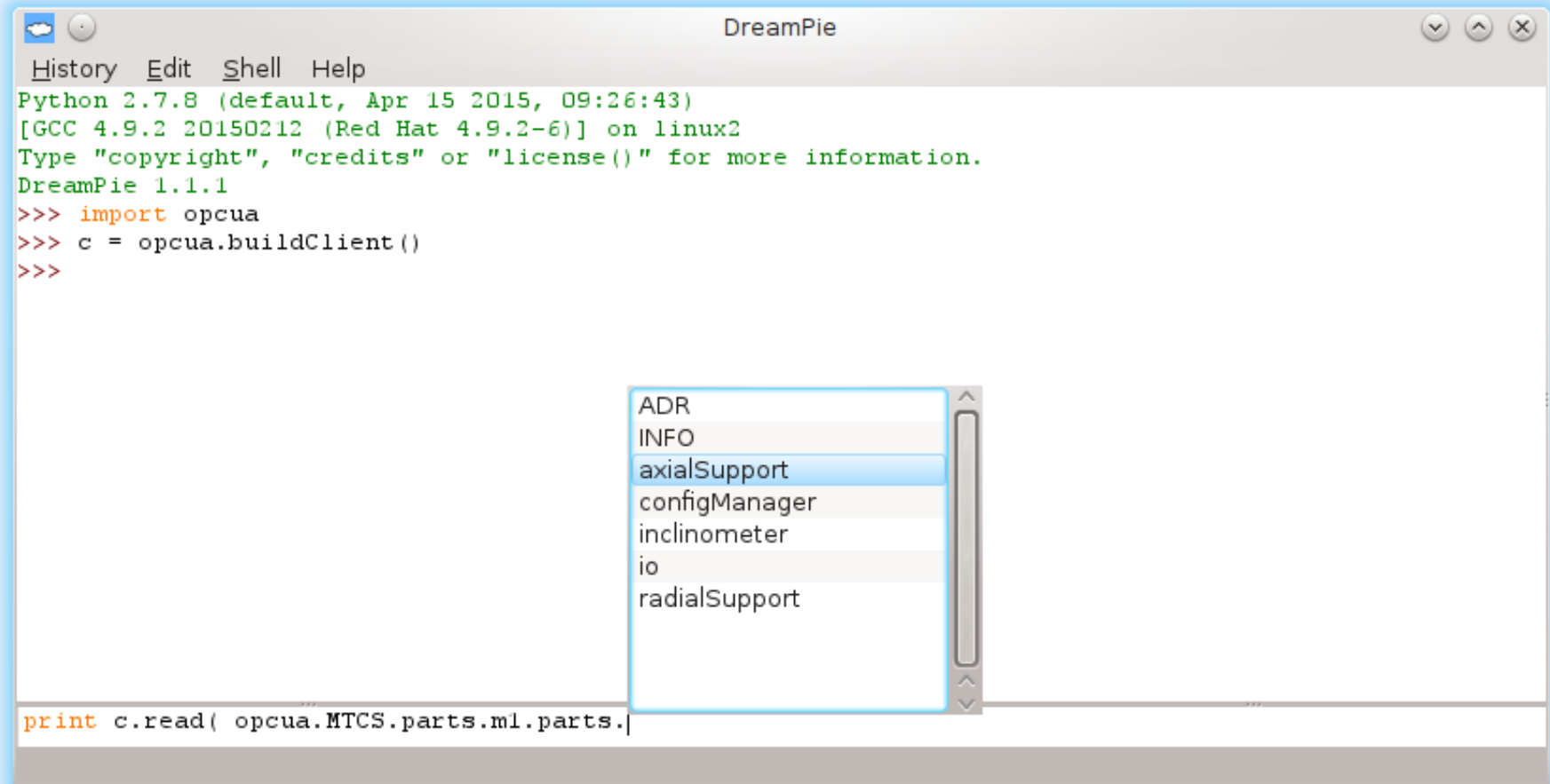
```
print c.read( opcua.MTCS.parts.ml.
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a window titled "DreamPie" with a menu bar (History, Edit, Shell, Help) and a terminal-like interface. The terminal displays the following text:

```
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, showing a list of items: ADR, INFO, axialSupport (highlighted), configManager, inclinometer, io, and radialSupport. At the bottom of the window, the following code is partially visible:

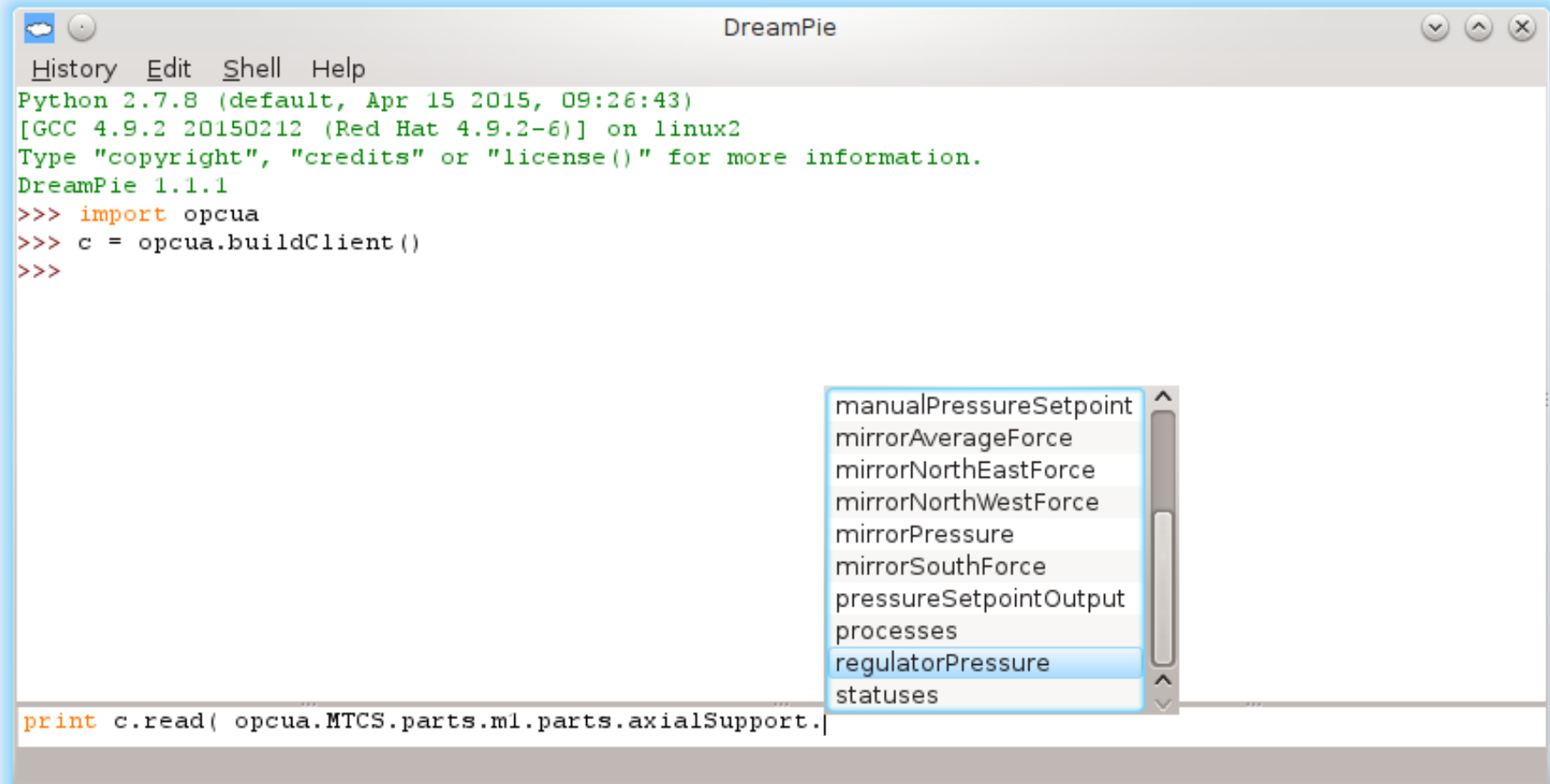
```
print c.read( opcua.MTCS.parts.ml.parts.)
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

manualPressureSetpoint
mirrorAverageForce
mirrorNorthEastForce
mirrorNorthWestForce
mirrorPressure
mirrorSouthForce
pressureSetpointOutput
processes
regulatorPressure
statuses

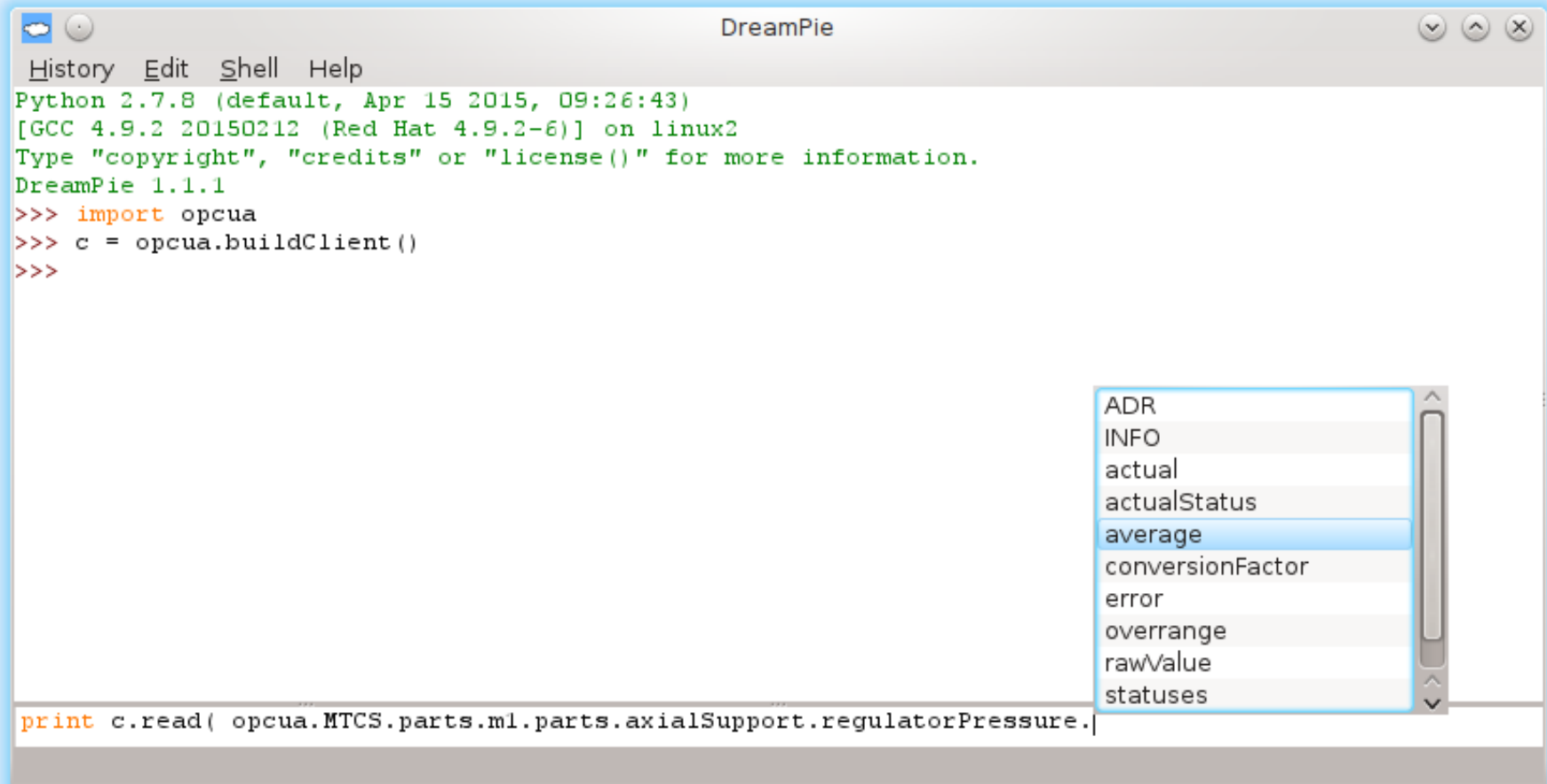
print c.read( opcua.MTCS.parts.ml.parts.axialSupport.
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following Python code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open on the right side of the terminal, showing a list of attributes: ADR, INFO, actual, actualStatus, **average** (highlighted), conversionFactor, error, overrange, rawValue, and statuses. At the bottom of the terminal, the following code is visible:

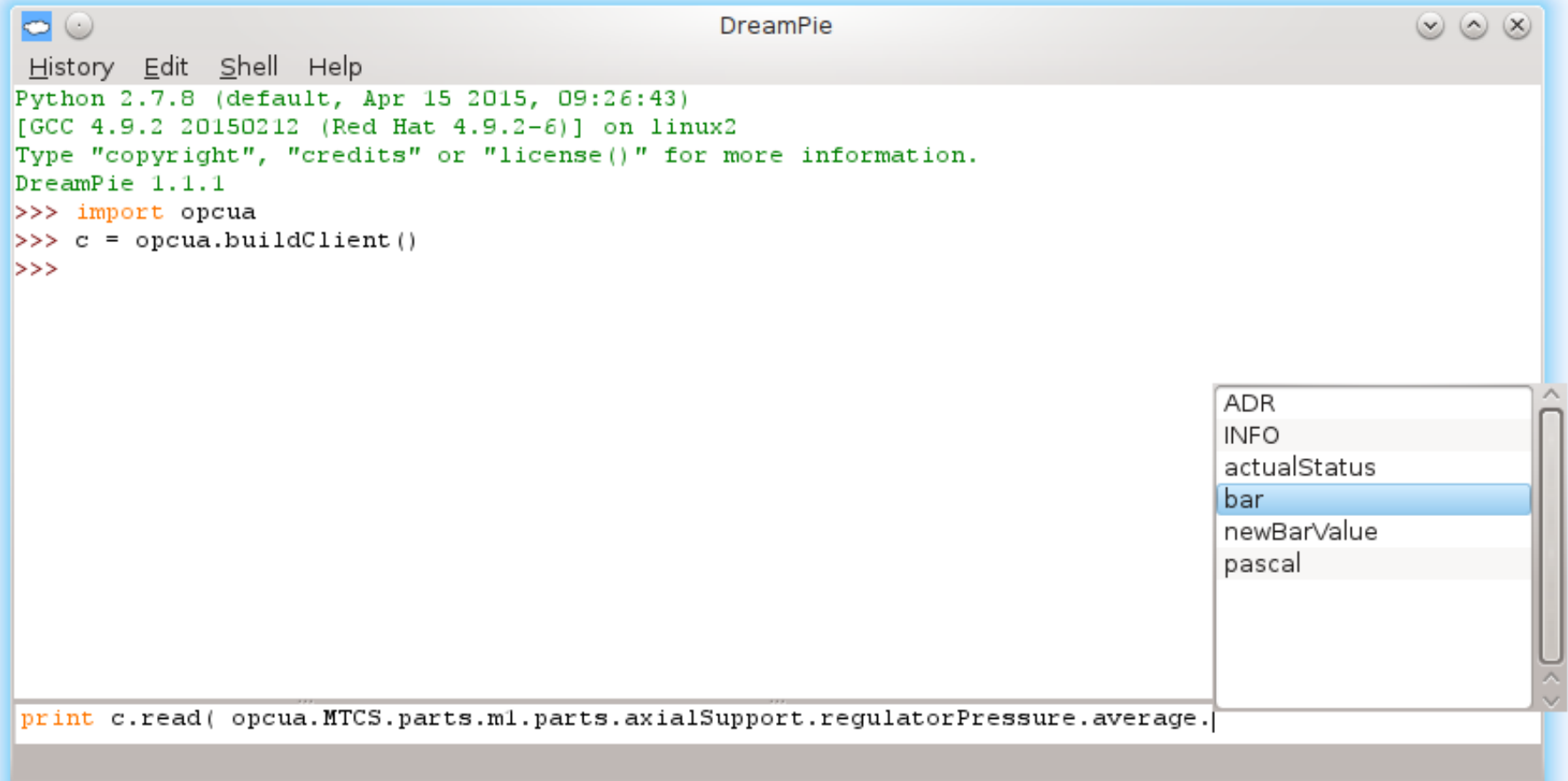
```
print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.)
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

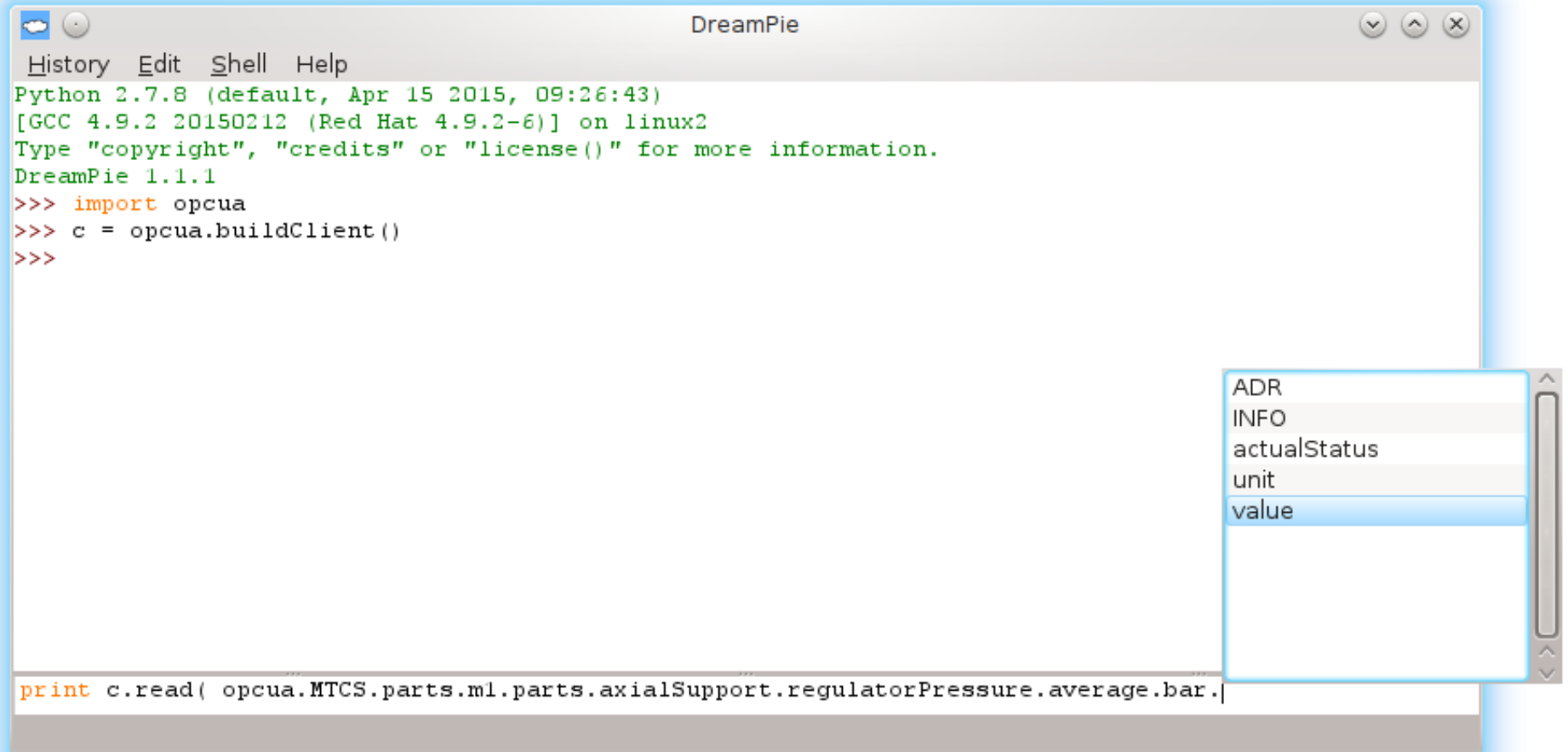
print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.average,
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.average.bar.
```

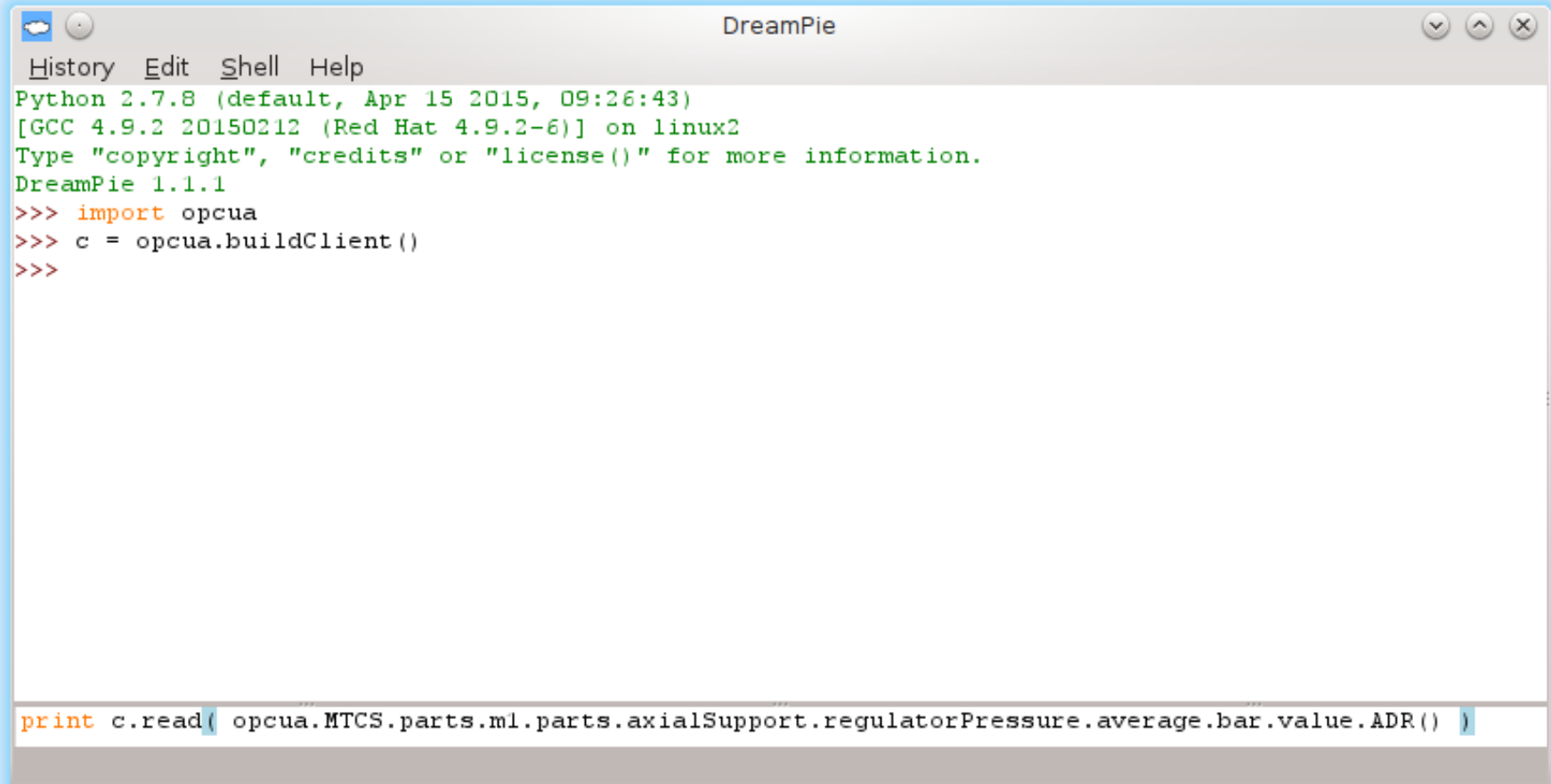
ADR
INFO
actualStatus
unit
value

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

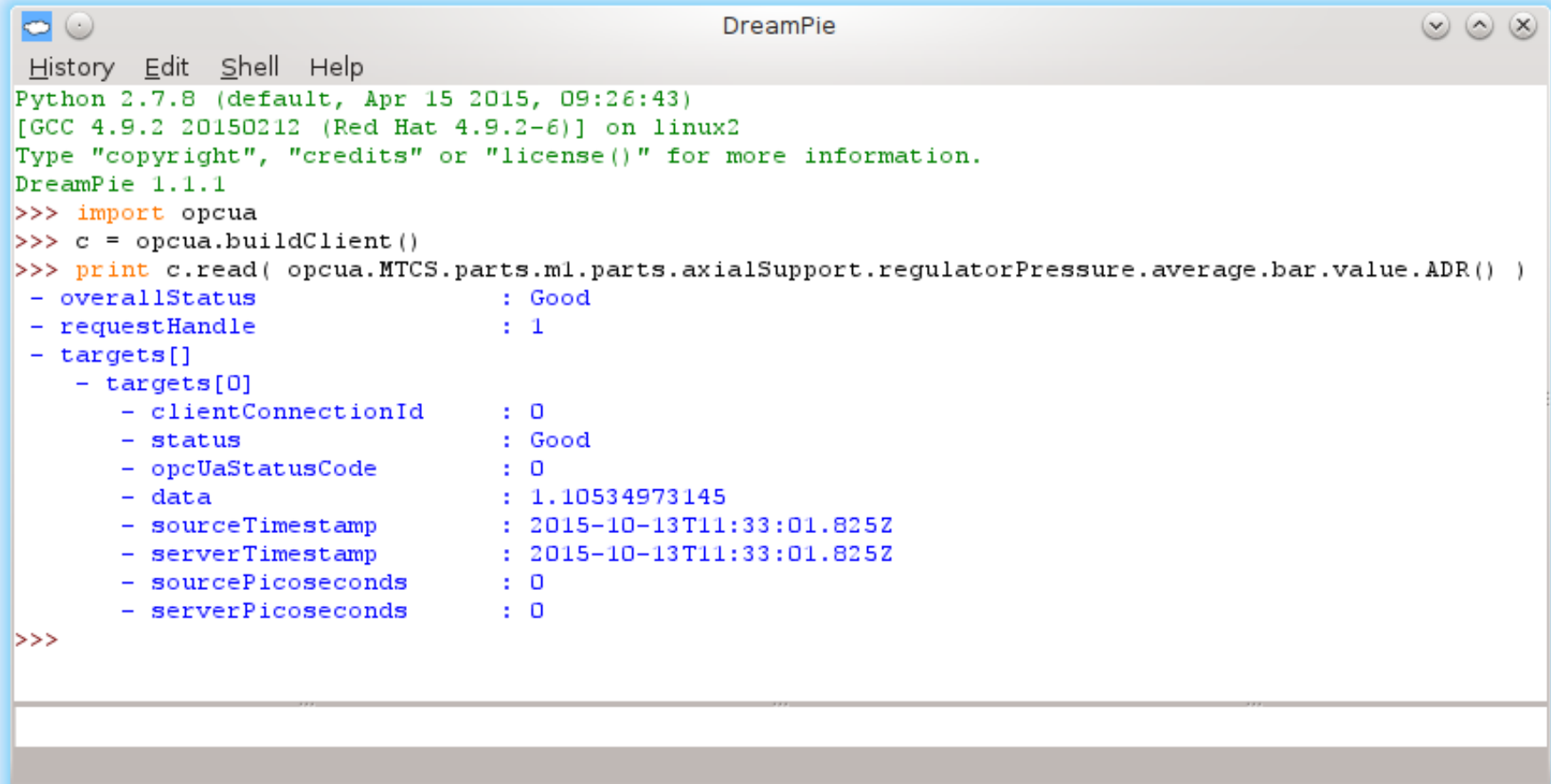
print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.average.bar.value.ADR() )
```


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



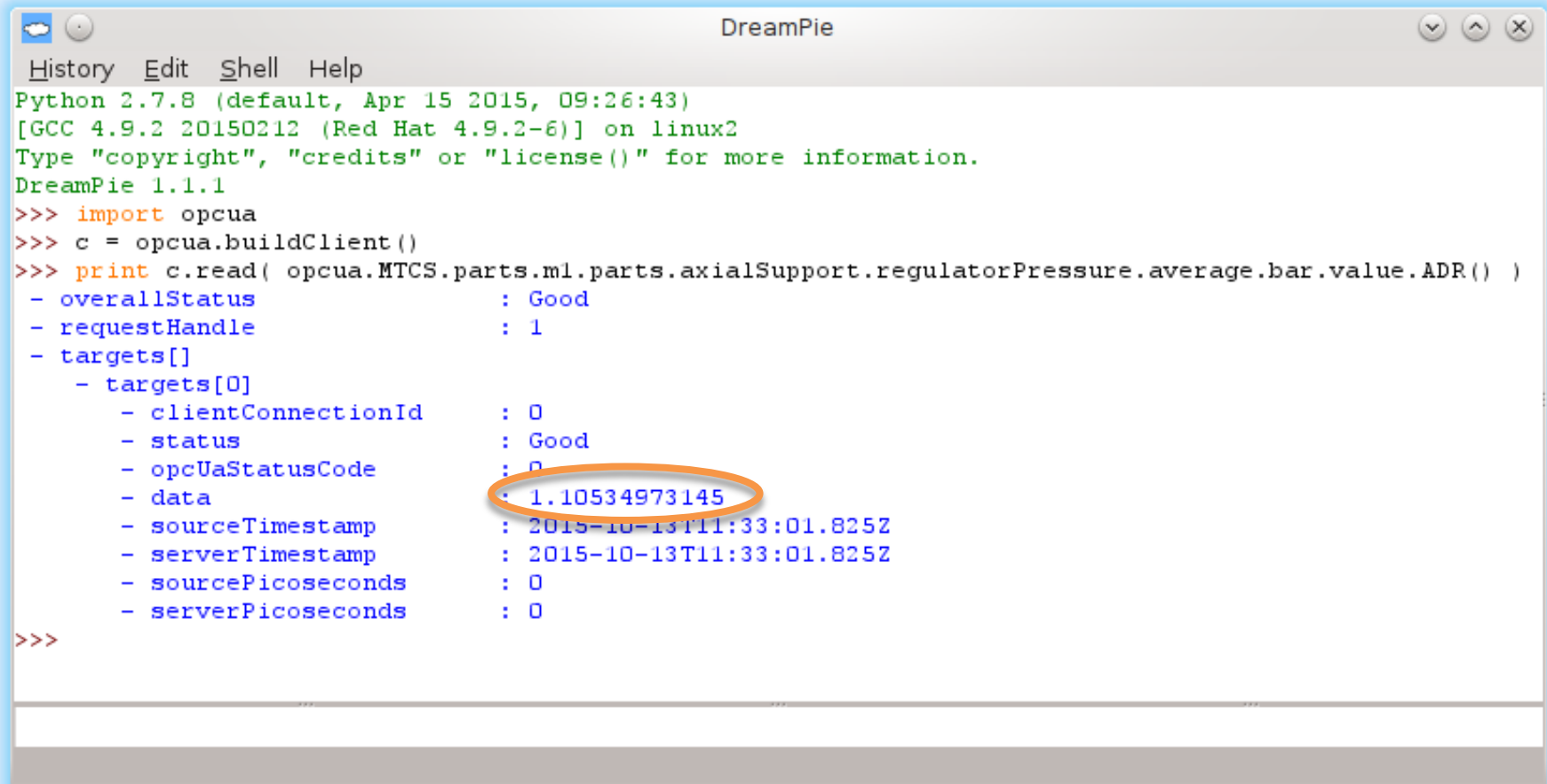
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>> print c.read( opcua.MTCS.parts.m1.parts.axialSupport.regulatorPressure.average.bar.value.ADR() )
- overallStatus          : Good
- requestHandle          : 1
- targets[]
  - targets[0]
    - clientConnectionId  : 0
    - status              : Good
    - opcUaStatusCode     : 0
    - data                : 1.10534973145
    - sourceTimestamp     : 2015-10-13T11:33:01.825Z
    - serverTimestamp     : 2015-10-13T11:33:01.825Z
    - sourcePicoSeconds   : 0
    - serverPicoSeconds   : 0
>>>
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Software design

- Generated Python code (client side)
 - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



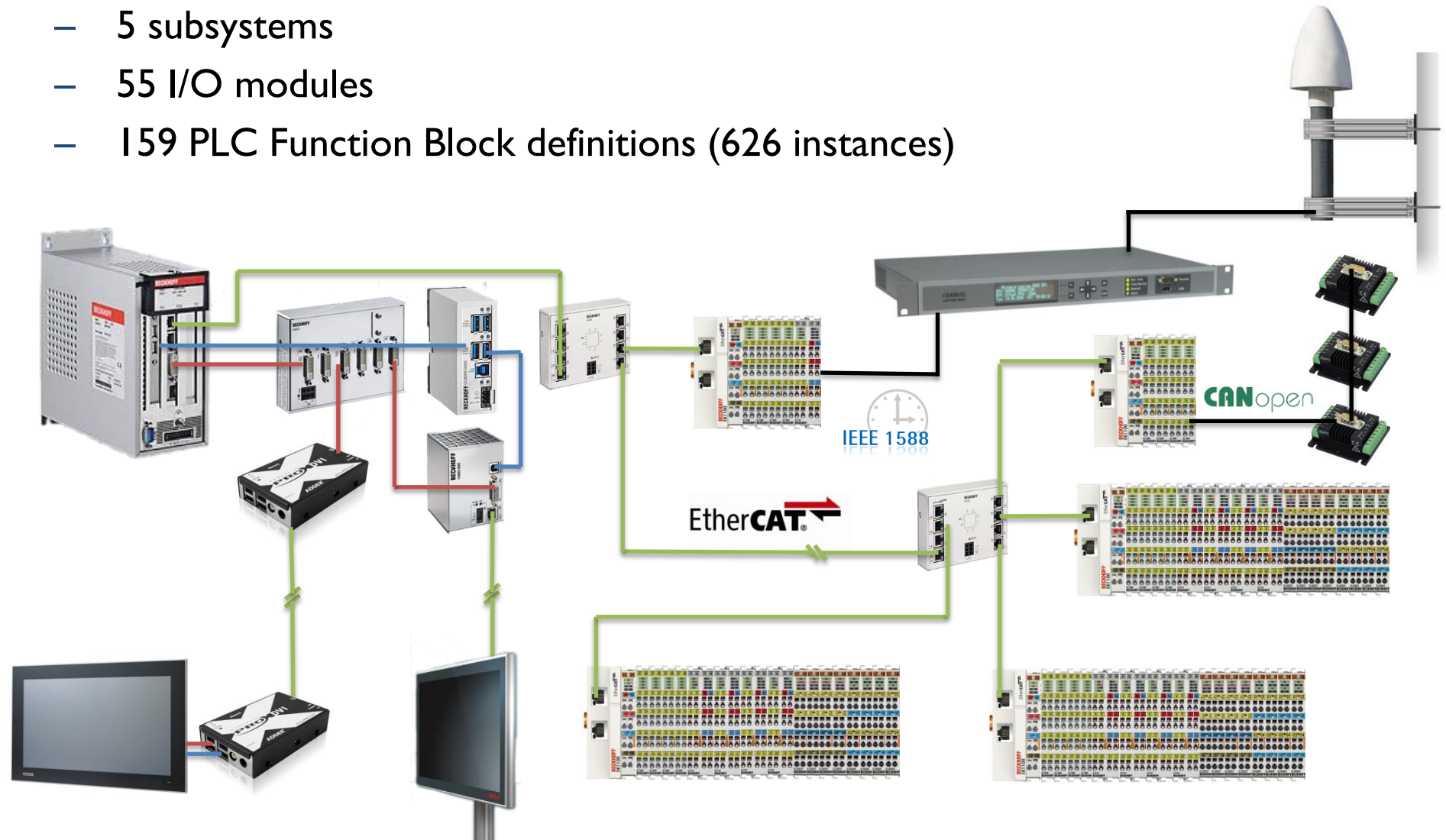
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>> print c.read( opcua.MTCS.parts.m1.parts.axialSupport.regulatorPressure.average.bar.value.ADR() )
- overallStatus          : Good
- requestHandle          : 1
- targets[]
  - targets[0]
    - clientConnectionId  : 0
    - status              : Good
    - opcUaStatusCode      : 0
    - data                : 1.10534973145
    - sourceTimestamp      : 2015-10-13T11:33:01.825Z
    - serverTimestamp      : 2015-10-13T11:33:01.825Z
    - sourcePicoSeconds    : 0
    - serverPicoSeconds    : 0
>>>
```

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

Results

- Currently in operation:
 - 1 PLC
 - 5 subsystems
 - 55 I/O modules
 - 159 PLC Function Block definitions (626 instances)



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Results



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Results



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Results

- User Interface (HMI) running **on** the PLC

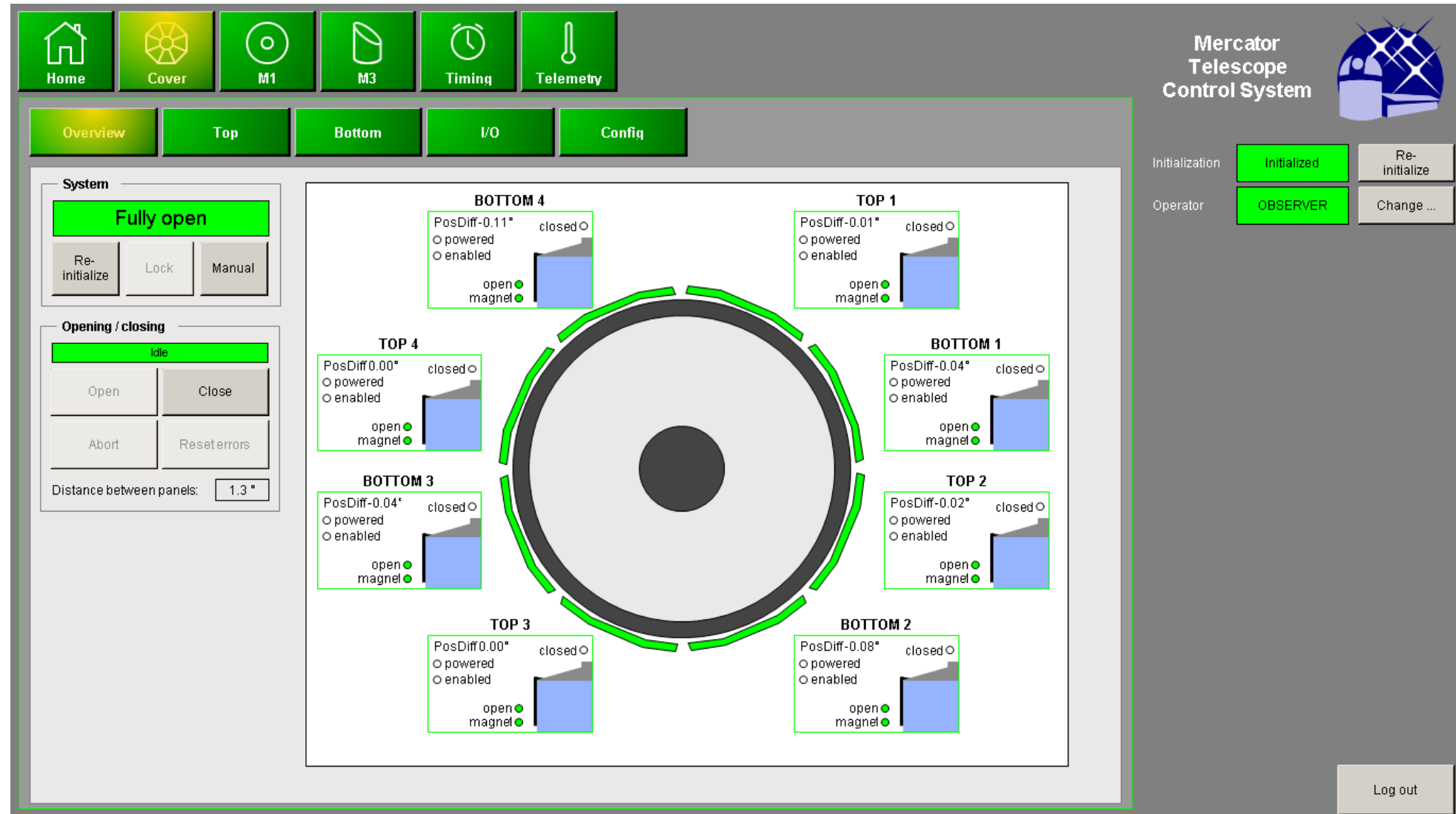


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Results

- User Interface (HMI) running **on** the PLC

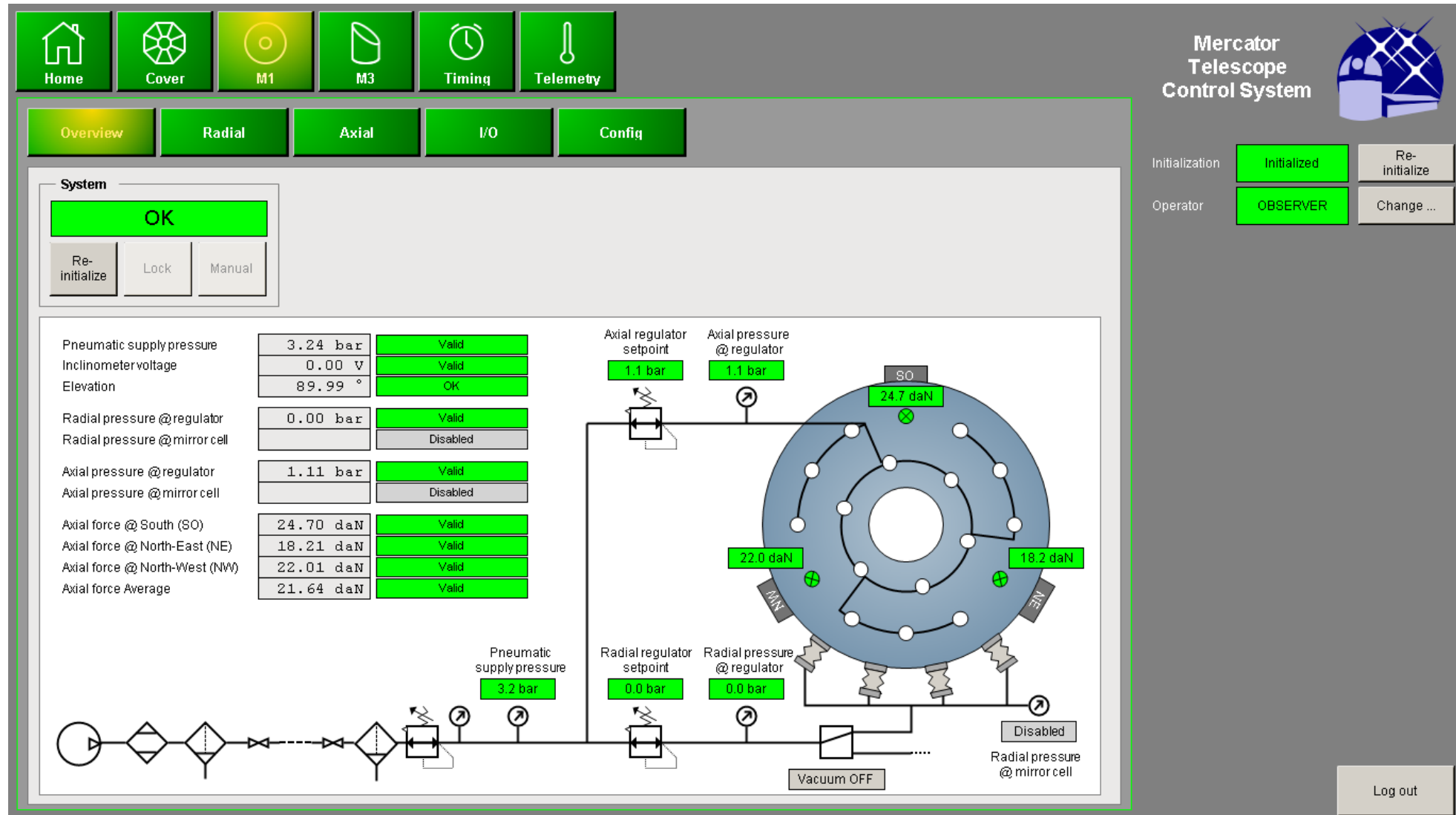


Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

Results

- User Interface (HMI) running **on** the PLC



Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

Conclusions

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

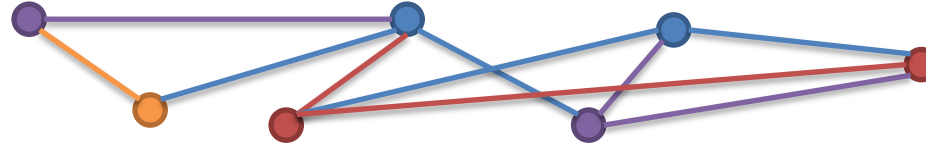
So, why semantics matter?

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

So, why semantics matter?

- I. Because every piece of information is just one query “away”



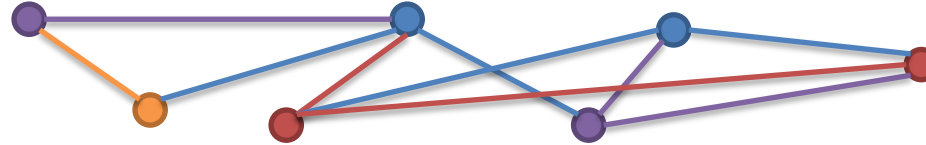
➔ **organize, integrate, browse, find (query) information**

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

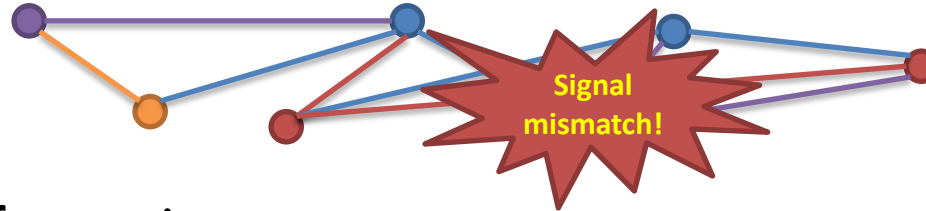
So, why semantics matter?

1. Because every piece of information is just one query “away”



➔ **organize, integrate, browse, find (query)** information

2. Because well defined semantics allow model verification



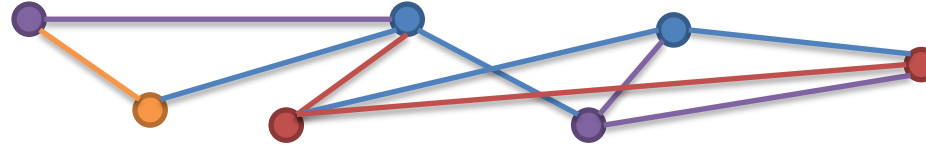
➔ **verify** information

Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

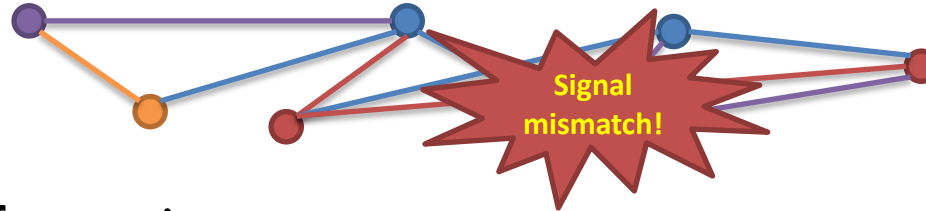
So, why semantics matter?

1. Because every piece of information is just one query “away”



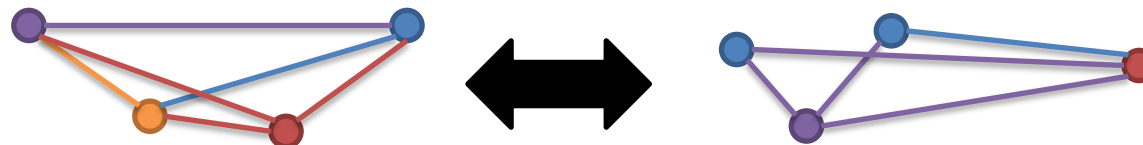
→ **organize, integrate, browse, find (query)** information

2. Because well defined semantics allow model verification



→ **verify** information

3. Because they're a key enabling technology for future “smart” systems



→ **share** information

A night sky with stars and a layer of clouds, with an observatory building and a dome in the foreground.

Thanks!

Any questions?

wim.pessemier@ster.kuleuven.be