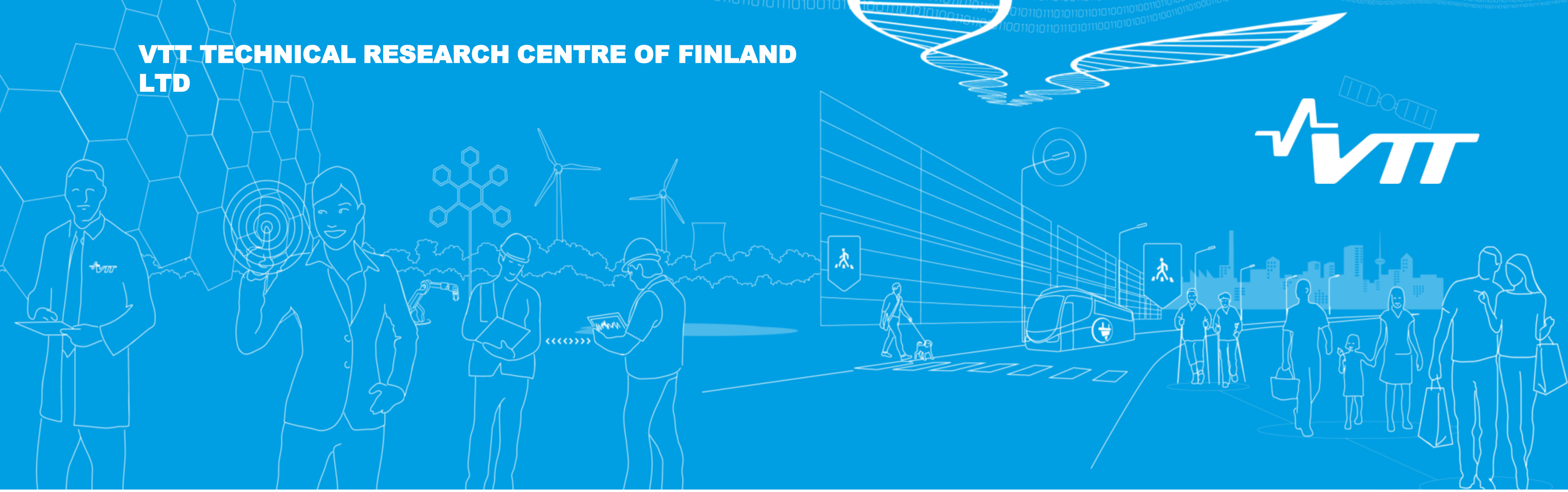
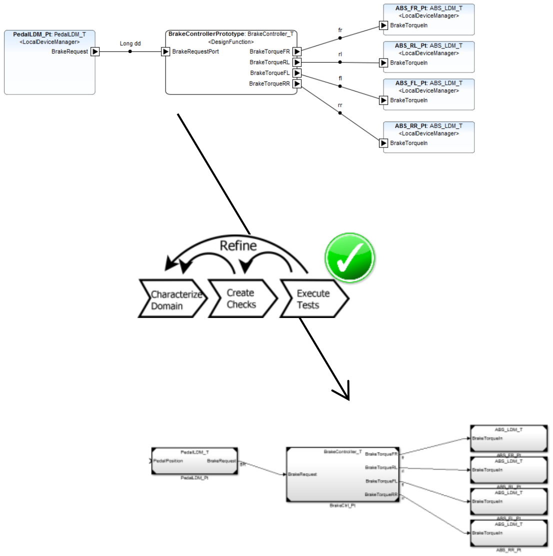


VTT TECHNICAL RESEARCH CENTRE OF FINLAND LTD

A Process for Model Transformation Testing



T. Kanstrén[1], M. Chechik[2], J-P. Tolvanen[3]

[1] VTT Technical Research Centre of Finland
[2] University of Toronto, Toronto, Canada
[3] MetaCase, Jyväskylä, Finland

Background

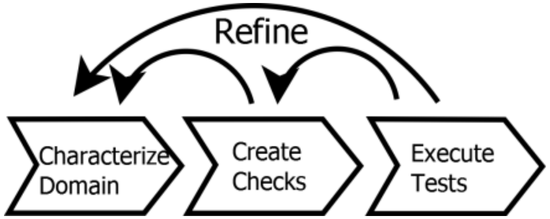
- A process initially defined while I was visiting at UofT, using their Class diagram to ER diagram as a case study in Eclipse ATL transformation environment
- After returning to Finland, extended with a more realistic case study of EAST-ADL model transformations in MetaEdit+
- In both cases, input domain is modelled using OSMO MBT, tests generated and executed against the real SUT (MT engine), the described process is fully applied and results verified in several iterations
- The paper gives practical experiences on building overall process for model transformation testing

A Model Transformation

- Use of models as design artifacts increasingly popular,
  - Model Driven Design,
  - Domain Specific Modeling,
  - Model-Based Testing, …
- Models need to be processed across different tools, transformed between abstraction levels, etc.
- Correctness of model transformation is crucial
  - If one model is wrong, that part of the system is broken
  - If the transformation is wrong, potentially all output is broken

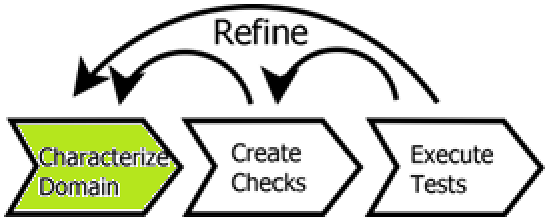
High-Level Process

1. Characterize Domain
2. Create Checks
3. Execute Tests



- During the first two phases, we build a test model to represent our understanding of
  - what the important properties of the transformation are,
  - how they are represented, and
  - how they are verified
- We then generate and execute tests, refine our understanding/test models continuously

Characterize Domain

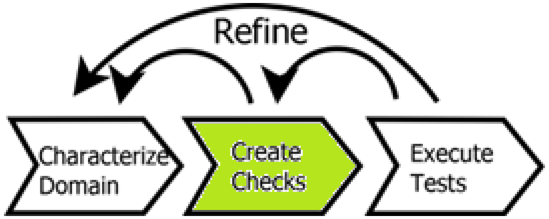


- A set of questions to guide the characterization:
  - What is the purpose of the transformation?
  - For each input (meta)model element:
    - Is it relevant (in scope) for the transformation?
    - How is it (or should be) affected by the transformation?
    - What should be produced for it in the output?
    - Does it have any other impact on the transformation?
- To identify
  - Transformation purpose, the important elements to test
  - To define a set of high-level transformation rules to analyze, the effective metamodel, and the coverage criteria

Coverage Criteria and Tool Support

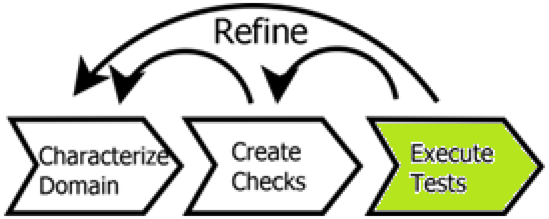
- Coverage category combinations
  - Various options (e.g., literature):
    - Generic categories based on any Effective Metamodel (EM)
    - Knowledge based (KB) for specific important elements
  - Our approach:
    - Start with generic options for EM
    - Refine tighter with KB pruning
    - 0,1,N/1,2,N, …
- White-box tools can assist in identifying the (current) EM
  - Also relevant for what is in coverage
  - However, does not tell if the EM is correct

Create Checks



- Using transformation rules and coverage criteria as input, define a set of test oracles for the defined input(s)
- Combining various types, e.g.,
  - Reference models, e.g., existing production samples
  - Generic checks, e.g., metamodel conformance
  - Rules (invariants) over generated inputs vs outputs
- Our process (iteratively repeated):
  - Define a check at least for each transformation rule.
  - Combine checks for the same elements.
  - Map the checks to the concrete output.
  - Review with domain expert.

Create & Execute Tests



- Create a set of tests for capture the domain characterization as well as the checks required
- Reference inputs and outputs as a starting point
- Large scale variation with test generators where needed
- Aiming to cover the coverage criteria defined

```
@Description("Creates a DFP instance in the input model")
@TestStep(weight = 5)
public void createDFP() {
  state.createDFP();
}

@Description("Creates a DFT instance in the input model")
@TestStep(weight = 2)
public void createDFT() {
  state.createDFT();
}

@Description("Guards the linkDFPtoDFT step, requiring a DFT and a free DFP to exist. ")
@Guard("LinkDFPtoDFT")
public boolean hasUnlinkedDFP() {
  return state.getUnlinkedDFPs().size() > 0 && state.getDFTs().size() > 0;
}
```

1. Generate input + checks

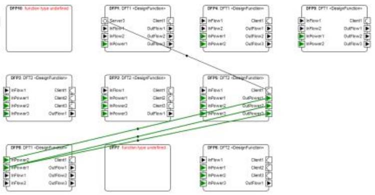
```
<?xml version="1.0" encoding="UTF-8"?>
<gxl xmlns="http://www.metacase.com/gxlGOPRR">
  <graph id="_19_99637" typeName="DesignFunctionType">
    <slot name="FunctionName">
      <value> <string>output1</string> </value>
    </slot>
    <slot name="Description">
….
```

2. Execute test script

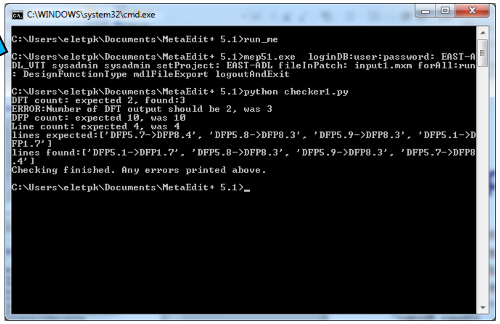
```
mep51.exe  loginDB:user:password:
EAST-ADL_VTT sysadmin sysadmin
setProject: EAST-ADL fileInPatch:
input1.mxm forAll:run:
DesignFunctionType mdlFileExport
logoutAndExit

python checker1.py
```

3. Input model

4. Execute transformation



5. Run checks

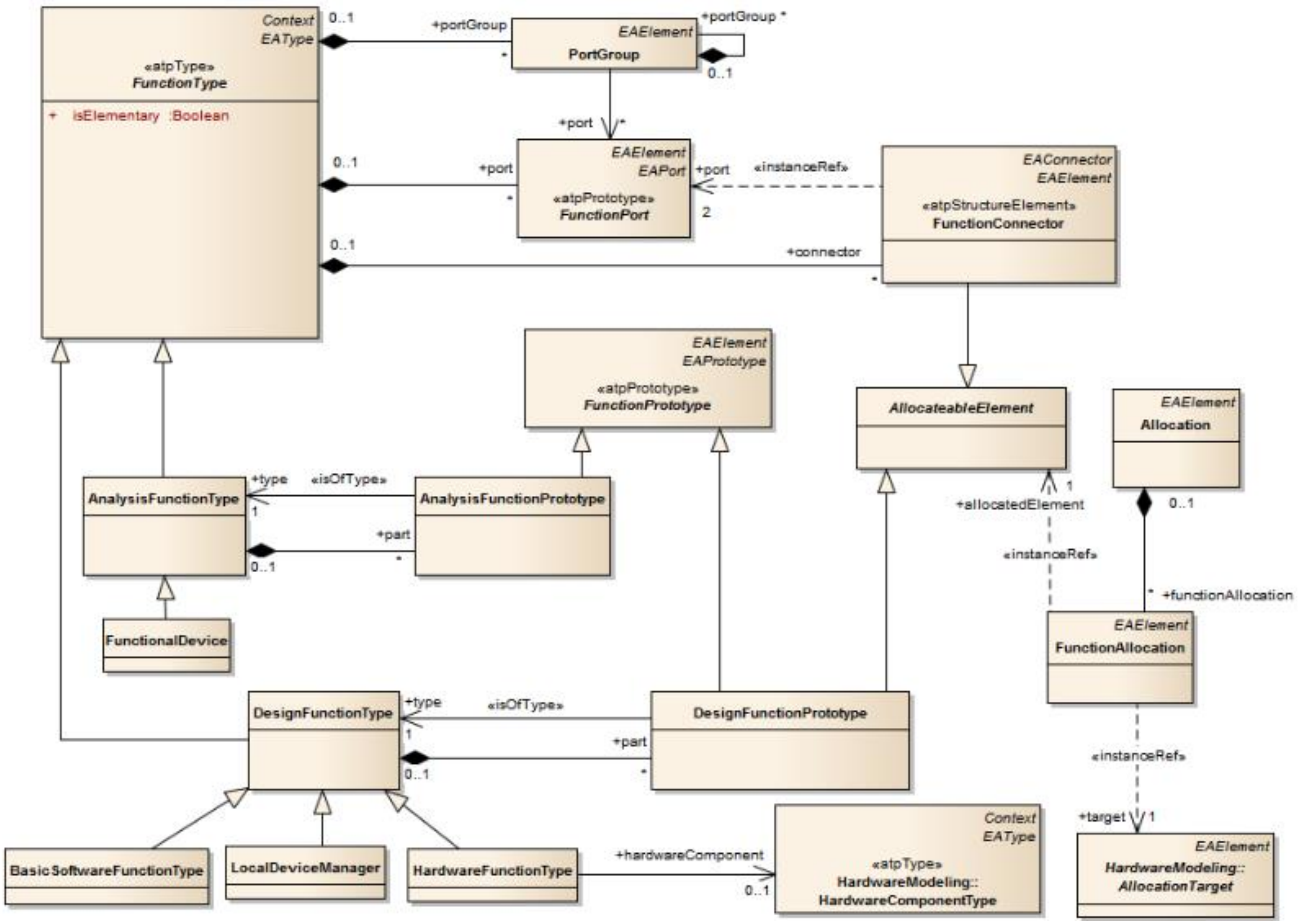


```
C:\Users\eletpk\Documents\MetaEdit+ 5.1>run_me

C:\Users\eletpk\Documents\MetaEdit+ 5.1>mep51.exe  loginDB:user:password: EAST-A
DL_VTT sysadmin sysadmin setProject: EAST-ADL fileInPatch: input1.mxm forAll:run
: DesignFunctionType mdlFileExport logoutAndExit

C:\Users\eletpk\Documents\MetaEdit+ 5.1>python checker1.py
DFT count: expected 2, found:3
ERROR:Number of DFT output should be 2, was 3
DFP count: expected 10, was 10
Line count: expected 4, was 4
lines expected:['DFP5.7->DFP8.4', 'DFP5.8->DFP8.3', 'DFP5.9->DFP8.3', 'DFP5.1->D
FP1.7']
lines found:['DFP5.1->DFP1.7', 'DFP5.8->DFP8.3', 'DFP5.9->DFP8.3', 'DFP5.7->DFP8
.4']
Checking finished. Any errors printed above.

C:\Users\eletpk\Documents\MetaEdit+ 5.1>_
```

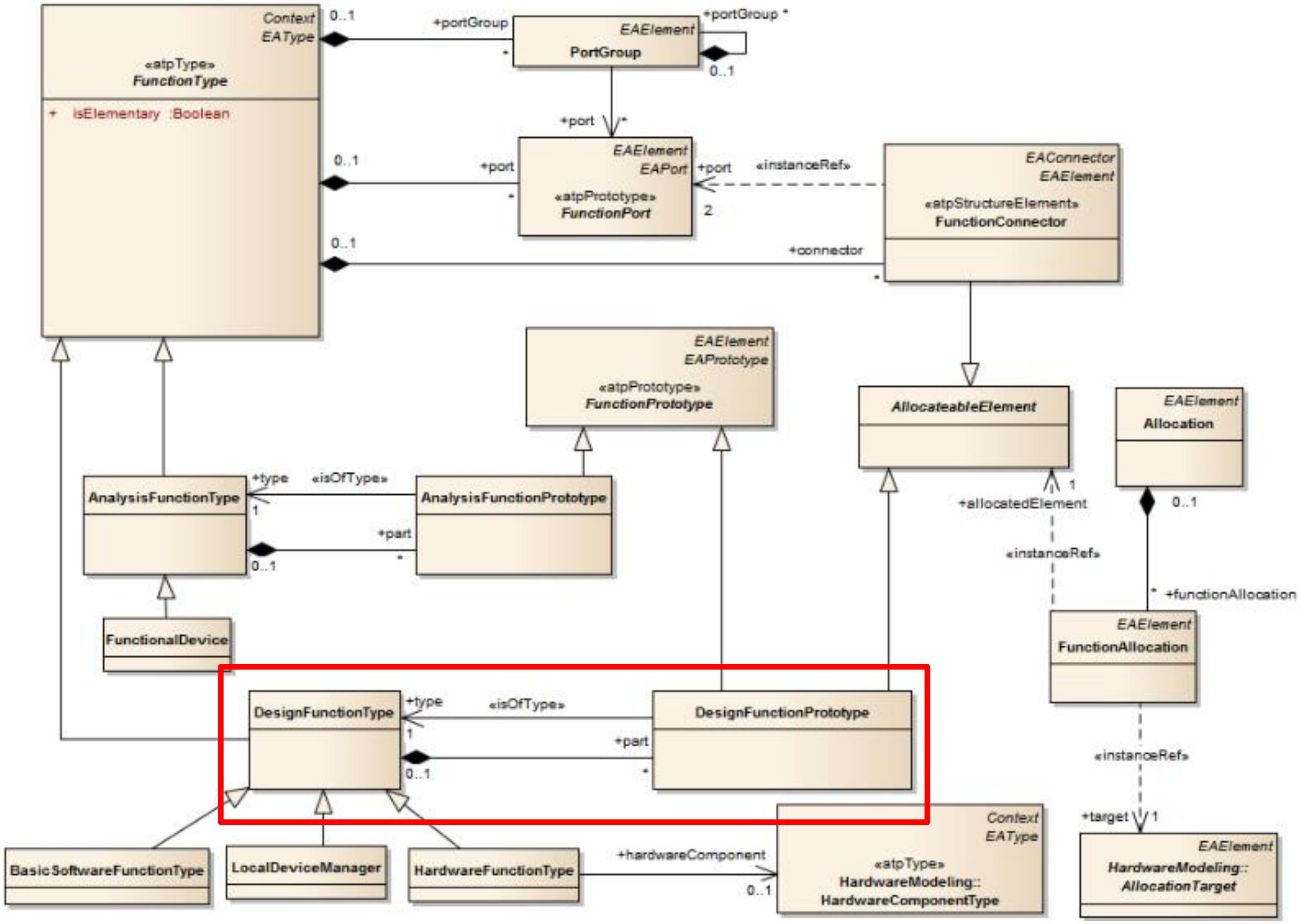
Example: EAST-ADL ME+ to Simulink MT

- EAST-ADL is an Architecture Definition Language (ADL) for the automotive domain
- Typical use case:
  - Use MetaEdit+ with EAST-ADL to define static architecture
  - Use Simulink to describe the component behavior
- MetaEdit+ provides 3 transformations
  - MetaEdit+ to Simulink
  - Simulink back to MetaEdit+
  - Verify that the two are still a match
- We take an example subset to illustrate the process

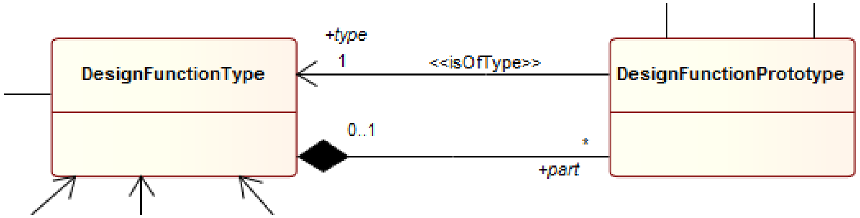
EAST-ADL Metamodel (partial, ~5%)



Transformation Effective Metamodel



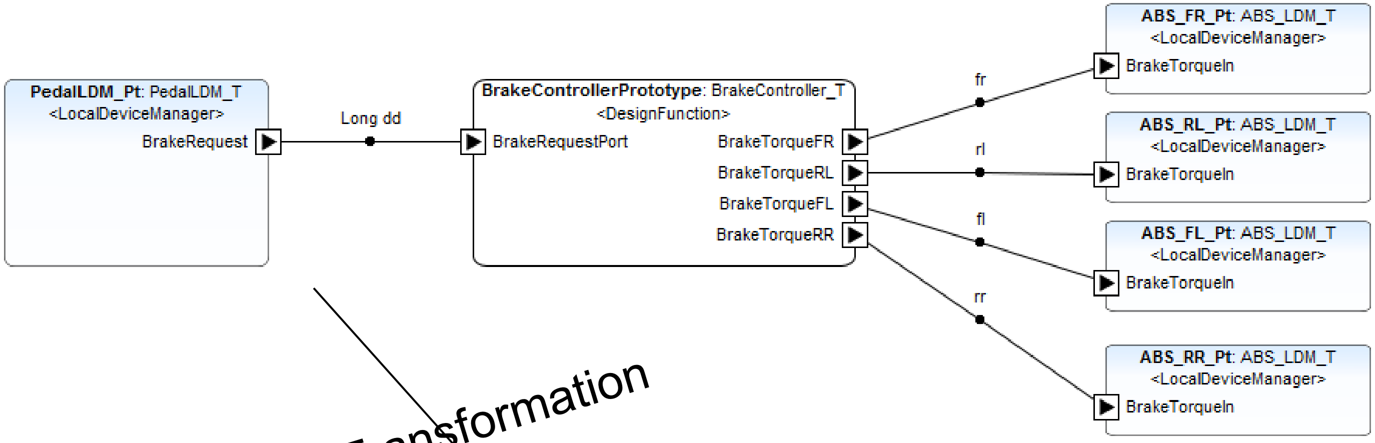
Rules



- ~70 rules identified
- Examples from transforming ME+ *DFP*’s to Simulink *Blocks*:
- For each DFP, create a *Block* and for this
- Add *BlockType* with static value *ModelReference*
- Add *Name*, with value of *Short name* from DFP.
- Add *SID*, with increasing unique integer.
- If *Description* is defined for DFP, add *Description* with this value from DFP.
- Add *AttributeFormatString*, with value of *Name* from DFP.
- Add *ModelNameDialog*, with value of *FunctionName* from DFP.
- Add *ModelReferenceVersion* with static value 1.0
- Add *List* and inside it
  - Add *ListType* with static value *InputPortNames*
  - For each *InFlowPort* in DFP
    - Add *port X* where X is an increasing integer, with value of *Short name* from *InFlowPort*
  - For each *InPowerPort* in DFP, do the same as for *InFlowPort*
  - For each *ServerPort* in DFP, do the same as for *InFlowPort*
- Add *List* and inside it
  - Add *ListType* with static value *outputPortNames*
  - For each *OutFlowPort* in DFP,
    - Add *port X* where X is increasing integer, with value of *Short name* from *OutFlowPort*
  - For each *OutPowerPort* in DFP, do the same as for *OutFlowPort*
  - For each *ClientPort* in DFP, do the same as for *OutFlowPort*

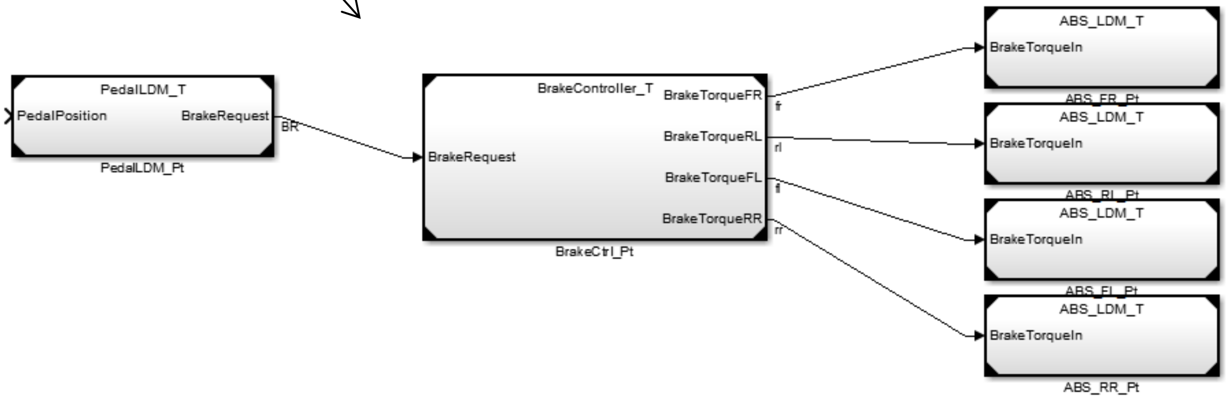
Example: A brakecontroller MT

MetaEdit+



ME+ Transformation

Simulink



Test Model Elements for ME+ to Simulink

- Adaptation of sequence/ state-based test generation
- Traditionally generating sequences of test steps, interacting with the SUT in different states
- Instead, model steps build test models, and MT is executed once in the end
- Coverage measured as model element combinations

Example rules & actions:

Rules	Actions
Always allowed	Create a DFT
Always allowed	Create a DFP
DFT’s > 0 && unlinked DFP’s > 0	Link DFT to DFP
DFT’s > 0	Add InFlowPort to DFT
DFT’s > 0	Add OutFlowPort to DFT
DFT’s with no description > 0	Add description to DFT

Coverage Elements

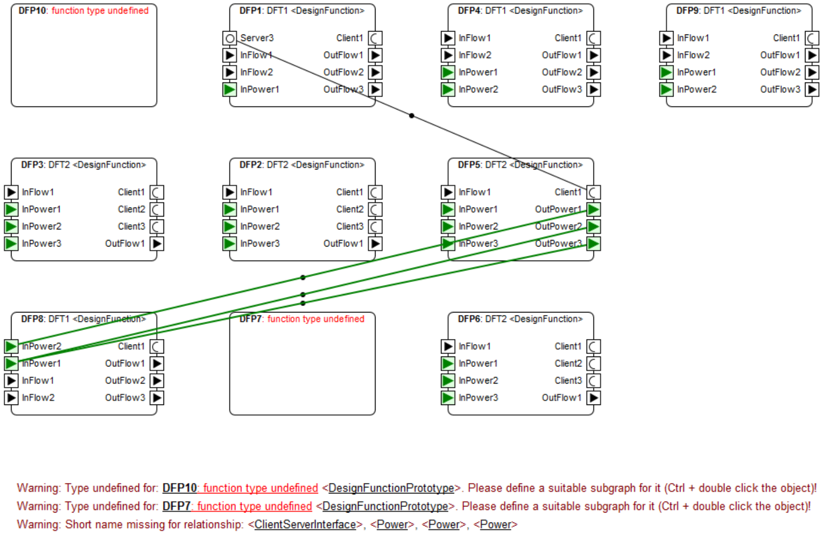
- Combinations of model elements as seen important
  - Base element count
  - DFT/DFP
  - DFP count + port type counts
  - DFP count + port type counts + connection counts
- ~300 combinations, fully covered in 25 tests

Example coverage categories

Model Element	Categories
DFP	1,2,N
InFlows/DFP	0,1,N
OutFlows/DFP	0,1,N
InPowers/DFP	0,1,N
OutPowers/DFP	0,1,N
Servers/DFP	0,1,N
Clients/DFP	0,1,N
Description	0,1

Example Generated Model

```
<?xml version="1.0" encoding="UTF-8"?>
<gxl xmlns="http://www.metacase.com/gxlGOPRR">
  <graph id="_19_99637" typeName="DesignFunctionType">
    <slot name="FunctionName">
      <value> <string>output1</string> </value>
    </slot>
    <slot name="Description">
      <value> <text>Holder for everything else</text> </value>
    </slot>
  <object id="178247" typeName="DesignFunctionPrototype">
    <slot name="Short name">
      <value> <string>DFP1</string> </value>
    </slot>
    <slot name="Description">
      <value> <text>Description178247</text> </value>
    </slot>
    <graph id="178272" typeName="DesignFunctionType">
      <slot name="FunctionName">
        <value> <string>DFT1</string> </value>
      </slot>
      <slot name="Description">
        <value> <text>g149M</text> </value>
      </slot>
      <object id="178274" typeName="InFlowPort">
        <slot name="Short name">
          <value> <string>InFlow1</string> </value>
        </slot>
       ….
```



Warning: Type undefined for: **DFP10**: function type undefined <DesignFunctionPrototype>. Please define a suitable subgraph for it (Ctrl + double click the object)!
Warning: Type undefined for: **DFP7**: function type undefined <DesignFunctionPrototype>. Please define a suitable subgraph for it (Ctrl + double click the object)!
Warning: Short name missing for relationship: <ClientServerInterface>, <Power>, <Power>, <Power>

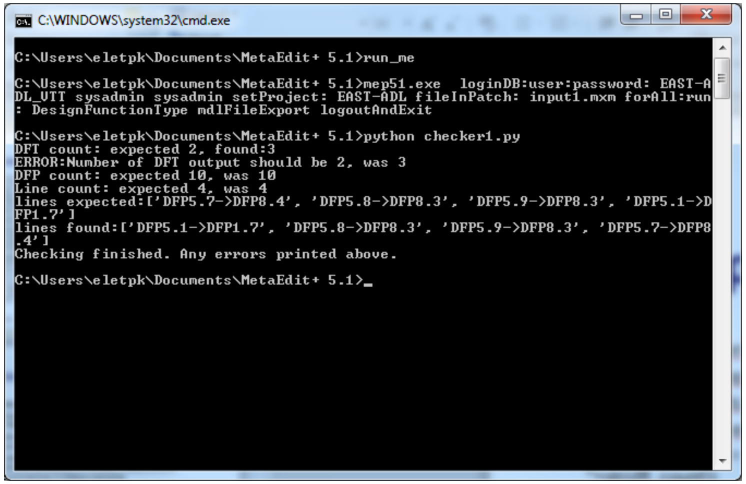
Coverage criteria met with this model:

- 1 Client – server line
- N InPower – OutPower lines
- N – ” – lines with single input
- N undefined DFT for DFP
- N unconnected lines (various type)
- ….

Example generated checks

```
def check(value, error):
dfts = glob.glob("reports/DFT*.mdl")
dft_count = len(dfts)
print("DFT count: expected "+str(2)+", found:"+str(dft_count))
check(dft_count == 2, "Number of DFT output should be 2, was
"+str(dft_count))
dfp_inports = {"DFP10":0, "DFP9":10, "DFP8":10, "DFP7":0,
"DFP6":5, "DFP5":5, "DFP4":10, "DFP3":5, "DFP2":5, "DFP1":10}
dfp_outports = {"DFP10":0, "DFP9":6, "DFP8":6, "DFP7":0,
"DFP6":9, "DFP5":9, "DFP4":6, "DFP3":9, "DFP2":9, "DFP1":6}
dfp_descriptions = {"DFP1":"Description178247",
"DFP2":"Description178251", "DFP3":"Description178253",
"DFP4":"Description178256", "DFP5":"Description178258",
"DFP6":"Description178260", "DFP7":"Description178264",
"DFP8":"Description178267", "DFP9":"Description178268",
"DFP10":None}
lines_expected = ["DFP5.7->DFP8.4", "DFP5.8->DFP8.3",
"DFP5.9->DFP8.3", "DFP5.1->DFP1.7"]
with open("reports/output1.mdl") as f:
    content = f.readlines()
in_line = False
for line in content:
    l = line.strip()
    if l.startswith("BlockType"):
        check(l.endswith("ModelReference"), "BlockType should
always be 'ModelReference'. For "+str(name)+" found "+l)
….
```

```
mep51.exe  loginDB:user:password: EAST-ADL_VTT
sysadmin sysadmin setProject: EAST-ADL fileInPatch:
input1.mxm forAll:run: DesignFunctionType mdlFileExport
logoutAndExit

python checker1.py
```



```
C:\Users\eletpk\Documents\MetaEdit+ 5.1>run_me

C:\Users\eletpk\Documents\MetaEdit+ 5.1>mep51.exe  loginDB:user:password: EAST-A
DL_VTT sysadmin sysadmin setProject: EAST-ADL fileInPatch: input1.mxm forAll:run
: DesignFunctionType mdlFileExport logoutAndExit

C:\Users\eletpk\Documents\MetaEdit+ 5.1>python checker1.py
DFT count: expected 2, found:3
ERROR:Number of DFT output should be 2, was 3
DFP count: expected 10, was 10
Line count: expected 4, was 4
lines expected:['DFP5.7->DFP8.4', 'DFP5.8->DFP8.3', 'DFP5.9->DFP8.3', 'DFP5.1->D
FP1.7']
lines found:['DFP5.1->DFP1.7', 'DFP5.8->DFP8.3', 'DFP5.9->DFP8.3', 'DFP5.7->DFP8
.4']
Checking finished. Any errors printed above.

C:\Users\eletpk\Documents\MetaEdit+ 5.1>_
```

Experiences

- Issues found in all stages of the process
- Modeling phase: ambiguities, misunderstandings and missing requirements
- Manually created and reference models find the most obvious issues in the implementation.
- Large scale, automatically generated tests are best at discovering issues with complex interactions and combinations of different elements.
- The modeling environment can already be used to constrain much of the input space beyond the metamodel

Error Types and Handling

- DSL constraints prevent creating illegal inputs
- DSL warns about illegal input but allows creating it and running the transformation
- DSL allows creating illegal input but prevents running the transformation
- Output is illegal but target tool (TT) opens it (showing errors)
- Output is illegal and TT does not even open it
- Output is illegal and TT does not recognize it
- Model is legal for source metamodel but not on target metamodel

- ➢ Flow to Power not possible, InFlow to InFlow not possible
- ➢ Prototype does not have type defined, port does not have data type
- ➢ (not in this case), but missing relationship name in XML export
- ➢ Type information deleted from prototype, still connections exist
- ➢ Transformation produces broken output, invalid elements, etc.
- ➢ End tag etc missing, input is garbage
- ➢ A port has several incoming flow connections: OK in EAST-ADL, not in Simulink)

Conclusions

- The process is a result of performing systematic MT testing using MBT tools, and distilling a generic process
  - Requires broad expertise and effort
  - Depends on how critical you see your transformation
- Broader, more cost-effective application could be achieved via integrating approach with DSM workbench tooling
  - Already access to metamodels, rules, etc.
  - For example, embed generators, build tests by picking metamodel elements and rules
- Example generator code available at: https://github.com/mukatee/dsm-mbt-example