The TTC 2017 Outage System Case for Incremental Model Views

Georg Hinkel
FZI Research Center of Information Technologies
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
hinkel@fzi.de

Abstract

To cope with the increased complexity, physical systems are more and more supported by software systems consisting of multiple subsystems. Usually, each of the subsystems uses standards relevant to the subsystem for interoperability with other tools. Thus, one faces the problem that information about the system as a whole is distributed across multiple models. To solve this problem, model views can be introduced to combine these models and extract application-specific knowledge. As an example, the smart grid is a cyber-physical system where one is interested to detect, manage and prevent system outages. The information necessary to do this is split among the standards IEC 61970/61968, IEC 61850 and IEC 62056. This paper presents a benchmark case and evaluation framework for joining information spread across multiple models into a single view, based on a model-based outage management system for smart grids. Because cyber-physical systems often require very fast response times to changes of underlying models, the benchmark focuses especially on the incremental computation of model views.

1 Introduction

The complexity of todays systems, for example cyber-physical systems, makes it inevitable to divide the system into multiple subsystems that operate in different domains. In many of these domains, standards exist that the respective subsystem has to comply with or for which many tools can be reused.

For example, the smart grid is a cyber-physical system that spans the physical structures of the electricity network and the system of software systems that monitor, control, and repair the system in case of outages [1]. Currently, many heterogeneous systems and standards have to interoperate to achieve the desired reliability, stability, and efficiency of the electricity network.

Because each of these standards describe different aspects of the system, models according to these standards have to be combined if multiple aspects are required to gain some insights about the system. Applying model-driven engineering, model views are a tool to extract information from multiple models without confronting the user with unnecessary information for a particular purpose, for example analysis.

In the area of smart grids, an additional challenge is the size of the models and the frequency of changes. In combination, this means that very large amounts of data have to be processed in a very short amount of time. However, the changes usually only affect small parts of the model, which is why an incremental view computation appears beneficial.

In: A. Garcia-Dominguez, F. Krikava and G. Hinkel (eds.): Proceedings of the 10th Transformation Tool Contest, Marburg, Germany, 21-07-2017, published at http://ceur-ws.org

In this paper, we propose a benchmark based on selected model views created by Mittelbach and Burger [1], [2] for a model-based outage management system for smart grids. Section 2 introduces the case in a bit more detail, in particular the involved standards. Section 3 defines two tasks of the benchmark. Section 4 introduces the benchmark framework. Section 5 explains how solutions to the benchmark are to be evaluated.

2 Case Description

In the area of smart grids, the relevant standards are IEC 61970/61968, IEC 61850 and IEC 62056. These standards are briefly described below (taken from [1]):

IEC 61970/61968 The IEC 61970 standard defines the *Common Information Model (CIM)*, which is used to describe the physical components, measurement data, control and protection elements, and the SCADA system. It is defined in UML notation. The IEC 61968 standard is an extension of the CIM for the distribution network [3]. It is also called *distributed CIM (DCIM)*

IEC 61850 This is a series of standards for substations with the purpose of supporting interoperability of intelligent electronic devices (IED) in substation automation systems. It defines the *Abstract Communication Service Interface* with a mapping to concrete communication protocols, the XML-based *Substation Configuration Description Language (SCL)*, and the *Logical Node (LN)* model, which describes power system functions [4].

IEC 62056 *COSEM (Companion Specification for Energy Metering)* is the international standard for data exchange for meter reading, tariff and load control in the domain of electricity metering. It works together with the *Device Language Message Specification (DLMS)*. Together, they provide a communication profile to transport data from metering equipment to the metering system and to define a data model and communication protocols for data exchange [5].

While these standards are useful in their domain, one has to combine the information represented by these standards to detect and prevent outage situations. Mittelbach and Burger presented a model-based outage system that synchronizes models of these standards and consists of a set of 15 views to help operators to manage outage situations [1], [2].

3 Tasks

In the scope of the proposed benchmark, we focus on two model views contained in the model-based outage management system. A rather simple view is created to detect outages, while a second slightly more complex view supports the prevention of outages.

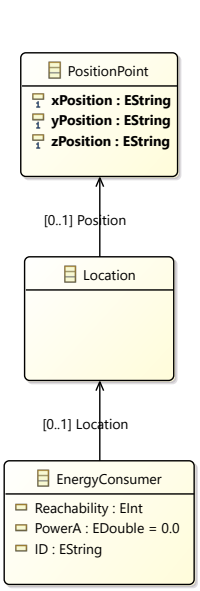
For both tasks, we present the original implementation of the view in ModelJoin [6], a language to specify both the view type and the view definition in a single specification through a language inspired by SQL joins. Currently, an idiomatic QVT-O model transformation is generated from this specification. Due to space limitations, we do not show the generated transformation, but it is available in the benchmark resources for reference.

3.1 Task 1: A view to detect outages

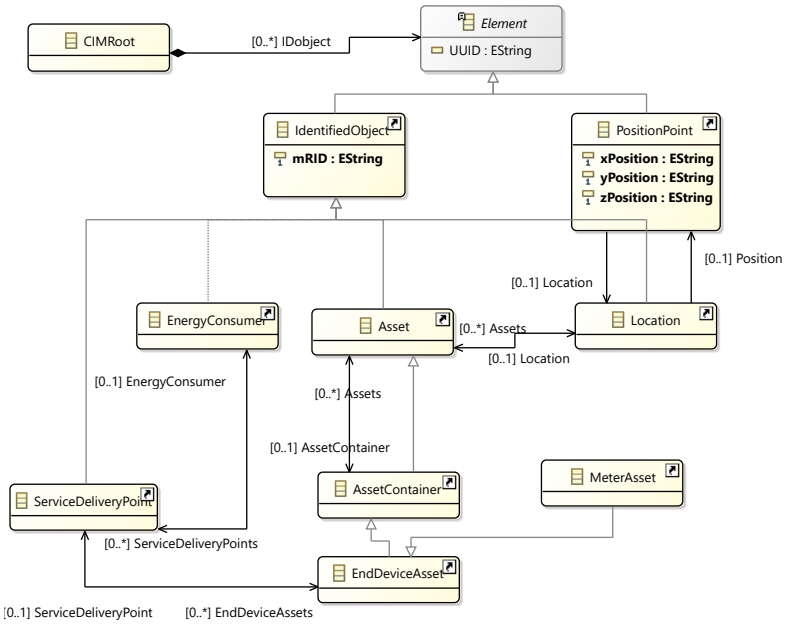
To detect an outage, we use the fact that a smart meter cannot send any data when it is cut off power supply. If this happens, the system can try to reach the meter but will receive a connection failure notification. This is used to detect outages without relying on customer feedback.

The information that a connection to a smart meter is lost is depicted in the CIM model. The relevant excerpt for this task is depicted in Figure 1b. It has to be matched with the corresponding physical devices in the COSEM model where its location is stored. The latter is depicted in Figure 1c.

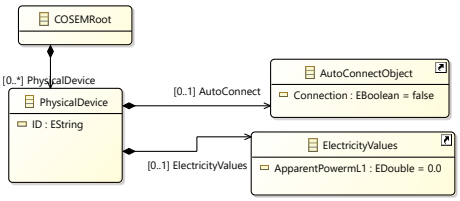
An implementation in ModelJoin is depicted in Listing 1.



(a) The Viewtype for the Outage Detection Task



(b) Excerpt from the CIM metamodel relevant for task 1



(c) Excerpt from the COSEM metamodel relevant for task 1

Figure 1: Metamodels in task 1

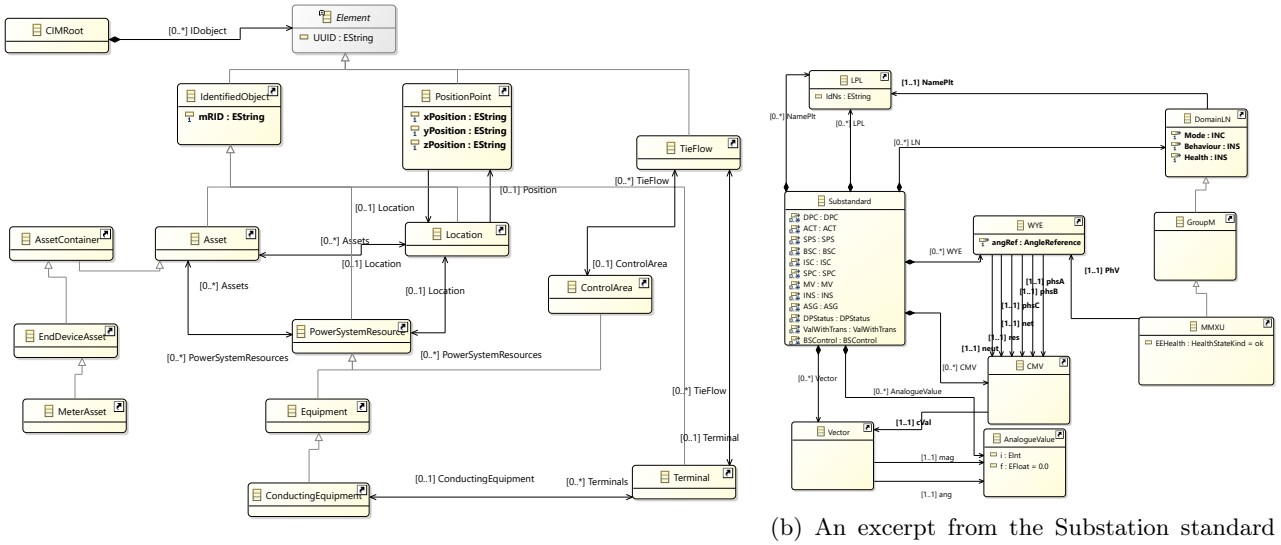
```
1  theta join CIM.IEC61968.Metering.MeterAsset with COSEM.PhysicalDevice where "CIM.IEC61968.Metering.MeterAsset.mRID␣=␣COSEM.
       PhysicalDevice.ID" as jointarget.EnergyConsumer {
2     keep calculated attribute "COSEM.PhysicalDevice.AutoConnect.Connection" as EnergyConsumer.Reachability:Integer
3     keep calculated attribute "COSEM.PhysicalDevice.ElectricityValues.ApparentPowermL1" as EnergyConsumer.PowerA:Double
4     keep calculated attribute "CIM.IEC61968.Metering.MeterAsset.ServiceDeliveryPoint.EnergyConsumer.mRID" as EnergyConsumer.ID:
          String
5     keep calculated attribute "if␣CIM.IEC61968.Metering.MeterAsset.ServiceDeliveryPoint.EnergyConsumer->oclIsKindOf(CIM.IEC61970.
          LoadModel.ConformLoad)␣then␣CIM.IEC61968.Metering.MeterAsset.ServiceDeliveryPoint.EnergyConsumer.ConformLoadGroup.
          SubLoadArea.LoadArea.ControlArea.mRID␣else␣CIM.IEC61968.Metering.MeterAsset.ServiceDeliveryPoint.EnergyConsumer.
          NonConformLoadGroup.SubLoadArea.LoadArea.ControlArea.mRID␣endif" as Consumer.ControlAreaID:String
6     keep outgoing CIM.IEC61968.Assets.Asset.Location as type jointarget.Location {
7        keep outgoing CIM.IEC61968.Common.Location.Position as type jointarget.PositionPoint {
8           keep attributes CIM.IEC61968.Common.PositionPoint.xPosition,
9           CIM.IEC61968.Common.PositionPoint.yPosition,
10          CIM.IEC61968.Common.PositionPoint.zPosition
11       }
12    }
13 }
```

Listing 1: Task 1 realized in ModelJoin

Based on the view specification in Listing 1, the view type in Figure 1a is generated.

A correct solution should match the meter assets in the CIM model with the physical devices in the COSEM model. For each of these matches, the result model should contain a model element that conforms to the generated view type class `EnergyConsumer`. For this element, a range of (partially derived) attributes shall be copied and the reference to the location of meter asset and physical device should be saved.

This reference to location and position point shall respect referential integrity. This means, if two meter assets in the CIM model reference the same location, their joins in the view should also reference the same `Location` element in the view.



(a) An excerpt from the CIM metamodel relevant for Task 2

(b) An excerpt from the Substation standard relevant for Task 2

Figure 2: Metamodels relevant for Task 2

3.2 Task 2: A view to prevent outages

The analysis algorithms to detect system disturbances proposed in [2] work on *phasor measurement data*: *Their basic concept is to compare the current phasor data of the traveling voltage wave with a historic set of normal phasor data and calculate an equality indicator like a correlation coefficient. This is compared with a certain benchmark. If it lies above, a failure is indicated. To enable this, the following information is necessary: a historic set of normal phasor data of that section, a matrix of the current phasor data and a calculation mechanism to compare the two followed by a comparison mechanism to decide if it is a failure or not.* [1]

Task 2 requires to match elements from all three domain standards. For the COSEM standard, the used metamodel excerpt is very similar to Task 1. The relevant metamodel excerpts for CIM and the substation standard are depicted in Figure 2a and Figure 2b, respectively.

The analysis viewtypes will not provide the analysis result but only the *matrix of phasor data* for the comparison. Six queries were defined in [1] that all have the same structure and provide the three-phase measurements of voltage, frequency, current, active power, reactive power and apparent power. An implementation of this matrix in ModelJoin is depicted in Listing 2.

```
1  theta join CIM.IEC61968.Metering.MeterAsset with substationStandard.LNNodes.LNGroupM.MMXU where "CIM.IEC61968.Metering.MeterAsset.
       mRID␣=␣substationStandard.LNNodes.LNGroupM.MMXU.NamePlt.IdNs" as jointarget.PMUVoltageMeter {
2     keep attributes CIM.IEC61970.Core.IdentifiedObject.mRID
3     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.phsA.cVal.mag.f" as PMUVoltageMeter.VoltageAMag:Double
4     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.phsA.cVal.ang.f" as PMUVoltageMeter.VoltageAAng:Double
5     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.phsB.cVal.mag.f" as PMUVoltageMeter.VoltageBMag:Double
6     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.phsB.cVal.ang.f" as PMUVoltageMeter.VoltageBAng:Double
7     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.phsC.cVal.mag.f" as PMUVoltageMeter.VoltageCMag:Double
8     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.phsC.cVal.ang.f" as PMUVoltageMeter.VoltageCAng:Double
9     keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.neut.cVal.mag.f" as PMUVoltageMeter.VoltageNeutMag:
          Double
10    keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.neut.cVal.ang.f" as PMUVoltageMeter.VoltageNeutAng:
          Double
11    keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.net.cVal.mag.f" as PMUVoltageMeter.VoltageNetMag:Double
12    keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.net.cVal.ang.f" as PMUVoltageMeter.VoltageNetAng:Double
13    keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.res.cVal.mag.f" as PMUVoltageMeter.VoltageResMag:Double
14    keep calculated attribute "substationStandard.LNNodes.LNGroupM.MMXU.PhV.res.cVal.ang.f" as PMUVoltageMeter.VoltageResAng:Double
15    keep supertype CIM.IEC61968.Assets.Asset as type jointarget.Asset {
16       keep outgoing CIM.IEC61968.Assets.Asset.Location as type jointarget.Location {
17          keep outgoing CIM.IEC61968.Common.Location.Position as type jointarget.PositionPoint {
18             keep attributes CIM.IEC61968.Common.PositionPoint.xPosition,
19             CIM.IEC61968.Common.PositionPoint.yPosition,
20             CIM.IEC61968.Common.PositionPoint.zPosition
21          }
22          keep outgoing CIM.IEC61968.Common.Location.PowerSystemResources as type jointarget.PowerSystemResource {
23             keep subtype CIM.IEC61970.Core.ConductingEquipment as type jointarget.ConductingEquipment {
24                keep outgoing CIM.IEC61970.Core.ConductingEquipment.Terminals as type jointarget.Terminal {
25                   keep outgoing CIM.IEC61970.Core.Terminal.TieFlow as type jointarget.TieFlow {
26                      keep outgoing CIM.IEC61970.ControlArea.TieFlow.ControlArea as type jointarget.ControlArea {
```

```
                    keep attributes CIM.IEC61970.Core.IdentifiedObject.mRID
                 }
              }
           }
        }
     }
  }
}

theta join CIM.IEC61968.Metering.MeterAsset with COSEM.PhysicalDevice where "CIM.IEC61968.Metering.MeterAsset.mRID = COSEM.
     PhysicalDevice.ID" as jointarget.PrivateMeterVoltage {
   keep attributes COSEM.PhysicalDevice.ID
   keep calculated attribute "COSEM.PhysicalDevice.ElectricityValues.VoltageL1" as PrivateMeterVoltage.VoltageA:Double
   keep calculated attribute "COSEM.PhysicalDevice.ElectricityValues.VoltageL2" as PrivateMeterVoltage.VoltageB:Double
   keep calculated attribute "COSEM.PhysicalDevice.ElectricityValues.VoltageL3" as PrivateMeterVoltage.VoltageC:Double
   keep supertype CIM.IEC61968.Metering.EndDeviceAsset as type jointarget.EndDeviceAsset {
      keep outgoing CIM.IEC61968.Metering.EndDeviceAsset.ServiceDeliveryPoint as type jointarget.ServiceDeliveryPoint {
         keep outgoing CIM.IEC61968.Metering.ServiceDeliveryPoint.EnergyConsumer as type jointarget.EnergyConsumer {
            keep attributes CIM.IEC61970.Core.IdentifiedObject.mRID
            keep subtype CIM.IEC61970.LoadModel.ConformLoad as type jointarget.ConformLoad {
               keep outgoing CIM.IEC61970.LoadModel.ConformLoad.LoadGroup as type jointarget.ConformLoadGroup {
                  keep supertype CIM.IEC61970.LoadModel.LoadGroup as type jointarget.LoadGroup {
                     keep outgoing CIM.IEC61970.LoadModel.LoadGroup.SubLoadArea as type jointarget.SubLoadArea {
                        keep outgoing CIM.IEC61970.LoadModel.SubLoadArea.LoadArea as type jointarget.LoadArea {
                           keep outgoing CIM.IEC61970.LoadModel.EnergyArea.ControlArea as type jointarget.ControlArea
                        }
                     }
                  }
               }
            }
            keep subtype CIM.IEC61970.LoadModel.NonConformLoad as type jointarget.NonConformLoad {
               keep outgoing CIM.IEC61970.LoadModel.NonConformLoad.LoadGroup as type jointarget.NonConformLoadGroup {
                  keep supertype CIM.IEC61970.LoadModel.LoadGroup as type jointarget.LoadGroup
               }
            }
         }
      }
   }
}
```

Listing 2: Three Phase Measurement Matrix Viewtype

"Such analysis viewtypes can be used together with a basic network topology view to calculate the exact location of a failure. Phasors are traveling waves in the system, which means that the failure travels with the wave through the grid. Therefore it is important to find its origin. The topology viewtype includes the length of the transmission line segments. They can be used together with the timestamp of the measured phasor to calculate from where the wave came and where it was when the failure started. This is its origin [7]." [1]

From the ModelJoin view definition in Listing 2, the view type depicted in Figure 3 is generated.

The biggest difference to Task 1 besides the increased complexity is the fact that this view definition keeps subtypes of some model elements. If an energy consumer in a service delivery point is a `ConformLoad`, then the view computation should be different to the case when the energy consumer is a `NonConformLoad`.

4 Benchmark Framework

The benchmark framework is based on the benchmark framework of the TTC 2015 Train Benchmark case [8] and supports a generator of change sequences, automated build and execution of solutions as well the visualization of the results using R. The source code and documentation of the benchmark as well as metamodels, reference solutions in ModelJoin and QVT-O, example models and example change sequences are publicly available online at `http://github.com/georghinkel/ttc2017SmartGrids`.

The benchmark consists of the following phases:

1. **Initialization:** In this phase, solutions may load metamodels and other infrastructure independent of the used models as required. Because time measurements are very hard to measure for this phase, the time measurement is optional.

2. **Loading:** The initial model instances are loaded.

3. **Initial:** An initial view is created.

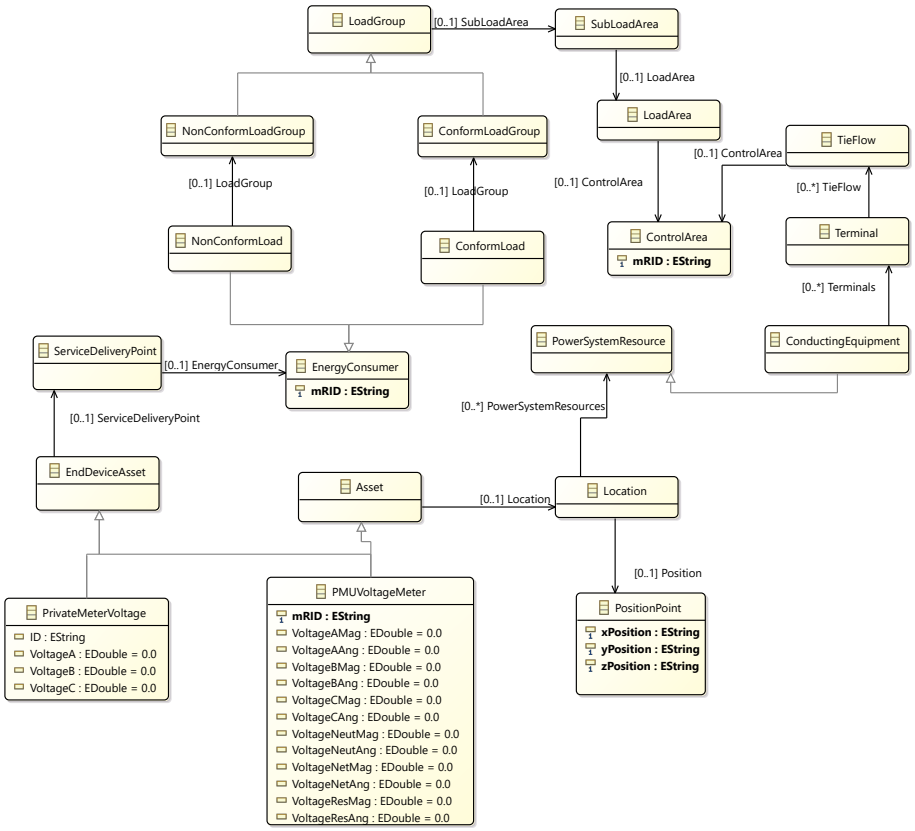


Figure 3: The Voltage Three-Phase Measurement Matrix

4. **Updates:** A sequence of change sequences is applied to the model. Each change sequence consists of several atomic change operations. After each change sequence, the view must be consistent with the changed source models, either by entirely creating the view from scratch or by propagating changes to the view result.

In the following subsections, the change sequences, solution requirements and the benchmark configuration are explained in more detail.

4.1 Change Sequences

To measure the incremental performance of solutions, the benchmark uses generated change sequences. These change sequences are in the `changes` directory of the benchmark resources. Additional change sequences can be generated using a generator contained in the benchmark resources, however the generator is implemented using NMF [9] and thus requires .NET Framework 4.5.1 to be installed.

The changes are available in the form of models. An excerpt of the metamodel is depicted in Figure 4: There are classes for each elementary change operation that distinguish between simple assignments and collection interactions, such as adding or removing single elements or resetting, which means erasing the collection contents. The true metamodel contains concrete classes that distinguish further between the type of feature, whether it is an attribute, association or composition change. In these concrete classes, the added, deleted or assigned items are included[1]. The change metamodel also supports change transactions where a source change implies some other changes, for example setting opposite references.

Unfortunately, the implementation to apply these changes is only available in NMF. Incremental tools are therefore asked to transform the change sequences into their own change representation. To ease the specification of batch solutions, the generator also outputs the models after each change sequence step.

[1] In a composite insertion, the added element is contained, otherwise only referenced.

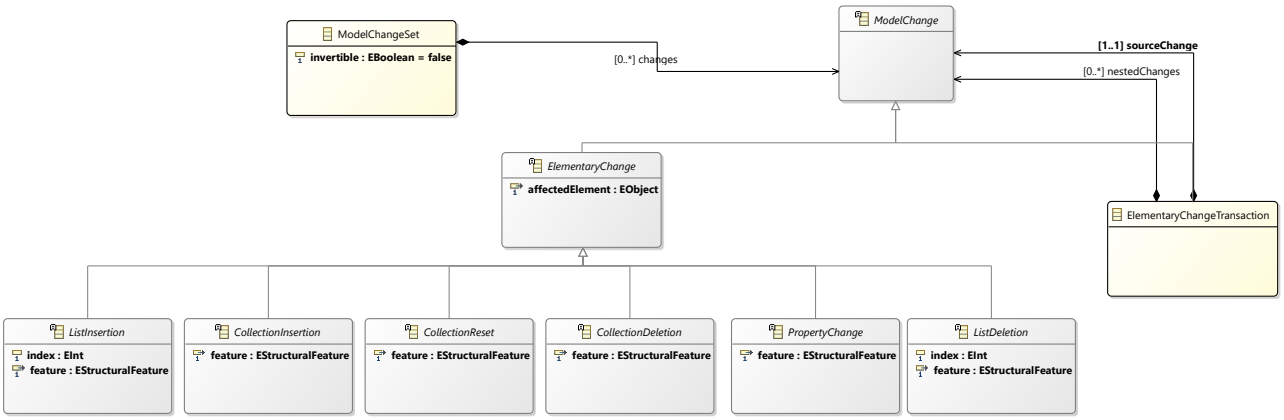


Figure 4: Metamodel of model changes (simplified)

4.2 Solution requirements

The solutions are required to perform the steps of the benchmark and report the following metrics after each step, in case of the update phase after every change sequence.

- **Tool:** The name of the tool.
- **View:** The view that is currently computed
- **ChangeSet:** The name of the change set that is currently run
- **RunIndex:** The run index in case the benchmark is repeated
- **Iteration:** The iteration (only required for the Update phase)
- **PhaseName:** The phase of the benchmark
- **MetricName:** The name of the reported metric
- **MetricValue:** The value of the reported metric

Solutions should report on the runtime of the respective phase in integer nanoseconds (**Time**), the working set in bytes (**Memory**) and the root element count in the created view (**Elements**). The memory measurement is optional. If it is done, it should report on the used memory after the given phase (or iteration of the update phase) is completed. Solutions are allowed to perform a garbage collection before memory measurement that does not have to be taken into account into the times. In the update phase, we are not interested in the time to load models or changes or the perhaps required transformation of changes, but only the pure view update, i.e. either recomputation of the view or propagation of the change.

To enable automatic execution by the benchmark framework, solutions should add a subdirectory to the `solutions` folder of the benchmark with a `solution.ini` file stating how the solution should be built and how it should be run. An example configuration for the ModelJoin solution is depicted in Listing 3. Because the solution contains the already compiled Jar archive, no action is required for build. However, solutions may want to run build tools like maven in this case to ensure the benchmark runs with the latest version.

```
1 [build]
2 default=echo ModelJoin solution is already compiled
3 skipTests=echo The ModelJoin solution is already compiled
4
5 [run]
6 OutageDetection=java -Xmx8G -jar solution.jar -view OutageDetection
7 OutagePrevention=java -Xmx8G -jar solution.jar -view OutagePrevention
```

Listing 3: An example `solution.ini` file

The repetition of executions as defined in the benchmark configuration is done by the benchmark. This means, for 5 runs, the specified command-line for a particular view will be called 5 times. These runs should all have the same prerequisites. In particular, solutions must not save intermediate data between different runs. Meanwhile, all iterations of the *Update* phase are executed in the same process and solutions are allowed (and encouraged) to save any intermediate computation results they like, as long as the results are correct after each change sequence.

The root path of the input models and changes, the run index, the number of iterations and the name of change sequences are passed using environment variables `ChangePath`, `RunIndex`, `Sequences` and `ChangeSet`. To demonstrate the usage of these environment variables, the benchmark framework also contains a demo solution which does nothing but print out a time csv entry using the provided environment variables.

4.3 Running the benchmark

The benchmark framework only requires Python 2.7 or above and R to be installed. R is required to create diagrams for the benchmark results. Furthermore, the solutions may imply additional frameworks. We would ask solution authors to explicitly note dependencies to additional frameworks necessary to run their solutions.

If all prerequisits are fulfilled, the benchmark can be run using Python with the command `python scripts/run.py`. Additional command-line options can be queried using the option `-help`.

```
{
  "Views": [
    "OutageDetection",
    "OutageAvoidance"
  ],
  "Tools": ["ModelJoin"],
  "ChangeSets": [
    "changeSequence1"
  ],
   "Sequences": 100
   "SequenceLength": 10
  "Runs": 5
}
```

Listing 4: The default benchmark configuration

The benchmark framework can be configured using JSON configuration files. The default configuration is depicted in Listing 4. In this configuration, both views are computed, using only the reference solution in ModelJoin, running the change sequence `changeSequence1` contained in the `changes` directory 5 times each. The exact commands created by the benchmark framework are determined using the solution configuration files described below.

To execute the chosen configuration, the benchmark can be run using the command line depicted in Listing 5.

```
&> python scripts/run.py
```

Listing 5: Running the benchmark

Additional commandline parameters are available to only update the measurements, create visualizations or generate new change sequences.

5 Evaluation

Solutions of the proposed benchmark should be evaluated by their completeness, correctness, conciseness, understandability, batch performance and incremental performance.

For each evaluation, a solution can earn 5 points for Task 1 and 5 points for Task 2. In the latter, we explain how the points are awarded for Task 1. The points for Task 2 are awarded equivalently.

5.1 Completeness & Correctness

Assessing the completeness and correctness of model transformations is a difficult task. In the scope of this benchmark, besides manual assessment by opponents, solutions are checked for the correct number of model elements in the result after each change.

Points are awarded according to the following rules:

- **0 points** The task is not solved.

- **1-4 points** The task is solved, but the number of elements in the result is either too high or too low.
- **5 points** The task is completely and correctly solved.

5.2 Conciseness

Detecting and especially forecasting an outage in a smart grid heavily relies on heuristics. Therefore, it is important to specify views in a concise manner.

- **0 points** The task is not solved.
- **1 point** The solution is the least concise.
- **5 points** The solution is the most concise.
- **1-5 points** All solutions in between are classified relative to the most and least concise solution.

To evaluate the conciseness, we ask every solution to note on the lines of code of their solution. This shall include the model views and glue code to actually run the benchmark. Code to convert the change sequence can be excluded. For any graphical part of the specification, we ask to count the lines of code in a HUTN[2] notation of the underlying model.

5.3 Understandability

Similarly to conciseness, it is important for maintainance tasks that the solution is understandable. However, as there is no appropriate metric for understandability available, the assessment of the understandability is done manually. For solutions participating in the contest, this score is collected using questionnaires at the workshop.

5.4 Performance

For the performance, we consider two scenarios: batch performance and incremental performance. For the batch performance, we measure the time the solution requires to create the view for existing models. For the incremental solution, we measure the time for the solution to propagate a given set of changes. Points are awarded according to the following rules:

- **0 points** The task is not solved.
- **1 point** The solution is the slowest.
- **5 points** The solution is the fastest.
- **1-5 points** All solutions in between are classified according to their speed.

The measurements for batch performance and for incremental performance are done separately. This means, solutions are allowed to run in a different configuration when competing for the batch performance than they use in the incremental setting. This is because in an application, usually only one of these aspects is particularly important.

5.5 Overall Evaluation

Due to their importance, the points awarded in completeness & correctness and understandability are doubled in the overall evaluation. Furthermore, due to the importance of incremental updates, we give the incremental performance a double weight.

Thus, each solution may earn up to 80 points in total (40 for each task).

Acknowledgements

We would like to thank Victoria Mittelbach for the permission to use her master thesis results for this case.

[2] `http://www.omg.org/spec/HUTN/`

References

[1] V. Mittelbach, "Model-driven Consistency Preservation in Cyber-Physical Systems," Master's thesis, Karlsruhe Institute of Technology (KIT), Germany.

[2] E. Burger, V. Mittelbach, and A. Koziolek, "Model-driven consistency preservation in cyber-physical systems," in *Proceedings of the 11th Workshop on Models@run.time co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016)*, (Saint Malo, France), CEUR Workshop Proceedings, 2016.

[3] "Iec 61970 energy management system application program interface (ems-api) - part 301 common information model (cim) base," 2011.

[4] "Iec 61850 communication networks and systems for power utility automation," 2015.

[5] D. U. Association, "Excerpt from companion specification for energy metering cosem interface classes and obis identification system," 2014.

[6] E. Burger, J. Henß, M. Küster, S. Kruse, and L. Happe, "View-Based Model-Driven Software Development with ModelJoin," *Software & Systems Modeling*, vol. 15, no. 2, pp. 472–496, 2014.

[7] L. Qianqian, X. Zeng, M. Xue, and L. Xiang, "A new smart distribution grid fault self-healing system based on traveling-wave," in *Industry Applications Society Annual Meeting, 2013 IEEE*, 2013, pp. 1–6.

[8] G. Szárnyas, O. Semeráth, I. Ráth, and D. Varró, "The TTC 2015 train benchmark case for incremental model validation," in *Proceedings of the 8th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences, L'Aquila, Italy, July 24, 2015.*, 2015, pp. 129–141.

[9] G. Hinkel, "NMF: A Modeling Framework for the .NET Platform," Karlsruhe Institute of Technology, Tech. Rep., 2016.