An NMF solution to the Smart Grid Case at the TTC 2017

Georg Hinkel

FZI FORSCHUNGSZENTRUM INFORMATIK

Sparse adoption of MDE in industry

- Tool support perceived insufficient [Sta06,Mo+13]
  - Much less manpower in tool development than IDEs such as Visual Studio, IntelliJ, …

- Developers hardly change their primary language [MR13]
  - Project requirements or code reuse

.NET Modeling Framework (NMF)

- Repository management in .NET
  - Generate code for metamodels
  - Load models
  - Save models
  - (Mostly) Compatible to EMF
- Multimode Model Synchronization
  - Incremental
  - Bidirectional
- Open source: http://github.com/NMFCode/NMF

NMF Synchronizations

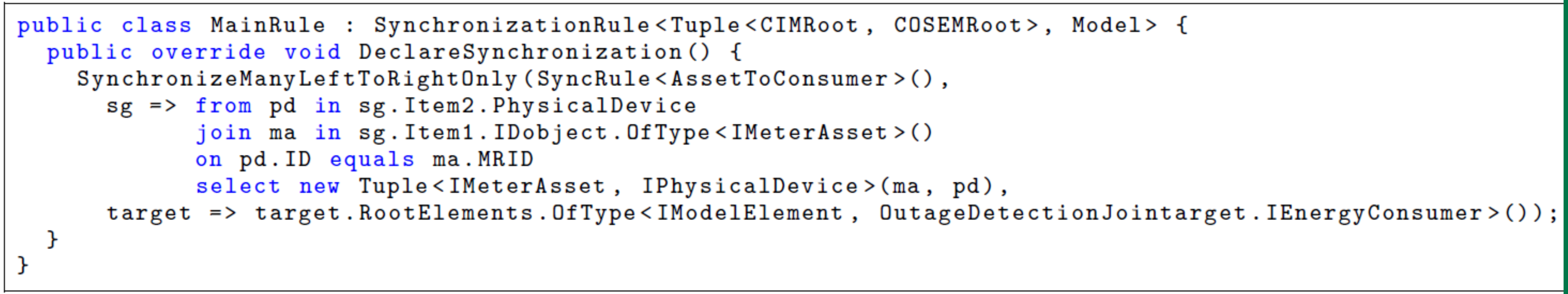
- Formal basis: Unidirectional Synchronization blocks

$$\begin{tikzcd}
A \arrow[r, Leftrightarrow, "\Phi_{A-C}"] \arrow[d, "f"'] & C \arrow[d, hook, "g"] \\
B \arrow[r, Leftrightarrow, "\Phi_{B-D}"'] & D
\end{tikzcd}$$

- Incremental model synchronization
  - Lens used for writing values
  - Incrementalization system used for incremental updates
  - Updates can be constructed directly for all types of changes
  - Synchronization is hippocratic

Joining Elements in the Outage Detection task

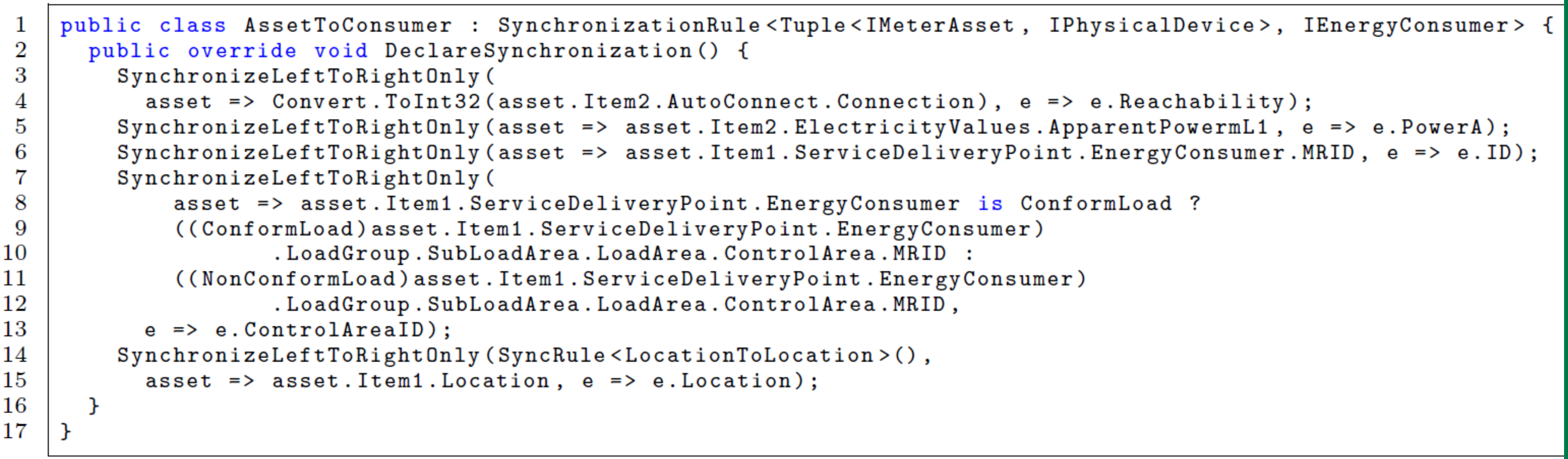
$$\begin{tikzcd}
CIMRoot \times COSEMRoot \arrow[r, leftrightarrow, "\Phi_{MainRule}"] \arrow[d, "(join)"'] & Model \arrow[d, hook, ".RootElements.OfType < EnergyConsumer >"] \\
(MeterAsset \times PhysicalDevice)* \arrow[r, leftrightarrow, "\Phi_{AssetToConsumer}"'] & EnergyConsumer*
\end{tikzcd}$$



```
public class MainRule : SynchronizationRule<Tuple<CIMRoot, COSEMRoot>, Model> {
  public override void DeclareSynchronization() {
    SynchronizeManyLeftToRightOnly(SyncRule<AssetToConsumer>(),
      sg => from pd in sg.Item2.PhysicalDevice
            join ma in sg.Item1.IDobject.OfType<IMeterAsset>()
            on pd.ID equals ma.MRID
            select new Tuple<IMeterAsset, IPhysicalDevice>(ma, pd),
      target => target.RootElements.OfType<IModelElement, OutageDetectionJointarget.IEnergyConsumer>());
  }
}
```

Synchronizing Element Content

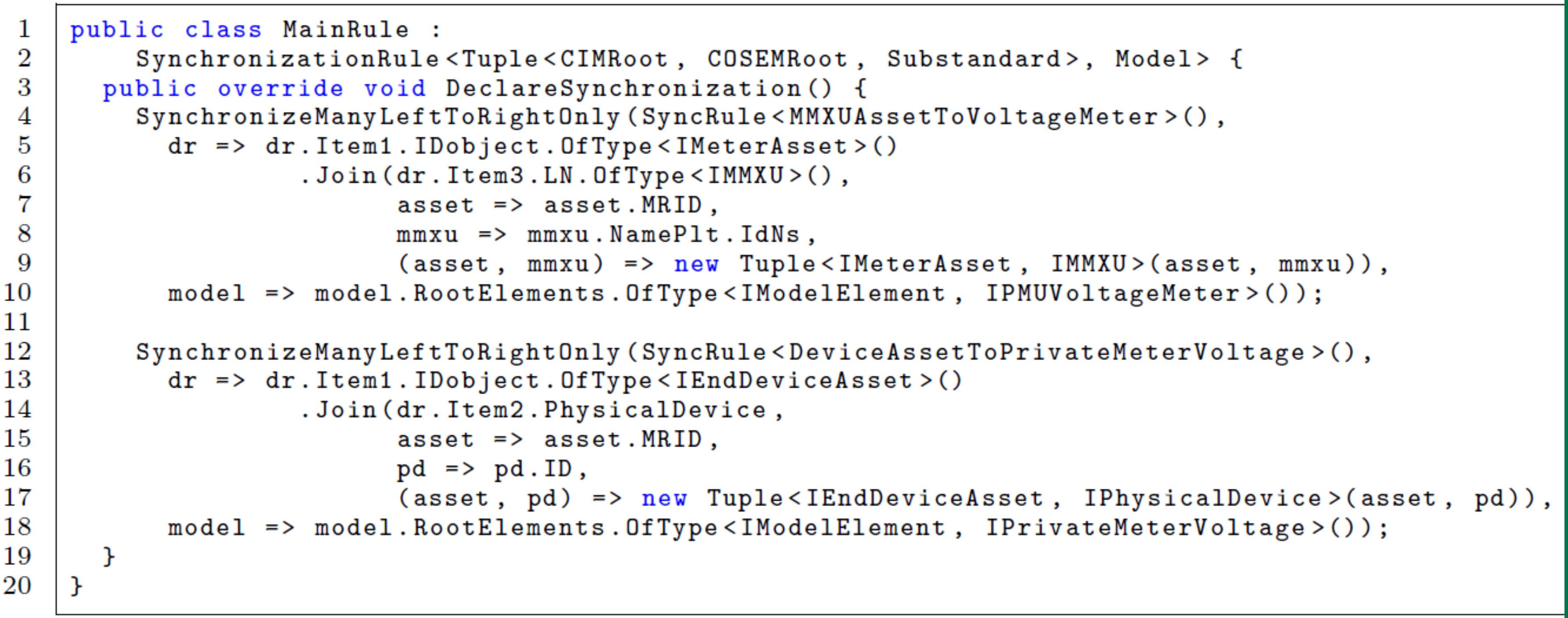
- To synchronize element contents, use further synchronization blocks
  - Simple synchronization blocks for attributes
  - Synchronization rules referring to other rules for references



```
public class AssetToConsumer : SynchronizationRule<Tuple<IMeterAsset, IPhysicalDevice>, IEnergyConsumer> {
  public override void DeclareSynchronization() {
    SynchronizeLeftToRightOnly(
      asset => Convert.ToInt32(asset.Item2.AutoConnect.Connection), e => e.Reachability);
    SynchronizeLeftToRightOnly(asset => asset.Item2.ElectricityValues.ApparentPowermL1, e => e.PowerA);
    SynchronizeLeftToRightOnly(asset => asset.Item1.ServiceDeliveryPoint.EnergyConsumer.MRID, e => e.ID);
    SynchronizeLeftToRightOnly(
        asset => asset.Item1.ServiceDeliveryPoint.EnergyConsumer is ConformLoad ?
        ((ConformLoad)asset.Item1.ServiceDeliveryPoint.EnergyConsumer)
               .LoadGroup.SubLoadArea.LoadArea.ControlArea.MRID :
        ((NonConformLoad)asset.Item1.ServiceDeliveryPoint.EnergyConsumer)
               .LoadGroup.SubLoadArea.LoadArea.ControlArea.MRID,
      e => e.ControlAreaID);
    SynchronizeLeftToRightOnly(SyncRule<LocationToLocation>(),
      asset => asset.Item1.Location, e => e.Location);
  }
}
```

Synchronization in the Outage Prevention task

- Two synchronization blocks, one for each of the queries

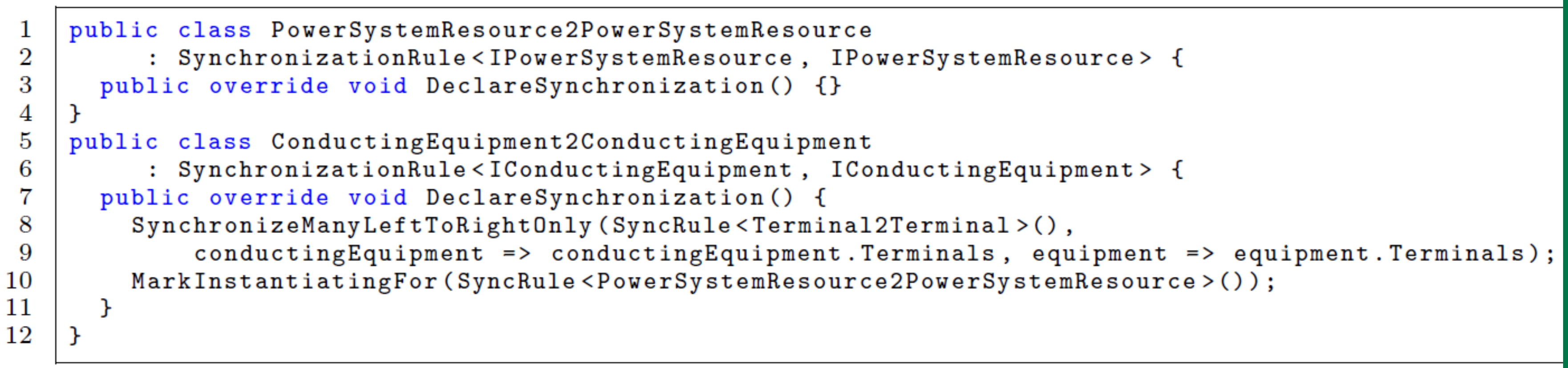


```
public class MainRule :
    SynchronizationRule<Tuple<CIMRoot, COSEMRoot, Substandard>, Model> {
  public override void DeclareSynchronization() {
    SynchronizeManyLeftToRightOnly(SyncRule<MMXUAssetToVoltageMeter>(),
      dr => dr.Item1.IDobject.OfType<IMeterAsset>()
              .Join(dr.Item3.LN.OfType<IMMXU>(),
                    asset => asset.MRID,
                    mmxu => mmxu.NamePlt.IdNs,
                    (asset, mmxu) => new Tuple<IMeterAsset, IMMXU>(asset, mmxu)),
      model => model.RootElements.OfType<IModelElement, IPMUVoltageMeter>());

    SynchronizeManyLeftToRightOnly(SyncRule<DeviceAssetToPrivateMeterVoltage>(),
      dr => dr.Item1.IDobject.OfType<IEndDeviceAsset>()
              .Join(dr.Item2.PhysicalDevice,
                    asset => asset.MRID,
                    pd => pd.ID,
                    (asset, pd) => new Tuple<IEndDeviceAsset, IPhysicalDevice>(asset, pd)),
      model => model.RootElements.OfType<IModelElement, IPrivateMeterVoltage>());
  }
}
```

Synchronization of inheritance Hierarchies

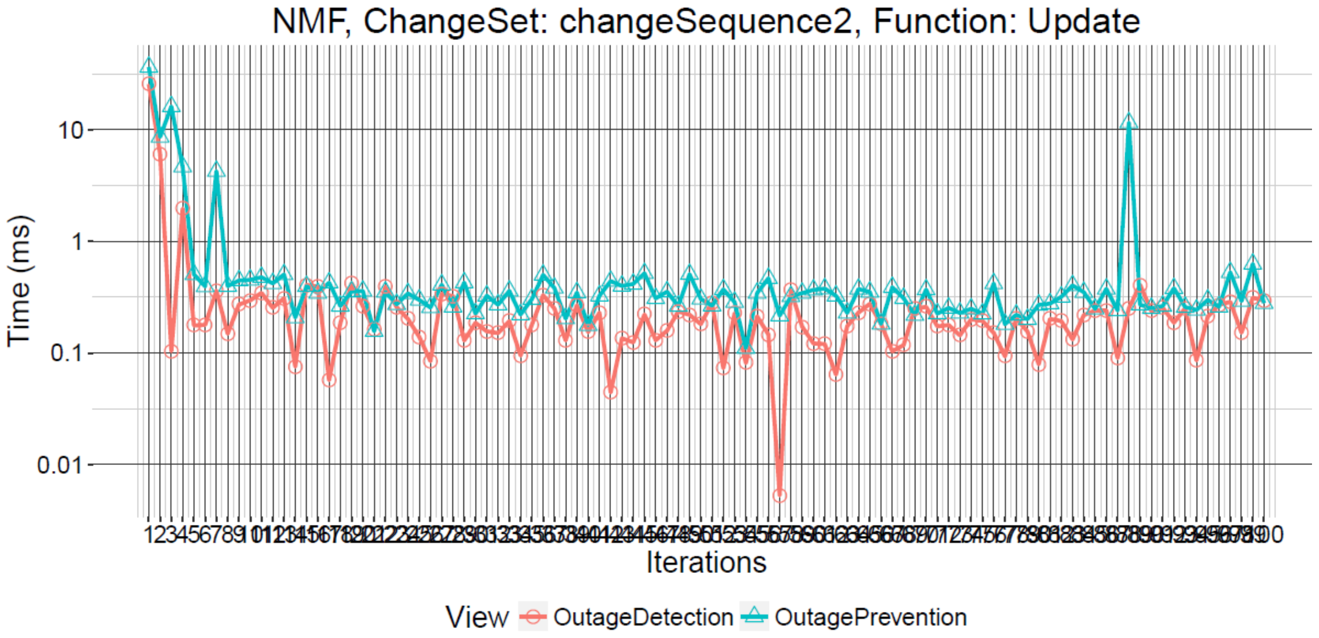
- Rule instantiation concept similar to rule inheritance in ATL



```
public class PowerSystemResource2PowerSystemResource
      : SynchronizationRule<IPowerSystemResource, IPowerSystemResource> {
  public override void DeclareSynchronization() {}
}
public class ConductingEquipment2ConductingEquipment
      : SynchronizationRule<IConductingEquipment, IConductingEquipment> {
  public override void DeclareSynchronization() {
    SynchronizeManyLeftToRightOnly(SyncRule<Terminal2Terminal>(),
        conductingEquipment => conductingEquipment.Terminals, equipment => equipment.Terminals);
    MarkInstantiatingFor(SyncRule<PowerSystemResource2PowerSystemResource>());
  }
}
```

Evaluation: Lines of Code

- Conciseness
  - 58 Lines of Code for Outage Detection
  - 195 Lines of Code for Outage Prevention
  - 140 Lines of Code to run the benchmark
- Performance in the range of milliseconds (standard laptop)



Conclusion

- Key advantages of the solution
  - Concise (about as concise as external languages)
  - Declarative incrementality
  - Correctness of the synchronization engine formally proven
  - Synchronization is hippocratic
  - Solution easily integrates into C# ➔ good tool support

- Drawbacks
  - No rule visualization available

hinkel@fzi.de

THANK YOU FOR YOUR ATTENTION