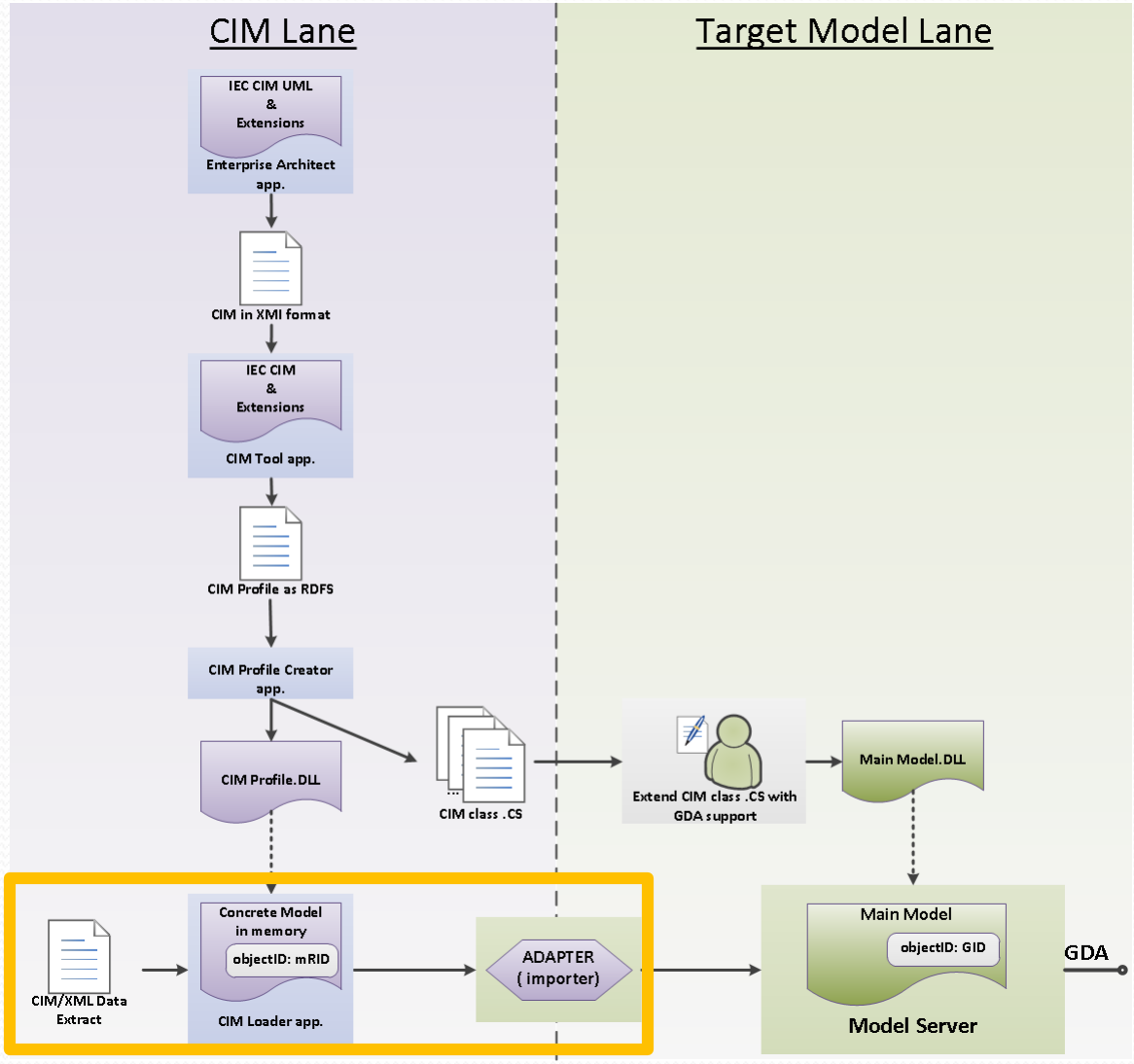
Standardi i modeliranje elektroenergetskih sistema

VEŽBA 9:

Implementacija adaptera za import CIM-baziranog modela podataka u Network Model Servis kroz GDA interfejs

Tok podataka pri inicijalizaciji modela elektroenergetske mreže



Adapter 1/3

- Transformiše CIM-XML file u *Delta* objekat
- Primenjuje *Delta* objekat na Network Model Servis kroz GDA interfejs

```
public Delta CreateDelta(Stream extract, SupportedProfiles extractType, out string log)...

public string ApplyUpdates(Delta delta)...
```

Adapter 2/3

- Metoda **`CreateDelta`**:
  - Load proces:
    - extract i profil koristi za kreiranje *ConcreteModel* instance
  - Transform proces:
    - na osnovu profila poziva odgovarajući importer
    - importer transformiše sadržaj *ConcreteModel* objekta u *Delta* objekat

```
public Delta CreateDelta(Stream extract, SupportedProfiles extractType, out string log)
{
    Delta nmsDelta = null;
    ConcreteModel concreteModel = null;
    Assembly assembly = null;
    string loadLog = string.Empty;
    string transformLog = string.Empty;

    if (LoadModelFromExtractFile(extract, extractType, ref concreteModel, ref assembly, out loadLog))
    {
        TransformModel(assembly, concreteModel, extractType, out nmsDelta, out transformLog);
    }
    log = string.Concat("Load report:\r\n", loadLog, "\r\nTransform report:\r\n", transformLog);

    return nmsDelta;
}
```

Adapter 3/3

- Metoda **`ApplyDelta`**:
  - koristi GDA interfejs ka Network Model Service-u
  - pripremljen *Delta* objekat prosleđuje na primenu

```
public string ApplyUpdates(Delta delta)
{
    string updateResult = "Apply Updates Report:\r\n";
    System.Globalization.CultureInfo culture = Thread.CurrentThread.CurrentCulture;
    Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");
    if ((delta != null) && (delta.NumberOfOperations != 0))
    {
        //// TO BE ADDED: NetworkModelService->ApplyUpdates
        //// updateResult = MMSHandler.ApplyUpdates(electricDelta);
    }
    Thread.CurrentThread.CurrentCulture = culture;
    return updateResult;
}
```

Importer 1/7

- Importer je implementiran za određeni CIM profil.
- Ima znanje kako se elementi definisani u profilu mapiraju na objekte ciljanog DMS modela:
  - mapiranje CIM klasa na DMS klasu
  - mapiranje CIM atribut na DMS atribut
  - Pažnja: mapiranje među modelima ne mora da bude 1-na-1!
- Implementira proces koji transformiše sadržaj *ConcreteModel* objekta u *Delta* objekat

Importer 2/7

- Metoda **`CreateNMSDelta`**:
  - poziva konverziju

```
public TransformAndLoadReport CreateNMSDelta(ConcreteModel cimConcreteModel)
{
    LogManager.Log("Importing PowerTransformer Elements...", LogLevel.Info);
    report = new TransformAndLoadReport();
    concreteModel = cimConcreteModel;
    delta.ClearDeltaOperations();

    if ((concreteModel != null) && (concreteModel.ModelMap != null))
    {
        try
        {
            // convert into DMS elements
            ConvertModelAndPopulateDelta();
        }
        catch (Exception ex)
        {
            string message = string.Format("{0} - ERROR in data import - {1}", DateTime.Now, ex.Message);
            LogManager.Log(message);
            report.Report.AppendLine(ex.Message);
            report.Success = false;
        }
    }
    LogManager.Log("Importing PowerTransformer Elements - END.", LogLevel.Info);
    return report;
}
```

Importer 3/7

- Konverzija modela:
  - kreiranje *ResourceDescription* instanci na osnovu CIM objekata
  - popunjavanje *Property*-a u okviru svakog *ResourceDescription* objekta
  - vrednost referenci medju objektima: bitan je redosled kojim se generišu globalni identifikatori

```
private void ConvertModelAndPopulateDelta()
{
    LogManager.Log("Loading elements and creating delta...", LogLevel.Info);

    //// import all concrete model types (DMSType enum)
    ImportBaseVoltages();
    ImportLocations();
    ImportPowerTransformers();
    ImportTransformerWindings();
    ImportWindingTests();

    LogManager.Log("Loading elements and creating delta completed.", LogLevel.Info);
}
```

Importer 4/7

- **`Import<Type>`** metode:
  - selektuju listu CIM objekata datog tipa iz *ConcreteModel*-a
  - za svaki objekat iz liste (mape) kreira se *ResourceDescription* instanca i popunjava sa *Property* podacima

```
private void Import<Type>()
{
   SortedDictionary<string, object> cimObjects = concreteModel.GetAllObjectsOfType("FTN.Type");
   if (cimObjects != null)
   {
      foreach (KeyValuePair<string, object> cimObjectPair in cimObjects)
      {
         FTN.Type cimObject = cimObjectPair.Value as FTN.Type;
         ResourceDescription rd = CreateTypeResourceDescription(cimObject);
         if (rd != null)
         {
            delta.AddDeltaOperation(DeltaOpType.Insert, rd, true);
            report.Report.Append("Type ID = ").Append(cimObject.ID).Append(" SUCCESSFULLY
converted to GID = ").AppendLine(rd.Id.ToString());
         }
      }
   }
}
```

Importer 5/7

- Prilikom transformacije u *ResourceDescription* potrebno je:
  - generisati GID
  - upamtiti mapiranje CIM identifikatora na GID

} *ImportHelper*

```
public class ImportHelper
{
    private Dictionary<DMSType, int> typeCounter = new Dictionary<DMSType, int>();
    private Dictionary<string, long> rdfIDtoGIDMapping = new Dictionary<string, long>();


    /// <summary> ...
    public int CheckOutIndexForDMSType(DMSType dmsType)...

    /// <summary> ...
    public void DefineIDMapping(string rdfID, long gid)...

    /// <summary> ...
    public long GetMappedGID(string rdfID)...
}
```

Importer 6/7

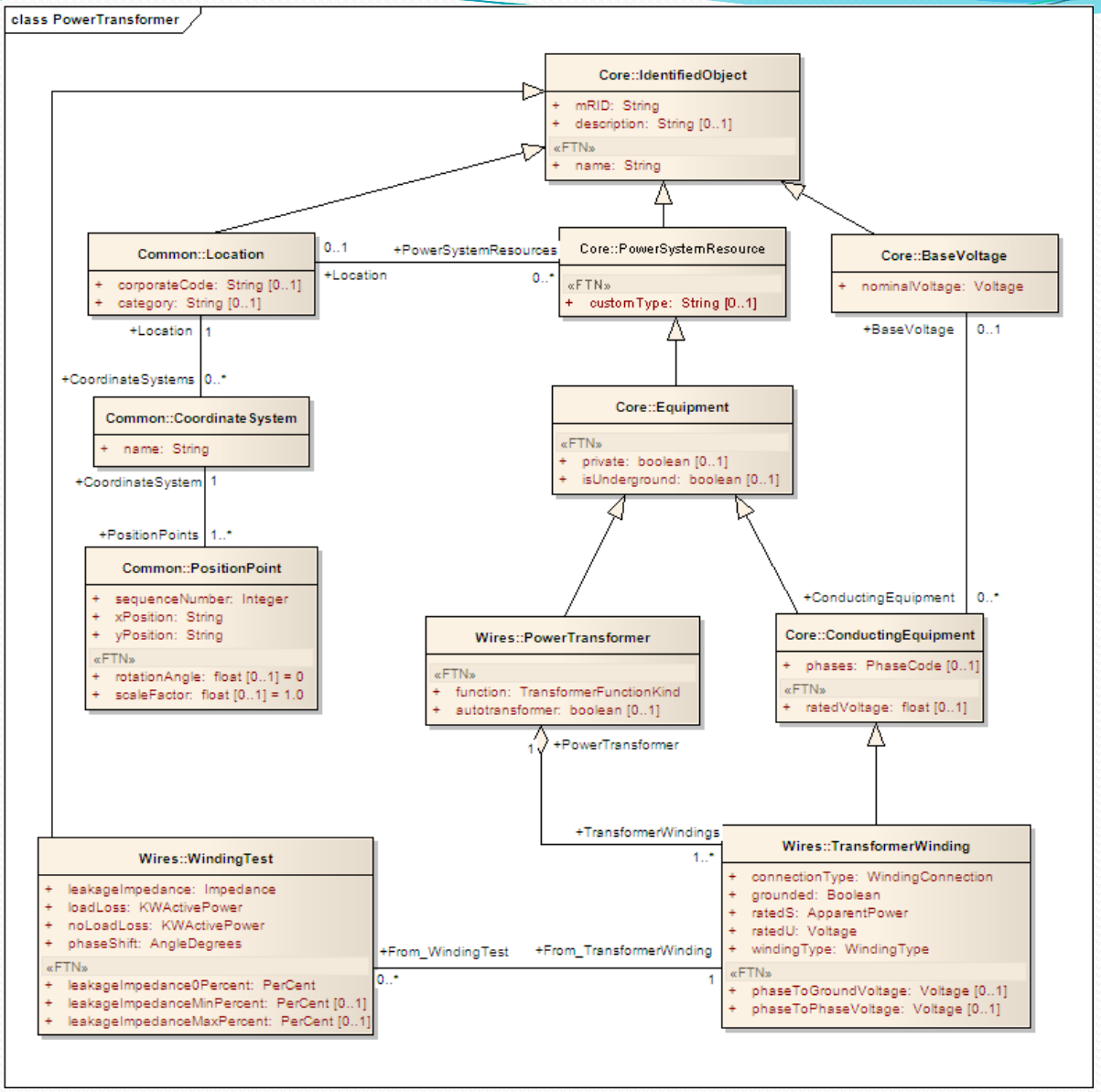
- mapirati atribute CIM objekta na ModelCode
- ispravno postaviti vrednost svakog Property-ja

Converter

```
public static class PowerTransformerConverter
{

    #region Populate ResourceDescription
    public static void PopulateIdentifiedObjectProperties(FTN.IdentifiedObject cimIdentifiedObject, ResourceDescription rd)...

    public static void PopulateLocationProperties(FTN.Location cimLocation, ResourceDescription rd)...

    public static void PopulatePowerSystemResourceProperties(FTN.PowerSystemResource cimPowerSystemResource, ResourceDescription rd,

    public static void PopulateBaseVoltageProperties(FTN.BaseVoltage cimBaseVoltage, ResourceDescription rd)...

    public static void PopulateEquipmentProperties(FTN.Equipment cimEquipment, ResourceDescription rd, ImportHelper importHelper, Tra

    public static void PopulateConductingEquipmentProperties(FTN.ConductingEquipment cimConductingEquipment, ResourceDescription rd,

    public static void PopulatePowerTransformerProperties(FTN.PowerTransformer cimPowerTransformer, ResourceDescription rd, ImportHel

    public static void PopulateTransformerWindingProperties(FTN.TransformerWinding cimTransformerWinding, ResourceDescription rd, Imp

    public static void PopulateWindingTestProperties(FTN.WindingTest cimWindingTest, ResourceDescription rd, ImportHelper importHelpe
    #endregion Populate ResourceDescription

    #region Enums convert
    public static PhaseCode GetDMSPhaseCode(FTN.PhaseCode phases)...

    public static TransformerFunction GetDMSTransformerFunctionKind(FTN.TransformerFunctionKind transformerFunction)...

    public static WindingType GetDMSWindingType(FTN.WindingType windingType)...

    public static WindingConnection GetDMSWindingConnection(FTN.WindingConnection windingConnection)...
    #endregion Enums convert
}
```

Importer 7/7

```
public static void PopulatePowerSystemResourceProperties(FTN.PowerSystemResource cimPowerSystemResource,
                                                          ResourceDescription rd, ImportHelper importHelper)
{
    if ((cimPowerSystemResource != null) && (rd != null))
    {
        PowerTransformerConverter.PopulateIdentifiedObjectProperties(cimPowerSystemResource, rd);

        if (cimPowerSystemResource.CustomTypeHasValue)
        {
            rd.AddProperty(new Property(ModelCode.PSR_CUSTOMTYPE, cimPowerSystemResource.CustomType));
        }
        if (cimPowerSystemResource.LocationHasValue)
        {
            long gid = importHelper.GetMappedGID(cimPowerSystemResource.Location.ID);
            rd.AddProperty(new Property(ModelCode.PSR_LOCATION, gid));
        }
    }
}
```


class PowerTransformer
Core::IdentifiedObject
+ mRID: String
+ description: String [0..1]
«FTN»
+ name: String
Common::Location
+ corporateCode: String [0..1]
+ category: String [0..1]
0..1
+PowerSystemResources
+Location
0..*
Core::PowerSystemResource
«FTN»
+ customType: String [0..1]
Core::BaseVoltage
+ nominalVoltage: Voltage
+Location
1
+BaseVoltage
0..1
+CoordinateSystems
0..*
Common::CoordinateSystem
+ name: String
Core::Equipment
«FTN»
+ private: boolean [0..1]
+ isUnderground: boolean [0..1]
+CoordinateSystem
1
+PositionPoints
1..*
Common::PositionPoint
+ sequenceNumber: Integer
+ xPosition: String
+ yPosition: String
«FTN»
+ rotationAngle: float [0..1] = 0
+ scaleFactor: float [0..1] = 1.0
+ConductingEquipment
0..*
Wires::PowerTransformer
«FTN»
+ function: TransformerFunctionKind
+ autotransformer: boolean [0..1]
Core::ConductingEquipment
+ phases: PhaseCode [0..1]
«FTN»
+ ratedVoltage: float [0..1]
1
+PowerTransformer
+TransformerWindings
1..*
Wires::WindingTest
+ leakageImpedance: Impedance
+ loadLoss: KWActivePower
+ noLoadLoss: KWActivePower
+ phaseShift: AngleDegrees
«FTN»
+ leakageImpedance0Percent: PerCent
+ leakageImpedanceMinPercent: PerCent [0..1]
+ leakageImpedanceMaxPercent: PerCent [0..1]
+From_WindingTest
+From_TransformerWinding
0..*
1
Wires::TransformerWinding
+ connectionType: WindingConnection
+ grounded: Boolean
+ ratedS: ApparentPower
+ ratedU: Voltage
+ windingType: WindingType
«FTN»
+ phaseToGroundVoltage: Voltage [0..1]
+ phaseToPhaseVoltage: Voltage [0..1]

Zadaci

1. U okviru klase *PowerTransformerImporter.cs* podržati import CIM tipova:
    1. TransformerWinding
    2. WindingTest