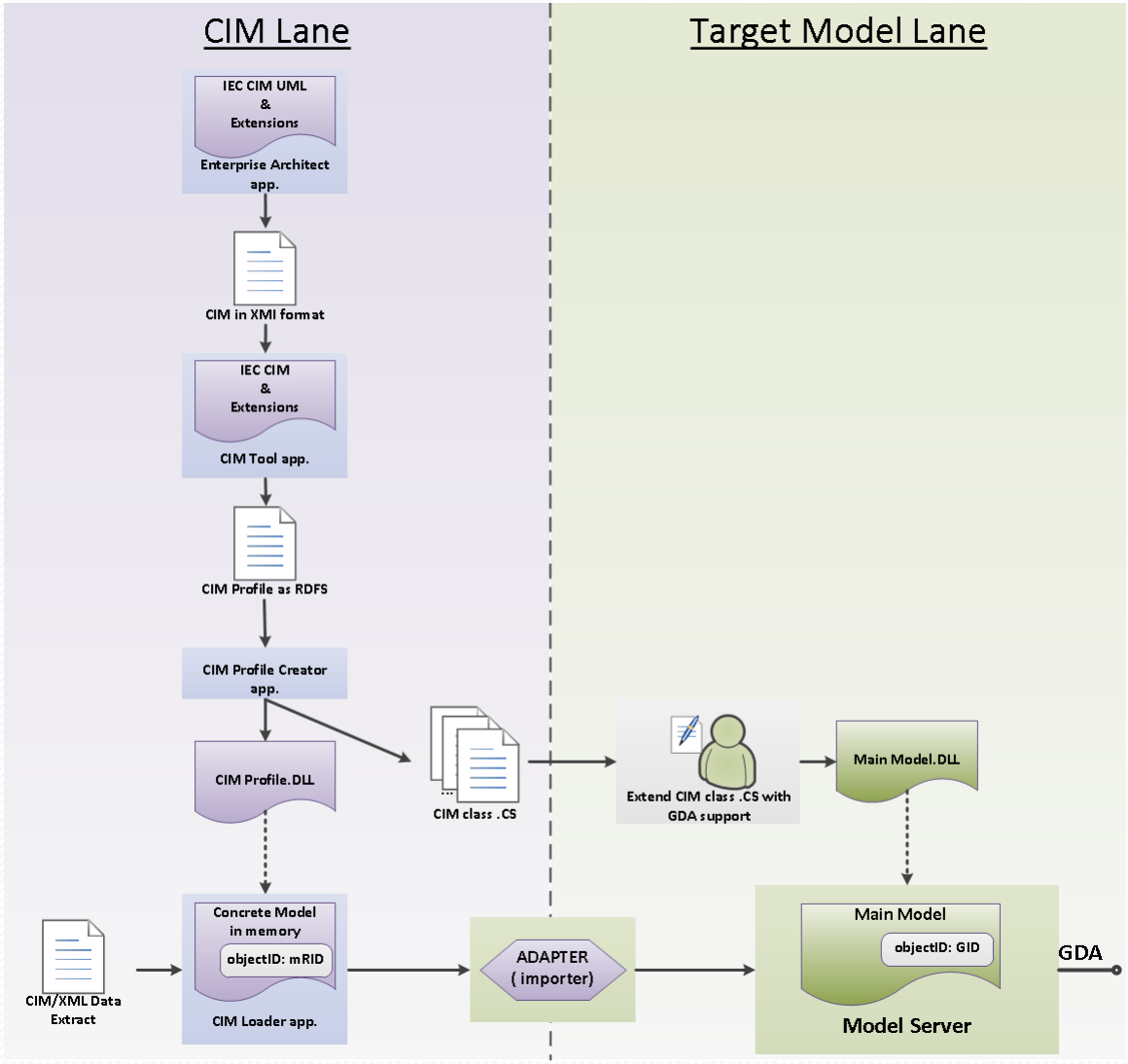
Standardi i modeliranje elektroenergetskih sistema

VEŽBA 11:

Dodavanje nove klase (bez dodatnih referenci) u Network Model Service

Tok podataka pri inicijalizaciji modela elektroenergetske mreže



Osnovne informacije o Network Model Service-u (NMS) 1/2

- *Expose*-uje *WCF* interfejs *INetworkModelGDAContract* na adresi *net.tcp://localhost:10000/NetworkModelService/GDA/*
- Podešavanja vezana za *binding* i *serviceBehavior* nalaze se u konfiguracionom fajlu servisa
- Obezbeđuje log; nivo logovanja i ime log fajla je moguće moguće menjati u konfiguracionom fajlu
- **Zahtevi za izmenom modela (*delta* objekte) koji stignu na servis čuvaju se u posebnoj datoteci definisanoj u konfiguracionom fajlu. Ukoliko dođe do spuštanja servisa, pri narednom podizanju učitavaju se svi podaci koji su do tada pristigli od strane adaptera.**

Osnovne informacije o Network Model Service-u (NMS) 2/2

- Deo konfiguracionog fajla servisa

```
<system.diagnostics>
  <trace autoflush="true">
    <listeners>
      <add type="System.Diagnostics.TextWriterTraceListener" name="TextWriter" initializeData="../NetworkModelService.log" />
    </listeners>
  </trace>
  <switches>
    <!-- 0 - Disabled
         1 = Error   - Gives error messages
         2 = Warning - Gives errors and warnings
         3 = Info    - Gives more detailed error information
         4 = Verbose - Gives verbose trace information.    -->
    <add name="TraceLevel" value = "Info" />
  </switches>
</system.diagnostics>

<connectionStrings>
  <add name="networkModelconnectionString"  connectionString="../NetworkModelData.data"/>
</connectionStrings>
```

Pravila implementacije Common-a 1/4

- Učitava ga svaka aplikacija u sistemu
- Opisuje model koji postoji na NMS (sadrži metapodatke o modelu)
- *CommonTrace* klasa se koristi za logovanje stanja aplikacije: statičkoj metodi *WriteTrace()* prosleđuje se nivo log-a i poruka koju želimo da zapišemo.

```
string message = "Starting Network Model Serivice...";
CommonTrace.WriteTrace(CommonTrace.TraceInfo, message);
```

- *ModelDefines.cs* definiše SVE *ModelCode*-ove *i DMSType*-ove (svi koji postoje su navedeni u tom .cs fajlu)

- Enums.cs sadrži definicije svih enumeracija koje postoje u modelu. Ovo su enumeracije koje predstavljaju tip nekog atributa klase u modelu.

```
using System;

namespace FTN.Common
{
    public enum PhaseCode : short...

    public enum TransformerFunction : short...

    public enum WindingConnection : short...

    public enum WindingType : short
    {
        None = 0,
        Primary = 1,
        Secondary = 2,
        Tertiary = 3
    }
}
```

Pravila implementacije Common-a 2/4

- *EnumDescs* klasa opisuje mapiranje enumeracija i atributa određene klase. Ukoliko je atribut neke klase tipa enimeracija potrebno je dodati mapiranje između *ModelCoda*-a koji je dodeljen tom atributu i tipa enumeracije.

```
public class EnumDescs
{
    private Dictionary<ModelCode, Type> property2enumType = new Dictionary<ModelCode, Type>();

    public EnumDescs()
    {
        property2enumType.Add(ModelCode.CONDEQ_PHASES, typeof(PhaseCode));
        property2enumType.Add(ModelCode.POWERTR_FUNC, typeof(TransformerFunction));
        property2enumType.Add(ModelCode.POWERTRWINDING_CONNTYPE, typeof(WindingConnection));
        property2enumType.Add(ModelCode.POWERTRWINDING_WINDTYPE, typeof(WindingType));
    }
...
```

- Klasa *PowerTransformer* sadrži atribut *function* koji je tipa enumeracija *TransformerFunction*.
EnumDescs **MORA** da definiše mapiranje *ModelCode.POWERTR_FUNC* na tip *TransformerFunction*.

```
public class PowerTransformer : Equipment
{
    private bool autotransformer = false;

    private TransformerFunction function;

    private List<long> transformerWindings = new List<long>();

    public PowerTransformer(long globalId)
        : base(globalId)
    {
    }
```

Pravila implementacije Common-a 3/4

- *ModelResourcesDesc* klasa obezbeđuje niz metoda za manipulaciju *ModelCode*-ovima na osnovu informacija koje su definisane u samoj vrednosti *ModelCode*-a (nasleđivanje, da li je klasa apstraktna ili ne, tip podatka, itd.):
  - *public static ModelCode FindFirstParent(ModelCode typeId)* – vraća *ModelCode* roditeljske klase
  - *public static bool InheritsFrom(ModelCode parentModelCode, ModelCode childModelCode)* – da li se *parentModelCode* nalazi bilo gde u hijararhiji nasleđivanja *childModelCode*-a
  - *public List<ModelCode> GetAllPropertyIds(ModelCode code)* – vraća *ModelCode*-ove koji su dodeljeni atributima neke klase.
  - …
- *ModelResourcesDesc* klasa definiše listu atributa (propertija) čija vrednost NE SME da se postavi od strane klijenta. Vrednosti ovih atributa određuje servis, a ukoliko klijent pokuša da postavi neku od ovih vrednost(kroz *Delta* objekat) servis će to odbiti. Pored globalnog identifikatora ovde se navode svi atributi koji su tipa lista referenci.

HashSet<ModelCode> notSettablePropertyIds – - sadrži *ModelCode*-ove atributa čije vrednosti određuje isključivo servis.

```
private void InitializeNotSettablePropertyIds()
{
    notSettablePropertyIds.Add(ModelCode.IDOBJ_GID);
    notSettablePropertyIds.Add(ModelCode.BASEVOLTAGE_CONDEQS);
    notSettablePropertyIds.Add(ModelCode.LOCATION_PSRS);
    notSettablePropertyIds.Add(ModelCode.POWERTRWINDING_TESTS);
}
```

Pravila implementacije Common-a 4/4

- Jedan od najvažnijih zadataka *ModelResourcesDesc* klase jeste da definiše kojim redosledom će se izvršavati operacije koje stignu na servis kao *Delta* objekat (insert, update, delete).
- Kada na servis stigne *Delta* objekat radi izmene modela (najčešće od strane adaptera) potrebano je da servis sortira operacije po tipu entiteta. Servis čita redosled iz *ModelResourcesDesc* klase i vrši soritranje pre primene izmena.
  private List<ModelCode> typeIdsInInsertOrder – definiše redosled sortiranja.

```
private void InitializeTypeIdsInInsertOrder()
{
    typeIdsInInsertOrder.Add(ModelCode.BASEVOLTAGE);
    typeIdsInInsertOrder.Add(ModelCode.LOCATION);
    typeIdsInInsertOrder.Add(ModelCode.POWERTR);
    typeIdsInInsertOrder.Add(ModelCode.POWERTRWINDING);
    typeIdsInInsertOrder.Add(ModelCode.WINDINGTEST);
}
```

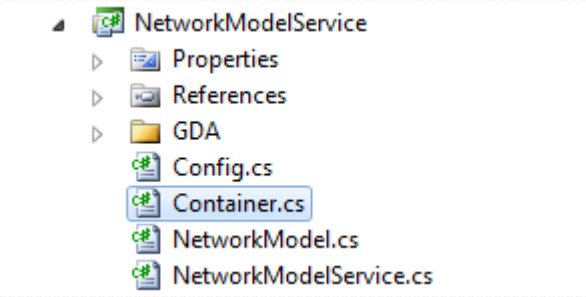
- Lista **MORA** da sadrži *ModelCode*-ove svih konkretnih klasa.
- Pravilo za formiranje redosleda ModelCode-ova u ovoj listi je sledeće:
  Ukoliko klasa A ima atribut tipa referenca čija je vrednost globalni identifikator klase B, tada klasa B ima atribut tipa lista referenci čija je vrednost lista globalnih identifikatora klase A. Tada kažemo da klasa A referencira klasu B, dok klasu B nazivamo target.
  U listi **UVEK** mora da se navede target (klasa B) pre klase koja ga referencira (klasa A).

Implementacija Network Model Service-a 1/5

- Klasa *Container* grupiše entitete istog tipa.
- Sadrži mapiranje globalnih idenitifikatora na njihove entitete

```
public class Container
{
    /// <summary>
    /// The dictionary of entities. Key = GlobaId, Value = Entity
    /// </summary>
    private Dictionary<long, IdentifiedObject> entities = new Dictionary<long, IdentifiedObject>();

    /// <summary>
    /// Initializes a new instance of the Container class
    /// </summary>
    public Container()...
```

- Zadužena je za kreiranje i pribavljanje entiteta koji ima odgovarajući globalni identifikator



Implementacija Network Model Service-a 2/5

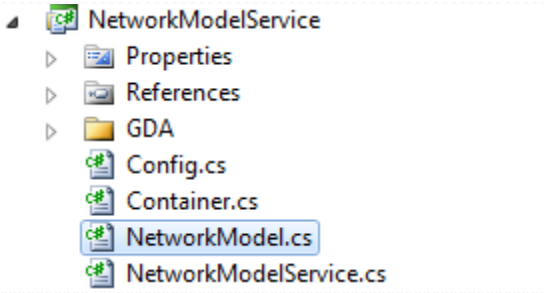
- Metoda *CreateEntity()* klase *Container* MORA da obezbedi kreiranje bilo koje konkretne klase na osnovu globalnog identifikatora. Iz globalnog identifikatora se "izvuče" tip entiteta i kreira odgovarajući entitet.
- Ukoliko se u model dodaje nova konkretna klasa potrebno je proširiti *CreateEntity()* metodu kako bi klasa *Container* postala „svesna" novog tipa entieta.

```
/// <summary> ...
public IdentifiedObject CreateEntity(long globalId)
{
    short type = ModelCodeHelper.ExtractTypeFromGlobalId(globalId);

    IdentifiedObject io = null;
    switch ((DMSType)type)
    {
        case DMSType.BASEVOLTAGE:
            io = new BaseVoltage(globalId);
            break;

        case DMSType.LOCATION:
            io = new Location(globalId);
            break;
        case DMSType.POWERTR:
            io = new PowerTransformer(globalId);
            break;
        case DMSType.POWERTRWINDING:
            io = new TransformerWinding(globalId);
            break;
        case DMSType.WINDINGTEST:
            io = new WindingTest(globalId);
            break;

        default:
```

Implementacija Network Model Service-a 3/5

- *NetworkModel* singleton klasa sadrži instance *Container* klase za svaki tip entiteta.

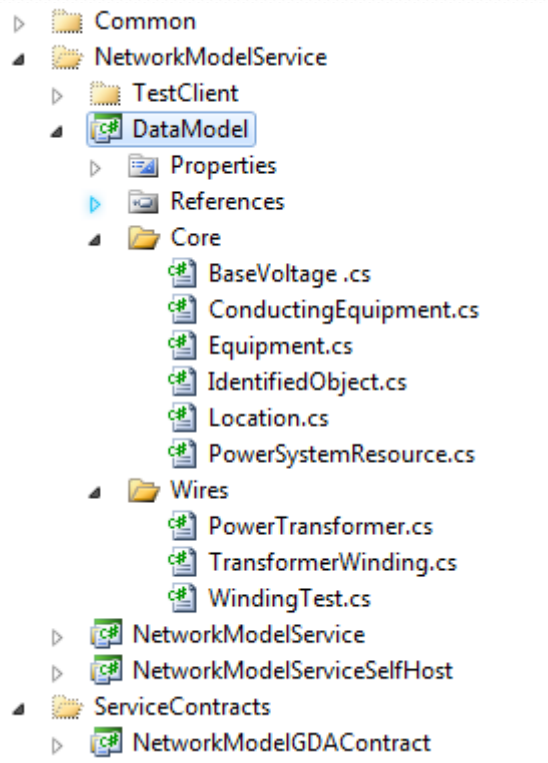
```
public class NetworkModel
{
    /// <summary>
    /// Dictionaru which contains all data: Key - DMSType, Value - Container
    /// </summary>
    private Dictionary<DMSType, Container> networkDataModel;
```



- Ukoliko je potrebno kreirati ili pribaviti neki entitet *NetworkModel* klasa pronalazi odgovarajuću instancu klase *Container* i prosleđuje joj zahtev.
- Implementacije *GDA* metoda nalaze se u *NetworkModel* klasi

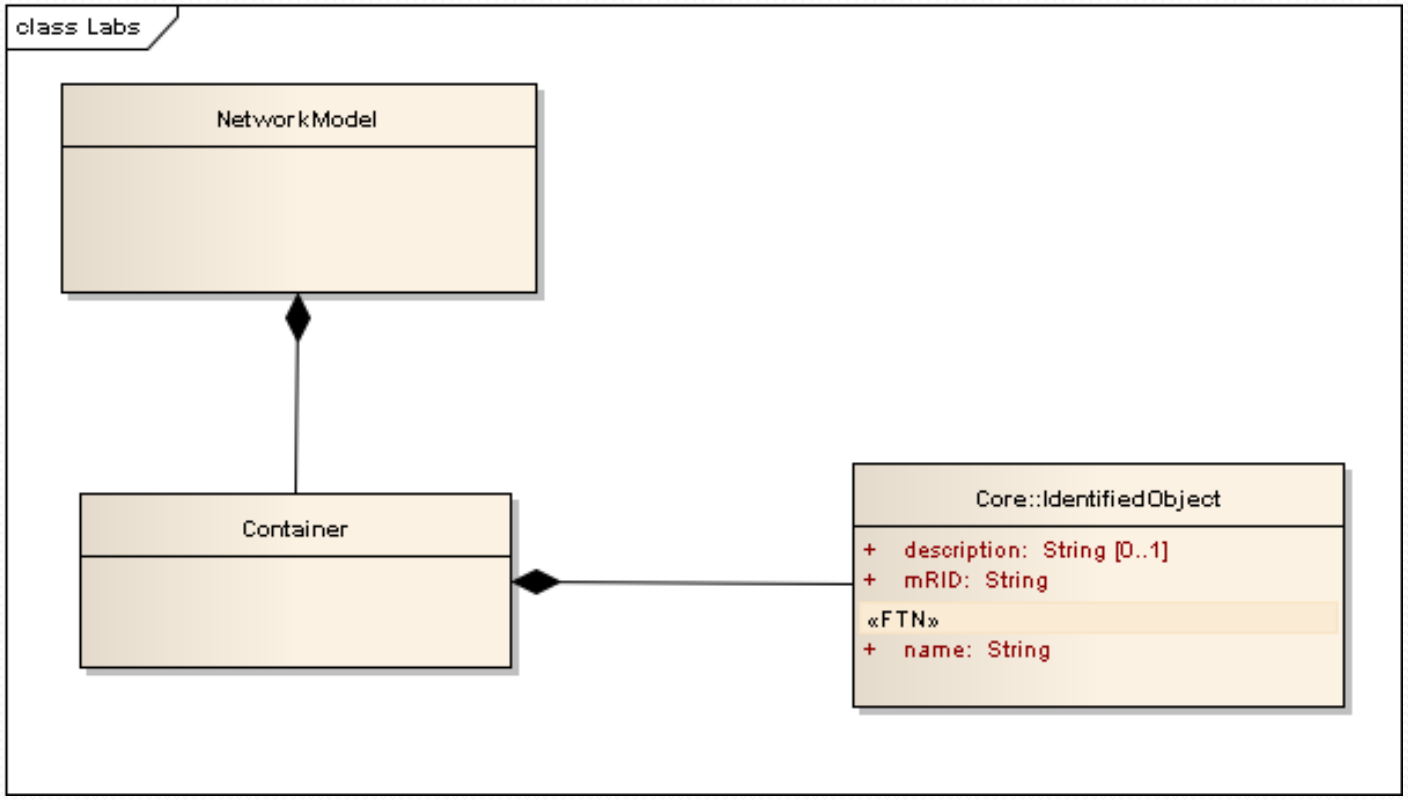
Implementacija Network Model Service-a 4/5

- *DataModel* projekat sadrži implementaciju samih klasa koje ulaze u model.
- Svaka klasa ima striktno definisanu strukturu: metode koje treba da implementira i kako da ih implementira. Na ovaj način bilo koja klasa (koja prati definisani način implementacije) može se dodati u model i servis će znati da manipuliše njom. (PODSETNIK: Ukoliko je klasa konkretna potrebno je proširiti *CreateEntity()* metodu klase *Container* kako bi servis mogao da kreira tu klasu)



Implementacija Network Model Service-a 5/5

- UML dijagram – organizacija *Network Model Service*-a



Struktura DataModel klasa 1/6

- Potrebno je implementirati nasleđivanje na osnovu definisanog modela
- Klasa sadrži *private* atribute definisane modelom i za svaki od njih *public* propertije (NAPOMENA: U ovom slučaju se misli na C# propertije neke klase, a ne *Property* objekte koje koristi GDA standard)
- Svaka klasa implementira konstruktor koji kao parametar prima globalni identifikator

```
public class PowerTransformer : Equipment
{
    private bool autotransformer = false;

    private TransformerFunction function;

    private List<long> transformerWindings = new List<long>();

    public PowerTransformer(long globalId)
        : base(globalId)
    {
    }

    public bool Autotransformer
    {
        get { return autotransformer;}
        set { autotransformer = value;}
    }

    public TransformerFunction Function
    {
        get { return function;}
        set { function = value;}
    }

    public List<long> TransformerWindings
    {
        get { return transformerWindings;}
        set { transformerWindings = value;}
    }
```

Struktura DataModel klasa 2/6

- Implementirati *Equals()* i *GetHashCode()* metode
- *Equals()* metoda proverava jednakost atributa koji pripadaju roditeljskoj klasi. Ukoliko se dobije potvrdan odgovor proveravaju se atributi tekuće klase.

```
public override bool Equals(object obj)
{
    if (base.Equals(obj))
    {
        PowerTransformer x = (PowerTransformer)obj;
        return (x.function == this.function && x.autotransformer == this.autotransformer &&
                CompareHelper.CompareLists(x.TransformerWindings, this.TransformerWindings, true));
    }
    else
    {
        return false;
    }
}

public override int GetHashCode()
{
    return base.GetHashCode();
}
```

Struktura DataModel klasa 3/6

- Kako bi servis na jedinstven način mogao da radi sa svakim tipom entiteta, potrebno je da klasa implementira metode koje će servis koristiti za GDA manipulaciju (*IAccess implementation* region).
- *IReference implementation* region sadrži metode koje servis koristi kako bi samostalno mogao da manipuliše atributima koji su tipa lista referenci (*target*) - detaljnije na sledećim vežbama.

```
#region IAccess implementation

public override bool HasProperty(ModelCode t)...

public override void GetProperty(Property prop)...

public override void SetProperty(Property property)...

#endregion IAccess implementation


#region IReference implementation

public override bool IsReferenced...

public override void GetReferences(Dictionary<ModelCode, List<long>> references, TypeOfReference refType)...

public override void AddReference(ModelCode referenceId, long globalId)...

public override void RemoveReference(ModelCode referenceId, long globalId)...

#endregion IReference implementation
```

Struktura DataModel klasa 4/6

- *HasProperty()* metoda daje odgovor na pitanje da li vrednost *ModelCode*-a odgovara nekom atributu klase. Prvo se proverava da li je to to atribut tekuće klase, ukoliko nije prelazi se na proveru da li je to atribut roditeljske klase.

```
public override bool HasProperty(ModelCode t)
{
    switch (t)
    {
        case ModelCode.POWERTR_AUTO:
        case ModelCode.POWERTR_FUNC:
        case ModelCode.POWERTR_WINDINGS:
            return true;

        default:
            return base.HasProperty(t);
    }
}
```

Struktura DataModel klasa 5/6

- *GetProperty()* metoda služi za konverziju atributa klase u *Property* objekat koji koristi GDA standard.
- Pročita se *propertyId (ModelCode)* prosleđenog properti objekta i proverava se da li on odgovara nekom od atributa tekuće klase. Ukoliko odgovara, vrednost tog atributa se zapiše kao vrednost prosleđenog propertija (*SetValue()* metoda). Ako *propertyId* ne odgovara nijednom atributu tekuće klase, poziva se *GetProperty()* metoda roditeljske klase.
- *GetProperty()* metoda MORA da obezbedi mogućnost čitanja svakog atributa klase.
- Ukoliko je atribut tipa enum MORA se konvertovati u *short* pre nego što se njegova vrednost zapiše u properti.

```
public override void GetProperty(Property prop)
{
    switch (prop.Id)
    {

        case ModelCode.POWERTR_FUNC:
            prop.SetValue((short)function);
            break;

        case ModelCode.POWERTR_AUTO:
            prop.SetValue(autotransformer);
            break;

        case ModelCode.POWERTR_WINDINGS:
            prop.SetValue(transformerWindings);
            break;

        default:
            base.GetProperty(prop);
            break;
    }
}
```

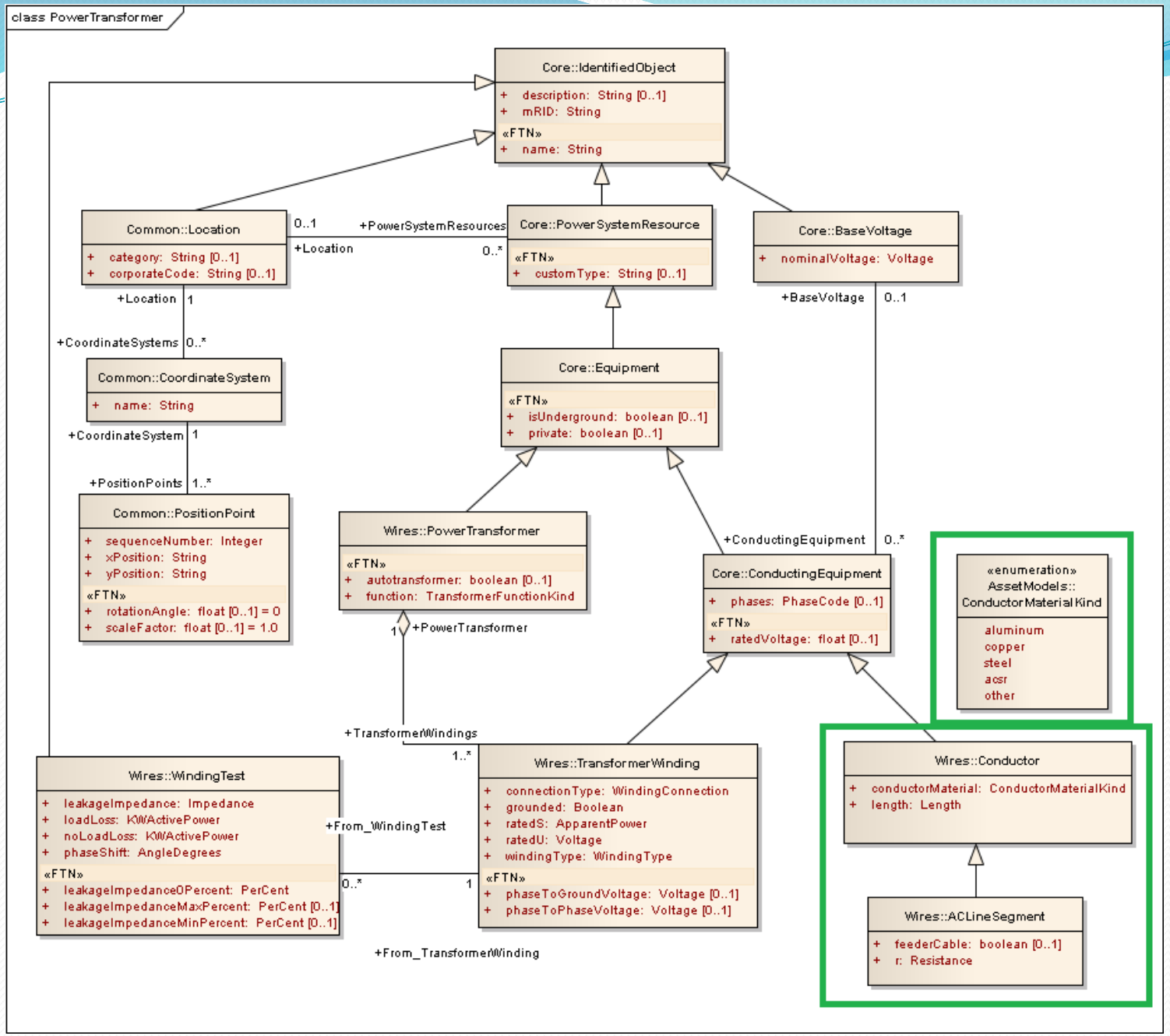
Struktura DataModel klasa 6/6

- *SetProperty()* metoda služi za postavljanje vrednosti atributa na osnovu *Property* objekta koji koristi GDA standard.
- Pročita se *propertyId (ModelCode)* prosleđenog properti objekta i proverava se da li on odgovara nekom od atributa tekuće klase. Ukoliko odgovara vrednost propertija se postavlja kao vrednost tog atributa. Ako *propertyId* ne odgovara nijednom atributu tekuće klase, poziva se *SetProperty()* metoda roditeljske klase.
- *SetProperty()* metoda NE SME da dozvoli postavljanje vrednosti *notSettable* atributa – atributa koji su navedeni u *ModelResourcesDesc* klasi kao atribute čiju vrednost postavlja servis (globalni identifikator i liste referenci (*target*))
- Ukoliko je atribut tipa enum, vrednost propertija je potrebno kastovati u odgovarajući tip enumeracije.

```
public override void SetProperty(Property property)
{
    switch (property.Id)
    {
        case ModelCode.POWERTR_AUTO:
            autotransformer = property.AsBool();
            break;

        case ModelCode.POWERTR_FUNC:
            function = (TransformerFunction)property.AsEnum();
            break;

        default:
            base.SetProperty(property);
            break;
    }
}
```

Rezime

- Prilikom dodavanja nove klase (ili atributa postojeće klase) potrebno je pratiti sledeće korake:
  1. Dodati odgovarajuće *ModelCode*-ove za nove klase i atribute
  2. Ukoliko su dodate nove enumeracije proširiti *Enums.cs* fajl
  3. Ukoliko je neki od atributa tipa enumeracija proširiti *EnumDescs* klasu dodatnim mapiranjem
  4. Ukoliko imamo *notSettable* atributa proširiti njima *ModelResourcesDesc* klasu (doraditi metodu *InitializeNotSettablePropertyIds()*)
  5. Ukoliko je dodata konkretna klasa proširiti *ModelResourcesDesc i Container* klase (metode *InitializeTypeIdsInInsertOrder() i CreateEntity()*)
  6. Implementirati nove klase i po potrebi doraditi postojeće na osnovu prethodno definisanih pravila.


class PowerTransformer
Core::IdentifiedObject
+ description: String [0..1]
+ mRID: String
«FTN»
+ name: String
Common::Location
+ category: String [0..1]
+ corporateCode: 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»
+ isUnderground: boolean [0..1]
+ private: 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
Wires::PowerTransformer
«FTN»
+ autotransformer: boolean [0..1]
+ function: TransformerFunctionKind
+ConductingEquipment
0..*
Core::ConductingEquipment
+ phases: PhaseCode [0..1]
«FTN»
+ ratedVoltage: float [0..1]
«enumeration»
AssetModels::
ConductorMaterialKind
aluminum
copper
steel
acsr
other
1
+PowerTransformer
+TransformerWindings
1..*
Wires::WindingTest
+ leakageImpedance: Impedance
+ loadLoss: KWActivePower
+ noLoadLoss: KWActivePower
+ phaseShift: AngleDegrees
«FTN»
+ leakageImpedanceOPercent: PerCent
+ leakageImpedanceMaxPercent: PerCent [0..1]
+ leakageImpedanceMinPercent: PerCent [0..1]
+From_WindingTest
0..*
1
Wires::TransformerWinding
+ connectionType: WindingConnection
+ grounded: Boolean
+ ratedS: ApparentPower
+ ratedU: Voltage
+ windingType: WindingType
«FTN»
+ phaseToGroundVoltage: Voltage [0..1]
+ phaseToPhaseVoltage: Voltage [0..1]
+From_TransformerWinding
Wires::Conductor
+ conductorMaterial: ConductorMaterialKind
+ length: Length
Wires::ACLineSegment
+ feederCable: boolean [0..1]
+ r: Resistance

Zadaci

1. Dodati apstraktnu klasu *Conductor*
2. Dodati konkretnu klasu *ACLineSegment*