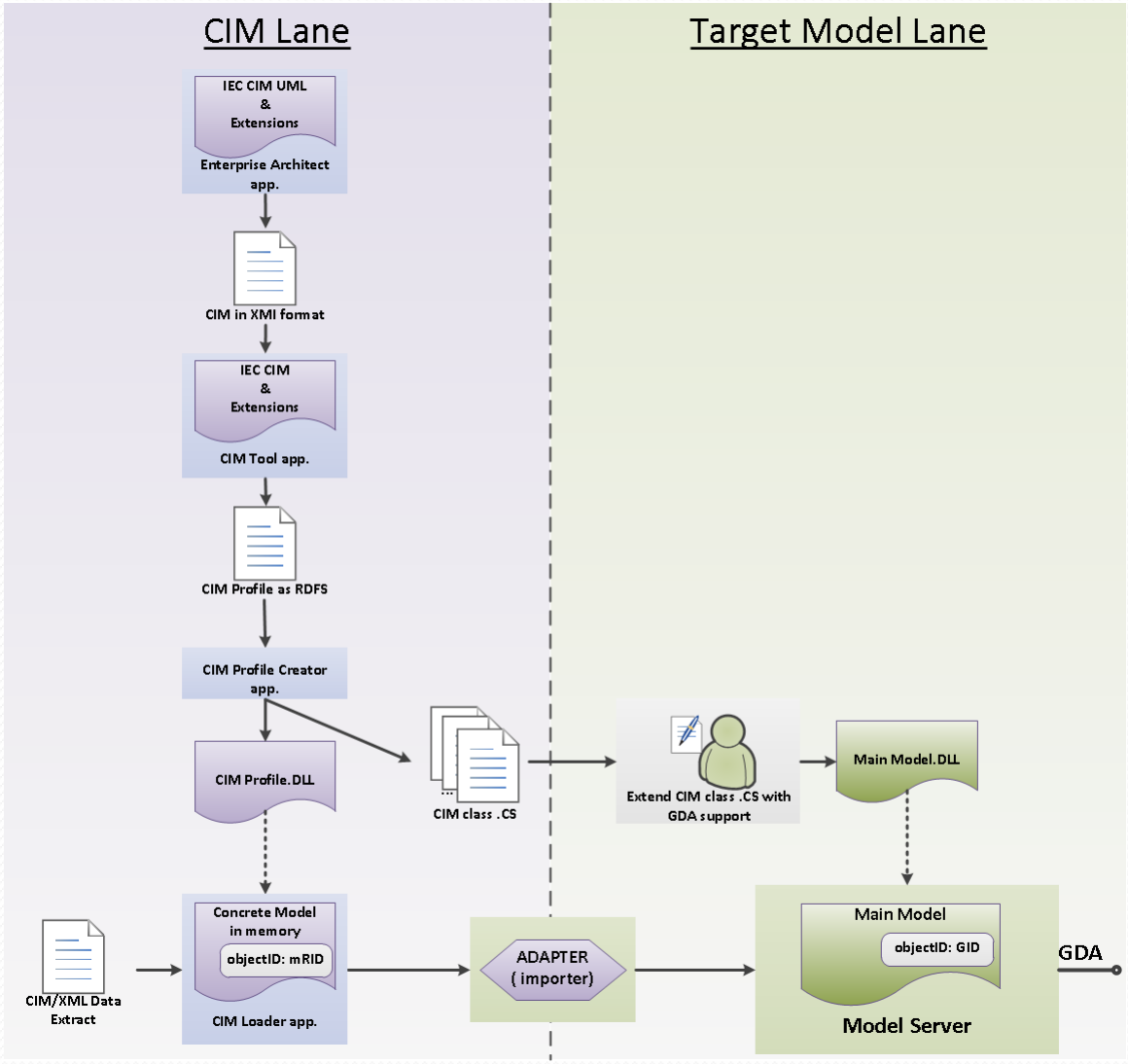
Standardi i modeliranje elektroenergetskih sistema

VEŽBA 8:

Uvod u Generic Data Access

Tok podataka pri inicijalizaciji modela elektroenergetske mreže



Generic Data Access (GDA) 1/2

- Omogućava pristup podacima bez ikakvog znanja o logičkoj šemi koja se koristi za unutrašnje skladištenje - implementaciji. Dovoljno je poznavanje definisanog modela, a u ovom slučaju je to CIM.
- Prvobitno je standardizovan u okviru OMG-a kao Data Access Facility (DAF) koji je pružao osnovnu sposobnost postavljanja upita nad nekim modelom. Daljom nadogradnjom nastaje GDA koji obezbeđuje naprednije funkcionalnosti (npr. filtriranje), čitanje i upis podataka.
- Bilo koju klasu i njene atribute iz internog modela prikazujemo u formi resursa pridržavajući se određenih pravila.

Generic Data Access (GDA) 2/2

- Standard formuliše upite i odgovore u vidu resursa, propertija i vrednosti.
- Resurs je svaki objekat koji poseduje jedinstveni identifikator
- Properti je neki aspekt resursa koji se može opisati. Kada se pojavi u upitu predstavljen je preko *ModelCode*-a. Svaki properti ima svoj domen – niz resursa na koji se može primeniti.
- Asocijacije između resursa se kreiraju preko propertija koji su tipa *Reference*

Property 1/2

- Opisuje atribut nekog objekta
- Sastoji se iz dva dela: Id i Value
- Id - Svaki properti je jednoznačno određen preko *ModelCode*-a
- Value – Sadrži vrednost atributa

```
/// <summary>
/// A class that describes property of generic model resource
/// </summary>

[DataContract]
public class Property : IComparable
{
    /// <summary>
    /// Code for property of model type
    /// </summary>
    private ModelCode id;

    /// <summary>
    /// Current value of the property
    /// </summary>
    private PropertyValue value;
```

Property 2/2

- Vrednost se setuje kroz konstruktor ili preko *SetValue(...)* metoda
- Vrednost se čita preko *As...* metoda

```
public void SetValue(bool boolValue)...
public void SetValue(byte byteValue)...
public void SetValue(short int16Value)...
public void SetValue(int int32Value)...
public void SetValue(long int64Value)...
public void SetValue(float floatValue)...
public void SetValue(double doubleValue)...
public void SetValue(string stringValue)...
```

```
public bool AsBool()...
public byte AsByte()...
public short AsEnum()...
public int AsInt()...
public long AsLong()...
public float AsFloat()...
public double AsDouble()...
public string AsString()...
public long AsReference()...
```

ResourceDescription

- Opisuje objekat
- Može da sadrži sve ili samo odabrane propertije
- Globalni identifikator objekta koji se opisuje predstavlja identifikator *ResourceDescription*-a

```
/// <summary>
/// A class that describes generic model resource
/// </summary>

[DataContract]
public class ResourceDescription
{
    private long id;
    private List<Property> properties = new List<Property>();

    public ResourceDescription()
    {
    }

    public ResourceDescription(long id)
    {
        this.id = id;
    }
```

Association

- Opisuje veze između objekata
- Sadrži identifikator propertija (*propertyId*) tipa referenca
- Vrednost propertija je globalni identifikator referenciranog objekta

```
[DataContract]
public class Association
{
    private bool inverse;
    private ModelCode propertyId;
    private ModelCode type;

    public Association()
    {
        this.inverse = false;
        this.propertyId = 0;
        this.type = 0;
    }
```

Metode za čitanje podataka

- Resource Query Service

- GetValues(…) – čitanje jednog resursa

```
public ResourceDescription GetValues(long resourceId,  List<ModelCode> propIds)
```

- GetExtentValues(…) – čitanje niza resursa za isti tip objekta

```
public int GetExtentValues(ModelCode code, List<ModelCode> propIds)
```

- GetRelatedValues(…) – čitanje referenciranog resursa

```
public int GetRelatedValues(long source, List<ModelCode> propIds , Association association)
```

- GetDescendentValues(…) – čitanje niza referencira resursa

```
public int GetDescendentValuesLinq(List<long> sources, List<ModelCode> propIds,
                                   List<Association> path, List<Association> tail)
```

Izmena (upis) podataka

- GDA standard predviđa izmenu podataka
- Definiše se objekat koji će nositi potrebne izmene modela - *Delta*
- Metoda *ApplyUpdate(…)* se poziva prilikom izmene modela

```
public UpdateResult ApplyUpdate(Delta delta)
```

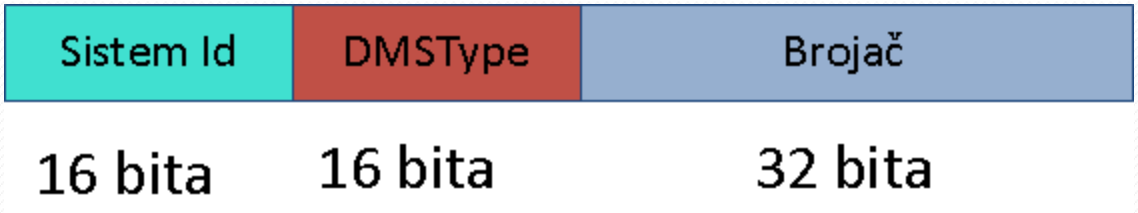
- Objekat tipa *Delta* sadži tri tipa operacija: dodavanja, ažuriranje i brisanje entiteta

```
[DataContract]
public class Delta
{
    private long id;
    private List<ResourceDescription> insertOps = new List<ResourceDescription>();
    private List<ResourceDescription> deleteOps = new List<ResourceDescription>();
    private List<ResourceDescription> updateOps = new List<ResourceDescription>();
    private bool positiveIdsAllowed;
```

Rad sa *Delta* objektom 1/2

- Klasa implementira metode potrebne za dodavanje operacija
  public void AddDeltaOperation(DeltaOpType type, ResourceDescription rd, bool addAtEnd)

- Ukoliko se dodaju insert operacije znači da je potrebno proslediti globalni identifikator koji ne postoji na servisu

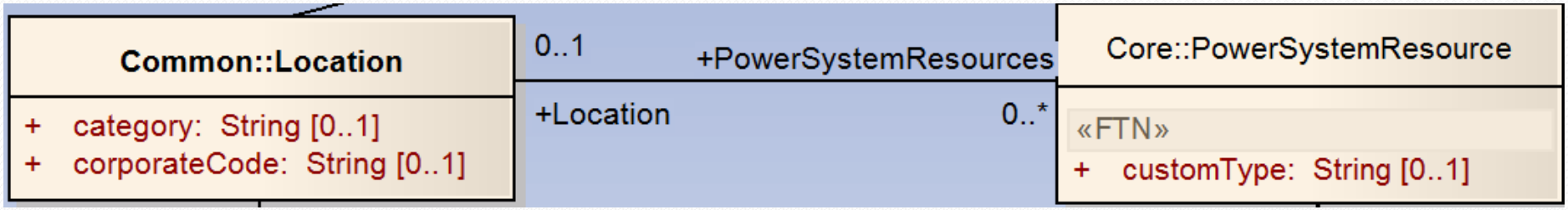


Kako klijentska strana ne bi morala (i ne može) da brine o jedinstvenosti brojača koji je deo globalnog identifikatora, potrebno je na najnižih 32 bita postaviti negativnu vrednost. Na taj način servis zna da se radi o novom entitetu i dodeljuje mu odgovarajuću vrednost brojača.

- *ModelCodeHelper* klasa implementira statičke metode pogodne za kreiranje globalnog identifikatora

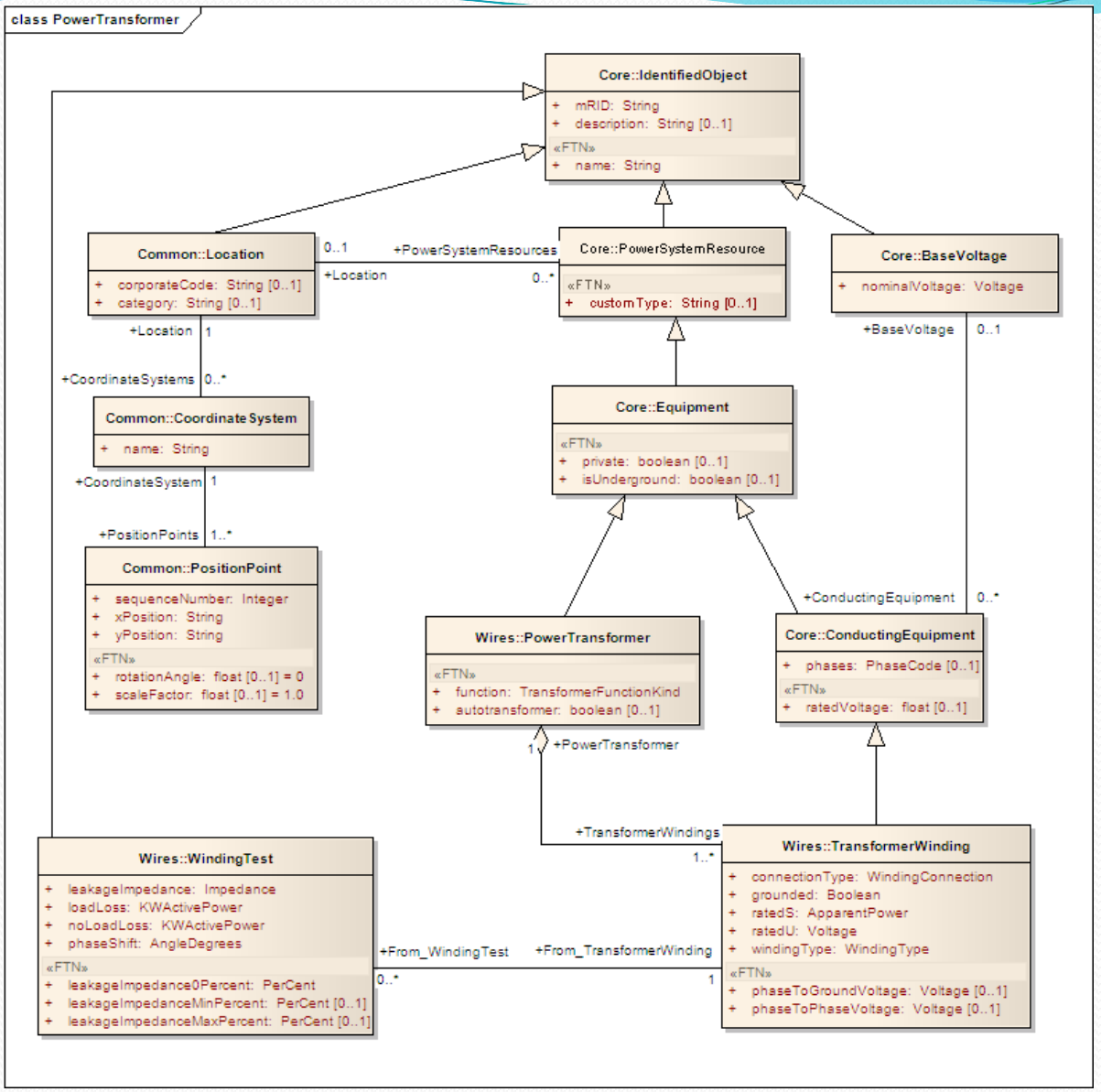
Rad sa *Delta* objektom 2/2

- Properti tipa reference kao vrednost nose globalni identifikator referenciranog objekta



- Implementaciono ograničenje:

UVEK se dodaju samo propertiji tipa referenca, NIKADA se ne dodaju propertiji tipa niz referenci. Propertiji tipa niz referenci se mogu samo čitati sa servisa, ne i zapisivati.


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. Kreirati objekat tipa *Delta* koji sadrži sledeće insert opercije:
   a) Dva resursa tipa *BaseVoltage*, prvom postaviti nominalni napon na 110 kV, drugom na 20 kV
   b) Resurs tipa *Location*
   c) Resurs tipa *PowerTransformer;* povezati ga na *Location* resurs
   d) Dva resursa tipa *TransformerWinding;* oba povezati na transformator; prvi *TransformerWinding* povezati na *BaseVoltage* od 110 kV, drugi na *BaseVoltage* od 20 kV
2. Programski kreirati .XML *file* u kome ce navedeni Delta objekat biti prikazan. Koristiti metodu *ExportToXml* implementiranu nad *Delta* klasom.