4. Gyakorlat: JET

Simon Balázs, BME IIT, 2013.

1 Feladat

Készítsünk JET generátort, amely az EMF metamodell alapján context diagramok (CD) és data-flow diagramok (DFD) szöveges reprezentációját képes generálni!

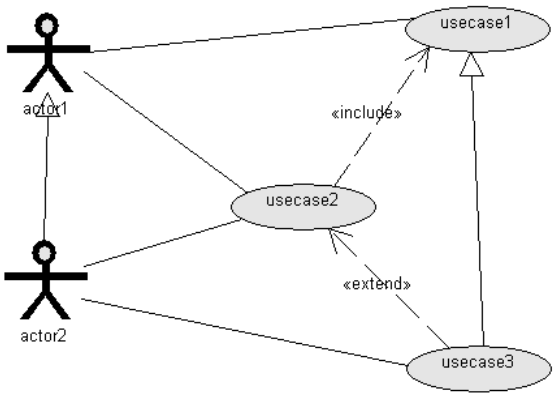
Lépések:

1. Olvassuk el a use-case kódgenerátor elkészítésére vonatkozó JET tutorial-t (2. fejezet)!
2. Készítsük el a 3. fejezet útmutatói alapján a CD és DFD diagramokhoz tartozó kódgenerátort!

2 JET Tutorial

2.1 Feladat

Készítsünk olyan kódgenerátort, amely a következő use-case diagram EMF modelljéből előállítja annak szöveges változatát:



A szöveges változat legyen:

```
use-case-diagram TestModel {
    actor actor1 {
        usecase1, usecase2
    }
    actor actor2: actor1 {
        usecase2, usecase3
    }
    use-case usecase1;
    use-case usecase2 includes usecase1;
    use-case usecase3: usecase1 extends usecase2;
}
```

2.2 A modell felépítése

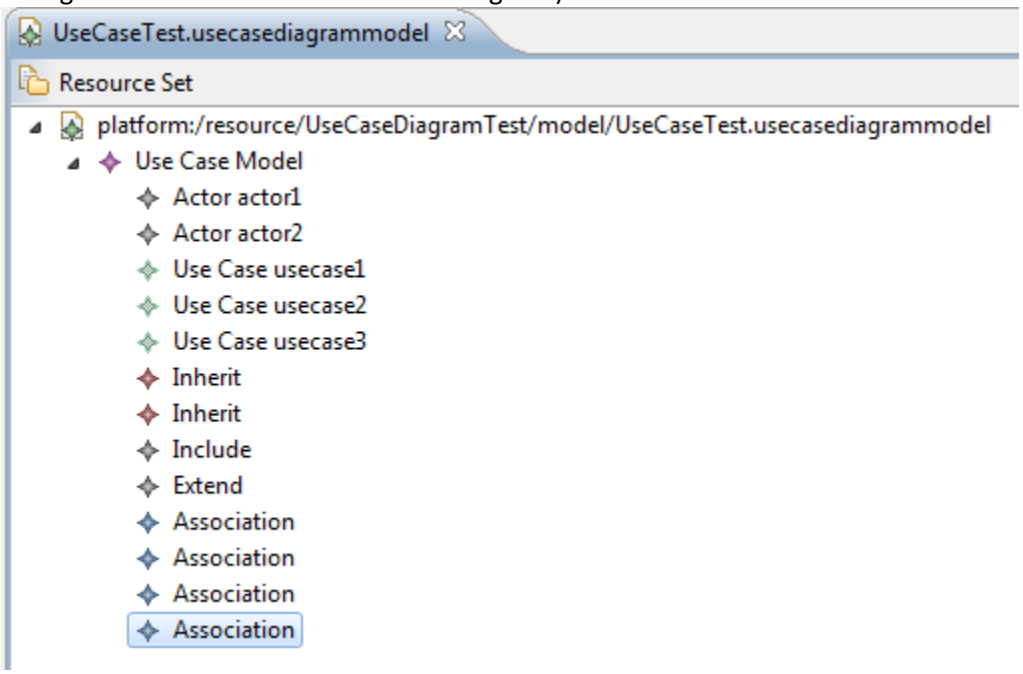
2.2.1 A modell felépítése grafikusan

A modell felépítése a 2. gyakorlaton elkészített GMF grafikus szerkesztő segítségével tehető meg. A grafikusan felépített diagram projektjének **model** könyvtárában a **.usecasediagrammodel** kiterjesztésű XML valójában egy XMI dokumentum, amit be lehet tölteni Java objektumgráfként az EMF implementációs osztályok segítségével.

2.2.2 A modell felépítése faként

Ha a GMF editor még nem áll rendelkezésre, akkor a modell az 1. gyakorlaton elkészített EMF editorban is elkészíthető faként. Ehhez:

1. Kattintsunk jobbgombbal a **UseCaseDiagram.editor** projekten, majd válasszuk a **Run As > Eclipse Application** menüpontot!
2. A megjelenő Eclipse-ben készítsünk egy üres EMF projektet!
3. A projektben pedig hozzunk létre a **New > Other… / Example EMF Model Creation Wizards / UseCaseDiagramModelModel**-t
4. Model object-nek válasszuk a **UseCaseModel**-t!
5. A fahierarchiában jobbgombbal kattintva az egyes csúcsokon gyermekeket adhatunk hozzájuk, tulajdonságaikat pedig a **Properties** ablakban módosíthatjuk. Hozzuk létre a fenti ábrának megfelelően a következő UseCase modellt (a Properties ablakban mindig állítsuk be a megfelelő neveket illetve asszociációvégeket):



6. Mentsük el a fájlt!

2.2.3 A modell betöltése XML-ből

Akár a grafikus, akár a fastruktúra változatot választottuk, az aktuális **workspace** mellett a **runtime-EclipseApplication** könyvtár alatt megtaláljuk a projektet, benne pedig a egy **.usecasediagrammodel** kiterjesztésű XML fájlt, ami valójában egy XMI. Ezt a fájlt Java kódból a következőképpen tölthetjük be:

```
import java.util.Map;

import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;

import UseCaseDiagramModel.UseCaseDiagramModelFactory;
import UseCaseDiagramModel.UseCaseModel;
import UseCaseDiagramModel.impl.UseCaseDiagramModelPackageImpl;

public class Program {
    public static void main(String[] args) {
        // Initialize the model
        UseCaseDiagramModelPackageImpl.init();
        // Retrieve the default factory singleton
        UseCaseDiagramModelFactory factory = UseCaseDiagramModelFactory.eINSTANCE;

        // Register the XMI resource factory for the .usecasediagrammodel extension
        Resource.Factory.Registry reg = Resource.Factory.Registry.INSTANCE;
        Map<String, Object> m = reg.getExtensionToFactoryMap();
        m.put("usecasediagrammodel", new XMIResourceFactoryImpl());

        // Obtain a new resource set
        ResourceSet resSet = new ResourceSetImpl();

        // Get the resource
        Resource resource = resSet.
            getResource(URI.createURI("UseCaseTest.usecasediagrammodel"), true);

        // Get the first model element and cast it to the right type
        UseCaseModel model = (UseCaseModel)resource.getContents().get(0);

        // The whole model is accessible from the "model" variable
    }
}
```

2.2.4 A modell felépítése Java kódból

A modell Java kódból is felépíthető, amennyiben ez számunkra kényelmesebb:

```
import UseCaseDiagramModel.Actor;
import UseCaseDiagramModel.Association;
import UseCaseDiagramModel.Extend;
import UseCaseDiagramModel.Include;
import UseCaseDiagramModel.Inherit;
import UseCaseDiagramModel.UseCase;
import UseCaseDiagramModel.UseCaseDiagramModelFactory;
import UseCaseDiagramModel.UseCaseModel;
import UseCaseDiagramModel.impl.UseCaseDiagramModelPackageImpl;

public class Program {

    public static void main(String[] args) {
        UseCaseDiagramModelPackageImpl.init();
        UseCaseDiagramModelFactory factory = UseCaseDiagramModelFactory.eINSTANCE;
```

```
        UseCaseModel model = factory.createUseCaseModel();
        model.setName("TestModel");

        Actor actor1 = factory.createActor();
        model.getClassifiers().add(actor1);
        actor1.setName("actor1");
        Actor actor2 = factory.createActor();
        model.getClassifiers().add(actor2);
        actor2.setName("actor2");
        UseCase uc1 = factory.createUseCase();
        model.getClassifiers().add(uc1);
        uc1.setName("usecase1");
        UseCase uc2 = factory.createUseCase();
        model.getClassifiers().add(uc2);
        uc2.setName("usecase2");
        UseCase uc3 = factory.createUseCase();
        model.getClassifiers().add(uc3);
        uc3.setName("usecase3");

        Association assoc1 = factory.createAssociation();
        model.getRelationships().add(assoc1);
        assoc1.setActor(actor1);
        assoc1.setUseCase(uc1);
        Association assoc2 = factory.createAssociation();
        model.getRelationships().add(assoc2);
        assoc2.setActor(actor1);
        assoc2.setUseCase(uc2);
        Association assoc3 = factory.createAssociation();
        model.getRelationships().add(assoc3);
        assoc3.setActor(actor2);
        assoc3.setUseCase(uc2);
        Association assoc4 = factory.createAssociation();
        model.getRelationships().add(assoc4);
        assoc4.setActor(actor2);
        assoc4.setUseCase(uc3);

        Include inc = factory.createInclude();
        model.getRelationships().add(inc);
        inc.setSource(uc2);
        inc.setTarget(uc1);
        Extend ext = factory.createExtend();
        model.getRelationships().add(ext);
        ext.setSource(uc3);
        ext.setTarget(uc2);
        Inherit inh1 = factory.createInherit();
        model.getRelationships().add(inh1);
        inh1.setSource(uc3);
        inh1.setTarget(uc1);
        Inherit inh2 = factory.createInherit();
        model.getRelationships().add(inh2);
        inh2.setSource(actor2);
        inh2.setTarget(actor1);

        // A teljes modell elérhető a “model” változóból
    }
}
```

2.3 Megoldás

1. Készítsünk egy új JET projektet: **File > New > Project… / Java / Java Project**
    1.1. Project name: **usecasegen**
    1.2. **Finish**
2. Kattintsunk a projekten jobbgombbal és válasszuk a **Properties** menüpontot!
    2.1. A **Java Build Path** beállításoknál a **Projects** fülön kattintsunk az **Add…** gombra és pipáljuk ki a **UseCaseDiagram** projektet!
    2.2. A **Libraries** fülön az **Add External Jars…** gombbal adjuk hozzá az Eclipse plugins könyvtárából a következő **jar** fájlokat (ahol X.Y.Z helyett egy verziószám szerepel):
    org.eclipse.emf.common_X.Y.Z.jar
    org.eclipse.emf.ecore_X.Y.Z.jar
    org.eclipse.emf.ecore.xmi_X.Y.Z.jar
    2.3. Az **OK** gombbal zárjuk be az ablakot!
3. Kattintsunk jobbgombbal a projekten és válasszuk a **New > Other… / Java Emitter Templates / Convert Projects to JET Projects** menüpontot!
    3.1. **Next**
    3.2. A **Projects** listában pipáljuk ki a **usecasegen** projektet!
    3.3. **Finish**.
4. Kattintsunk jobbgombbal a **template** alkönyvtáron és válasszuk a **New > File** menüpontot!
    4.1. File name: **XucdGenerator.xucdjet**
    4.2. **Finish**
    4.3. A build-del hibaüzenet azért jön fel, mert a fájl üres. Nyugodtan figyelmen kívül hagyhatjuk.
    4.4. Írjuk be a következő sort a fájl elejére, majd mentsük el:

```
<%@ jet package="usecasegen" class="XucdGenerator" %>
```

    4.5. Ennek hatására a hiba megszűnik.
5. Kattintsunk jobbgombbal a projekten, majd válasszuk a **Properties** menüpontot!
    5.1. A **JET Settings**-nél a **Source Container**-be írjuk be a következő értéket: **src**
    5.2. **OK**
    5.3. Így az **src** könyvtárba kerülnek a JET által készített Java kódok.
    5.4. Töröljük ki a feleslegesen a projektben maradt **usecasegen** könyvtárat!
6. Töltsük ki az **XucdGenerator.xucdjet** fájlt:

```
<%@ jet package="usecasegen" class="XucdGenerator" imports="UseCaseDiagramModel.*" %>
<% UseCaseModel model = (UseCaseModel)argument; %>
use-case-diagram <%= model.getName() %> {
<%
    for (Classifier classifier: model.getClassifiers()) {
        if (classifier instanceof Actor) {
            Actor actor = (Actor)classifier;
%>
    actor <%= actor.getName() %><%
            String delim = ": ";
            for (Relationship rel: model.getRelationships()) {
                if (rel instanceof Inherit) {
                    Inherit inh = (Inherit)rel;
                    if (inh.getSource() == actor) {
%><%= delim %><%= inh.getTarget().getName() %><%
                        delim = ", ";
                    }
                }
```

```
            }
%> {

        <%
            delim = "";
            for (Relationship rel: model.getRelationships()) {
                if (rel instanceof Association) {
                    Association assoc = (Association)rel;
                    if (assoc.getActor() == actor) {
%><%= delim %><%= assoc.getUseCase().getName() %><%
                        delim = ", ";
                    }
                }
            }
%>
    }
<%
        } else if (classifier instanceof UseCase) {
            UseCase useCase = (UseCase)classifier;
%>
    use-case <%= useCase.getName() %><%
            String delim = ": ";
            for (Relationship rel: model.getRelationships()) {
                if (rel instanceof Inherit) {
                    Inherit inh = (Inherit)rel;
                    if (inh.getSource() == useCase) {
%><%= delim %><%= inh.getTarget().getName() %><%
                        delim = ", ";
                    }
                }
            }
            delim = " extends ";
            for (Relationship rel: model.getRelationships()) {
                if (rel instanceof Extend) {
                    Extend ext = (Extend)rel;
                    if (ext.getSource() == useCase) {
%><%= delim %><%= ext.getTarget().getName() %><%
                        delim = ", ";
                    }
                }
            }
            delim = " includes ";
            for (Relationship rel: model.getRelationships()) {
                if (rel instanceof Include) {
                    Include inc = (Include)rel;
                    if (inc.getSource() == useCase) {
%><%= delim %><%= inc.getTarget().getName() %><%
                        delim = ", ";
                    }
                }
            }
%>;
<%
        }
    }
%>
}
```

7. Mentsük el a fájlt. Ennek hatására létrejön az **src** könyvtárban egy **usecasegen.XucdGenerator** Java osztály. Ha az **UseCaseGenerator.xucdjet** fájl helyes, akkor a Java osztály hibamentesen lefordul.
8. Készítsünk főprogramot a generátor tesztelésére (**usecasegen.UseCaseGeneratorProgram** osztály):

```
package usecasegen;

import UseCaseDiagramModel.Actor;
import UseCaseDiagramModel.Association;
import UseCaseDiagramModel.Extend;
import UseCaseDiagramModel.Include;
import UseCaseDiagramModel.Inherit;
```

```java
import UseCaseDiagramModel.UseCase;
import UseCaseDiagramModel.UseCaseDiagramModelFactory;
import UseCaseDiagramModel.UseCaseModel;
import UseCaseDiagramModel.impl.UseCaseDiagramModelPackageImpl;

public class XucdGeneratorProgram {

    public static void main(String[] args) {
        UseCaseDiagramModelPackageImpl.init();
        UseCaseDiagramModelFactory factory = UseCaseDiagramModelFactory.eINSTANCE;

        UseCaseModel model = factory.createUseCaseModel();
        // ... a modell betöltése a model változóba ...

        XucdGenerator generator = new XucdGenerator();
        System.out.println(generator.generate(model));
    }

}
```

9. A program futásának eredménye:

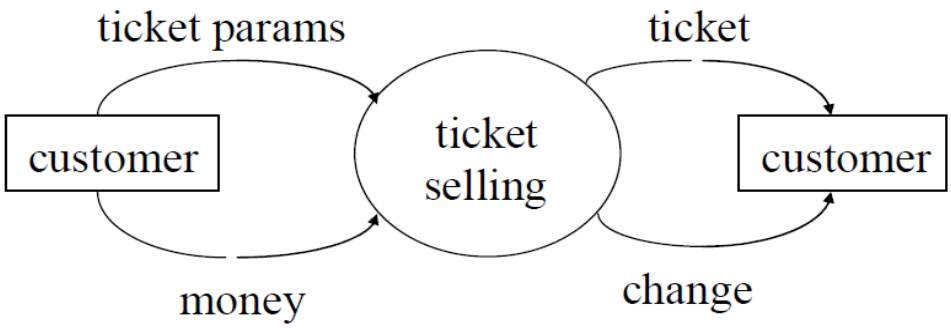
```
use-case-diagram TestModel {
    actor actor1 {
        usecase1, usecase2
    }
    actor actor2: actor1 {
        usecase2, usecase3
    }
    use-case usecase1;
    use-case usecase2 includes usecase1;
    use-case usecase3: usecase1 extends usecase2;
}
```

3 CD és DFD generátor készítése

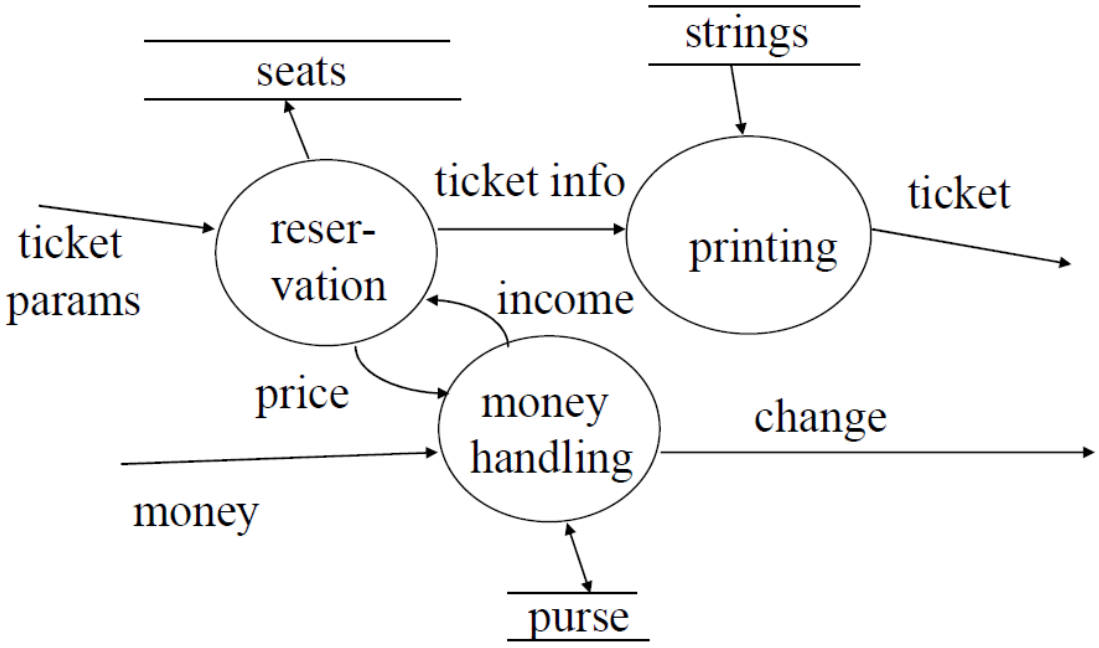
Készítsünk JET kódgenerátort, amely egy EMF modellben leírt CD illetve DFD alapján előállítja a modell szöveges változatát.

3.1 Példa

Vegyük a következő CD-t:



És a következő DFD-t:



Ezek szöveges reprezentációja legyen a következő:

```
contextdiagram cinema
{
    terminator customer;

    process ticket_selling;

    dataflow ticket_params: customer -> ticket_selling;
    dataflow money: customer -> ticket_selling;
    dataflow ticket: ticket_selling -> customer;
    dataflow change: ticket_selling -> customer;
}

dataflowdiagram ticket_selling
{
    store seats;
    store purse;
    store strings;

    process reservation;
    process money_handling;
    process printing;

    dataflow ticket_params: -> reservation;
    dataflow money: -> money_handling;
    dataflow ticket: printing -> ;
    dataflow change: money_handling -> ;
    dataflow ticket_info: reservation -> printing;
    dataflow price: reservation -> money_handling;
    dataflow income: money_handling -> reservation;
    dataflow: reservation -> seats;
    dataflow: money_handling -> purse;
    dataflow: purse -> money_handling;
}
```

3.2 Feladat

A következő lépéseket kell végrehajtani:

1. Építsük fel a 3.1. szakaszban bemutatott CD és DFD modelleket Java kódból vagy XML-ből!
2. Készítsünk JET generátort, amely a példaként megadott kimeneti formátumot állítja elő!