5. Gyakorlat: Xtext

Simon Balázs, BME IIT, 2013.

1 Feladat

Készítsünk Xtext nyelvtani elemzőt, amely a generátor által előállított szöveges DSL reprezentációt visszatranszformálja az EMF metamodellnek megfelelő context diagrammá (CD) illetve data-flow diagrammá (DFD)!

Lépések:

1. Olvassuk el a use-case DSL nyelvtani elemzőjét megvalósító Xtext tutorial-t (2. fejezet)!
2. Készítsük el a 3. fejezet útmutatói alapján a CD és DFD diagramokhoz tartozó nyelvtani elemzőt!

2 XText Tutorial

2.1 Feladat

Készítsünk egy olyan XText nyelvtant, amely képes feldolgozni a következő use-case diagram szöveges DSL-t:

```
use-case-diagram UseCaseTest
{
    use-case U;
    use-case V : U;
    use-case W
        includes U;
    use-case X
        extends U;
    use-case Y
        extends W, X
        includes U, V;
    use-case Z : Y
        extends W, X
        includes U, V;

    actor A { U, V }
    actor B : A { W, X }
    actor C : A, B { Y, Z }
}
```

A nyelvtan segítségével építsünk absztrakt szintaxis fát (AST) a fenti fájlból, majd az AST-t transzformáljuk az előző fejezetben megalkotott use-case metamodellnek megfelelő use-case modellre!

Erre a következők miatt van szükség. Az Xtext által előállított AST modell is valójában egy EMF metamodell, azonban ez nem teljesen egyezik az első gyakorlaton elkészített saját EMF metamodellel, ezért az Xtext metamodelljének megfelelő modelleket át kell transzformálni a saját metamodellünknek megfelelő modellekké.

Megjegyzés: új DSL készítésekor sokszor célszerűbb az Xtext-tel kezdeni az egész folyamatot, és az általa felépített metamodellt használni EMF metamodellként is. Így nem lesz szükség kétfajta metamodell egyidejű karbantartására.

2.2 Megoldás

1. **File > New… > Project / Xtext / Xtext Project**
    1.1. **Project name:** usecasedsl
    1.2. **Language / Name:** usecasedsl.UseCaseDsl
    1.3. **Language / Extensions:** xucd
    1.4. Szedjük ki a pipát a **Create SDK feature project** elől, mivel a kódgenerálást más technológiával valósítjuk meg
    1.5. Finish

2. A **UseCaseDsl.xtext** fájlba írjuk be a következő nyelvtant:

```
grammar usecasedsl.UseCaseDsl with org.eclipse.xtext.common.Terminals
generate useCaseDsl "http://www.UseCaseDsl.usecasedsl"

Model:
    diagram=UseCaseDiagram;

UseCaseDiagram:
    'use-case-diagram' name=ID '{'
        (declarations+=Declaration)*
    '}';

Declaration:
    Actor | UseCase;

Actor:
    'actor' name=ID (':' supers+=[Actor] (',' supers+=[Actor])*)? '{'
        useCases+=[UseCase] (',' useCases+=[UseCase])*
    '}';
UseCase:
    'use-case' name=ID (':' supers+=[UseCase] (',' supers+=[UseCase])*)?
        ('extends' extends+=[UseCase] (',' extends+=[UseCase])*)?
        ('includes' includes+=[UseCase] (',' includes+=[UseCase])*)? ';';
```

3. Kattinsunk jobbgombbal a **UseCaseDsl.xtext** belsejében és válasszuk ki a **Run As > Generate Xtext Artifacts** menüpontot!
   3.1. Első futtatáskor a generátor rákérdez, hogy használni kívánjuk-e az ANTLR3 parser generátort. A kérdésre válaszoljunk igennel!
4. Kattintsunk jobbgombbal a **usecasedsl** projekten, majd válasszuk a **Run As > Eclipse Application** menüpontot!
   4.1. Készítsünk egy üres projektet: **File > New > Project... / General / Project**
   4.2. Készítsünk egy **UseCaseTest.xucd** fájlt ebben a projektben! A megjelenő **Do you want the Xtext nature to be added to this project?** kérdésre válaszoljunk igennel!
   4.3. Gépeljük be a feladatkiírásban szereplő szöveget! Használjuk a **Ctrl+Space** billentyűkombinációt is! Figyeljük meg, hogy syntax highlighting és intellisense segítség, illetve hiba esetén a megfelelő helyen hibajelzés!
   4.4. Mentsük el a fájlt és zárjuk be az új Eclipse-et!
5. Az eredeti Eclipse-ben készítsünk egy új Java projektet **UseCaseDiagramConverter** néven!
6. Kattintsunk a projekten jobbgombbal és válasszuk a **Properties** menüpontot!
   6.1. A **Java Build Path** beállításoknál a **Projects** fülön kattintsunk az **Add…** gombra és pipáljuk ki a **UseCaseDiagram** illetve **usecasedsl** projekteket!
   6.2. 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
   com.google.inject_X.Y.Z.jar
   6.3. Az **OK** gombbal zárjuk be az ablakot!
7. Készítsük el a következő **usecasediagram.converter.Dsl2Emf** osztályt:

```
package usecasediagram.converter;
```

```java
import java.util.HashMap;

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

import usecasedsl.UseCaseDslStandaloneSetup;
import usecasedsl.useCaseDsl.Actor;
import usecasedsl.useCaseDsl.Declaration;
import usecasedsl.useCaseDsl.Model;
import usecasedsl.useCaseDsl.UseCase;
import usecasedsl.useCaseDsl.UseCaseDiagram;
import UseCaseDiagramModel.Association;
import UseCaseDiagramModel.Classifier;
import UseCaseDiagramModel.Extend;
import UseCaseDiagramModel.Include;
import UseCaseDiagramModel.Inherit;
import UseCaseDiagramModel.UseCaseDiagramModelFactory;
import UseCaseDiagramModel.UseCaseModel;
import UseCaseDiagramModel.impl.UseCaseDiagramModelPackageImpl;

public class Dsl2Emf {

    public static void main(String[] args) {
        new UseCaseDslStandaloneSetup().createInjectorAndDoEMFRegistration();

        ResourceSet rs = new ResourceSetImpl();
        Resource resource = rs.getResource(URI.createURI("./UseCaseTest.xucd"),
                true);
        EObject eobject = resource.getContents().get(0);

        Model dslModel = (Model) eobject;
        UseCaseDiagram dslDiagram = dslModel.getDiagram();
        UseCaseModel model = Dsl2Emf.transformUseCaseDsl2Emf(dslDiagram);

        for (Classifier classifier: model.getClassifiers())
        {
            System.out.println(classifier.getName()+
                                " ("+classifier.getClass().getName()+")");
        }
    }

    public static UseCaseModel transformUseCaseDsl2Emf(UseCaseDiagram dslDiagram) {
        UseCaseDiagramModelPackageImpl.init();
        UseCaseDiagramModelFactory factory = UseCaseDiagramModelFactory.eINSTANCE;

        UseCaseModel useCaseModel = factory.createUseCaseModel();
        useCaseModel.setName(dslDiagram.getName());

        HashMap<Actor, UseCaseDiagramModel.Actor> actorMapping =
            new HashMap<Actor, UseCaseDiagramModel.Actor>();
        HashMap<UseCase, UseCaseDiagramModel.UseCase> useCaseMapping =
            new HashMap<UseCase, UseCaseDiagramModel.UseCase>();
        for (Declaration dslDeclaration : dslDiagram.getDeclarations()) {
            if (dslDeclaration instanceof Actor) {
                Actor dslActor = (Actor) dslDeclaration;
                UseCaseDiagramModel.Actor actor = factory.createActor();
                actor.setName(dslActor.getName());
                actorMapping.put(dslActor, actor);
                useCaseModel.getClassifiers().add(actor);
            } else if (dslDeclaration instanceof UseCase) {
                UseCase dslUseCase = (UseCase) dslDeclaration;
                UseCaseDiagramModel.UseCase useCase = factory.createUseCase();
                useCase.setName(dslUseCase.getName());
                useCaseMapping.put(dslUseCase, useCase);
                useCaseModel.getClassifiers().add(useCase);
            }
        }
        for (Declaration dslDeclaration : dslDiagram.getDeclarations()) {
            if (dslDeclaration instanceof Actor) {
                Actor dslActor = (Actor) dslDeclaration;
                UseCaseDiagramModel.Actor actor = actorMapping.get(dslActor);
```

```
                for (Actor superActor : dslActor.getSupers()) {
                    Inherit inherit = factory.createInherit();
                    inherit.setSource(actor);
                    inherit.setTarget(actorMapping.get(superActor));
                    useCaseModel.getRelationships().add(inherit);
                }
                for (UseCase useCase : dslActor.getUseCases()) {
                    Association association = factory.createAssociation();
                    association.setActor(actor);
                    association.setUseCase(useCaseMapping.get(useCase));
                    useCaseModel.getRelationships().add(association);
                }
            } else if (dslDeclaration instanceof UseCase) {
                UseCase dslUseCase = (UseCase) dslDeclaration;
                UseCaseDiagramModel.UseCase useCase = useCaseMapping
                        .get(dslUseCase);
                for (UseCase superUseCase : dslUseCase.getSupers()) {
                    Inherit inherit = factory.createInherit();
                    inherit.setSource(useCase);
                    inherit.setTarget(useCaseMapping.get(superUseCase));
                    useCaseModel.getRelationships().add(inherit);
                }
                for (UseCase includedUseCase : dslUseCase.getIncludes()) {
                    Include include = factory.createInclude();
                    include.setSource(useCase);
                    include.setTarget(useCaseMapping.get(includedUseCase));
                    useCaseModel.getRelationships().add(include);
                }
                for (UseCase extendedUseCase : dslUseCase.getExtends()) {
                    Extend extend = factory.createExtend();
                    extend.setSource(useCase);
                    extend.setTarget(useCaseMapping.get(extendedUseCase));
                    useCaseModel.getRelationships().add(extend);
                }
            }
        }
        return useCaseModel;
    }

}
```

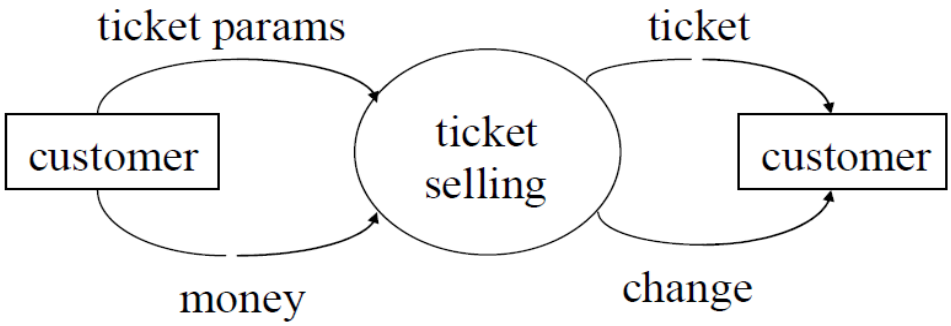
8. Figyeljük meg, hogy egy HashMap-re is szükség volt! Ennek az az oka, hogy az örökléseknél illetve az asszociációknál hivatkozott elemek deklarációja a hivatkozó elem deklarációja után is következhet. Emiatt először minden elemet létre kell hozni, és csak utána következhet az öröklések és asszociációk beállítása.

3 CD és DFD nyelvtani elemző készítése

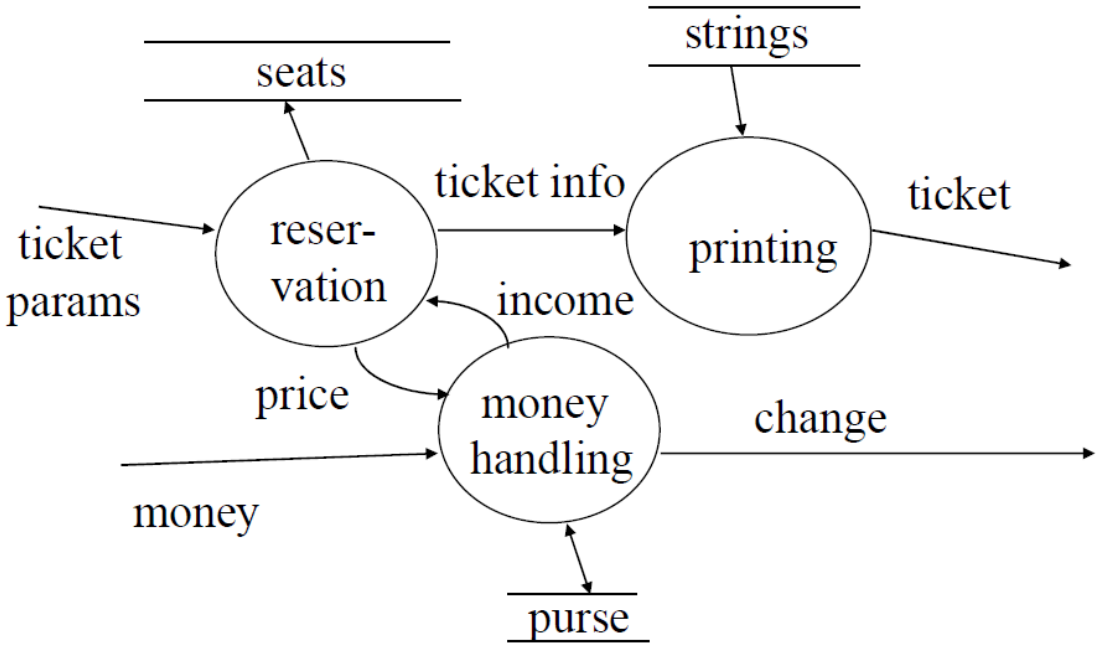
Készítsünk Xtext nyelvtani elemzőt, amely egy CD illetve DFD szöveges leírás alapján előállítja azok modelljét a korábban már elkészített EMF metamodellnek megfelelően!

3.1 Példa

Vegyük a következő CD-t:



És a következő DFD-t:



Ezek szöveges reprezentációja 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. Készítsünk Xtext elemzőt a 3.1-es szakaszban bemutatott forráskódok feldolgozására!
2. Elemezzük a 3.1-es pontban bemutatott forráskódokat a nyelvtan segítségével! Ellenőrzésképpen a feldolgozás után futtassuk le a 3. gyakorlaton készített kódgenerátort és vizsgáljuk meg, hogy ugyanazt a kódot kaptuk-e vissza!