Modellek programozott feldolgozása

Laboratórium ismerető

Célkitűzések

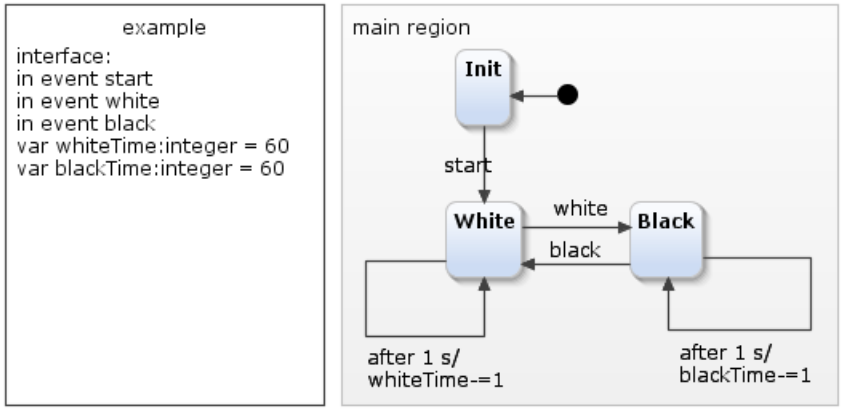
A mérés célja, hogy megismerkedjünk a modellezőeszközök belső felépítésével, modellek kezelésének módjaival. A mérés során tanultak alapján újabb funkciókkal kiegészíthetünk létező modellezőeszközöket, és kódgenerálási és ellenőrzési technikák bevezetésével nagyban javíthatjuk a produktivitást.

A mérés során az Eclipse Modeling Framework (EMF) eszközzel készített modellek kezelését mutatjuk be, amely napjaink legnépszerűbb modellező keretrendszere. Az EMFben megtalálható funkciókat a már korábban megismert Yakindu állapotgép-modellező eszközön szemléltetjük.

Modellezési Alapismeretek

A technológiai részletek bemutatása előtt tisztázzunk pár modellezéssel kapcsolatos alapfogalmat! Egy modellezőeszköznek alapvetően két célja van: egyrészt hogy modellezőnyelvekhez nyújtson szerkesztőfelületet, másrészt hogy az elkészült modellek felhasználásával hasznos műveleteket hajtsanak végre automatikusan.

Yakindu esetén például állapotgépeket modellezhetünk. Az **1. ábrán** látható állapotgép egy sakkórának a leegyszerűsített viselkedésmodelljét írja le. Az állapotgép 3 bemenő eseményre reagál: `start`, `white` és `black` eseményekre, melyekkel elindíthatjuk a játékot, valamint a játékosok átadhatják egymásnak a kört. Az állapotgép definiál még két egész értékű változót is (`whiteTime` és `blackTime`), melyek értéke másodpercenként csökken az adott játékos körében. Az elkészült állapotgépeket a tervezőeszköz segítségével a fejlesztés során szimulációval ellenőrizhetjük, illetve automatikusan C vagy Java kódot generálhatunk. Amennyiben szükséges, Yakinduval kapcsolatos ismeretek felfrissítheti az **Állapotalapú modellezés**[1] című segédanyag áttekintésével.



Ábra 1: Példa Yakindu állapotgép

[1] http://docs.inf.mit.bme.hu/remo-jegyzet/allapot-alapu-modellezes.pdf

Egy *modellezési nyelv* megtervezésekor célunk, hogy megadjuk a tervezőeszköz által szerkeszthető összes modell halmazát. Az ehhez szükséges szabályokat kétféle osztályba soroljuk:

- A modellek logikai felépítésével kapcsolatos szabályokat *absztrakt szintaxisnak* nevezzük. Ebbe az osztályba tartozó szabályok határozzák meg például a lehetséges típusokat és elemek közötti lehetséges kapcsolatokat (Yakindu esetén milyen állapotok lehetnek és melyik két állapot között mehet tranzíció). Ezen kívül gyakran absztrakt szintaxisnak nevezzük egy modellnek az ábrázolástól mentes felépítését is.
- A modellek megjelenítésével kapcsolatos szabályokat *konkrét szintaxisnak* nevezzük. Ebbe az osztályba tartozó szabályok határozzák meg például a kulcsszavakat, a kommentezési szabályokat vagy a diagramokban szereplő grafikus objektumok színét és alakját.

Amikor modellezőeszközökhöz írunk saját programot, általában az absztrakt szintaxissal kapcsolatos részletek fontosak csak; kódgenerálás esetén sem számít például, hogy a diagramon melyik oldalán voltak az állapotok. Egy modellezőnyelv absztrakt szintaxisát tipikusan *metamodellel* és *jólformáltsági kényszerekkel* határozhatjuk meg.

- A metamodell meghatározza a modellezési nyelv legfontosabb fogalmait, kapcsolatait, és a modellek alapvető szerkezetét.
- Jólformáltsági kényszerek kiegészítik a metamodellt egyéb szabályokkal, amelyeket minden helyes modellnek teljesítenie kell.

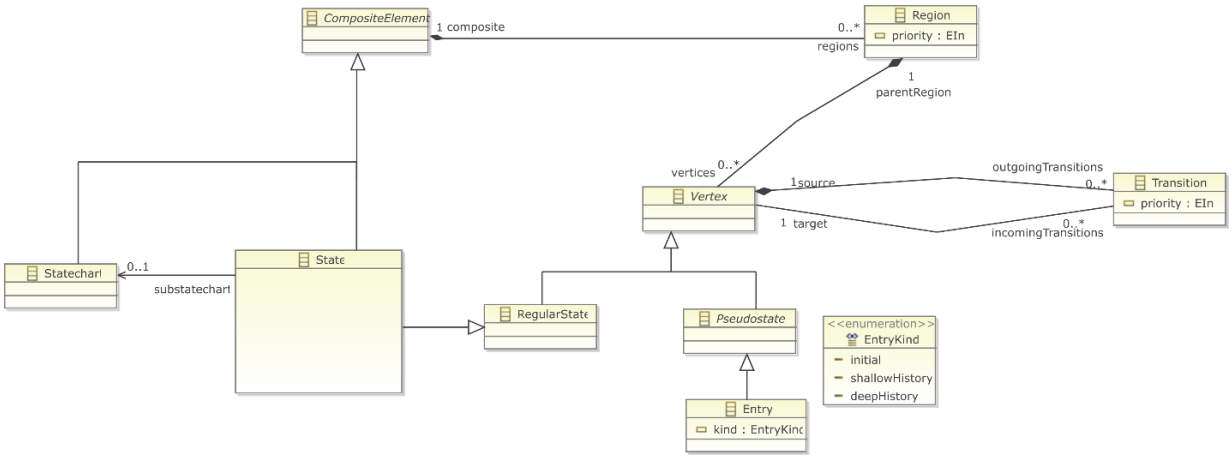
Eclipse Modeling Framework

Az Eclipse Modeling Framework (EMF) egy széles körben elterjedt, Eclipse tervezőeszközbe épülő metamodellező keretrendszer. Segítségével UML osztálydiagramokhoz hasonló jelölési rendszerrel határozzák meg metamodelleket, amelyből a keretrendszer automatikusan előállítja a modellezőeszközhöz szükséges forráskódot. A **2. ábrán** látható a Yakindu tervezőeszköz által használt metamodell egy részlete, melyet közvetlenül a modellezőeszközből exportáltunk. Az EMF metamodellek a következő elemekből épülnek fel:

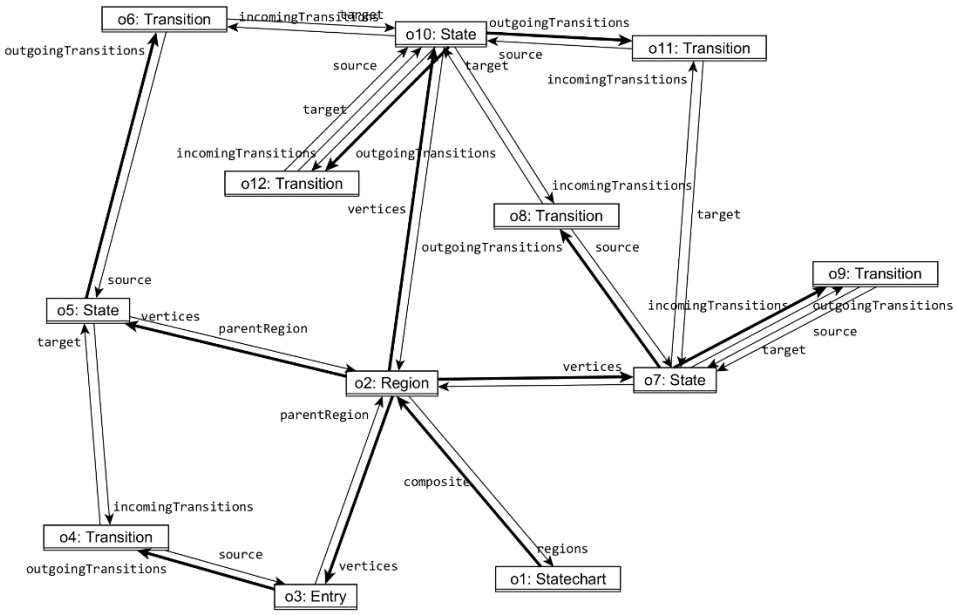
- Osztályok (`EClass`): A modellezőnyelv alapfogalmait (mint például az állapot és a tranzíció a **2. ábrán**) osztályokkal jelöljük. Osztályok között *öröklési* kapcsolatok lehetnek (többszörös öröklés megengedett), illetve néhány osztály *absztrakt*, ami dőlt betűvel van jelölve (például `Vertex`). Ezek jelentése megegyezik az UML osztálydiagramoknál megismertekkel.
- Enum típusok (`EEnum`): Olyan típus, aminek felsoroljuk a lehetséges értékeit. `EntryKind` esetén egy kezdőállapot lehet `initial`, vagy valamilyen history állapot (`shallowHistory` illetve `deepHistory`). A különböző értékeket `EEnumLiteral`nak nevezzük.
- Referenciák (`EReference`): Osztályok közötti kapcsolatokat referenciákkal modellezhetjük. A referenciák *multiplicitását* megadhatjuk `min..max` formában, ahol a * szimbólum jelöli a korlátlan mennyiséget. Néhány referencia *tartalmazási élnek* van jelölve a referencia kiindulásánál lévő fekete rombusszal (mint `vertices`). Fontos megjegyezni, hogy EMF esetén a modellekben lévő összes objektumnak a tartalmazási élekkel egy fát kell alkotniuk! Két egymással mindig ellentétes irányba mutató *inverz referenciát* (mint például egy tranzíció esetén a `source` és `outgoingTransitions`) egy vonallal jelölünk.

- Attribútumok (`EAttribute`): Osztályok és primitív adattípusok (`Integer`, `String`) illetve `EEnum`ok közötti kapcsolatokat attribútumokkal jelöljük. Attribútumok is rendelkeznek multiplicitással.

Tekintsük át a Yakindu modellek felépítéséről! A **2. ábrán** látható a Yakindu állapotgépek metamodellje, a **3. ábrán** pedig a korábban bemutatott állapotgép példának a gráfszerű kirajzolása, ahol a csomópontok az objektumokat, az élek a referenciákat jelölik. `CompositeElement` jelöli azokat az elemeket, amelyek több állapotot is tárolhatnak. Ilyen például a `Statechart`, ami az egész állapotgépet reprezentálja, illetve a közönséges `State`, ami a rendes állapotokat jelöli (mint a **2. ábrán** az `Init`, `White` és a `Black`). Egy `CompositeElement` több régiót (`Region`) tartalmazhat, amelyek mindegyike több csomópontot (`Vertex`) tartalmazhat. A csomópontokat közük a Ábra 2két részre osztja: a közönséges állapotokra és olyan pszeudoállapotokra, mint például a kezdőállapot (`Entry`). A csomópontok között tranzíciók vezetnek (`Transition`), ahol a `source` referencia mutatja az az előző, a `target` referencia a következő állapotot. Egy metamodell definiálhat `enum` típushoz hasonló felsorolásokat is, az `EntryKind` például a kezdőállapot fajtáját határozza meg.



Ábra 2: Yakindu állapotgép metamodell legfontosabb elemei.



Ábra 3: Yakindu állapotgép példánymodell gráfként ábrázolása

Modellek kezelése forráskódból

Amint korábban említettük, az EMF keretrendszer a metamodellből automatikusan elkészíti a modellező-eszközhöz szükséges forráskód nagy részét. A továbbiakban bemutatjuk, hogy milyen forráskódot generál az EMF, és hogy hogyan lehet kiegészíteni saját programrészletekkel a modellezőeszközt:

- `Factory` osztály: EMF minden metamodellhez generál egy singleton factory osztályt, amellyel példányosíthatjuk a modellünk objektumait. Például Yakinduban egy állapotot az alábbi kódrészlettel hozhatunk létre:

```
import org.yakindu.sct.model.sgraph.SGraphFactory;
State s = SGraphFactory.eInstance().createState();
```

- `EClass`ok: Metamodellünk minden osztályából egy Java interfész (például `State.java`) és egy osztály (`StateImpl.java`) generálódik, úgy, hogy betartsa metamodellben felírt öröklési szabályokat. Ökölszabályként elmondható, hogy a saját kódunkban mindig az csak interfészt használjuk. A generált osztályok mindegyike leszármazik az `EObject` osztályból, ami több modellezéssel kapcsolatos hasznos metódust is tartalmaz.
- EReference és EAttribute: Egy osztály minden referenciájához és attribútumához `getter` és `setter` függvényeket kapunk, amelyek segítségével lekérdezhetjük vagy módosíthatjuk a modellünket. Amennyiben az adott tulajdonság multiplicitása nem 1, úgy egy megfelelő típusú listát kapunk vissza, amit a megszokott listaműveletekkel járhatunk be vagy módosíthatunk.

```
public interface Vertex {
	Region getParentRegion();
	void setParentRegion(Region value);
	EList<Transition> getIncomingTransitions();
	EList<Transition> getOutgoingTransitions();
```

```
}
```

Fontos megjegyezni, hogy a tartalmazási fa hierarchiát és az inverz referenciákat az implementáció automatikusan kezeli. Tehát az alábbi példa esetén a `parentRegion` referencián kívül az inverz `vertices` referencia is automatikusan átállítódik, és a csomópont átkerül egy másik régióba.

```
Vertex v = …
Region other = …
v.setParentRegion(other);
```

- Tartalmazási hierarchia bejárása: EMF-ben támogatást kapunk a modellek tartalmazási hierarchia szerinti bejárására az `eAllContents` függvény segítségével. Az alábbi kódrészlet például bejárja az állapotgépünket, és kiírja az összes állapotának nevét.

```
Statechart statechart = …
TreeIterator<EObject> iterator = statechart.eAllContents();
while (iterator.hasNext()) {
	EObject content = iterator.next();
	if(content instanceof State) {
		State state = (State) content;
		System.out.println(state.getName());
	}
}
```

- Modellek mentése és betöltése: Az EMF keretrendszer olyan osztályokat generál a metamodellből, amelyek beépítve támogatják a modellek kimentését vagy betöltését. A modelljeinket az alábbi kódrészlet segítségével menthetjük ki:

```
// létrehozunk egy resource-t a megfelelő elérési úttal
Resource resource =
(new ResourceSetImpl()).createResource(URI.createURI("elérési út"));
// beletesszük az elmentésre váró modell
// tartalmazási hierarchiájának gyökerét
resource.getContents().add(statechart);
// Elmentjük
resource.save(null);
```

A modell betöltését az alábbi kódrészlettel végezhetjük:

```
Resource resource = (new ResourceSetImpl()).getResource(URI.cre-
ateURI("elérési út"), true);
// lekérdezzük a Resource tartalmát
EObject content = resource.getContents().get(0);
// a content tartalmát kasztoljuk az általunk várt típusra
Statechart statechart = (Statechart) content;
```

Hivatkozások

[1] Rendszermodellezés segédanyag: Állapotalapú modellezés, http://docs.inf.mit.bme.hu/remo-jegyzet/allapot-alapu-modellezes.pdf

[2] Modell alapú rendszertervezés: Domain-Specific Modeling, http://inf.mit.bme.hu/sites/default/files/materials/category/kateg%C3%B3ria/oktat%C3%A1s/msc-t%C3%A1rgyak/model-lalap%C3%BA-szoftvertervez%C3%A9s/16/03_MDSD_DSM_EMF_2016.pdf

[3] Modell alapú rendszertervezés: Introduction to the Eclipse Modeling Framework, https://github.com/FTSRG/lecture-notes/wiki/2016_emf

[4] Eclipse Modeling Framework (EMF) – Tutorial, http://www.vogella.com/tutorials/EclipseEMF/article.html