Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő Diplomaterv sablon dokumentum tartalma. Ügyeljen a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

* a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
* mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést, ill. diplomatervezést kívánunk!

FeladatkiÍrás

A feladatkiírást a tanszék saját előírása szerint vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.

C:\Users\szarnyasg\Downloads\bme_logo_nagy.eps

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Készítette

Haraszin PéterKonzulens

Dr. Bergmann Gábor

2015

Tartalomjegyzék

[Összefoglaló 6](#_Toc417232079)

[Abstract 7](#_Toc417232080)

[1. Bevezetés 8](#_Toc417232081)

[1.1. Formázási tudnivalók 8](#_Toc417232082)

[1.1.1. Címsorok 8](#_Toc417232083)

[1.1.2. Képek 8](#_Toc417232084)

[1.1.3. Táblázatok 8](#_Toc417232085)

[1.1.4. Kódrészletek 8](#_Toc417232086)

[1.1.5. Irodalomjegyzék 9](#_Toc417232087)

[1.1.6. Margók 9](#_Toc417232088)

[2. Utolsó simítások 10](#_Toc417232089)

[3. Összefoglalás 11](#_Toc417232090)

[Köszönetnyilvánítás 12](#_Toc417232091)

[Ábrák jegyzéke 13](#_Toc417232092)

[Táblázatok jegyzéke 14](#_Toc417232093)

[Irodalomjegyzék 15](#_Toc417232094)

[Függelék 16](#_Toc417232095)

Hallgatói nyilatkozat

Alulírott Haraszin Péter, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2015. 04. 21.

Haraszin Péter

# Összefoglaló

A mérnöki modellezésben jelentős szerep jut a viselkedésmodellezésnek, különösen az állapot alapú (Statechart) és folyamat alapú (Business Process) formalizmusoknak. A tanszéken folyó oktatásban már eddig is megjelentek ezek a modellek, például önálló házi feladat lehet egy megadott szöveges specifikációnak megfelelően viselkedő modell elkészítése.

A programozás oktatásában elterjedtek a házi feladatok automatikus értékelését végző eszközök; ez az ellenőrzés tesztesetek alapján, tehát dinamikusan (a működést végrehajtva) és szúrópróbaszerűen történik. Viselkedésmodell jellegű feladatoknál erre még nem ismert kiforrott megoldás. Ugyanakkor a viselkedésmodellek jellegükből adódóan a működésük statikus (futtatás nélküli) elemzését is lehetővé tehetnek bizonyos szempontok szerint.

A téma motivációját az adja, hogy a BME VIK mérnökinformatikus BSc képzésében oktatott Rendszermodellezés c. tárgy teljesítésének egyik követelménye az, hogy a hallgatók egy konkrét viselkedésmodellező technológiával (Yakindu Statechart Tools) készítsenek el egy állapotgépmodellt. A tárgy 2015 tavaszától a korábbi hetedik féléves oktatás helyett az új tanulmányi rend keretében már a második félévtől lesz megtartva; ebből következően a korábbiaknál jóval nagyobb hallgatói létszámra lehet számítani, és ez egyben jóval több ellenőrizendő házi feladatot is jelent.

A szakdolgozat célja az, hogy a modellező eszközzel elkészített házi feladatok tömeges ellenőrzését sikerüljön hatékonyabbá tenni, és az eleve rossz megoldásokat kiszűrni, ezzel az ellenőrzéssel járó terhek egy részét levéve az oktatók válláról. A hallgató feladata megismerkedni a Yakindu Statechart Tools technológiával felhasználói és fejlesztői szinten, és megvizsgálni az eszközzel elkészíthető modellek automatikus feldolgozásának, statikus ellenőrzésének, dinamikus futtatásának (tesztelésének) módját és lehetőségeit. A dolgozatban meg kell tervezni, meg kell valósítani és ki kell értékelni egy szoftvereszközt, amely képes a Yakindu feladatmegoldások tömeges kiértékelésére statikus és dinamikus eljárásokkal.

# Abstract

English abstract of the thesis work. This summarises the content of the thesis in 0.5–1 pages and is uploaded to the Thesis Work Portal as well.

# Bevezetés

A következő fejezet pár példán keresztül bemutatja a diplomatervekben és szakdolgozatokban szokásosan előkerülő formázások megvalósítását.

## Formázási tudnivalók

A dokumentum folyószövegéhez használjuk a **Normál** (angol Word esetén Normal) stílust.

### Címsorok

A fejezetcímek esetén a **Címsor 1–4** (Heading 1–4) stílusokat használjuk.

### Képek

A képhez használjuk a **Kép** stílust. Képaláírást a képen jobb gombbal kattintva a **Képaláírás beszúrása…** opcióval adhatunk hozzá, így az automatikusan **Képaláírás** (Caption) stílusú lesz.



1.1. ábra. Példa képaláírásra

Képek hivatkozásához jelöljük ki a képaláírásban a sorszámot (pl. „1.1.”), majd kattintsunk a **Könyvjelző** gombra, majd hozzunk létre egy könyvjelzőt (pl. „bmelogo” névvel). Ezután a **Kereszthivatkozás** gombra kattintva a **Hivatkozástípus**t állítsuk **Könyvjelző**re és válasszuk ki a **bmelogo** könyvjelzőt. Így ehhez hasonló hivatkozásokat készíthetünk: lásd az XXX. ábrán.

### Táblázatok

A dolgozatban szereplő táblázatokat az 1.1. táblázat mintájára érdemes elkészíteni.

|  |  |  |
| --- | --- | --- |
| ****Fejléc**** | ****Opció A**** | ****Opció B**** |
| 1. sor |  |  |
| 2. sor |  |  |

1.1. táblázat. Példa táblázat feliratára

### Kódrészletek

Kódrészletek beillesztése esetén használjuk a **Kód** stílust.

using System;

namespace MyApp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Szia Világ!");

}

}

}

### Irodalomjegyzék

Az irodalomjegyzék kezelése többféleképpen is megoldható, az alábbiakban két egyszerű módszert ismertetünk.

#### Kereszthivatkozásokkal

A kereszthivatkozásokkal történő irodalomjegyzék egy megfelelően formázott felsorolás, melynek egyes elemeire (bekezdéseire) mutatnak hivatkozások. Jelen dokumentum ezt a megközelítést alkalmazza.

Az Irodalomjegyzékben szereplő hivatkozásokat **Irodalomjegyzék bejegyzés** stílussal formázzuk, a címüket pedig a **Kiemelés** stílussal emeljük ki.

A szövegbe a hivatkozásokat a **Kereszthivatkozás beszúrása** (Insert cross-reference) funkcióval helyezzük el (példa egy így beszúrt hivatkozásra: [1]), így azok később frissíthetők a hivatkozások átrendezése esetén (lásd 2. fejezet).

#### Források kezelése

A kereszthivatkozások alternatívája, hogy a hivatkozott műveket először felvesszük a szerkesztőprogram adatbázisába. Új műveket a **Hivatkozás** fülön a **Források kezelése** alatt az **Új…** gombbal vehetünk fel. A szerzőket érdemes a **Szerző** mező mellett található **Szerkesztés** gomb használatával felvenni.

Az irodalomjegyzéket az **Irodalomjegyzék** gomb alatt az **Irodalomjegyzék** opcióval szúrhatjuk be a dokumentumba. A hivatkozások stílusa a **Stílus** gomb alatt állítható be, a javasolt stílus az **IEEE**.

### Margók

Az **Oldalbeállítás** menüben ellenőrizzük a **Kötésmargó** beállítását. Amennyiben a dolgozat kétoldalas nyomtatással készül, a **Több oldal** beállításnál válasszuk a **Margók tükrözése** opciót.

# Technikai háttér

## Eclipse Modeling Framework (EMF)

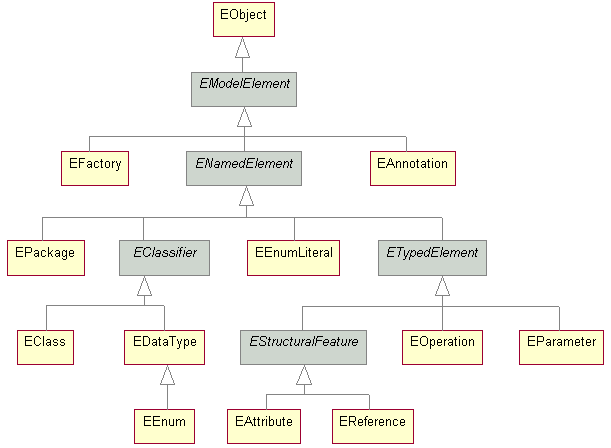
Az informatikai rendszerek komplexitásának növekedésével egyes programozói hibák elkerülése, a stabilitás és a termelékenység növelése érdekében egyre nagyobb igény merült fel a szoftverek különböző elemeinek magasabb absztrakciós szinteken történő kezelésére, a tervezés és a fejlesztés felgyorsítására, a hagyományos kódközpontú fejlesztésről az ún. modellvezérelt fejlesztési paradigmákra való áttérésre. A modellvezérelt fejlesztés középpontjában magasszintű modellek megalkotása szerepel, így a fejlesztő egy bonyolult rendszer egyszerűsített nézetét alakíthatja, amely egyszerűbbé teheti a tervezést. A modellkészítés megfelelő támogatással többnyire szigorú szabályokon alapszik, ezért az elkészített rendszer megbízhatósága is növekedhet. A modellvezérelt fejlesztés során alapvető elvárás a modellek és ezáltal a programstruktúra vizuális megjeleníthetősége, illetve az elkészített modellekből történő automatikus kódgenerálás különböző programnyelvekre. Bármilyen modellváltozás a generált kód megváltozásával is kell, hogy járjon, így a generált kód az elkészített modellel szinkronban kell, hogy legyen. Napjainkban a modellvezérelt fejlesztéshez használt eszközöknél további elvárás a modellek futtatása, különböző eszközök segítségével történő automatizált elemzése és ellenőrzése, illetve a modellekből történő automatikus kódgenerálás is valamilyen programozási nyelvre. Így a fejlesztő az általa akár grafikus felületen elkészített modellnek megkaphatja egy alapvető keretét programkód formájában, ezzel jelentősen leegyszerűsítve az implementációt is.

A modellvezérelt fejlesztési paradigmák alapja az Object Management Group (OMG) konzorcium által 2001-ben bevezetett ún. modellvezérelt architektúra[[1]](#footnote-1) fogalma, amely egy számos technológiát magába foglaló, később szabványosított szoftvertervezési megközelítés. Ennek középpontjában az áll, hogy a rendszer alapvető működését a tervezés során egy platformfüggetlen modell (platform-independent model (PIM)) írja le, majd ebből a konkrét megvalósítás során szülessen egy ettől elkülönülő platformspecifikus modell (platform-specific model (PSM)) adott céleszköz(ök)től függően. Ez a leképezés szintén modellekkel írható le, és a két modell közötti átfordítást általában automatizált eszközök végzik el. Az éles elkülönítés előnye az, hogy a rendszer egészére vonatkozó tervezési hibák egy része már a rendszerépítés korai fázisában kiszűrhető, hiszen a modell helyességére vonatkozó ellenőrzések egy platformtól nem függő modellen végezhetők el.

Az ismertetett technológiák igen robusztus infrastruktúrát igényelnek. Az OMG modellezési szabványaira építve (ahhoz nem szigorúan ragaszkodva) az Eclipse Foundation egy modellezési keretrendszert hozott létre, az ún. Eclipse Modeling Frameworköt. Ez egy Eclipse fejlesztőkörnyezetre épülő, mára nagyon széles körben, ipari célokra is használt, nyílt forrású modellező és kódgenerálást is lehetővé tévő keretrendszer, amellyel strukturált adatmodellen alapuló eszközök, alkalmazások készíthetőek. Alapja a Java programozási nyelv. A modellek specifikációjának leírása az OMG konzorcium által szabványosított XML Metadata Interchange (XMI) segítségével történik, amellyel a különböző eszközök közötti platformfüggetlen metaadatcsere válik lehetővé XML-formátumban. Az adatok mentése és visszatöltése tehát ilyen módon valósul meg.

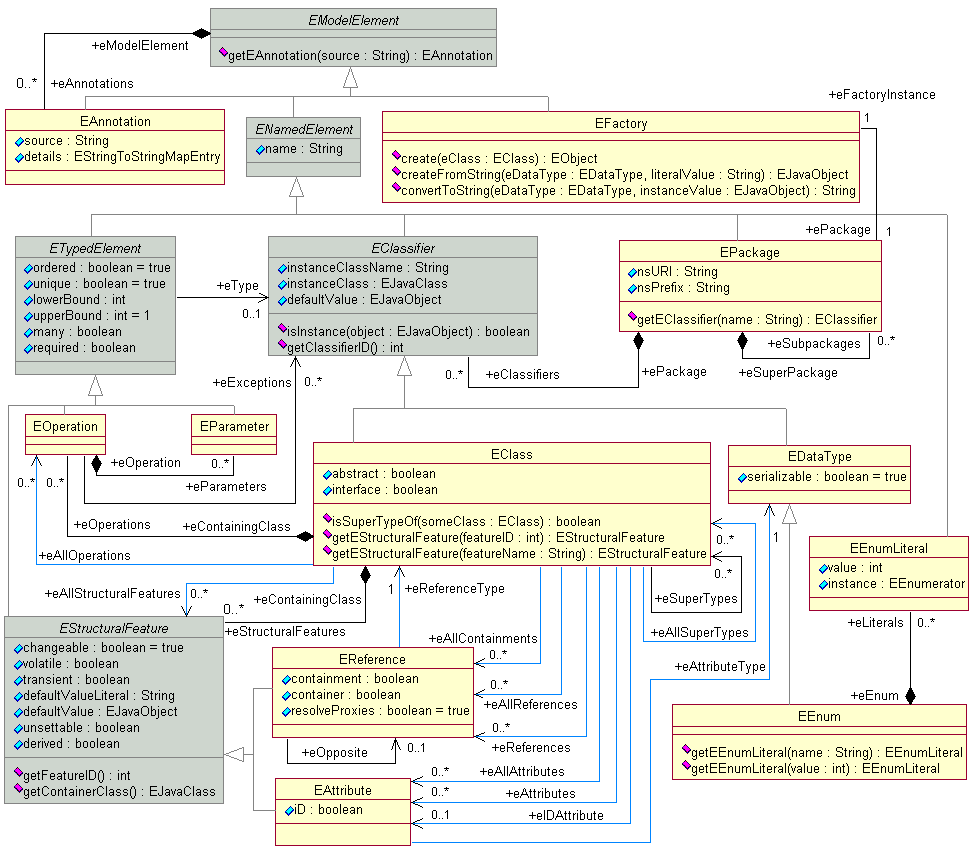
Az EMF lehetőséget biztosít több formátumból (pl. UML-leírás, adott sémára épülő XML-dokumentum, kommentekben annotált Java-kód, a saját XMI-formátuma, stb.) történő modellimportálásra. A modell vizuális elkészítésére egy egyszerű grafikus alapú szerkesztő is a fejlesztők rendelkezésére áll. A modellező keretrendszer az elkészített modellekből a Java-osztályok automatikus legenerálására futásidejű támogatást nyújt, valamint lehetőséget biztosít a modell kódból történő lekérdezésére és manipulálására is.

Az EMF segítségével elkészített modellekhez szükség van egy ún. metamodellre is, amely magának a modellezésnek a felsőbb szintű szintaxisát írja le, tehát azt, hogy miként épülnek fel ezek a struktúrák. Az EMF magját képező metamodell az ún. Ecore. Az Ecore a nyelvben definiált osztályokról ad információt, felépítése a Unified Modeling Language (UML) általános célú modellező nyelv osztálydiagramjainak felépítésére hasonlít: az osztályok között hierarchia adható meg, a köztük lévő kapcsolatok pedig az osztályok összekötésével adhatók meg. Az Ecore-komponensek hierarchiája a 2. ábrán látható.



2. ábra: Az Ecore-komponensek hierarchiája[[2]](#footnote-2)

Az Ecore-komponensek relációi, attribútumai és metódusai a 3. ábrán láthatók.



3. ábra: Az Ecore-komponensek relációi, attribútumai és metódusai[[3]](#footnote-3)

A későbbi fejezetek megértéséhez röviden elmagyarázom az egyes komponensek szerepét[[4]](#footnote-4):

* EObject: a Java általános Object típusához hasonlóan itt is létezik egy általános „objektum” gyökérkomponens, amely az Ecore összes egyéb összetevőjének őse
* EModelElement: absztrakt ősosztály minden egyéb osztályhoz az Ecore-ban
* EFactory: osztályok példányosításáért felelős, illetve adattípus-példányok karakterláncból vagy karakterláncokká történő konvertálásáért
* EAnnotation: az Ecore-modellek egyes objektumaihoz az alapvető információkon kívül további információk adhatók annotációk segítségével, ez az osztály e feladat ellátásáért felelős
* ENamedElement: az Ecore-modellben szereplő legtöbb osztály absztrakt ősosztálya, amely a „name” (név) attribútumot definiálja, hogy a modellek egyes elemei névvel legyenek elláthatók
* EPackage: hasonlóan a Java csomag fogalmához, az Ecore-modellek egyes elemei is csomagokba, vagyis egy tartalmazási egységbe szervezettek, ennek az osztálynak egy példánya azt a csomagot határozza meg, amelybe az adott elem sorolható
* EClassifier: az osztályokhoz és adattípusokhoz tartozó általános absztrakt ősosztály
  + EClass: egy osztályt reprezentál, 0 vagy több attribútummal és 0 vagy több referenciával ellátva
  + EDataType: adattípus, egy attribútum típusát reprezentálja (pl. int, float, java.util.Date)
    - EEnum: értékek egy listáját felsoroló típus (enumeráció)
* EEnumLiteral: az EEnum típushoz tartozó szöveges vagy egész számot reprezentáló literál
* ETypedElement: absztrakt ősosztály a
  + EStructuralFeature: egy osztály strukturális jellemzőihez (pl. attribútumokhoz) tartozó absztrakt ősosztály
    - EAttribute: osztályok attribútumai, névvel és típussal ellátva
    - EReference: referencia más osztályokra
  + EOperation: egy osztályhoz tartozó metódusokat meghatározó osztály
  + EParameter: egy metódus bemeneti paramétere

## Yakindu

A Yakindu Statechart Tools egy nyílt forráskódú, Eclipse-beépülőként elérhető modellező eszköz, mellyel eseményvezérelt rendszerek specifikálásához, fejlesztéséhez tudunk állapotgépeket készíteni.

Az állapotgépmodellt az Eclipse fejlesztőkörnyezetben tudjuk elkészíteni. Ehhez a modellező nézetben egy jól áttekinthető, felhasználóbarát grafikus felületet kapunk, amelyen az állapotgép különböző elemeit egy eszköztárról tudjuk behúzni a tervezőfelületre. Állapotokat rakhatunk le, közéjük irányított éleket húzhatunk be, jelezve az állapotváltás irányát, ezekre az élekre az egyik állapotból a másikba történő átmenet feltételét jelentő eseményeket írhatunk fel, stb. (TODO: ide egy példa kapásból képpel, képaláírással!)

Kapunk egy szövegmezőt is, amelyen keresztül egy ilyen célra használható statechart nyelv segítségével különböző interfészeket definiálhatunk, és ezek leírásait adhatjuk meg. Itt bejövő, ill. kimenő eseményeket, változókat, operátorokat, stb. határozhatunk meg, amelyek az állapotgép működ(tet)ésében lehetnek hasznunkra. (TODO: ide még egy példa képpel.)

Az eszköz validációt is támogat, így ellenőrizhető a modellünk helyessége, például az, hogy nem használtunk-e olyan eseményt az állapotok közti éleken, amelyet semelyik interfésznél nem definiáltunk. Ezenkívül szimulálhatjuk az állapotgép működését, melynek során animálva láthatjuk az egyik állapotból a másikba történő átmenetet, amennyiben pl. kiváltunk egy eseményt.

Nagyon fontos szolgáltatása a Yakindunak, hogy támogat különböző platformokra, programozási nyelvekre történő kódgenerálást (C, C++, Java). Ezzel az elkészített állapotgépünk működéséhez szükséges alapvető kódbázist, egy keretet generál le a program. A feladat megoldása során ezt a szolgáltatást is ki fogjuk használni, hiszen a hallgató által elkészített állapotgép viselkedését is szeretnénk kódból ellenőrizni.

A „motorháztető alatt” az adatmodell összekötött objektumok gráfja, amit szabványosan az Eclipse Modeling Framework (EMF) technológiával valósítottak meg, amelynek célja, hogy lehetővé tegye modellek magas szintű, akár grafikus megalkotását és menedzselését. A felhasználó által összekattintott állapotgép tehát egy fastruktúrába rendezett EMF-modellre képeződik le. Ez az egész alapja, amely akkor válik igazán érdekessé, amikor az ellenőrzés első lépéseként bejárjuk ezt a modellt a statikus ellenőrzés során. A modell alapvető felépítését tehát ismerni kell a feladat elvégzéséhez.

Ha a Yakindu-állapotgéphez tartozó, \*.sct-kiterjesztésű fájlra jobb egérgombbal kattintunk, majd Open with… > Other > Sample Ecore Model Editor menüpontra kattintunk, akkor a (TODO: ide egy szemléltető ábra az objektumhierarchiáról, hogy el tudják képzelni, miről van szó) látható hierarchikus struktúrához hasonlót figyelhetünk meg:

TODO: kép

Az ábrán látható faszerkezetet szoktuk az ún. „absztrakt szintaxisnak” nevezni, ami a nyelvtani szabályokat tartalmazza, a korábban látható, kattintgatós rész pedig az ún. „konkrét szintaxis”, ami a megjelenítéshez tartozik. A képen látható módon, hierarchikusan épül fel az adatmodell, objektumok és attribútumok, relációk olvashatók ki belőle.

## Feladatok

A házi feladatok ellenőrzéséhez definiálnunk kell a teendőket. Alapvetően két nagyobb részfeladatra oszlik a munka, először azt szeretnénk ellenőrizni, hogy bizonyos szabályoknak megfelel-e az állapotgép, másodszor pedig azt, hogy az elvártak szerint viselkedik-e. Az alábbiakban ezt a két részfeladatot fogom részletesebben kifejteni.

### Statikus ellenőrzés

A hallgatónak az állapotgép elkészítésekor be kell tartania bizonyos alapvető szabályokat, különben a házi feladata eleve nem fogadható el. A szabályok betartásának ellenőrzése tehát egy előzetes szűrő, ha ezeket a szabályokat az illető valahol megsértette, akkor már nem is érdemes tovább vizsgálódni.

Példa az alapvető szabályokra:

* Az interfészdefiníciók egyezzenek az általunk meghatározottakkal.
  + A feladat kiadásakor a tanszéki munkatársak előre el fogják készíteni az interfészdefiníciókat. Ezzel a hallgatónak tehát nem szükséges foglalkoznia, sőt, kifejezetten tiltott újabb elemet beleraknia (nem szabad persze problémát okoznia szóköz, sortörés vagy tabulátor – akár véletlen – beszúrásának), a neveket módosítania, stb. Így kiszűrjük az esetleges, még fel nem fedezett visszaélési lehetőségeket is (hiszen így a generált kódba nem kerülhet olyan rész, amire nem számítunk).
* Az állapotgép ne tartalmazzon tiltott elemet.
  + A hívási folyamatot modellező állapotgép a korábbi példákban tartalmazott időzítést, pl. az „after 2 s” valamelyik élre történő beírásával azt érhetjük el, hogy az állapot 2 másodperc elteltével egy másikba váltson. Egy lehetséges követelményként például azt is kitűzhetjük, hogy ilyen jellegű időzítést a modell egyáltalán ne tartalmazhasson, tehát a hallgató ne használhasson ilyen szolgáltatást (mert például a feladatleírás alapján nem szabadna ilyen eszközhöz folyamodni). Ha mégis megteszi, akkor a feladata nem elfogadható.

A szabályok listája tetszőlegesen bővíthető a felmerülő igények szerint. A feladatot az Eclipse Modeling Framework lehetőségeit kihasználva, Java-kóddal fogjuk végrehajtani, melynek segítségével bejárjuk a modellt, és a szabályoknak való megfelelést vizsgáljuk. Ezekhez JUnit-teszteseteket definiálunk (például megfogalmazzuk a „tartalmaz-e időzítést a modellünk?” kérdést egy tesztesetre leképezve), és kódból, automatizáltan fogjuk ezek lefutásának sikerességét vizsgálni. Ez alapján tudjuk eldönteni, hogy továbbléphetünk-e a későbbi feladatokra, vagy nem is érdemes további ellenőrzéseket végrehajtani.

### A viselkedés tesztelése

Azt is szeretnénk tesztelni, hogy az állapotgép az általunk meghatározott követelmények szerint, helyesen viselkedik-e: például eljut-e bizonyos állapotokba, egyes bemenetek hatására ad-e valamilyen kimenetet, és így tovább. Ezt a legenerált kód alapján fogjuk vizsgálni, tesztelés alapú ellenőrzéssel: ennek során azt kell megnézni, hogy a hallgató által elkészített állapotgép adott bemenetekre helyes (tehát a feladatleírásban definiált) kimenetet ad-e. Ehhez lesz egy előredefiniált tesztkészletünk. Például ha X bemenetre Y kimenetet kaptunk, és ez is az elvárt, akkor továbbléphetünk a következő ellenőrizendő pontra, ha azonban nem kaptuk meg az elvárt kimenetet a próbálkozásokra, akkor valószínűleg rossz a megoldás.

Egy lehetséges módszer az, hogy valamelyik interfészben különböző operationöket (műveleteket) definiálunk, amelyekre feliratkozhatunk ún. callback-metódusokkal, amire lehetőséget biztosít a Yakindu: olyan kódot generál, ami ezt lehetővé teszi. Ennek segítségével a különböző definiált callback-ekben olyan műveleteket hajthatunk végre, amelyek teljesülésére a JUnit-tesztekben majd vizsgálódhatunk: legegyszerűbb példaként ha adott egy kimenet kiírásra került a parancssori kimenetre, és az az elvártaknak megfelelő érték, akkor az adott rész átment a teszten, tehát a feladatleírás adott részét teljesítettük. Ezt nyilván végig kell vizsgálni az összes elvárt inputra. Ez úgy lehetséges, hogy az adott tesztelés során „manuálisan” hívjuk meg az egyes állapotváltásra szolgáló metódusokat. Egy ilyen módszerre láthatunk példát a hivatalos dokumentáció „Integration with client code” című részében.

### A megvalósítás

A megvalósítás során lesz egy Eclipse-projektünk, ami tartalmazni fogja egy referenciaként szolgáló állapotgép-fájlt (\*.sct-kiterjesztéssel) és a kódgenerálásért felelős fájlt (\*.sgen-fájlt). Amikor az ellenőrzés elindul, akkor

1. először is a projektet megtisztítjuk (clean request az IDE felé),
2. kitöröljük az src-gen könyvtár esetleges tartalmát (amibe a generált kódok kerülnek)
3. a hallgatótól kapott \*.sct-fájlt bemásoljuk a projektbe,
4. majd megpróbáljuk EMF-erőforrásként betölteni azt, megvizsgáljuk, hogy az interfészleírás egyezik-e a referenciamodell leírásával,
5. ezután megkíséreljük a bejárását, ezt követően esetleges tiltott elemek után keresgélünk,
6. majd ha ez mind megvan, legeneráltatjuk a kódot,
7. buildeltetjük a kódot
8. teszteljük az állapotgép viselkedését a generált kód manuális meghívogatásával, a bemenetekre adott kimenetek vizsgálatával.

A megvalósításhoz Eclipse-beépülőket hoztam létre. Ezek mind megtalálhatóak egy GitHub repository-ban: https://github.com/peterharaszin/hu.bme.mit.remo.scverif. A felelősségek szétválasztása érdekében feladatonként külön-külön plugineket hoztam létre:

* hu.bme.mit.remo.scverif.headless
  + Ez a projekt volt felelős a grafikus felület nélküli tesztelésekért (headless mode).
  + Függ a „hu.bme.mit.remo.scverif.processing” és „hu.bme.mit.remo.scverif.test.callhandling” projektektől.
  + Készítettem hozzá egy .product-fájlt , aminek segítségével a program könnyen tesztelhető az Overview fülön, a „Launch an Eclipse application” linkre kattintva. Ezenkívül az „Eclipe Product export wizard” segítségével gyorsan exportálható a projekt. Ez egy új Eclipse-példányt hoz létre, amely a ténylegesen szükséges plugineket tölti be, de nem indít el grafikus felületet. A futtatható állomány neve „RemoHeadlessLauncher” (Windows-környezetben kiegészítve az .exe-kiterjesztéssel). Ezt a -data "/path/to/workspace" argumentummal ellátva úgy tudjuk paraméterezni a futtatható állományt, hogy az adott munkaterület kerüljön betöltésre. Ez azért hasznos, mert így lehet egy szeparált workspace-ünk is, amiben egyedül az a projekt található, amellyel a tesztelést végre szeretnénk hajtani.
  + A headless futtatás során azonban exceptionök egész sorát kapom: mint a vizsgálatok során kiderült, ennek az az oka, hogy az Xtext builder grafikus felülethez kötődő függőségeket is tartalmaz, ami azt eredményezi, hogy gyakorlatilag ez esetben – jelenleg legalábbis – lehetetlen a teljesen „headless” működés. Ez tehát azt jelenti, hogy grafikus felület elindítására mindenképp szükség lesz – ettől még az automatizálás működik, csak egy grafikus felülettel nem rendelkező szerveren keresztül (például terminálból) a kód nem lesz futtatható.
* hu.bme.mit.remo.scverif.ui
  + Ebben a projektben találhatóak a felhasználói felülethez kötődő részletek.
  + Definiáltam több kiterjesztési pontot („extension point”) is, hogy saját elemeket jelenítsek meg az Eclipse felhasználói felületének kontextusmenüjében is (jobb egérklikkre előugró menü) az egyes fájlokra kattintva, illetve az eszköztárra is saját ikonokat raktam ki, így onnan is elindítható a feldolgozás.
  + A jobb egérklikkre megjelenő menüpont esetén fájlkiterjesztés-ellenőrzést is végrehajtottam, és ettől függően jelenítettem csak meg az adott menüpontot. Az állapotgépfájlra való kattintás esetén a \*.sct-kiterjesztést, a kódgenerálásért felelős fájl esetén a \*.sgen-kiterjesztést kerestem. A menüpontok az elvártak szerint csak az adott kiterjesztésű fájlok esetén jelennek meg.
  + Az eszköztáron a „Start processing ReMo homeworks” ikonra kattintva indul el a feldolgozás.
  + Ez elindítja a fentebb vázolt teljes folyamatot, ennek részeként például a jelenleg ebben a projektben lévő JUnit-teszteket is, és kiírja ezek eredményét (átment-e a teszten, vagy sem).
  + A DoRemoJobs osztály egy folyamatjelzővel mutatja, hogy épp hol tart a feldolgozás.
  + Az elindítás módja legegyszerűbben itt is a .product-fájlon keresztül lehetséges: rá kell kattintani hu.bme.mit.remo.scverif.ui.product fájlra kétszer, majd az „Overview” fülön a „Launch an Eclipse application” linkre kattintani, ez egy új Eclipse-példányt fog elindítani a szükséges pluginekkel. Így tesztelhetjük is a működést. Ezt követően kattintsunk az eszköztáron a „Start processing ReMo homeworks” ikonra.
* hu.bme.mit.remo.scverif.processing
  + Ebben a projektben alapvetően csak a feldolgozási műveletek zajlanak. Ez tartalmazza a StatechartAnalyzer osztályt is, ami az állapotgépmodell bejárásáért, a Time Event kereséséért és hasonlókért felelős.
* hu.bme.mit.remo.scverif.test.callhandling
  + Ez a bevezetőben már említett, telefonhívási folyamatot bemutató, egyszerűsített modell.
  + Ezen teszteljük a komplett projekt működését.

### További célok

Annak érdekében, hogy a házi feladatok ellenőrzésének folyamata megbízhatóan működjön, a feladat még további alapos finomításra szorul. Alaposabb statikus ellenőrzésnek kell alávetni a modellt, és a viselkedés tesztelését is szofisztikáltabb módszerekkel kell megoldani.

Egy távlati lehetőség a szúrópróbaszerű tesztek helyett a viselkedés kimerítő vizsgálata lehetne, ún. formális módszerek alkalmazásával. Ez egy összetett feladat, és komoly kihívásokat rejt magában, de jóval megbízhatóbbá teheti az ellenőrzési procedúrát.

# Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl + A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a „Hiba! A könyvjelző nem létezik.” szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó metaadatok kitöltése (szerző, cím, kulcsszavak stb.). Ez Word 2013 alatt a **Fájl** | **Információ** | **Tulajdonságok** | **Dokumentumpanel megjelenítése** gombra kattintva érhető el.
* Kinézet ellenőrzése PDF-ben: a dokumentum elkészítése után feltétlenül ellenőrizzük a kapott PDF dokumentumot is.

# Összefoglalás

TODO

# Ábrák jegyzéke

[1.1. ábra. Példa képaláírásra 8](#_Toc396824939)

# Táblázatok jegyzéke

[1.1. táblázat. Példa táblázat feliratára 8](#_Toc396824940)

# Irodalomjegyzék

1. Jeney Gábor, Hogyan néz ki egy igényes dokumentum? Néhány szóban az alapvető tipográﬁai szabályokról, Budapesti Műszaki és Gazdaságtudományi Egyetem, Híradástechnikai Tanszék, Budapest, 2007. május 9., online: <http://mcl.hu/~jeneyg/foliak.pdf>
2. William Strunk Jr., E. B. White, The Elements of Style, Fourth Edition, Longman, 4th edition, 1999.
3. Levendovszky, J., Jereb, L., Elek, Zs., Vesztergombi, Gy., Adaptive statistical algorithms in network reliability analysis, Performance Evaluation – Elsevier, Vol. 48, 2002, pp. 225-236
4. National Instruments, LabVIEW grafikus fejlesztői környezet leírása, <http://www.ni.com/> (2014. aug.)
5. Fowler, M., UML Distilled, 3rd edition, ISBN 0-321-19368-7, Addison-Wesley, 2004

# Függelék

A függelék szövege.

1. TODO: link [↑](#footnote-ref-1)
2. forrás: <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details>, letöltés dátuma: 2015. március 25. [↑](#footnote-ref-2)
3. forrás: <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#package_description>, letöltés dátuma: 2015. március 31. [↑](#footnote-ref-3)
4. <http://www.vogella.com/tutorials/EclipseEMF/article.html> [↑](#footnote-ref-4)