FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

MagicDraw állapottérképek formális verifikációja

Szakdolgozat

Készítette
Gáti László Dávid

Konzulens
Farkas Rebeka

2018. november 29.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Gáti László Dávid*, 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(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált 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.

Budapest, 2018. november 29.

Gáti László Dávid
hallgató

Kivonat

Komplex rendszerek tervezése során szükségessé válik hogy fejlett modellező eszközök álljanak rendelkezésre, hiszen gyakran szükség van ezen alkalmazások különböző funkcióinak igénybevételére, pl. validációra, kódgenerálásra, verifikációra. Az iparban az egyik legelterjedtebb eszköz a MagicDraw ami lehetőséget biztosít rendszerek tervezésére és ezek viselkedésének modellezésére. MagicDraw-t gyakran használnak kritikus rendszerek tervezésére, melyek esetén elengedhetetlen, hogy megbizonyosodjunk a rendszer működésének helyességéről.

Bár a MagicDraw már sok hasznos funkcióval rendelkezik, azonban a formális verifikációt nem támogatja. Az ipari alkalmazhatóság szempontjából jelentős előnyökkel járna ezeknek a módszereknek a támogatása. A Gamma egy a tanszéken fejlesztett modellező eszköz, amely nem rendelkezik a MagicDraw-éhoz hasonlóan kiterjedt eszközkészlettel, azonban viselkedésmodellek formális verifikációját támogatja. Dolgozatomban MagicDraw modellek formális verifikációját azáltal teszem lehetővé, hogy megvalósítok egy leképzést a két eszköz modelljei között.

Dolgozatomban bemutatom a Gamma és MagicDraw eszközöket és a leképzés megvalósításához szükséges háttérismereteket. Továbbá részletesen kitérek az implementáció során használt technológiákra és elvekre. A bemutatott módszert egy példán szemléltem és elméleti megfontolások alapján értékelem. Az elkészített eszköz lehetővé teszi modellek egy tágabb halmazának formális verifikációját.

Abstract

The development of complex systems makes it neccessary to gain access to such modern modelling tools that support functionalities like validation, code generation and verification. One of the most well-known tool in the indsutry is MagicDraw. A tool for desinging systems and to model their behavior. MagicDraw is often used during the development of fault tolerant systems. It is mandatory to ensure that such systems operate correctly.

Although MagicDraw has very rich a functionallity it lacks the ability to perform formal verification. The use of such methods would be very beneficial for industrial use. Gamma is a modelling tool developed at the Department which is not as extensive in functions as MagicDraw is but it does support the verification of behavioral models. In my thesis on supporting formal verification of MagicDraw models I rely on performing a transformation between the models of the two tools.

The thesis introduces both Gamma and Magicdraw tools and the background knowledge needed to understand the transformation method. Furhtermore I describe in details the concepts and techologies used in my implementation. I introduce the described method via an example and I evaluete the results based on theoretical considerations. The finished product enables the formal verifications on a wider set of models.

1. fejezet

Bevezetés

Nagyméretű és komplex tervezése során szükségessé válik, hogy olyan fejlett modellező eszközök álljanak rendelkezésünkre, melyek nem csupán gördülékenyebbé teszik a modell alapú fejlesztés menetét, de funkcióikkal lehetővé teszik rendszerek átfogó vizsgálatát: validációját azaz szintaktikai ellenőrzését, verifikálását a működésének helyességének ellenőrzését. A modellekből lehet kódot generálni, ami meggyorsítja a fejlesztést és csökkenti az implementáció során vétett programozói hibák számát.

Modelleket érdemes rajzok, diagramok segítségével definiálni, ezek ugyanis könnyebben megérthetők és hibákat is könnyebb észrevenni, mint egy kód alapú leírás esetében. A legelterjedtebb diagram alapú modellezési nyelvek az UML és a SysML melyek absztrakciók alkalmazásával rendszerek implementációtól független leírását teszik lehetővé. Ezek vagy hasonló modellezési nyelvek alkalmazása egy ipari projekt esetében kiemelkedően fontosak, ugyanis lehetővé teszik a rendszerek elemzését hibák felderítése céljából. Ezek felfedezése és kiküszöbölése még a tervezési fázisban nem jár jelentős többletköltségekkel hiszen nem kell az implementációt megváltoztatni, termékeket visszahívni egy esetleg utólag felfedezett tervezési hiba miatt.

Az iparban egy széleskörű alkalmazott eszköz a MagicDraw, ami többek között UML és SysML nyelveken teszi lehetővé modellek készítését. Ezeken magas szinten leírható egy rendszer felépítése és működése. MagicDraw-t gyakran alkalmaznak kritikus rendszerek modellezésére. Ezek esetében elengedhetetlen, hogy matematikai precizitással megbizonyosodjunk működésének helyességéről, hiszen egy meghibásodás komoly anyagi veszteségekkel vagy akár ember életek veszélyeztetésével járhat. Bár a MagicDraw már sok hasznos funkcióval rendelkezik, formális verifikációt végrehajtására még nincs lehetőség. Az ipar szempontjából jelentős előnyökkel járna viselkedésmodellek formális verifikációjának támogatottsága. A Gamma nevű eszköz a tanszéken fejlesztett állapottérképek modellezésére és ezek fejlesztésének támogatására készített eszköz, ami elsősorban Eclipses környezetekben alkalmazható. Ugyan a Gamma eszközkészlete nem annyira kiterjedt mint a MagicDraw-é, de az állapottérképek formális verifikációja a funkcióinak részét képezi. A Gamma saját nyelvet alkalmaz az állapottérképek leírására ami kifejezetten úgy lett megalkotva, hogy támogassa más nyelvek leképezhetőségét például Yakinduét. Dolgozatomban állapotgépek formális verifikációját azáltal teszem lehetővé, hogy egy leképzést biztosítok a MagicDraw és a Gamma modelljei között és a Gamma formális verifikációhoz kötődő funkcióit MagicDraw-n belül is végrehajthatóvá teszem.

Dolgozatomban bemutatom a Gamma és MagicDraw eszközöket és a leképzés megvalósításához szükséges háttérismereteket, részletesen kitérve azokra az alapvető ismeretekre és tehcnológiákra amelyek lehetővé teszik a feladat megvalósítását. Továbbá ismertetem azokat az elméleti is implementációs kihívásokat melyekkel a feladat végrehajtása során

szembe kellett néznem és az ezekre alkalmazott megoldásaimat, ezek előnyeivel és hátrányait egybevetve.

Hasonló munkák például maga a Gamma hasonló hiszen az eszköz már megvalósít egy leképzést saját nyelve: a már említett Yakindu - Gamma leképzést.

A dolgozat feléptése a következő: a 2. fejezetben bemutatom a munkám megértéséhez szükséges alapismereteket, a 3. fejezetben ismertetem az alkalmazott megoldást elméleti és gyakorlati szempontból.

2. fejezet

Háttérismeretek

2.1. Modellek

Modelleket a tudomány számos területén alkalmazunk, ez lehetővé teszi az előtt vizsgálni a megvalósítandó rendszert, hogy azt ténylegesen létre kellene hozni. A vizsgálatoknak számos módja és célja lehet. A látványtervező bemutat egy látványtervet és a megfelelő *stakeholderek* ezt értékelik, vagy egy rendszerről komplex matematikai módszerekkel kell eldönteni, hogy stabil-e. A modellek leírása sokféle módon történhet, de célszerű egy olyan standardizált jelölési rendszert alkalmazni, hogy a többi szakember is értelmezni, vizsgálni tudja a modellt.

A jelölési rendszer vagy modellezési formalizmus lehet egy szöveges leírás is, például egy programkód, de sokszor célszerű vizuális megoldást használni, szimbólumokat tartalmazó diagramokat alkalmazni. Ezek sok esetben kifejezőbbek, lényegre törőbbek, és elsősorban könnyebben értelmezhetőek mint a szöveges leírások.

2.2. Állapot alapú modellezés

Az állapottérkép egy diagram ami irányított gráfot tartalmaz, ahol a csomópontok állapotokat, az élek állapotátmeneteket definiálnak. Az így kapott gráfot állapotgépnek nevezzük. Az állapotátmenetek állapotok közötti lehetséges átjárásokat definiálnak és általában feltételhez kötöttek, ami jellemzően valamilyen esemény bekövetkezése vagy logikai feltétel teljesülése. A feltételek teljesülésekor az állapotváltás bekövetkezik, ezt szokás *tüzelésnek* is hívni. Az állapotgép legfőbb jellemzője, hogy adott időpillanatban melyik állapotok aktívak. Állapotváltások aktív állapotból történnek, ilyenkor a kiindulási állapot inaktívvá válik a célállapot pedig aktívvá (a kiinduló és célállapot megegyezhet).

A működés kezdetekor az első aktív állapotot kezdő állapotnak hívják, ez egy különleges ún. pszeudoállapot, ami az állapotgép belépési pontja és általában azonnal átléptetésre kerül egy másik állapotba.

2.2.1. Állapottérképek UML 2-ben

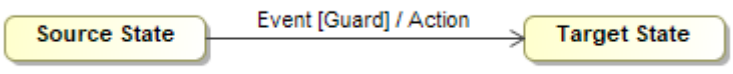
Előzőekben az állapot alapú modellezés alapjai kerültek bemutatásra. A következőkben az állapottérképek egy konkrét specifikációja kerül bemutatásra az UML 2 szerint.

- *Állapot (State)*: az állapottérképek csomópontjai, melyeket UML 2-ben is állapotoknak hívnak. Az állapotok rendelkeznek:
  - név: az állapot neve
  - be/kilépési akció: be és kilépés során végrehajtandó cselekvés.

- *Kezdő állapot (Initial State)*: pszeudoállapot, régiónként egy szerepelhet belőle és a régió belépési pontjaként szolgál. Jelölése fekete színezett kör (2.2 ábra).
- *Végállapot (Final State)*[1] (final state): A végső állapot, a régió terminálasi pontja, ha egy állapotgép összes régiója egy végső állapotba ért akkor az állapotgép is terminál. Szimbóluma fehér körben egy kisebb színezett fekete kör (2.2 ábra).
- *Terminál állapot (Terminal State)*: pszeudoállapot, ami az egész állapotgépet azonnal terminálja. Ezt hibák lekezelésére lehet például alkalmazni, jelölése kis kereszt (2.2 ábra).
- *Állapot átmenet (Transition)*: az állapottérképek élei, a lehetséges állapotváltozásokat definiálják. Az állapotátmenetek rendelkeznek:
  - Triggerekkel
  - Őrfeltételekkel
- *Trigger*: Állapot váltást kiváltó esemény amely lehet:
  - Változás esemény (Change Event): valamely változó értéknek a megváltozása.
  - Üzenet esemény (Message Event): üzenet típusú objektumnak az érkezése, ami ebben a kontextusban kérésnek felel meg. Az ilyen típusú kommunikáció kétféle eseménytől függ, az üzenet elküldésétől és annak küldőjétől és az üzenet fogadásától és fogadójától. A kérés lehet egy metódus hívás vagy egy jel (*Signal*) fogadása.
  - Időzítés esemény (Time Event): idő változásához kötött esemény.

Fontos megjegyezni, hogy a dolgozat nem tér ki részletesen az események kiváltásának kérdésére, valamint az események küldésénél és fogadásánál szerepet játszó portokra, és interfészekre, ezen elemek a dolgozat szempontjából irrelevánsnak tekinthetők.

- *Őrfeltétel (Guard)*: tágabb értelemben egy logikai kifejezés, melynek teljesülnie kell, hogy az adott állapotátmenet bekövetkezhessen. UML-ben ezek megszorításként *(Constraint)* vannak értelmezve, ebben az értelemben a megszorításnak való megfelelés az állapotváltás feltétele.
- *Akció*: Különböző események bekövetkezésekor, mint állapotváltások, belépés állapotokba, kilépés állapotokból, vagy maga az állapotban maradás, lehetőségünk van viselkedéseket végrehajtani. UML szerint ezek lehetnek: Activityk, Állapotgépek, Interakció[2] OpaqueBehavior[3]



2.1. ábra. Állapotok és köztük definiált állapotátmenet, triggerrel, őrfeltétellel és actionnel

[1] UML 2-ben a végső állapotot nem pszeudoállapotként hanem állapotként van definiálva https://www.omg.org/spec/UML/2.0

[2] Interakció modell elemek között, leírásához a jellegétől függően többféle diagram használható(Szekvencia, Kommunikációs, Időzítés).

[3] szöveges, UML-től eltérő nyelvvel specifikált viselkedés.

Gyakran előfordul, hogy általánosabb állapotot célszerű felbontani részállapotokra. Az egyszerű állapottérképek elemeivel, ez a fajta hierarchikus viszony az állapotok között nehezen ábrázolható, ezért célszerű további elemek használata.

- *Régió*: állapotokat tartalmazó egység, az állapottérkép mindig tartalmaz egy régiót amibe az állapotok definiálhatók. Régiók létezhetnek egymással párhuzamosan ilyenkor a végrehajtásuk párhuzamosan történik.
- *Összetett állapot(Composite State)*: ha az állapotnak vannak további belső állapotai is, ha az állapot aktív akkor legalább egy belső is aktív, ha az állapotgép egy állapotváltás hatására kilép a kompozit állapotból akkor a belső állapotokból is kilép.
- *History State*: olyan pszeudo állapot, amely egy régióban megjegyzi az utolsó aktív állapotot, kilépéskor, ha a régióba visszalépünk a history state visszaállítja a megjegyzett állapotot. Amennyiben nincs előző állapot az ő belőle húzott állapotátmenet cél állapota lesz aktív. Két féle History Statet különböztetünk meg Shallow és Deep Historyt. Előbbi csak adott régión belül jegyzi meg az állapotot míg utóbbi a tartalmazott régiók állapotait is megjegyzi és visszaállítja.

  A *HistroyStatek* szintaktikája a 2.2 ábrán látható.

2.2. ábra. Kezdőállapot, DeepHistory, ShallowHistory, Termináló állapot és Végállapot

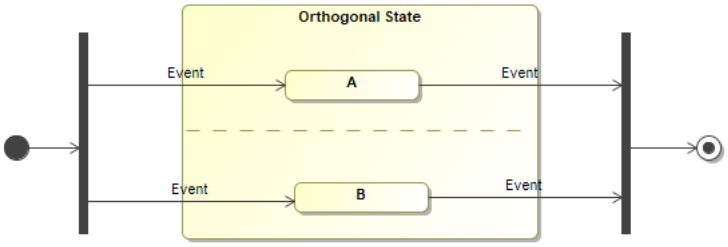
Rendszerünket egy időpillanatban több egymástól független állapot is jellemezheti. Ezt a viselkedést párhozamos régiók alkalmazásával lehet leírni.

- *Ortogonális állapot (Orthogonal State)*: olyan összetett állapot ami két vagy több régiót tartalmaz.
- *Fork*: pszeudoállapot, ami egy beérkező átmenetet szétbont több átmenetre, amiknek a cél állapotuk ortogonális régiókban találhatók. A kimenő átmeneteken nem lehet se trigger, sem pedig őrfeltétel.
- *Join*: pszeudoállapot, ami több beérkező átmenetet kapcsol össze eggyé. Az átmenetek ortogonális régiókból kell, hogy induljanak és nem lehet rajtuk trigger vagy őrfeltétel. A join szinkronizációs funkcionalitással bír: addig nem lehet tovább lépni belőle amíg minden beérkező átmenet végre nem hajtódott.

  Fork/Join alkalmazásával ki tudjuk kényszeríteni, hogy részrendszereink elérjenek egy adott állapotot, mielőtt a végrehajtás folytatódhatna. Jelölésük a 2.3 ábrán látható.

Rendszereink leírásakor előfordulhatnak ismétlődő részek, amiket célszerű egyszer leírni és újra felhasználni.

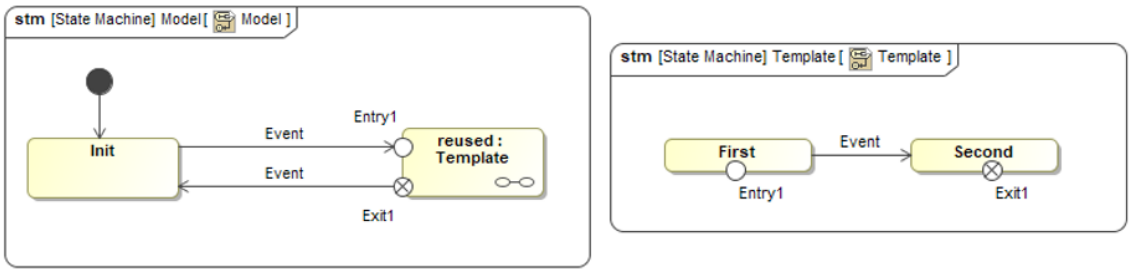
- *Submachine state*: Egy olyan állapot, ami egy állapottérképre hivatkozik, ez lehetővé teszi, hogy egy állapottérkép többszöri felhasználását, akár más-más kontextusban



2.3. ábra. Példa: fork-join

- *Belépési pont(Entry Point)*: egy pszeudoállapot, ami egy állapotgép vagy egy kompozit állapot belépési pontját reprezentálja, célja egységbe zárni az állapotot vagy az állapotgépet. Továbbá léteznie kell egy állapotátmenetnek közte és egy az állapot vagy állapottérkép fő régiója között. Jele kis fehér kör.
- *Kilépési pont(Exit point)*: mint a belépési pont, de ez kilépési pontot reprezentál, jele kis fehér kör áthúzással.
- *Kapcsolódási pont referencia(Connection Point Reference)*: *Submachine Stateben* definiált be és kilépési pontokra tudunk vele hivatkozni, ez lehetővé teszi, hogy a *Submachine Stateben* leírt belső állapotokhoz is felvehessünk állapotátmeneteket.

Egy állapottérképen a belépési és kilépési ponttal élek lehetséges kezdő illetve végpontjait tudjuk definiálni. Újrafelhasználásnál a behivatkozott állapottérképen ezekre referálhatunk Kapcsolódási Pont Referenciákkal. Az ezekbe húzott állapotátmenetek úgy tekintendők mintha kezdő vagy végpontjuk az az állapot lenne amihez a belépési vagy kilépési rendelve van. A *Submachine State* szintaktikáját és használatát a 2.4 ábra szemlélteti.



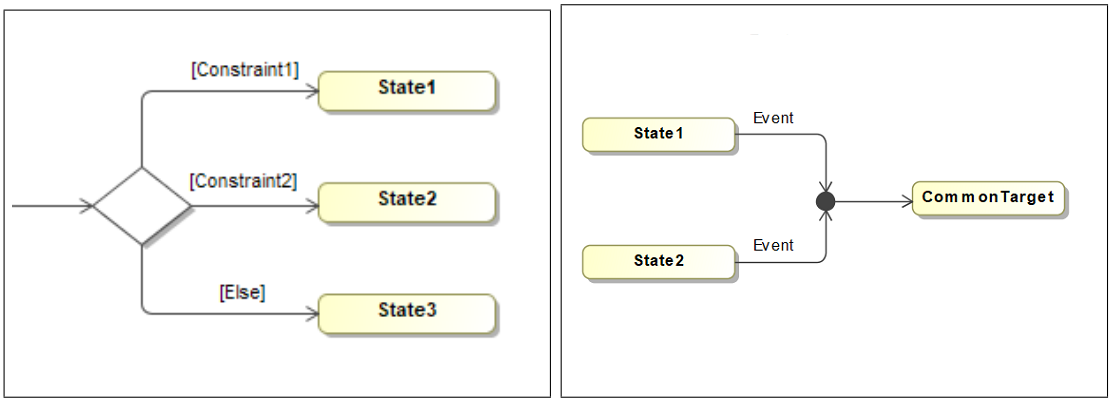
2.4. ábra. Állapottérkép újrafelhasználása Submachine State segítségével

Logikailag állapotátmenetek is összetartozhatnak, ezért célszerű bizonyos esetekben egyesíteni, vagy szétbontani őket több alternatív átmenetre.

- *Csomópont (Junction)*: pszeudoállapot, több állapotátmenet összekapcsolása és egyként kezelése, például ha a cél állapotuk ugyan az és logikailag összetartoznak vagy egy beérkező átmenet szétbontása több átmenetre. Ilyenkor lehetőség van őrfeltételt rakni az átmenetekre, ezeknek a kiértékelése viszont még azelőtt történik, hogy bármelyik átmenet végrehajtásra kerülne, ezért egy ilyen ágat szokás statikus feltételes ágnak nevezni.
- *Döntés (Choice)*: hasonló mint a csomópont, viszont az őrfeltételek az elágazásba való belépéskor értékelődnek ki dinamikusan. Ezt jellemzően alternatív útvonalak

megadására használjuk, hasonló mint a programozási nyelvekben *"if than else"* elágazás. Fontos megjegyezni, hogy az elágazás és a csomópont esetében is lehetséges, hogy több átmenet is tüzelhetne, ilyenkor az érvényre jutó átmenet kiválasztása nem determinisztikus módon történik ezért elkerülendő, például *else* őrfeltétel alkalmazásával.

A csomópont szintaxisa egy a kezdő állapotnál kisebb színezett kör, a döntésé pedig egy rombusz. (2.5 ábra).



2.5. ábra. Döntés és Csomópont szintakitkája

2.3. MagicDraw

A MagicDraw a No Magic [4] nevű cég által fejlesztett modellező eszköz, amivel a modellek előállításán kívül lehetőségünk van ezeket szimulálni, validálni, vagy akár kóddá alakítani. Az eszköz első sorban UML modelleket lehet készíteni, de plug-innal lehetőségünk van SysML [3] modelleket is létrehozni.

A SysML egy általános-célú modellezési nyelv ami az UML egy részének kiragadásával és annak kibővítésével keletkezett. SysMLel struktúrát és viselkedést lehet leírni magas szinten. Alapeleme a Blokk, ami UML-ben a Classnak felel meg. A Blokk egy absztrakt egység ami bárminek megfeleltethető, így a modellezendő rendszer is általában egy Blokként jelenik meg. A Blokkok definícióját és tartalmazási hierarchiáját Block Definition Diagramokkal írhatjuk le. Egy másik megemlítendő diagramfajta az Internal Block Diagram, amivel a Blokkok és portjaik között tudunk kapcsolatokat definiálni.

Viselkedést Állapottérképekkel és Activity Diagrammokkal szokás leírni SysMLben. Utóbbival munka- és adatfolyamokat, ahol a folyamat lépései activityk és actionök. Állapottérképekkel reaktív rendszereket szokás leírni, a rendszer eseményekre reagál, ezek határozzák meg a viselkedését, szemben az Activity Diagrammokkal, amivel adott bemenetből valamilyen kimenet előállításának folyamatát írjuk le.

2.3.1. Állapottérképek SysMLben

Az állapottérképeket SysML modellek részeiként, más modell elemekhez vannak viselkedésként hozzárendelve, a triggereket aktiváló események a modell strukturális leírásában vannak definiálva. Mivel a dolgozat kizárólag állapottérképekkel foglalkozik ezért implicit feltételezzük, hogy minden állapottérképhez tartozik egy Blokk, egy Blokk viselkedését pontosan egy állapottérkép írja le. A Blokkon definiálva van egy Port ami az események fogadásáért felelős. Minden állapotátmenet ettől a Porttól várja az eseményeket.

2.3.2. Plug-in fejlesztése MagicDrawhoz

A MagicDraw lehetővé teszi, hogy harmadik fél plusz funkciókat adhasson az eszközhöz plug-inok formájában. A MagicDraw Java nyelven íródott, plug-int is ezen a nyelven van lehetőség fejleszteni, ehhez egy Api-t kapunk amit Open Api-nak [5] hívnak, ez teszi lehetővé, a modell elemek kóddal történő manipulációját, és a grafikus interfész kiegészítését, saját funkcionalitással.

A MagicDraw indulásakor bejárja a plug-in könyvtárat plug-in leíró fájlokat tartalmazó könyvtárak után. Ezek írják le melyik Java osztály reprezentálja a plug-int, ennek le kell öröklődnie a *com.nomagic.magicdraw.plugins.Plugin* osztályból. A MagicDraw plugins managere meghívja ennek az osztálynak az init metódusát, amiben GUI elemeket tudunk regisztrálni, vagy egyéb funkcionalitást hozzáadni az eszközhöz.

```
public class MyPlugin extends com.nomagic.magicdraw.plugins.Plugin {

   public void init(){
      //plugin belépési pontja
   }

   public boolean close(){
      //plugin leáll
   }

   public boolean isSupported(){
      //feltételek teljesülése a plug-in betöltéséhez
      return true;
   }

}
```

A SysML elemei sztereotipizált UML elemek, a SysML plug-in SysML szintaktikát használ, de a létrehozott elemek a MagicDraw UML meta-modellje szerint lesznek példányosítva megfelelően sztereotipizálva, ezért kód szinten is eszerint érhetőek el.

A modell elemek ugyan saját MagicDraws implementációval rendelkeznek, de EMF [4]-es interfaceket is realizálnak ezért lehetőségünk van EMF Apijának használatára a plugin fejlesztése során.

2.4. Eclipse Modelling Framework

Az Eclipse Modelling Framework (EMF) Eclipse plug-inok egy halmaza, amik lehetővé teszik adatmodellek létrehozását és ebből kód generálását. Az EMF kétféle modellt különböztet meg a meta-modellt és példány modellt. A példány modell struktúráját a meta-modell írja le. A modell egy konkrét példánya a meta-modellnek.

Egy EMF meta-modell `ecore` és egy `genmodel` leíró fájlból áll. Előbbi magát a modellt utóbbi pedig a modell generálására vonatkozó információkat tartalmaz.

Az *EMF persistence framework* lehetővé teszi modellek perzisztens tárolását XMI és XML alapokon. A fájlrendszerben található modelleket *Resource*-ok reprezentálnak melyeket egy URI séma azonosít. A *Resource*-ok *ResourceSet*-ekben találhatók. A modellek közötti hivatkozások feloldásához a modelleket tartalmazó *Resource*-oknak azonos *ResourceSet*-ekben kell lenniük.

[4]Eclipse Modelling Framework https://www.eclipse.org/modeling/emf/

Eclipses környezetekben az EMF-nek nagy jelentősége van. Xtext[5]-el együtt alkalmazva saját DSL-eket lehet fejleszteni saját nyelvtannal és absztrakt szintaxissal.

2.5. Viatra

Az Eclipse Viatra Framework[6] egy modell transzformációs eszköz, ami lehetővé teszi modellek hatékony eseményvezérelt transzformációját és lekérdezését. A modell lekérdezésekhez egy külön nyelvet Viatra Query Language(VQL)-t használ. VQL lekérdezésekből Java osztályok generálódnak, melyek segítségével a lekérdezések és transzformációk futtathatók.

Viatra használatában MagicDraw plugin fejlesztése során is van lehetőség. A Viatra for MagicDraw egy plug-in ami lehetővé teszi a Viatra Api használatát más plug-inokban.

2.6. Formális verifikáció

A modell alapú fejlesztés egyik nagy előnye, hogy már tervezési fázisban tudjuk vizsgálni rendszerünk egyes aspektusait. A rendszer helyes működésének ellenőrzését verifikációnak hívják. Ez történhet tesztekkel, szimulációval, vagy formális módszerekkel.

A formális módszerek előnye, hogy a rendszer helyességéről matematikailag precíz bizonyítást adnak, nem szükséges az elvárt kimenetek meghatározása, csak megkötések megfogalmazása. Továbbá teljesek ezért nem kell lefedettséggel foglalkozni mint tesztelésnél. Megkötés sérülésekor, vissza lehet követni, azt az utat ami a megszorítás megsértéséhez vezetett (*execusion trace*).

Hátránya, hogy nehezen skálázható, erőforrás igényes és egy összetett rendszert formális módszerekre való visszavezetése gyakran nehézkes.

Állapottérképek formális verifikációjára már léteznek megoldások. A Gamma keretrendszer például képes állapottérképek verifikációjának elvégzésére. Ehhez az Uppaal [1] [2] nevű eszközt használja fel.

2.7. Gamma Framework

A Gamma Statechart Composition Framework[6] komponens alapú reaktív rendszerek tervezésére, validálására, verifikálására és kód generálásra lett létrehozva. Az eszköz egyik alap komponensei az állapottérlépek amiknek a leírásához egy saját nyelvet Gamma Statechart Languaget (2.6 ábra) definiál, ami lehetővé teszi külső modellező eszközök integrálását is. Az első integrált eszköz a Yakindu Statechart Tools [7], aminek a modelljeit Gamma képes a saját maga által definiált modellekké (2.7 ábra) transzformálni és feldolgozni.

Az eszköz egy másik komponensei az ún. Kompozit komponensek, amiket Gamma Composition Languagel lehet definiálni. Ezek írják le a rendszer felépítését az állapottérképek portjait, az azok által realizált interfészeket és a közöttük definiált kapcsolatokat.

A keretrendszer még két saját nyelvet definiál. Az egyik a Gamma Constraint Language amiven constrainteket lehet definiálni, ami egy általános módja típus definíciók, változók, függvények deklarálásához és kifejezések specifikálásához.

A másik nyelv pedig Gamma Interface Language, ami interfészek definiálását teszi lehetővé, amik a kapcsolódó komponensek egymás felé nyújtanak. Az interfész határozza meg, hogy milyen események fogadhatóak, vagy küldhetőek. Ezek az interfészekben deklarálandók és irányuk lehet, *IN, OUT, INOUT.*

[5]Xtext: https://www.eclipse.org/Xtext/

[6]Viatra: https://projects.eclipse.org/projects/modeling.viatra

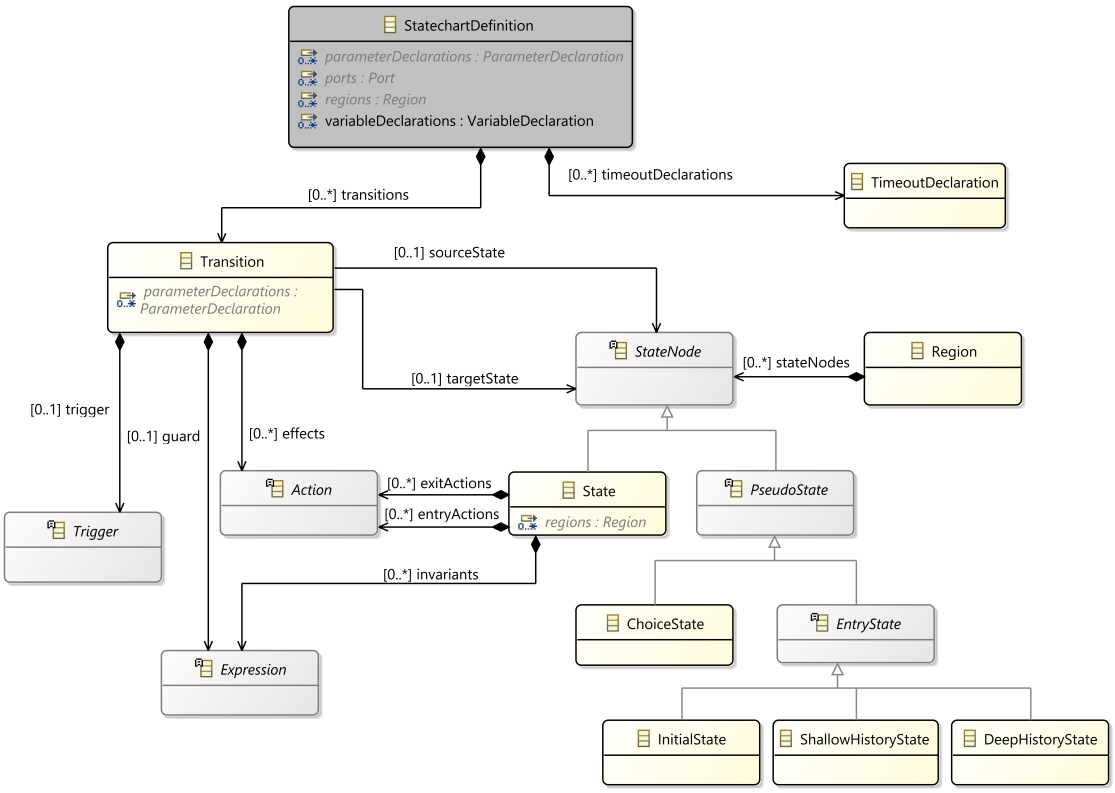
```
package Exapmle

statechart MonitorStatechart [] {

transition from Red to Blue
transition from Entry0 to Red

   region main_region {
      initial Entry0
      state Red
      state Blue
   }
}
```

2.6. ábra. Gamma Statechart konkrét szöveges szintaxisa



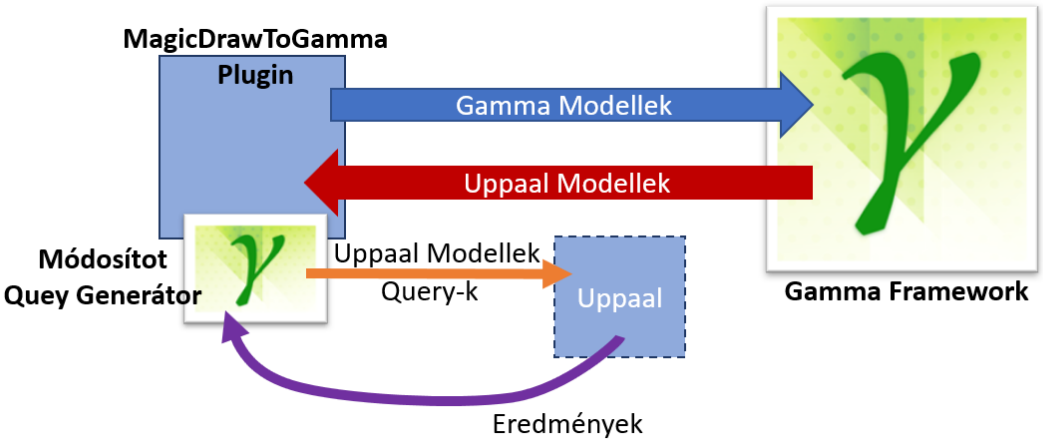
2.7. ábra. Gamma állapottérkép absztrakt szintaxisa

3. fejezet

MagicDrawToGamma plugin

3.1. Koncepció

Állapottérképek formális verifikációjának támogatása MagicDraw-ban, egy plug-in fejlesztésével lett megvalósítva. A plug-in függ a Viatra For MagicDraw-tól, ami lehetővé teszi modellek transzformációját Viatra segítségével. A plug-in legfontosabb funkciója MagicDraw modellek Gamma modellekké való transzformációja. A letranszformált Gamma nyelvű modelleket az keretrendszer kezelni tudja, a verifikáció elvégzéséhez az eszköznek csak egyes részei szükségesek. A megoldást a 3.1 ábra szemlélteti. A felhasználónak lehetősége van megkötések megfogalmazására a plugin-al és elvégezni a verifikációt és megtekinteni az eredményt azaz, hogy teljesülnek-e a megkötések vagy sem.



3.1. ábra. Koncepció

3.2. Fejlesztőkörnyezet

A fejlesztés megkezdéséhez szükséges volt összeállítani egy olyan fejlesztőkörnyezetet amivel hatékonyan lehet plug-int fejleszteni. MagicDraw biztosít egy ún. *skeletont* Eclipsehez és IntelliJhez is plug-in fejlesztéséhez, de a fejlesztés nem ezek segítségével hanem az IncQueryLabs által készített skeleton felhasználásával valósult meg. Ennek oka, hogy a hivatalos *skeletonok* nem vagy csak részben működtek, a mögöttes infrastruktúra megismerése és javítása pedig túl hosszadalmas és a feladat szempontjából irreleváns lett volna.

A skeleton egy Eclipse project, viszont Gradlet használ a projekt fordításához és a dependenciák kezeléséhez. Ez sokszor inkonzisztenciákhoz vezetett az egyik legnagyobb

probléma a Viatra Querik generálása csak Eclipsel lehet generálni. A kódbázis egy része nem Javában hanem Xtendben íródott, a Viatra modell transzformációk implementálása ezzel a nyelvvel egyszerűbb. Azok az osztályok melyeknél nem volt indokolt, jellemzően a MagicDraw felhasználói felületeinél, azok Java 8-ban lettek implementálva.

A dolgozat elkészítése idején a MagicDraw 19-es verziója is elérhető volt a plug-in azonban még nem ehhez, hanem a 18.5-ös verziójához készült. A kódbázis azonban kompatibilis lehet még az újabb verziókkal is, amennyiben az állapottérképeket érintő meta-modellek nem változnak.

Gamma(2.0) a verifikációt Uppaal segítségével végzi el. Ehhez előállít egy leírást a rendszerről és egy queryt ami a rendszerrel szemben támasztott követelményeket írja le. Utóbbi megírásához biztosít egy UI elemet amivel a felhasználó az Uppaal ismerete nélkül is képes a követelmények definiálására. Ez a funkció teljesen át lett emelve Gammából módosított implementációval a MagicDrawToGammába. A Gamma az Uppaalra az operációs rendszeren keresztül hív át, ezért az Uppaalt külön kell telepíteni és konfigurálni.

3.3. MagicDraw - Gamma transzformáció

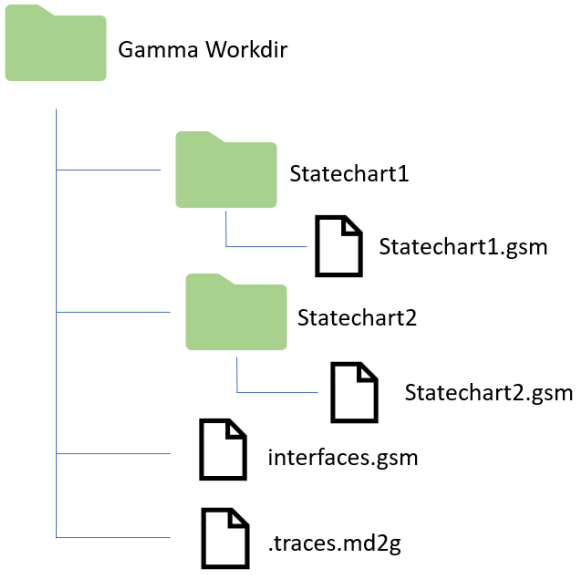
A MagicDraw - Gamma transzfromációt egy menü elemmel lehet elindítani. Ehhez szükséges, hogy a projekthez hozzá legyen rendelve egy külső könyvtár (Gamma Workspace) ami a leképzett és perzisztensen eltárolandó modellek helyét jelöli. A hozzárendelt könyvtár abszolút elérését String formában a projekt tárolja, emiatt sem a projekt nem migrálható, sem pedig maga a Gamma Workspace. Továbbá a könyvtár karbantartása ebben a verzióban még a felhasználó felelőssége, ugyanis a plug-in nem töröl a könyvtárból csak hozzáad és módosít, ennek akkor van jelentősége, ha leképzés után egy állapottérkép el lett távolítva a MagicDraw modellből és utána újra le lett képezve. Ebben az esetben a régebben leképzett Gamma állapottérképek is megmaradnak.

3.3.1. Gyökér elemek létrehozása:

A plugin a transzformáió elején létrehoz egy *ResourceSet*-et, amibe készít egy *Resource*-t a *Gamma Workpace* gyökerébe .s.md2g néven, továbbá egy Resource-t `interfaces.gms` néven. Az előbbi Resource egy segédstruktúrát tartalmaz ami a leképzett elemek visszakereshetőségéül szolgál (ld. 3.3.8. alszakasz), utóbbi pedig a modellben definiált interfaceket tárolja.

Az eszköz ezután Viatra Queryk segítségével megkeresi az állapotgépeket a MagicDrawban és végig iterál rajtuk. Minden állapottérképen kigyűjti az éleken használt Signal Eventeket és létrehoz a Signaléval azonos néven egy Eventet, a Gamma metamodelljének megfelelően. A létrehozott eventek egy Interfacen kerülnek definiálásra aminek a neve megegyezik az állapottérképével. Az interface ezután belekerül az interfaceket tároló *Resourceba*. Az eventek iránya INOUT ez lehetővé teszi azt is, hogy az állapotgép magának küldjön eseményt.

A leképzett állapottérképek külön Resourceokba kerülnek amik a Gamma Workspaceben egy külön az állapottérkép nevével megegyező könyvtárba kerülnek, ugyan ezen a néven .gsm kiterjesztéssel (3.2-es ábra). (A Resource gyökere nem egy StatechartDefinition, hanem egy Package, ennek a neve szintén megegyezik az állapottérkép nevével)



3.2. ábra. Létrehozott fájlstruktúra

3.3.2. Alap struktúra kialakítása:

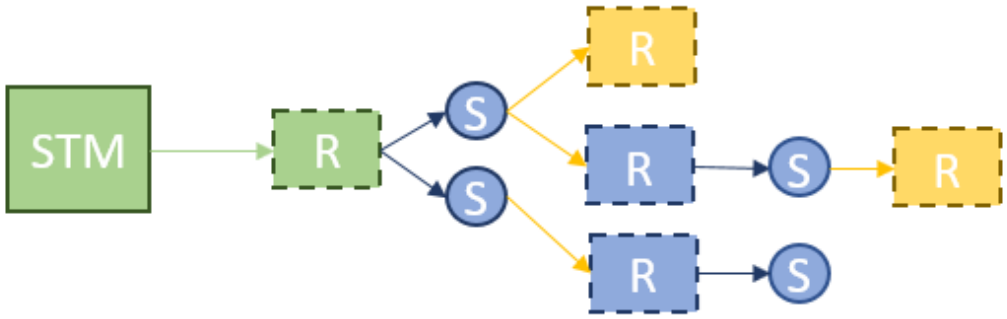
Az előző lépések során azok az elemek állnak elő, melyek a modellek gyökér elemeiként fognak szolgálni. A következő lépés létrehozni az állapottérképek egy belső, alap struktúráját, amit az állapotok és állapotátmenetek határoznak meg. A Viatra for MagicDraw projekt megnyitásakor készít egy ViatraQueryEngine-t aminek a Scope-ja a megnyitott modellre terjed ki. A transzformációs szabályok regisztrálása ezen az engine-en történik, létrejön azonban még egy aminek a Scope-ja magába foglalja az első lépésben lérehozott *Resource*okat és a MagicDraw modellt is. A készülő modellben történő keresések, és a visszakövetések ezzel az engine-el történnek. Alap struktúra kialakításának lépcsői:

1. Fő régiók leképzése (olyan régió aminek a szülője állapotgép).
2. Régiókban található állapotok leképzése.
3. Állapotokban található régiók leképzése.

A második lépésben a Trace Modell alapján megkeressük a már leképzett régiót a Gamma Modellben és beletesszük az újonnan létrehozott és a MagicDraw modellnek megfelelően elnevezett állapotot, ha a régió még nincs leképezve akkor létrejön, és abba kerül bele az állapotot. Ez a lépés olyan részgráfokat is eredményezhet amikbe nem vezet út a gyökér elemekből. Ennek kiküszöbölése a harmadik. lépés ami, ugyan ezt a műveletet hajtja végre csak a másik irányból, tehát az állapotok párjait keressük meg, amikbe régiókat helyezünk el. Ezek a régiók már létezhetnek ilyenkor nem új régió jön létre hanem a már meglévő kerül az állapotba. A működés során a MagicDraw modell jól formáltsága kihasznált és elvárt, továbbá az is ki van használva, hogy régió csak *State*ben és Állapotgépben lehet. A régiókat tartalmazó állapotok kompozit és ortogonális állapotnak tekintendők.

A tartalmazási gráf összefüggővé válását a 3.3 ábra szemlélteti.

Az irányított nyilak tartalmazást jelölnek, a bekeretezett téglalap StatechartDefinition, a szagatott vonallal körbevett Region és a körök Statek.



3.3. ábra. Állapotok és régiók leképzésének menete, 1. lépcső: zöld, 2. lépcső: kék, 3. lépcső: sárga

3.3.3. Pszeudoállapotok átalakítása:

Az állapotátmenetek leképzése előtt a pszeudoállapotok kerülnek leképzésre. Ezen a ponton már létezik minden régió amit tartalmazhatja őket. A leképzés legtöbb esetben támogatott viszont, egyes elemek tartalmazása esetén nem lehet a verifikációt végrehajtani. Az elemeket és párjaikat a 3.1 táblázat mutatja.

MagicDraw	Gamma	verifikálható
InitialState	InitialState	igen
Chioce	Choice	igen
Junction	Merge	nem
Fork	Fork	nem
Join	Join	nem
TerminalState	nincs	-
Conn. PointReference	nincs	-
EntryPoint	nincs	-
ExitPoint	nincs	-

3.1. táblázat. Pszeudoállapotok párosítása.

3.3.4. Állapotátmenetek leképzése:

A következő lépés az állapotátmenetek átalakítása. Ezen a ponton már az összes olyan elem leképzésre került, amely az állapotátmenetek kezdő, vagy végpontjaként szolgálhat. Egy MagicDraw modellben az állapotátmenetek régiók tartalmazzák, szemben Gammával, ahol a StatechartDefinition közvetlen gyerekei. A tartalmazó-tartalmazott, Statemahcine - Tranisiton párok megkeresése a következő patternekkel történik.

```
pattern RegionsInRegion(container: Region, region: Region){
    Region.subvertex(container, vertex);
    State.region(vertex, region);
}
pattern RegionsInStatemachine(stateMachine: StateMachine, subregion: Region){
    find MainRegions(stateMachine, subregion);
} or {
    find RegionsInRegion+(region, subregion);
    StateMachine.region(stateMachine, region);
```

```
}
pattern TranisitonsInStateMachine(stateMachine: StateMachine, transition:
    Transition){
    find RegionsInStatemachine(stateMachine, region);
    Region.transition(region, transition);
}
```

A StateMachine Gamma modellbeli párját a *Trace modell* segítségével lehet megtalálni és hozzáadni a megfelelő állapotátmenetet.

Az átmenetek leképzése után már elérhetőséget lehet is vizsgálni.

Változók leképzése Változókat MagicDraw-ben attribútumként van lehetősége definiálni. FOLYT

3.3.5. Triggerek leképzése:

A MagicDrawban definiálható *Triggerek* pontosabban az őket kiváltó események közül jelenleg kettő támogatott. Egyik a *SignalEvent* a másik pedig *Time Event*. Előbbi *Event-Triggerre* képződik le. A felhasználónak a *SignalEvent* forrásával most még nem kell foglalkoznia, hiszen ezekhez automatikusan generálódik egy *Interface* és egy *Port*.

A *Time Eventek* MagicDraw-ban két féle típusúak lehetnek: relatív és abszolút. Utóbbit a plugin-in még nem támogatja. A relatív típusú *Time Eventek* a Gamma *Timeout* mechanizmusának feleltethető meg. Ez három részből áll: *StatechartDefinition*-ön definiált *TimeoutDeclaration*, akció ami ennek beállítja az értékét (ami lehet szekundumban, vagy milliszekundumban mért) és maga a *Trigger* ami hivatkozik a deklarációra. A *Timeout-Declaration* és az érték beállítása implicit történik a felhasználónak az időt kell megadnia trigger felvételekor a MagicDraw modellben.

3.3.6. Őrfeltételek leképzése:

őrfeltételek definiálása MagicDraw állapottérképeken Opaque Expressionökkel[1] történik, ezért ezt le kell fordítani és modell alapú leírássá konvertálni. Kifejezéseket Gammában a Constraint modellel lehet leírni. Ehhez tartozik egy nyelvtan is ami Xtext segítségével képes a Gamma Constraint nyelvből EMF alapú Constraint Modell példánymodellt fordítani. A MagicDrawToGamma ezen verziója az őrfeltételek leképzéséhez egy saját egyszerűsített implementációt használ. Ennek vannak megkötései is: kifejezések nem ágyazhatók, vagy láncolhatók, változókra és paraméterekre lehet hivatkozni, de csak a gazda StatechartDefinitionban ezen felül csak logikai és Long értékeket lehet használni.

3.3.7. Akciók leképzése:

A MagicDraw akciói viselkedések (*Behavior*) lehetnek, ezek leképzését a MagicDrawToGamma nem támogatja, kivéve a *Functional Behavior* viselkedést, amit egy *Opaque Expression* definiál. Ebben kétféle akciót lehet végrehajtani: értékadást és *Signal* küldést. Ezek szintaktikája a következő:

```
    set variableName := 1;
    raise Event
```

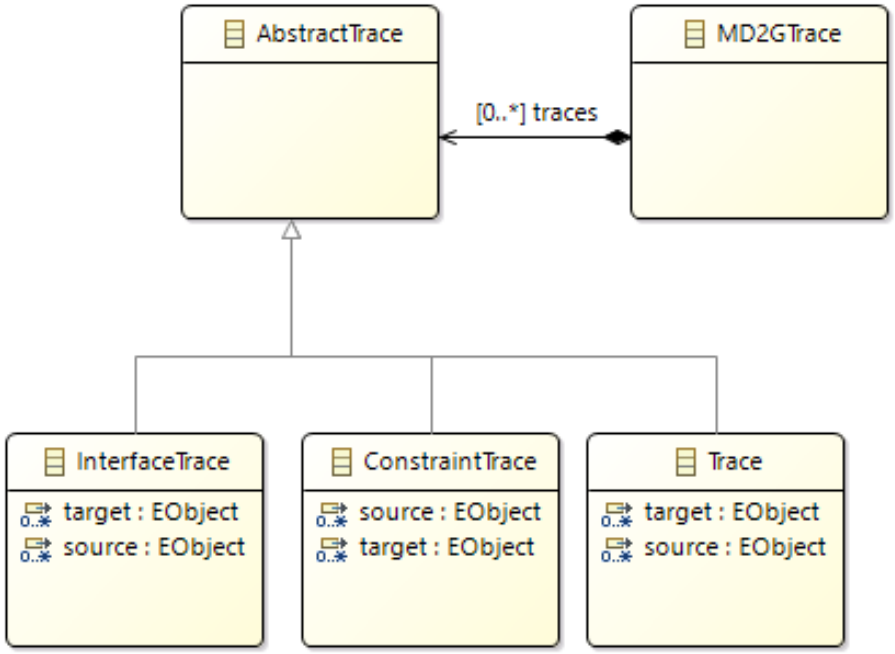
FOLYT

[1] Szöveges nyelvel leírt kifejezés

3.3.8. Trace Modell

A leképzések végrehajtása során fontos követni, hogy mely elemek képződtek le és mely elemekké. Erre a célra a MagicDrawToGamma bevezet egy Trace modell[2] nevű segédstruktúrát, ami lehetővé teszi a megfeleltetések visszakereshetőségét Viatrával, továbbá a modell sorosítása és háttértáron való tárolása is megoldott. A modell EMF-ben definiálva és háromféle *Trace*-t különbötet meg. Az Gamma állapottérképek alap elemeit `Trace`-ek az interfészek elemeit `InterfaceTrace`-k és a *Contraint* modell elemeit `ConstraintTracek` kötik össze MagicDraw párjukkal, amiből le lettek képezve.

A három `Trace` típust egy `AbstractTrace` osztály fogja össze, és `MD2GTrace`-ben tárolhatók. (3.4 ábra)



3.4. ábra. Trace modell EMF definíciója

3.4. A Verifikáció végrehajtása

A verifikáció végrehajtásához két dologra van szükség. A modellnek egy leírására amit az Uppaal képes értelmezni és előállítani egy időzített automatát belőle és az Uppaal Query nyelvén megfogalmazott feltételekre. A MagicDrawToGamma kétféle lehetőséget biztosít a verifikáció végrehajtására.

1. Uppaal közvetlen használata
2. Uppaal Query Generator

Első lehetőség időzített automaták leírásának előállítása amit a felhasználó megnyithat *UPPAAL*-al. Ennek végrehajtása két részből áll. Először a Gamma modellt át kell alakítani, ez a `StatechartToUppaalTransformer` osztályon keresztül történik, ami a Gamma része.

[2]Nem összekeverendő a a 2.6 fejezetben említett *Execusion Tracel*

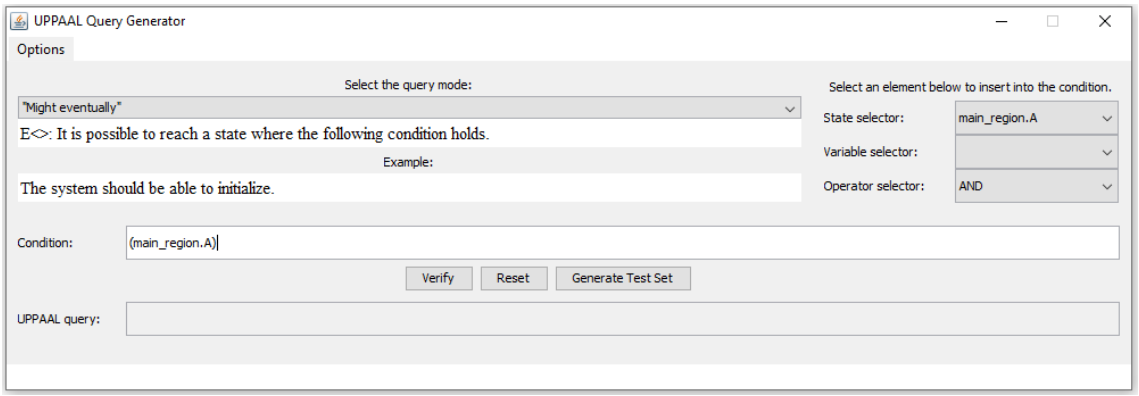
```
//initialize with a Gamma Package
StatechartToUppaalTransformer transformer = new StatechartToUppaalTransformer(p);
SimpleEntry<NTA, G2UTrace> entry = transformer.execute();
```

Ennek a kimenete két EMF alapú modell, egy *trace modell* (`GU2Trace`) és egy időzített autómata (`NTA`). Az **időzített autómata**[3] modelljét ezután egy formális leírásként kell szerializálni amit az UPPAAL képes beolvasni. Ez a Gamma `UppaalModelSerializer` osztálya állítja elő.

```
//NTA, parentFolder, fileName
UppaalModelSerializer.saveToXML(entry.getKey(), selected.getPath() , name
    +".xml");
```

A kimenetek a *Gamma Workspace-ben* az állapottérkép könyvtárába kerülnek, ahonnan a felhasználó megnyithatja őket *UPPAAL*-ban.

A második lehetőség az *Uppaal Query Genertor* (3.5 ábra) használata. Ez egy segédablak amivel a felhasználó egy grafikus felhaszálói interfészen tud *UPPAAL Query*-ket definiálni, így a felhasználónak nem szükséges elsajátítania az *UPPAAL Query*-k szintaxisát.



3.5. ábra. UPPAAL Query Generator

A Query Generátor implementációja a Gammából át lett emelve a MagicDraw pluginba és az implementáció módosítva lett, hogy ebben a környezetben is tudjon működni. A Gamma egyik funkciója, hogy a formális verifikáció során esetleg keletkező ellenpéldákból képes teszteseteket és szimulációt generálni Yakinduhoz, ez a MagicDrawToGamma plugin jelenlegi verziójában le van tiltva.

3.5. Plulgin működés közben

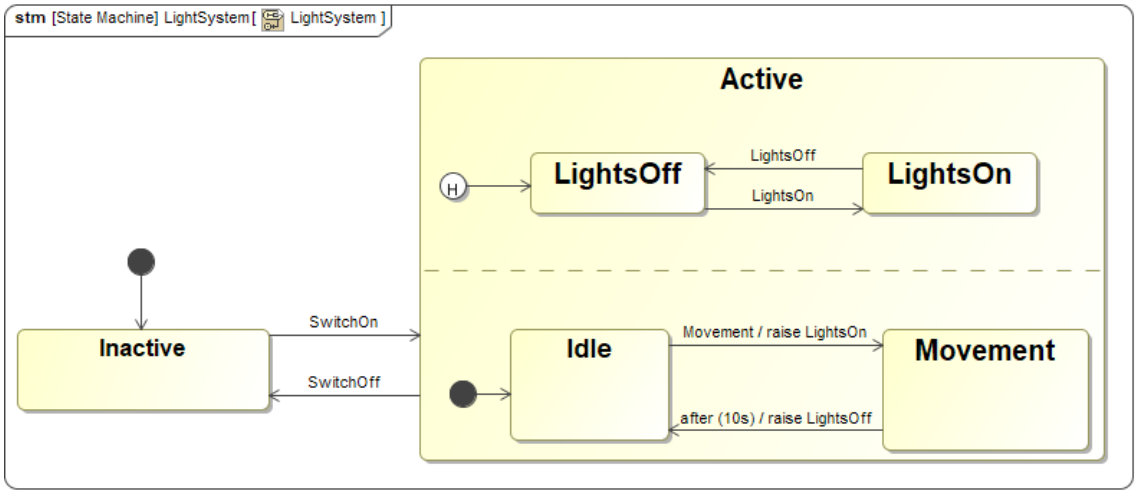
Az előző alfejezetek a MagicDrawToGamma főbb funkcióinak ismertetéséről és a funkciók megvalósítását mutatták be. Ebben az alfejezetben a tényleges működéséről lesz szó egy példán keresztül.

3.5.1. Szemléltető példa

Példa specifikációja: egy kerti világítórendszer mozgás érzékelővel van felszerelve, ha a rendszer aktív és mozgást érzékel tíz másodpercre bekapcsolja a lámpákat majd ezután kikapcsolja azokat. A rendszer inaktív állapotában nem ég egyetlen lámpa sem.

[3] a formális verifikáció az időzített autómaták formalizmusán törénik

Rendszer állapot alapú definíciója: A rendszer két jól megkülönböztethető állapotból áll `Acitve` és `Inactive`. Az `Active` állapot felbontható két belső állapotra, hogy égnek-e vagy sem a lámpák: `LightsOn`, `LightsOff`. Ezeken kívül bevezethető még két állapot, hogy érzékel-e épp mozgást a rendszer vagy sem: `Idle`, `Movement`. Ezek alapján a rendszer egy lehetséges leírása lehet:



3.6. ábra. Lehetséges leírás

Megjegyzés Az egyszerűség kedvéért tekintsük, úgy hogy a folytonos mozgás során bekövetkezett `LightsOn` - `LigtsOff` - `LightsOn` váltások olyan gyorsan végbe mennek, hogy fizikailag nem képesek a lámpák lekapcsolni, hogy a villodzással ne kelljen foglalkozni

4. fejezet

Értékelés

4.1. Alternatíva - kódgenerálás

A Gamma saját nyelvtanokkal rendelkezik, melyek Xtext segítségével vannak implementálva. Ez a technológia lehetővé teszi, hogy saját szintaxis alapján, lehessen kódot írni és EMF példánymodellt generálni a leírásból.

Ezt a mechanikát ki lehet használni a modell transzformáció alternatívájaként: a MagicDraw modellből Gamma Statechart Language szintaxisának megfelelő kódot lehetne generálni és ezt Xtext segítségével leparseolni.

A visszakövethetőség is megoldható, ehhez a nyelvtant annotációkkal kéne kibővíteni, amik jelölnék az eredeti elemeket.

Ennek előnye, hogy a kimeneteket utána tovább lehetne importálni Eclipse-be és abban folytatni a fejlesztést.

Hátránya viszont, hogy a skálázhatóságot sokkal nehezebb megoldani, és kevésbé flexibilis mint a transzformáció.

4.2. Kifejezések kifejező ereje

A MagicDrawToGamma által támogatott Guard és Action definíciók korlátozottabban használhatók a jelenlegi implementációban, mint amit egyébként a Gamma támogatni tudna, ezért célszerű lenne ezek bővítése, hogy ugyan azzal a kifejező erővel bírjanak mint a Gammában megfogalmazható párjaik.

A jelenlegi implementáció egy saját készítésű String parser ami Gamma Expressionöket állít elő. Ennek a bővítése igen nehézkes, viszont viszonylagos egyszerűsége miatt nem jár túl nagy overhaedel és nem ad hozzá még több dependenciát a projekthez.

Alternatív megvalósításként potenciálisan lehetne használni Xtextet és a már létező nyelvtant, az ez irányba tett kísérletek, azonban a megoldás túlzott komplexitását látszanak igazolni. A komplikációk fő oka az volt, hogy a nyelvtant alkotó egy szabály szerint parse-olt kódrészletek esetében a referenciák nem oldódtak fel.

5. fejezet

Továbbfejlesztési lehetőségek

5.1. IBD - Composition Language

A Gamma Statechart Composition Framework egyik legfontosabb funkciója, hogy lehetővé teszi állapottérképekből, mint komponensekből egy komplett rendszer leírását. Ilyesfajta leírás SysML-ben az Internal Block Diagram(IBD).

Az IBD-k leképzésének támogatásával a felhasználók képessé válnának komplex reaktív rendszereket leírni és verifikálni.

5.2. Szimuláció generálása

Az UPPAAL opcionálisan előállítja azokat az utakat melyek sértik a megkötéseket. A Gamma Framework képes ezekből kódot és Yakindu szimulációt előállítani.

A MagicDraw is rendelkezik egy szimulátorral Cameo Simulation Toolkit[1] néven. A Cameo szimulátor plug-in a No Magic terméke. Segítségével modelleket lehet debugolni, szimulálni és UI prototyping funkcionalítással rendelkezik.

A szimulációkat modell elemekkel is fel lehet konfigurálni Execution Configuration Classok segítségével, ezért potenciálisan modell transzformációkkal elő lehet állítani szimulációt.

5.3. Validation kit

A VIATRA-va inkrementális és reaktív tulajdonsági lehetővé teszik modell transzformációk végrehajtását, ha a modellt változik. Ezt a funkcionalitást kihasználva lehetőséget kapunk, hogy létrehozzuk validációs szabályok VQL-ben leírt halmazát és ezeket futás időben folyamatosan ellenőrizve a MagicDraw API-ján keresztül felannotálhatjuk azokat az elemeket amik nem felelnek meg a ezeknek szabályoknak. Ezeket a MagicDraw megjeleníti a GUI-ján.

Ezek a validációs szabályok lehetnek figyelmeztetések, hogy melyik elemek nem képezhetőek le, vagy leképezhetőek de nem támogatott a verifikációjuk, ezzel növelve a felhasználói élményt, hogy ne a transzformációk végrehajtása alatt értesüljenek a potenciális hibákról.

[1] Cameo simulation toolkit: https://www.nomagic.com/product-addons/magicdraw-addons/cameo-simulation-toolkit

6. fejezet

Összefoglalás

- Leképzés megvalósítása
  - Odafigyeltem a szemantikai különbségekre
  - Meta-modell elemeinek megfeleltetése
- leképzés implementációja
- fejlesztőkörnyezet kialakítása
- elméleti megközelítésből értékeltem a munkám
- lehetővé tettem, hogy az eszközön belül elvégezhető legyen a verifikáció

Ennek eredményeként létrejött egy olyan MagicDraw plug-in amivel lehetőség nyílik az állapottérképek formális verifikációjának végrehajtásába.

6.1. Future work

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Irodalomjegyzék

[1] Johan Bengtsson – Kim Larsen – Fredrik Larsson – Paul Pettersson – Wang Yi: Uppaal—a tool suite for automatic verification of real-time systems. In *International Hybrid Systems Workshop* (konferenciaanyag). 1995, Springer, 232–243. p.

[2] Johan Bengtsson – Kim G Larsen – Fredrik Larsson – Paul Pettersson – Wang Yi – Carsten Weise: New generation of uppaal. In *Proc. Int. Workshop on Software Tools for Technology Transfer (STTT'98)* (konferenciaanyag). 1998, 43–52. p.

[3] Object Management Group: *OMG System Modeling Language*. OMG, 2017. 05. `https://www.omg.org/spec/SysML/About-SysML/`.

[4] NoMagic Inc. URL `https://www.nomagic.com/`.

[5] NoMagic Inc.: *MagicDraw Open API javadoc*. NoMagic Inc., 2017. 04. `http://jdocs.nomagic.com/185/`.

[6] Vince Molnár – Bence Graics – András Vörös – István Majzik – Dániel Varró: The gamma statechart composition framework. 2018, ICSE.

[7] Yakindu Statechart Tools: Yakindu.
URL `https://www.itemis.com/en/yakindu/state-machine/`.

Függelék