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

Monitor szintézis időzített üzenet szekvencia specifikáció alapján

Diplomaterv

Készítette
Bakai István Bálint

Konzulens
Dr. Majzik István

2021. november 27.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Bakai István Bálint*, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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, 2021. november 27.

Bakai István Bálint
hallgató

Kivonat

A monitorozással történő hibadetektálás kiemelt fontosságú egy kritikus rendszer működtetésében és karbantartásában. A monitorozás sok hibát fel tud deríteni, amiket a tesztek nem feltétlenül tudnak kideríteni.

A diplomaterv feladat célja az volt, hogy a korábban elkészített monitor komponens generátort kiegészítsem úgy, hogy időzített üzenet szekvencia specifikáció alapján is képes legyen monitor komponenseket generálni. Ilyen követelményeket egyszerűen specifikálhatunk TPSC (Timed Property Sequence Chart) diagramokkal. A korrábban elkészített Xtext alapú PSC nyelvet kiegészítettem a TPSC tulajdonságaival.

A monitor generálás következő lépése, hogy a TPSC diagramokból időzített automatákat generálunk (Timed Automaton). A TA fogja megadni, hogy a megfigyelt kommunikáció helyes viselkedést jelent-e. A szakdolgozatomban készített automata generátort kibővítettem és most már képes a minta alapú módszert használva TA automatákat generálni a TPSC diagramjainkból. A generátor előállítja az automatát.

A szöveges scenario leírásból generált automata alapján legenerálható a monitor forráskódja. A monitor forráskód generátor előállítja a monitor *Java* implementációját, ami képes egy rendszer monitorozására adott követelmény alapján.

A diplomaterv további feladatai közzé tartozik a TPSC scenario-k vizualizációja, a generált forráskódok tesztelése és a generált monitor komponens illesztése a Gamma keretrendszerrel. A cél az, hogy elosztott komponens alapú rendszerek szimulációja közben monitorozható legyen a TPSC üzenet szekvencia specifikáció teljesülése illetve az ebben rögzített tulajdonságok megsértése. Továbbá a monitorozás működését is demonstrálni kell.

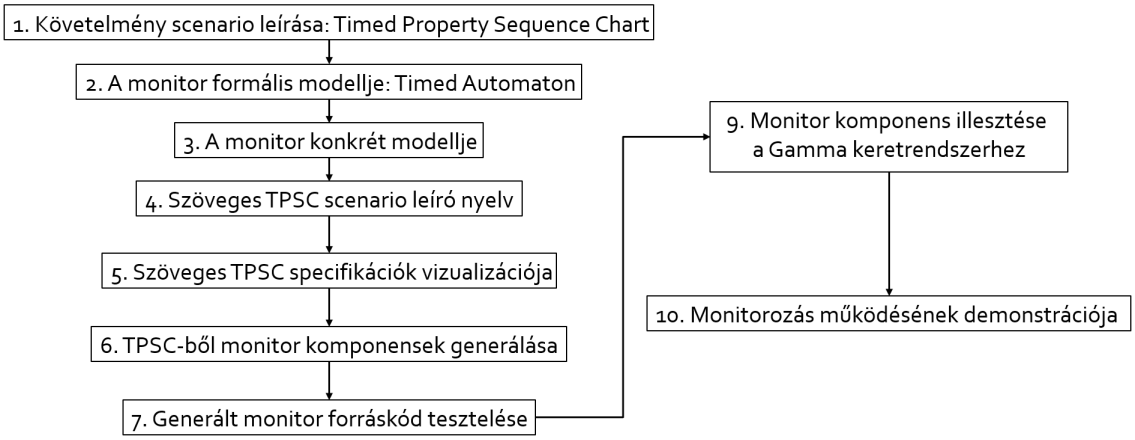
Az időzítési feltételeket tartalmazó üzeneteket tudja értelmezni és feldolgozni a monitor, valamint a teszt keretrendszer első verziója is elkészült.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

1. fejezet

Bevezetés



1.1. ábra. Monitor generátor kibővítése.

Diplomamunkám során az volt a cél, hogy a „Monitor komponensek generálása kontextusfüggő viselkedés ellenőrzése” című szakdolgozatom során elkészített monitor komponens generátort kibővítsem úgy, hogy támogassa időzítési feltételek megadását. A monitor generálás terve látható a 1.1. ábrán. Az Önálló laboratórium keretében az volt a feladatom, hogy a szakdolgozat során definiált szöveges PSC diagram leíró nyelvet kibővitsem a TPSC elemeivel. Ezután az automata generátort kell úgy kibővíteni, hogy a TPSC diagramokból tudjon TA automatákat generálni. Egy monitor forráskód generátor pedig az automata alapján elkészítheti a monitor forráskódját.

A szöveges TPSC scenario leírásához el kell készítenünk a diagram vizualizációját, hogy grafikusan megtekinthessük a definiált szcenáriót. Ehhez felhasználható a ”Modell alapú rendszertervezés” tárgy keretében készített PSC diagram szerkesztő alkalmazás. A következő a generált monitor forráskód tesztelése, majd ezután ezt illeszük a Gamma keretrendszerhez. Ezzel az a célunk, hogy elosztott komponens alapú rendszerek szimulációja közben monitorozható legyen a TPSC üzenet szekvencia specifikáció teljesülése illetve az ebben rögzített tulajdonságok megsértése.

A Diplomatervezés 1 tárgy keretében elkészítettem a monitor forráskód generátort és elkezdtem annak tesztelését. A hátramaradó feladatok közé tartozik a tesztelés befejezése, a diagramok vizualizációja és a monitor komponens illesztése a Gamma keretrendszerhez.

A dolgozatomat a háttérismeretek összefoglalásával kezdem. Először bemutatom a legelterjetebb formalizmusokat, amelyek időfüggő viselkedés specifikálására szolgálnak. Ezután a TPSC formalizmust mutatom be és a felhasznált technológiákat. A dolgozatomat a

kivőbített szöveges TPSC leíró nyelv bemutatásával folytatom. Ezt a TPSC specifikációk vizualizációjáról szoló fejezet követi.

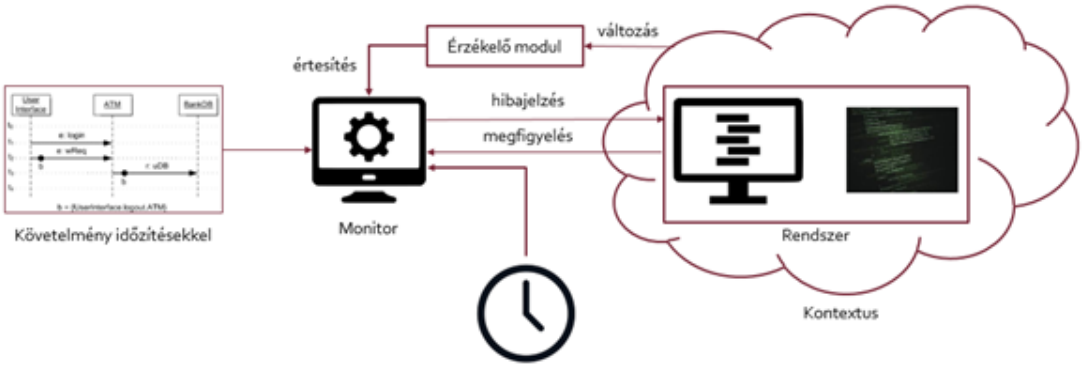
A következő fejezet a monitor forráskód generálásról szól és annak teszteléséről. Ezt követi egy fejezet, amely a monitor komponens illesztését mutatja be a Gamma keretrendszerhez és elosztott komponensúű rendszerek monitorozását. A dolgozatomat egy összefoglalóval zárom.

2. fejezet

Háttérismeretek

2.1. A monitorozás alapjai

Egy monitor feladata az, hogy futási időben egy rendszert megfigyeljen, elemezzen és egy adott követelmény alapján felismerje a rendszer helytelen viselkedését. Ezt a helytelen viselkedést jelzi a rendszernek, de néhány esetben a rendszer működését is befolyásolhatja. Ez egy kritikus rendszernél különösen fontos lehet hiszen, egy ilyen rendszernél elvárt, hogy folyamatosan biztonságosan tudjon működni. Ezen kívül, a monitornak időmérésre is szüksége van, mert a követelmény tartalmazhat időziteseket is. Az 1.1. ábra bemutatja a monitorozás koncepcióját.



2.1. ábra. Kontextusfüggő rendszerek monitorozása időzítési feltételekkel.

A scenario alapú monitorozás során a kommunikáció megfigyelésével szeretnénk felismerni a problémákat a rendszerünkben. A rendszerben lévő objektumok közti interakciókat, üzeneteket fogja megfigyelni a monitor. A követelményt scenario formájában adjuk meg az üzenet szekvenciák specifikálásához. Szekvencia diagramok segítségével egyszerűen megadhatunk ilyeneket. A diagramokat a későbbiekben olyan alakra kell majd hoznunk, hogy abból a monitor létrehozható legyen.

2.2. Időfüggő viselkedés specifikálására alkalmas formalizmusok áttekintése

2.2.1. MSC - Message Sequence Chart

Az egyik legelterjedtebb scenario alapú modellezésre használt vizuális formalizmus a Message Sequence Charts (MSC). A nyelv célja két, vagy több üzeneteket cserélő objektum között az interakciónak a leírása. A Unified Modelling Language (UML) 2.0 szekvencia diagram leíró részét nagyban inspirálta ez a nyelv. Az MSC főbb elemei:

- MSC head, lifeline és end
- Objektum létrehozása
- Üzenet csere
- Függvényhívás és válasz
- Timer-ek
- Idő intervallumok
- Összetett szerkezetek: alt, opt, parallel, loop (high-MSC)

Az összetett szerkezetek a high-MSC (h-MSC) nevű MSC kiterjesztésben találhatók.

Ezzel a nyelvvel már könnyedén lehet a scenarioban lévő üzeneteket specifikálni, és azokat a rendszer komponenseket, amelyek ezeket az üzeneteket egymásnak küldik.

Az UML és az MSC sokban hasonlítanak, de az alapelveik különbözőek.

MSC-ben a függőleges vonalak („life line”-ok) autonóm entitásokat képviselnek, míg a szekvencia diagramok esetén ezek egy objektumot reprezentálnak. MSC esetén az entitásoknak nem szükséges ugyanazon a számítógépen lenniük.

MSC-ben egy átmenet egy aszinkron üzenetet reprezentál, amely két entitás között jött létre, míg az UML szekvencia diagram leíró nyelvében, egy átmenet egy függvényhívást jelent.

Az MSC-nek sajnos sok hiányossága is tapasztalható. Hiányzik belőle az üzenetek típusossága. Egy követelmény megfogalmazásakor fontos, hogy meg tudjuk mondani, melyek az elvárt üzenetek vagy, hogy melyek jelentenek hibát. Az is nagy hiányosságnak számít,hogy az üzenetekre nem lehetséges megkötéseket megadni. Ez egyben megnehezíti egy követelmény leírását is.

2.2.2. MTL - Metric Temporal Logic

A

2.2.3. MITL - Metric Interval Temporal Logic

2.2.4. PSC – Property Sequence Chart

A Message Sequence Chart-nak nagyon sok hiányossága van. Nem lehet vele megkötéseket definiálni vagy egy üzenetről eldönteni, hogy az egy elvárt vagy nem kívánt üzenet. Ebből kifolyólag az MSC nem egy alkalmas nyelv arra, hogy az üzenet szekvenciáinkat részletesebben specifikálni tudjuk vele.

A Property Sequence Chart[1] az MSC egy kiterjesztése. Sok új elemet vezet be ami nincs az MSC-ben, melyek megtekinthetők a 2.1. táblázaton, mint az üzenet típusokat: sima üzenet (e), elvárt üzenet (r) és nem kívánt üzenet (f). Így specifikálhatjuk, hogy

Tulajdonság	MSC	PSC
Nem kívánt üzenet	-	Fail message
Elvárt üzenet	-	Required message
Sima üzenet	Default message	Regular message
Megkötött sorrendezés	-	Strict sequencing
Gyenge sorrendezés	Seq	Loose sequencing
Üzenet megkötések	-	Constraint
Alternatív lehetőségek	h-MSC	Alternative operator
Párhuzamos művelet	h-MSC	Parallel operator
Ciklus	h-MSC	Loop operator

2.1. táblázat. Az MSC összehasonlítása a PSC-vel

mely üzenetek azok amik helyes viselkedésre utalnak és azok amelyek nem. Az elvárt üzenetek azok az üzenetek amelyeknek feltétetlen meg kell történiük a rendszer működése során. Egy sima üzenet nem jelent hibát a monitor szempontjából ha nem történik meg, viszont ha megjelenik, akkor a scenario-ban utána következő üzenetek ellenőrzésére kell áttérni. Szigorú sorrendezésre is ad megoldást a PSC, ami azt jelenti, hogy megadhatjuk explicit az üzenetek sorrendjét a követelményünkben. A PSC-ben egy üzenetre megkötést is rakhatunk. Megadhatjuk, hogy melyek azok az üzenetek amik nem kívántak az üzenetünk előtt vagy után. A különböző PSC tulajdonságok megtalálhatok a 2.1. ábrán. A nyelv a következő tulajdonságokat támogatja:

- *Sima üzenet (e)*: egy üzenet a scenario-ban, amely ha nem történik meg az a monitor szempontjából nem jelent hibát. Viszont ha megjelenik, akkor a scenario-ban utána következő üzenetek ellenőrzésére kell áttérni. Egy előfeltételt reprezentál.
- *Elvárt üzenet (r)*: egy üzenet amelynek elmaradása hibajelzéshez kell vezessen.
- *Nem kívánt üzenet (f)*: amennyiben a monitor egy ilyen üzenetet detektál, akkor hibát jelez.
- *Üzenet megkötés (constraint)*: Egy üzenetre lehet megkötést is helyezni. Egy megkötés több üzenetet tartalmazhat. Két fajta megkötést definiál a nyelv: múlt- és jövőbéli. A múltbéli üzenet megkötés esetén, az üzenetünk megtörténte előtt, a megkötésben szereplő üzenetek egyike se történhet meg. Jövőbéli megkötés esetén, pedig az üzenetünk megtörténte után nem történhetnek meg a megkötésben szereplő üzenetek.
- *Megkötött sorrendezés (strict ordering)*: A PSC az üzenet lefutási sorrendjének a specifikálására is lehetőséget ad. Egy „a” üzenet megtörténte után, egy adott „b” üzenetnek kell bekövetkeznie. Ha „b” üzenet helyet egy másik üzenet követi az „a” üzenetet, akkor a monitor hibát jelez.

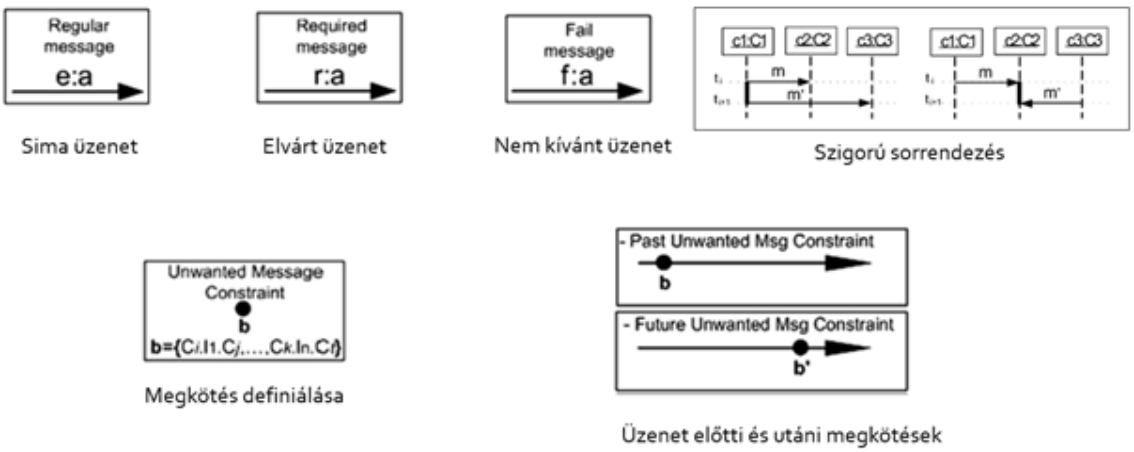
A nyelv támogat összetett szerkezeteket is:

- *Alt operátor*: az alt operátorral alternatív üzenet szekvenciákat lehet definiálni.
- *Par operátor*: a par operátorral meghatározható az üzenet szekvenciák párhuzamos futása.
- *Loop operátor*: a loop operátorral megadhatjuk, hogy egy üzenet szekvencia többször is lefuthat egymás után.

Egy üzeneten egyszerre megkötés és sorrendezés is lehet. Az üzenetek típusát az átmenetén lévő karakterrel jelöljük. Az „e” karakter jelzi, hogy az üzenet sima, az „r” karakter az elvárt üzenetet jelenti, az „f” pedig a nem kívánt üzenetet. Azt meg kell jegyezni, hogy nem kívánt üzenetekre nem lehet jövőbéli megkötéseket rakni. Ezen kívül, ha egy üzeneten megkötött sorrendezést alkalmazunk, akkor nem lehet rajta múltbéli megkötés.

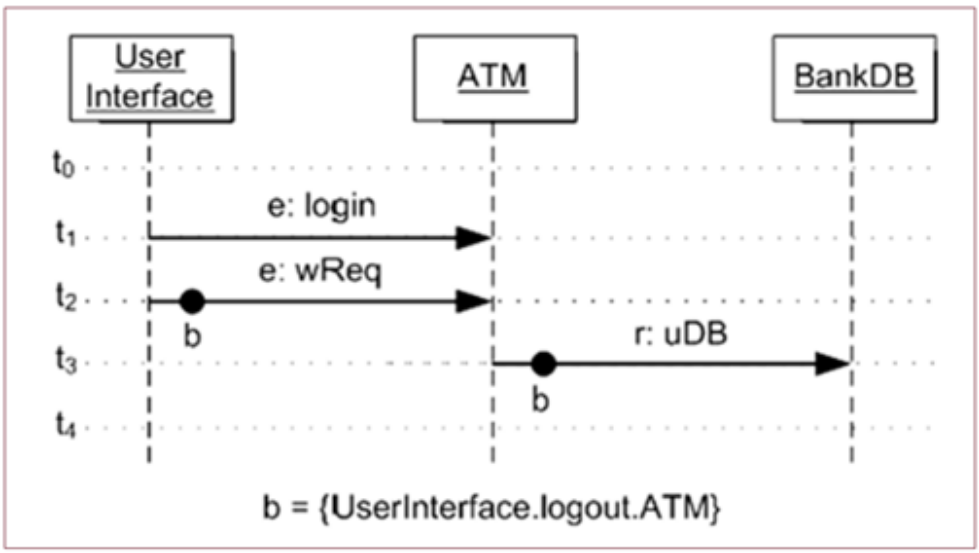
A megkötéseket egy ponttal jelöljük, amit az átmeneten helyezünk el. Ha a pont az átmenet elején van (a feladóhoz közel), akkor az múltbéli megkötést jelöl. Ha pedig a végén helyezkedik el, akkor a jövőbéli megkötést jelöli. A megkötésben lévő üzeneteket egy lista formájába lehet megadni ’’ ’’ jelek közt, a következő módon: <megkötés neve> = C1.I1.Cj, ..., Ck.In.Ct, ahol az üzenetek vesszővel elválasztva, „Feladó.Üzenet.Címzett” formában szerepelnek. A specifikált megkötés nevét pedig az átmeneten lévő pont alá írjuk.

Az üzenetek megkötött sorrendezésének jelölésénél az objektum „life line” vonalát az érintett átmenetek közt folytonossá változtatjuk



2.2. ábra. A PSC különböző elemei[1].

Az üzeneteket a következő formában adjuk meg: *Feladó.Üzenet.Címzett.*



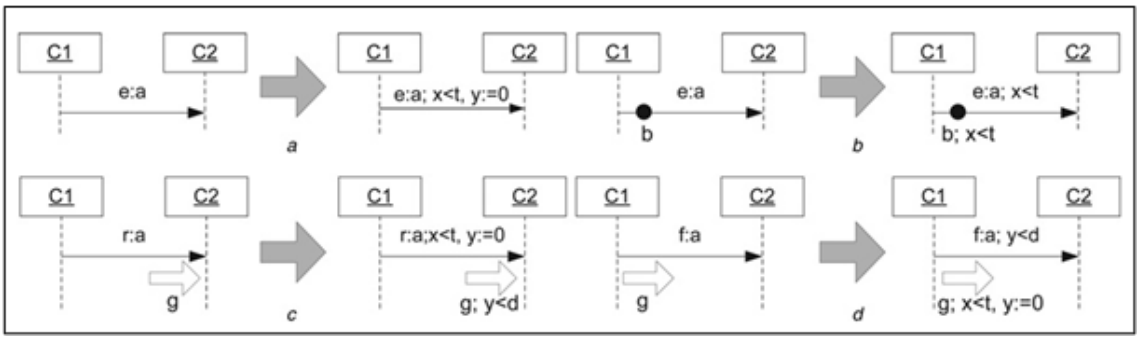
2.3. ábra. PSC diagram egy ATM rendszer működésének ellenőrzésére[1].

A 2.2. ábrán láthatunk egy példát arra, hogy egy követelményt hogyan lehet definiálni. Ez a PSC diagram egy ATM rendszer működését figyeli. Először a felhasználó egy *login* üzenettel bejelentkezik az ATM-be majd egy *wReq* üzenettel egy lekérdezést hajt végre. Ezen az üzeneten van egy megkötés, az üzenet előtt nem történthet kijelentkezés, *logout*. Az ATM ezután, ha nem történt *logout* egy elvárt üzenetet küld a Bank adatbázisába.

A scenario-ink specifikálására most már rendelkezésünkre áll egy grafikus nyelv. A következőkben az lesz a feladatunk, hogy ezeket a diagramokat úgy transzformáljunk, hogy monitor kódot lehessen belőlük készíteni.

2.3. A TPSC bemutatása

A TPSC[2] a PSC-nek egy kiterjesztése. A PSC üzenetekre időzítési feltételeket specifikálhatunk.



2.4. ábra. PSC kiterjesztése időzítési feltételekkel[2].

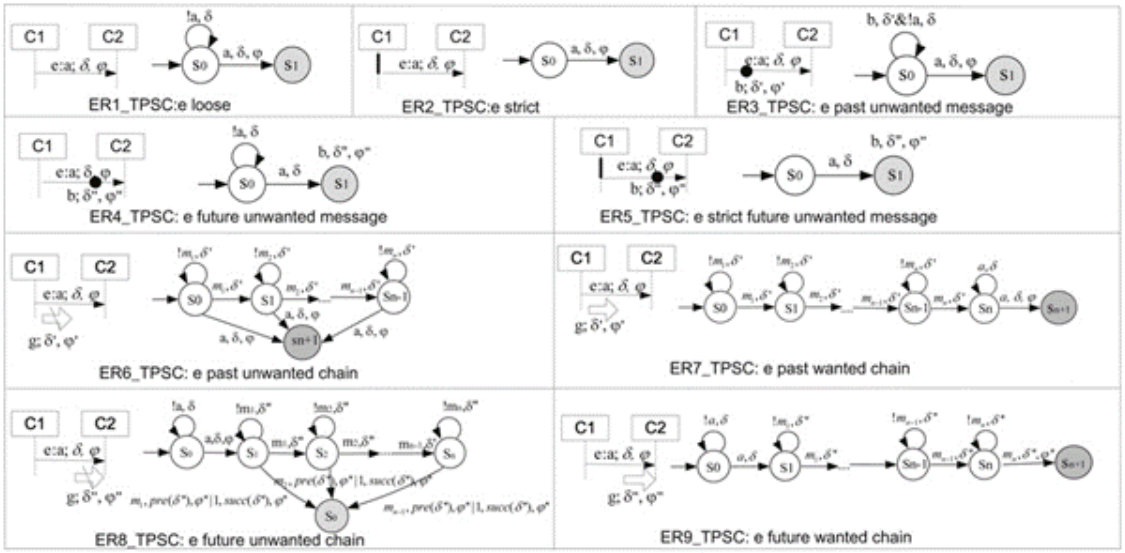
A TPSC óraváltozókat (x, y) használ az időzítéshez. Ezekre meg lehet adni feltételeket, valamint az óraváltozót lehet nullázni. A nullázással adott eseménytől (pl. üzenet vételétől) kezdve indul az időzítés, majd rákövetkező események időbeliségét ez alapján lehet ellenőrizni.

A 3.1. ábrán látható, hogy például az *e: a* sima üzenet *e: a; x < t, y := 0* üzenetre bővül. Elvárjuk, hogy az *a* üzenet *t* idő előtt történjen meg és egy *y* óraváltozót nullázunk. Az *e: a* üzenet egy sima üzenet, szóval ha nem történik meg a specifikált idő intervallumban az nem jelent hibát. Viszont *r: a* üzenetnél már elvárt, hogy *t* időn belül megtörténjen. *f: a* üzenet esetében viszont akkor jelez hibát a monitor, ha üzenet megtörtént *t* időn belül.

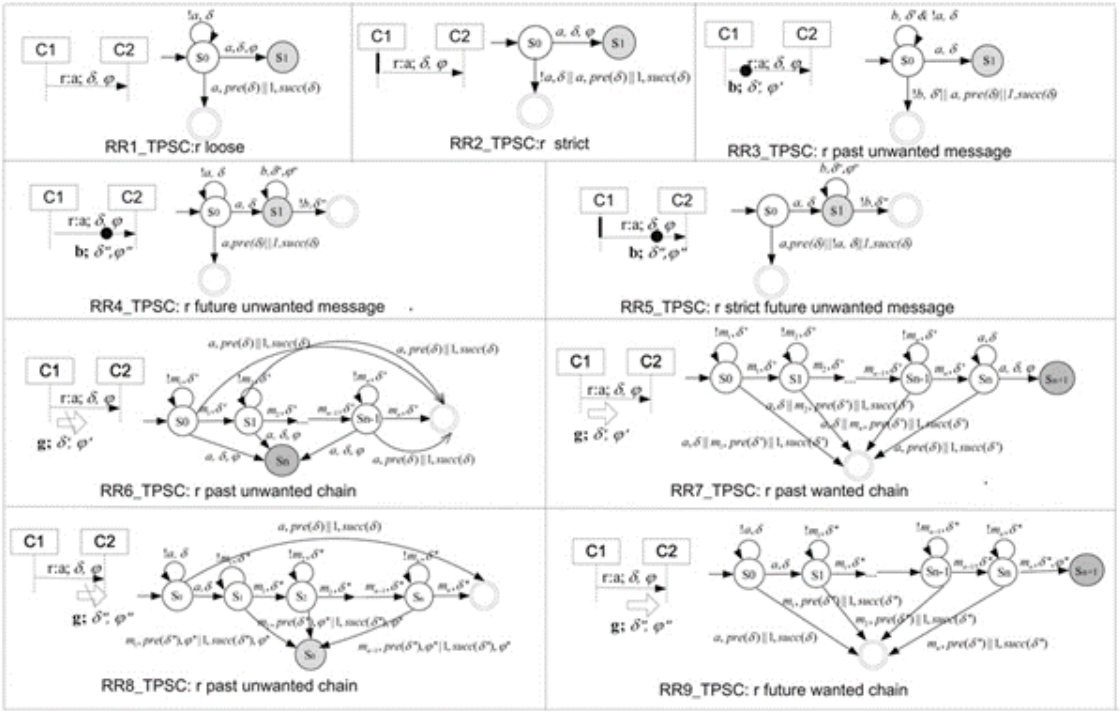
Egy megkötésre is meg lehet adni időzítési feltételt. Így megadhatjuk, hogy mennyi ideig nem szabad jönnie a megkötésben szereplő nem kívánt üzenetek egyikének. Ha a feltételben megadott idő után történik akkor az nem jelent hibát a monitor szempontjából.

2.4. TPSC szcenário alapú automata konstrukció

A monitorozás alapja, hogy TPSC scenario-kból időzített automatákat (Timed Automata) tudjunk készíteni. Egy TA állapotokból, elfogadó állapotokból, feltételekből, akciókból és bemenetekkel címkézett állapotátmenetekből áll. Akkor fogad el egy bemenet sorozatot, ha ennek során elérünk az automata végállapotába. Ha elfogadó állapotot érünk el, akkor az a monitor szempontjából hibát jelent. Az alapelv az az, hogy minden TPSC elemhez tartozik egy minta automata (pattern) ami leírja a szemantikáját. Például a 4.1. és 4.2. ábrákon láthatóak a különböző PSC üzenetekhez tartozó minták.

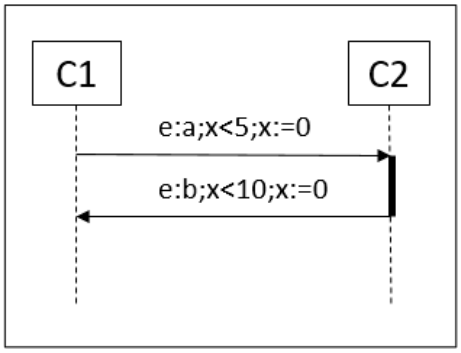


2.5. ábra. Sima üzenetekhez tartozó minták[2].

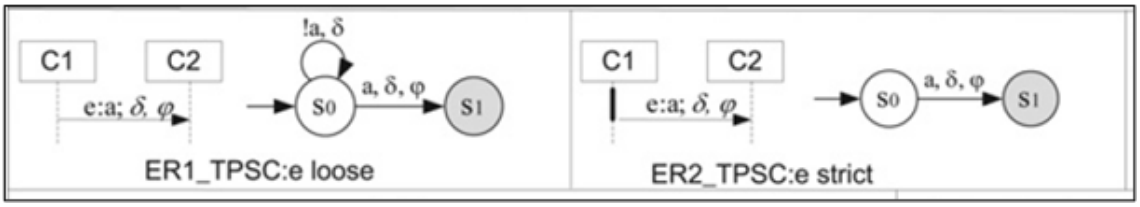


2.6. ábra. Elvárt üzenetekhez tartozó minták[2].

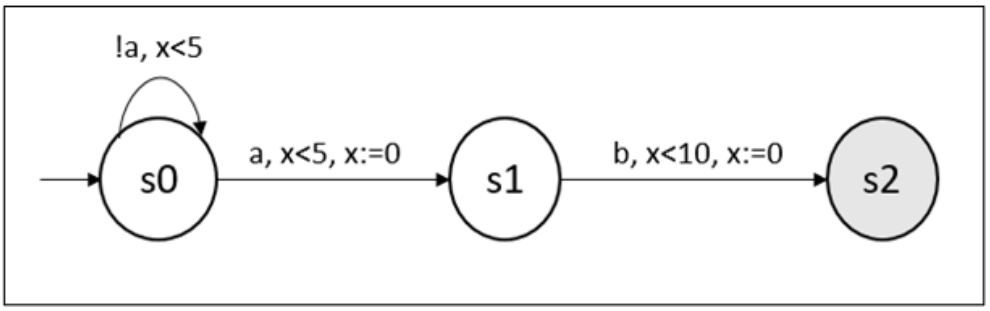
A minta automatánkban található szürke állapotok reprezentálják a végállapotokat. A megkötéseket az automatáknál egy átmenettel definiáljuk ami a nem kívánt üzenetek negáltjainak az ÉS kapcsolata. Például az 4.1. ábrán lévő 3. automata mintán látható „b, δ'&!a, δ" címkéjű hurokélen, a „b" címke minden olyan üzenetnek felel meg, amelyek nincsenek a nem kívánt üzenetek halmazában. A címke teljes jelentése az, hogy ha δ' időn belül nem jött nem kívánt üzenet és δ időn belül nem az „a" üzenet jött akkor maradjunk az s0 állapotban. Ezekből az automata részekből lesznek meghatározott illesztési szabályokkal a scenario-hoz tartozó teljes időzített automaták. Ennek az az alapelve, hogy a scenario-n végig menve az előző minta végállapotát a következő minta kezdőállapotával kell egyesíteni. Ezt a folyamatot mutatják be a 4.3., 4.4. és 4.5. ábrák.



2.7. ábra. TPSC részlet.



2.8. ábra. Alkalmazott automata minták.



2.9. ábra. Az összeállított automata.

A monitor szempontjából az automata akkor jelez hibás működést ha elfogadó állapotba kerül. Ilyenkor a követelmény már nem teljesíthető. Ha az automata egy sima állapotban van, akkor az helyes működést jelent viszont a követelmény még ekkor sem teljesült. A követelmény akkor teljesül amikor az automata a végső (FINAL) állapotba kerül és nem érkezik további amely elmozditaná őt onnan.

2.5. A felhasznált technológiák

2.5.1. Eclipse

Az *Eclipse* egy nyílt forráskódú, platformfüggetlen keretrendszer. Első sorban fejlesztői környezetként használják a fejlesztők. A keretrendszert tovább lehet bővíteni mindenféle plugin telepítésével, így például modellezésre is alkalmas lehet. A diplomaterv során használt Eclipse plugin-ek:

- Xtext
- Xtend
- Sirius

A munkafolyamat egy *workspace*-en belül történik, ahol a fejlesztő létrehozhatja a saját projekteit. Több *workspace*-t is létre lehet hozni és azok között váltani.

Különböző fajta projekteknek különböző nézetei lehetnek. Például egy modellező projektnek van külön model nézete az eszközön belül vagy egy *Java* projektnek *Java* nézete. A nézetek az eszközön megjelenített szövegszerkesztő, fájlkezelő vagy egyéb funkció elhelyezésért, megjelenitéséért felelnek.

2.5.2. Xtext

Az *Xtext Eclipse* plugin-el programozási és domain specifikus nyelveket lehet fejleszteni. A nyelvünk elemeit és szabályait egy nyelvtan segítségével definiálhatjuk. Az *Xtext* keretrendszer több eszközt nyújt a nyelvünkhöz. Például egy parser-t, egy fordítót és egy szerkesztőt. A plugin még egy *Xtend* alapú kódgenerátort is generál a nyelvünkhöz, amivel a nyelvünkhöz tetszőleges kódot tudunk generálni.

```
Scenario:
  'scenario' name=ID '{'
  scenariocontents+=ScenarioContent*
  '}'
;

ScenarioContent:
  alt+=Alt | message+=Message | par+=Par | loop+=Loop | paramConstraint+=ParameterConstraint

;

Message:
  LooseMessage | StrictMessage | PastMessage | FutureMessage | StrictFutureMessage
  | RequiredLooseMessage | RequiredStrictMessage | RequiredPastMessage | RequiredFutureMessage |
     RequiredStrictFutureMessage
  | FailMessage | FailStrictMessage | FailPastMessage
;

LooseMessage:
  'message' name=ID '(' (params+=Params | constantparams+=ConstantParams) ')'
  sender=[Object] '->' receiver=[Object]
  ('clockConstraint' '{' cConstraint=ClockConstraintExpression '}')?
  (resetclock=ResetClock)? ';'
;
```

2.1. kódrészlet. Xtext nyelvtan elemei.

A 2.1-es kódrészlet az Xtext nyelvtan elemeit mutatja be. Egy nyelvtani szabályt egy tetszőleges név megadásával hozhatunk létre. A szabályunkat pedig a név után lévő ":" és ";" közzé írhatjuk. Idézőjelek közé írhatjuk a nyelvünkhöz tartozó kulcsszavakat,

például *'scenario'* vagy idezőjelek között kapcsos zárójel. Ilyen kulcsszavak megadásával jól tudjuk formázni a nyelvtani szabályunkat. Attributumok megadásával tudjuk tárolni a nyelvi elemünk értékeit, változóit. Például a *name* attribútum egy *ID* elemet tárol, ami egy azonosítónak felel meg. Egy attribútumba több elemet is rakhatunk lista szerüen. Ezt a * karakterrel adhatjuk meg, például *scenariocontents+=ScenarioContent**, ahol a *scenariocontents* több *ScenarioContent*-beli elemet tárol.

A nyelvtanukban megadhatunk elágazó szabályokat is a "|" karakterrel. Ilyen szabály például a *Message*. Továbbá hivatkozhatunk már létrehozott nyelvi elemekre is a szabályunkban a "[]" szintakszissal. Például a *LooseMessage sender* attribútuma egy már létrehozott *Object* elemre hivatkozik.

A "?" karakter opciónális nyelvi elemek megadására szolgál.

Fordítás után a nyelvi elemeinkből *Xtend* és *Java* osztályok generálodnak.

2.5.3. Xtend

Az *Xtend* egy magasszintű programozási nyelv. A *Java Virtual Machine* plaformot használja. Szintaktikailag és szemantikailag nagyon hasonlít a *Java* nyelvhez, mondhatni a Java kibővítése. Az *Xtend* osztályokból *Java* osztályok készülnek.

Egy nagyon hasznos funkciója az *Xtend*-nek a *template*. Sablonokat tudunk létrezhozni vele függvényeken belül, amely megkönnyíti a kódgenerálást. Így könnyedén tudunk hivatkozni az *Xtext* nyelvi elemeinkre függvény paramétereken keresztül és egyedi kódgenerátort fejleszteni a nyelvünkhöz. A sablon függvényen belül a "«»" karaktereket használva tudunk hivatkozni az *Xtext* nyelvi elemeinkre.

2.5.4. Sirius

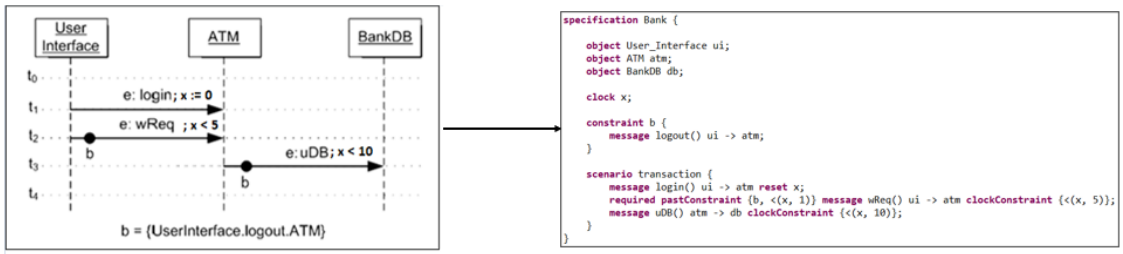
A *Sirius* eszköz segítségével létrehozhatunk saját grafikus modellező alkalmazásainkat. Egy szerkesztő környezetet alkothatunk, amivel a modellünk elemeit hozhatjuk létre vagy szerkeszthetjük grafikusan. A plugin az *Entity Modelling Framework* keretrendszer használja a modellek feldolgozásához és ilyen *EMF* modelleket jelenít meg.

3. fejezet

Szöveges PSC leíró nyelv kibővítése

A nyelvet az Xtext technológia segítségével definiáltam. A nyelv két új elemmel bővült:

- időzített feltétel
- óraváltozó nullázása



3.1. ábra. TPSC diagramból szöveges leírás az Xtext nyelv használatával.

A 6.1. ábrán látható, hogy egy TPSC diagramot hogyan tudunk leírni a nyelvünk segítségével. Definiálhatjuk a diagramban szereplő objektumokat, a megkötéseket amiket használni fogunk és végül, hogy milyen üzenetek vannak a követelményünkben.

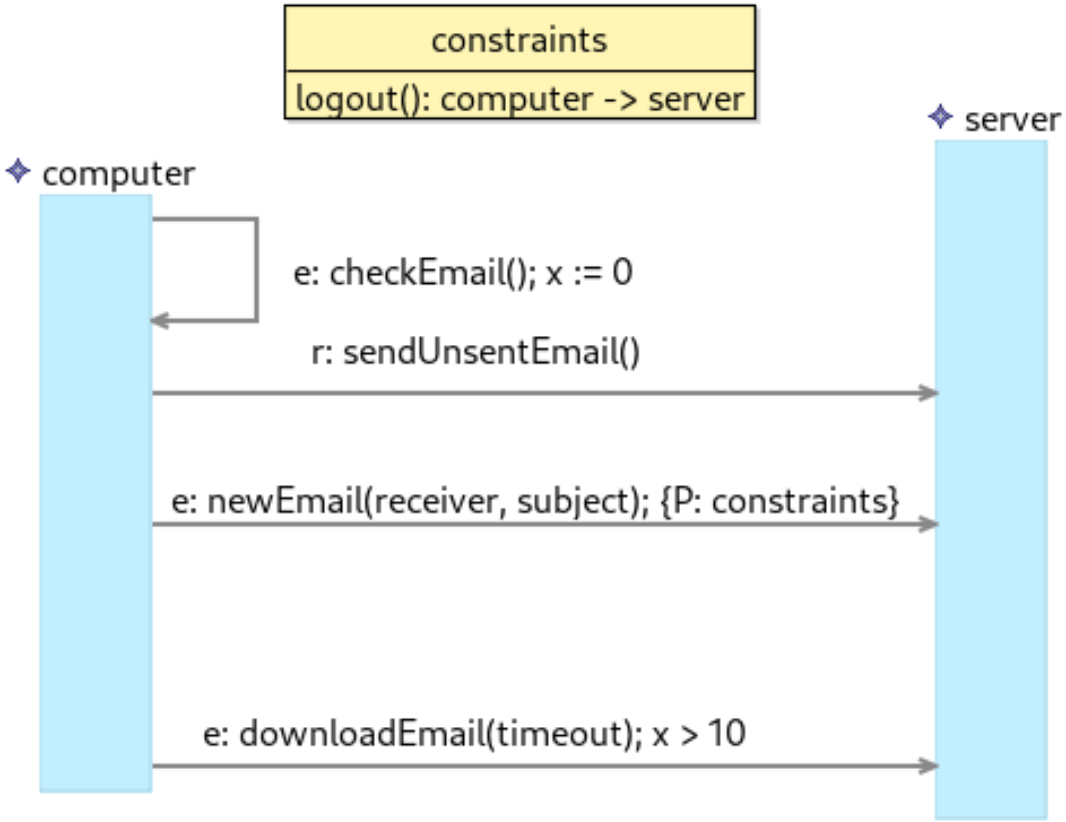
3.2. ábra. Egy TPSC üzenet felépítése a definiált Xtext nyelvben.

A 6.2. ábrán látszik, hogy megjelenik a *clockConstraint* kulcsszó ami egy időzítési feltétel megadására szolgál. A kulcsszó után kapcsos zárójelek közt megadható a feltétel. A *reset* kulcsszó az óraváltozó nullázására szolgál.

4. fejezet

TPSC specifikációk vizualizációja

A specifikációk vizualizációjához a *Modell alapú rendszertervezés* tárgy során készített PSC vizualizációs Sirius alkalmazást használtam fel. Először kiegészítettem az alkalmazást TPSC elemek vizualizációjával. Egy *XML* generátor előállítja a specifikáció *XMLs* leírását, amit a Sirius alkalmazás képes feldolgozni és előállítani a hozzá tartozó diagramot.



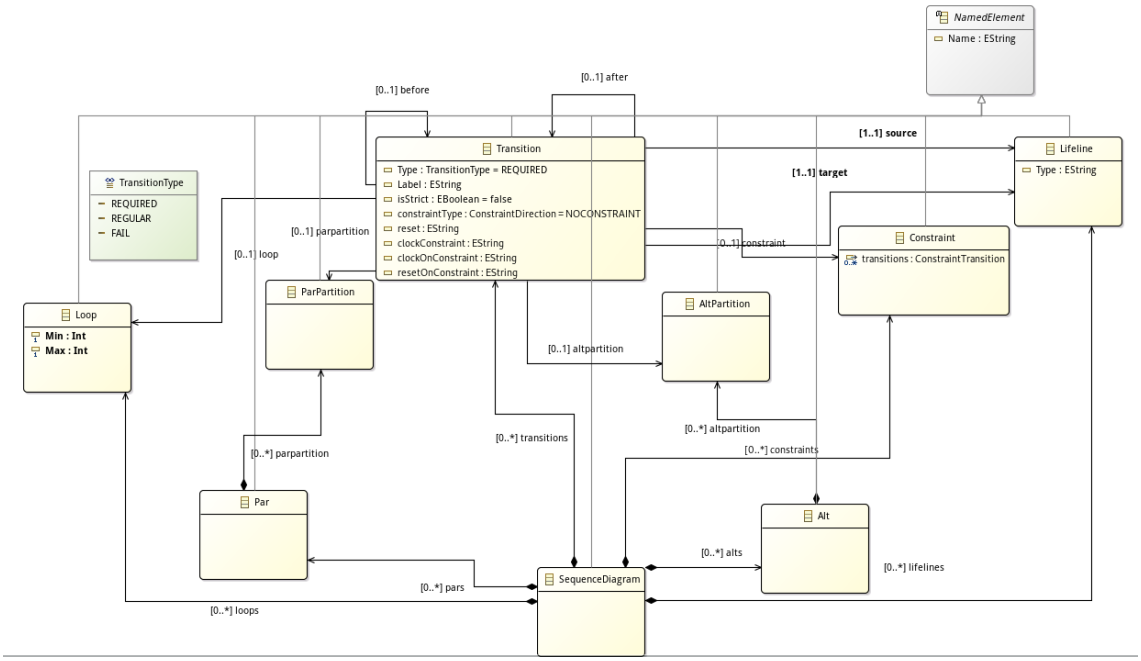
4.1. ábra. Szenárió diagram.

A diagramon az egyes objektumok kék *lifeline* formájában jelenek meg. Minden üzenethez tartozik egy nyíl. A nyíl címkéjére írjuk az üzenet összes tulajdonságát. A nyíl eleje és vége *lifeline*-okat kötnek össze, amik az üzenet feladóját és fogadóját jelzik. A megkötéseket egy sárga táblázat formájában reprezentáljuk, amelybe bele írjuk az összes megkötésben szereplő üzenetet.

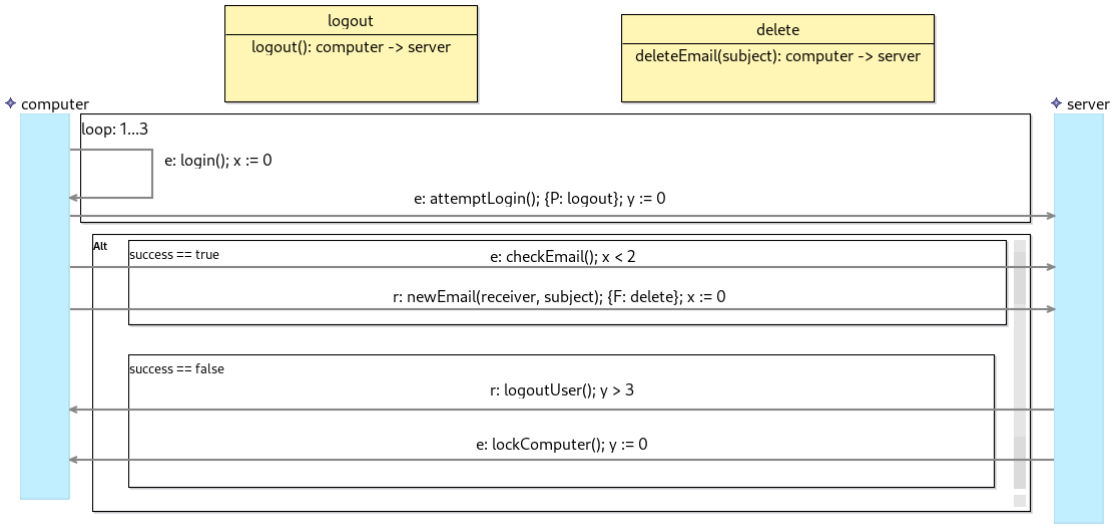
A diagramon keretek formájában jelenek meg az operátorok.

```
<?xml version="1.0" encoding="UTF-8"?>
<minotor:SequenceDiagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:minotor="hu.bme.
    mit.mdsd.xboyz.erdiagram" Name="Email">

<lifelines Name="computer" Type="Computer"/>
<lifelines Name="server" Type="Server"/>

<constraints Name="constraints">
<transitions Name="logout()" source="//@lifelines.0" target="//@lifelines.1"/>
</constraints>

<transitions Name="checkEmail()" Type="REGULAR" Label="e: checkEmail()" source="//@lifelines.0"
    target="//@lifelines.0"  after="//@transitions.1" reset="x"/>
<transitions Name="sendUnsentEmail()" Type="REQUIRED" Label="r: sendUnsentEmail()" source="//
    @lifelines.0" target="//@lifelines.1" before="//@transitions.0" after="//@transitions.2"/>
<transitions Name="computerError()" Type="FAIL" Label="f: computerError()" source="//@lifelines.1"
    target="//@lifelines.0" before="//@transitions.1" after="//@transitions.3"/>
<transitions Name="serverError()" Type="FAIL" Label="f: serverError()" source="//@lifelines.0"
    target="//@lifelines.1" before="//@transitions.2" after="//@transitions.4"/>
<transitions Name="newEmail(receiver, subject)" Type="REGULAR" Label="e: newEmail(receiver, subject)
    " source="//@lifelines.0" target="//@lifelines.1" before="//@transitions.3" after="//
    @transitions.5" constraint="//@constraints.0" constraintType="PAST"/>
<transitions Name="downloadEmail(timeout)" Type="REGULAR" Label="e: downloadEmail(timeout)" source="
    //@lifelines.0" target="//@lifelines.1" before="//@transitions.4"   clockConstraint="x &gt; 10"
    />
</minotor:SequenceDiagram>
```

4.1. kódrészlet. Szenárió diagram xml leírása.



4.2. ábra. Sirius alkalmazáshoz tartozó model ER diagramja.



4.3. ábra. Operátorokat tartalmazó szenárió diagram.

5. fejezet

Monitor forráskód generálás

5.1. Időzített automata generátor

5.1.1. Az automata generátor célja

Az önálló laboratórium során elkészített automata generátort kibővítettem úgy, hogy támogassa a TPSC elemekhez tartozó automata minták generálását. Bemenetként egy TPSC scenario szöveges leírását kapja meg amiből a minta alapú módszerrel generál egy TA automatát.

A 7.1. és 7.2. kódrészleteken látható, hogy a monitor generátor támogatja az alt, par és loop operátorokat tartalmazó TPSC-khez tartozó TA-k generálását is. A 7.2. kódrészlet a generált automata "never claim" leírását tartalmazza. A Never claim a Promela nyelv része, ezzel egy rendszer viselkedését lehet definiálni. Továbbá a generátor képes az üzenet paraméterek kezelésére. Például az alt operátor feltételét képes feldolgozni és azt elhelyezni a generált automata megfelelő élén.

```
specification Bank {

  object UserInterface ui;
  object ATM atm;
  object BankDB db;

  bool success = true;

  constraint b {
    message logout() ui->atm;
  }

  scenario transaction {
    message login(success) ui->atm;

    alt (equals(success, true)) {
      message wReq() ui->atm;
      message uDB() atm->db;
    } (equals(success, false)) {
      message loginUnsuccessful() ui->atm;
      message lockMachine() required atm->ui;
    }
  }
}
```

5.1. kódrészlet. Alt operátort tartalmazó scenario.

```
bool success = true;

never{ /*transactionMonitor*/
T0_init:
 if
 :: (!(ui.login(success).atm)) -> goto T0_init
 :: (ui.login(success).atm) -> goto T0_q1
 fi;
T0_q1:
 if
 :: (epsilon) -> goto T0_qinit0
 fi;
T0_qinit0:
 if
 :: (epsilon; success == true) -> goto T0_q2
 :: (epsilon; success == false) -> goto T0_q5
 fi;
T0_qfinal1:
 if
 fi;
T0_q2:
 if
 :: (!(ui.wReq().atm)) -> goto T0_q2
 :: (ui.wReq().atm) -> goto T0_q3
 fi;
T0_q3:
 if
 :: (!(atm.uDB().db)) -> goto T0_q3
 :: (atm.uDB().db) -> goto T0_q4
 fi;
T0_q4:
 if
 :: (epsilon) -> goto T0_qfinal1
 fi;
T0_q5:
 if
 :: (!(ui.loginUnsuccessful().atm)) -> goto T0_q5
 :: (ui.loginUnsuccessful().atm) -> goto T0_q6
 fi;
T0_q6:
 if
 :: (!(atm.lockMachine().ui)) -> goto T0_q6
 :: (!(atm.lockMachine().ui)) -> goto accept_q7
 :: (atm.lockMachine().ui) -> goto T0_q8
 fi;
accept_q7:
 if
 fi;
T0_q8:
 if
 :: (epsilon) -> goto T0_qfinal1
 fi;
}
```

5.2. kódrészlet. Alt operátort tartalmazó scenario never claim leírása.

5.1.2. Az automata generátor megvalósítása

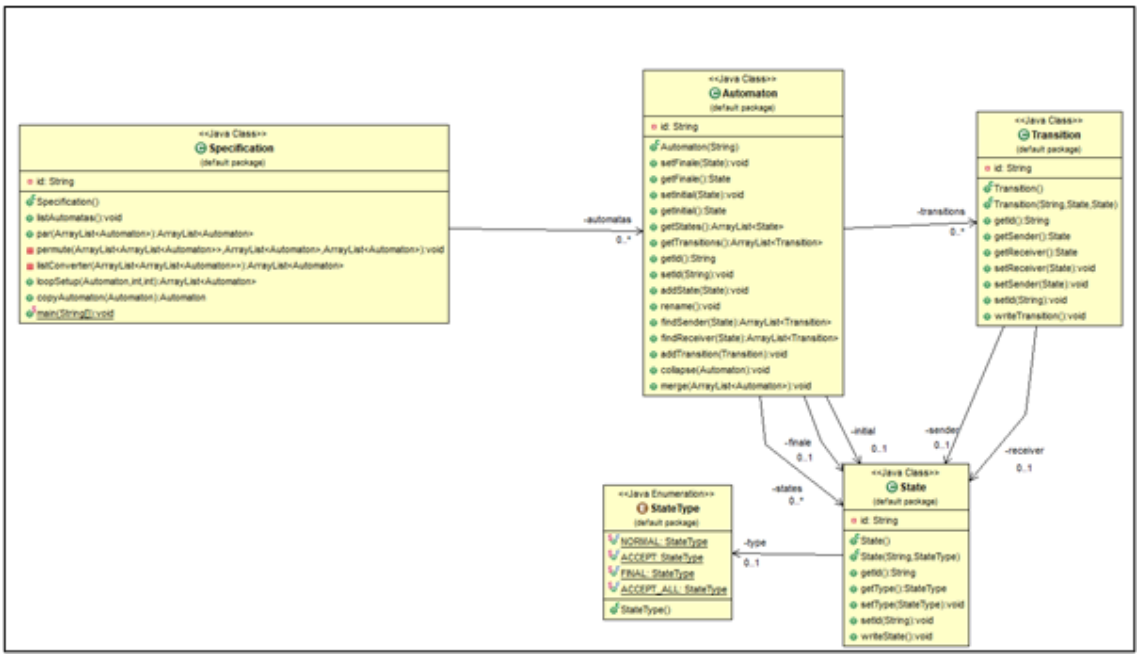
A generátorhoz az Xtend technológiát használtam. Minden egyes TPSC üzenethez legenerálja a hozzá tartozó minta automatát, majd elvégzi azok összecsatolását.

Az időzített automaták generálásához egy adatstruktúrát definiáltam, amely a következő Java osztályokból és interfészekből áll:

- AltExpressionInterface
- Automaton
- BasicTransition

- ClockConstraint
- ClockExpressionInterface
- Constraint
- EpsilonTransition
- OperatorFunctions
- State
- StateType
- Transition
- UnwantedConstraint
- WantedConstraint

A 7.1. ábrán látható az adatstruktúra UML osztály diagramja.



5.1. ábra. Az adatstruktúra UML osztály diagramja.

Az automatában lévő állapotok implementációja a State osztályban található. Két attribútuma van: id(String), a címkéje tárolására, és type(StateType), az állapot típusa.

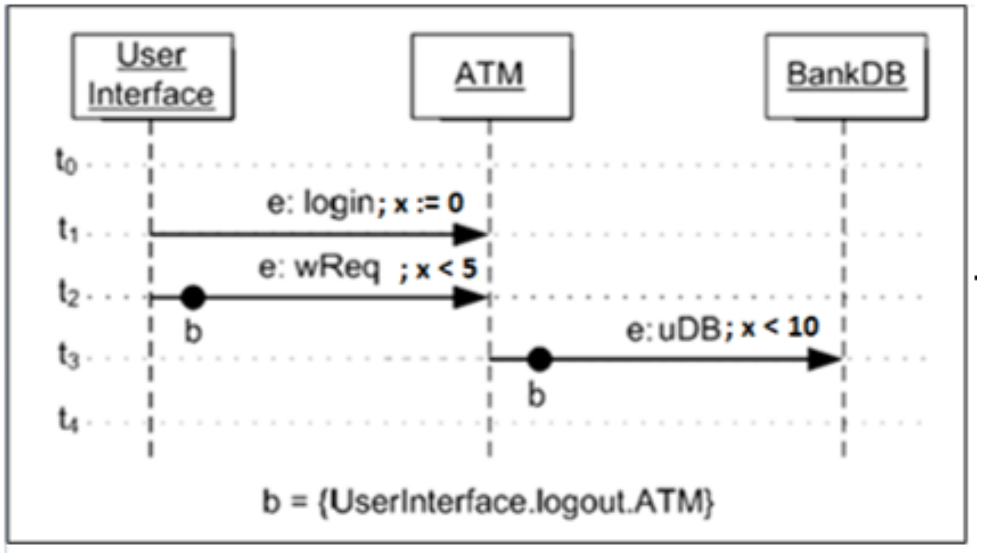
Az állapot típusának a megadására a StateType enum osztályt definiáltam. NORMAL, ACCEPT, FINAL értékeket lehet benne eltárolni. Az átmenetek implementációjáért felelős osztály a Transition. Három tag változója van: id(String) az üzenet, sender(State), a feladó állapot, és receiver(State) a fogadó állapot.

Az időzített automata implementációja az Automaton osztályban található. Itt tároljuk az automatában lévő állapotokat és a köztük lévő átmeneteket egy-egy listában. Az Automaton osztály addState(State) és addTransition(Transition) függvényeivel lehet új állapotot és átmenetet hozzáadni az automatához, a collapse(Automaton) függvényével pedig két automatát egyesíteni. Ezt a függvényt használtam az implementációban a minta

automaták egyesítésére. Ezen kívül az osztálynak van egy merge(ArrayList<Automaton>) függvénye. Ez a függvény a minta automata ágak egyesítésére szolgál.

A Specification osztály feladata, hogy összeállítsa a szöveges leírásban specifikált TPSC scenariohoz tartozó időzített automatát. Ezt követően az automata Never Claim leírását egy .txt kiterjesztésű fájlba írja.

5.1.3. Mintapélda



5.2. ábra. Példa TPSC diagram.

```
specification Bank {

  object User_Interface ui;
  object ATM atm;
  object BankDB db;

  clock x;

  constraint b {
    message logout() ui -> atm;
  }

  scenario transaction {
    message login() ui -> atm reset x;
    required pastConstraint {b, <(x, 1)} message wReq() ui -> atm clockConstraint {<(x, 5)};
    message uDB() atm -> db clockConstraint {<(x, 10)};
  }
}
```

5.3. kódrészlet. TPSC scenario szöveges leírása.

A 7.3, 7.4. kódrészleteken és 7.2. ábrán látható, hogy a generátor milyen időzített automatát generál a megadott TPSC scenario-ból.

```
never{ /*transactionMonitor*/
T0_init:
 if
 :: (!(ui.login().atm); ) -> goto T0_init
 :: (ui.login().atm; x = 0) -> goto T0_q1
 fi;
T0_q1:
 if
 :: (!(ui.wReq().atm); x < 5 & !(ui.logout().atm); x < 1)) -> goto T0_q1
 :: (ui.wReq().atm; x < 5) -> goto T0_q3
 :: ((!(!(ui.logout().atm); x < 1); x < 5) || (ui.wReq().atm; )) || (1, x >= 5))) -> goto accept_q2
 fi;
accept_q2:
 if
 fi;
T0_q3:
 if
 :: (!(atm.uDB().db); x < 10;) -> goto T0_q3
 :: (atm.uDB().db; x < 10;) -> goto T0_q4
 fi;
T0_q4:
 if
 fi;
}
```

5.4. kódrészlet. Generált időzített automata never claim formátumban.

5.2. Monitor forráskód generátor

5.2.1. A monitor interfészei

A monitorozás alatt lévő rendszer egy közös interfészen keresztül kommunikál a monitorral. Az interfész Java implementációja a 8.1. kódrészleten tekinthető meg. A monitor azt vizsgálja, hogy a rendszer a scenario szerint működik-e.

Monitor interfész:

- update(): a monitorban tárolt rendszer állapotát frissíti a paraméterben kapott üzenet alapján.
- goodStateReached(): a rendszer aktuális állapotát jelzi.
- requirementSatisfied(): jelzi hogy a rendszer megfelel-e a követelménynek
- errorDetected(): detektált hiba jelzésére szolgál

Az update() függvényt a rendszer hívja, hogy továbbítsa az üzenetet a monitornak. Paraméterként az üzenet küldőjét (sender), fogadóját (receiver), üzenet nevét (messageType) és az üzenet paramétereit várja (parameters). A goodStateReached() függvény jelzi, hogy a rendszer jelenlegi működése megfelel-e a követelménynek. A requirementSatisfied() függvény visszaadja, hogy a követelmény teljesült-e. Ha a követelménynek megfelelt a rendszer viselkedése akkor igazat ad vissza, amúgy hamisat.

Az üzenetek megfigyeléséhez szükséges segédfüggvényeket a kommunikációs infrastruktúrához kézzel kell megírni. Ezek a monitort az update() függvényen keresztül hívják.

```
public interface IMonitor {
  public boolean goodStateReached();
  public void update(String sender, String receiver, String messageType, String[] parameters);
  public boolean requirementSatisfied();
  public void errorDetected(String sender, String receiver, String messageType, String[] parameters)
     ;
}
```

5.5. kódrészlet. Monitor interfész Java implementációja.

Az időzitő komponenshez tartozik egy időzitő interfész amin keresztül elérhető a komponens. Ezen az interfészen keresztül lehet az óraváltozokat lekérdezni vagy nullázni. Két függvénye van:

- getClock(String clock): óraváltozó lekérdezése név alapján
- resetClock(String clock): óraváltozó nullázása név alapján

```
public interface IClock {
  public long getClock(String clock);
  public void resetClock(String clock);
}
```

5.6. kódrészlet. Időzitő interfész Java implementációja.

A 8.2. kódrészlet tartalmazza a IClock java interfészt.

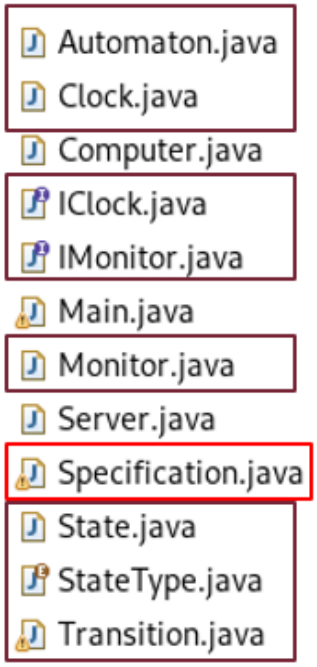
5.2.2. A monitor forráskód megvalósítása

A generált forráskód struktúrája egy statikus és egy generált dinamikus részből áll. A statikus részbe az időzitett automata java osztályai kerülnek:

- State: egy állapotot leíró osztály
- Transition: egy élet reprezentáló osztály
- Automaton: egy automatát megvalósitó osztály

A monitor interfész, a monitor java osztálya, az időzítő interfész és a hozzá tartozó java osztály is ebbe a részbe tartozik.

A dinamikus részben található a Specification Java osztály, ami a scenario alapján generált automata forráskódját tartalmazza. Ezt a 8.1. ábra mutatja be. A dinamikus rész pirosan van bekeretezve és a statikus rész pedig sötét vörösen.



5.3. ábra. A monitor forráskódjának struktúrája.

A szükséges forráskódok generálásához az Xtend technológiát használtam.

5.2.3. Mintapélda

A 8.3. kódrészleten látható egy scenario követelmény, amit egy okos telefon működésére specifikáltunk. Az okos telefonon van egy zene lejátszási lista generáló alkalmazás. A követelményben azt várjuk el, hogy ha a felhasználó megnyitja az alkalmazást akkor a belső kamera készít az arcáról egy képet. A kép alapján eldönti, hogy milyen a felhasználó kedve és az alapján előállít egy zene lejátszási listát.

A 8.5. kódrészleten látható az okos telefon és a monitor közti kapcsolat megvalósítása Java kódban. A monitor a rendszertől kapott üzenetek alapján jelzi, hogy a követelmény alapján mi a rendszer állapota.

```
specification Photo{

  object User user;
  object Device device;
  object Database db;

  constraint error {
    message closeApp() user -> device;
  }

  scenario playlist_generation{
    message openApp() user -> device;
    message accessWebcam() device -> device;
    required message getPhoto() device -> user;
    fail message cameraOffline() user -> device;
    required strict message retrieveMood() device -> db;
    required message retrieveMusic() device -> db;
    strict message generatePlaylist() db -> device;
  }
}
```

5.7. kódrészlet. Okos telefon működésére megadott scenario követelmény.

A 8.6. kódrészlet az okos telefon Java osztálya. Megtekinthető a monitor és az eszköz közti kommunikáció megvalósítása is.

A 8.7. kódrészleten látszik, hogy a rendszer a működése elején nem felelt meg a monitor követelményének. Amikor a működése végére ért akkor a monitor jelezte, hogy a követelmény teljesült a „Good state” üzenettel. A mintához tartozó Specification osztály a függelékben található. A generált automatát a konstruktorában állítja elő.

5.2.4. Összetett szerkezetek

A monitor forráskód generátor támogatja az alt, par vagy loop operátorokat tartalmazó scenariokat is. Erre példát a függelékben lehet találni.

```
never{ /*playlist_generationMonitor*/
T0_init:
 if
 :: (!(user.openApp().device)) -> goto T0_init
 :: (user.openApp().device) -> goto T0_q1
 fi;
T0_q1:
 if
 :: (!(device.accessWebcam().device)) -> goto T0_q1
 :: (device.accessWebcam().device) -> goto T0_q2
 fi;
T0_q2:
 if
 :: (!(device.getPhoto().user)) -> goto T0_q2
 :: (!(device.getPhoto().user)) -> goto accept_q3
 :: (device.getPhoto().user) -> goto T0_q4
 fi;
accept_q3:
 if
 fi;
T0_q4:
 if
 :: (!(user.cameraOffline().device)) -> goto T0_q6
 :: (!(user.cameraOffline().device)) -> goto T0_q4
 :: (user.cameraOffline().device) -> goto accept_q5
 fi;
accept_q5:
 if
 fi;
T0_q6:
 if
 :: (device.retrieveMood().db) -> goto T0_q8
 :: (!(device.retrieveMood().db)) -> goto accept_q7
 fi;
accept_q7:
 if
 fi;
T0_q8:
 if
 :: (!(device.retrieveMusic().db)) -> goto T0_q8
 :: (!(device.retrieveMusic().db)) -> goto accept_q9
 :: (device.retrieveMusic().db) -> goto T0_q10
 fi;
accept_q9:
 if
 fi;
T0_q10:
 if
 :: (db.generatePlaylist().device) -> goto T0_q11
 fi;
T0_q11:
 if
 fi;
}
```

5.8. kódrészlet. Generált automata Never claim formátumba.

```
public class Main {
  public static void monitorStatus(String status) {
    System.out.println(status);
  }

  public static void main(String[] args) {
    Specification specification = new Specification();
    specification.listAutomatas();
    IMonitor monitor = new Monitor(specification.getAutomata().get(0));

    User user = new User();
    Device device = new Device();
    Database db = new Database();
    user.device = device;
    device.user = user;
    device.db = db;
    db.device = device;
    user.monitor = monitor;
    device.monitor = monitor;
    db.monitor = monitor;

    user.init();
  }
}
```

5.9. kódrészlet. Az okos telefon és hozzá tartozó monitor fel konfigurálásának Java implementációja.

```
public class Device {
  public IMonitor monitor;
  public User user;
  public Database db;

  void openApp() {
    monitor.update("user", "device", "openApp", new String[] {});
    accessWebcam();
  }

  void accessWebcam() {
    monitor.update("device", "device", "accessWebcam", new String[] {});
    user.getPhoto();
    db.retrieveMood();
    db.retrieveMusic();
  }

  void cameraOffline() {
    monitor.update("user", "device", "cameOffline", new String[] {});
  }

  void generatePlaylist() {
    monitor.update("db", "device", "generatePlaylist", new String[] {});
  }
}
```

5.10. kódrészlet. Az okos telefon Java osztálya.

```
transition: user.openApp().device
q1
System is in bad state.
transition: device.accessWebcam().device
q2
System is in bad state.
transition: device.getPhoto().user
q4
System is in bad state.
transition: !(user.cameraOffline().device)
q6
System is in bad state.
transition: device.retrieveMood().db
q8
System is in bad state.
transition: device.retrieveMusic().db
q10
System is in bad state.
transition: db.generatePlaylist().device
q11
System is in good state.
```

5.11. kódrészlet. Monitor kimenete a rendszer működésének egyes fázisaiban.

5.2.5. Időzítési feltételek

A monitor forráskód generátor támogatja az időzítési feltételeket tartalmazó scenariokat is. A 8.8. kódrészletben található scenario első üzenetén a ”reset x” címke jelzi a monitornak, hogy az ”x” óraváltozó nullázni kell. Egy óraváltozó felvételét is a ”reset” címkével lehet végrehajtani. A 8.9. kódrészlet a példához tartozó Main java osztály leírását tartalmaza, a 8.10. kódrészlet pedig a rendszerhez tartozó Computer java osztályt. A 8.11. kódrészleten megtekinthető a monitor kimenete. A kimenet végén lévő ”System is in good state.” üzenet jelzi, hogy a rendszer működése megfelelt a követeménynek. A kimenetben szereplő ”bad state” üzenetek megtévesztőek lehetnek. A dolgozat 4. fejezetében említettem, hogy milyen esetekben helyes vagy helytelen a monitor szempontjából a rendszer viselkedése. A ”bad state” üzenet nem feltétlen jelenti azt, hogy a rendszer viselkedése helytelen csupán azt, hogy a feltétel még nem teljesült. A monitor tovább szeretném fejleszteni a diplomaterv során, úgy hogy pontosabban jelezze a rendszer működésének állapotát.

```
specification Email {

  object Computer computer;
  object Server server;

  clock x;

  constraint constraints{
    message logout() computer -> server;
  }

  constraint c {
    message login() server -> computer;
  }

  scenario sendEmail{
    message checkEmail() computer -> computer reset x;
    message sendUnsentEmail() required computer -> server;
    message newEmail() computer -> server pastConstraint {constraints};
    message downloadEmail() computer -> server clockConstraint {x < 10};
  }
}
```

5.12. kódrészlet. Időzítési feltételeket tartalmazó scenario.

```
public class Main {
  public static void monitorStatus(String status) {
    System.out.println(status);
  }

  public static void main(String[] args) {
    Specification specification = new Specification();
    specification.listAutomatas();
    IClock clock = new Clock();
    IMonitor monitor = new Monitor(specification.getAutomata().get(0), clock);

    Server server = new Server(monitor);
    Computer computer = new Computer(server, monitor);
  }
}
```

5.13. kódrészlet. Időzítéses példához tartozó Main osztály.

Az óraváltozók és időzítések megvalósításához az ”org.apache.commons.lang3” könyvtár ”time” csomag *StopWatch* osztályát használtam. Ha az automata élén van egy időzítési feltétel akkor a monitor komponens az időzítő komponenstől elkéri a feltételben szereplő

```
public class Computer {
  public Server server;
  public IMonitor monitor;

  Computer(Server server, IMonitor monitor) {
    this.server = server;
    this.monitor = monitor;
    monitor.update("computer", "computer", "checkEmail", new String[] {});
    checkEmail();
  }

  void checkEmail() {
    monitor.update("computer", "server", "sendUnsentEmail", new String[] {});
    server.sendUnsentEmail();

    monitor.update("computer", "server", "newEmail", new String[] {});
    server.newEmail();

    monitor.update("computer", "server", "downloadEmail", new String[] {});
    server.downloadEmail();
  }
}
```

5.14. kódrészlet. A Computer java osztálya.

```
Received Message: computer.checkEmail().computer
Transition: !(computer.checkEmail().computer)
Transition: computer.checkEmail().computer
transition triggered: computer.checkEmail().computer
q1
System is in bad state.
Received Message: computer.sendUnsentEmail().server
Transition: !(computer.sendUnsentEmail().server)
Transition: !(computer.sendUnsentEmail().server)
Transition: computer.sendUnsentEmail().server
transition triggered: computer.sendUnsentEmail().server
q3
System is in bad state.
Received Message: computer.newEmail().server
Transition: !(computer.logout().server) & !(computer.newEmail().server)
Transition: computer.newEmail().server
transition triggered: computer.newEmail().server
q4
System is in bad state.
Received Message: computer.downloadEmail().server
Transition: !(computer.downloadEmail().server)
Transition: computer.downloadEmail().server
transition triggered: computer.downloadEmail().server
q5
System is in good state.
```

5.15. kódrészlet. Időzítéses példa monitor kimenete.

óraváltozóban tárol időt és kiértekeli a feltételt. Ha a feltétel teljesül, akkor az időzítés szempontjából a tranzíció megtörténhet.

6. fejezet

A generált monitor forráskód helyességének tesztelése

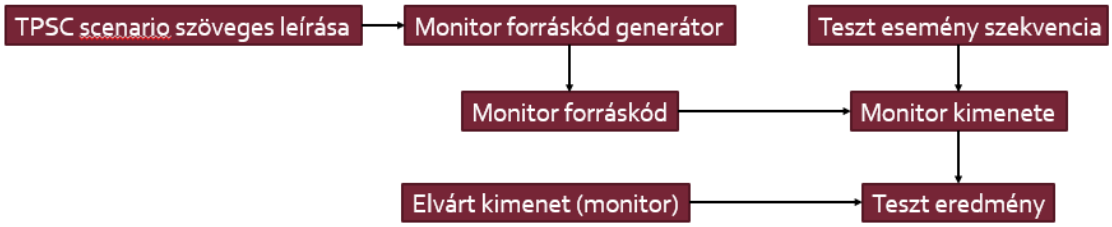
6.1. Tesztelési célok

A generált monitor forráskód tesztelésére a következő célokat fogalmazuk meg:

- Az összes üzenet típus megjelenése a különböző teszt szenáriókban
- Időzített feltételek helyes kiértékelése
- Üzenet megkötések tesztelése
- Alt és Par operátorok esetén, az üzenet szekvencia ágak helyes kiértékelése
- Loop operátor esetén a minimális és maximálás üzenet ismétlődések tesztelése
- Összetett szenárió tesztelése, ami több operátort tartalmaz
- Egymást követő elvárt üzeneteket tartalmazó szenárió tesztelése
- Egymást követő fail üzeneteket tartalmazó szenárió tesztelése
- Regular üzenet tesztelése (ha megjelenik a rendszer működésében akkor ki kell értékelni a szenárió többi részét, ha nem jelenik meg a monitor helyes működést kell jelezzen és, hogy a követelmény nem teljesült)
- Több óraváltozót tartalmazó szenárió tesztelése
- Egymást követő üzenetek kombinációinak tesztelése (pl. elvárt üzenetet követő fail, elvárt üzenetet követő reguláris, stb.)

6.2. Monitor forráskód generátor tesztelése

A generált monitor forráskodját integrációs tesztek segítségével szeretnénk tesztelni. A teszteinkben egy példa rendszer java implementációját integráljuk a generált monitor komponenssel. Az Xtext keretrendszer a specifikált DSL (Domain Specific Language) nyelvhez generál egy Maven plugin-t. Ezt a plugin-t betölthetjük egy egyszerű maven projektbe és használhatjuk is az elkészített DSL nyelvünket, azaz létrehozhatunk a projektben a saját DSL-ünkhöz tartozó fájlokat, melyekben megadhatjuk saját scenario-inkat.



6.1. ábra. Integrációs tesztelés folyamatábrája.

A 9.2. ábrán megtekinthető az integrációs tesztelés tervének folyamatábrája. A következők tesztelési kategoriáink:

- Scenario operátorok tesztelése
- Üzenet paraméterek tesztelése
- Időzítések tesztelése

A scenario operátorok tesztelésénél az a célunk, hogy a monitor a követelmény különböző ágait figyelembe véve helyes kimenetet adjon. Például loop operátor esetén a minimum, köztes és maximum üzenet szekvencia ismétléseknél is helyes legyen a monitor kimenete. Ha a maximumnál többször szerepel az üzenet szekvencia akkor hibát kell, hogy jelezen. Üzenet paraméterek tesztelése esetén azt szeretnénk vizsgálni, hogy a monitor helyesen értelmezi e az üzenet paramétereket. Az időzítések tesztelésénél az a fontos, hogy a monitor képes-e az óraváltozók alapján az időzitési feltételeket kiértékelni. Például ha az üzenet a feltétel alapján időben érkezik meg akkor helyes kimenetet adjon vissza ha feltétel szerint később érkezik meg akkor a monitornak hibát kell jeleznie.

Egy integrációs teszt akkor sikeres ha a monitor kimenete megegyezik az elvárt kimenettel. Az elvárt kimenetet a rendszer manuális tesztelése határoz meg. A tesztelő dönti el, hogy helyesen kell hogy működjön vagy sem amit a monitornak jeleznie kell.

Az Xtext keretrendszer a definiált DSL nyelvünkhöz generál egy Maven projekt architektúrát. A nyelvünk így elérhető maven plugin formájában is, amit felhasználhatunk az integrációs teszteinkhez. Elég csupán egy maven projektet felkonfigurálni a saját dsl plugin-ünkkel és elkészítethetjük a saját tesztelési keretrendszerünket. Ezek a maven projektek a szülő projektünkben helyezkedhetnek el, így a projekt struktúrában közvetlen a nyelvünk mellett vannak. A 9.3. ábrán és a 9.1. kódrészléten látható egy ilyen integrációs teszthez tartozó maven projekt felépítése és a hozzá tartozó teszteset.

6.2. ábra. Példa integrációs teszt projekt struktúrája.

A 9.3. ábrán lévő *generated* csomag tartalmazza a scenariohoz tartozó generált automata forráskódját és a monitor forráskódját. A *hu.bme.mit.dipterv.text.example* csomag a tesztelt rendszer *Java* implementációját tartalmazza.

A 9.2. kódrészlet a Maven teszt kimenetét tartalmazza.

Ezt a projekt struktúrát felhasználva a teszteink köré tudunk egy Maven alapú Continuous Integration-t (CI) állítani.

A függelékben található egy példa unit teszt eset az automata generátor tesztelésére.

```
package hu.bme.mit.dipterv.text.example;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

import generated.Specification;
import generated.IClock;
import generated.IMonitor;
import generated.Monitor;
import generated.Clock;

public class MonitorPassingTest {

  @Test
  public void testMonitorPassing() {
    Specification specification = new Specification();
    specification.listAutomatas();
    IClock clock = new Clock();
    IMonitor monitor = new Monitor(specification.getAutomata().get(0), clock);

    Server server = new Server(monitor);
    Computer computer = new Computer(server, monitor);
    Assertions.assertTrue(monitor.goodStateReached());
  }
}
```

6.1. kódrészlet. Integrációs teszteset.

```
-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running hu.bme.mit.dipterv.text.example.MonitorPassingTest
q0 NORMAL
q1 NORMAL
q2 ACCEPT
q3 NORMAL
q4 NORMAL
q5 FINAL
!(computer.checkEmail().computer) q0->q0
computer.checkEmail().computer q0->q1
!(computer.sendUnsentEmail().server) q1->q1
!(computer.sendUnsentEmail().server) q1->q2
computer.sendUnsentEmail().server q1->q3
!(computer.logout().server) & !(computer.newEmail().server) q3->q3
computer.newEmail().server q3->q4
!(computer.downloadEmail().server) q4->q4
computer.downloadEmail().server q4->q5
Received Message: computer.checkEmail().computer
Transition: !(computer.checkEmail().computer)
Transition: computer.checkEmail().computer
transition triggered: computer.checkEmail().computer
q1

Received Message: computer.sendUnsentEmail().server
Transition: !(computer.sendUnsentEmail().server)
Transition: !(computer.sendUnsentEmail().server)
Transition: computer.sendUnsentEmail().server
transition triggered: computer.sendUnsentEmail().server
q3

Received Message: computer.newEmail().server
Transition: !(computer.logout().server) & !(computer.newEmail().server)
Transition: computer.newEmail().server
transition triggered: computer.newEmail().server
q4

Received Message: computer.downloadEmail().server
Transition: !(computer.downloadEmail().server)
Transition: computer.downloadEmail().server
transition triggered: computer.downloadEmail().server
q5

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.025 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

6.2. kódrészlet. Integrációs teszteset eredménye.

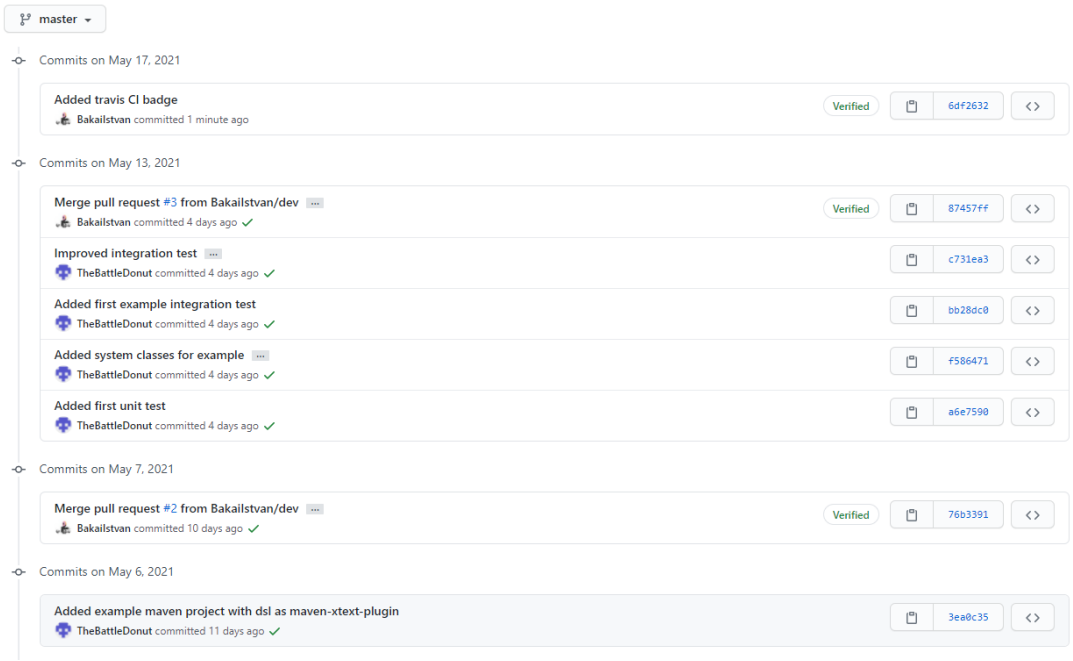
6.3. Continuous Integration

6.3.1. Github Actions CI

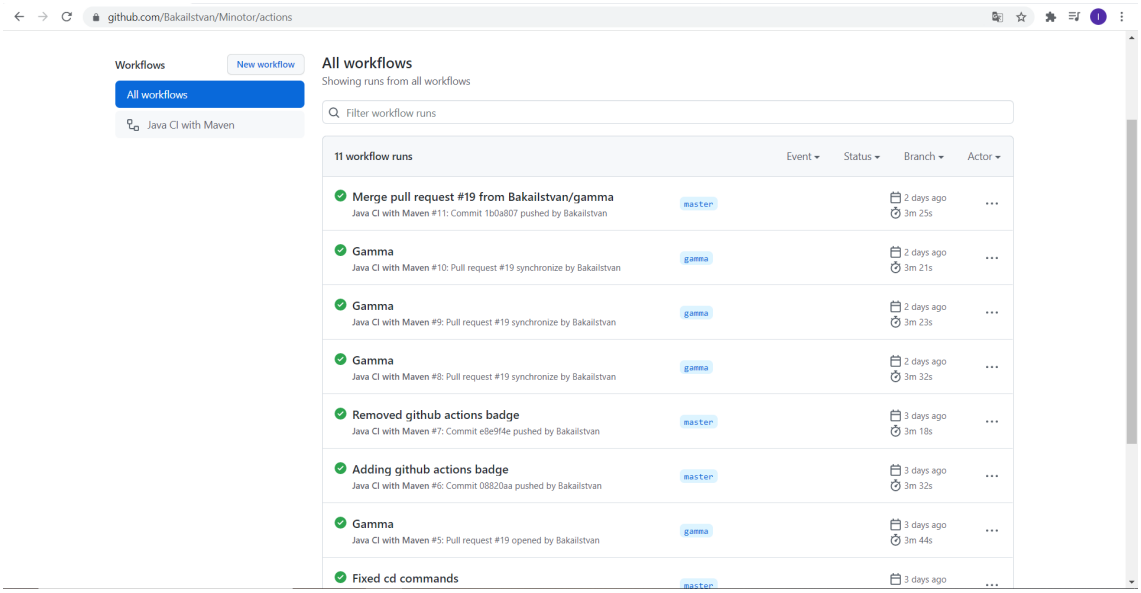
Az időzített automata és monitor forráskód generátorok automatikus tesztelése a Github Actions segítségével történik. A CI minden feltöltött új commit esetén lefut. A CI különböző fázisai a következők:

- I. Teljes Xtext projekt fordítása (Maven)
- II. Intergrációs tesztek futtatása

A generátorokhoz tartozó Xtext projekten lefut egy Maven build, amely az új feltöltött verzió tartalmazza. Ez a build állítja elő a DSL nyelvhez tartozó Maven plugin-t is. A fordítással együtt lefutnak a unit tesztek is. Ezt követően hajtódnak végre az integrációs tesztek, amelyek a frissen fordított Maven plugint használják. Ha az összes fázis sikeresen lefutott, akkor az adott commit nem rontott el semmilyen korábbi funkciót. A CI-hoz tartozó script-et a 9.3. kódrészlet tartalmazza.



6.3. ábra. GitHub repository commit-ok és hozzá tartozó CI check-ek.



6.4. ábra. Github Actions CI build-ek eredményei.

```
# This workflow will build a Java project with Maven, and cache/restore any dependencies to improve
     the workflow execution time
# For more information see: https://help.github.com/actions/language-and-framework-guides/building-
     and-testing-java-with-maven

name: Java CI with Maven

on:
    push:
    branches: [ master ]
    pull_request:
    branches: [ master ]

jobs:
    build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
        java-version: '11'
        distribution: 'adopt'
        cache: maven
    - name: Build with Maven
        run: mvn clean install -U
    - name: Test example project
        run: cd hu.bme.mit.dipterv.text.example; mvn clean install -U
    - name: Test mobileexample project
        run: cd hu.bme.mit.dipterv.text.mobileexample; mvn clean install -U
    - name: Test altexample project
        run: cd hu.bme.mit.dipterv.text.altexample; mvn clean install -U
    - name: Test parexample project
        run: cd hu.bme.mit.dipterv.text.parexample; mvn clean install -U
    - name: Test operatorexample project
        run: cd hu.bme.mit.dipterv.text.operatorexample; mvn clean install -U
    - name: Test gamma integration project
        run: cd hu.bme.mit.dipterv.text.gammaexample; mvn clean install -U
```

6.3. kódrészlet. Github Actions CI-hoz tartozó .yml script.

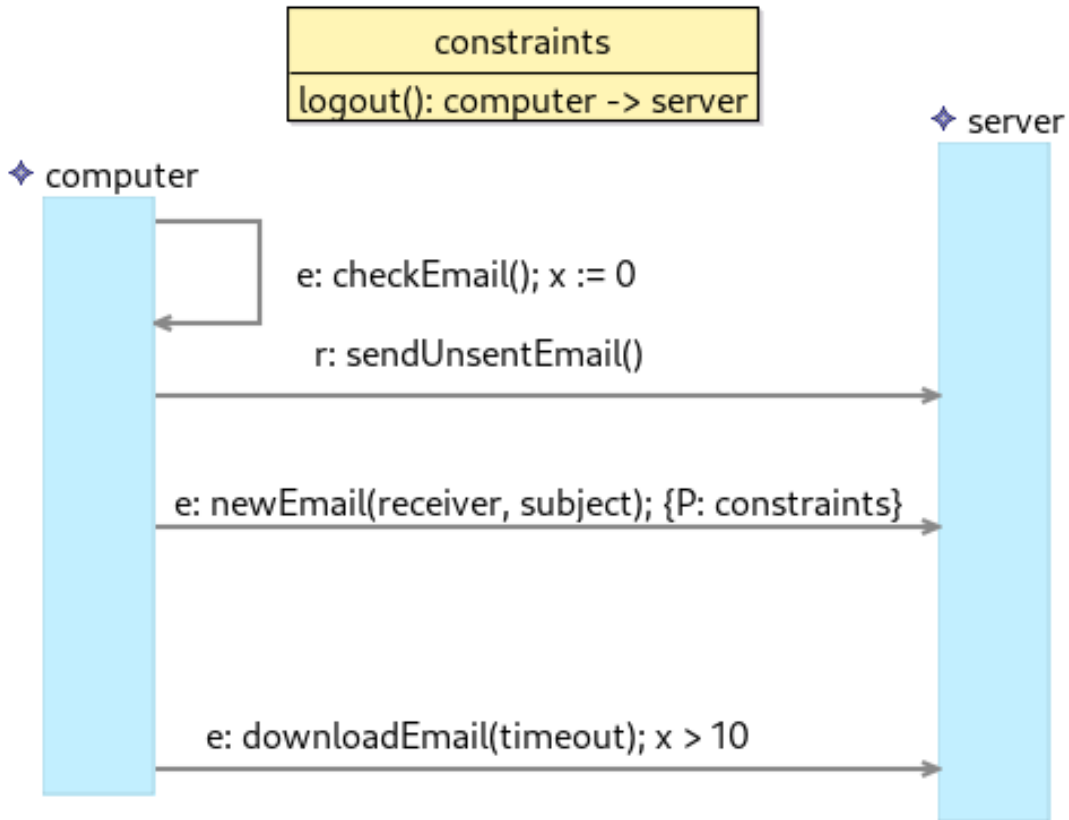
6.4. Tesztesetek

6.4.1. Egyszerű időzítési megkötéseket tartalmazó tesztszenárió

A tesztesetekhez tartozó scenario követelmény megtalálható az x. ábrán.

```
specification Email {

  object Computer computer;
  object Server server;

  integer timeout = 10;
  string receiver = "John";
  string subject = "Next meeting";

  clock x;

  constraint constraints {
    message logout() computer -> server;
  }

  scenario sendEmail{
    message checkEmail() computer -> computer reset x;
    required message sendUnsentEmail() computer -> server;
    pastConstraint {constraints} message newEmail(receiver, subject) computer -> server;
    message downloadEmail(timeout) computer -> server clockConstraint {>(x,10)};
  }
}
```

6.4. kódrészlet. Integrációs teszteset.



6.5. ábra. Szenárió diagram.

A rendszerben egy szervergépből és egy felhasználói számitogépből áll. A szerver egy e-mail szervert szimulál, aminek a számitogép különböző kéréseket küldhet. Például lekérdezheti tőle a kapott e-mail vagy új e-mail küldhet. A követelményben leírjuk, hogy a rendszernek mi a helyes viselkedése e-mail küldés esetén. Ha a computer a *checkEmail* hívást használva talál elküldendő e-mail az továbbítja a szervernek. Ezt az *elvárt sendUnsentEmail* üzenet jelzi. Ezt követően meg kell jelenjen a rendszer működésében a *newEmail* üzenet. A ehelyett *logout* üzenet érkezik a hibás működést jelent. A *newEmail* üzenetet a *downloadEmail* üzenet követi. Ezen az üzeneten van egy 10 másodperces időzítési feltétel, ami a letöltést szimulálja.

A szenárióhoz tartozó tesztesetek a következők:

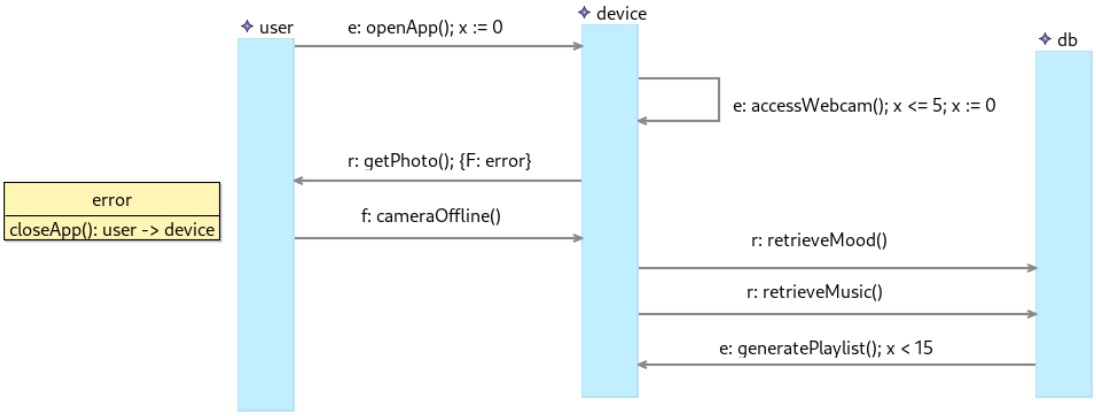
- testNetworkRequirementSatisfied
- testNetworkNoErrors
- testNetworkWithErrors
- testNetworkWithNoDelay
- testNetworkFirstFail
- testNetworkSecondFail

A *testNetworkRequirementSatisfied* tesztesetben a rendszer helyes működését szimuláljuk és azt ellenőrizzük, hogy a generált monitor képes ezt érzékelni és jelzi. A *testNetworkNoErrors* azt vizsgálja, hogy a monitor képes-e érzékelni, hogy a rendszer nem felelt meg a követelménynek. Itt úgy manipuláljuk a teszt rendszer, hogy lehagyjuk a letöltés részt a működésből. Ilyenkor a rendszer nem felel meg a követelménynek, viszont még jó állapotban marad, mert nem történt hiba. A *testNetworkWithErrors* tesztesetnél a *sendUnsentEmail* üzenet után egy *logout* üzenetet küldünk a monitor és azt vizsgáljuk képes-e detektálni ezt a hibát. A *testNetworkWithNoDelay*-nél pedig túl gyorsan küldjük a működés végén a *downloadEmail* üzenetet és azt ellenőrizzük képes e a monitor ezt a hibát érzékelni.

6.4.2. Többféle üzenetet és megkötést tartalmazó egyszerű tesztszenárió

```
specification Photo{

    object User user;
    object Device device;
    object Database db;

    clock x;

    constraint error {
        message closeApp() user -> device;
    }

    scenario playlist_generation{
        message openApp() user -> device reset x;
        message accessWebcam() device -> device clockConstraint {<=(x, 5)} reset x;
        required futureConstraint {error} message getPhoto() device -> user;
        fail message cameraOffline() user -> device;
        required strict message retrieveMood() device -> db;
        required message retrieveMusic() device -> db;
        strict message generatePlaylist() db -> device clockConstraint {<(x, 15)};
    }
}
```

6.5. kódrészlet. Integrációs teszteset.



6.6. ábra. Szenárió diagram.

A tesztrendszerünk egy zene lista generáló alkalmazás, ami egy felhasználót, mobileszközt és adatbázist tartalmaz. A felhasználó ”kedve” alapján generálja a listát, amit az arc kifejezése alapján határoz meg. A követelményben a rendszer alap működése van leírva egészen az elejétől, amikor a felhasználó megnyítja az alkalmazást. A követelmény leírás megtekinthető az x. ábrán.

A szenárióhoz tartozó tesztesetek a következők:

- testMobileRequirementSatisfied
- testMobileFutureConstraint
- testMobileFutureConstraintEarly
- testMobileWithError
- testMobileWithDelay

- testMobileWithTooMuchDelay
- testMobileMissingRequiredMessage
- testMobileRequiredEventually
- testMobileRequiredNotReceived

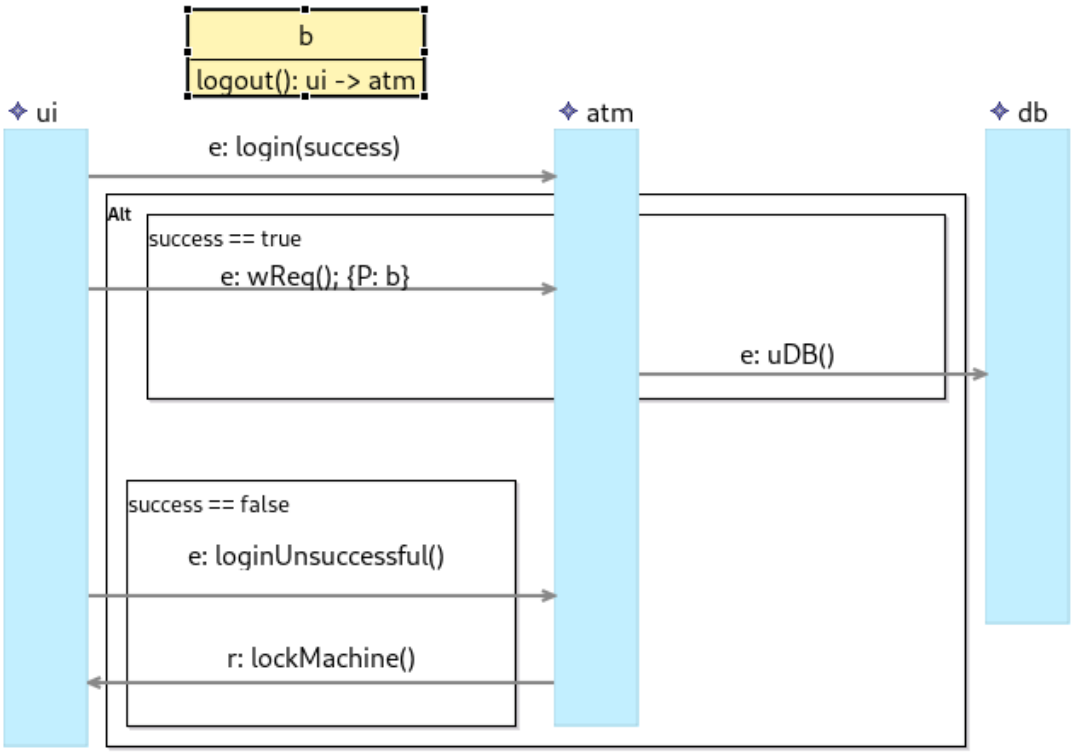
A teszteseteket a monitor hiba detektáló képeségét tesztelik. A *testMobileWithDelay* és *testMobileWithTooMuchDelay* tesztesetek az $x <= 5$ időzítési feltétel beteljesülését ellenőrzik. A *testMobileWithDelay* 5 másodperces késleltetéssel küldjük az *accessWebcam* üzenetet, míg a másiknál 6 másodperces késleltetéssel. Az első esetben a monitornak helyes működést kell érzékelnie a következőben pedig hibás működést.

6.4.3. Alt operátort tartalmazó tesztszenárió

A tesztszenárióhoz tartozó rendszerünk egy banki rendszer. A szenárió megtalálható az x. ábrán.

```
specification Bank {

    object UserInterface ui;
    object ATM atm;
    object BankDB db;

    bool success = true;

    constraint b {
        message logout() ui->atm;
    }

    scenario transaction {
        message login(success) ui->atm;

        alt (equals(success, true)) {
            pastConstraint {b} message wReq() ui->atm;
            message uDB() atm->db;
        } (equals(success, false)) {
            message loginUnsuccessful() ui->atm;
            required message lockMachine() atm->ui;
        }
    }
}
```

6.6. kódrészlet. Integrációs teszteset.



6.7. ábra. Szenárió diagram.

A rendszer egy felhasználói felületből, ATM-ből és banki adatbázisból áll. A követelményünkben két lehetséges működést írunk le. A *success* paraméter jelzi, hogy melyik működés a helyes. A szenárióhoz tartozó tesztesetek a következők:

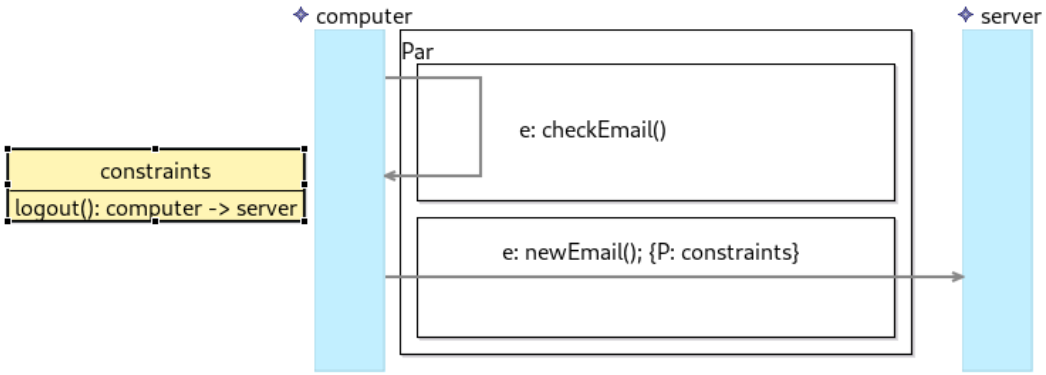
- testBankMonitorPassing
- testBankMonitorFailing
- testBankMonitorFalseCasePassing
- testBankMonitorFalseCaseFailing

A tesztesetekkel azt vizsgáljuk, hogy a monitor képes-e a helytelen ág lefutását hibának érzékelni és a helyes viselkedés esetén detektálni a követelmény teljesítését.

6.4.4. Par operátort tartalmazó tesztszenárió

```
specification Email {
    object Computer computer;
    object Server server;

    constraint constraints{
        message logout() computer -> server;
    }

    scenario email {
        par {
            case checkEmail {
                message checkEmail() computer -> computer;
            }

            case newEmail {
                pastConstraint {constraints} message newEmail() computer -> server;
            }
        }
    }
}
```

6.7. kódrészlet. Integrációs teszteset.



6.8. ábra. Szenárió diagram.

Ehhez a tesztszenárióhoz az első szenárióban lévő teszt rendszert használtuk fel. A szenárióhoz tartozó tesztesetek a következők:

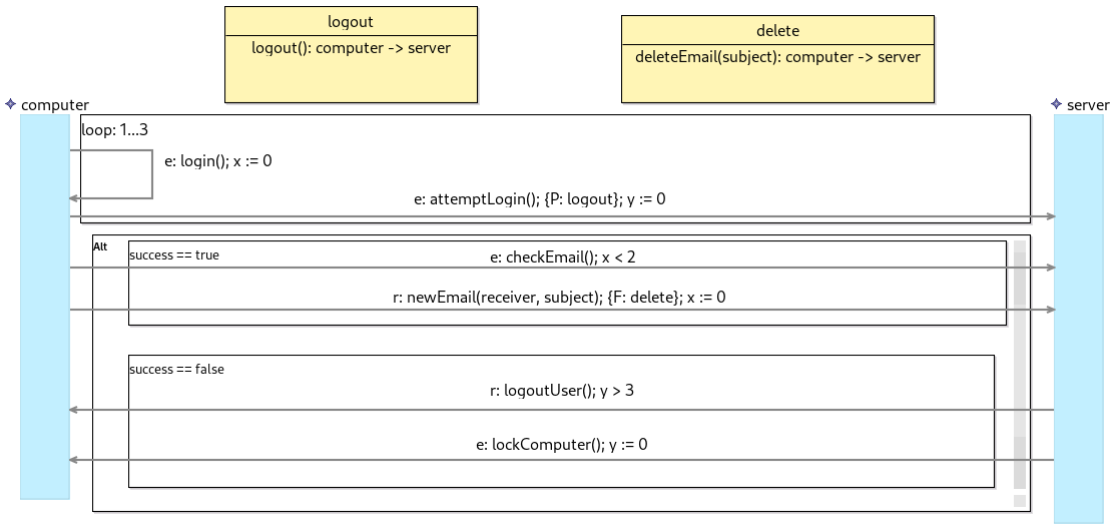
- testNetworkRequirementSatisfied
- testNetworkOtherRequirementSatisfied

Azt vizsgáljuk, hogy a monitor képes mindkét permutáció esetén érzékelni a követelmény teljesítését.

6.4.5. Komplex tesztszenárió loop és alt operátorokkal

```
specification Connection {

    object Computer computer;
    object Server server;

    string receiver = "John";
    string subject = "Next Meeting";
    bool success = false;

    clock x;
    clock y;

    constraint logout {
        message logout() computer -> server;
    }

    constraint delete {
        message deleteEmail(subject) computer -> server;
    }

    scenario authentication {
        loop (1, 3) {
            message login() computer -> computer reset x;
            pastConstraint {logout} message attemptLogin() computer -> server reset y;
        }

        alt (equals(success, true)) {
            message checkEmail() computer -> server clockConstraint {<(x, 2)};
            required futureConstraint {delete} message newEmail(receiver, subject) computer ->
     server reset x;
        } (equals(success, false)) {
            required message logoutUser() server -> computer clockConstraint {>(y, 3)};
            message lockComputer() server -> computer reset y;
        }
    }
}
```

6.8. kódrészlet. Integrációs teszteset.



6.9. ábra. Szenárió diagram.

- testNetworkRequirementSatisfied

- testNetworkRequirementSatisfiedTwice
- testNetworkRequirementSatisfiedThreeTimes
- testNetworkRequirementSatisfiedFourTimes
- testNetworkAltTrueCase
- testNetworkAltTrueCaseSatisfied
- testNetworkLogoutTooFast
- testNetworkLogoutConstrait

6.5. Tesztelés összefoglaló

Tesztelési célok	**Egyszerű tesztszenárió**	**Több üzenetet tartalmazó tesztszenárió**	**Alt operátort tartalmazó tesztszenárió**	**Par operátort tartalmazó tesztszenárió**	**Komplex tesztszenárió**
Egyszerű üzenet megjelenése	X	X	X	X	X
Elvárt üzenet megjelenése	X	X	X	-	X
Nem kívánt (fail) üzenet megjelenése	X	X	-	-	-
Strict üzenet tesztelése	-	X	-	-	-
Időzítési feltételek tesztelése	X	X	-	-	X
Past megkötés tesztelése	X	-	X	X	X
Future megkötés tesztelése	-	X	-	-	X
Alt operátor tesztelése	-	-	X	-	X
Par operátor tesztelése	-	-	-	X	-
Loop operátor tesztelése	-	-	-	-	X
Több operátort tartalmazó szenárió tesztelése	-	-	-	-	X
Egymást követő elvárt üzenetek	-	X	-	-	-
Egymást követő fail üzenetek	X	-	-	-	-
Sima üzenet tesztelése	X	X	X	X	X
Több óraváltozó	-	-	-	-	X

Elvárt után fail üzenet	X	-	-	-	-
Fail után elvárt üzenet	-	X	-	-	-

6.1. táblázat. Összefoglaló táblázat

7. fejezet

A monitor integrálása a Gamma keretrendszerben tervezett komponensekkel

7.1. Gamma keretrendszer

A *Gamma* keretrendszerrel komponensalapú reaktív rendszereket lehet tervezni, és az a célja hogy támogassa az elosztott rendszerek modelallapú fejlesztését. A létrehozott rendszert a keretrendszerrel lehet tesztelni, ellenőrizni és szimulálni is. A keretrendszernek a *Yakindu* modellező eszköz használja, amit kiegészít egy modellező réteggel ahol meg valósítható a komponensek közti kommunikációs hálózat. A komponensek hierarchikusan helyezkednek el egymás mellett, ami megkönnyíti egyes komponensek többszöri felhasználását. A tervezett rendszerhez a keretrendszer képes Java forráskódot is generálni. A modelhez generálhatoak tesztesetek vagy akár verifikálható az UPPAAL model ellenőrző eszköz használatával.

A monitor illesztéséhez a *Gamma tutorial* csomagjában lévő rendszert használtam. Ez egy irányitórendszer modelje, ami egy kereszteződésben lévő közlekedési lámpák működtetésért felel. A jelző lámpák általános három fokozatú lámpák és a piros-zöld-sárga-piros jelzéseket ismétlik. A rendszer támogat még egy megszakító állapotot, amit a rendőrség kapcsolhat be. Ilyenkor minden lámpa sárgán villog.

7.2. Generált monitor integrációja

A keretrendszer a *gamma.monitor* csomagba generálja a beépített monitor forráskódja. Ezt a *Gamma* monitort cseréltem le a saját a monitor komponensemhez tartozó forráskóddal megvalósítva a szükséges interfészeket.

```
public void runComponent() {
    Queue<Event> eventQueue = getProcessQueue();
    while (!eventQueue.isEmpty()) {
            Event event = eventQueue.remove();
            switch (event.getEvent()) {
                case "LightInputs.DisplayNone":
                    update("controller", "light", "displayNone", new HashMap<String, Object>());
                break;
                case "LightInputs.DisplayYellow":
                    update("controller", "light", "displayYellow", new HashMap<String, Object>());
                break;
                case "LightInputs.DisplayRed":
                    update("controller", "light", "displayRed", new HashMap<String, Object>());
                break;
                case "LightInputs.DisplayGreen":
                    update("controller", "light", "displayGreen", new HashMap<String, Object>());
                break;
                default:
                    throw new IllegalArgumentException("No such event!");
            }
    }
    notifyListeners();
}
```

7.1. kódrészlet. Monitor komponenshez tartozó kódrészlet.

A *Gamma* rendszer és a monitor komponens közti kommunikáció megvalósítása a 7.1-es kódrészletben látható. Ez a rendszer eseményeit továbbítja a generált monitornak a megfelelő alakra alakítva.

```
specification Light{

    object TrafficLight light;
    object Controller controller;

    scenario sendEmail{
        message displayRed() controller -> light;
        fail message displayRed() controller -> light;
    }
}
```

7.2. kódrészlet. Szenárió szöveges leírása.

A 7.2-es kódrészlet tartalmazza a követelmény szcenáriót. Elvárjuk, hogy ha pirosan világított a lámpa akkor a következő állapotta a lámpának ne legyen megint piros. Ezt egy *fail* üzenet segítségével tudjuk elérni.

A 7.3-as kódrészlet a monitor kimenetét. A monitor helyes működést érzékelt, a rendszer jó állapotban van és megfelelt a követelménynek.

```
Received Message: controller.displayRed().light
[Monitor] available transition are:
-----------------------------------
controller.displayRed().light, q0->q1, ,
!(controller.displayRed().light), q0->q0, ,
=-----------------------------------=
Transition: controller.displayRed().light, q0->q1, ,
transition triggered: controller.displayRed().light, q0->q1, ,
q1
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayGreen().light
[Monitor] available transition are:
-----------------------------------
!(controller.displayRed().light), q1->q3, ,
controller.displayRed().light, q1->q2, ,
!(controller.displayRed().light), q1->q1, ,
=-----------------------------------=
Transition: !(controller.displayRed().light), q1->q3, ,
transition triggered: !(controller.displayRed().light), q1->q3, ,
q3
[GammaMonitorTest] Received status from monitor: System is in good state.
[GammaMonitorTest] Received status from monitor: Requirement satisfied
[GammaMonitorTest] Monitor reported that the requirement was satisfied
```

7.3. kódrészlet. Monitor kimenete.

8. fejezet

Összefoglalás

A célként kitűzött scenario alapú monitor generátor kibővítése részlegesen sikerült.

A szöveges scenario leíró nyelv támogatja TPSC diagramok specifikálását. Az automata generátor támogatja a TPSC tulajdonságokhoz tartozó minta automaták generálását és képes az üzenet paramétereket is értelmezni. A generátor támogatja az alt, loop és par operátorokat tartalmazó TPSC-khez tartozó időzített automaták generálását.

A generátor a scenario-hoz tartozó automata alapján képes egy monitor forráskódjának generálására. Legenerálja a megfelelő interfészeket amik a monitor és rendszer közti kommunikációhoz szükségesek. Ha az üzenetek megfigyeléséhez szükséges segédfüggvényeket a kommunikációs infrastruktúrához megvalósítják, akkor a monitor képes a rendszer viselkedésének ellenőrzésére.

A monitor forráskód generátor a par, loop és alt operátorokat támogatja. Továbbá az időzítési feltételeket tartalmazó üzeneteket is tudja értelmezni.

A további feladatok közzé tartozik a tesztek megírása és a teszt környezet véglegesítése. Továbbá meg kell valósítani a szöveges scenario leírások vizualizációját és a monitor komponenst illeszteni a Gamma keretrendszerhez.

9. fejezet

Források

1 M.Autili – P. Inverardi – P.Pelliccione, Graphical scenarios for specifying temporal properties: an automated approach in Automated Software Engineering 14(3):293-340, September 2007, https://link.springer.com/article/10.1007%2Fs10515-007-0012-6

2 Pengcheng Zhang - Hareton Leung, Web services property sequence chart monitor: A tool chain for monitoring BPEL – based web service composition with scenario-based specifications in IET Software 7(4):222-248, August 2013

3 Xtext, https://www.eclipse.org/Xtext/

4 Xtend, https://www.eclipse.org/xtend/

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.

Függelék

F.1. A 8.3. fejezet minta példájához tartozó Specification osztály

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Collections;
import java.util.Comparator;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class Specification{
  private String id = "spec1";
  private ArrayList<Automaton> automatas;

  public Specification(){
    automatas = new ArrayList<Automaton>();
    String str;
    String str1;
    String pre;
    String succ;
    State actualState;
    State acceptState;
    State finalState;
    State newState;
    State acceptState_new;
    Automaton a = new Automaton("playlist_generation");
    Automaton b;
    Map<String, Automaton> altauto;
    ArrayList<Automaton> parauto;
    Automaton loopauto;
    Automaton expression;
    int counter = 0;

    b = new Automaton("auto7");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);

    b.addTransition(new Transition("!(" + "user" + "." +
      "openApp" + "("
      + ")"

      + "." + "device)", actualState, actualState));

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new Transition("user" + "." +

    "openApp" + "("
```

```
    + ")"

    + "." + "device" , actualState, newState));
    b.addState(newState);
    b.setFinale(newState);
    a.collapse(b);

    b = new Automaton("auto7");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);

    b.addTransition(new Transition("!(" + "device" + "." +
      "accessWebcam" + "("
      + ")"

      + "." + "device)", actualState, actualState));

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new Transition("device" + "." +

    "accessWebcam" + "("
    + ")"

    + "." + "device" , actualState, newState));
    b.addState(newState);
    b.setFinale(newState);
    a.collapse(b);

    b = new Automaton("auto3");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);

    b.addTransition(new Transition("!(" + "device" + "." +
    "getPhoto" + "("
    + ")"

    + "." + "user" + ")", actualState, actualState));

    acceptState = new State("q" + counter, StateType.ACCEPT);
    counter++;

    b.addTransition(new Transition("!(" + "device" + "." +
      "getPhoto" + "("
      + ")"

      + "." + "user" + ")", actualState, acceptState));

    b.addState(acceptState);

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new Transition("device" + "." +
    "getPhoto" + "("
    + ")"
    + "." + "user", actualState, newState));
    b.addState(newState);
    b.setFinale(newState);
    a.collapse(b);

    b = new Automaton("auto5");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);


    finalState = new State("q" + counter, StateType.FINAL);
```

```
    counter++;
    b.addState(finalState);
    b.setFinale(finalState);

    b.addTransition(new Transition("!(" + "user" + "." +
          "cameraOffline" + "("
          + ")"
          + "." + "device)", actualState, finalState));

    b.addTransition(new Transition("!(" + "user" + "." +
      "cameraOffline" + "("
      + ")"
      + "." + "device)", actualState, actualState));

    newState = new State("q" + counter, StateType.ACCEPT);
    counter++;
    b.addTransition(new Transition("user" + "." +
    "cameraOffline" + "("
    + ")"
    + "." + "device" , actualState, newState));
    b.addState(newState);
    a.collapse(b);


    b = new Automaton("auto9");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);

    finalState = new State("q" + counter, StateType.FINAL);
    counter++;
    acceptState = new State("q" + counter, StateType.ACCEPT);
    counter++;
    b.addTransition(new Transition("device" + "." +
    "retrieveMood" + "("
    + ")"
    + "." + "db" , actualState, finalState));
    b.addTransition(new Transition("!(" + "device" + "." +
    "retrieveMood" + "("
    + ")"
    + "." + "db" + ")", actualState, acceptState));
    b.addState(acceptState);
    b.addState(finalState);
    b.setFinale(finalState);
    a.collapse(b);
    b = new Automaton("auto3");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);

    b.addTransition(new Transition("!(" + "device" + "." +
    "retrieveMusic" + "("
    + ")"

    + "." + "db" + ")", actualState, actualState));

    acceptState = new State("q" + counter, StateType.ACCEPT);
    counter++;

    b.addTransition(new Transition("!(" + "device" + "." +
      "retrieveMusic" + "("
      + ")"

      + "." + "db" + ")", actualState, acceptState));

    b.addState(acceptState);

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new Transition("device" + "." +
```

```
  "retrieveMusic" + "("
  + ")"
  + "." + "db", actualState, newState));
  b.addState(newState);
  b.setFinale(newState);
  a.collapse(b);


  b = new Automaton("auto12");
  actualState = new State("q" + counter, StateType.NORMAL);
  counter++;
  b.addState(actualState);
  b.setInitial(actualState);

  newState = new State("q" + counter, StateType.FINAL);
  counter++;
  b.addTransition(new Transition("db" + "." +
  "generatePlaylist" + "("
  + ")"
  + "." + "device", actualState, newState));
  b.addState(newState);
  b.setFinale(newState);
  a.collapse(b);
  a.rename();
  automatas.add(a);
}

public void listAutomatas(){
  for(Automaton a : this.automatas){
    for(State s : a.getStates()){
      s.writeState();
    }

    for(Transition t : a.getTransitions()){
      t.writeTransition();
    }
  }
}

public List<Automaton> getAutomata() {
  return automatas;
}

public ArrayList<Automaton> par(ArrayList<Automaton> automatas) {
        ArrayList<ArrayList<Automaton>> automataList = new ArrayList<>();
        permute(automataList, new ArrayList<>(), automatas);
        return listConverter((automataList));
}

  private void permute(ArrayList<ArrayList<Automaton>> list, ArrayList<Automaton> resultList,
   ArrayList<Automaton> automatas) {
      if (resultList.size() == automatas.size()) {
          list.add(new ArrayList<>(resultList));
      } else {
          for (int i = 0; i < automatas.size(); i++) {
              if (resultList.contains((automatas.get(i)))) {
                  continue;
              }

              resultList.add(automatas.get(i));
              permute(list, resultList, automatas);
              resultList.remove(resultList.size() - 1);
          }
      }
  }

  private ArrayList<Automaton> listConverter(ArrayList<ArrayList<Automaton>> list) {
      ArrayList<Automaton> result = new ArrayList<>();
      for (ArrayList<Automaton> alist : list) {
          Automaton newauto = new Automaton("listConverter");
          for (Automaton auto : alist) {
              newauto.collapse(copyAutomaton(auto));
```

```
            }
            result.add(newauto);
        }
        return result;
    }

    public Map<String, Automaton> loopSetup(Automaton loopauto, int min, int max) {
            Map<String, Automaton> result = new HashMap<>();

              for (int i = min; i <= max; i++) {
                  Automaton newauto = new Automaton("loopauto" + i);
                  for (int j = 0; j < i; j++) {
                      newauto.collapse(copyAutomaton(loopauto));
                  }
                  result.put("loop" + i, newauto);
              }
              return result;
          }

    public Automaton copyAutomaton(Automaton referenceAuto) {
            Automaton result = new Automaton("copy automaton");
            int count = 0;
            State previousSender = new State();
            State referencePreviousSender = new State();

            for (Transition t : referenceAuto.getTransitions()) {
                State sender = new State();
                State receiver = new State();
                Transition transition = new Transition();
                Automaton temp = new Automaton("temp");

                transition.setId(t.getId());

                if (t.getSender() == referencePreviousSender) {
                    receiver.setId("c" + count);
                    count++;
                    receiver.setType(t.getReceiver().getType());

                    transition.setSender(previousSender);
                    transition.setReceiver(receiver);
                    temp.addState(previousSender);
                    temp.addState(receiver);
                    temp.setInitial(previousSender);
                    temp.setFinale(receiver);
                } else {
                    if (t.getSender() == t.getReceiver()) {
                        sender.setId("c" + count);
                        count++;
                        sender.setType(t.getSender().getType());

                        transition.setSender(sender);
                        transition.setReceiver(sender);

                        temp.addState(sender);
                        temp.setInitial(sender);
                        temp.setFinale(sender);
                    } else {
                        sender.setId("c" + count);
                        count++;
                        sender.setType(t.getSender().getType());

                        receiver.setId("c" + count);
                        count++;
                        receiver.setType(t.getReceiver().getType());

                        transition.setSender(sender);
                        transition.setReceiver(receiver);

                        temp.addState(sender);
                        temp.addState(receiver);
                        temp.setInitial(sender);
                        temp.setFinale(receiver);
```

```
                    }
                    previousSender = sender;
                    referencePreviousSender = t.getSender();
                }

                temp.addTransition(transition);
                result.collapse(temp);
            }

            return result;
        }

  public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException{
    Specification specification = new Specification();
    specification.listAutomatas();
    boolean acceptState = false;
}
```

F.1.1. kódrészlet. Specification osztály.

F.2. Monitor forráskód generátor - operátorok támogatása

```
public class Main {
  public static void monitorStatus(String status) {
    System.out.println(status);
  }

  public static void main(String[] args) {
    Specification specification = new Specification();
    specification.listAutomatas();
    IMonitor monitor = new Monitor(specification.getAutomata().get(0));

    UserInterface ui = new UserInterface();
    ATM atm = new ATM();
    BankDB db = new BankDB();
    ui.atm = atm;
    atm.ui = ui;
    atm.db = db;
    ui.monitor = monitor;
    atm.monitor = monitor;
    db.monitor = monitor;

    ui.start();
  }
}
```

F.2.1. kódrészlet. 7.1. scenariohoz tartozó Main osztály.

```
public class ATM {
  public IMonitor monitor;
  public BankDB db;
  public UserInterface ui;

  public void logout() {
    monitor.update("ui", "atm", "logout", new String[] {});
  }

  public void login(boolean success) {
    monitor.update("ui", "atm", "login", new String[] {"success"});
    success = true;
  }

  public void wReq() {
    monitor.update("ui", "atm", "wReq", new String[] {});
    db.uDB();
  }

  public void loginUnsuccessful() {
    monitor.update("ui", "atm", "loginUnsuccesful", new String[] {});
    ui.lockMachine();
  }
}
```

F.2.2. kódrészlet. 7.1. scenariohoz tartozó rendszer ATM Java osztálya.

```
Received Message: ui.login(success).atm
Transition: !(ui.login(success).atm)
Transition: ui.login(success).atm
transition: ui.login(success).atm
q1
System is in bad state.
Received Message: ui.wReq().atm
Transition: epsilon
PrevTransition: epsilon
transition: epsilon
qinit0
System is in bad state.
Transition: epsilon; success == false
PrevTransition: epsilon; success == false
transition: epsilon; success == false
q5
System is in bad state.
Transition: ui.loginUnsuccessful().atm
Transition: !(ui.loginUnsuccessful().atm)
Transition: epsilon; success == true
PrevTransition: epsilon; success == true
transition: epsilon; success == true
q2
System is in bad state.
Transition: ui.wReq().atm
transition: ui.wReq().atm
q3
System is in bad state.
Received Message: atm.uDB().db
Transition: !(atm.uDB().db)
Transition: atm.uDB().db
transition: epsilon
qfinal1
System is in good state.
```

F.2.3. kódrészlet. 7.1. scenario monitor kimenete.

```
specification spec1{

  object Computer computer;
  object Server server;

  constraint constraints{
    message logout() computer -> server;
  }

  scenario email{
    loop (1, 3) {
      message checkEmail() computer -> computer;
      message newEmail() computer -> server pastConstraint {constraints};
    }
  }
}
```

F.2.4. kódrészlet. Loop operátort tartalmazó scenario.

```
Received Message: computer.checkEmail().computer
Transition: epsilon; loop2
PrevTransition: epsilon; loop2
transition: epsilon; loop2
q0
System is in bad state.
Transition: computer.checkEmail().computer
Transition: !(computer.checkEmail().computer)
Transition: epsilon; loop3
PrevTransition: epsilon; loop3
transition: epsilon; loop3
q5
System is in bad state.
Transition: computer.checkEmail().computer
Transition: !(computer.checkEmail().computer)
Transition: epsilon; loop1
PrevTransition: epsilon; loop1
transition: epsilon; loop1
q12
System is in bad state.
Transition: computer.checkEmail().computer
transition: computer.checkEmail().computer
q13
System is in bad state.
Received Message: computer.newEmail().server
Transition: !(computer.logout().server) & !(computer.newEmail().server)
Transition: computer.newEmail().server
transition: epsilon
qfinal1
System is in good state.
```

F.2.5. kódrészlet. F.2.4. scenariohoz tartozó monitor kimenet.

```
public class Main {
  public static void monitorStatus(String status) {
    System.out.println(status);
  }

  public static void main(String[] args) {
    Specification specification = new Specification();
    specification.listAutomatas();
    IMonitor monitor = new Monitor(specification.getAutomata().get(0));

    Server server = new Server();
    Computer computer = new Computer(server, monitor);
  }
}
```

F.2.6. kódrészlet. F.2.4. scenariohoz tartozó Main osztály.

F.3. Automata generátor unit teszt

```
@Test def testSimpleMessage() {

    '''
      specification spec1{

        object Computer computer;

        scenario email {
          message checkEmail() computer -> computer;
        }
      }
    '''.assertCompilesTo('''
    MULTIPLE FILES WERE GENERATED

    File 1 : /myProject/./src-gen/Automaton.java

    package generated;

    import java.util.ArrayList;
    import java.util.Map;

    public class Automaton {
      private String id;
      private ArrayList<State> states;
      private ArrayList<Transition> transitions;
      private State initial;
      private State finale;

      public Automaton(String id) {
        this.id = id;
        this.states = new ArrayList<>();
        this.transitions = new ArrayList<>();
      }

      public void setFinale(State state){
        this.finale = state;
      }

      public State getFinale(){
        return this.finale;
      }

      public void setInitial(State state){
        this.initial = state;
      }

      public State getInitial(){
        return this.initial;
      }

      public ArrayList<State> getStates(){
        return states;
      }

      public ArrayList<Transition> getTransitions(){
        return transitions;
      }

      public String getId() {
        return id;
      }

      public void setId(String id) {
        this.id = id;
      }

      public void addState(State newState){
        states.add(newState);
      }
```

```
 public void rename(){
    int counter = 0;
    int _counter = 0;
    for(State s : this.states) {
      if(s.getId().equals("qinit") || s.getId().equals("qfinal") || s.getId().equals("
qaccepting")){
        s.setId(s.getId() + _counter);
        _counter++;
      }else{
      s.setId("q" + counter);
      counter++;
    }
    }
 }

 public ArrayList<Transition> findSender(State state){
   ArrayList<Transition> senderTransitions = new ArrayList<Transition>();
   for(Transition t : this.transitions){
     if(t.getSender().getId().equals(state.getId()))
       senderTransitions.add(t);
   }
   return senderTransitions;
 }

 public ArrayList<Transition> findReceiver(State state){
    ArrayList<Transition> receiverTransitions = new ArrayList<Transition>();
    for(Transition t : this.transitions){
     if(t.getReceiver().getId().equals(state.getId()))
     receiverTransitions.add(t);
    }
    return receiverTransitions;
  }

 public void addTransition(Transition newTransition){
      if(transitions.isEmpty()){
        transitions.add(newTransition);
        return;
      }

      for(Transition t : transitions){
        if(t.getId().equals(newTransition.getId())
          && t.getSender().equals(newTransition.getSender())
          && t.getReceiver().equals(newTransition.getReceiver()))
            return;
      }

      transitions.add(newTransition);
  }

  public void collapse(Automaton automaton){
      if(this.states.isEmpty() && this.transitions.isEmpty()){
        for (State s : automaton.states)
          this.addState(s);


        for (Transition t : automaton.transitions)
          this.addTransition(t);

        this.initial = automaton.initial;
        this.finale = automaton.finale;

      }else {
        ArrayList<Transition> receive = findReceiver(this.finale);
        ArrayList<Transition> send = findSender(this.finale);

        for (State s : automaton.states)
          this.addState(s);


        for (Transition t : automaton.transitions)
          this.addTransition(t);
```

```
          for (Transition t : receive)
            t.setReceiver(automaton.initial);


          for (Transition t : send)
            t.setSender(automaton.initial);


          this.states.remove(finale);
          this.finale = automaton.finale;
        }
    }
    public void merge(Map<String, Automaton> automatas){

              State qinit = new State("qinit", StateType.NORMAL);
              State qfinal = new State("qfinal", StateType.FINAL);

              if(this.states.isEmpty() && this.transitions.isEmpty()){
                this.initial = qinit;
              }else{
                this.addTransition(new Transition("epsilon", this.finale, qinit));
              }

              this.addState(qinit);
              this.addState(qfinal);
              this.finale = qfinal;

              for (Map.Entry<String, Automaton> a : automatas.entrySet()) {
                for (Transition t : a.getValue().transitions)
                  this.addTransition(t);

                for (State s : a.getValue().states) {
                  this.addState(s);
                  if (s.getType().equals(StateType.FINAL))
                    this.addTransition(new Transition("epsilon", s, qfinal));
                }
                this.addTransition(new Transition("epsilon; " + a.getKey(), qinit, a.getValue().
 initial));
              }
               }
}

File 2 : /myProject/./src-gen/Clock.java

package generated;

import java.util.HashMap;
import java.util.Map;

import org.apache.commons.lang3.time.StopWatch;

public class Clock implements IClock {
  private Map<String, StopWatch> stopwatches;

  public Clock() {
    stopwatches = new HashMap<>();
  }

  @Override
  public long getClock(String clock) {
    if (stopwatches.containsKey(clock)) {
      return stopwatches.get(clock).getTime();
    }

    return -1;
  }

  @Override
  public void resetClock(String clock) {
    if (stopwatches.containsKey(clock)) {
      stopwatches.get(clock).reset();
```

```
      } else {
        StopWatch watch = new StopWatch();
        watch.start();
        stopwatches.put(clock, watch);
      }
    }

  }

  File 3 : /myProject/./src-gen/EventCreator.java

    package generated;

    public class EventCreator {
      private Monitor monitorInterface;

      public EventCreator(
        Monitor monitorInterface
      ) {
        this.monitorInterface = monitorInterface;
      }


      public void appear(String name) {
      }

      public void disappear(String name) {
      }

      public void changeTo(String event) {
      }
    }

  File 4 : /myProject/./src-gen/IClock.java

  package generated;

  public interface IClock {
    public long getClock(String clock);
    public void resetClock(String clock);
  }

  File 5 : /myProject/./src-gen/IMonitor.java

  package generated;

  public interface IMonitor {
    public boolean goodStateReached();
    public void update(String sender, String receiver, String messageType, String[] parameters);
    public boolean requirementSatisfied();
    public void errorDetected(String sender, String receiver, String messageType, String[]
   parameters);
  }

  File 6 : /myProject/./src-gen/Monitor.java

  package generated;

  import java.util.ArrayList;
  import java.util.Arrays;
  import java.util.List;
  import java.util.ListIterator;

  public class Monitor implements IMonitor {
    private Automaton automaton;
    private State actualState;
    private List<State> goodStates;
    private boolean requirementFullfilled;
    private IClock clock;

    public Monitor(Automaton automaton
           , IClock clock) {
```

```
    this.automaton = automaton;
    this.actualState = automaton.getInitial();
    this.goodStates = new ArrayList<State>();
    this.goodStates.add(automaton.getFinale());
    this.requirementFullfilled = true;
    this.clock = clock;
  }

  @Override
  public boolean goodStateReached() {
    return this.goodStates.contains(actualState) && this.requirementFullfilled;
  }

  @Override
  public void update(String sender, String receiver, String messageType, String[] parameters) {
    List<Transition> transitions = automaton.findSender(this.actualState);
    String receivedMessage = getReceivedMessage(sender, receiver, messageType, parameters);
    System.out.println("Received Message: " + receivedMessage);
    boolean edgeTriggered = false;

    ListIterator<Transition> iterator = transitions.listIterator();
    while (iterator.hasNext()) {
      Transition transition = iterator.next();
      System.out.println("Transition: " + transition.getId());
      if (transition.getId().contains("epsilon")) {
        List<Transition> newTransitions = new ArrayList<Transition>(automaton.findSender(
transition.getReceiver()));

        for (Transition t : newTransitions) {
          iterator.add(t);
          iterator.previous();
        }
        Transition prevTransition = iterator.previous();
        System.out.println("PrevTransition: " + prevTransition.getId());

        if (transitions.size() == 1) {
          this.actualState = transition.getReceiver();
          updateMonitorStatus(transition);
        }

        iterator.remove();
        if (!transitions.stream().anyMatch(t -> t.getId().contains("epsilon"))) {
          iterator = transitions.listIterator();
        }
      } else if (transitions.stream().anyMatch(t -> t.getId().contains("epsilon"))) {
        // do nothing
      } else if (transition.hasClock()) {
        if (transition.clockConditionSatisfied(clock.getClock(transition.getClock()))) {
          edgeTriggered = updateState(transition, messageType, sender, receiver, parameters,
receivedMessage);
        }
      } else {
        edgeTriggered = updateState(transition, messageType, sender, receiver, parameters,
receivedMessage);
      }

      if (edgeTriggered) {
        break;
      }
    }

    if (!edgeTriggered) {
      this.requirementFullfilled = false;
      System.out.println("Failure: receivedMessage didn't match any transitions.");
    }
  }

  private boolean updateState(Transition transition
                , String messageType
                , String sender
                , String receiver
                , String[] parameters
```

```
                , String receivedMessage) {

    if (!transition.getId().contains("&")) {
      if (!transition.getId().contains("!")
       && transition.getMessageType().equals(messageType)
       && transition.getSenderName().equals(sender)
       && transition.getReceiverName().equals(receiver)
       && Arrays.equals(transition.getParameters(), parameters)) {

        this.actualState = transition.getReceiver();
        updateMonitorStatus(transition);
        return true;
      } else if (transition.getId().contains("!")
          && (!transition.getMessageType().equals(messageType)
          || !transition.getSenderName().equals(sender)
          || !transition.getReceiverName().equals(receiver)
          || !Arrays.equals(transition.getParameters(), parameters))) {

        this.actualState = transition.getReceiver();
        updateMonitorStatus(transition);
        return true;
      }
    } else if (!transition.getId().contains(receivedMessage)) {
      this.actualState = transition.getReceiver();
      updateMonitorStatus(transition);
      return true;
    }

    return false;
  }

  private void updateMonitorStatus(Transition transition) {
    if (actualState.getType().equals(StateType.FINAL)) {
      List<Transition> transitions = automaton.findSender(this.actualState);
      if (transitions.size() == 1
       && transitions.stream().anyMatch(t -> t.getId().equals("epsilon"))
       && transitions.get(0).getReceiver().getType().equals(StateType.FINAL)) {
        transition = transitions.get(0);
        this.actualState = transition.getReceiver();
      }
    }

    System.out.println("transition triggered: " + transition.getId());
    System.out.println(actualState.getId());

    //TODO: fix Main class imports
    /*if (goodStateReached()) {
      Main.monitorStatus("System is in good state.");
    } else {
      Main.monitorStatus("System is in bad state.");
    }*/

    if (transition.hasReset()) {
      clock.resetClock(transition.getReset());
    }
  }

  private String getReceivedMessage(String sender, String receiver, String messageType, String[]
  parameters) {
    String receivedMessage = sender + "." + messageType + "(";
    for (String param : parameters) {
      receivedMessage += param;
      if (!(parameters[parameters.length - 1]).equals(param)) {
        receivedMessage += ", ";
      }
    }
    receivedMessage += ")." + receiver;

    return receivedMessage;
  }

  @Override
```

```
  public boolean requirementSatisfied() {
    return this.actualState == this.automaton.getFinale();
  }

  @Override
  public void errorDetected(String sender, String receiver, String messageType, String[]
 parameters) {
    // TODO Auto-generated method stub

  }
}
```

File 7 : /myProject/./src-gen/Specification.java

```
  package generated;

  import java.io.FileNotFoundException;
  import java.io.PrintWriter;
  import java.io.UnsupportedEncodingException;
  import java.util.ArrayList;
  import java.util.HashMap;
  import java.util.Collections;
  import java.util.Comparator;
  import java.util.Arrays;
  import java.util.List;
  import java.util.Map;
  import java.util.Set;
  import java.util.TreeSet;

  public class Specification{
    private String id = "spec1";
    private ArrayList<Automaton> automatas;

    public Specification(){
      automatas = new ArrayList<Automaton>();
      String str;
      String str1;
      String pre;
      String succ;
      State actualState;
      State acceptState;
      State finalState;
      State newState;
      State acceptState_new;
      Automaton a = new Automaton("email");
      Automaton b;
      Map<String, Automaton> altauto;
      ArrayList<Automaton> parauto;
      Automaton loopauto;
      Automaton expression;
      int counter = 0;

      b = new Automaton("auto7");
      actualState = new State("q" + counter, StateType.NORMAL);
      counter++;
      b.addState(actualState);
      b.setInitial(actualState);

      b.addTransition(new Transition("!(" + "computer" + "." +
        "checkEmail" + "("
        + ")"

        + "." + "computer)", actualState, actualState));

      newState = new State("q" + counter, StateType.FINAL);
      counter++;
      b.addTransition(new Transition("computer" + "." +

      "checkEmail" + "("
      + ")"

      + "." + "computer" , actualState, newState));
```

```
        b.addState(newState);
        b.setFinale(newState);
        a.collapse(b);

        a.rename();
        automatas.add(a);
      }

      public void listAutomatas(){
        for(Automaton a : this.automatas){
          for(State s : a.getStates()){
            s.writeState();
          }

          for(Transition t : a.getTransitions()){
            t.writeTransition();
          }
        }
      }

      public List<Automaton> getAutomata() {
        return automatas;
      }

      public ArrayList<Automaton> par(ArrayList<Automaton> automatas) {
          ArrayList<ArrayList<Automaton>> automataList = new ArrayList<>();
          permute(automataList, new ArrayList<>(), automatas);
          return listConverter((automataList));
      }

      private void permute(ArrayList<ArrayList<Automaton>> list, ArrayList<Automaton> resultList,
    ArrayList<Automaton> automatas) {
        if (resultList.size() == automatas.size()) {
          list.add(new ArrayList<>(resultList));
        } else {
          for (int i = 0; i < automatas.size(); i++) {
            if (resultList.contains((automatas.get(i)))) {
              continue;
            }

            resultList.add(automatas.get(i));
            permute(list, resultList, automatas);
            resultList.remove(resultList.size() - 1);
          }
        }
      }

      private ArrayList<Automaton> listConverter(ArrayList<ArrayList<Automaton>> list) {
        ArrayList<Automaton> result = new ArrayList<>();
        for (ArrayList<Automaton> alist : list) {
          Automaton newauto = new Automaton("listConverter");
          for (Automaton auto : alist) {
            newauto.collapse(copyAutomaton(auto));
          }
          result.add(newauto);
        }
        return result;
      }

      public Map<String, Automaton> loopSetup(Automaton loopauto, int min, int max) {
        Map<String, Automaton> result = new HashMap<>();

        for (int i = min; i <= max; i++) {
          Automaton newauto = new Automaton("loopauto" + i);
          for (int j = 0; j < i; j++) {
            newauto.collapse(copyAutomaton(loopauto));
          }
          result.put("loop" + i, newauto);
        }
        return result;
      }
```

```
    public Automaton copyAutomaton(Automaton referenceAuto) {
        Automaton result = new Automaton("copy automaton");
        int count = 0;
        State previousSender = new State();
        State referencePreviousSender = new State();

        for (Transition t : referenceAuto.getTransitions()) {
          State sender = new State();
          State receiver = new State();
          Transition transition = new Transition();
          Automaton temp = new Automaton("temp");

          transition.setId(t.getId());

          if (t.getSender() == referencePreviousSender) {
            receiver.setId("c" + count);
            count++;
            receiver.setType(t.getReceiver().getType());

            transition.setSender(previousSender);
            transition.setReceiver(receiver);
            temp.addState(previousSender);
            temp.addState(receiver);
            temp.setInitial(previousSender);
            temp.setFinale(receiver);
          } else {
            if (t.getSender() == t.getReceiver()) {
              sender.setId("c" + count);
              count++;
              sender.setType(t.getSender().getType());

              transition.setSender(sender);
              transition.setReceiver(sender);

              temp.addState(sender);
              temp.setInitial(sender);
              temp.setFinale(sender);
            } else {
              sender.setId("c" + count);
              count++;
              sender.setType(t.getSender().getType());

              receiver.setId("c" + count);
              count++;
              receiver.setType(t.getReceiver().getType());

              transition.setSender(sender);
              transition.setReceiver(receiver);

              temp.addState(sender);
              temp.addState(receiver);
              temp.setInitial(sender);
              temp.setFinale(receiver);
            }
            previousSender = sender;
            referencePreviousSender = t.getSender();
          }

          temp.addTransition(transition);
          result.collapse(temp);
        }

        return result;
      }

    public static void main(String[] args) throws FileNotFoundException,
UnsupportedEncodingException{
      Specification specification = new Specification();
      specification.listAutomatas();
      boolean acceptState = false;

      PrintWriter writer = new PrintWriter("spec1" + ".txt", "UTF-8");
```

```
      for(Automaton a : specification.automatas){
        writer.println("");
        writer.println("never{ /*" + a.getId()+ "Monitor" + "*/");
        for(State s : a.getStates()){
          if(s == a.getInitial()){
            writer.println("T0_init:");
            writer.println(" if");
            for(Transition t : a.findSender(s)){
              if(t.getReceiver() == a.getInitial()){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_init");
              }else if(t.getReceiver().getType().equals(StateType.NORMAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT_ALL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_all" );
              }else if(t.getReceiver().getType().equals(StateType.FINAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_" + t.
getReceiver().getId());
              }
            }
            writer.println(" fi;");
          }else if(s.getType().equals(StateType.NORMAL)){
            writer.println("T0_" + s.getId() + ":");
            writer.println(" if");
            for(Transition t : a.findSender(s)){
              if(t.getReceiver() == a.getInitial()){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_init");
              }else if(t.getReceiver().getType().equals(StateType.NORMAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT_ALL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_all" );
              }else if(t.getReceiver().getType().equals(StateType.FINAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_" + t.
getReceiver().getId());
              }
            }
            writer.println(" fi;");
          }else if(s.getType().equals(StateType.ACCEPT_ALL) && !acceptState){
            writer.println("accept_all:");
            writer.println("skip");
            acceptState = true;
          }else if(s.getType().equals(StateType.FINAL)){
            writer.println("T0_" + s.getId() + ":");
            writer.println(" if");
            for(Transition t : a.findSender(s)){
              if(t.getReceiver() == a.getInitial()){
                writer.println(" :: (" + t.getId() + ")" + "->" + " goto T0_init");
              }else if(t.getReceiver().getType().equals(StateType.NORMAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT_ALL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_all" );
              }else if(t.getReceiver().getType().equals(StateType.FINAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_" + t.
getReceiver().getId());
              }
            }
            writer.println(" fi;");
          }else if(s.getType().equals(StateType.ACCEPT)){
            writer.println("accept_" + s.getId() + ":");
            writer.println(" if");
            for(Transition t : a.findSender(s)){
```

```
              if(t.getReceiver() == a.getInitial()){
                writer.println(" :: (" + t.getId() + ")" + "->" + " goto T0_init");
              }else if(t.getReceiver().getType().equals(StateType.NORMAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT_ALL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_all" );
              }else if(t.getReceiver().getType().equals(StateType.FINAL)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto T0_" + t.getReceiver()
.getId());
              }else if(t.getReceiver().getType().equals(StateType.ACCEPT)){
                writer.println(" :: (" + t.getId() + ") " + "->" + " goto accept_" + t.
getReceiver().getId());
              }
            }
            writer.println(" fi;");
          }

        }
        writer.println("}");
      }
      writer.close();

      PrintWriter xmlWriter = new PrintWriter("spec1" + ".xml", "UTF-8");
      xmlWriter.println("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
      xmlWriter.println("<!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://
www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'>");
      xmlWriter.println("<nta>");
      for (Automaton a : specification.automatas) {
        xmlWriter.println("\t<declaration>");
        String previous = "";
        List<String> existingChannels = new ArrayList<String>();
        for (Transition t : a.getTransitions()) {
          List<String> items = Arrays.asList(t.getId().split("\\s*;\\s*"));
          if (Collections.frequency(existingChannels, items.get(0)) == 0) {
            if (t.getId().startsWith("[") || t.getId().startsWith("![")) {
            } else if (!previous.equals(items.get(0).replaceAll("\\(", "_").replaceAll("\\)", "_
").replaceAll("\\.", "__").replaceAll("!", "not").replaceAll("&", "_and_").replaceAll("\\s", ""
))){
              xmlWriter.println("chan " + items.get(0).replaceAll("\\(", "_").replaceAll("\\)",
"_").replaceAll("\\.", "__").replaceAll("!", "not").replaceAll("&", "_and_").replaceAll("\\s",
"") + ";");
              previous = items.get(0).replaceAll("\\(", "_").replaceAll("\\)", "_").replaceAll("
\\.", "__").replaceAll("!", "not").replaceAll("&", "_and_").replaceAll("\\s", "");
              existingChannels.add(items.get(0));
            }
          }
        }



        xmlWriter.println("\t</declaration>");
        xmlWriter.println("\t<template>");
        xmlWriter.println("\t\t<name>" + a.getId() + "</name>");
        xmlWriter.println("\t\t<declaration>");
        xmlWriter.println("\t\t</declaration>");

        int statecounter = 0;

        for (State s : a.getStates()) {
          xmlWriter.println("\t\t<location id=\"" + s.getId() + "\" x=\"" + statecounter + "\" y
=\"" + statecounter + "\">");
            if (s.getType().equals(StateType.NORMAL)) {
              xmlWriter.println("\t\t\t<name x=\"" + statecounter + "\" y=\"" + (statecounter +
0.5) + "\">" + s.getId() + "</name>");
            } else if (s.getType().equals(StateType.ACCEPT)) {
              xmlWriter.println("\t\t\t<name x=\"" + statecounter + "\" y=\"" + (statecounter +
0.5) + "\">ACCEPT_" + s.getId() + "</name>");
            } else if (s.getType().equals(StateType.FINAL)) {
              xmlWriter.println("\t\t\t<name x=\"" + statecounter + "\" y=\"" + (statecounter +
0.5) + "\">FINAL_" + s.getId() + "</name>");
            }
```

```
          xmlWriter.println("\t\t</location>");
          statecounter++;
        }

        xmlWriter.println("\t\t<init ref=\"q0\"/>");

        for (Transition t : a.getTransitions()) {
          boolean doubletransition = false;
          xmlWriter.println("\t\t<transition>");
          xmlWriter.println("\t\t\t<source ref=\"" + t.getSender().getId() + "\"/>");
          xmlWriter.println("\t\t\t<target ref=\"" + t.getReceiver().getId() + "\"/>");
          if (t.getId().startsWith("[") || t.getId().startsWith("![")) {
            xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + t.getId().substring
(0, t.getId().indexOf("]")).replaceAll("<", "&lt;").replaceAll(">", "&gt;").replace("[", "") +
"</label>");
          } else {
            List<String> items = Arrays.asList(t.getId().split("\\s*;\\s*"));

            if (items.size() >= 1) {
              xmlWriter.println("\t\t\t<label kind=\"synchronisation\" x=\"" + t.getSender().
getId().substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get
(0).replaceAll("\\(", "_").replaceAll("\\)", "_").replaceAll("\\.", "__").replaceAll("!", "not"
).replaceAll("&", "_and_").replaceAll("\\s", "") + "?</label>");
            }

            if (items.size() >= 2) {
              if(items.get(1).contains("1,")) {
                List<String> stringList = Arrays.asList(items.get(1).split("\\s*\\|\\|\\s*"));
                doubletransition = true;
                if (!stringList.get(0).startsWith("(")) {
                  xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + stringList.get(0).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll("\\(",
 "").replaceAll("\\)", "").replaceAll(",", " and") + "</label>");
                } else {
                  xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + stringList.get(0).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll(",", "
 and") + "</label>");
                }
              } else {
                if (!items.get(1).startsWith("(")) {
                  xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get(1).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll("\\(",
 "").replaceAll("\\)", "").replaceAll(",", " and") + "</label>");
                } else {
                  xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get(1).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll(",", "
 and") + "</label>");
                }
              }
            }

            if (items.size() >= 3) {
              xmlWriter.println("\t\t\t<label kind=\"assignment\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get(2).
replaceAll("&", "&amp;") + "</label>");
            }
          }
          xmlWriter.println("\t\t</transition>");
          if (doubletransition) {
            xmlWriter.println("\t\t<transition>");
            xmlWriter.println("\t\t\t<source ref=\"" + t.getSender().getId() + "\"/>");
            xmlWriter.println("\t\t\t<target ref=\"" + t.getReceiver().getId() + "\"/>");
            if (t.getId().startsWith("[") || t.getId().startsWith("![")) {
              xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + t.getId().substring
(0, t.getId().indexOf("]")).replaceAll("<", "&lt;").replaceAll(">", "&gt;").replace("[", "") +
"</label>");
```

```
            } else {
              List<String> items = Arrays.asList(t.getId().split("\\s*;\\s*"));

              if (items.size() >= 1) {
                xmlWriter.println("\t\t\t<label kind=\"synchronisation\" x=\"" + t.getSender().
getId().substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get
(0).replaceAll("\\(", "_").replaceAll("\\)", "_").replaceAll("\\.", "__").replaceAll("!", "not"
).replaceAll("&", "_and_").replaceAll("\\s", "") + "?</label>");
              }

              if (items.size() >= 2) {
                if(items.get(1).contains("1,")) {
                  List<String> stringList = Arrays.asList(items.get(1).split("\\s*\\|\\|\\s*"));
                  doubletransition = true;
                  if (!stringList.get(1).startsWith("(")) {
                    xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId()
.substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + stringList.get(1).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll("\\(",
 "").replaceAll("\\)", "").replaceAll(",", " and") + "</label>");
                  } else {
                    xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId()
.substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + stringList.get(1).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll(",", "
 and") + "</label>");
                  }
                } else {
                  if (!items.get(1).startsWith("(")) {
                    xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId()
.substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get(1).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll("\\(",
 "").replaceAll("\\)", "").replaceAll(",", " and") + "</label>");
                  } else {
                    xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId()
.substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get(1).
replaceAll("&", "&amp;&amp;").replaceAll("<", "&lt;").replaceAll(">", "&gt;").replaceAll(",", "
 and") + "</label>");
                  }
                }
              }

              if (items.size() >= 3) {
                xmlWriter.println("\t\t\t<label kind=\"assignment\" x=\"" + t.getSender().getId
().substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get(2).
replaceAll("&", "&amp;") + "</label>");
              }
            }
            xmlWriter.println("\t\t</transition>");
          }
        }

        xmlWriter.println("\t</template>");
        xmlWriter.println("\t<template>");
        xmlWriter.println("\t\t<name>" + a.getId() + "_environment</name>");
        xmlWriter.println("\t\t<declaration>");
        xmlWriter.println("\t\t</declaration>");
        xmlWriter.println("\t\t<location id=\"q0\" x=\"0\" y=\"0\">");
        xmlWriter.println("\t\t\t<name x=\"1\" y=\"1\">q0</name>");
        xmlWriter.println("\t\t</location>");
        xmlWriter.println("\t\t<init ref=\"q0\"/>");

        Set<Transition> unique_transitions = new TreeSet<Transition>(new Comparator<Transition
>() {
          @Override
          public int compare(Transition t1, Transition t2) {
            List<String> items1 = Arrays.asList(t1.getId().split("\\s*;\\s*"));
            List<String> items2 = Arrays.asList(t2.getId().split("\\s*;\\s*"));

            return !(items1.get(0).equals(items2.get(0))) ? 1 : 0;
          }
        });

        unique_transitions.addAll(a.getTransitions());
```

```
        for (Transition t : unique_transitions) {
          xmlWriter.println("\t\t<transition>");
          xmlWriter.println("\t\t\t<source ref=\"q0\"/>");
          xmlWriter.println("\t\t\t<target ref=\"q0\"/>");
          if (t.getId().startsWith("[") || t.getId().startsWith("![")) {
            xmlWriter.println("\t\t\t<label kind=\"guard\" x=\"" + t.getSender().getId().
substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + t.getId().substring
(0, t.getId().indexOf("]")).replaceAll("<", "&lt;").replaceAll(">", "&gt;").replace("[", "") +
"</label>");
          } else {
            List<String> items = Arrays.asList(t.getId().split("\\s*;\\s*"));

            if (items.size() >= 1) {
              xmlWriter.println("\t\t\t<label kind=\"synchronisation\" x=\"" + t.getSender().
getId().substring(1) + ".5\" y=\"" + t.getSender().getId().substring(1) + ".5\">" + items.get
(0).replaceAll("\\(", "_").replaceAll("\\)", "_").replaceAll("\\.", "__").replaceAll("!", "not"
).replaceAll("&", "_and_").replaceAll("\\s", "") + "!</label>");
            }

          }
          xmlWriter.println("\t\t</transition>");
        }

        xmlWriter.println("\t</template>");
      }

      xmlWriter.println("\t<system>");
      xmlWriter.println("receiver = " + specification.automatas.get(0).getId() + "();");
      xmlWriter.println("sender = " + specification.automatas.get(0).getId() + "_environment();"
);
      xmlWriter.println("system receiver, sender;");
      xmlWriter.println("\t</system>");

      xmlWriter.println("</nta>");
      xmlWriter.close();
    }
  }
```

File 8 : /myProject/./src-gen/State.java

```
package generated;

public class State {
  private String id;
  private StateType type;

  public State() {
    this.id = "q0";
    this.type = StateType.NORMAL;
  }

  public State(String id, StateType stateType) {
    this.id = id;
    this.type = stateType;
  }

  public String getId() {
    return id;
  }

  public StateType getType() {
    return type;
  }

  public void setType(StateType type) {
    this.type = type;
  }

  public void setId(String id) {
    this.id = id;
  }
```

```
  public void writeState(){
    System.out.println(this.id + " " + this.type);
  }
}

File 9 : /myProject/./src-gen/StateType.java

package generated;

public enum StateType {
  NORMAL, ACCEPT, FINAL, ACCEPT_ALL
}


File 10 : /myProject/./src-gen/Transition.java

package generated;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Transition {
  private String id;
  private State sender;
  private State receiver;

  private String reset;
  private String clock;
  private int value;
  private boolean smaller;

  public Transition() {
      this.id = "t0";
      this.sender = new State();
      this.receiver = new State();
      this.smaller = false;
  }

  public Transition(String id, State sender, State receiver) {
    if (id.equals("1")) {
      this.id = "true";
    } else {
      this.id = id;
    }

    if (id.contains(";")) {
      List<String> list = new ArrayList<String>(Arrays.asList(id.split(";")));
      this.id = list.get(0);
      if (list.get(1) != null) {
        if (list.get(1).contains(" = 0")) {
          this.reset = list.get(1).substring(1, 2);
        } else if (list.get(1).contains("<")) {
          this.value = Integer.parseInt(list.get(1).substring(list.get(1).length() - 2, list.get
(1).length())) * 1000000000;
          this.clock = list.get(1).substring(1, 2);
          this.smaller = true;
        }
      }
    }

    /*if (countChar(this.id, '!') % 2 == 0 && this.id.contains("!")) {
      formatID();
    }*/

    this.sender = sender;
    this.receiver = receiver;
  }

  private int countChar(String someString, char someChar) {
    int count = 0;
```

```
    for (int i = 0; i < someString.length(); i++) {
      if (someString.charAt(i) == someChar) {
        count++;
      }
    }

    return count;
  }

  public boolean hasReset() {
    return reset != null && !reset.isEmpty();
  }

  public String getReset() {
    return reset;
  }

  public boolean hasClock() {
    return clock != null && !clock.isEmpty();
  }

  public String getClock() {
    return clock;
  }

  public boolean clockConditionSatisfied(long time) {
    if (this.smaller) {
      return time < value;
    }

    return false;
  }

  void formatID() {
    this.id = this.id.substring(5, this.id.length() - 1);
  }

  public String getMessageType() {
    String messageType = this.id.substring(2, this.id.length() - 1);

    messageType = messageType.substring(messageType.indexOf(".") + 1);
    messageType = messageType.substring(0, messageType.indexOf("("));

    return messageType;
  }

  public String[] getParameters() {
    String messageType = this.id.substring(2, this.id.length() - 1);

    messageType = messageType.substring(messageType.indexOf("(") + 1);
    messageType = messageType.substring(0, messageType.indexOf(")"));

    if (messageType.equals("")) {
      return new String[0];
    }

    return messageType.split(",");
  }

  public String getSenderName() {
    String sender = this.id;
    if (this.id.contains("!")) {
      sender = this.id.substring(2, this.id.length() - 1);
    }

    sender = sender.substring(0, sender.indexOf("."));

    return sender;
  }

  public String getReceiverName() {
```

```
        String receiver = this.id;
        if (this.id.contains("!")) {
          receiver = this.id.substring(2, this.id.length() - 1);
        }

        receiver = receiver.substring(receiver.indexOf(")") + 2);

        return receiver;
      }


      public String getId() {
        return id;
      }

      public State getSender() {
        return sender;
      }

      public State getReceiver() {
        return receiver;
      }

      public void setReceiver(State receiver) {
        this.receiver = receiver;
      }

      public void setSender(State sender) {
        this.sender = sender;
      }

      public void setId(String id) {
        if(id.equals("1")){
          this.id = "true";
        }else{
          this.id = id;
        }
      }

      public void writeTransition(){
        System.out.println(this.id + " " + this.sender.getId() + "->" + this.receiver.getId());
      }
    }

''')
}
```

F.3.1. kódrészlet. példa unit teszteset.