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. május 12.

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. május 12.

Bakai István Bálint
hallgató

Kivonat

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

Az önálló laboratórium feladat célja az volt, hogy a „Monitor komponensek generálása kontextusfüggő viselkedés ellenőrzésére” című szakdolgozatomban definiált szöveges PSC (Property Sequence Chart) leíró nyelvet kibővítsem úgy, hogy időzítési feltételeket is meg lehessen adni a követelményben. Ilyen követelményeket egyszerűen specifikálhatunk TPSC (Timed Property Sequence Chart) diagramokkal. Az Xtext alapú nyelvet kiegészítettem a TPSC tulajdonságaival.

A monitor generálás következő lépése, hogy a PSC diagramokból TA 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.

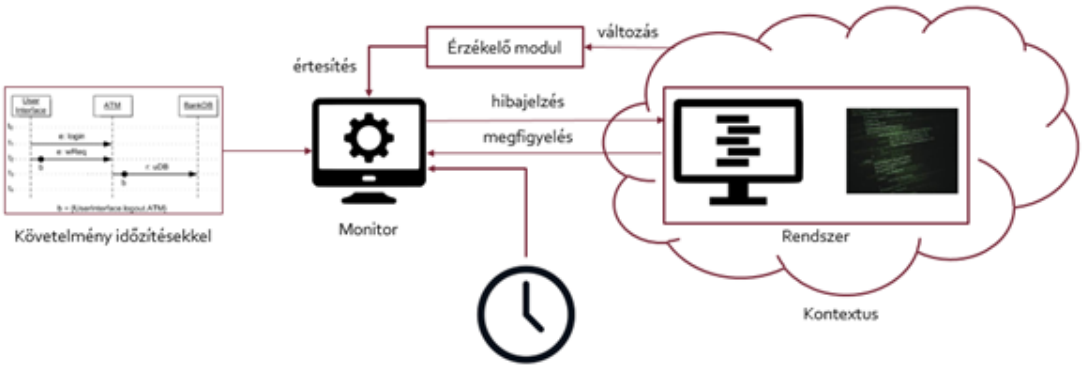
Abstract

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 - Rendszerkomponensek közti kommunikáció monitorozása

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. Egy rendszer viselkedése lehet kontextusfüggő, amit a monitornak figyelembe kell venni. Például, egy gépjármű fékezésének vezérlését befolyásolja a terep, amin épp halad. A rendszer működése tehát függ a környezetétől. Ezért, hogy a viselkedését ellenőrizni tudjuk, a monitornak információval kell rendelkeznie a környezetben történő változásokról. Ezen kívül, a monitornak időmérésre is szüksége van, mert a követelmény tartalmazhat időziteseket is. Az 1.1. ábra bemutatja a monitorozás koncepcióját.



1.1. 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 specifikálására. Szekvencia diagramok segítségével egyszerűen megadhatunk ilyeneket. A diagramokat a későbbiekben olyan alakra kell majd hoznunk, hogy abból a monitor létrehozható legyen.

2. fejezet

PSC – Property Sequence Charts

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.

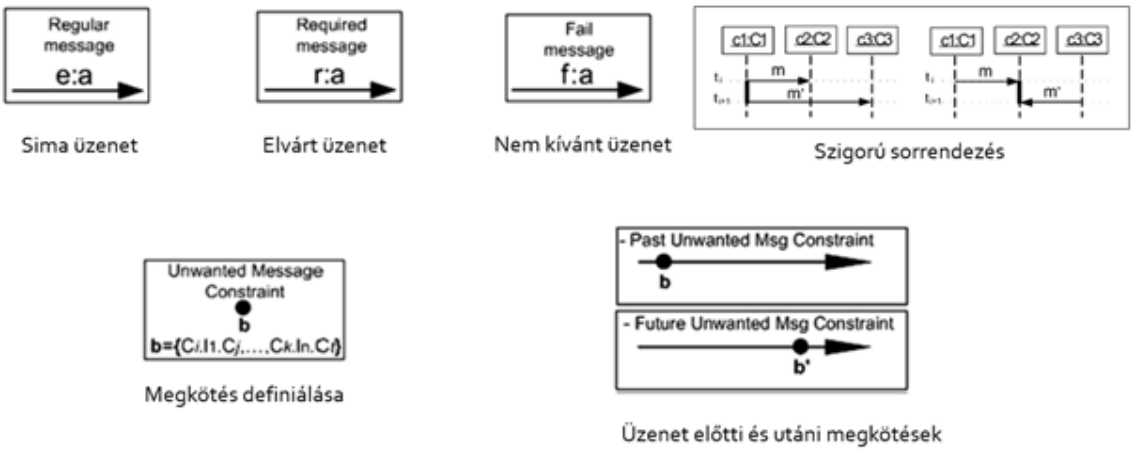
Tulajdonság	**MSC**	**PSC**
Nem kívánt üzenet	-	Fail message
Elvárt üzenet	-	Required message
Sima üzenet	Default message	Regular message
Megkötött sorrendezés	-	Strict sequencing
Gyenge sorrendezés	Seq	Loose sequencing
Üzenet megkötések	-	Constraint
Alternatív lehetőségek	h-MSC	Alternative operator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

2.1. táblázat. Az MSC összehasonlítása a PSC-vel

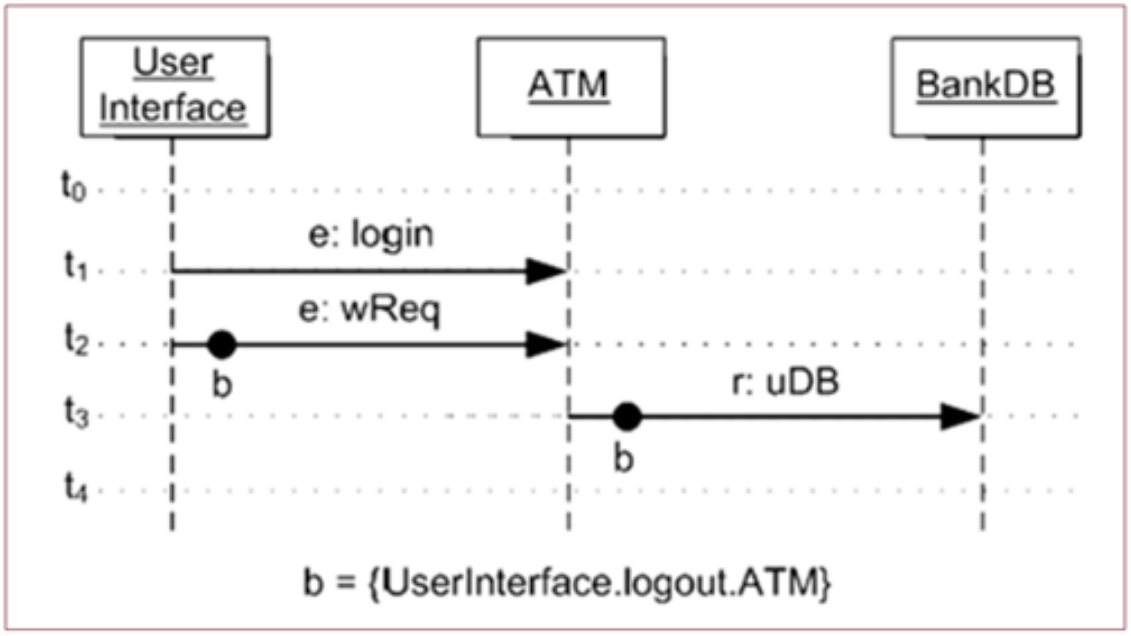
A Property Sequence Chart[1] az MSC egy kiterjesztése. Sok új elemet vezet be ami nincs az MSC-ben, melyek megtekinthetők az 2.1. táblázaton, mint az üzenet típusokat: sima üzenet (e), elvárt üzenet (r) és nem kívánt üzenet (f). Így specifikálhatjuk, hogy mely üzenetek azok amik helyes viselkedésre utalnak és azok amelyek nem. Az elvárt üzenetek azok az üzenetek amelyeknek feltétetlen meg kell történiük a rendszer működése során. Egy sima üzenetnek 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.

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

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, még pedig hogy az üzenet előtt nem történt kijelentkezés, *logout*. Az ATM ezután, ha nem történt *logout* egy elvárt üzenetet küld a Bank adatbázisába.



2.1. ábra. A PSC különböző elemei[1].



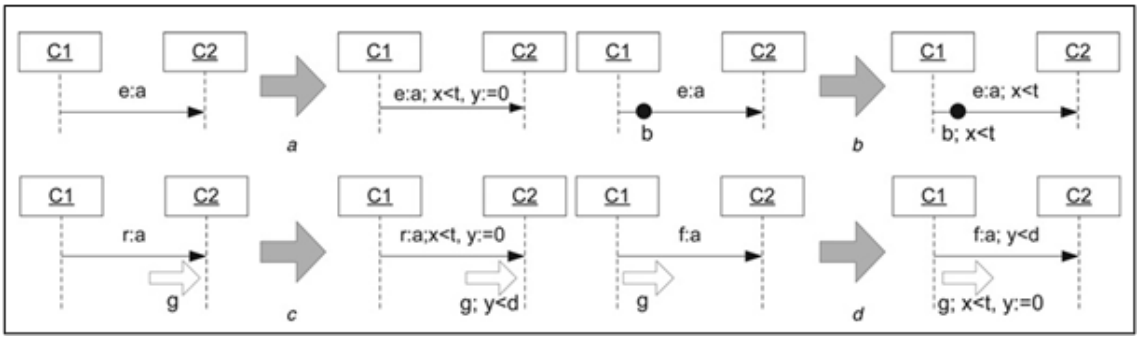
2.2. ábra. PSC diagram egy ATM rendszer működésének az ellenőrzésére[1].

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.

3. fejezet

TPSC – Timed Property Sequence Charts

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



3.1. 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 lehet időzítést indítani, majd rákövetkező események időbeliségét ellenőrizni.

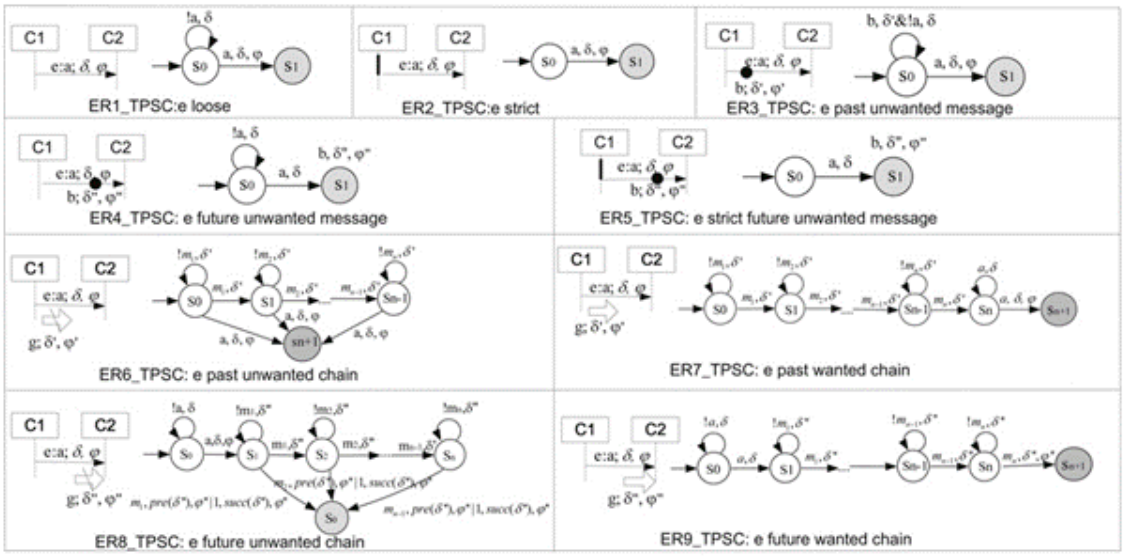
A 3.1. ábrán látható, hogy például az *e: a* sima üzenet *e: a; x < t, y := 0* üzenetre bővül. Elvárjuk, hogy az *a* üzenet *t* idő előtt történjen meg és egy *y* óraváltozót nullázunk. Az *e: a* üzenet egy sima üzenet, szóval ha nem történik meg a specifikált idő intervallumban az nem jelent hibát. 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.

4. fejezet

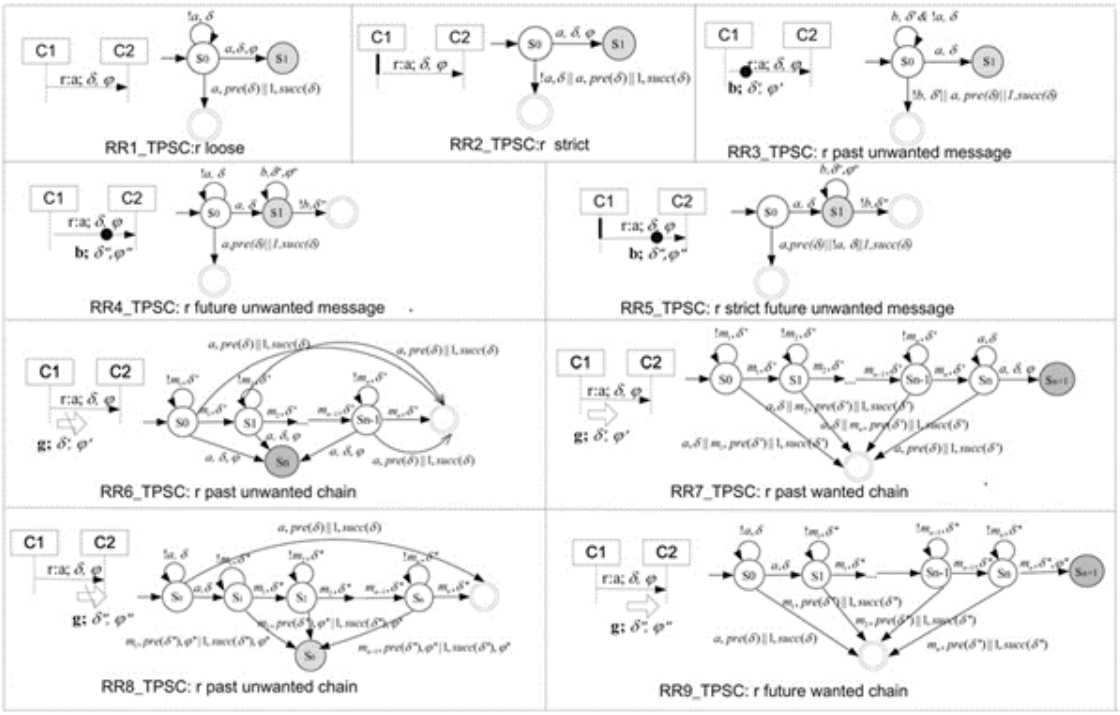
TPSC scenario-ból automata készítése

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.

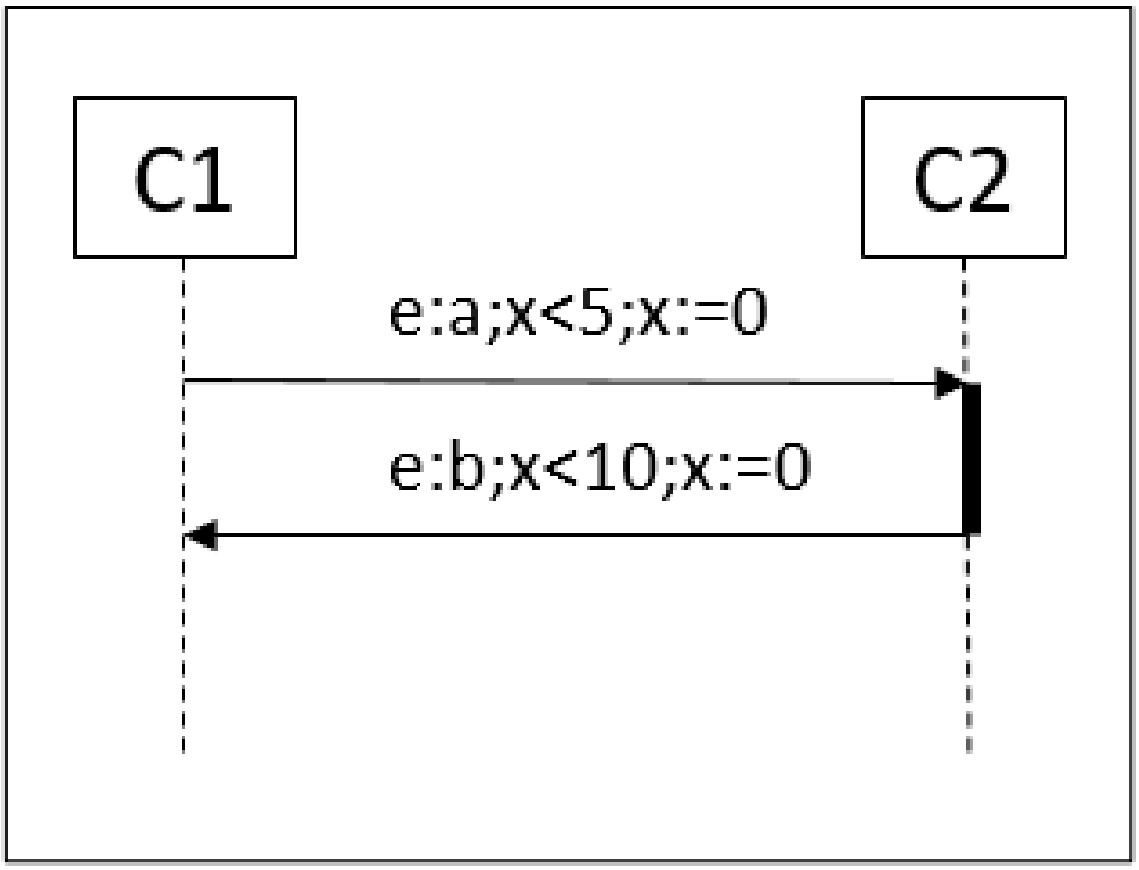


4.1. ábra. Sima üzenetekhez tartozó minták[2].

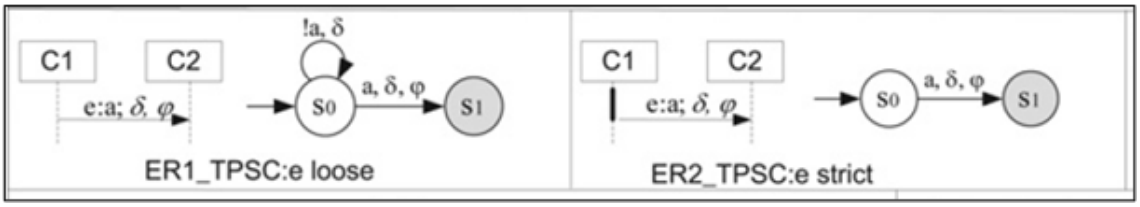
A minta automatánkban található szürke állapotok reprezentálják a végállapotokat. A megkötéseket az automatáknál egy átmenettel definiáljuk ami a nem kívánt üzenetek negáltjainak az ÉS kapcsolata. Például az 4.1. ábrán lévő 3. automata mintán látható „b, δ'&!a, δ" címkéjű hurokélen, a „b" azt jelzi, hogy amig nem jött nem kivánt üzenet addig maradjunk az S0 állapotban. A címke teljes jelentése az, hogy ha δ' időn belül nem jött nem kivánt 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.



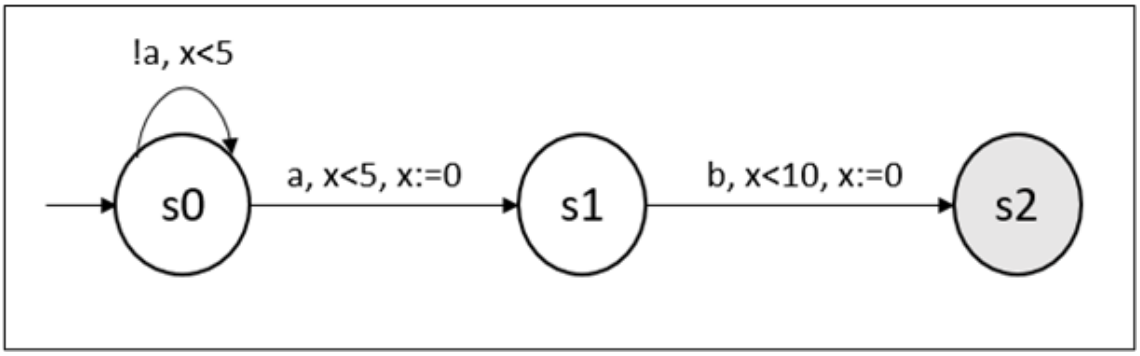
4.2. ábra. Elvárt üzenetekhez tartozó minták[2].



4.3. ábra. TPSC részlet.



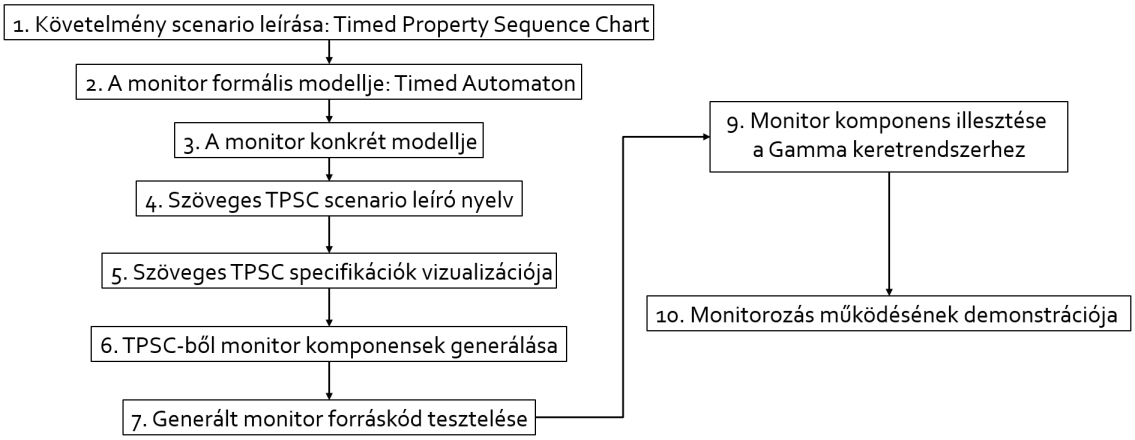
4.4. ábra. Alkalmazott automata minták.



4.5. ábra. Az összeállított automata.

5. fejezet

Monitor generálás terv és feladat megfogalmazása



5.1. ábra. Monitor generátor kibővítése.

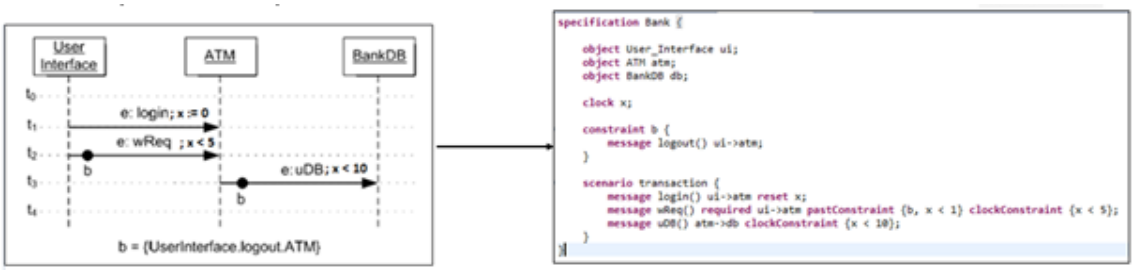
A cél az, hogy a „Monitor komponensek generálása kontextusfüggő viselkedés ellenőrzése” című szakdolgozatom során elkészített monitor komponens generátort kibővíteni úgy, hogy támogassa időzítési feltételek megadását. A monitor generálás terve látható a 5.1. ábrán. Először az a feladatunk, hogy a szakdolgozat során definiált szöveges PSC diagram leíró nyelvet kibővitsük 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 Diplomatervezés 1 tárgy keretében a tervnek a negyedik, hatodik és hetedik részeivel foglalkoztam.

6. fejezet

Szöveges PSC leíró nyelv kibővítése

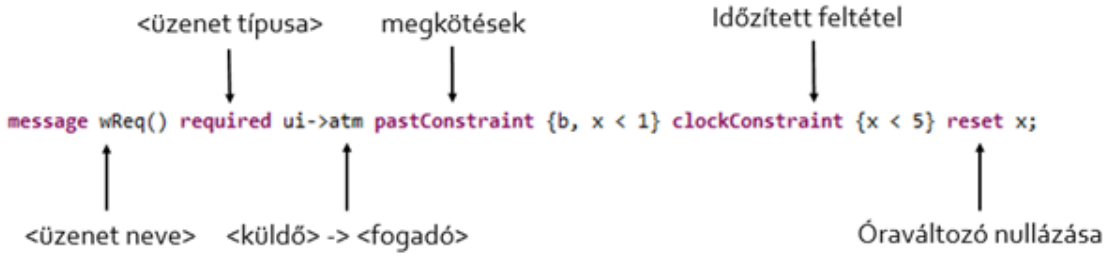
A nyelvet az Xtext technológia segítségével definiáltam. A nyelv két új elemmel bővült:

- időzített feltétel
- óraváltozó nullázása



6.1. 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.



6.2. 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.

7. fejezet

Időzített automata generátor

7.1. Az automata generátor célja

A diplomatervezés 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. 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;
    }
  }
}
```

7.1. kódrészlet. Alt operátort tartalmazó scenario.

7.2. 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 áll:

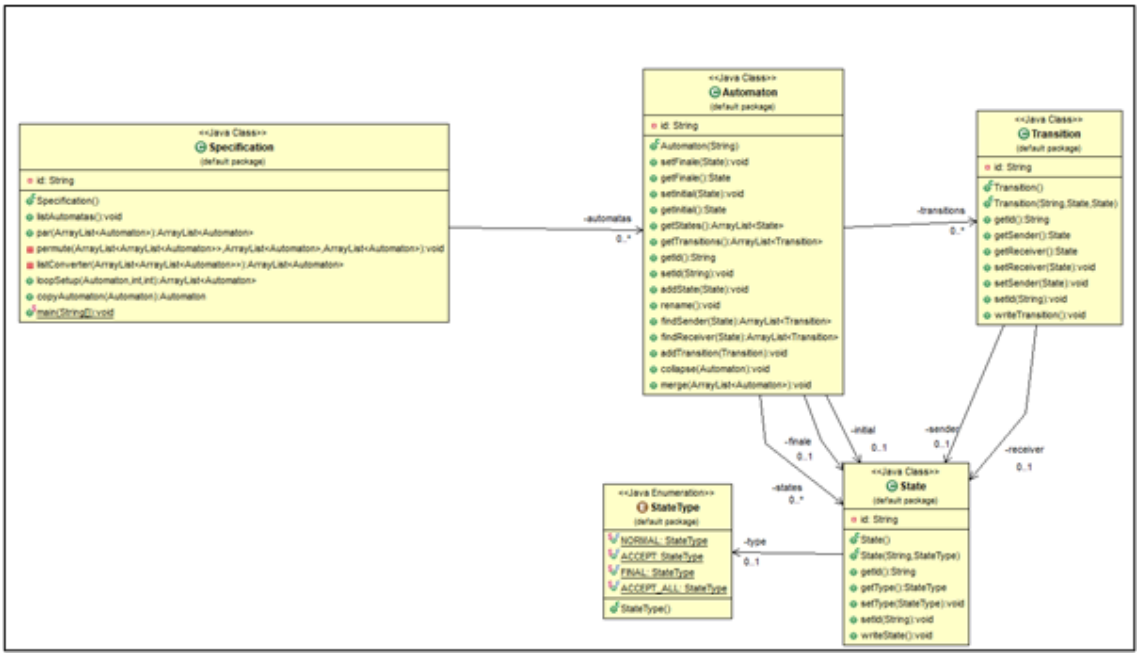
```
bool success = true;

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;
}
```

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

- State
- StateType
- Transition
- Automaton
- Specification

A 7.1. ábrán látható 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.



7.1. ábra. Az adatstruktúra UML osztály diagramja.

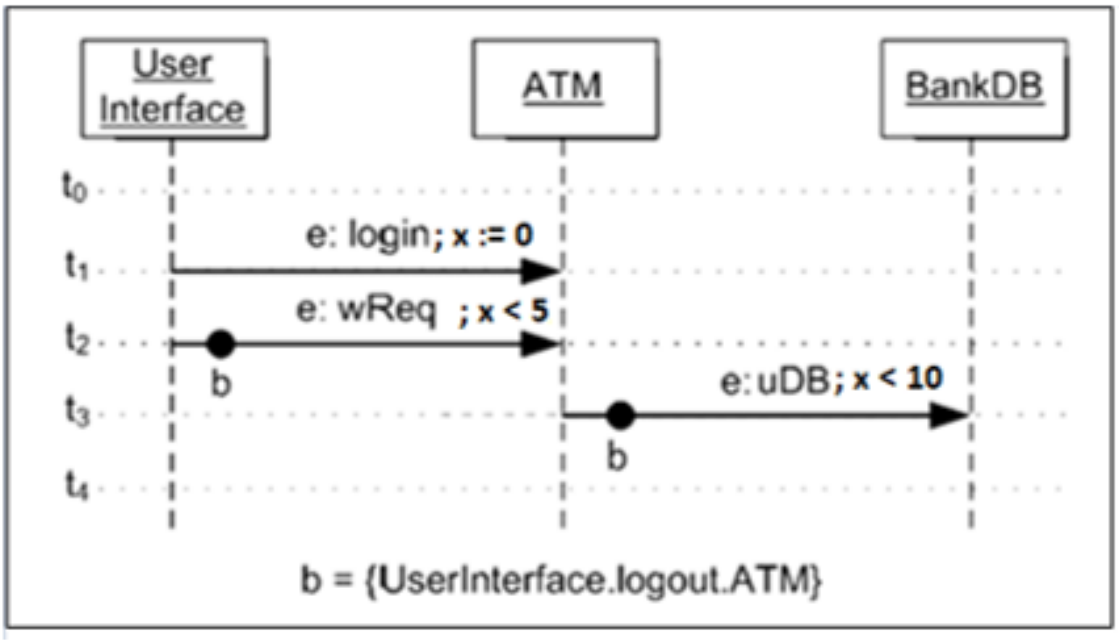
Az állapot típusának a megadására a StateType enum osztályt definiáltam. NORMAL, ACCEPT, FINAL értékeket lehet benne eltárolni. Az átmenetek implementációjáért felelős osztály a Transition. Három tag változója van: id(String) az üzenet, sender(State), a feladó állapot, és receiver(State) a fogadó állapot.

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 az előző fejezetben definiált merge függvény implementációja.

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.

7.3. Minta példa

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



7.2. 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;
    message wReq() required ui -> atm pastConstraint {b, x < 1} clockConstraint {x < 5};
    message uDB() atm -> db clockConstraint {x < 10};
  }
}
```

7.3. kódrészlet. TPSC scenario szöveges leírása.

```
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;
}
```

7.4. kódrészlet. Generált időzített automata never claim formátumban.

8. fejezet

Monitor forráskód generátor

8.1. 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() függvényt a monitor hívja amikor a rendszer állapota változik. Ha a rendszer nem elfogadó állapotban van akkor igazat ad vissza, ha pedig nem akkor hamisat. A requirementSatisfied() függvényt is a monitor hívja. 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)
     ;
}
```

8.1. 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);
}
```

8.2. kódrészlet. Időzitő interfész Java implementációja.

A 8.2. kódrészlet tartalmazza a IClock java interfészt.

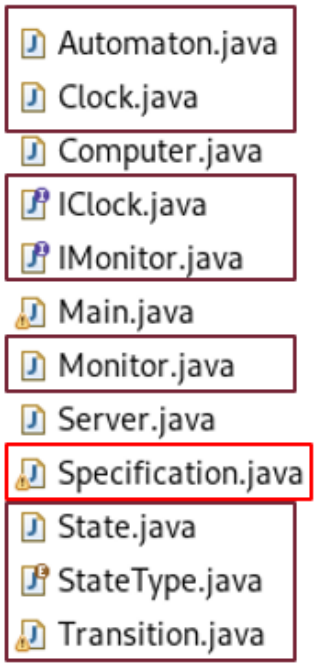
8.2. A monitor forráskód megvalósítása

A generált forráskód struktúrája egy statikus és egy 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 feketén.



8.1. ábra. A monitor forráskódjának struktúrája.

A szükséges forráskódok generálásához az Xtend technológiát használtam. Az előző fejezetben ismertetett generátort kiegészítettem egy Monitor Java osztállyal, ami a monitor forráskódjának implementációját tartalmazza.

8.3. Minta példa

A 8.3. kódrészleten látható egy scenario követelmény, amit egy példa 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 spec1 {
  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;
    message getPhoto() required device -> user;
    message cameraOffline() fail user -> device;
    message retrieveMood() required strict device -> db;
    message retrieveMusic() required device -> db;
    message generatePlaylist() strict db -> device;
  }
}
```

8.3. 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.

8.4. 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;
}
```

8.4. 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();
  }
}
```

8.5. kódrészlet. Az okos telefon és hozzá tartozó monitor összecsatolásának 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[] {});
  }
}
```

8.6. 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.
```

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

8.5. 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.

```
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};
  }
}
```

8.8. 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);
  }
}
```

8.9. kódrészlet. Időzítéses példához tartozó Main osztály.

```
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();
  }
}
```

8.10. 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.
```

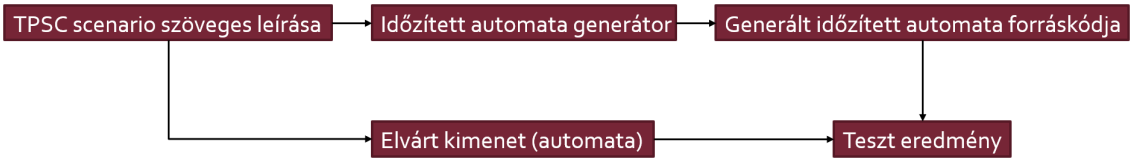
8.11. kódrészlet. Időzítéses példa monitor kimenete.

9. fejezet

Tesztelési terv

9.1. Időzített automata generátor tesztelése

A generált időzített automaták forráskodját unit tesztek segítségével szeretnénk tesztelni. Az Xtext keretrendszer által nyujtott eszközök erre célra jól alkalmazhatok.



9.1. ábra. Unit tesztelés folyamatábrája.

A 9.1. ábrán látható a unit tesztelés tervének folyamatábrája. A teszteléssel az a célunk, hogy minél nagyobb magabiztossággal biztosítjuk a generátor helyes működését. Tesztelési kategoriák:

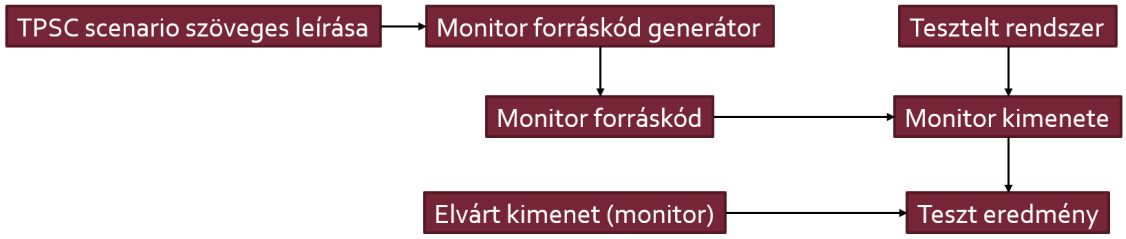
- Üzenet típusok egyenkénti tesztelése
- Üzenet típusokból összeállított kombinációk tesztelése
- Scenario operátorok tesztelése

Az üzenet típusok egyenkénti tesztelésénél az összes üzenet típust le szeretnénk fedni. Az üzenet kombinációkból a főbb eseteket szeretnénk tesztelni. Például sima üzenet után required üzenet és ezek váltakozása a fail üzeneteket is bele értve. A scenario operátoroknál azt fontos tesztelni, hogy a különálló üzenetekhez tartozó automata minták jól illeszkedjenek a operátorral elátott üzenetek automatájával.

Egy unit teszt akkor sikeres ha a generált automata forráskódja megegyezik az elvárt automata forráskodjával.

9.2. Monitor forráskód generátor tesztelése

A generált monitor forráskodját integrációs tesztek segítségével szeretnénk tesztelni. Az Xtext keretrendszer a specifikált dsl nyelvhez generál egy maven plugin-t. Ezt a plugin-t betölthetjük egy egyszerű maven projektbe és használhatjuk is az elkészített dsl nyelvünket.



9.2. 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.

10. fejezet

Összefoglalás

A célként kitűzött scenario alapú monitor generátor kibővítése részlegesen sikerült.

A szöveges scenario leíró nyelv támogatja TPSC diagramok specifikálását. Az automata generátort 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.

11. 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.