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. december 9.

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. december 9.

Bakai István Bálint
hallgató

Kivonat

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

A diplomaterv projektem célja az volt, hogy a szakdolgozatom során elkészített monitor komponens generátort kiegészítsem úgy, hogy időzített üzenet szekvencia specifikáció alapján is képes legyen monitor komponenseket generálni. A generált monitor feladata az üzenet szekvencia által specifikált viselkedés ellenőrzése. Ilyen követelményeket egyszerűen specifikálhatunk *TPSC* (*Timed Property Sequence Chart*) diagramokkal. A szakdolgozatom során 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 úgy, hogy képes legyen a minta alapú módszert használva *TA* automatákat generálni a *TPSC* diagramokból.

A szöveges szcenárió 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 diplomatervezés feladatom része továbbá a *TPSC* szcenáriók vizualizációja, a generált forráskódok szisztematikus 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. Végezetül az utolsó feladat a monitorozás működésének demonstrációja.

Abstract

Runtime verification of a critical system is essential for its operation and maintenance. With runtime verification we can discover a lot of errors that may stay undiscovered after testing.

The goal of this thesis is to further enhance the previously created monitor generator, so that it is able to generate monitor source code from scenario containing clock constraints. We can specify this sort of scenario using the *TPSC* (*Timed Property Sequence Chart*) diagrams. During my *BSc* thesis, I have developped a language using *Xtext* for specifying *PSC* diagrams via text. I have extended this language so that *TPSC* diagrams can be created using a textual format.

The next step of monitor generation is to convert the *TPSC* scenarios into *Timed Automata*. The *TA* will serve as the representation which indicates if the system's behaviour has satisfied the *TPSC* requirement or not. I have further enhanced the automaton generator created during my *BSc* thesis so that it is able to generate *Timed Automata* from *TPSC*s.

We can synthesize the monitor source code using the generated automaton from the *TPSC* scenario. The monitor source code generator creates the *Java* implementation of the monitor which is able to perform runtime verification of a system based on a specified requirement.

The remaining tasks for the thesis are the visualization of *TPSC* scenarios, the systematic testing of generated monitor source code and the integration of the generated monitor with the *Gamma* framework. The goal is to be able to monitor component-based system based on a requirement specified with a *TPSC* scenario. The last task is to demonstrate the runtime verification of a system.

1. fejezet

Bevezetés

Diplomamunkám során az volt a cél, hogy a „*Monitor komponensek generálása kontextusfüggő viselkedés ellenőrzése*” [1] című szakdolgozatom során elkészített monitor komponens generátort kibővítsem úgy, hogy támogassa időzítési feltételek megadását. A szakdolgozat során egy olyan monitor komponens generátort készítettem, amely képes volt *PSC* (*Property Sequence Chart*) [10] diagramok szöveges leírásából *Büchi* [10] automatákat generálni. A generátornak a kimenete a generált *Büchi* automata *Java* implementációja, amely felhasználható monitorozásra. Az *Önálló laboratórium* keretében az volt a feladatom, hogy a szakdolgozatom során definiált szöveges *PSC* diagram leíró nyelvet kibővítsem úgy, hogy időzítési feltételeket is meg lehessen adni a szcenáriókban. A *Timed Property Sequence Chart* [7] formalizmust választottam az időzített feltételek bevezetéséhez. Definiáltam a meglévő *PSC* szöveges leíróhoz új nyelvi elemeket, amelyekkel a *TPSC* elemeket lehet szöveges formában leírni. Ezután az automata generátort kellett úgy kibővíteni, hogy a *TPSC* diagramokból tudjon *TA* időzített automatákat [7] generálni. Ennek érdekében, a szakdolgozat során készített *Büchi* automata generátort bővítettem ki úgy, hogy az új *TPSC* szöveges leírásokból időzített automatákat generáljon. Egy monitor forráskód generátor pedig az automata alapján elkészítheti a monitor forráskódját.

A szöveges *TPSC* szcenárió leírása alapján el kell készítenünk a diagram vizualizációját, hogy grafikusan megtekinthessük a definiált szcenáriót. Ehhez felhasználható a "*Modell alapú rendszertervezés*" című tárgy keretében általam készített *PSC* diagram szerkesztő alkalmazás. A következő a generált monitor forráskód tesztelése, majd ezután a kód illesztése a *Gamma* keretrendszerhez [5]. 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 hátramaradó feladatok a monitor forráskód generátor kibővítése és annak tesztelése, a diagramok vizualizációja és a monitor komponens illesztése a *Gamma* keretrendszerhez.

A dolgozatomat a háttérismeretek összefoglaló fejezettel kezdem. Először bemutatom a legelterjedtebb formalizmusokat, amelyek időfüggő viselkedés specifikálására szolgálnak. Ezután a *TPSC* formalizmust mutatom be és a felhasznált technológiákat. A dolgozatomat a kibővített szöveges *TPSC* leíró nyelv bemutatásával folytatom, amit a harmadik fejezetben írok le. Ezt a *TPSC* specifikációk vizualizációjáról szóló fejezet követi.

Az ötödik fejezet a monitor forráskód generálásról szól és annak teszteléséről. Ezt követi a hatodik fejezet, amelyben bemutatom a monitor komponens illesztését a *Gamma* keretrendszerhez, valamint ismertetem az elosztott komponensű rendszerek monitorozását. A *Gamma* keretrendszer komponens alapú elosztott rendszerek tervezését, kódgenerálását

segíti, így érdemes a keretrendszerhez a generált monitort illeszteni. A dolgozatomat egy összefoglalóval zárom.[1]

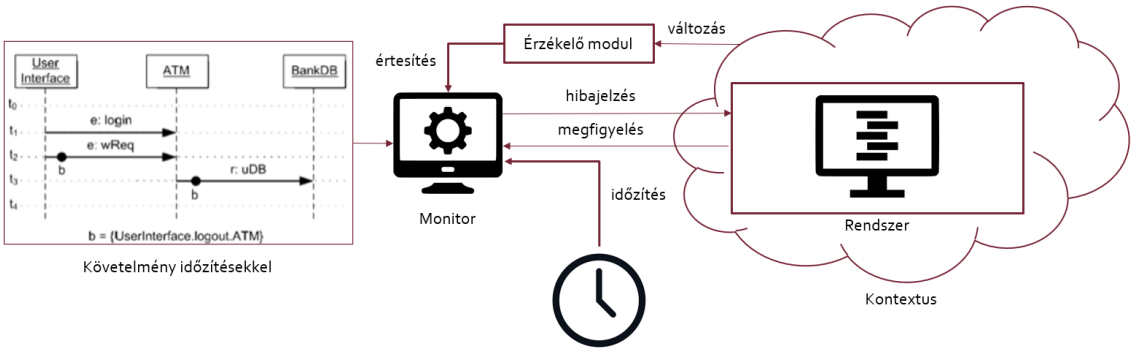
[1] A diplomatervhez készített alkalmazások forráskódjai a következő *Github repository*-ban elérhetők: `https://github.com/BakaiIstvan/Minotor`.

2. fejezet

Háttérismeretek

2.1. A monitorozás alapjai

Egy monitor feladata az, hogy egy rendszert futási időben megfigyeljen, elemezzen és adott követelmények 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 időzítéseket is. Az 2.1 ábra bemutatja a monitorozás koncepcióját.



2.1. ábra. Kontextusfüggő rendszerek monitorozása időzítési feltételekkel [1].

Több monitorozási módszert ismerünk, ilyen például az „*online*” [6] monitorozás, amelynél futási időben történik az ellenőrzés. Ebben az esetben a rendszernek olyan érzékelői vannak, amelyek képesek a rendszer viselkedését detektálni, és továbbítani azt egy feldolgozó egységnek. Ez utóbbi a kapott adatok alapján el tudja dönteni, hogy a rendszer helyesen működik-e, illetve amennyiben hibát érzékel, azt képes jelezni. Az olyan monitort, amely a hiba érzékelését követően megpróbálja a hibát helyreállítani, „beavatkozó” monitornak hívjuk. A „nem beavatkozó” monitor pedig csak a hibát képes jelezni.

A fentiekben leírt monitorozási módszernek a párja az „*offline*” [6] monitorozás. Ennél a módszernél a regisztrált viselkedés elemzése nem futásidőben, hanem azon kívül (futás után) történik.

A szcenárió 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 szcenárió formájában adjuk meg az üzenet szekvenciák specifikálásával. Ezt szekvencia diagramok segítségével könnyen

megtehetjük. 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 és UML - Unified Modelling Language

Az egyik legelterjedtebb szcenárió alapú modellezésre használt vizuális formalizmus a *Message Sequence Charts (MSC)* [16]. A nyelv célja két, vagy több üzeneteket cserélő objektum közötti interakciók leírása. A *Unified Modelling Language (UML)* 2.0 [8] szekvencia diagram leíró részét nagyban inspirálta ez a nyelv [16]. 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)* [16] nevű *MSC* kiterjesztésben találhatók.

Ezzel a nyelvvel könnyedén specifikálhatjuk a szcenárióban lévő üzeneteket, valamint 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 („*lifeline*"-ok) autonóm entitásokat képviselnek, míg a szekvencia diagramok esetén ezek egy-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, például hiányzik belőle az üzenetek típusossága. Egy követelmény megfogalmazásakor fontos, hogy meg tudjuk mondani, melyek az elvárt üzenetek, vagy melyek jelentenek hibát. Az is nagy hiányosságnak számít, hogy az üzenetekre nem lehetséges megkötéseket megadni. Ez megnehezíti egy követelmény leírását is.

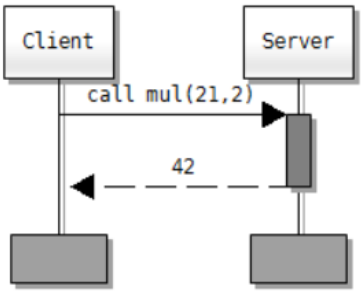
A 2.2 ábrán megtekinthető egy példa MSC diagram.

2.2.2. MTL - Metric Temporal Logic

A temporális logikák olyan formális rendszerek amelyekkel kijelentések igazságának logikai időbeli változását vizsgálhatjuk [9]. A kijelentések időbeli vizsgálásához temporális operátorokat használunk: *mindig*, *valamikor*, *mielőtt*, *"addig, amíg"*, *"azelőtt, hogy" stb...*

A temporális logikákat két osztályba sorolhatjuk:

- *lineáris*



2.2. ábra. Példa MSC diagram [16].

- *elágazó*

A lineáris temporális logikáknál a modell egy lefutását vizsgáljuk. A logikai időt egy állapotsorozatnak tekintjük, ahol minden állapotnak egy rákövetkezője van. Az elágazó temporális logikáknál viszont az összes lehetséges végrehajtást tekintjük. A lineárissal ellentétben itt egy állapotnak több rákövetkezője is lehet és a logikai idő egy fa alakjában jelenik meg.

Az *MTL* [9] egy lineáris temporális logika, amellyel logikai jelek időzítésbeli tulajdonságait specifikálhatjuk.

A szintaxisa a *Linear Temporal Logic*-hoz hasonlít:

- atomi kijelentések véges halmaza AP
- $\neg$ és $\vee$ logikai operátorok
- U_I (*Until*) temporális modális operátor, ahol I egy nem negatív számokból álló intervallum
- S_I (*Since*) temporális modális operátor
- F_I (*Finally*) temporális modális operátor
- G_I (*Globally*) temporális modális operátor

A pU_Iq („p *Until* q”) kifejezés akkor lesz igaz t időben, ha létezik olyan $t' \in I + t$ idő amire a következők teljesülnek:

- t' időben q igaz
- minden egyes $t'' \in T$ ahol $t < t'' < t'$ és $T \subseteq R_+$ időben p igaz

A pS_Iq („p *Since* q”) kifejezés akkor lesz igaz t időben, ha létezik olyan $t' \in I - t$ idő amire a következők teljesülnek:

- t' időben q igaz
- minden egyes $t'' \in T$ ahol $t' < t'' < t$ időben p igaz

A F_Iq („*Finally* q”) kifejezés akkor teljesül t időben, ha létezik olyan $t' \in I + t$ idő, amiben q teljesül. A G_Iq („*Globally* q”) kifejezés akkor teljesül t időben, ha minden $t' \in I + t$ időben q teljesül.

Például a $G(p \rightarrow F_1 q)$ *MTL* formula azt követeli meg, hogy minden olyan állapotot, amelyben p igaz pontosan egy időegységgel később egy olyan állapotnak kell követnie, amelyben q teljesül [9].

A *past-MTL* megfelel a teljes *MTL*-nek az *Until* operátor kivételével. Hasonlóan, a *future-MTL* a teljes *MTL* a *Since* operátor nélkül.

Az egyes cikkekben változó, hogy az *MTL*-t hogyan definiálják. Az *MTL* megegyezik az előbbi *future-MTL*-el vagy a teljes szintaxissal van definiálva.

2.2.3. MITL - Metric Interval Temporal Logic

Az *MITL* [12] egy részhalmaza az *MTL*-nek. Az *MITL* temporális logika nem képes a következő követelmény megadására: „A esemény pontosan öt időegység óta történt". Ezt gond nélkül megfogalmazhatjuk *MTL*-ben. *MITL*-ben csak a következőt tudjuk leírni: „A esemény négy és öt időegység között történt".

Például a $G(F_{0,1}p)$ *MITL* formula azt fejezi ki, hogy minden egy hosszú intervallumban megjelenik p [12].

A temporális logikák nagy hátránya az, hogy ha hosszú szekvenciális követelményeket szeretnénk leírni, akkor nagyon hosszú és bonyolult formulákat kell használnunk. Ezeket viszont nagyon nehéz megírni és nehezen olvasható ki belőlük a követelmény. A szcenárió alapú formalizmusok sokkal intuitívabbak és ilyen követelmények megadására sokkal alkalmasabbak. Egy szcenárió egy konkrét lefutást ír le, amíg a temporális logikák egy állapotára tudnak követelményt specifikálni.

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.

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* [10] az *MSC* egy kiterjesztése. Sok új elemet vezet be ami nincs az *MSC*-ben, melyek megtekinthetők a 2.1 táblázatban, mint például az üzenet típusokat: sima üzenet (e), elvárt üzenet (r) és nem kívánt üzenet (f). Így specifikálhatjuk, hogy mely üzenetek azok, amelyek helyes viselkedésre utalnak és melyek azok, amelyek nem. Szigorú sorrendezésre is ad megoldást a *PSC*, ami azt jelenti, hogy az üzenetek sorrendjét explicit megadhatjuk a követelményünkben. A gyenge sorrendezés és a szigorú sorrendezés között az a különbség, hogy szigorú sorrendezés esetén a szekvenciában lévő üzenetek között nem jelenhet meg más üzenet. Gyenge sorrendezésnél megengedjük, hogy az üzenetek között más üzenetek is megjelenjenek, amelyeket nem vettünk be a követelménybe. A *PSC*-ben egy üzenetre megkötést is tehetünk. Megadhatjuk, hogy melyek azok az üzenetek, amelyek nem kívántak az üzenetünk észlelése előtt vagy után. A különböző *PSC* tulajdonságok megtalálhatók a 2.3 ábrán.

A nyelv a következő tulajdonságokat támogatja:

- *Sima üzenet (e)*: egy üzenet a szcenárióban, amely ha nem történik meg, az a monitor szempontjából nem jelent hibát. Viszont ha megjelenik, akkor a szcenárióban 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 sem 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 specifikálására is lehetőséget ad, például egy „a" üzenet megtörténte után egy adott „b" üzenetnek kell bekövetkeznie. Ha „b" üzenet helyett egy másik üzenet követi az „a" üzenetet, akkor a monitor hibát jelez. Nem megkötött sorrendezés esetén nem jelent hibát, ha „a" és „b" üzenet között más üzenetek is megjelennek.

A nyelv támogat összetett szerkezeteket is:

- *Alt operátor*: az *alt* operátorral alternatív üzenet szekvenciákat lehet definiálni.
- *Par operátor*: a *par* operátorral megadható 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 a címkéjén 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 formájában 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.

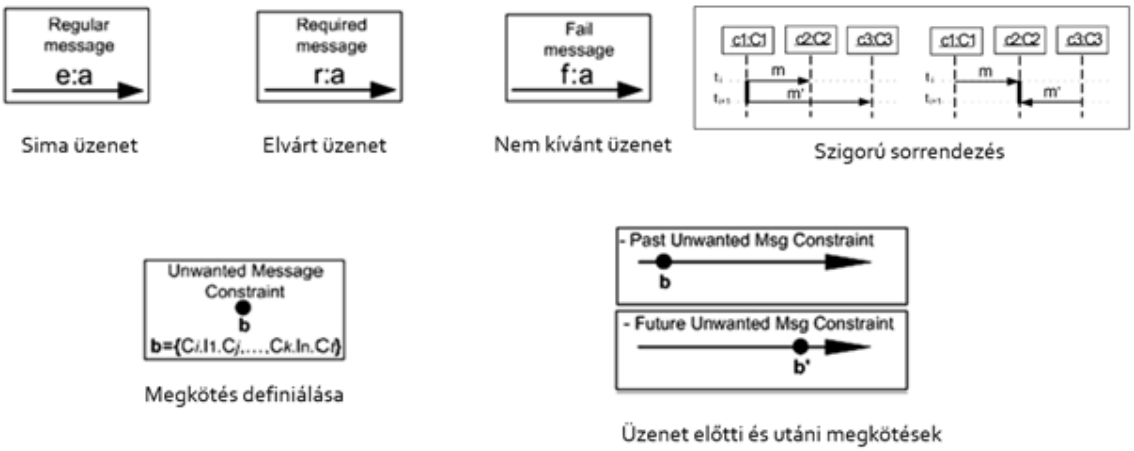
Az üzenetek megkötött sorrendezésének jelölésénél az objektum „*lifeline*" vonalát az érintett átmenetek közt folytonossá változtatjuk.

A 2.4 ábrán láthatunk egy példát arra, hogy egy követelményt hogyan lehet definiálni. Ez a *PSC* diagram egy *ATM* rendszer működését specifikálja. 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 történhet kijelentkezés, *logout*. Az *ATM* ezután, ha nem történt *logout*, egy elvárt üzenetet küld a *Bank* adatbázisába.

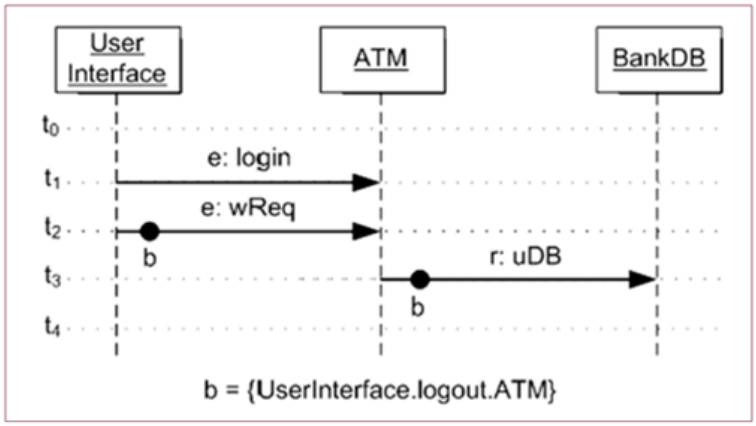
A szcenárióink specifikálására így rendelkezésünkre áll egy grafikus nyelv. A következőkben az lesz a feladatom, hogy ezekből a diagramokból valósítsam meg a monitor kódjának generálását.

2.3. A TPSC bemutatása

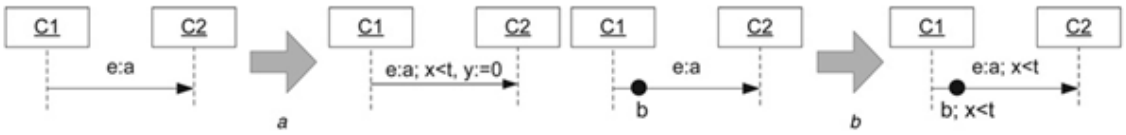
A *TPSC* [7] a *PSC*-nek egy kiterjesztése, melynek segítségével a *PSC* üzenetekre időzítési feltételeket specifikálhatunk.



2.3. ábra. A *PSC* különböző elemei [10].



2.4. ábra. *PSC* diagram egy *ATM* rendszer működésének ellenőrzésére [10].



2.5. ábra. *PSC* kiterjesztése időzítési feltételekkel[2].

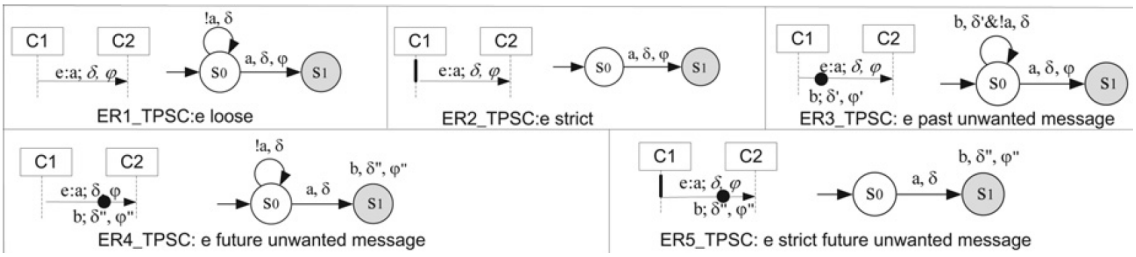
A *TPSC* óraváltozókat (pl. x, y) használ az időzítéshez. Ezekre meg lehet adni feltételeket, valamint az óraváltozót lehet nullázni. A nullázással adott eseménytől (pl. üzenet vételétől) kezdve indul az időzítés, majd rákövetkező események időbeliségét ez alapján lehet ellenőrizni.

A 2.5 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, vagyis ha nem történik meg a specifikált idő intervallumban, az nem jelent hibát. Viszont *r: a* üzenetnél ugyanilyen feltétel mellett már elvárt, hogy *t* időn belül megtörténjen. *f: a* üzenet esetében viszont akkor jelez hibát a monitor, ha az üzenet megtörtént *t* időn belül.

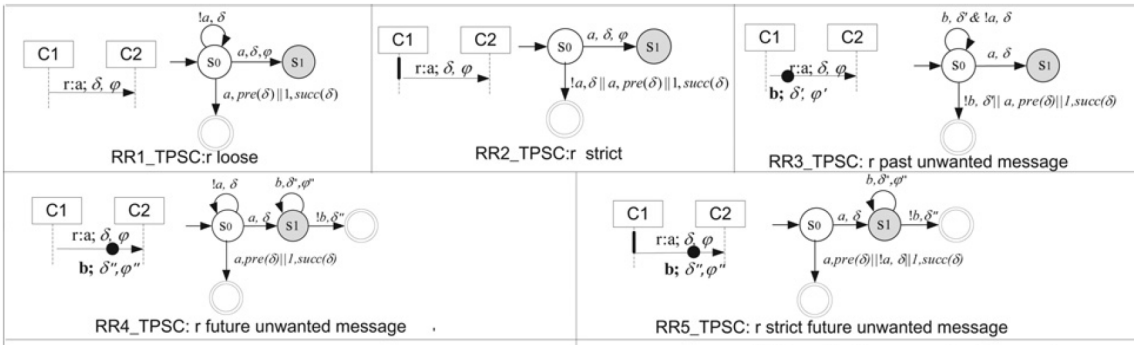
Egy múltbeli vagy jövőbeli megkötésre is meg lehet adni időzítési feltételt, így például megadhatjuk, hogy mennyi ideig nem szabad jönnie a megkötésben szereplő nem kívánt üzenetek egyikének.

2.4. TPSC szcenárió alapú automata konstrukció

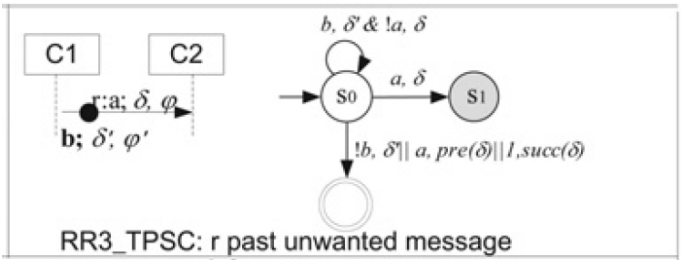
A monitorozás alapja, hogy *TPSC* szcenáriókból időzített automatákat (*Timed Automata*) tudjunk készíteni. Egy *TA* állapotokból, elfogadó állapotokból, feltételekből, akciókból és bemenetekkel címkézett állapotátmenetekből áll. Ezek a következő sorrendben jelennek meg az állapotátmeneteken: bemenet, feltétel és akció. A bemenet lehet egy konkrét esemény, ilyen például az „a” üzenet, vagy több eseményre utaló kifejezés. Például a „$!a$” kifejezés minden olyan üzenet, amelyik nem az „a” üzenet. Egy feltétel ebben az esetben egy időzítési feltétel, ezekhez óraváltozókat használhatók. Egy akció lehet például egy ilyen óraváltozónak a nullázása. Ha a feltétel teljesül és az adott üzenet illeszkedik az állapotátmeneten lévő bemenetre, akkor az megvalósulhat és a rajta lévő akció megtörténik. Az automata akkor fogad el egy bemenet sorozatot, ha ennek során elérünk az automata végállapotába. Ha hibaállapotot érünk el, akkor az a monitor szempontjából hibát jelent. Az alapelv az, hogy minden *TPSC* elemhez tartozik egy minta automata (*pattern*) [7], ami leírja a szemantikáját. Például a 2.6 és 2.7 ábrákon láthatók a különböző *PSC* üzenetekhez tartozó minták.



2.6. ábra. Sima üzenetekhez tartozó minták [7].



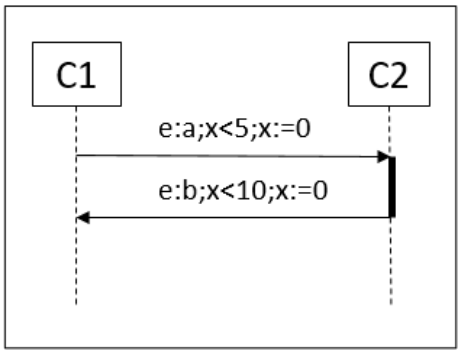
2.7. ábra. Elvárt üzenetekhez tartozó minták [7].



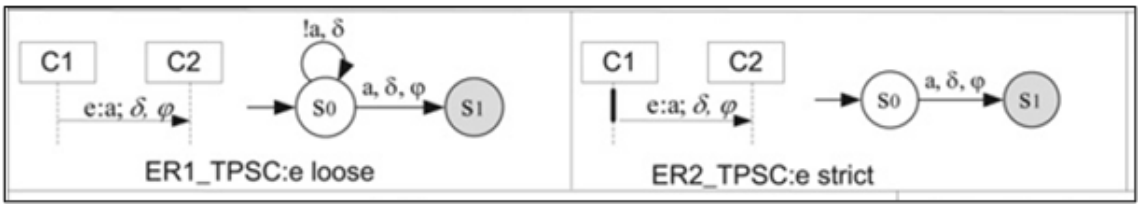
2.8. ábra. Elvárt múltbéli megkötést tartalmazó üzenethez tartozó minta [7].

A minta automatánkban található szürke állapotok reprezentálják a végállapotokat. Egy megkötés nem kívánt üzenetek negáltjainak az *ÉS* kapcsolata, az időzített automatán egy átmenet formájában jelenik meg. Például a 2.8 ábrán lévő automata mintán látható „b, δ'&!a, δ” címkéjű hurokélen, a „b” címke minden olyan üzenetnek felel meg, amelyek nincsenek a megkötésben lévő 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. A „a, δ” címkéjű átmenet az „s0” állapotból az „s1” végállapotba mutat. Az átmenet akkor valósul meg, ha „a” üzenet érkezik és „δ” időzítési feltétel igaz. Ebben az esetben teljesül a feltétel. Ha viszont a megkötésben szereplő nem kívánt üzenet érkezik és „δ’” igaz, vagy „a” üzenet érkezik „δ” előtt, vagy bármilyen üzenet érkezik „δ” után, akkor a hibaállapotba lépünk át.

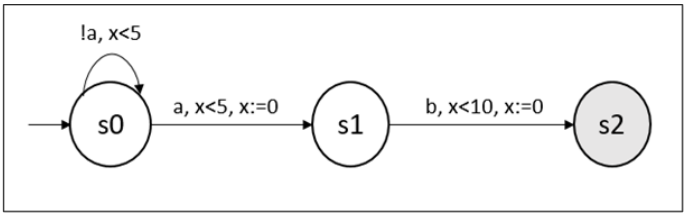
Ezekből az automata részekből lesznek meghatározott illesztési szabályokkal a szcenárióhoz tartozó teljes időzített automaták. Ennek az az alapelve, hogy a szcenárión végig menve az előző minta végállapotát a következő minta kezdőállapotával kell egyesíteni. Ezt a folyamatot mutatják be a 2.9, 2.10 és 2.11 ábrák. Ebben a példában a 2.10 ábrán lévő baloldali automata „s1” állapotát kellett egyesíteni a jobb oldali automata „s0” állapotával.



2.9. ábra. TPSC részlet.



2.10. ábra. Alkalmazott automata minták [7].



2.11. ábra. Az összeállított automata.

A monitor szempontjából az automata akkor jelez hibás működést, ha olyan bemenetet kap, amely egyik élre sem illeszkedik. 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 üzenet, amely elmozdítaná őt onnan. A 2.11 ábrán az „s2” szürke állapot a végsőállapot.

2.5. A felhasznált technológiák

2.5.1. Eclipse

Az *Eclipse* [2] 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 projektem során a következő *Eclipse plugin*-okat használtam::

- *Xtext* [15]
- *Xtend* [14]
- *Sirius* [13]

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

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

2.5.2. Xtext

Az *Xtext* [15] *Eclipse plugin*-nel 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 idézőjelek között kapcsos zárójel. Ilyen kulcsszavak megadásával jól tudjuk formázni a nyelvtani szabályunkat. Attribútumok 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 szerűen. Ezt a * karakterrel adhatjuk meg, például *scenariocontents+=ScenarioContent**, ahol a *scenariocontents* több *ScenarioContent*-beli elemet tárol. A += karakterrel pakolhatjuk az új *ScenarioContent*-jeinket a *scenariocontents* tömbbe.

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 opcionális nyelvi elemek megadására szolgál.

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

2.5.3. Xtend

Az *Xtend* [14] egy magasszintű programozási nyelv, amely a *Java Virtual Machine* platformot használja. Szintaktikailag és szemantikailag nagyon hasonlít a *Java* nyelvhez, mondhatni a *Java* kibővítése. Az *Xtend* fordító a létrehozott osztályokból *Java* osztályokat generál. Így akár könnyedén ki tudjuk szervezni a projektünket egy *.jar* fájlba.

Egy nagyon hasznos funkciója az *Xtend*-nek a *template*, amely segítségével sablonokat hozhatunk létre a függvényeken belül, így megkönnyítve a kódgenerálást. Így 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.

```
def compile(Scenario scenario)'''
   FOR  sc : scenario.scenariocontents
     FOR  m : sc.message
       generateMessage(m)
       a.collapse(b);
      ENDFOR
    ENDFOR
'''
```

2.2. kódrészlet. Xtend template.

A 2.2 ábra egy *Xtend* sablon függvényt mutat be. Láthatjuk, hogy a "«»" karaktereket használva hivatkozhatunk a paraméterben átadott *Xtext* nyelv béli objektum attribútumaira. Itt például a *Scenario scenariocontents* tömbjén megyünk végig.

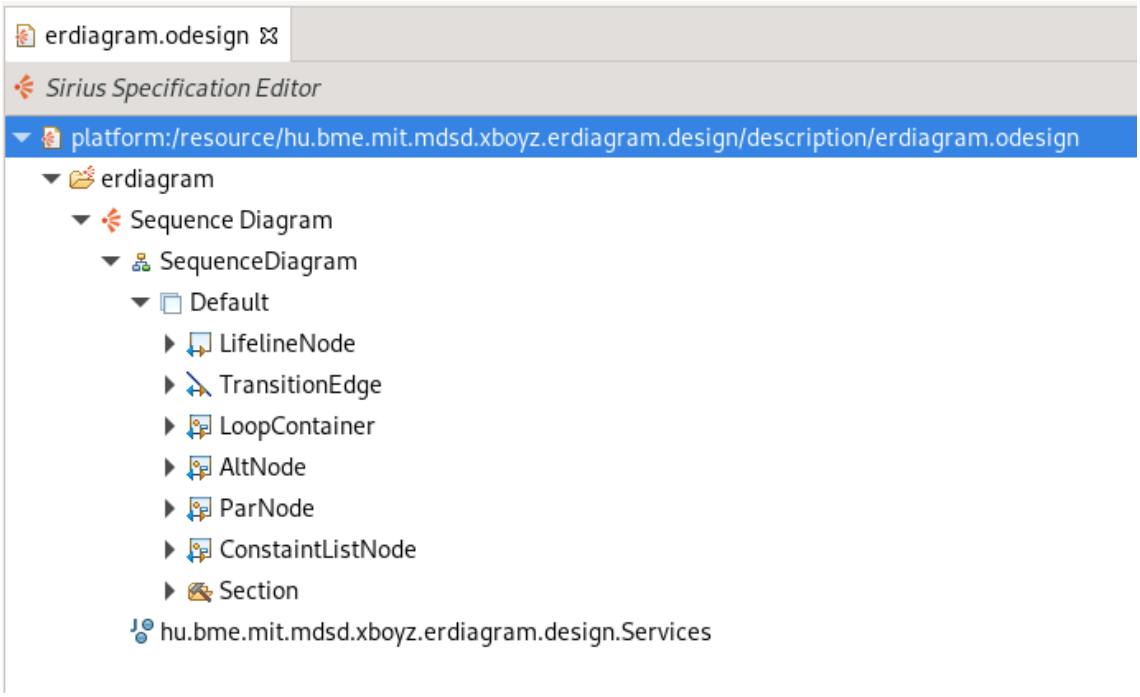
2.5.4. Sirius

A *Sirius* [13] 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* [4] keretrendszert használja a modellek feldolgozásához és ilyen *EMF* modelleket jelenít meg. Ehhez az *EMF* modellünk összes eleméhez meg kell adnunk egy megjelenítést.

Elkészíthetjük saját egyedi szerkesztőnket, de választhatunk a *Sirius* által nyújtott szerkesztőkből is:

- Diagram
- Szekvencia diagram
- Táblázat

Az egyes modell elemeinkhez tartozó grafikus megjelenítést egy *viewpoint* létrehozásával tudjuk megtenni. A 2.12 ábrán látható egy példa *viewpoint*. A *viewpoint*-hoz hozzá fűzhetünk egy *representation*-t amely a konkrét megjelenítésünket tartalmazza. Itt adhatjuk meg, hogy egyes modell elemeink milyen formában jelenjenek meg.



2.12. ábra. *Sirius* alkalmazáshoz tartozó *viewpoint*.

Emellett alkalmazhatunk feltételes megjelenítést is adott típusú modellelemünkre, ha az adott tulajdonságban eltér a többi elemtől.

3. fejezet

Szöveges PSC leíró nyelv kibővítése

A szakdolgozatom során készített szöveges *PSC* leíró nyelvet [1] az *Xtext* technológia segítségével definiáltam. A nyelv fő elemei:

- objektumok (az üzenetekkel kommunikáló rendszerkomponensek)
- üzenetek
- megkötések
- szcenárió
- *alt* operátor
- *loop* operátor
- *par* operátor

A nyelvben egy objektumot az „*object*" kulcsszóval lehet bevezetni. Meg kell adni az objektum típusát és a nevét, ezután a definíciót egy ';'-vel lehet lezárni. Megkötések specifikálásához a „*constraint*" kulcsszót kell használni. Elég csupán a megkötés nevét megadni és '{' '}' jelek közt, a megkötéshez tartozó üzeneteket.

Egy sima üzenet megadásához a „*message*" kulcsszó szükséges. A „*message*" kulcsszó előtt lévő „*fail*" vagy „*required*" kulcssavakkal adhatunk meg nem kívánt vagy elvárt üzeneteket. A „*strict*" kulcsszóval adhatjuk meg, hogy az üzenet sorrendje megkötött. Az üzenet küldőjét és fogadóját „*<küldő> -> <fogadó>*" formában jelezzük. Múltbéli vagy jövőbeli megkötéseket a „*pastConstraint*" vagy „*futureConstraint*" kulcsszavakkal adhatunk meg. A kulcsszó után, '{' '}' jelek közt adhatjuk meg, hogy melyik előzőleg definiált megkötést helyezzük el az üzeneten. Egy üzenet definíció végére szükséges ';'-t tenni, így elválasztjuk a többi üzenettől.

A nyelvet két új elemmel bővítettem ki:

- időzített feltétel
- óraváltozó nullázása

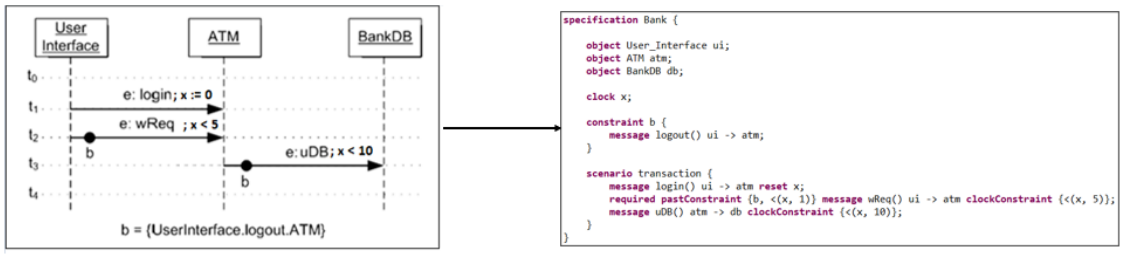
Az új nyelvi elemek bevezetéséhez az eredeti üzenet leíró *Xtext* szabályt (*Message*) kellett kibővítenem. A kibővített nyelvtani szabályt a 3.2 kódrészlet tartalmazza. Az eredeti nyelvtani szabályt a 3.1 kódrészlet tartalmazza. Bevezettem egy új *clockConstraint* kulcsszót, amivel egy időzítési feltételt lehet megadni. Ezt a *cConstraint* változóban tárolom. A

```
Message:
    'message' name=Name (required?='required')? (fail?='fail')? (strict?='strict')?
    sender=[Object] '->' receiver=[Object]
    (past?='past')? (future?='future')? (constraint?='constraint')?
    ('{')? (c=[Constraint])? ('}')? ';'

;
```

3.1. kódrészlet. Eredeti *Message* nyelvtani szabály [1].

```
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)? ';'
;
```

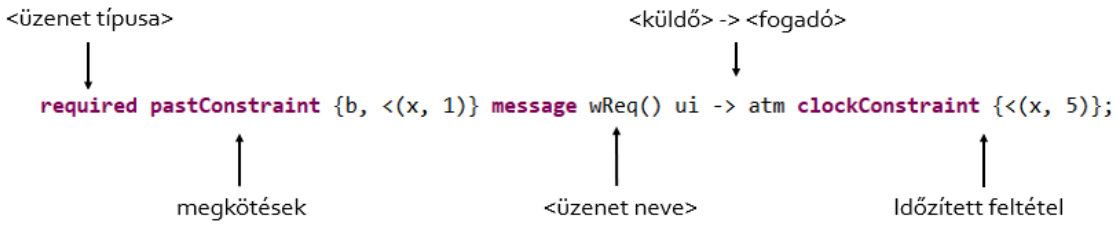
3.2. kódrészlet. Kibővített *Message* nyelvtani szabály.



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

resetClock attribútum tárolja a visszaállítandó óraváltozó nevét. A teljes nyelvtan definíció megtekinthető a *Függelékben* (F.4.1).

A 3.1 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 3.2 ábrán látszik, hogy megjelenik a *clockConstraint* kulcsszó ami egy időzítési feltétel megadására szolgál. A kulcsszó után kapcsos zárójelek közt megadható a feltétel. Egy időzítési feltételt egy adott óraváltozóra adhatunk meg. Négy operátort definiál ehhez a nyelv:

- < (kisebb)
- > (nagyobb)
- <= (kisebb egyenlő)
- >= (nagyobb egyenlő)

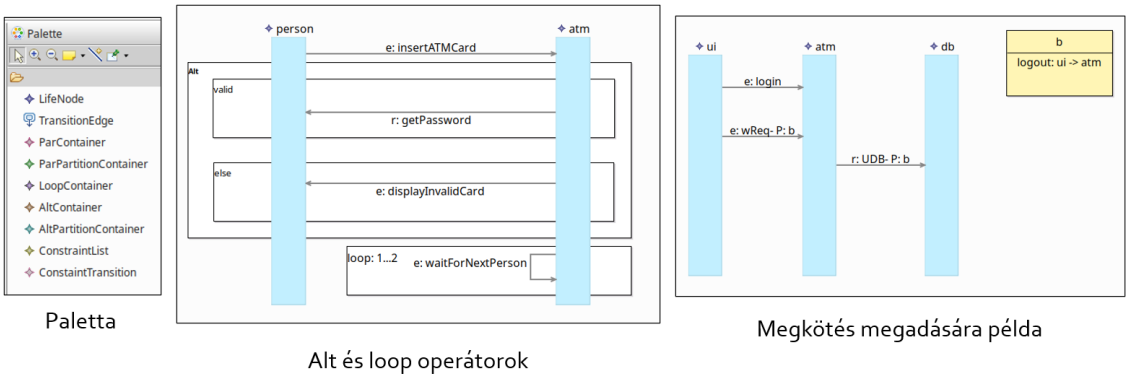
Az operátor után '(' ')' jelek között adható meg az óraváltozó, majd utána az érték. Például a *<(x, 5)* kifejezés azt jelenti, hogy az x óraváltozóban lévő időértéknek kisebbnek kell lennie öt időegységnél.

Időzítési feltételt egy megkötésre is megadhatunk, '{' '}' jelek között, ','-vel elválasztva a megkötés után. 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. Az alkalmazással egy szerkesztő segítségével tudunk *PSC* diagramokat rajzolni. Egy palettáról lehet kiválasztani az egyes elemeket és az egérrel elhelyezni őket a diagramon. Ezután pedig megadhatók a paramétereik, például egy üzenetnél, hogy melyik objektum a küldője és fogadója, vagy hogy milyen megkötések vannak rajta, egy objektumnál, hogy mi a neve és típusa. A 4.1 ábrán megtekinthető a paletta és néhány példa diagram.



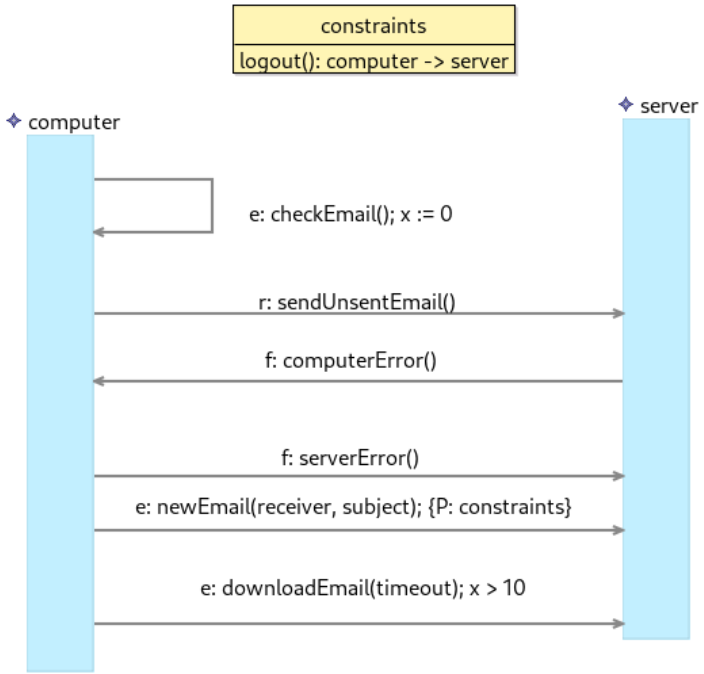
4.1. ábra. *PSC* vizualizációs alkalmazás bemutatása.

Először kiegészítettem az alkalmazást *TPSC* elemek vizualizációjával. Ehhez az alkalmazáshoz tartozó *EMF* modellben fel kellett vegyem, hogy egy üzeneten elhelyezhető legyen időzítési feltétel is. Ezt ugyanígy meg kellett tegyem a megkötéseknél is és ezen kívül az óraváltozók nullázását is hozzáadtam a modellhez. Ezután a vizualizációhoz hozzáadtam, hogy az időzítési feltétel és az óraváltozó nullázása megjelenjen az átmenetek címkéin. Készítettem egy *Xtend* alapú *XML* generátort, ami képes a szöveges *TPSC* leírás alapján legenerálni a hozzá tartozó *XML* leírást. Ezt a leírást képes a *Sirius* alkalmazás feldolgozni és előállítani a hozzá tartozó diagramot. Egy ilyen generált *XMl* fájl megtekinthető a 4.2 kódrészleten.

A vizualizációs diagramon az egyes objektumok kék *lifeline* formájában jelenek meg. Minden üzenethez tartozik egy nyíl, amely címkéjére írjuk az üzenet összes tulajdonságát. Például a megkötések '{' '}' jelek között jelennek meg. Egy *P* betű jelöli, hogy a megkötés múltbéli és egy *F* betű ha jövőbeli. 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. Egy

```
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)};
    }
}
```

4.1. kódrészlet. Szcenárió szöveges leírása.



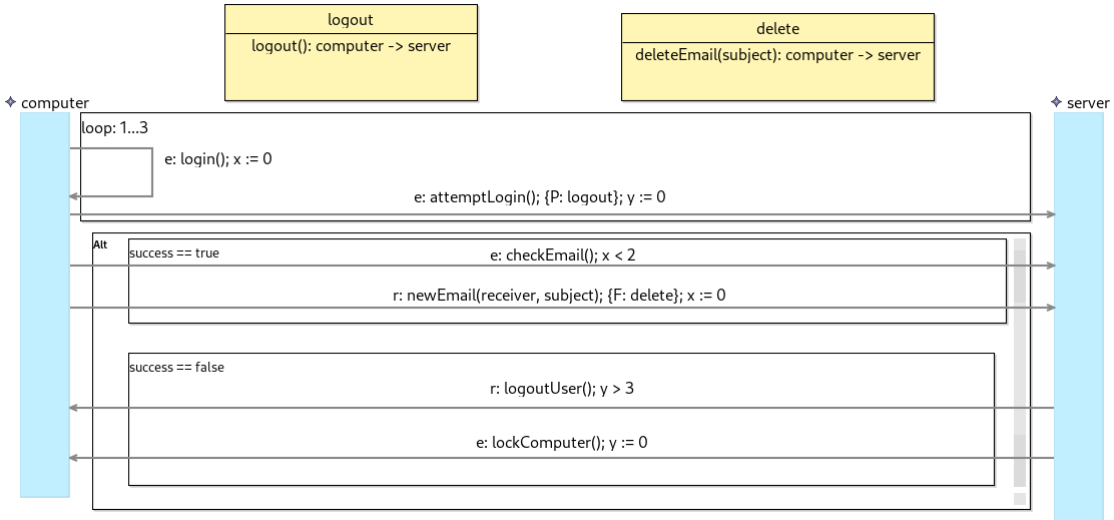
4.2. ábra. *Sirius* által előállított szcenárió diagram.

ilyen diagramot mutat be a 4.2 ábra. A diagram szöveges leírását pedig a 4.1 kódrészlet tartalmazza.

A diagramon keretek formájában jelenek meg az operátorok. Ilyen operátorokat a 4.3 diagramon lehet megtekinteni.

```
<?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>
```

4.2. kódrészlet. Szenárió diagram xml leírása.



4.3. ábra. Operátorokat tartalmazó szcenárió 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. Az eredeti automata generátor *PSC* szöveges leírásokhoz tartozó *Büchi* automatákat tudta összeállítani. Az volt a feladatom, hogy a meglévő *Büchi* automata mintákat lecseréljem a korábban ismertetett időzített automata mintákra és felvegyem a hiányzó mintákat, amelyek az időzítési feltételeket tartalmazó üzenetekhez tartoznak. A kibővített automata generátor bemenetként egy *TPSC* szcenárió szöveges leírását kapja meg, amelyből a minta alapú módszerrel generál egy *TA* automatát.

A 5.1 és 5.2 kódrészleteken látható, hogy a monitor generátor támogatja az összetett operátorokat tartalmazó *TPSC*-khez tartozó *TA*-k generálását is. A generátor készít egy szöveges leírást a generált automatához, ehhez a *Never claim* nyelvet használja. A *Never claim* [11] a *Promela* nyelv része, ezzel egy rendszer viselkedését lehet definiálni. Egy *Never claim* leírás a *never* kulcsszóval kezdődik és '{' '}' jelek között vannak az automata állapotai lista szerűen felsorolva egymás alatt. Minden állapothoz tartozó kimenő átmenet az *if* és *fi* kulcssavak között található. Egy átmenet leírása a címkéjét tartalmazza és egy -> jelzi, hogy melyik állapotba visz át. A 5.1 kódrészleten lévő szöveges *TPSC* leíráshoz tartozó generált automata *Never claim* leírását a 5.2 kódrészlet tartalmazza. Továbbá a generátor képes az üzenet paraméterek kezelésére.

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:

- *Automaton*, az automata implementációját tartalmazza, eltárolja az állapotokat és az átmeneteket
- *ClockConstraint*, időzítési feltételek *Java* implementációját tartalmazza
- *Constraint*, megkötések implementációját tartalmazza
- *State*
- *StateType*

```
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 szcenárió.

- Transition

Az automatában lévő állapotok implementációja a *State* osztályban található. Két attribútuma van:

- id(*String*): az állapot címkéje
- type(*StateType*): az állapot típusa

Az állapot típusának a megadására a *StateType enum* osztályt definiáltam. Az *enum* a következő értékeket veheti fel:

- NORMAL, egy sima állapotot jelöl
- ACCEPT, elfogadó állapot jelöl (a monitor szempontjából ez egy hibaállapot)
- FINAL, a végállapotot jelöli

Az átmenetek implementációjáért felelős osztály a *Transition*. Három tagváltozója van:

- id(*String*): az átmenet címkéje
- reset(*String*): visszaállítandó óraváltozó neve
- sender(*State*): a forrás állapot
- receiver(*State*): a cél állapot
- constraint(*Constraint*): az átmeneten lévő megkötés
- clockConstraint(*ClockConstraint*): az átmeneten lévő időzítési feltétel

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

```
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 szcenárió never claim leírása.

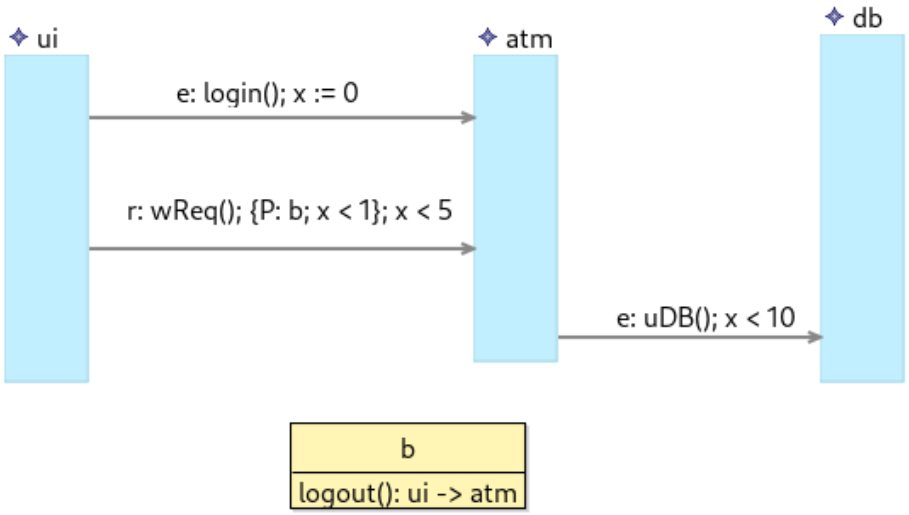
új állapotot és átmenetet hozzáadni az automatához, a *collapse(Automaton)* függvényével pedig két automatát egyesíteni. Ezt a függvényt használtam az implementációban a minta automaták egyesítésére, ezen kívül az osztálynak van egy *merge(ArrayList<Automaton>)* függvénye. Ez a függvény az *alt* operátor ágaiban lévő minta automaták összefésülésére szolgál.

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

5.1.3. Mintapélda

```
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.



5.1. ábra. Példa TPSC diagram.

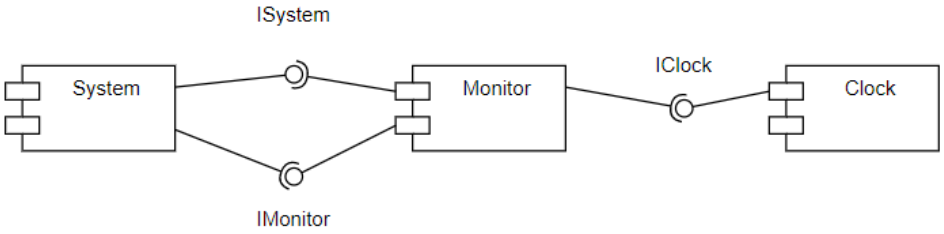
A 5.3, 5.4 kódrészleteken és 5.1 ábrán látható, hogy a generátor milyen időzített automatát generál a megadott *TPSC* szcenárióból.

```
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



5.2. ábra. Monitor architektúra diagram.

A 5.2 ábrán látható a megfigyelt rendszer és monitor architektúra diagramja. A monitorozás alatt lévő rendszer egy *IMonitor* interfészen keresztül kommunikál a monitorral. Az interfész *Java* implementációja a 5.5 kódrészleten tekinthető meg. A monitor azt vizsgálja, hogy a rendszer a szcenárió szerint viselkedik-e. A monitornak a következő interfészt kell megvalósítania:

- *update()*: ezzel a függvénnyel lehet továbbítani a rendszer üzeneteit a monitor számára. Bemenetként az üzenet feladóját, címzettjét, nevét és paramétereit kapja. A monitor frissíti a belső automatájának állapotát.
- *goodStateReached()*: a rendszer ezen a függvényen keresztül kérdezi le a monitortól, hogy jó állapotban van-e a követelmény szempontjából, azaz nem történt-e hiba. Visszatérési értéke egy *boolean* érték.
- *requirementSatisfied()*: a rendszer ezzel a függvénnyel kérdezheti le, hogy megfelel-e a követelménynek. Visszatérési értéke egy *boolean* érték.
- *errorDetected()*: ez a függvény tartalmazza a hibaelhárító funkcionalitás implementációját. A rendszer ezt a függvényt használhatja hibaelhárításra hiba detektálás esetén. Bemenetként azt az üzenetet kapja meg, amely hatására előjött a hiba az *update()* függvényhez hasonlóan.
- *noMoreMessages()*: a rendszer ezen a függvényen keresztül jelzi a monitornak a kommunikáció végét.

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, Map<String, Object>
     parameters);
  public boolean requirementSatisfied();
  public void errorDetected(String sender, String receiver, String messageType, Map<String, Object>
     parameters);
  public void noMoreMessages();
}
```

5.5. kódrészlet. Monitor interfész Java implementációja.

Az időzítési feltételek kiértékelése érdekében a monitornak szüksége van egy időzítő komponensre. Az időzítő komponenshez tartozik egy időzítő interfész amin keresztül el-

érhető a komponens. Ezen az interfészen keresztül lehet az óraváltozókat 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. Időzítő interfész Java implementációja.

Az 5.6 kódrészlet tartalmazza az időzítő interfész implementációját, az *IClock Java* interfészt.

A monitor az *ISystem* interfészen keresztül tud a rendszernek üzeneteket küldeni a megfigyelt viselkedésről. Ezt az interfészt a monitorozott rendszer valósítja meg. Három függvénye van:

- *receiveMonitorStatus()*: a monitor jelzi a rendszer felé a követelmény alapján az aktuális státuszt. Bemenetként a monitor által küldött üzenetet kapja meg.
- *receiveMonitorError()*: a monitor jelzi a rendszer felé, ha hibát detektált. Bemenetként két üzenetet kap: a hibát okozó üzenetet és az utolsó üzenetet, amikor még a rendszer jó állapotban volt a követelmény szempontjából.
- *receiveMonitorSuccess()*: a monitor jelzi a rendszer felé, ha teljesült a követelmény.

```
public interface ISystem {
  public void receiveMonitorStatus(String message);
  public void receiveMonitorError(String actualMessage, String lastAcceptedMessage);
  public void receiveMonitorSuccess();
}
```

5.7. kódrészlet. Rendszer interfész Java implementációja.

Az 5.7-es kódrészlet tartalmazza az *ISystem 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 időzített automata *Java* osztályai kerülnek:

- State: egy állapotot leíró osztály
- Transition: egy átmenetet reprezentáló osztály
- Automaton: egy automatát megvalósító osztály

Ezek segédosztályok, amelyek a generált automata reprezentálására szolgálnak. A statikus rész a *hu.bme.mit.dipterv.text.util* csomagban található meg, amely a 5.3 ábrán látható.

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 szcenárió alapján generált automata forráskódját tartalmazza. Az osztály konstruktorában jön létre egy *Automaton* objektum, amihez hozzáadjuk a leírás alapján a megfelelő állapotokat és átmeneteket *State* és *Transition* objektumokat használva.

A szükséges forráskódok generálásához az *Xtend* technológiát használtam.

5.3. ábra. Az *util* csomag tartalma.

5.2.3. Mintapélda

A 5.8 kódrészleten látható egy szcenárió követelmény szöveges leírása, amit egy okostelefon működésére specifikáltunk. A 5.4 ábrán látható a leíráshoz tartozó diagram vizualizációja, az 5.9 kódrészlet pedig a generált automata leírását tartalmazza.

Az okostelefonon van egy zene lejátszási lista generáló alkalmazás. A követelményben azt várjuk el, hogy ha a felhasználó megnyitja az alkalmazást akkor az előlapi kamera készít az arcáról egy képet. A kép alapján eldönti, hogy milyen a felhasználó kedve és az alapján előállít egy zene lejátszási listát.

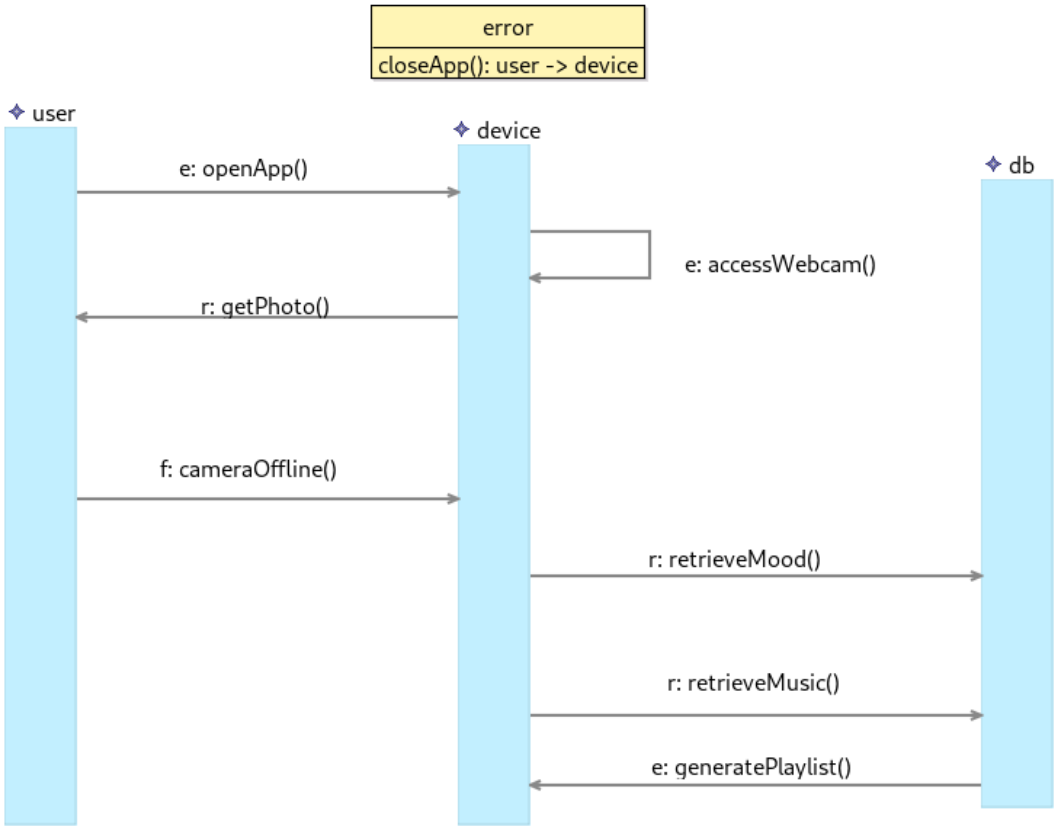
A 5.10 kódrészleten látható az okostelefon és a monitor közti kapcsolat megvalósítása *Java* kódban. A rendszerben lévő *User*, *Device* és *Database* osztályok mind attribútumként átveszik a *Monitor* osztályt, amely megvalósítja az *IMonitor* interfészt.

```
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;
  }
}
```

5.8. kódrészlet. Okostelefon működésére megadott szcenárió követelmény.

A 5.11 kódrészlet az okostelefon *Java* osztálya. Megtekinthető a monitor és az eszköz közti kommunikáció megvalósítása is. A *Device* osztály a tágváltozójaként tárolt *IMonitor*-nak küldi az üzeneteket az *update()* függvény használatával.

A 5.12 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 rendszer jó állapotban van a „*Good state*” üzenettel és, hogy a követelmény teljesült a "*Requirement Satisfied*" üzenettel. A mintapéldához tartozó *Specification* osztály a *Függelékben* található (F.1.1), amelynek a konstruktorában található a generált időzített automata.



5.4. ábra. A 5.8 leíráshoz tartozó diagram.

```
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;
}
```

5.9. 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();
  }
}
```

5.10. 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 HashMap<String, Object>());
    accessWebcam();
  }

  void accessWebcam() {
    monitor.update("device", "device", "accessWebcam", new HashMap<String, Object>());
    user.getPhoto();
    db.retrieveMood();
    db.retrieveMusic();
  }

  void cameraOffline() {
    monitor.update("user", "device", "cameOffline", new HashMap<String, Object>());
  }

  void generatePlaylist() {
    monitor.update("db", "device", "generatePlaylist", new HashMap<String, Object>());
  }
}
```

5.11. kódrészlet. Az okos telefon Java osztálya.

```
Received Message: user.openApp().device
q1
[System] Received status from monitor: System is in good state.
[Clock] reset x
Received Message: device.accessWebcam().device
q2
[System] Received status from monitor: System is in good state.
Received Message: device.getPhoto().user
q5
[System] Received status from monitor: System is in good state.
Received Message: someone.message().else
q7
[System] Received status from monitor: System is in good state.
Received Message: device.retrieveMood().db
q9
[System] Received status from monitor: System is in good state.
Received Message: device.retrieveMusic().db
q11
[System] Received status from monitor: System is in good state.
Received Message: db.generatePlaylist().device
q12
[System] Received status from monitor: System is in good state.
[System] Received status from monitor: Requirement satisfied
```

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

5.2.4. Összetett szerkezetek

A monitor forráskód generátor támogatja az *alt*, *par* vagy *loop* operátorokat tartalmazó szcenáriókat is. Ezt funkcionalitást a tesztelés során mutatom be részletesebben.

5.2.5. Időzítési feltételek

A monitor forráskód generátor támogatja az időzítési feltételeket tartalmazó szcenáriókat is. A 5.13 kódrészletben található szcenárió első üzenetén a "*reset x*" címke jelzi a monitornak, hogy az "*x*" óraváltozót nullázni kell. Egy óraváltozó felvételét is a "*reset*" címkével lehet végrehajtani. Ezt az óraváltozót használhatjuk majd a szcenáriónk későbbi üzeneteinél időzítési feltételek megadására. Például a 5.13 kódrészletben lévő szcenárió esetén, ha a monitor megkapja a *checkEmail()* üzenetet, akkor létrehozz egy "*x*" óraváltozót mivel ilyen még nem létezett előtte. Ha a szcenárió végére érünk és a monitor megkapja a *downloadEmail()* üzenetet, akkor a monitor kiértékeli az üzeneten lévő időzítési feltételt az óraváltozóban tárolt időérték alapján. Ha az előbbi feltétel teljesült akkor tovább lépteti az időzített automatát. A 5.14 kódrészlet a példához tartozó *Main Java* osztály leírását tartalmazza, a 5.15 kódrészlet pedig a rendszerhez tartozó *Computer Java* osztályt. A 5.16 kódrészleten megtekinthető a monitor kimenete. A kimenet végén lévő "*System is in good state.*" üzenet jelzi, hogy a rendszer jó állapotban van.

```
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};
  }
}
```

5.13. kódrészlet. Időzítési feltételeket tartalmazó szcenárió

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 óraváltozóban tárolt időt és kiértekeli a feltételt. Ha a feltétel teljesül, akkor az időzítés szempontjából a tranzíció megtörténhet. A monitor akkor jelez hibát, ha az átmeneten lévő időzítési feltétel nem teljesült és az üzenet más átmenetre sem illeszkedik.

```
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);
  }
}
```

5.14. 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 HashMap<String, Object>());
    checkEmail();
  }

  void checkEmail() {
    monitor.update("computer", "server", "sendUnsentEmail", new HashMap<String, Object>());
    server.sendUnsentEmail();

    monitor.update("computer", "server", "newEmail", new HashMap<String, Object>());
    server.newEmail();

    monitor.update("computer", "server", "downloadEmail", new HashMap<String, Object>());
    server.downloadEmail();
  }
}
```

5.15. kódrészlet. A Computer java osztálya.

```
Received Message: computer.checkEmail().computer
q1
[System]Received status from monitor: System is in good state.
Received Message: computer.sendUnsentEmail().server
q3
[System]Received status from monitor: System is in good state.
Received Message: computer.newEmail(receiver, subject).server
q4
[System]Received status from monitor: System is in good state.
Received Message: computer.downloadEmail(timeout).server
q5
[System]Received status from monitor: System is in good state.
[System]Received status from monitor: Requirement satisfied
```

5.16. kódrészlet. Időzítéses példa monitor kimenete.

6. fejezet

A generált monitor forráskód helyességének tesztelése

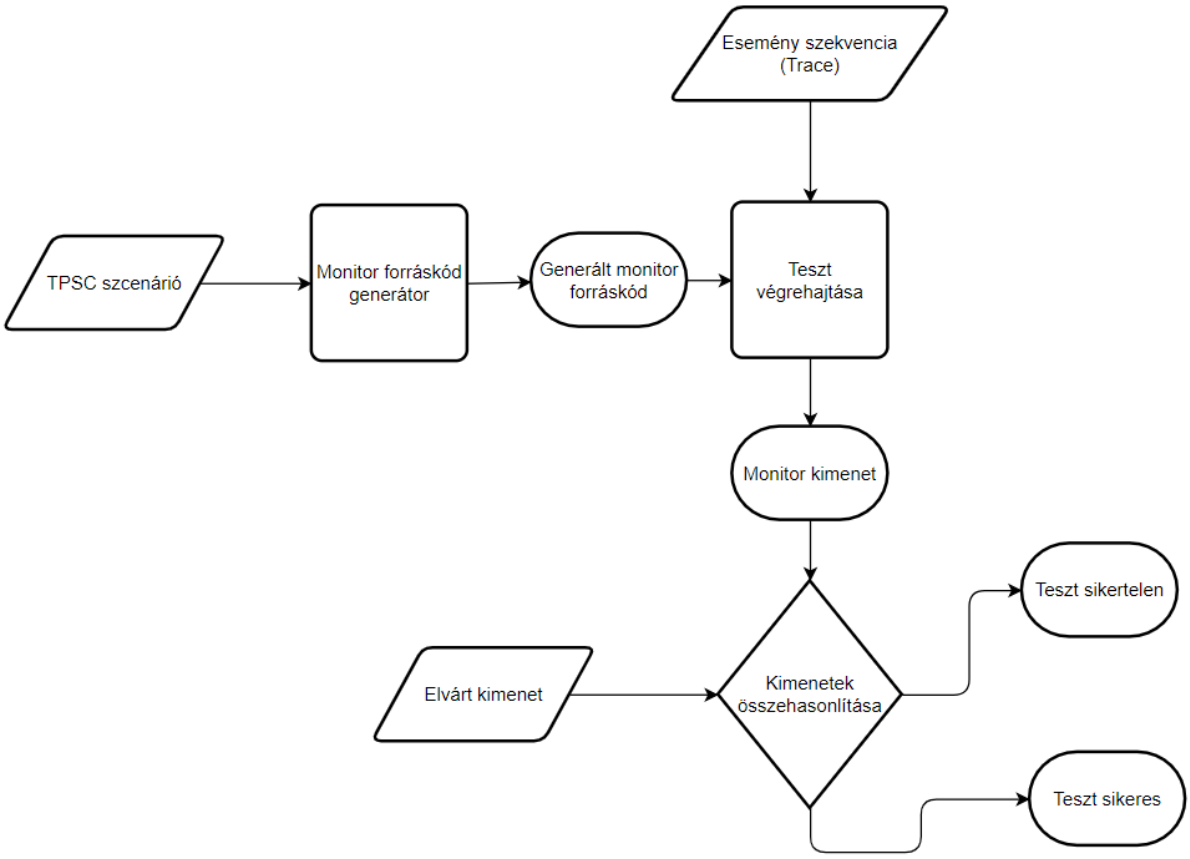
6.1. Tesztelési célok

A generált monitor forráskód tesztelésére a következő célokat fogalmazzuk meg:

- Az összes üzenet típus megjelenése a különböző teszt szcenáriókban
- 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 szcenárió tesztelése, ami több operátort tartalmaz
- Egymást követő elvárt üzeneteket tartalmazó szcenárió tesztelése
- Egymást követő *fail* üzeneteket tartalmazó szcenárió tesztelése
- *Regular* üzenet tesztelése (ha megjelenik a rendszer működésében, akkor ki kell értékelni a szcenárió többi részét, ha nem jelenik meg, a monitor helyes működést kell jelezzen és hogy a követelmény nem teljesült)
- Több óraváltozót tartalmazó szcenárió tesztelése
- Egymást követő különböző típusú üzenetek kombinációinak tesztelése (pl. elvárt üzenetet követő *fail*, elvárt üzenetet követő reguláris, stb.)

6.2. Monitor forráskód generátor tesztelése

Egy teszt bemenetként egy *TPSC* leírást, egy szimulált esemény szekvenciát és egy elvárt kimenetet fog kapni. A tesztelt monitornak kézzel küldünk üzeneteket, konkrét esemény lefutását. A monitor kimenetét összehasonlítjuk az elvárt kimenettel, amelyet kézzel állítunk össze figyelembe véve azt, hogy a szimulált rendszer működésének elméletben meg kell-e felelnie a követelménynek vagy sem. Ez két információt tartalmaz, hogy a rendszer jó állapotban van-e és teljesíti-e a követelményt. Ha egyezik akkor sikeres a teszt, ellenkező esetben pedig sikertelen.



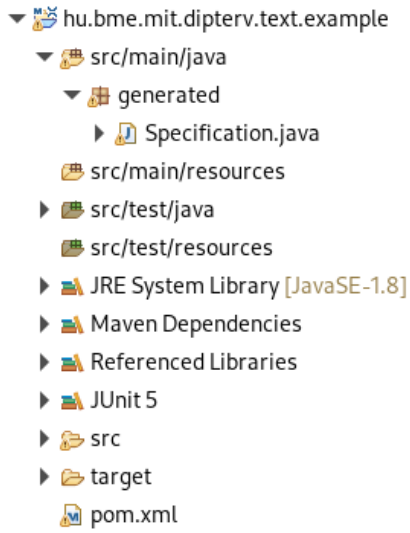
6.1. ábra. Tesztelés folyamatábrája.

Az *Xtext* keretrendszer a specifikált *DSL (Domain Specific Language)* nyelvhez generál egy *Maven* [3] *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 szcenárióinkat.

Az esemény szekvenciában lévő üzeneteket a monitor *update()* függvényénét használva adjuk át neki. A teszteset végén pedig a *JUnit* tesztelési keretrendszer *Assertions* osztályát használjuk, hogy a monitor kimenetét összehasonlítsuk az elvárt kimenettel. A teszteredmény ennek az összehasonlításnak az eredménye lesz.

A 6.1 ábrán megtekinthető a tesztelés tervének folyamatábrája.

Az *Xtext* keretrendszer által nyújtott *Maven plugin*-t felhasználhatjuk a tesztjeinkhez. Elég csupán egy *Maven* projektet felkonfigurálni a saját *DSL plugin*-ünkkel és elkészíthetjük a saját tesztelési keretrendszerünket. A keretrendszerünk tartalmazza a tesztszcenárióhoz készített *TPSC* szöveges leírásának a fájlját, a monitor forráskód generátort *Maven plugin* formájában és egy *Java* osztályt, amely a teszteseteket tartalmazza. Ezen kívül függőségként még be kell töltenünk az *util* csomagot, hogy hozzáférhessünk a monitor statikus forráskódjához is. Ezek a *Maven* projektek a szülő projektünkben helyezkedhetnek el, így a projekt struktúrában közvetlen a nyelvünk mellett vannak. A 6.2 ábrán látható egy ilyen teszthez tartozó *Maven* projekt felépítése



6.2. ábra. Példa *Maven* teszt projekt struktúrája.

A 6.2 ábrán lévő *generated* csomag tartalmazza a szcenárióhoz tartozó generált automata forráskódját.

A 6.1 kódrészlet a *Maven* teszt kimenetét tartalmazza. Látható a kimenet végén lévő összegzésen, hogy egy teszt futott le, amely sikeres volt. Ezen kívül megtekinthetőek a monitor működésének részletes folyamatai, például hogy milyen üzeneteket kapott, milyen állapotokba lépett át és egyes állapotoknak milyen kimenő átmeneteik voltak és a bejövő üzenet melyikre illeszkedett.

Ezt a projekt struktúrát felhasználva a tesztjeink köré tudunk egy *Maven* alapú *Continuous Integration*-t (*CI*) állítani a generált monitor forráskód folyamatos ellenőrzése érdekében.

```
-------------------------------------------------------
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
```

6.1. kódrészlet. Teszteset eredménye.

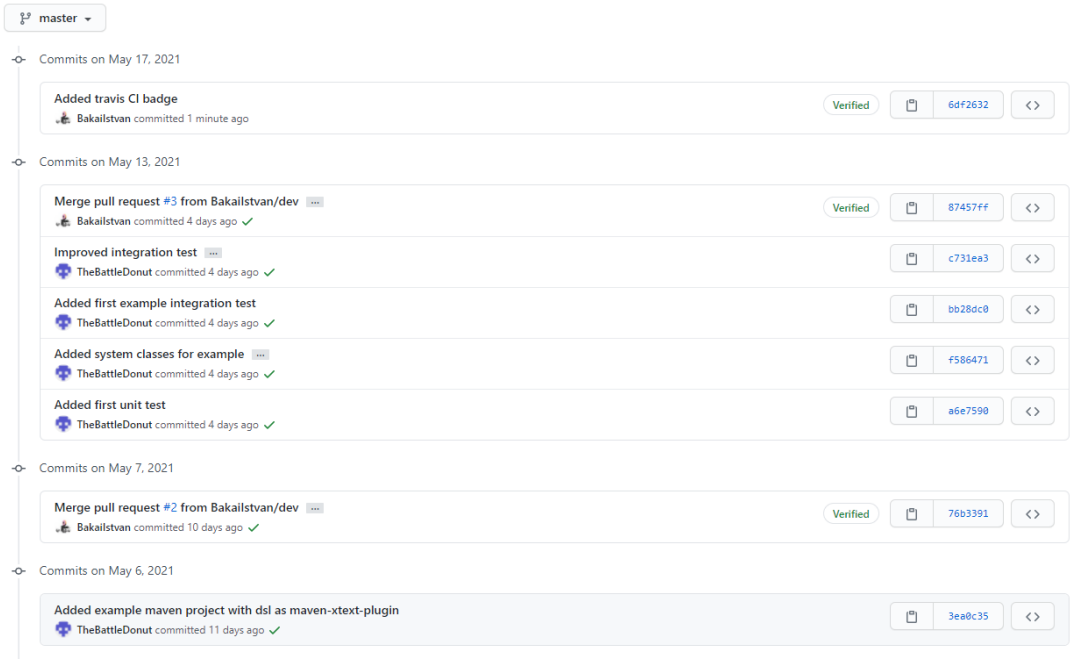
6.3. Continuous Integration

6.3.1. Github Actions CI

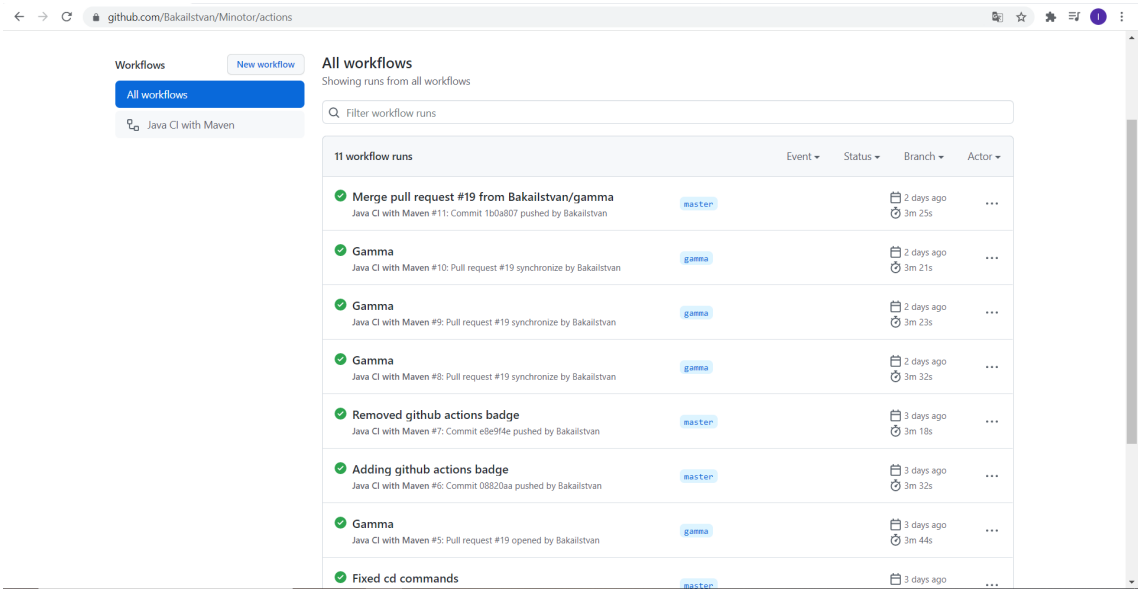
Az időzített automata és monitor forráskód generátorok automatikus tesztelése a *Github Actions* segítségével történik. A *CI* minden feltöltött új *commit* esetén lefut. A *CI* különböző fázisai a következők:

- I. Teljes *Xtext* projekt fordítása (*Maven*)
- II. Tesztek futtatása

A generátorokhoz tartozó *Xtext* projekten lefut egy *Maven build*, amely az új feltöltött verziót tartalmazza. Ez a *build* állítja elő a *DSL* nyelvhez tartozó *Maven plugin*-t is. Ezt követően hajtódnak végre a tesztek, amelyek a frissen fordított *Maven plugin*-t használják. Ha az összes fázis sikeresen lefutott, akkor az adott változtatás nem rontott el semmilyen korábbi funkciót. A *CI*-hoz tartozó *script*-et a 6.2 kódrészlet tartalmazza. A *script* először az *Xtext* nyelv és a hozzá tartozó monitor forráskód generátor új verzióját fordítja le és készíti el belőle a *Maven plugin*-t. Ezután végig megy az összes teszt projekten és egyesével fordítja azokat és végrehajtja a hozzájuk tartozó teszteket. Ha egyik tesztszcenárió se tért vissza hibával, akkor az új *commit* a tesztek szempontjából beilleszthető a *repository*-ba. A *script*-et úgy állítottam be, hogy az ellenőrzés a *master* ágon minden *commit*-ra, illetve a *master* ágra irányuló összes *pull request*-re fusson le. A 6.3 és 6.4 ábrákon látható, hogy minden új *commit*-ra, amely a fő ágra kerül lefut a *CI*.



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
```

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

6.4. Tesztesetek

Ebben a fejezetben az elkészült teszteseteket és hozzájuk tartozó tesztszcenáriókat mutatom be. Ezeknek a teszteseteknek az a céljuk, hogy a fejezet elején ismertetett tesztelési célokat teljesítsék. Az alfejezetek az elkészült tesztelési *Maven* projekteknek felelnek meg, amelyekben bemutatom a hozzájuk tartozó tesztszcenáriókat és a szcenárióhoz tartozó teszteseteket. Az egyszerűbb tesztszcenáriókat mutatom be először és a végén térek ki a komplexebb tesztekre.

6.4.1. Egyszerű időzítési megkötéseket tartalmazó tesztszcenárió

A tesztesetekhez tartozó szcenárió követelmény megtalálható az 6.3 kódrészleten. A szcenárióhoz tartozó diagram a 6.5 ábrán látható

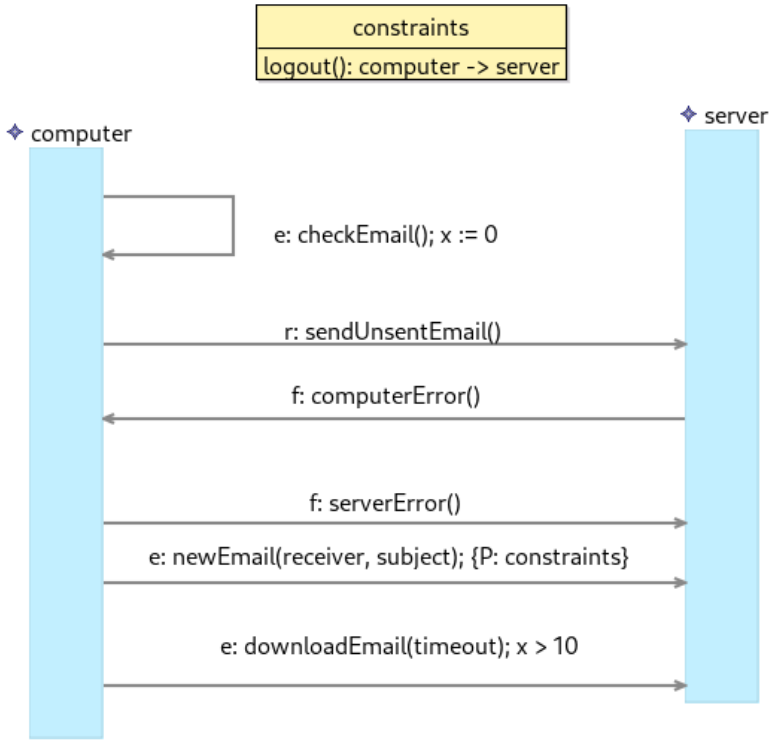
```
specification Email{

    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;
        fail message computerError() server -> computer;
        fail message serverError() computer -> server;
        pastConstraint {constraints} message newEmail(receiver, subject) computer -> server;
        message downloadEmail(timeout) computer -> server clockConstraint {>(x,10)};
    }
}
```

6.3. kódrészlet. Tesztesethez tartozó szcenárió szöveges leírása.

A rendszer egy szervergépből és egy felhasználói számítógépből áll. A szerver egy *e-mail* szervert szimulál, aminek a számítógép különböző kéréseket küldhet. Például lekérdezheti tőle a kapott *e-mail*-t vagy új *e-mail*-t 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*-t az továbbítja a szervernek. Ezt az *elvárt sendUnsentEmail* üzenet jelzi. Ha ezt az üzenetet követően egy *computerError()* üzenet jön az a hibát jelent. *serverError()* üzenet esetén is hibát fog jelezni a monitor. Ezt követően meg kell jelenjen a rendszer működésében a *newEmail* üzenet. Ha 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 szcenárióhoz tartozó tesztesetek a következők:

- *testNetworkRequirementSatisfied*, a rendszer helyes működését szimuláljuk és azt ellenőrizzük, hogy a generált monitor képes ezt érzékelni és jelzi.
- *testNetworkNoErrors*, azt vizsgálja, hogy a monitor képes-e érzékelni, hogy a rendszer nem felelt meg a követelménynek. Itt úgy manipuláljuk a teszt rendszert, hogy lehagyjuk a letöltés részt a működésből. Ilyenkor a rendszer nem felel meg a követelménynek, viszont még jó állapotban marad, mert nem történt hiba.



6.5. ábra. Szcenárió diagram vizualizációja.

- *testNetworkWithErrors*, a *sendUnsentEmail* üzenet után egy *logout* üzenetet küldünk a monitornak és azt vizsgáljuk képes-e detektálni ezt a hibát.
- *testNetworkWithNoDelay*, túl gyorsan küldjük a működés végén a *downloadEmail* üzenetet és azt ellenőrizzük képes e a monitor ezt a hibát érzékelni.
- *testNetworkFirstFail*, itt a *sendUnsentEmail()* után küldünk egy *computerError()* üzenetet és ellenőrizzük, hogy a monitor ezt hibának jelzi-e.
- *testNetworkSecondFail*, itt ugyanúgy mint az előzőnél, azt ellenőrizzük, hogy a monitor a *serverError()* üzenet megjelenését hibának érzékeli-e.

A 6.4 kódrészlet *testNetworkRequirementSatisfied* teszteset implementációját tartalmazza. Először létrehozzuk a monitort és átadjuk neki a szöveges leírás alapján generált időzített automatát. Ezután végig megyünk az esemény szekvencián, amit a monitornak az *update()* függvény segítségével juttatunk el. A teszteset végén pedig összevetjük a monitor kimenetét az elvárt kimenettel. Például ennél a tesztesetnél a monitornak azt kell jeleznie, hogy a rendszer jó állapotban van, megfelelt a követelménynek és nem történt hiba.

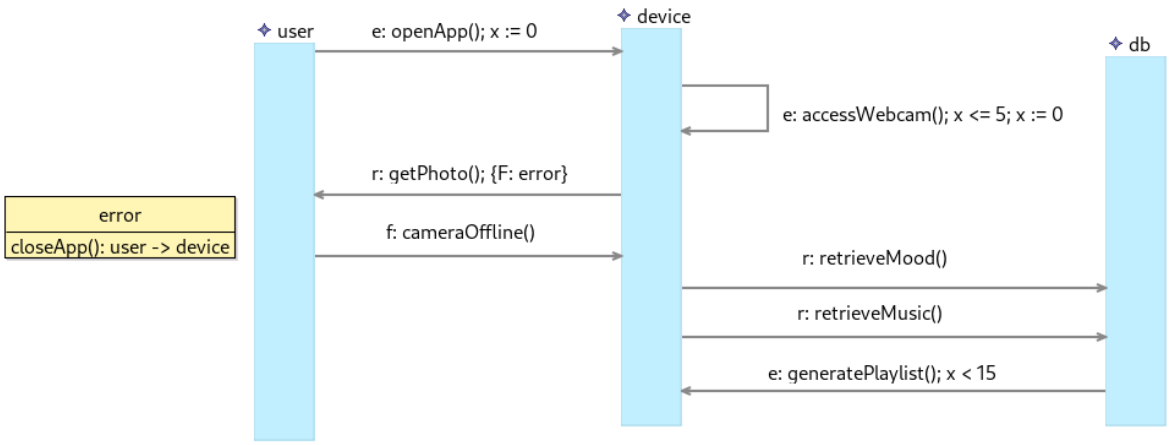
```
@Test
public void testNetworkRequirementSatisfied() {
    resetValues();
    Specification specification = new Specification();
    specification.listAutomatas();
    IClock clock = new Clock();
    IMonitor monitor = new Monitor(specification.getAutomata().get(0), clock, this);

    monitor.update("computer", "computer", "checkEmail", new HashMap<String, Object>());
    monitor.update("computer", "server", "sendUnsentEmail", new HashMap<String, Object>());
    monitor.update("computer", "server", "updateEmail", new HashMap<String, Object>());
    monitor.update("server", "computer", "updateAccount", new HashMap<String, Object>());
    LinkedHashMap<String, Object> hm = new LinkedHashMap<String, Object>();

    hm.put("receiver", "John");
    hm.put("subject", "Next meeting");
    monitor.update("computer", "server", "newEmail", hm);
    try {
        TimeUnit.SECONDS.sleep(11);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    monitor.update("computer", "server", "downloadEmail", Map.of("timeout", 10));

    Assertions.assertTrue(monitor.goodStateReached());
    Assertions.assertTrue(monitor.requirementSatisfied());
    Assertions.assertTrue(requirementSatisfied);
    Assertions.assertFalse(errorDetected);
}
```

6.4. kódrészlet. *testNetworkRequirementSatisfied* teszteset.

6.4.2. Többféle üzenetet és megkötést tartalmazó egyszerű tesztszcenárió

```
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. Második tesztesethez tartozó szcenárió.



6.6. ábra. Második szcenárióhoz tartozó diagram vizualizációja.

A tesztrendszerünk egy zene lista generáló alkalmazás, ami egy felhasználót, mobileszközt és adatbázist tartalmaz. A felhasználó "*kedve*" alapján generálja a listát, amit az arckifejezése alapján határoz meg. A követelményben a rendszer alap működése van leírva egészen az elejétől, amikor a felhasználó megnyitja az alkalmazást. A követelmény leírás megtekinthető az 6.5 kódrészleten és a hozzá tartozó vizualizáció a 6.6 ábrán.

A szcenárióhoz tartozó tesztesetek a következők:

- testMobileRequirementSatisfied, a követelmény teljesülését ellenőrizzük.
- testMobileFutureConstraint, azt vizsgáljuk hogy a monitor hibát jelez-e, ha megkötésbeli üzenet érkezik.
- testMobileFutureConstraintEarly, azt vizsgáljuk hogy a monitor hibát jelez-e, ha a megkötésbeli üzenet az üzenet szekvencia elején érkezik.

- testMobileWithError, a monitor hibát jelez-e, amikor a *cameraOffline()* üzenet érkezik.
- testMobileWithDelay, késleltetjük az *accessWebcam()* üzenet megérkezését úgy, hogy az időzítési feltétel teljesüljön és vizsgáljuk, hogy a monitor ezt helyesen kezeli-e.
- testMobileWithTooMuchDelay, késleltetjük az *accessWebcam()* üzenetet az időzítési feltételt elrontva és azt várjuk el, hogy a monitor ezt hibának jelezze.
- testMobileMissingRequiredMessage, vizsgáljuk hogy a monitor hibát jelez-e, ha nem kapja meg az *retrieveMood()* elvárt üzenetet.
- testMobileRequiredEventually, a monitor helyes működést kell jelezzen, ha az elvárt üzenetet megkapja, még ha más üzenetek is előjönnek ez alatt.
- testMobileRequiredNotReceived, a monitor nem érzékeli a *retrieveMusic()* üzenetet, ezért hibát kell jelezzen.

A tesztesetek a monitor hiba detektáló képeségét tesztelik. A *testMobileWithDelay* és *testMobileWithTooMuchDelay* tesztesetek az $x <= 5$ időzítési feltétel beteljesülését ellenőrzik. A *testMobileWithDelay* tesztesetben 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 tesztszcenárió

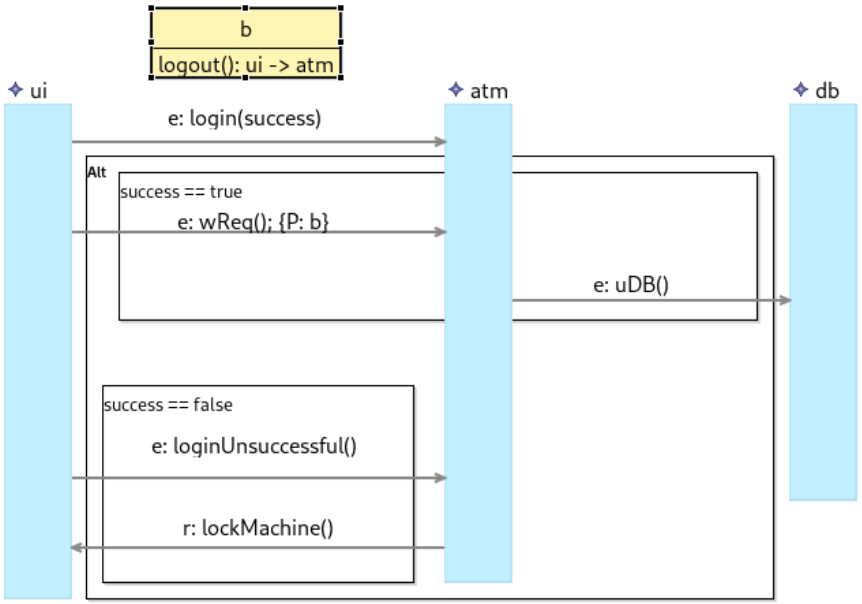
A tesztszcenárióhoz tartozó rendszerünk egy banki rendszer. A szcenárió megtalálható az 6.6 kódrészleten és a hozzá tartozó vizualizáició a 6.7 ábrán látható.

```
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. Alt operátort tartalmazó tesztszcenárió.

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 szcenárióhoz tartozó tesztesetek a következők:

- testBankMonitorPassing, az operátorban lévő első ágnak megfelelően szimuláljuk a rendszer működését és elvárjuk, hogy a monitor helyes működést jelezzen.



6.7. ábra. Alt operátort tartalmazó szcenárió diagram

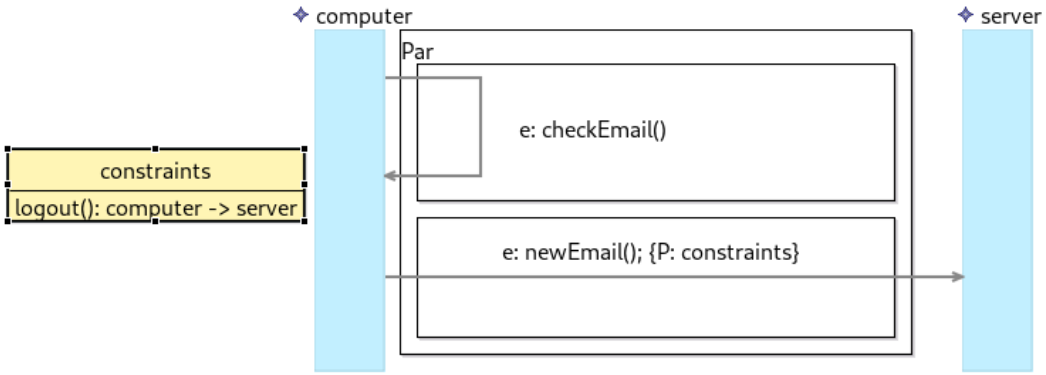
- testBankMonitorFailing, a monitornak hibát kell jeleznie, ha az operátor első ágától eltér a működés.
- testBankMonitorFalseCasePassing, a monitornak helyes működést kell jeleznie, ha a rendszert úgy szimuláljuk ahogy az operátor második ágában le van írva.
- testBankMonitorFalseCaseFailing, a monitor hibát jelez, ha nem az operátor második ága szerint működik a rendszer.

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. Par operátort tartalmazó tesztszcenárió.



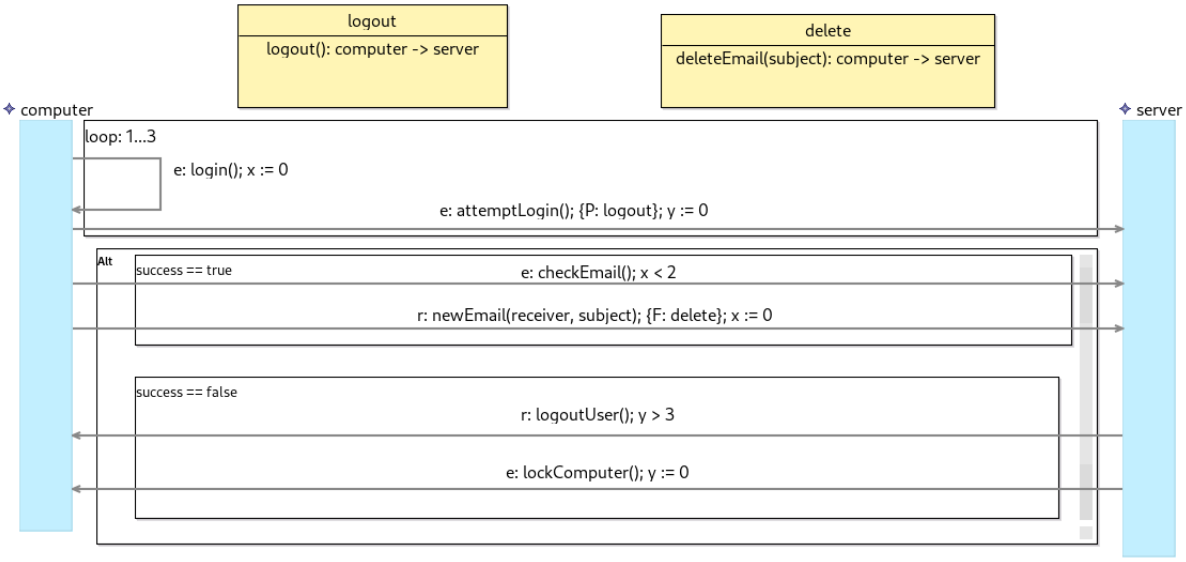
6.8. ábra. Par operátort tartalmazó szcenárió diagram.

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

- testNetworkRequirementSatisfied, az első permutáció szerint működtetjük a szimulált rendszert, a monitornak helyes működést kell jeleznie.
- testNetworkOtherRequirementSatisfied, a második permutáció szerint működtetjük a rendszert, a monitornak helyes működést kell jeleznie.

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

6.4.5. Komplex tesztszenárió loop és alt operátorokkal



6.9. ábra. Komplex szcenárió diagram vizualizációja.

Ebben a tesztszcenárióban olyan szcenáriók támogatását teszteljük, amelyekben több operátor is megjelenik. A tesztesethez tartozó szcenárió a 6.8 kódrészletben található meg

```
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(success) 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. Komplex teszteset szcenáriója.

és a diagram vizualizációja a 6.9 ábrán látható. Ezen kívül azt is vizsgáljuk, hogy a generált monitor képes-e több óraváltozó kezelésére is.

A szcenárióban a *loop* operátort használva egy felhasználó többszörös bejelentkezési próbálkozását írjuk le. Utána egy *alt* operátorral mondjuk meg, hogy minek kell történie sikeres vagy sikertelen bejelentkezés esetén.

A tesztszcenárióhoz tartozó tesztesetek:

- testNetworkRequirementSatisfied, a követelmény teljesülését ellenőrizzük.
- testNetworkRequirementSatisfiedTwice, a követelménynek akkor is teljesülnie kell, ha a *loop* operátorban lévő üzenetek kétszer egymás után jelennek meg.
- testNetworkRequirementSatisfiedThreeTimes, az üzenetek háromszor egymás után jelennek meg.
- testNetworkRequirementSatisfiedFourTimes, a monitornak hibát kell jeleznie, ha négyszer érzékeli az üzeneteket, melyek a *loop* operátorban vannak, hiszen az meghaladja a maximumot.
- testNetworkAltTrueCase, a monitor hibát jelez, ha úgy működtetjük a szimulált rendszer, hogy az *alt* operátor igaz ágában lévő üzenetek jelenjenek meg.
- testNetworkAltTrueCaseSatisfied, a monitor helyes működést kell jelezzen, ha ugyanúgy szimuláljuk a rendszer működését, mint az előző tesztesetben.

- testNetworkLogoutTooFast, ha túl gyorsan érkezik meg a *logoutUser()* üzenet, akkor a monitornak hibát kell jeleznie.
- testNetworkLogoutConstrait, ha olyan üzenet érkezik, amely a megkötésben szerepel, akkor a monitornak hibát kell jeleznie.

A teszteseteket úgy állítottam össze, hogy ellenőrizem mind a monitor hiba detektáló képességét, mind a helyes működés detektálását.

6.5. Tesztelés összefoglaló

A 6.1 táblázat mutatja be a tesztelés eredményét. Sikerült az összes tesztelési célnak megfelelni, emelett biztosítáni a szisztematikus ellenőrzést. Minden teszt sikeresen lefutott.

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 kivánt (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	-	-	-	-
Egyszerű 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 modellalapú fejlesztését. A létrehozott rendszert a keretrendszerrel lehet tesztelni, ellenőrizni és szimulálni is. A modellhez generálhatók tesztesetek, vagy akár verifikálható az *UPPAAL* modell ellenőrző eszköz használatával. A keretrendszer a *Yakindu* modellező eszközt használja, amit kiegészít egy modellező réteggel ahol leírhatók a komponensek közötti interakciók. A tervezett rendszerhez a keretrendszer képes *Java* forráskódot is generálni.

A monitor illesztéséhez a *Gamma tutorial* csomagjában lévő példarendszert használtam. Ez egy irányitórendszer modellje, 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 forráskódját. A példarendszer monitorozott kereszteződésének az implementációja a *gamma.monitoredcrossroad* csomagban található. Ez a csomag a *ReflectiveMonitoredCrossroad* és a *MonitoredCrossroadInterface* interfészt megvalósító *MonitoredCrossroad* osztályokat tartalmazza. A *MonitoredCrossroad* osztály attribútumai között megtalálható a *MonitorInterface* interfészt megvalósító eredeti *Monitor* objektum. Ezt az eredeti monitort cseréltem le a saját a monitor komponensemhez tartozó forráskóddal megvalósítva a szükséges interfészeket.

A *Gamma* rendszer és a monitor komponens közti kommunikáció megvalósítása a 7.1-es kódrészletben látható. Ez a kódrészlet az általam készített *GammaMonitor* osztálynak a része, ami leszármazik a generált monitor *Java* osztályából, ami az *util* csomagban található. A rendszer eseményeket detektálja és olyan alakra alakítja az *update()* függvény segítségével, amellyel a monitor képes értelmezni azokat.

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

```
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.

```
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.

```
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.

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

```
specification Light{

    object TrafficLight light;
    object Controller controller;
    object Police police;

    bool success = false;

    clock x;

    scenario trafficLight {
        message policeInterruptRaised(success) police -> controller reset x;

        alt(equals(success, false)) {
            message displayRed() controller -> light;
            message displayGreen() controller -> light;
            message displayYellow() controller -> light;
        } (equals(success, true)) {
            message displayYellow() controller -> light clockConstraint {>=(x,1)} reset x;
            message displayNone() controller -> light clockConstraint {>=(x,1)} reset x;
            message displayYellow() controller -> light clockConstraint {>=(x,1)} reset x;
            message displayNone() controller -> light clockConstraint {>=(x,1)};
        }

    }
}
```

7.4. kódrészlet. Szenárió szöveges leírása.

A 7.4-es kódrészlet egy másik követelményt tartalmaz, amit a rendszeren szeretnénk vizsgálni. Azt várjuk, hogy ha történt rendőrség általi *interrupt* kérés, akkor a lámpa sárgán villogjon. Ha pedig nem volt ilyen *interrupt*, akkor azt várjuk el, hogy a lámpa helyesen viselkedjen. Ennek a követelménynek az ellenőrzéséhez már nem volt elég csupán az eredeti *Monitor* osztályt lecserélni, hanem a *MonitoredCrossroad* osztályba is segédfüggvényeket kellett készíteni az illesztés érdekében. A 7.5 kódrészlet tartalmazza az illesztés kiegészítését. Itt a rendőrségi *interrupt* kérést továbbítjuk a monitornak. A 7.6 kódrészlet tartalmazza a *MonitoredCrossroad* osztály *PriorityOutput*-jának kiegészítését. Így továbbítja a kereszteződés a lámpájának állásait a monitor felé. A rendszer mindkét esetben a követelménynek megfelelően működött. A szcenárióhoz tartozó monitor kimenetek a *Függelékben* találhatók meg (F.3.1, F.3.2).

```
@Override
public void raisePolice() {
    monitor.update("police", "controller", "policeInterruptRaised", Map.of("success", true));
    crossroad.getPolice().raisePolice();
}
```

7.5. kódrészlet. *Gamma* illesztéshez tartozó kódrészlet.

```
// Class for the setting of the boolean fields (events)
private class PriorityOutputUtil implements LightCommandsInterface.Listener.Provided {
    @Override
    public void raiseDisplayNone() {
        isRaisedDisplayNone = true;
        monitor.update("controller", "light", "displayNone", new HashMap<String, Object>());
    }

    @Override
    public void raiseDisplayYellow() {
        isRaisedDisplayYellow = true;
        monitor.update("controller", "light", "displayYellow", new HashMap<String, Object>());
    }

    @Override
    public void raiseDisplayRed() {
        isRaisedDisplayRed = true;
        monitor.update("controller", "light", "displayRed", new HashMap<String, Object>());
    }

    @Override
    public void raiseDisplayGreen() {
        isRaisedDisplayGreen = true;
        monitor.update("controller", "light", "displayGreen", new HashMap<String, Object>());
    }
}
```

7.6. kódrészlet. *MonitoredCrossroad* osztály kiegészítése a monitor illesztésével.

8. fejezet

Összefoglalás

A célként kitűzött szcenárió alapú monitor generátor kibővítése sikerült.

A szöveges szcenárió 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 szcenáriókat a szöveges leírásuk alapján diagramok formájában vizualizálom. Egy *XML* generátor teszi ezt lehetővé, amely a szöveges leírás alapján elkészíti a hozzá tartozó diagram *XML* leírását.

A monitor forráskód generátor a szcenárióhoz tartozó automata alapján képes egy monitor forráskódjának generálására. Legenerálja a megfelelő interfészeket, amelyek 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 generált monitor forráskód helyességét alapos tesztelés segítségével ellenőriztem. A monitor forráskód szisztematikus helyességét egy *CI* rendszerrel ellenőrzöm, amely minden változtatás esetén ellenőrzi, hogy a projekt tesztjei helyesek-e.

Végezetül a monitor komponenst illesztettem a *Gamma* keretrendszerhez, így demonstrálva a monitorozás működését egy elosztott komponensű rendszer viselkedésének ellenőrzésével.

A monitor forráskód generátor tovább bővíthető úgy hogy, láncolt megkötéseket is támogasson. A szcenárió követelményünkben hasznos lehet olyan megkötéseket definiálni, amelyek nem egy üzenet halmazra vonatkoznak, hanem egy kisebb üzenet szekvenciára. Elképzelhető olyan eset, ahol adott üzenetek külön-külön nem jelentenek hibát a rendszer működésére nézve, viszont ha adott sorrendben érkeznek, az már hibás lehet. Ehhez bevezethető egy új nyelvi elem a meglévő *TPSC* leíró nyelvbe, amely ilyen láncolt megkötések megadására szolgál. Az automata generátorhoz is hozzáadhatók az új nyelvi elemekhez tartozó automata minták.

Irodalomjegyzék

[1] Bakai István Bálint: Monitor komponensek generálása kontextusfüggő viselkedés ellenőrzésére. Szakdolgozat (Budapesti Műszaki és Gazdaságtudományi Egyetem). 2019. 12.

[2] Eclipse Foundation: Eclipse. https://www.eclipse.org/.

[3] The Apache Software Foundation: Maven. https://maven.apache.org/.

[4] Entity Modeling Framework: Entity modeling framework. https://www.eclipse.org/modeling/emf/.

[5] FTSRG Hibatűrő Rendszerek Kutatócsoport: Gamma statechart composition framework. https://inf.mit.bme.hu/node/6028.

[6] M. Leucker: Teaching runtime verificatio. *Springer*, 2011.

[7] 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. *IET Software*, 7. évf. (2013. 08) 4. sz., 222–248. p.

[8] Object Management Group (OMG): *UML: superstructure version 2.0*. Object Management Group (OMG), 2004.

[9] J. Ouaknine – J. Worrell: On metric temporal logic and faulty turing machines. *Springer-Verlag*, LNCS 3921. évf. (2006. 08), 217–230. p.

[10] M.Autili – P. Inverardi – P.Pelliccione: Graphical scenarios for specifying temporal properties: an automated approach in automated software engineering. *Springer LNCS*, 14. évf. (2007. 09) 3. sz., 293–340. p. https://link.springer.com/article/10.1007%2Fs10515-007-0012-6.

[11] Promela: Never claim. https://spinroot.com/spin/Man/never.html.

[12] T. Feder R. Alur – T. A. Henzinger: The benefits of relaxing punctuality. *Journal of the ACM*, 43. évf. (1996) 1. sz., 116–146. p.

[13] Sirius: Sirius. https://www.eclipse.org/sirius/.

[14] Xtend: Xtend. https://www.eclipse.org/xtend/.

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

[16] ITU-T Recommendation Z.: *Message sequence charts*. ITU Telecom. Standardisation Sector, 1999.

Függelék

F.1. A 5.2.3. fejezet minta példájához tartozó Specification osztály

```
package generated;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.Map.Entry;
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;
import java.util.AbstractMap;

import util.Automaton;
import util.BasicTransition;
import util.ClockConstraint;
import util.UnwantedConstraint;
import util.WantedConstraint;
import util.State;
import util.StateType;
import util.Transition;
import util.NeverClaimWriter;
import util.OperatorFunctions;
import util.AltExpressionInterface;

public class Specification{
  private String id = "Photo";
  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;
    OperatorFunctions opFunctions = new OperatorFunctions();
    Automaton a = new Automaton("playlist_generation");
    Automaton b;
    Map<String, Entry<AltExpressionInterface, Automaton>> altauto;
    ArrayList<Automaton> parauto;
    Automaton loopauto;
    Automaton expression;
    int counter = 0;
```

```
    b = new Automaton("auto30");
    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 BasicTransition(actualState
                      , newState
                      , null
                      , "user" + "." +
                        "openApp" + "("
                        + ")"

                        + "." + "device"
                      , null
                      , null));

    b.addTransition(new BasicTransition(actualState
                      , actualState
                      , null
                      , "!(" + "user" + "." +
                        "openApp" + "("
                        + ")"

                        + "." + "device" + ")"
                      , null
                      , null));
    b.addState(newState);
    b.setFinale(newState);
    a.collapse(b);
    b = new Automaton("auto30");
    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 BasicTransition(actualState
                      , newState
                      , null
                      , "device" + "." +
                        "accessWebcam" + "("
                        + ")"

                        + "." + "device"
                      , null
                      , null));

    b.addTransition(new BasicTransition(actualState
                      , actualState
                      , null
                      , "!(" + "device" + "." +
                        "accessWebcam" + "("
                        + ")"

                        + "." + "device" + ")"
                      , null
                      , null));
    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);

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

```
    b.addState(acceptState);

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new BasicTransition(actualState
                      , newState
                      , null
                      , "device" + "." +
                        "getPhoto" + "("
                        + ")"

                        + "." + "user"
                      , null
                      , null));

    b.addTransition(new BasicTransition(actualState
                      , actualState
                      , null
                      , "!(" + "device" + "." +
                        "getPhoto" + "("
                        + ")"

                        + "." + "user" + ")"
                      , null
                      , null));

    b.addTransition(new BasicTransition(actualState
                      , acceptState
                      , null
                      , "!(" + "device" + "." +
                        "getPhoto" + "("
                        + ")"

                        + "." + "user" + ")"
                      , null
                      , null));

    b.addState(newState);
    b.setFinale(newState);
    a.collapse(b);
    b = new Automaton("auto4");
    actualState = new State("q" + counter, StateType.NORMAL);
    counter++;
    b.addState(actualState);
    b.setInitial(actualState);

    b.addTransition(new BasicTransition(actualState
                      , actualState
                      , null
                      , "!(" + "user" + "." +
                        "cameraOffline" + "("
                        + ")"

                        + "." + "device" + ")"
                      , null
                      , null));

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

    b.addTransition(new BasicTransition(actualState
                      , finalState
                      , null
                      , "!(" + "user" + "." +
                        "cameraOffline" + "("
                        + ")"

                        + "." + "device" + ")"
                      , null
                      , null));

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

```
    counter++;

    b.addTransition(new BasicTransition(actualState
                      , acceptState
                      , null
                      , "user" + "." +
                        "cameraOffline" + "("
                        + ")"

                        + "." + "device"
                      , null
                      , null));

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

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

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new BasicTransition(actualState
                      , newState
                      , null
                      , "device" + "." +
                        "retrieveMood" + "("
                        + ")"

                        + "." + "db"
                      , null
                      , null));

    b.addTransition(new BasicTransition(actualState
                      , acceptState
                      , null
                      , "!(" + "device" + "." +
                        "retrieveMood" + "("
                        + ")"

                        + "." + "db" + ")"
                      , null
                      , null));

    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);

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

    newState = new State("q" + counter, StateType.FINAL);
    counter++;
    b.addTransition(new BasicTransition(actualState
                      , newState
                      , null
                      , "device" + "." +
                        "retrieveMusic" + "("
                        + ")"
```

```
                      + "." + "db"
                    , null
                    , null));

  b.addTransition(new BasicTransition(actualState
                    , actualState
                    , null
                    , "!(" + "device" + "." +
                      "retrieveMusic" + "("
                      + ")"

                      + "." + "db" + ")"
                    , null
                    , null));

  b.addTransition(new BasicTransition(actualState
                    , acceptState
                    , null
                    , "!(" + "device" + "." +
                      "retrieveMusic" + "("
                      + ")"

                      + "." + "db" + ")"
                    , null
                    , null));

  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 BasicTransition(actualState
                    , newState
                    , null
                    , "db" + "." +
                      "generatePlaylist" + "("
                      + ")"

                      + "." + "device"
                    , null
                    , null));
  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()){
      System.out.println(t.toString());
    }
  }
}

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

public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException {
```

```
    Specification specification = new Specification();
    specification.listAutomatas();

    NeverClaimWriter ncWriter = new NeverClaimWriter();
    ncWriter.writeNeverClaim("Photo", specification.automatas);
  }
}
```

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. szcenárióhoz 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. szcenárióhoz tartozó rendszer *ATM Java* osztálya.

F.3. Gamma illesztéshez tartozó monitor kimenetek

```
[Automaton] Setting final transition
[Automaton] Setting final transition
[GammaMonitorTest] Resetting values
Received Message: controller.displayRed().light
[Monitor] available transition are:
-----------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
```

```
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. szcenárió 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.

```
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=----------------------------------=
Transition: police.policeInterruptRaised(success).controller, q0->q1, ,
[Monitor] available transition are:
-----------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=----------------------------------=
Transition: !(police.policeInterruptRaised(success).controller), q0->q0, ,
transition triggered: !(police.policeInterruptRaised(success).controller), q0->q0, ,
q0
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayRed().light
```

```
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.

```
[Monitor] available transition are:
-----------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=---------------------------------=
Transition: police.policeInterruptRaised(success).controller, q0->q1, ,
[Monitor] available transition are:
-----------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=---------------------------------=
Transition: !(police.policeInterruptRaised(success).controller), q0->q0, ,
transition triggered: !(police.policeInterruptRaised(success).controller), q0->q0, ,
q0
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayGreen().light
[Monitor] available transition are:
-----------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
```

```
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=----------------------------------=
Transition: police.policeInterruptRaised(success).controller, q0->q1, ,
[Monitor] available transition are:
------------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=----------------------------------=
Transition: !(police.policeInterruptRaised(success).controller), q0->q0, ,
transition triggered: !(police.policeInterruptRaised(success).controller), q0->q0, ,
q0
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: police.policeInterruptRaised(success).controller
[Monitor] available transition are:
------------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=----------------------------------=
Transition: police.policeInterruptRaised(success).controller, q0->q1, ,
transition triggered: police.policeInterruptRaised(success).controller, q0->q1, ,
q1
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayYellow().light
[Monitor] available transition are:
------------------------------------
epsilon, q1->qinit0
=----------------------------------=
Transition: epsilon, q1->qinit0
[EpsilonTransition]epsilon, q1->qinit0 canTrigger is true
PrevTransition: epsilon, q1->qinit0
[Monitor] available transition are:
------------------------------------
epsilon, qinit0->q2
epsilon, qinit0->q4
=----------------------------------=
Transition: epsilon, qinit0->q2
[EpsilonTransition]epsilon, qinit0->q2 canTrigger is true
PrevTransition: epsilon, qinit0->q2
[Monitor] available transition are:
------------------------------------
!(controller.displayYellow().light), q2->q2, ,
controller.displayYellow().light, q2->q3, ,
epsilon, qinit0->q4
=----------------------------------=
Transition: !(controller.displayYellow().light), q2->q2, ,
[Monitor] available transition are:
------------------------------------
!(controller.displayYellow().light), q2->q2, ,
controller.displayYellow().light, q2->q3, ,
epsilon, qinit0->q4
=----------------------------------=
Transition: controller.displayYellow().light, q2->q3, ,
[Monitor] available transition are:
------------------------------------
!(controller.displayYellow().light), q2->q2, ,
controller.displayYellow().light, q2->q3, ,
epsilon, qinit0->q4
=----------------------------------=
Transition: epsilon, qinit0->q4
[EpsilonTransition]epsilon, qinit0->q4 canTrigger is false
[EpsilonTransition]epsilon, qinit0->q4 canTrigger is false
[Monitor] available transition are:
------------------------------------
!(controller.displayYellow().light), q2->q2, ,
controller.displayYellow().light, q2->q3, ,
=----------------------------------=
Transition: !(controller.displayYellow().light), q2->q2, ,
[Monitor] available transition are:
------------------------------------
!(controller.displayYellow().light), q2->q2, ,
controller.displayYellow().light, q2->q3, ,
=----------------------------------=
```

```
Transition: controller.displayYellow().light, q2->q3, ,
transition triggered: controller.displayYellow().light, q2->q3, ,
q3
[GammaMonitorTest] Received status from monitor: System is in good state.
transition triggered: epsilon, q3->qfinal1
qfinal1
[GammaMonitorTest] Received status from monitor: Requirement satisfied
[GammaMonitorTest] Monitor reported that the requirement was satisfied
```

F.3.1. kódrészlet. *Gamma* monitor kimenet.

```
[Automaton] Setting final transition
[Automaton] Setting final transition
[GammaMonitorTest] Resetting values
Received Message: police.policeInterruptRaised(success).controller
[Monitor] available transition are:
-----------------------------------
police.policeInterruptRaised(success).controller, q0->q1, ,
!(police.policeInterruptRaised(success).controller), q0->q0, ,
=-----------------------------------=
Transition: police.policeInterruptRaised(success).controller, q0->q1, ,
transition triggered: police.policeInterruptRaised(success).controller, q0->q1, ,
q1
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayRed().light
[Monitor] available transition are:
-----------------------------------
epsilon, q1->qinit0
=-----------------------------------=
Transition: epsilon, q1->qinit0
[EpsilonTransition]epsilon, q1->qinit0 canTrigger is true
PrevTransition: epsilon, q1->qinit0
[Monitor] available transition are:
-----------------------------------
epsilon, qinit0->q2
epsilon, qinit0->q4
=-----------------------------------=
Transition: epsilon, qinit0->q2
[EpsilonTransition]epsilon, qinit0->q2 canTrigger is false
[EpsilonTransition]epsilon, qinit0->q2 canTrigger is false
[Monitor] available transition are:
-----------------------------------
epsilon, qinit0->q4
=-----------------------------------=
Transition: epsilon, qinit0->q4
[EpsilonTransition]epsilon, qinit0->q4 canTrigger is true
PrevTransition: epsilon, qinit0->q4
[Monitor] available transition are:
-----------------------------------
controller.displayRed().light, q4->q5, ,
!(controller.displayRed().light), q4->q4, ,
=-----------------------------------=
Transition: controller.displayRed().light, q4->q5, ,
transition triggered: controller.displayRed().light, q4->q5, ,
q5
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayRed().light
[Monitor] available transition are:
-----------------------------------
controller.displayGreen().light, q5->q6, ,
!(controller.displayGreen().light), q5->q5, ,
=-----------------------------------=
Transition: controller.displayGreen().light, q5->q6, ,
[Monitor] available transition are:
-----------------------------------
controller.displayGreen().light, q5->q6, ,
!(controller.displayGreen().light), q5->q5, ,
=-----------------------------------=
Transition: !(controller.displayGreen().light), q5->q5, ,
transition triggered: !(controller.displayGreen().light), q5->q5, ,
q5
[GammaMonitorTest] Received status from monitor: System is in good state.
```

```
Received Message: controller.displayGreen().light
[Monitor] available transition are:
-----------------------------------
controller.displayGreen().light, q5->q6, ,
!(controller.displayGreen().light), q5->q5, ,
=---------------------------------=
Transition: controller.displayGreen().light, q5->q6, ,
transition triggered: controller.displayGreen().light, q5->q6, ,
q6
[GammaMonitorTest] Received status from monitor: System is in good state.
Received Message: controller.displayYellow().light
[Monitor] available transition are:
-----------------------------------
controller.displayYellow().light, q6->q7, ,
!(controller.displayYellow().light), q6->q6, ,
=---------------------------------=
Transition: controller.displayYellow().light, q6->q7, ,
transition triggered: controller.displayYellow().light, q6->q7, ,
q7
[GammaMonitorTest] Received status from monitor: System is in good state.
transition triggered: epsilon, q7->qfinal1
qfinal1
[GammaMonitorTest] Received status from monitor: Requirement satisfied
[GammaMonitorTest] Monitor reported that the requirement was satisfied
```

F.3.2. kódrészlet. *Gamma* monitor kimenet rendőrségi példa.

F.4. 3. fezetben ismertett TPSC szöveges leíró nyelvhez tartozó Xtext nyelvtan

```
grammar hu.bme.mit.dipterv.text.MinotorDsl with org.eclipse.xtext.common.Terminals

generate minotorDsl "http://www.bme.hu/mit/dipterv/text/MinotorDsl"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Domain:
  (specification='specification')? (name=ID)? ('{')?
  objects+=Object*
  parameters+=Parameter*
  clocks+=Clock*
  constraints+=Constraint*
  scenarios+=Scenario* ('}')?
;

AttributeValue:
  value=STRING | value=REAL | value=NUMBER | value='true' | value='false'
;

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)? ';'
;

StrictMessage:
  'strict'
  message+=LooseMessage
;

PastMessage:
  'pastConstraint'
  '{' c=[Constraint] (',' constraintexp=ClockConstraintExpression)? (',' resetinconstraint=ResetClock
    )? '}'
  message+=LooseMessage
;

FutureMessage:
  'futureConstraint'
  '{' c=[Constraint] (',' constraintexp=ClockConstraintExpression)? (',' resetinconstraint=ResetClock
    )? '}'
   message+=LooseMessage
;

StrictFutureMessage:
  'strict'
  futureMessage+=FutureMessage
;

RequiredLooseMessage:
  'required'
  message+=LooseMessage
;

RequiredStrictMessage:
  'required'
  strictMessage+=StrictMessage
;

RequiredPastMessage:
  'required'
  pastMessage+=PastMessage
;

RequiredFutureMessage:
  'required'
  futureMessage+=FutureMessage
;

RequiredStrictFutureMessage:
  'required'
  strictFutureMessage+=StrictFutureMessage
;

FailMessage:
  'fail'
  message+=LooseMessage
;

FailStrictMessage:
  'fail'
  strictMessage+=StrictMessage
;

FailPastMessage:
  'fail'
  pastMessage+=PastMessage
;

ResetClock:
  'reset' clock=[Clock]
;
```

```
ClockConstraint:
  GreaterClockConstraint | SmallerClockConstraint | GreaterEqualClockConstraint |
     SmallerEqualClockConstraint
;

GreaterClockConstraint:
  '>' '(' clock=[Clock] ',' constant=NUMBER ')'
;

SmallerClockConstraint:
  '<' '(' clock=[Clock] ',' constant=NUMBER ')'
;

GreaterEqualClockConstraint:
  '>=' '(' clock=[Clock] ',' constant=NUMBER ')'
;

SmallerEqualClockConstraint:
  '<=' '(' clock=[Clock] ',' constant=NUMBER ')'
;

ClockConstraintExpression:
    ClockConstraint | NotClockConstraintExpression | AndClockConstraintExpression
;

NotClockConstraintExpression:
  'not' '(' notClockConstraint=ClockConstraint ')'
;

AndClockConstraintExpression:
  lclockconstraint=ClockConstraint 'and' rclockconstraint=ClockConstraint
;

Params:
    (params+=[Parameter])? (',' params+=[Parameter])*
;

ConstantParams:
  (values+=AttributeValue)? (',' values+=AttributeValue)*
;

Parameter:
  type=Type name=ID ('=')? (value=AttributeValue)? ';'
;

ParameterConstraint:
  'assertParameter' '[' param=[Parameter] operator+=Operator value+=AttributeValue ']' 'in' object=[
     Object] ';'
;

Operator:
  greater?='>' | smaller?='<' | greaterequals?='>=' | smallerequals?='<=' | equals?='==' | notequals
     ?='!='
;

Clock:
  'clock' name=ID ';'
;

enum Type:
  integer | float | bool | string
;

terminal NUMBER:
  ('0'..'9')*
;

terminal REAL:
  ('0'..'9')* '.' ('0'..'9')*
;

ObjectType:
```

```
  'object' name=ID
;

Object:
  object+=ObjectType name=ID ';'
;

Constraint:
  'constraint' name=ID '{'
    messages+=Message*
  '}'
;

Alt:
  'alt' expressions+=Expression*
;

Expression:
  '(' altCondition=LogicalExpression ')' '{'
    messages+=Message*
  '}'
;

LogicalExpression:
  UnaryLogicalExpression | BinaryLogicalExpression
;

BinaryLogicalExpression:
  EqualsExpression | EqualsBooleanExpression | GreaterThanExpression | LesserThanExpression |
  AndExpression | OrExpression
;

AndExpression:
  'and' '(' lhs=LogicalExpression ',' rhs=LogicalExpression ')'
;

OrExpression:
  'or' '(' lhs=LogicalExpression ',' rhs=LogicalExpression ')'
;

EqualsExpression:
  'equals' '(' lhs=[Parameter] ',' rhs=NUMBER ')'
;

EqualsBooleanExpression:
  'equals' '(' lhs=[Parameter] ',' rhs='true' ')' | 'equals' '(' lhs=[Parameter] ',' rhs='false' ')'
;

GreaterThanExpression:
  'greater' '(' lhs=[Parameter] ',' rhs=NUMBER ')'
;

LesserThanExpression:
  'lesser' '(' lhs=[Parameter] ',' rhs=NUMBER ')'
;

UnaryLogicalExpression:
  NotLogicalExpression
;

NotLogicalExpression:
  'not' '(' operand=LogicalExpression ')'
;

Par:
  'par' '{' parexpression+=ParExpression* '}'
;

ParExpression:
  'case' name=ID '{' messages+=Message* '}'
;
```

```
Loop:
  'loop' '(' min=NUMBER ',' max=NUMBER ')' '{'
    messages+=Message*
  '}'
;
```

F.4.1. kódrészlet. *Xtext* nyelvtan specifikációja.