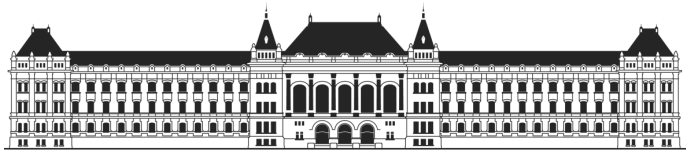


MŰEGYETEM 1782
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Nyelvek és keretrendszerek tesztek specifikálására

PhD tézisfüzet

Micskei Zoltán

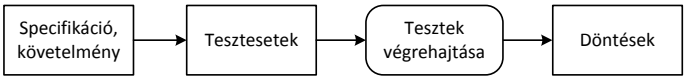
Konzulens:
Majzik István, PhD (BME)
Hélène Waeselynck, PhD (LAAS-CNRS)

Budapest, 2013.

1. Előzmények és célkitűzések

A tesztelés a szoftverfejlesztés meghatározó, de rendkívül erőforrás-igényes részfeladata. Az IEEE definíciója szerint a tesztelés egy olyan „tevékenység, amelyben egy rendszert vagy egy komponenst meghatározott feltételek mellett végrehajtunk, az eredményeket megfigyeljük vagy rögzítjük, és ezek alapján a rendszer vagy a komponens bizonyos jellemzőjét kiértékeljük" [IEE10].

Az 1. ábra áttekinti a tesztelés leegyszerűsített folyamatát: a rendszer specifikációja (specification) és követelményei (requirement) alapján teszteseteket (test case) származtatunk, ezeket végrehajtjuk, majd döntést (verdict) hozzunk az eredmények alapján, pl. hogy sikeres vagy sikertelen volt a lefutás. Természetesen ehhez még rengeteg részletet ki kell előbb dolgoznunk. A rendszer milyen részét szükséges tesztelni? Hogyan fogjuk kiértékelni a kapott eredményeket? Ki dönti el, hogy sikeres vagy sikertelen-e egy lefutás? Ezeknek a kérdéseknek a megválaszolásához még további úgynevezett tesztelési termékre (test artifact) van szükség.



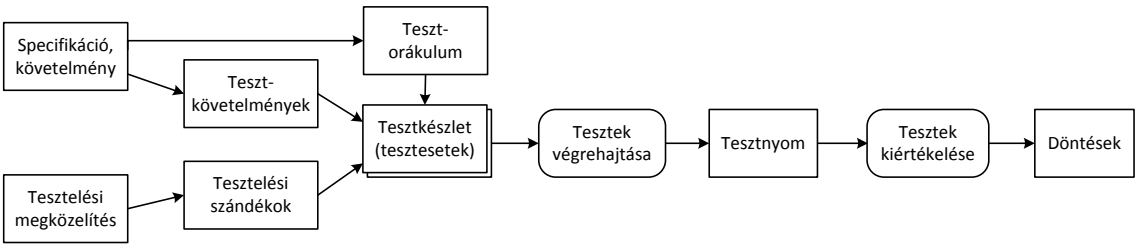
1. ábra. A tesztelési folyamat áttekintése

A 2. ábra bemutatja a további, tipikus tesztelési termékeket[1]. A magas szintű felhasználói követelményekből és a specifikációkból *tesztkövetelmény*eket származtatunk, amik meghatározzák, hogy a rendszernek adott helyzetekben hogyan kell vagy lehet működnie. A *tesztelési megközelítés* (test approach) összegyűjti, hogy mikor és hogyan kell tesztelnünk, például, hogy milyen folyamatokat, technikákat, tesztelési szinteket és eszközöket kell alkalmazni. A *tesztelési szándék*ok (test purpose) meghatározzák, hogy a rendszer funkcionalitásának mely részét kell a tesztelésnek lefednie. Ezek alapján *teszteset*eket készítünk, amik leírják a felhasználandó bemeneteket, az elvárt eredményt és a szükséges végrehajtási környezetet. Az elvárt kimenetet a *teszt-orákulum* (test oracle) segítségével határozhatjuk meg. Később a teszteseteket implementáljuk, és *tesztillesztő*k (test adapter) segítségével lefuttatjuk őket egy *végrehajtó környezet*ben (test execution environment), ami a *SUT*-ból (system under test) és úgynevezett *tesztdublőr*ökből (test double) áll, amik szimulálják a környezetet vagy a rendszer többi komponensét (ilyen dublőr lehet például egy csonk vagy meghajtó). A végrehajtás során *tesztnyom*okat (test trace) rögzítünk, amik a SUT válaszait és a tesztkörnyezetben bekövetkezett változásokat tartalmazhatják. Végül a végrehajtás eredményeit kiértékeljük, és döntéseket rendelünk az egyes futtatásokhoz. A lehetséges döntések halmaza tipikusan sikeres (pass), sikertelen (fail), hibás (error, a tesztvégrehajtóban következett be valami hiba) vagy nem meggyőző (inconclusive, se sikeres, se sikertelen döntés nem hozható). Ezeket a tesztelési termékeket és a támogató eszközöket lehet egy *tesztkeretrendszer*be (test framework) szervezni.

A szoftvertesztelésnek kiforrott irodalma van (pár alapvető fontosságú könyv például [Bei90; MS04]), számos módszert és technikát dolgoztak ki különböző típusú rendszerek teszteléséhez. Azonban ezeknek a módszereknek az alkalmazásához szükségesek olyan *tesztelési nyelv*ek, amiknek a segítségével a fenti termékeket precízen lehet definiálni a tervezés során.

Az értekezésben bemutatott kutatás arra irányult, hogy (i) milyen *nyelveket* lehet használni tesztelési termékek leírására olyan alkalmazási területeken, ahol jelenleg még nincs kiforrott megoldás erre, és (ii) a nyelvek felhasználásával hogyan lehet *tesztkeretrendszereket* készíteni, amik az adott terület specialitásait figyelembe véve végzik el a tesztelést.

[1] Mivel a tesztelés nagyon általános terület, ezért ezeket a fogalmakat sokféleképpen definiálták már. Jelen értekezés az IEEE terminológiát [IEE10] használja néhol kiegészítve az ISTQB kifejezéseivel [IST10].



2. ábra. Részletes tesztelési termékek

1.1. Meglévő tesztelési nyelvek és módszerek

A fejezet bemutat elsőként néhány nyelvet, amiket tesztelési termékek leírására használhatunk. Ezután ismerteti, hogy hogyan lehet a Unified Modeling Language (UML) nyelvet tesztelési termékek modellezésére felhasználni. Végül áttekinti azokat a tesztelési megközelítéseket, amiket az értekezés használ később.

1.1.1. Példák tesztelési termékek leírására használható nyelvekre

A tesztelési termék típusától függően sokféle módszert és jelölésrendszert javasoltak már az elmúlt évtizedekben. Tesztelési szándékok leírására alkalmasak például a címkézett átmeneti rendszerek [JJ05] vagy a temporális logikai formulák [Hon+01]. Tesztkövetelmények megfogalmazhatóak UML-modellek segítésével [BL02]. Tesztesetek definiálására használható a TTCN-3 [ITU07] vagy az ATML [IEE11] nyelv. A teszt-orákulum kifejezhető időzített automaták segítségével [Hes+08] vagy egy SDL-leírásban [Koc+98].

Ha a rendszer működésének egy részletét szeretnénk csak megragadni, mint ahogyan ezt egy tesztkövetelmény vagy szándék teszi, akkor egy nagyon gyakori megközelítés a grafikus *forgatókönyv-leíró* nyelvek (scenario language) használata [PJ04; KSH07]. Segítségükkel intuitív módon lehet különböző szereplők közötti kommunikációt megfogalmazni. Az ilyen nyelveknek több változata is létezik, az International Telecommunication Union (ITU) szervezet Message Sequence Chart (MSC) [ITU11] nyelve volt az egyik első közülük. A nyelv 1993-as létrehozása óta többször is frissítették már. A később megjelent Live Sequence Chart (LSC) [DH01] olyan kiegészítéseket javasolt, amikkel könnyen meg lehet különböztetni a kötelező és a lehetséges viselkedési mintákat. Mivel az értekezés szoftveres rendszerekre összpontosított, ezért a lehetséges tesztelési és modellezési jelölések közül az UML nyelv volt a leginkább kapcsolódó.

1.1.2. Az UML 2 felhasználása tesztek leírására

Az Object Management Group (OMG) szervezet által kidolgozott Unified Modeling Language (UML) [OMG11b] az egyik leggyakrabban használt nyelv szoftveres rendszerek modellezésére. Az UML kiterjedett eszköztámogatással rendelkezik, és a szoftverfejlesztés számos részfeladatát tudja támogatni a követelmények megfogalmazásától kezdve a telepítési leírók specifikálásáig. Cook friss cikkében [Coo12] egy jó áttekintést ad az UML történetéről és a nyelv fejlődéséről.

A tesztelési tevékenységek támogatására egy külön UML-profilt is létrehoztak. Az *UML 2 Testing Profile* [OMG05] segítségével egy UML-modellben le lehet írni (i) a tesztarchitektúrát, (ii) a tesztesetek viselkedését és (iii) a tesztadatok tartalmát. A tesztarchitektúrát tipikusan UML sztereotípiákkal ellátott komponensekkel modellezik. A tesztesetek és eljárások viselkedését az UML-ben megtalálható forgatókönyv-leíró nyelv, a *szekvencia diagramok* segítségével adják meg. A profil lehetővé teszi, hogy jelöljék, hogy mikor milyen döntést rendelnek egy lefutáshoz. A tesztadatok leírásánál kihagyott és alapértelmezett értékek definiálhatók.

A szekvencia diagramok UML 1.x-ben megtalálható első verziója a nagyon hasonló volt az egyszerű MSC diagramokhoz, azaz a kommunikáló felek közötti üzeneteket jelenítette meg. Az UML 2-vel megjelent új verzió komoly változásokat hozott, a nyelvet kiegészítették új, összetett elemekkel (amelyek segítségével például le lehet írni párhuzamos illetve vagylagos lefolyásokat). Ezen kívül megjelentek olyan, a tesztelés szempontjából nagyon fontos nyelvi elemek, amiknek a segítségével kötelező, tiltott működést vagy figyelmen kívül hagyható üzeneteket lehet ábrázolni. Azonban ezeknek az elemeknek a jelentését, azaz a *szemantikájukat*, csak természetes nyelvi szövegekkel specifikálták, ami többféle, egymásnak akár ellentmondó értelmezést is lehetővé tett.

Éppen ezért mielőtt a szekvencia diagramokat felhasználnánk tesztelési termékek leírására vagy megpróbálnánk kiegészíteni a nyelvet olyan új elemekkel, amiknek a segítségével leírhatók bizonyos alkalmazási területe sajátosságai, előbb a nyelvben meglévő úgynevezett *szemantikus választási pontokat* kell azonosítani, és meg kell határozni, hogy ezek hogyan befolyásolják a tesztelési célú felhasználásokat.

1.1.3. Az értekezésben felhasznált tesztelési megközelítések

A különböző tesztelési tevékenységek osztályozhatóak annak megfelelően, hogy milyen szinten kell őket végrehajtani. A tipikus kategóriák például a modul vagy egységteszt (amely csak egy modullal foglalkozik), az integrációs teszt (amely már több modul együttműködését vizsgálja) és a rendszerteszt (a teljes rendszer és esetlegesen a környezet vizsgálata). Az értekezésben kidolgozott módszerek a *rendszertesztelés* szintjére fókuszálnak.

A funkcionalitás tesztelésére sokféle módszer adódik még rendszertesztelés esetén is. A módszerek csoportosításának egyik tipikus módja, amelyet a protokolltesztelési kutatásokban gyakran használnak, az *aktív* és *passzív* tesztelés [AMN12] megkülönböztetése. Aktív tesztelés esetén a tesztek a SUT funkcióit közvetlen bemenetekkel hívják meg. Ez azonban néhány esetben nem alkalmazható, például amikor a SUT-nak nincs közvetlen interfésze vagy a SUT egy bonyolult környezet része. Ilyenkor a passzív tesztelési technikák nyújtanak alternatívát a SUT működésének megfigyelésével és egy *végrehajtási nyom* rögzítésével, majd később a nyom ellenőrzésével, hogy az abban észlelt viselkedés vajon megfelel-e a specifikációnak. Ezt a megközelítést gyakran használják elosztott rendszerek esetén, amikor a tesztkeretrendszer nem próbál meg folyamatosan bemeneteket szolgáltatni az egyes csomópontoknak, hanem pusztán csak létrehoz egy kezdeti elrendezést, elindítja ebből a tesztelendő rendszert, és később csak megfigyeli a csomópontok kommunikációját. Az értekezésben vizsgált alkalmazási területeken néhol az aktív, néhol a passzív tesztelés bizonyult hatékonyabb megközelítésnek.

Rendszertesztelés esetén nem csak az alapvető funkcionalitást, hanem további *nemfunkcionális követelményeket* is kell vizsgálni. A nem-funkcionális követelmények magukban foglalják a teljesítményt vagy a szolgáltatásbiztonság különböző tulajdonságait [Avi+04], mint például a rendelkezésre állás vagy a robusztusság. Az ilyen követelményeket vizsgáló tesztek általában két részből állnak: a *munkaterhelés* (workload) a rendszer normál működését váltja ki, míg a *hibaterhelés* (faultload) különböző hibákat és különös igénybevételt jelentő feltételeket tartalmaz. Az alapján, hogy ez a két terhelés hogyan viszonyul egymáshoz, különböző jellemzőket lehet vizsgálni. Például egy API robusztusságának tesztelése esetén csak hibaterhelést hajtunk végre a rendszer publikus interfészén, míg stressztesztet esetén csak nagymértékű munkaterhelést használunk. Az értekezésben bemutatott kutatás egy része a robusztusság vizsgálatára koncentrált. Az IEEE definíciója szerint a robusztusság „annak jellemzője, hogy egy rendszer vagy egy komponens mennyire tud helyesen működni rendkívüli bemenetek vagy nagy igénybevételt jelentő környezeti feltételek mellett”.

Ám azonban ahogyan új alkalmazási területek jelennek meg, ezeket a meglévő nyelveket és

klasszikus módszereket felül kell vizsgálni, hogy nem jelentkeznek-e eddig nem ismert kihívások a használatuk során.

1.2. Új alkalmazási területek

Az értekezés a következő újfajta alkalmazási területeket vizsgálta, amik számos kihívást állítanak a tesztelési tevékenységek elé.

1.2.1. Magas rendelkezésre állású köztesrétegek

A rendelkezésre állás manapság már a mindennap használt számítási platformokon is fontos tényező. A magas rendelkezésre állás (high availablity, HA) eléréséhez menedzselhető redundanciát kell a rendszerünkben biztosítani. Azonban az általános megoldásokat, amikkel ezt a redundanciát kezelni lehet és a kiesések mértékét le lehet csökkenteni, az alkalmazásoktól függetlenül is meg lehet valósítani egy úgynevezett HA köztesrétegben. Az ilyen köztesréteg rendszerek szabványosítására vezető IT cégek megalakították a *Service Availability Forum* (SA Forum) konzorciumot, amely kidolgozta az *Application Interface Specification* (AIS) [SAF07] javaslatot. Az AIS-t több gyártó is implementálta a saját megoldásában.

Természetes igényként jelenik a közös specifikációt megvalósító megoldások összehasonlítása. A leggyakrabban az ilyen összehasonlítások a teljesítményre vagy a funkcionalitásra szorítkoznak, azonban egy HA köztesréteg esetén azok szolgáltatásbiztonsága is egy ugyanolyan fontos tulajdonság.

A HA köztesrétegek sajátosságait a következőkben lehet összefoglalni.

Állapotalapú működés A HA köztesrétegek tesztelésének komplexitása abból adódik, hogy azok működése erősen állapotalapú, azaz a megfelelő inicializáló hívások végrehajtása nélkül, amik a kívánt állapotba hozzák a rendszer különböző komponenseit, a publikus interfész legtöbb hívása rögtön hibajelzéssel tér vissza a funkcionalitás végrehajtása nélkül. Például egy komponens állapotának az ellenőrzése csak egy visszahívandó függvényben lehetséges, amit korábban a kapcsolódáskor regisztrálni kellett.

A robusztusság fontos tulajdonság Mivel az ilyen rendszerekkel szemben támasztott rendelkezésre állási követelmények nagyon szigorúak, a HA köztesrétegeknek még a váratlan szituációkat is kezelniük kell. Általános esetben a tesztelés az összes helyes és a főbb hibás esetek lefedésére koncentrál. Azonban egy HA köztesréteg esetén lehetőleg az összes hibás esetet vizsgálni kell, hiszen egy valamelyik komponensben lévő apró hiba az egész rendszert elérhetetlenné teheti, ha a köztesréteg nem elég robusztus.

A robusztusságot, többek között, teszteléssel is lehet ellenőrizni. A robusztusságtesztelés célja az, hogy aktiválja a rendszerben lévő potenciális robusztussági hibaokokat[2] (tervezési vagy programozói hibák eredményeit) nem megfelelő bemenetek vagy környezeti helyzetek használatával.

Habár a HA köztesrétegek robusztusságának vizsgálata új kutatási terület volt, a korábbi robusztusság tesztelési módszerek jó kiindulási alapot szolgáltattak. A korai kísérletek parancssori programok robusztusságát vizsgálták nagyszámú véletlen bemenet generálásával [MFS90] (ezt a technikát „fuzzing”-nak nevezik). Mivel a vizsgált HA köztesrétegek közös interfésszel rendelkeznek, a Ballista projektben [KDD08] bevezetett típusspecifikus tesztelés módszere is felhasználható. Ott POSIX-alapú operációs rendszerekhez generáltak robusztusság teszteket az API-ban

[2] A klasszikus [Avi+04] fault–error–failure terminológiának a hibaok–hiba–hibajelenség fordítását használjuk.

használt típusokhoz tartozó helyes és helytelen értékek megadásával. Ez a módszer adaptálható a HA köztesrétegek teszteléséhez is.

A HA köztesrétegekhez tartozó tesztek leírásában a következők a kihívások: (i) a meglévő tesztelési nyelvek főleg a specifikációnak való megfelelésre és nem a robusztusságra koncentráltak, (ii) egy HA köztesréteg egy összetett rendszer, aminek sokféle bemenete idézhet elő hibajelenséget, és (iii) az API-ja állapotalapú, amiben ráadásul sok függvény, típus és paraméter van definiálva.

1.2.2. Kontextusra érzékeny mobil számítási rendszerek

A mobil számítási rendszerek olyan eszközökből állnak (pl. laptop, okostelefon, intelligens autó stb.), amik képesek mozogni egy adott területen belül, és közben különböző hálózatokhoz csatlakozhatnak vezeték nélküli kapcsolatok segítségével. Az ilyen rendszerek sajátos jellemzői a következők.

Kontextusra érzékenység A kontextus lehet „bármilyen információ, ami felhasználható a felhasználó és az alkalmazás közötti interakciók szempontjából releváns szituációk meghatározására" [BDR07]. Például mobil rendszerek esetén ez az információ lehet a fizikai szenzorok által gyűjtött adat (például hely, idő, sebesség) vagy akár az aktuális hálózati kapcsolat valamilyen paramétere. Az ilyen rendszereknek működésük során figyelembe kell venniük az ilyen kontextus információkat is.

Dinamikus, változó környezet Mobil számítási rendszerek esetén a rendszer felépítése, az abban résztvevő mobil eszközök száma nem fix. A struktúra változik az idővel, ahogy új csomópontok jelennek meg vagy bizonyos eszközöket leállítanak. Azonban nem csak a csomópontok száma, hanem a közöttük lévő kapcsolatok tulajdonságai is változnak. Ahogy a csomópontok folyamatosan mozoghatnak, össze- és szétkapcsolódhatnak más csomópontokkal. Kommunikációs kapcsolatok jöhetnek létre vagy tűnhetnek el, így a kommunikációs topológia is folyamatosan változik.

Helyi kommunikáció ismeretlen felekkel Ad hoc mobil hálózatok esetén egy természetes kommunikációs forma a helyi üzenetszórás (broadcast). Ezt használja például sok mobil alkalmazásban lévő felderítési réteg (pl. csoporttagsági vagy útvonal-felderítési protokoll). Mivel a rendszer topológiája nem ismert, ezért a küldő csomópont nem tudja előre, hogy kihez fog eljutni az üzenete. Az adási távolságon belül mindenki megkaphatja és később bárki válaszolhat.

A korábban bemutatott teszteléshez használt nyelvek (lásd 1.1.1. szakasz) főleg statikus elrendezések leírásához lettek kifejlesztve. Objektumok létrejöttét vagy törlését meg lehet ugyan jeleníteni néhány nyelvben, de ezek általában nem alkalmasak arra, hogy kommunikációs felek gyakori megjelenését vagy eltűnését ábrázolni lehessen velük. Voltak korábbi munkák, amik ennek leküzdésére kiegészítéseket javasoltak (pl. [BM04; SE04]), de ezek főleg mobil ágensekre vagy logikai kód mobilitásra összpontosítottak. Ezen kívül a meglévő nyelvek a felek közötti üzenetekre koncentrálnak elsősorban, és nehézkes annak a megadása, hogy mi az aktuális üzenet kontextusa, azaz milyen legyen akkor a rendszer vagy a környezet állapota. Tehát kiterjesztések szükséges az eddigi modellezési nyelvekhez, amik segítenek figyelembe venni a kontextusra érzékeny mobil rendszerek sajátosságait.

1.3. Az új kihívások összefoglalása

Ahogy a korábbi részek mutatták, léteznek ugyan kapcsolódó tesztelési megközelítések és nyelvek, azonban ezeket hozzá kell illeszteni vagy ki kell terjeszteni a bemutatott két alkalmazási területhez. A következő kihívásokban foglalható össze a disszertációban bemutatott kutatás motivációja.

1. kihívás: Robusztusságtesztelés kiterjesztése HA köztesrétegekre. Hogyan lehet a tesztbemeneteket megfogalmazni HA köztesrétegek esetén tesztelési termékekben, hogy azok később támogassák az ilyen rendszerek automatikus robusztusságtesztelését?

2. kihívás: Mobil rendszerek leírása tesztelési termékekben. Hogyan lehet dinamikus, gyakran változó struktúrákat és ismeretlen kommunikációs feleket megadni tesztelési termékekben úgy, hogy azok később ellenőrizhetőek legyenek a tesztelés során?

Egy teljesen új tesztelési nyelv kidolgozása helyett megpróbáltuk felhasználni a meglévő nyelveket amennyire csak lehetséges volt. Azonban az újrafelhasználáshoz többek között az szükséges, hogy az adott nyelvnek egyértelmű és precíz szemantikája legyen. A korábbi forgatókönyvleíró nyelvek közül mi az UML 2 szekvencia diagramokra koncentráltunk. De, mint az korábban már szerepelt, a szekvencia diagramoknak sokféle értelmezése lehetséges. Ezért ahhoz, hogy a teszteléshez szükséges kiegészítéseket definiáljuk, előbb az UML 2 szekvencia diagramok szemantikáját kell részletesen tanulmányozni.

3. kihívás: UML 2 szekvencia diagramok szemantikájának vizsgálata. Milyen szemantikus választási lehetőségek vannak az UML 2 szekvencia diagramokban, mik a lehetséges opciók, amiket figyelembe kell venni egy esetleges teszteléssel kapcsolatos kiegészítés bevezetése során?

Ezek alapján az értekezés célja az volt, hogy olyan teszt keretrendszereket definiáljon, amik választ adnak a fenti kihívásokra, és létrehozzon olyan nyelveket, amik ahhoz szükségesek, hogy a tesztkeretrendszerekben a tesztelési termékeket le tudjuk írni.

2. Kutatási módszertan és új tudományos eredmények

Solheim és Stølen szerint [SS07] egy kutatási tevékenység lehet *alapkutatás* (kutatásnak, amelynek célja új ismeretek szerzése) vagy *alkalmazott kutatás* (kutatás, amelynek célja gyakorlati problémák megoldása). Ezen kívül megkülönböztethetjük a *klasszikus kutatási* módszertant (amely hipotézist állít fel, majd azt igazolja kísérletekkel vagy megfigyelésekkel) és a *technológiai kutatási* módszertant (amelynek célja új és jobb termékek[3] készítése). A technológiai kutatás általában az alkalmazott kutatások közé tartozik, és legtöbbször ez is egy iteratív folyamat: (i) a megoldandó probléma vizsgálata során összegyűjtjük az új termékekkel kapcsolatos elvárásokat, (ii) megalkotjuk az új terméket valamilyen innovatív módszer alkalmazásával, majd (iii) az elkészült új terméket kiértékeljük az elvárásoknak megfelelően.

Az értekezésben szereplő kutatást alkalmazott, technológiai kutatásként lehet kategorizálni, hiszen a célja az, hogy jobb tesztelési termékeket javasoljon gyakorlati problémák megoldásához. Az értekezésben szereplő új termékek egy része új modellezési nyelv, ezért fontos áttekinteni azt, hogy milyen módszerek segítségével lehet egy új modellezési nyelvet megalkotni.

[3] Az angol eredeti erre az „artefact” szóval hivatkozik, amit jobb híján termékként fordítottunk, de ez nem azonos a „product” szó jelentésével.

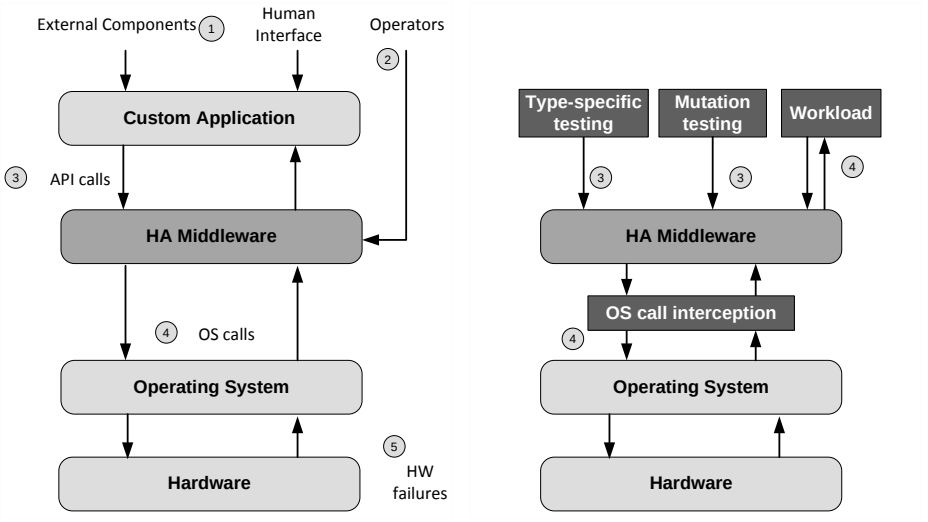
Nyelvek definiálása Az azonosított kihívások megoldásához szükséges volt meglévő modellezési nyelveket kiegészíteni és újakat alkotni. Egy új nyelv létrehozása összetett feladat, a pontos definícióhoz a következő alkotóelemek megadása szükséges [OMG11a].

- *Absztrakt szintaxis*: megadja a nyelv főbb fogalmi elemeit és azok kapcsolatait. Az absztrakt szintaxist az automatikus feldolgozás során használjuk fel, és grafikus modellezési nyelvek esetén tipikusan metamodellek segítségével adjuk meg.
- *Konkrét szintaxis*: megadja a nyelv emberek számára szánt interfészét (grafikus vagy szöveges jelölések). A konkrét szintaxis elemeit az absztrakt szintaxis elemeihez rendeljük hozzá.
- *Jólformáltsági szabályok*: további kényszereket definiálnak az absztrakt szintaxis elemeire, amikkel összetettebb feltételeket lehet megfogalmazni, amiket a helyes modelleknek teljesíteniük kell.
- *Szemantika*: megadja a nyelvi elemek jelentését, tipikusan egy korábban már pontosan definiált szemantikus tartományra való leképezéssel.

A fejezet további részei összefoglalják az értekezés új tudományos eredményeit, amelyek a korábban azonosított kihívásokat oldják meg.

2.1. HA köztesrétegek robusztusságának tesztelése

Az AIS specifikációnak megfelelő köztesrétegek tesztelési megközelítésének kidolgozása során az első lépés azoknak a lehetséges bemeneteknek az azonosítása volt, amik képesek az esetleges robusztussági hibaokokat aktiválni. A 3a. ábra ezeket a bemeneti forrásokat ábrázolja egy elosztott HA rendszer csomópontján.



(a) Robusztussági hibaokok aktiválása (b) Robusztusságtesztelési technikák

3. ábra. A HA köztesrétegek tesztelési megközelítésének áttekintése

A kidolgozott tesztelési megközelítés a következő közvetlen forrásokra összpontosított, mert ezeknek a forrásoknak az alapos tesztelése a közvetett források által aktiválható hibák nagy részét is lefedi.

A köztesréteg szabványosított API-jának hívásai azért kiemelten fontosak a robusztusság vizsgálatának szempontjából, mert az alkalmazásokban és a külső komponensekben lévő hibák és az emberi hibák ezen keresztül érhetik el a köztesréteget. Az API-hívások tesztelése azért jelent kihívást, mert az AIS interfészének függvényei állapotalapúak, azaz a használatukhoz megfelelő inicializáló hívások, köztesréteg beállítások és egyéb tesztkomponensek szükségesek, egyébként azonnal hibajelzéssel térnek vissza.

Az operációs rendszer (OS) rendszerhívásainak hibái azért kerültek bele a tesztelési megközelítésbe, mert azok nem csak magának az OS-nek a hibáit képviselik (ami a mai kiforrott operációs rendszerek esetében elég ritka), hanem a rendszer további szoftverkomponenseinek, a hardvernek és a közvetlen környezetnek a hibái is a rendszerhívások hibaüzeneteiben jelennek meg. Például egy lemez írási hibájáról vagy hálózati kommunikációs hibáról is egy rendszerhívás visszatérési értékén keresztül értesül a köztesréteg.

Ezek alapján az alábbi tesztelési megközelítést dolgoztuk ki, ami a fenti hibák lefedésére három technika kombinációját alkalmazza (lásd 3b. ábra).

- *Típusspecifikus tesztelés*: A köztesréteg API függvényeinek robusztusságát tesztelni kell abban az esetben, ha nem megfelelő értékeket használunk paraméterként. Ehhez az összes függvényt meg kell hívni a lehetséges helyes és helytelen bemeneti értékek fontosabb kombinációival. A típusspecifikus tesztelési technika egy ilyen tesztkészlet rendszerezett előállítását teszi lehetővé a szükséges helyes és helytelen értékek típusonkénti megadásával.

- *Mutációalapú tesztelés*: A köztesréteg bizonyos állapotai csak összetett hívási sorozatok segítségével érhetőek el, ám az API robusztusságát ezekben az állapotokban is ellenőrizni kell. Ehhez tehát először a köztesréteget ezekbe az állapotokba kell vezetni, majd innen meghívni az API függvényeit helytelen bemenetekkel. Az értekezésben javasolt megoldás azt használja fel, hogy a köztesréteg meglévő funkcionális tesztkészlete felhasználható ezeknek az állapotoknak az elérésére, hiszen az abban szereplő tesztesetek általában lefedik az összes fontos rendszerállapotot. Így tehát oly módon lehet előállítani robusztussági teszteseteket, hogy mutációs operátorok segítségével módosítjuk a funkcionális tesztkészlet elemeit úgy, hogy azok tipikus hibaokokat utánozzanak (pl. hívási sorrend felcserélése, paraméterértékek lecserélése hibásra).

- *OS hívások eltérítése*: Annak megvizsgálására, hogy hogyan viselkedik a köztesréteg a felhasznált alsóbb szintű szolgáltatások hibajelenségei esetén, az OS rendszerhívásait eltérítjük és módosítjuk. Ennek a megvalósításához egy csomagoló (wrapper) komponenst használunk, ami a köztesréteg és az operációs rendszer könyvtárai közé ékelődik be. Ezen kívül az ilyen tesztelési technika alkalmazásához szükséges még egy munkaterhelést biztosító alkalmazás is, ami biztosítja, hogy a köztesrétegnek meg kelljen hívnia a lehetséges OS rendszerhívásokat.

A robusztussági tesztkeretrendszer ezt a három technikát a következő módszert felhasználva valósította meg:

1. Először a szükséges tesztelési termékek és azok követelményeit gyűjtöttük össze.

2. Ezután a szükséges teszt nyelveket dolgoztuk ki, amikkel le lehet írni az elkészítendő tesztelési termékeket.

3. Végül olyan eszközöket készítettünk el, amik képesek a nyelvi leírásokból generálni a tesztelési termékeket.

A származtatott robusztussági tesztkészletet három különböző köztesréteg megvalósításon hajtottuk végre több kísérlet során, hogy össze tudjuk hasonlítani azok robusztusságát. A kísérletek sikeresen azonosítottak többfajta robusztussági hibajelenséget az egyes megvalósításokban.

1. tézis *Azonosítottam a robusztussági hibaokok lehetséges aktiválási módjait (állapotmentes és állapotalapú API hívásokon valamint alsóbb szolgáltatásokon keresztül), és megterveztem olyan nyelveket, amikkel le lehet írni a szükséges tesztelési termékeket. Megalkottam olyan algoritmusokat és eszközöket, amikkel ezekből a nyelvi leírásokból teszteket lehet generálni. Megvalósítottam ezeket a nyelveket és eszközöket egy olyan tesztkeretrendszerben, ami képes közös specifikáción alapuló HA köztesrétegek robusztusságát összehasonítani.*

Az 1. tézis eredményeit szolgáltató kutatásokat Fransic Tammel (Nokia Research Center) közösen végeztem. Az ő hozzájárulása a kutatás átfogó irányának és az elkészítendő eszközöknek a kijelölése volt [Tam09].

Az 1. tézis eredményeit az értekezés 2. fejezete mutatja be. A tézishez a következő publikációk kapcsolódnak: [2], [5], [6], [8], [10], [11], [12].

2.2. Szemantikus választások az UML 2 szekvencia diagramokban

Amikor először vizsgáltuk, hogy hogyan lehet tesztkövetelményeket leírni a mobil rendszerek területén UML 2 szekvencia diagramok segítségével, akkor abba a problémába ütköztünk, hogy a szekvencia diagramokhoz javasolt különböző formális szemantikák még a legalapvetőbb diagramokat is máshogy kezelik. Kiderült, hogy rengeteg apró variáció van az egyes nyelvi elemek értelmezésében. Ráadásul ezek a választási lehetőségek és azok következményei a legtöbb esetben nem egyértelműek. Éppen ezért szükség volt arra, hogy áttekintsük a korábbi, szemantikával kapcsolatos megközelítéséket a tesztkövetelményeket leíró nyelv kidolgozása előtt.

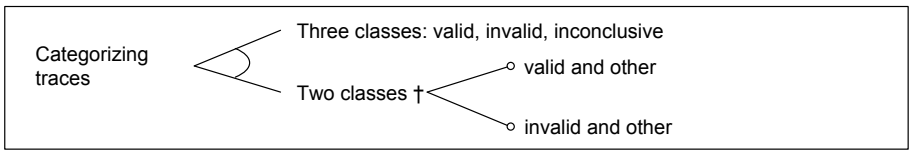
A kutatás során a következő módszert alkalmaztuk.

1. Áttekintettük a kapcsolódó irodalmat, és kielemeztük az UML 2 szekvencia diagramokhoz javasolt formális szemantikákat.
2. Összegyűjtöttük és osztályoztuk a szabványban és a korábbi megoldásokban megtalálható szemantikus választásokat.
3. Az egyes választásokhoz tartozó lehetőségek következményeit kiértékeltük.

Mivel az így összegyűjtött választási lehetőségek másoknak is hasznosak lehetnek, akik a szekvencia diagramokat fel szeretnék használni vagy ki szeretnék terjeszteni, ezért az eredmények bemutatásához egy könnyen érthető ábrázolási formát kerestünk. Általában a formális szemantikákat azért használják fel később kevesen, mert a megértésükhöz komoly elméleti jártasság szükséges, éppen ezért mi egy gyakorlati szakemberek, mérnökök által is könnyen érthető jelölésrendszert használtunk (a jelölések az úgynevezett feature modeleken alapulnak [Kan+90]). A 4. ábra egy példát mutat a különböző lehetőségek ábrázolására az általunk kidolgozott kategorizálásból. A példában a *Categorizing traces* választásnak két egymást kizáró lehetősége van, amik közül az egyiket még tovább lehet finomítani.

Az 1. táblázat[4] összefoglalja a beazonosított magasszintű szemantikai választási lehetőségeket, amiket meg kell vizsgálni a szekvencia diagramok alkalmazása előtt. A választási lehetőségek a legalapvetőbb kérdésektől (pl. mit is tartalmaz egyáltalán egy futási nyom) az összetett nyelvi elemekig terjednek.

[4] A táblázatban az UML-metamodell elemei az eredeti angol elnevezésekkel szerepelnek.



4. ábra. A *Categorizing traces* választáshoz tartozó lehetőségek

2. tézis *Azonosítottam és kategorizáltam a szemantikus választási lehetőségeket az UML 2 szekvencia diagramokban. A lehetőségeket egy strukturált keretbe rendeztem, amelyhez egy könnyen érthető jelölésrendszer tartozik, amit fel lehet később használni a nyelv szemantikájának adott területhez való alakításához.*

A 2. tézis eredményeit az értekezés 3. fejezete mutatja be. A tézishez a következő publikációk kapcsolódnak: [1], [13].

1. táblázat. Az UML 2 szekvencia diagramok szemantikus választási lehetőségei

Egy egyszerű Interaction értelmezése	Mit tartalmaz egy nyom? Nyomok lehetséges kategorizálása Teljes és részleges nyomok
CombinedFragment bevezetése	Töredékek összefűzése
Részleges rendezések számítása	A diagram feldolgozása Alap formalizmusok Választások és predikátumok
Gate bevezetése	Gate és CombinedFragment Formal és actual típusú Gate
Megfelelőségi operátorok értelmezése	Assert és Negate Ignore és Consider Megfelelőségi operátorok összetett diagramokban Egyszerre érvényes és érvénytelen nyomok

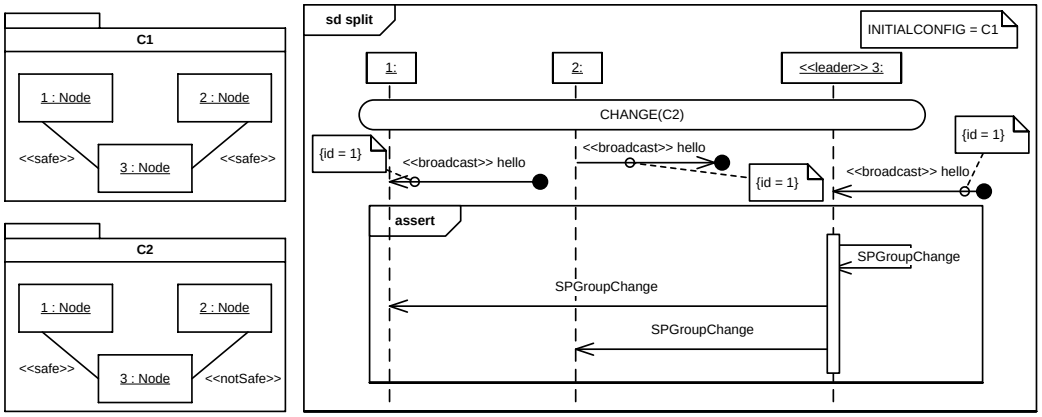
2.3. Tesztelési nyelv és keretrendszer mobil rendszerekhez

Először, hogy jobban megértsük a mobil rendszerek tesztelésével kapcsolatos kihívásokat, egy esettanulmányt [7] végeztünk egy mobil csoporttagsági protokoll tesztelésével. Az esettanulmány tapasztalatai a következők voltak. Az UML alkalmas volt egy csomópont felépítésének és működésének modellezésére, de nem volt megfelelő a több csomópont közötti viselkedés leírására. Egyrészt, mint azt korábban említettük, a szekvencia diagramok szemantikája nem egyértelmű, másrészt mobil rendszerek esetén az alkalmazások működése nem csak az aktuális bemenettől függ, hanem fontos a környezeti információ is, például a jelenlegi hely adatai. A tesztelést végrehajtó motornak nem csak a SUT-nak szánt üzeneteket kell megadnia, hanem be kell tudnia állítani ezeket a környezeti információkat is.

Ezen tapasztalatok alapján kiegészítéseket javasoltunk forgatókönyv-leíró nyelvekhez a környezeti információ megadására, és a kiegészítéseket beépítettük az UML 2 szekvencia diagramok nyelvébe, így létrehozva a TERMOS (Test Requirement language for Mobile Setting) nyelvet, amellyel tesztkövetelményeket lehet leírni mobil rendszerek esetén.

- A nyelv absztrakt szintaxisa egyrészt egy UML-profil segítségével lett megadva (ami az UML javasolt kiegészítési lehetősége), másrészt a szekvencia diagramok metamodelljében további kényszereket és korlátozásokat fogalmaztunk meg.
- A konkrét szintaxis a meglévő grafikus elemeket használta fel. Így a TERMOS nyelven megadott követelményeket a meglévő UML eszközök segítségével le lehet írni, ami megkönnyíti az új nyelv használatát.
- További jólformáltsági kényszerek lettek definiálva, hogy ne lehessen olyan követelményeket létrehozni, amiket később nehezen lehet ellenőrizni (például egymásba ágyazott tagadó operátorok).
- A nyelv szemantikájának definiálása a 2. tézisben azonosított szemantikus választási lehetőségek alapján történt. A formális szemantika egy, az LSC nyelvhez javasolt szemantikán [Klo03] alapult, ami minden diagramhoz egy véges automatát rendel.

Az 5. ábra egy példát mutat egy mobil rendszerhez megfogalmazott tesztkövetelményre. Az ábra bal oldala a környezetet leíró térbeli nézetet tartalmazza, míg a jobb oldal a kommunikációs üzenetek viszonyát adja meg a konfigurációs változásokhoz képest.



(a) Térbeli nézet (b) Esemény nézet

5. ábra. Példa követelmény egy mobil rendszerhez

A tesztkeretrendszerben szükséges továbbá még egy módszer, amivel a követelményeket ki lehet értékelni. A tesztek futtatása során részletes nyomokat rögzítünk, amik az üzeneteket és a kommunikációs topológiában bekövetkezett változásokat tartalmazzák. Ezek után a módszer beazonosítja, hogy a követelménybeli szerepeknek melyik valós objektumok felelhetnek meg. Végül a követelményben megfogalmazott modalitások alapján (kötelező, tilos) egy-egy döntés rendelődik a tesztfuttatásokhoz.

3. tézis *Megalkottam egy nyelvet tesztkövetelmények leírására, amely mobil rendszerekben használható. A nyelv szintaktikáját az UML metamodelljéhez definiált kiegészítésekkel adtam meg, a*

szemantikáját pedig egy automataalapú formális műveleti szemantika segítségével definiáltam. A nyelv képes helyi üzenetszórás valamint a topológia változásának kifejezésére, és rendelkezik a megfelelő korlátozásokkal, hogy ellenőrizhetőek legyenek később a megfogalmazott követelmények.

A mobil rendszerekhez szánt tesztkeretrendszer kidolgozása egy közös kutatás kereteiben történt Nicolas Rivière-rel és Minh Duc Nguyennel (LAAS-CNRS). A tesztnyomok és követelmények párosítását elvégző GraphSeq eszköz Minh Duc Nguyen PhD értekezésének eredménye volt [Ngu09]. Hamvas Áron, akinek én voltam a diplomakonzulense, implementált egy eszközt a követelmények kiértékelésének megvalósítására.

A 3. tézis eredményeit az értekezés 4. fejezete mutatja be. A tézishez a következő publikációk kapcsolódnak: [3], [4], [7], [9], [14].

3. Az eredmények gyakorlati alkalmazása

A PhD értekezés eredményeinek gyakorlati alkalmazásai a következőképpen foglalhatóak össze.

3.1. AIS-alapú köztesrétegek robusztusságának összehasonlítása

Az *1. tézisben* bemutatott módszert több különböző HA köztesréteg megoldás robusztusságának tesztelésére használtam fel.

- A robusztussági tesztkészletet végrehajtottam három különböző megvalósításon: a nyílt forráskódú OpenAIS [Opea] és OpenSAF [Opeb] rendszereken, valamint a Fujitsu-Siemens cég SAFE4TRY termékén. A tesztelések eredményeit több helyen is publikáltuk [5], [6], [8].
- Az OpenAIS rendszerben felderített hibákat jelentettem a szoftver fejlesztői közösségének, majd később a teljes tesztkészletet publikussá és forráskóddal együtt elérhetővé tettük [BME07].
- Az OpenAIS és OpenSAF rendszerek robusztusságának tesztelésével kapott eredményeket feltöltöttük a nyilvános AMBER Data Repository (ADR) [AMM10] tárhelyre. Az ADR egy olyan tárhely, ahol mérési és benchmark eredményeket lehet megosztani és elemezni.

3.2. Mobil rendszerek tesztelése

A *2. tézisben* bemutatott csoportosítást, amely az UML 2 szekvencia diagramok nyelvében megtalálható szemantikus választásokat rendszerezte, használtuk fel a *3. tézis*ben szereplő TERMOS nyelv megalkotása során [4].

A TERMOS nyelvet a HIDENETS EU FP6 kutatási projekt [HID09] keretein belül alkottuk meg. A HIDENETS projekt (Highly dependable IP-based networks and services) olyan módszereket dolgozott ki, amelyeknek a segítségével elosztott, mobilitást is támogató alkalmazások (például autók és infrastruktúra-csomópontok közötti kommunikáció) ellenálló képessége és szolgáltatásbiztonsága vizsgálható meg.

Ezeket az eredményeket később felhasználtuk és továbbfejlesztettük a folyamatban lévő ARTEMIS R3-COP kutatási projektben [R3C11] egy autonóm rendszerekhez használható tesztelési nyelvben. Az R3-COP projekt (Resilient Reasoning Robotic Co-operating Systems) olyan módszerek és technikák kifejlesztését tűzte ki célul, amiknek a segítségével biztonságos és robusztus robotok és autonóm rendszerek fejlesztését lehet támogatni.

4. Publikációs lista

Lektorált publikációk száma:	16
Független hivatkozások száma:	33

4.1. A tézisekhez kapcsolódó publikációk

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. tézis:		•			•	•		•		•	•	•		
2. tézis:	•												•	
3. tézis:			•	•			•		•					•

Folyóiratcikk

[1] Z. Micskei and H. Waeselynck. "The many meanings of UML 2 Sequence Diagrams: a survey". In: *Software and Systems Modeling* 10.4 (2011), pp. 489–514. DOI: 10.1007/s10270-010-0157-9

Fejezet szerkesztett könyvben

[2] Z. Micskei, H. Madeira, A. Avritzer, I. Majzik, M. Vieira, and N. Antunes. "Robustness Testing Techniques and Tools". In: *Resilience Assessment and Evaluation of Computing Systems*. Ed. by K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel. Springer, 2012, pp. 323–339. DOI: 10.1007/978-3-642-29032-9_16

A saját hozzájárulásom a robusztusság tesztelést áttekintő szakasz volt a fejezetben.

[3] G. Pintér, Z. Micskei, A. Kövi, Z. Égel, I. Kocsis, G. Huszerl, and A. Pataricza. "Model-Based Approaches for Dependability in Ad-Hoc Mobile Networks and Services". In: *Architecting Dependable Systems V*. Springer, 2008, pp. 150–174. DOI: 10.1007/978-3-540-85571-2_7

A mobil rendszerek tesztelését leíró szakasz volt a saját hozzájárulásom.

Nemzetközi konferencia

[4] H. Waeselynck, Z. Micskei, N. Rivière, Á. Hamvas, and I. Nitu. "TERMOS: a Formal Language for Scenarios in Mobile Computing Systems". In: *Mobile and Ubiquitous Systems: Computing, Networking, and Services (MobiQuitous 2010)*. Ed. by P. Sénac, M. Ott, and A. Seneviratne. Sydney, Australia, Dec. 2010, pp. 285–296. DOI: 10.1007/978-3-642-29154-8_24

A különálló térbeli és esemény nézet, valamint a teszt keretrendszer ötlete közös, oszthatatlan hozzájárulás. A saját hozzájárulásom a TERMOS nyelv szintaxisának és szematikájának megtervezése volt.

[5] A. Kövi and Z. Micskei. "Robustness Testing of Standard Specifications-Based HA Middleware". In: *30th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, June 2010, pp. 302–306. DOI: 10.1109/ICDCSW.2010.73

A robusztusság tesztelési keretrendszer (a megközelítés és az eszközök) a saját hozzájárulásom. Kövi András készítette elő, hajtotta végre és elemezte a teszteket az OpenSAF köztesrétegen.

[6] Z. Micskei, I. Majzik, and F. Tam. "Comparing Robustness of AIS-Based Middleware Implementations". In: *International Service Availability Symposium (ISAS 2007)*. Ed. by M. Malek, M. Reitenspieß, and A. van Moorsel. Vol. 4526. LNCS. Springer, May 2007, pp. 20–30. DOI: 10.1007/978-3-540-72736-1_3

A robusztusság tesztelési keretrendszer ötlete közös hozzájárulás. A keretrendszer eszközeinek implementálása és a tesztelési kísérletek végrehajtása a saját hozzájárulásom.

[7] H. Waeselynck, Z. Micskei, M. D. Nguyen, and N. Rivière. "Mobile Systems from a Validation Perspective: a Case Study". In: *Sixth International Symposium on Parallel and Distributed Computing (ISPDC '07)*. Ed. by D. Kranzlmüller, W. Schreiner, and J. Volkert. Hagenberg, Austria: IEEE Computer Society, July 2007, pp. 85–92. DOI: 10.1109/ISPDC.2007.37

A specifikáció és az implementáció statikus ellenőrzése a saját hozzájárulásom a cikkben. Minh Duc Nguyen végezte el a GMP protokoll dinamikus tesztelését.

[8] Z. Micskei, I. Majzik, and F. Tam. "Robustness Testing Techniques for High Availability Middleware Solutions". In: *International Workshop on Engineering of Fault Tolerant Systems (EFTS2006)*. Luxembourg, Luxembourg: University of Luxembourg, June 2006, pp. 55–66

A robusztusság tesztelési keretrendszer ötlete közös hozzájárulás. Az egyes teszt eszközök megtervezése és megvalósítása a saját hozzájárulásom.

Helyi konferencia

[9] Z. Micskei. "Specifying Tests for Ad-Hoc Mobile Systems". In: *15th PhD Mini-Symposium*. Budapest University of Technology and Economics. Budapest, Hungary: Department of Measurement and Information Systems, Feb. 2008, pp. 32–35

[10] Z. Micskei. "Robustness Comparison of High Availability Middleware Systems". In: *14th PhD Mini-Symposium*. Budapest University of Technology and Economics. Budapest, Hungary: Department of Measurement and Information Systems, Feb. 2007, pp. 70–73

[11] Z. Micskei. "Robustness Testing of High Availability Middleware Solutions". In: *13th PhD Mini-Symposium*. Budapest University of Technology and Economics. Budapest, Hungary: Department of Measurement and Information Systems, Feb. 2006, pp. 36–37

[12] Z. Micskei. "Nagy Rendelkezésre Állást Biztosító Köztes Rétegek Robosztusság Tesztelése". In: *Proceedings of Tavaszi Szél 2006*. Kaposvár, Hungary: Doktoranduszok Országos Szövetsége, May 2006, pp. 295–298

Kutatási jelentés

[13] Z. Micskei and H. Waeselynck. *A survey of UML 2.0 sequence diagrams' semantics*. Tech. rep. 08389. Laboratoire d'Analyse et d'Architecture des Systemes (LAAS), Aug. 2008, pp. 1–37

[14] Z. Micskei, H. Waeselynck, M. D. Nguyen, and N. Rivière. *Analysis of a group membership protocol for Ad-hoc networks*. Tech. rep. 06797. Laboratoire d'Analyse et d'Architecture des Systemes (LAAS), Nov. 2006, pp. 1–42

Az esettanulmány értékelése oszthatatlan hozzájárulás. A saját hozzájárulásom a specifikáció statikus átolvasása és az implementáció visszatervezése.

4.2. További publikációk

Folyóiratcikk

[15] I. Kocsis, A. Pataricza, Z. Micskei, A. Kövi, and Z. Kocsis. "Analytics of Resource Transients in Cloud Based Applications". In: *Int. Journal of Cloud Computing* (). To appear. URL: http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijcc

[16] G. Pintér, Z. Micskei, and I. Majzik. "Supporting design and development of safety critical applications by model based tools". In: *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös nominatae Sectio Computatorica* XXX (2009), pp. 61–78

Nemzetközi konferencia

[17] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik. "A Concept for Testing Robustness and Safety of the Context-Aware Behaviour of Autonomous Systems". In: *Agent and Multi-Agent Systems. Technologies and Applications*. Ed. by G. Jezic, M. Kusek, N.-T. Nguyen, R. Howlett, and L. Jain. Vol. 7327. LNCS. Springer, June 2012, pp. 504–513. DOI: 10.1007/978-3-642-30947-2_55

[18] I. Kocsis, A. Pataricza, Z. Micskei, I. Szombath, A. Kövi, and Z. Kocsis. "Cloud Based Analytics for Cloud Based Applications". In: *1st International IBM Cloud Academy Conference*. Research Triangle Park, USA, Apr. 2012, pp. 1–23

[19] F. Bouquet, R. Breu, J. Jurjens, F. Massacci, V. Meduri, Z. Micskei, F. Piessens, K. Stolen, and D. Varró. "SecureChange: Security Engineering for Lifelong Evolvable Systems". In: *European Future Technologies Conference and Exhibition (FET09)*. Poster Session. Prague, Czech Republic, Apr. 2009, pp. 101–102

[20] L. Gönczy, I. Majzik, A. Horváth, D. Varró, A. Balogh, Z. Micskei, and A. Pataricza. "Tool Support for Engineering Certifiable Software". In: *Electronic Notes in Theoretical Computer Science* 238.4 (2009). Proc. of 1st Workshop on Certification of Safety-Critical Software Controlled Systems (SafeCert 2008), pp. 79–85. DOI: 10.1016/j.entcs.2009.09.008

[21] I. Majzik, Z. Micskei, and G. Pintér. "Development of Model Based Tools to Support the Design of Railway Control Applications". In: *Computer Safety, Reliability, and Security*. Ed. by F. Saglietti and N. Oster. Vol. 4680. LNCS. Springer Berlin / Heidelberg, Sept. 2007, pp. 430–435. DOI: 10.1007/978-3-540-75101-4_41

[22] G. Pintér, Z. Micskei, and I. Majzik. "Supporting Design and Development of Safety Critical Applications by Model Based Tools". In: *10th Symposium on Programming Languages and Software Tools (SPLST 2007)*. Ed. by Z. Horváth, L. Kozma, and V. Zsók. Dobogókő, Hungary: Eotvos University Press, June 2007, pp. 61–75

[23] Z. Micskei and I. Majzik. "Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers". In: *Dependability of Computer Systems (DepCoS-RELCOMEX 2006)*. IEEE Computer Society, May 2006, pp. 191–198. DOI: 10.1109/DEPCOS-RELCOMEX.2006.37

Helyi konferencia

[24] Z. Micskei. "Automatikus tesztgenerálás modell ellenőrzővel". In: *X. Fiatal Műszakiak Tudományos Ülésszaka (FMTU)*. ed. by E. Bitay. Erdélyi Múzeum Egyesület. Kolozsvár, Románia, Mar. 2005, pp. 47–50

Hivatkozások

[AMM10] R. Almeida, N. Mendes, and H. Madeira. "Sharing Experimental and Field Data: The AMBER Raw Data Repository Experience". In: *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*. 2010, pp. 313–320. DOI: 10.1109/ICDCSW.2010.75.

[AMN12] C. Andrés, M. G. Merayo, and M. Núñez. "Formal passive testing of timed systems: theory and tools". In: *Software Testing, Verification and Reliability* (2012). DOI: 10.1002/stvr.1464.

[Avi+04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Trans. Dependable Secur. Comput.* 1 (2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[BDR07] M. Baldauf, S. Dustdar, and F. Rosenberg. "A survey on context-aware systems". In: *Int. J. Ad Hoc Ubiquitous Comput.* 2.4 (2007), pp. 263–277. DOI: 10.1504/IJAHUC.2007.014070.

[Bei90] B. Beizer. *Software Testing Techniques*. 2nd Edition. International Thomson Press, 1990.

[BL02] L. Briand and Y. Labiche. "A UML-Based Approach to System Testing". In: *Software and Systems Modeling* 1.1 (2002), pp. 10–42. DOI: 10.1007/s10270-002-0004-8.

[BM04] E. Belloni and C. Marcos. "MAM-UML: An UML Profile for the Modeling of Mobile-Agent Applications". In: *Chilean Computer Science Society, International Conference of the*. 2004, pp. 3–13. DOI: 10.1109/QEST.2004.14.

[BME07] BME. *Robustness test suite for OpenAIS framework*. 2007. URL: http://mit.bme.hu/~micskeiz/pages/robustness_testing.html#aisrobustness.

[Coo12] S. Cook. "Looking back at UML". In: *Software and Systems Modeling* (2012), pp. 1–10. DOI: 10.1007/s10270-012-0256-x.

[DH01] W. Damm and D. Harel. "LSCs: Breathing Life into Message Sequence Charts". In: *Formal Methods in System Design* 19.1 (2001), pp. 45–80. DOI: 10.1023/A:1011227529550.

[Hes+08] A. Hessel, K. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. "Testing Real-Time Systems Using UPPAAL". In: *Formal Methods and Testing*. Vol. 4949. LNCS. 2008, pp. 77–117. DOI: 10.1007/978-3-540-78917-8_3.

[HID09] HIDENETS. *Highly dependable ip-based networks and services*. EU FP6 Specific Targeted Research Project (STREP), IST 026979. 2009. URL: http://www.hidenets.aau.dk.

[Hon+01] H. S. Hong, I. Lee, O. Sokolsky, and S. D. Cha. "Automatic Test Generation from Statecharts Using Model Checking". In: *In Proceedings of FATES'01, Workshop on Formal Approaches to Testing of Software, volume NS-01-4 of BRICS Notes Series*. 2001, pp. 15–30.

[IEE10] Institute of Electrical and Electronics Engineers. *Systems and software engineering – Vocabulary*. Standard 24765:2010. 2010, pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835.

[IEE11] Institute of Electrical and Electronics Engineers. *IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML*. Standard 1671-2010. 2011, pp. 1–388. DOI: 10.1109/IEEESTD.2011.5706290.

[IST10] International Software Testing Qualifications Board. *Standard glossary of terms used in Software Testing*. Version 2.1. 2010. URL: http://istqb.org/display/ISTQB/Downloads.

[ITU07] International Telecommunication Union. *Testing and Test Control Notation version 3: TTCN-3 core language*. Recommendation Z.161. 2007. URL: http://www.itu.int/rec/T-REC-Z.161.

[ITU11] International Telecommunication Union. *Message Sequence Chart (MSC)*. Recommendation Z.120. 2011. URL: http://www.itu.int/rec/T-REC-Z.120.

[JJ05] C. Jard and T. Jéron. "TGV:, theory, principles and algorithms". In: *International Journal on Software Tools for Technology Transfer (STTT)* 7.4 (2005), pp. 297–315. DOI: 10.1007/s10009-004-0153-x.

[Kan+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. CMU/SEI-90-TR-021. Carnegie-Mellon University Software Engineering Institute, 1990.

[KDD08] P. Koopman, K. Devale, and J. Devale. "Interface Robustness Testing: Experience and Lessons Learned from the Ballista Project". In: *Dependability Benchmarking for Computer Systems*. Ed. by K. Kanoun and L. Spainhower. John Wiley & Sons, Inc., 2008, pp. 201–226. DOI: 10.1002/9780470370506.ch11.

[Klo03] J. Klose. "Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior". PhD thesis. Carl von Ossietzky Universitat Oldenburg, 2003.

[Koc+98] B. Koch, J. Grabowski, D. Hogrefe, and M. Schmitt. "Autolink-a tool for automatic test generation from SDL, specifications". In: *Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE, Workshop on*. 1998, pp. 114–125. DOI: 10.1109/WIFT.1998.766305.

[KSH07] H. Kugler, M. J. Stern, and E. J. A. Hubbard. "Testing scenario-based models". In: *Proceedings of the 10th international conference on Fundamental approaches to software engineering*. (Braga, Portugal). FASE'07. 2007, pp. 306–320. DOI: 10.1007/978-3-540-71289-3_24.

[MFS90] B. P. Miller, L. Fredriksen, and B. So. "An empirical study of the reliability of UNIX utilities". In: *Commun. ACM* 33.12 (1990), pp. 32–44. DOI: 10.1145/96267.96279.

[MS04] G. J. Myers and C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[Ngu09] M. D. Nguyen. "Méthodologie de test de systèmes mobiles : Une approche basée sur les sénarios". PhD thesis. Université Paul Sabatier - Toulouse III, 2009.

[OMG05] Object Management Group. *UML Testing Profile v1.0 (U2TP)*. 2005. URL: http://www.omg.org/technology/documents/formal/test_profile.htm.

[OMG11a] Object Management Group. *Unified Modeling Language (UML) 2.4.1 Infrastructure Specification*. formal/2011-08-05. 2011.

[OMG11b] Object Management Group. *Unified Modeling Language (UML) 2.4.1 Superstructure Specification*. formal/2011-08-06. 2011.

[Opea] OpenAIS. *OpenAIS Standards Based Cluster Framework*. URL: http://www.openais.org.

[Opeb] OpenSAF. *OpenSAF Open Service Availability Framework*. URL: http://www.opensaf.org.

[PJ04] S. Pickin and J.-M. Jézéquel. "Using UML Sequence Diagrams as the Basis for a Formal Test Description Language". In: *Integrated Formal Methods*. Vol. 2999. LNCS. 2004, pp. 481–500. DOI: 10.1007/978-3-540-24756-2_26.

[R3C11] R3-COP. *Resilient Reasoning Robotic Co-operating Systems*. ARTEMIS research project nr. 100233. 2011. URL: http://www.r3-cop.eu/.

[SAF07] Service Availability Forum (SA Forum). *Application Interface Specification (AIS)*. 2007. URL: http://www.saforum.org.

[SE04] K. Saleh and C. El-Morr. "M-UML: an extension to UML for the modeling of mobile agent-based software systems". In: *Information and Software Technology* 46.4 (2004), pp. 219–227. DOI: 10.1016/j.infsof.2003.07.004.

[SS07] I. Solheim and K. Stølen. *Technology research explained*. Tech. rep. SINTEF A313. SINTEF ICT, 2007.

[Tam09] F. Tam. "Service Availability Standards for Carrier-Grade Platforms: Creation and Deployment in Mobile Networks". Tampere University of Technology, 2009. URL: http://URN.fi/URN:NBN:fi:tty-200904281053.