ÖNELLENŐRZÉS ÉS FUTÁSIDEJŰ VERIFIKÁCIÓ SZÁMÍTÓGÉPES PROGRAMOKBAN

OTKA T-046527

A KUTATÁS EREDMÉNYEI

ZÁRÓJELENTÉS

2004-2006.

Témavezető:

dr. Majzik István

Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

1. A kutatás célja, a munkatervben vállalt kutatási program

Az informatikai rendszerek hibajelenségeinek jelentős része a futtató hardver tranziens hibái és a szoftver bennmaradó tervezési hibái miatt következik be. A tranziens hibákat a programok tervezési időben történő verifikációja (helyességbizonyítása) nem képes megakadályozni, ráadásul ennek költséges és ezért nem teljes körű végrehajtása miatt maradhatnak tervezési hibák is a programokban. Ezért van létjogosultságuk a futás közben történő ellenőrzési technikáknak, amelyek a hibakezeléshez szükséges hibadetektálást végzik. A futtató rendszer szintjén történő, általános megvalósítás (pl. az operációs rendszerekbe épített védelmek) segítségével a hibás működés csak nagy vonalakban fogható meg (pl. illegális memóriaterülethez való hozzáférés). Az ilyen, mechanizmus szintű hibadetektáló eljárások hibafedését jól kiegészítik az alkalmazás-specifikus ellenőrzések. A korszerű tervezői rendszerekben ezeket - éppen úgy, mint a futatott kódot is - a szoftver modell alapján, formális alapokon célszerű megvalósítani.

A kutatás célja volt tehát olyan, futásidejű hibadetektáló eljárások kidolgozása, amelyek alkalmazás-specifikusak, de az ad-hoc jellegű megvalósítás helyett illeszkednek a (fél)formális modelleken alapú szoftver fejlesztéshez, és hatékony eszközökkel támogatottak. Ennek megfelelően a kutatás legjelentősebb részfeladatai a következők voltak:

- A jó (elvárt) és a hibás működés megkülönböztetésére alkalmas követelmények illetve referencia modell megadásához szükséges *specifikációs nyelv* kidolgozása. A specifikációs nyelv a futásidőben megfigyelhető viselkedést írja le, és a megvalósíthatóságot is figyelembe veszi, tehát a tervezés során használt rendszermodellen értelmezett, annak szemantikájához illeszkedik. A kutatás elsősorban a beágyazott és biztonságkritikus rendszerekben az eseményvezérelt programok viselkedésének megadására elterjedt UML állapottérkép modellekre koncentrált, azok műveleti szemantikáját alapul véve.
- Hatékony *ellenőrzési technikák* kidolgozása a specifikált követelmények teljesítésének ellenőrzéséhez. Fontos megjegyezni, hogy a tervezési időben használatos formális módszerek (modell ellenőrzés, ekvivalencia ellenőrzés, tételbizonyítás) algoritmusai erőforrásigényük és komplexitásuk miatt közvetlenül nem alkalmasak a futásidőben való végrehajtásra, tehát új módszerek kidolgozására van szükség.
- A modell alapú tervezési folyamatba érdemes bevonni a hibadetektálás tervezése mellett a *hibakezelés* (ami tipikusan kivételkezelés a modern programozási nyelvekben) tervezését és ellenőrzését is, így teljessé téve nagy megbízhatóságú illetve biztonságos rendszerek tervezésének támogatását.

2. A kutatás során kidolgozott új módszerek, eljárások

A kitűzött feladatok teljesítésében az alábbiakban felsorolt kutatási eredményeket értük el:

1. Specifikációs nyelv kidolgozása futásidőben ellenőrizhető követelmények megadására: Temporális logikákat korábban sikerrel alkalmaztak véges állapot-átmeneti rendszerek modelljének tervezési időben történő kimerítő ellenőrzésére. Ezt az eszközkészletet ültettük át a megvalósítások futási idejű verifikációjához: kifejlesztettünk egy *temporális logika alapú specifikációs nyelvet* (SC-LTL) UML állapottérkép megvalósításokhoz. Az SC-LTL nyelv alkalmas a rendszert érő események illetve a rendszer által tanúsított viselkedés (állapotok, akciók) sorrendiségével kapcsolatos követelmények leírására, így közvetlenül biztosítja a végrehajtási szekvenciákra vonatkozó előírások megfogalmazását is. Figyelembe veszi az UML állapottérkép szemantika által támogatott állapot- és eseményhierarchiát, valamint lehetővé teszi a hivatkozást minden lényeges modell elemre (kilépési és belépési eseményekre, akciókra is). A nyelv szemantikáját az állapottérkép esemény-feldolgozó lépésein (run-to-completion step) definiált Kripke tranzíciós rendszereken (KTS) fogalmaztuk meg. A nyelvi leírást és szemantikát 2005-ben véglegesítettük és publikáltuk az Architecting Dependable Systems III. kötetben található könyvfejezetben [1].
Kiemelendő, hogy a lineáris temporális logika definiálását az UML 2.0 állapottérképek egy általunk kidolgozott, a mérnöki gondolkodásmódhoz is közelálló *műveleti szemantikája* alapján végeztük, így biztosítva a modell szintjén a teljesen formalizált (matematikailag precíz) kezelést. A szemantikai definíció alapjait és a kapcsolódó logikát egy beadás előtt álló PhD disszertáció mellett a Software Engineering and Fault Tolerance című könyvben megjelenés alatt álló könyvfejezetünk [3] rögzíti (a formális szemantikára való utalásként a temporális logika nevét PSC-LTL-re módosítottuk).

2. Hatékony futásidőbeli ellenőrzés algoritmus kidolgozása az SC-LTL követelmények futásidőbeli ellenőrzésére: Az SC-LTL alapú követelmények futásidőbeli ellenőrzését a program forráskódjába illesztett ellenőrző kódrészletekkel (az angol terminológia szerint assertions) oldjuk meg. A korábban publikált általános célú következtetőrendszerekre alapozott értelmező elvű trace ellenőrzőknek nagy lenne az erőforrásigényük a közvetlen SC-LTL szemantikán alapuló naiv ellenőrzés esetén, mivel ugyanazt a logikai kifejezést a futás során újra és újra ellenőrizniük kell. Az erőforrásigény csökkentése érdekében erre egy *hatékony új algoritmust dolgoztunk ki*, amely automatikus kódgeneráláson és natív kódú ellenőrzésen alapul. Ez a hardverből ismert logikai kapuhálózat analógiájára épül, és háromértékű logika alkalmazásával valósítja meg a logikai rész-kifejezések összegyűjtését, *normalizálását* és

ellenőrzését. Az alkalmazott normál formát, a javasolt adatstruktúrákat, az ellenőrző algoritmust és a forráskódban való implementálás módját részleteiben a HASE'05 konferencia cikkünk [8] mutatja be. Összetett SC-LTL kifejezések esetén így több százszoros gyorsulás volt mérhető; összességében a temporális logikai kifejezés méretével növekvő gyorsulási tényezőről számolhattunk be. Ez lehetővé teszi a módszer alkalmazását kis erőforrás-felhasználású (olcsó mikrokontrollereket alkalmazó) beágyazott rendszerekben is.

3. Módszer kidolgozása viselkedés modell (részletes tervezési specifikáció) alapján történő futásidőbeli hibadetektálásra: Kidolgoztunk egy módszert annak ellenőrzésére, hogy egy eseményvezérelt rendszer futásidőbeli viselkedése *illeszkedik-e a tervezés során megalkotott UML állapottérkép specifikációhoz*. Így detektálni lehet, ha egy alkalmazás működése során olyan viselkedést mutat, amely megsérti a rendszer formális modelljét. A módszer lényeges tulajdonsága, hogy a viselkedést nem a forráskód szintű megvalósításhoz, hanem a magasabb szintű absztrakt modellhez hasonlítja. Ennek alapján lehetőséget ad a futásidőbeli tranziens hardver hibák mellett az *implementációs hibák* detektálására is. A módszert a WADS-2004 konferencia cikk ismerteti [5].
Előzetes vizsgálatokat végeztünk a megvalósított futásidőbeli ellenőrzés hatékonyságával kapcsolatban különféle *modell alapú megvalósítási mintákat* (forráskód szintézis mintákat) figyelembe véve. A vizsgálatok eredményei azt mutatják, hogy az így kialakított futásidőbeli ellenőrzés – a megvalósítási mintától függően – az effektív hibák akár 60%-át képes detektálni (a hardver által detektált hibák mellett). Az architektúrát a WADS-2004 konferenciacikk vonatkozó része [5], a mérések eredményeit az EUROMICRO-2004 konferenciacikk [4] ismerteti.

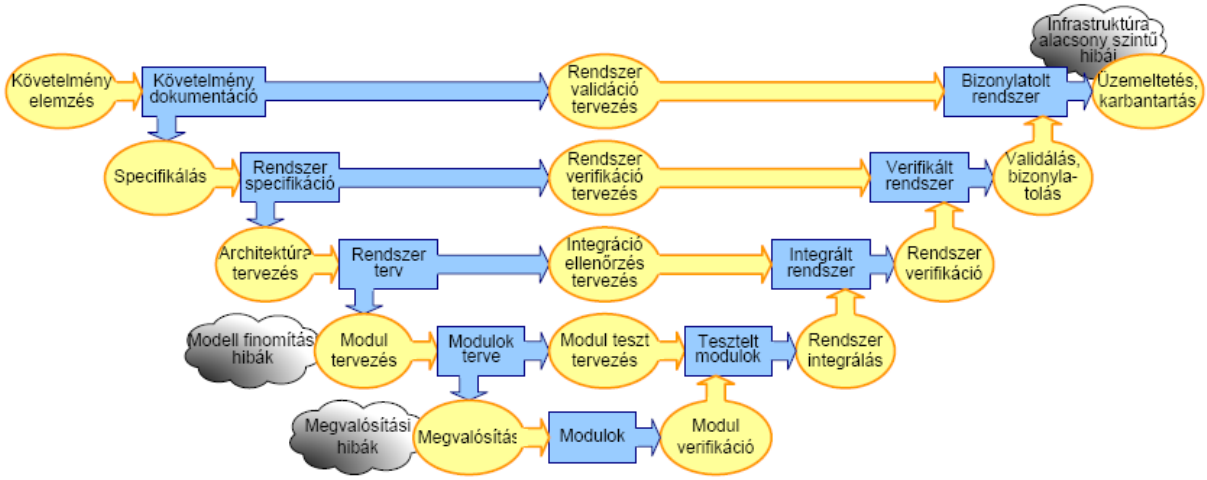
4. Dinamikus program analízis keretrendszer kidolgozása: A magas szintű specifikációs nyelven alapuló és a részletes viselkedési modellen alapú futásidőbeli ellenőrzések *egymást kiegészítve* járulhatnak hozzá a fejlesztési folyamat során bekerülő *tervezési és implementációs hibák illetve a működés közbeni hardver hibák* detektálásához. Az SC-LTL nyelv alkalmas a rendszert érő események illetve a rendszer által tanúsított viselkedés (akciók) sorrendiségével kapcsolatos követelmények leírására, így közvetlenül biztosítja a végrehajtási szekvenciákra vonatkozó előírások megfogalmazását. A részletes modell alapú ellenőrzések összevetik a specifikált és a tényleges viselkedést. A dinamikus program analízisnek így két szintjét dolgoztuk ki (ld. 1. ábra):

- A fejlesztés korai fázisaiban (a követelményanalízis után) a tervező a program biztonságos működéséhez tartozó követelményeket formalizálhatja az SC-LTL temporális logika segítségével. A formalizálás alapja a program még durva viselkedési modellje. Az SC-LTL-ben leírt követelményeket a programba illesztett, automatikusan generált

kódrészletek segítségével ellenőrizzük. Így a későbbi fejlesztési fázisokban előforduló tervezési illetve modell finomítási hibák következményei is kimutathatók.

- A fejlesztés előrehaladtával rendelkezésre álló részletes viselkedési modell alapján történik az állapot- és akciószekvenciák teljes ellenőrzése, a modellből automatikusan generált, futásidejű monitorozást biztosító úgynevezett watchdog kód segítségével. Ennek célja elsősorban a megvalósítási hibák és a működési hibák (a futtató hardver infrastruktúra alacsony szintű hibáinak) felderítése.

Így egy modellbázisú, sokoldalúan konfigurálható keretrendszer áll a fejlesztő rendelkezésére (ennek áttekintése az Architecting Dependable Systems III könyvfejezetben [1] található). A keretrendszerhez szükséges automatikus kódgenerátorok prototípusait elkészítettük, a prototípusok alapján elvégeztük a hibafedés kísérleti kiértékelését. Ezt a Software Engineering and Fault Tolerance című könyvben megjelenés alatt álló könyvfejezetünk [3] dokumentálja.



1. ábra. A futásidőbeli hibadetektálás által lefedett hibatípusok

5. Modellezési konvenció kidolgozása a futásidőbeli ellenőrzéshez kapcsolódó hibakezelés modellezéséhez és szintéziséhez: A futásidőbeli hibadetektálás olyan eszközt ad a szoftverfejlesztők kezébe, amelyre a hibakezelési mechanizmusok alapozhatók. A korábbi megközelítések esetén absztrakciós rést jelentett, hogy a hibadetektálási mechanizmusok a megvalósítás alacsony absztrakciós szintjéhez (forráskód utasítások) kötődnek, a hibakezelési mechanizmusokat viszont a tervezés korai fázisaitól figyelembe kell venni a szoftver strukturális illetve dinamikus modelljeiben. Ezen probléma kezelése érdekében kidolgoztuk, hogyan lehet a programozási nyelvek *kivétel-kezelési koncepcióit alkalmazni UML állapottérképekben*, biztosítva ezáltal, hogy a hibadetektálási mechanizmusok jelzéseit már az absztrakt modellben meg tudjuk jeleníteni. Ez a hibakezelés modell alapú automatikus szintézisét is lehetővé teszi. A modellezési módszert a FIDJI-2004 konferenciacikk publikálja [6]. A hiba utáni helyreállítás modell alapú tervezését mutatja be a hazai CSCS szimpózium előadás [7].

További kiterjesztésként kidolgoztuk a kivételkezelés *modell alapú verifikációját*. Ez a kivételkezeléssel kiterjesztett UML állapottérkép modell ellenőrzésével történik, a modell lezárása és hierarchikus automata formátumba való transzformációja révén. Ez utóbbi lépés lehetővé teszi a korábbi kutatásokban már kidolgozott modell ellenőrző használatát és nem igényli új eszköz implementálását. A FIDJI konferencián elhangzott előadásunk alapján meghívást kaptunk annak bővítésére, ez a változat a Scientific Engineering of Distributed Java Applications című könyv fejezeteként jelent meg [2].

6. Modell alapú tesztgenerálás: A program részletes viselkedés modellje a futásidőbeli ellenőrzés támogatása mellett lehetővé teszi az implementáció utáni *strukturális tesztelés modell alapú megvalósítását* is. A strukturális tesztelés célja, hogy a tesztekkel a program (a modellben reprezentált) minden végrehajtási ágának viselkedését megvizsgáljuk, reprezentatív bemeneti adatok segítségével. A kutatási téma kiterjesztéseként megvizsgáltuk az automatikus tesztgenerálás lehetőségeit UML állapottérkép modellek alapján. Kidolgoztuk a modell ellenőrző megfelelő paraméterezését a tesztgenerálási feladathoz, valamint mintakísérleteket végeztünk ipari alkalmazási példák valamint valós időzítést tartalmazó rendszerek esetére is. A kísérletek eredményét a DEPCOS-2006 konferencián [10] valamint a Fiatal Műszakiak X. Tudományos Ülésszakán [9] megjelent cikkek mutatják be.

3. Az eredmények felhasználásának, hasznosításának lehetőségei

Az elméleti eredmények ipari hasznosításának első lépései történtek meg a 2006-ban lezárult GVOP-3.1.1-2004-05-0523/3.0 sz. „Kötöttpályás közlekedési rendszerek konstruktív, EU-konform biztonsági minősítése” című kutatás-fejlesztési projektben. A projekt egyik részfeladatának célja volt olyan általános célú módszer kidolgozása és eszköztámogatásának megvalósítása, ami a *szoftver architektúra modelljére* épülve az *implementáció fázisáig* támogatja futásidőbeli hibadetektáló eljárások beépítését. Ennek keretében vasúti vezérlőrendszerekhez készülő szoftverekhez alkalmaztuk a futásidejű verifikációt (illeszkedve az MSZ EN 50128 fejlesztési szabvány „Defenzív programozás”, „Meghibásodás-bizonyító programozás” és „Megvalósított esetek tárolása” előírásaihoz), ehhez kidolgoztuk a *kódgenerálási keretrendszert*, valamint mintakísérleteket végeztünk. A kódgenerátor eszköz alkalmazásával az előírásokat közvetlenül megvalósító programkód áll a fejlesztő rendelkezésére: az ellenőrző blokkok az implementáció során pontosan átemelhetők vagy a generált kódváz felhasználása esetén közvetlenül rendelkezésre állnak. Ezzel a szabványban előírt módszerek és intézkedések egy része (ld. a fenti hivatkozásokat) automatikus, validált eszközökkel biztosítható.

4. Módosulás az eredeti kutatási tervhez képest

A kutatási tervet érintő *személyi változás* volt, hogy Kovács Péter Tamás doktorandusz hallgató a projekt első éve után kilépett a kutatóhelyről. Helyére, a projekt második évétől kezdve, Micskei Zoltán doktorandusz hallgató került. Nyári munkaként Nemes Csaba és Simon Gábor hallgatók segítettek be a megvalósításba és a kísérleti kiértékelésbe.
Tartalmi változás a következő volt: A kutatási tervben rögzítettük, hogy a futásidőbeli ellenőrzés elsődleges alkalmazási területe a beágyazott, biztonságkritikus rendszerek köre. Ugyanakkor felvetettük, hogy a hibás működés kockázata miatt egyre fontosabbá válik az e-business jellegű rendszerek futásidőbeli ellenőrzése is, elsősorban az üzleti logikát megvalósító szoftver rétegben (alkalmazás szerver). Ennek vizsgálatára mintakísérletek elvégzését terveztük. Ez azonban elmaradt, mivel az e-business jellegű workflow rendszerekben az UML helyett a BPEL (Business Process Execution Language) nyelv használata terjedt el. Ennek formalizálása és az SC-LTL specifikációs nyelv BPEL-hez való illesztése jelentős, be nem tervezett erőforrásokat igényelt volna (megemlíthető, hogy ez az illesztés más kutatási projekt keretében a kutatóhelyen még folyamatban van).
A *költségtervet érintő változás*, hogy az eredmények könyvfejezet formájában történő publikálása 2006-ban a tervezettnél kevesebb konferencia utazást igényelt. A könyvfejezetek terjedelme, rangja és publicitása azonban felülmúlja a konferencia kiadványokban megjelent előadásokét, így természetes döntés volt ezek előtérbe helyezése a konferencia előadásokkal szemben. Ugyancsak preferáltuk ezt a publikálási formát a folyóirat cikkek lassú (több éves) átfutási idejével szemben is.

5. A kutatás megvalósításához kapott egyéb támogatások

A kutatás megvalósításához részben hozzájárultak a következő források is:

- A témavezető „Hibatűrő rendszerek formális verifikációja és validációja” kutatási programjához kapott MTA Bolyai János Kutatói Ösztöndíjat. A kutatási program elsősorban a tervezés során végrehajtható ellenőrzésekre koncentrált.
- A hibadetektáló eljárások hatékonyságának kiértékeléséhez felhasználtuk azt az adatbányászaton alapuló eljárást, amit az „Intelligens mérésfeldolgozás alkalmazása megbízható IT szolgáltatások tervezéséhez” című, P-19/03 számú Portugál-Magyar Kétoldalú Kormányközi TéT Együttműködési Projekt keretében dolgoztunk ki (projekt partner a Coimbrai Egyetem volt). Az adatbányászati technika lehetővé tette a hibadetektálásban kulcsszerepet játszó tényezők (hibatípus, hibadetektálási technika) automatikus azonosítását.

Irodalomjegyzék

Könyvfejezetek:

1. Pintér G; Majzik I: Runtime Verification of Statechart Implementations. In: de Lemos R; Gacek C; Romanovsky A (eds.), Architecting Dependable Systems III, Berlin: Springer Verlag, 2005, pp 148-172, ISBN 3-540-28968-5
2. Pintér G; Majzik I: Modeling and Analysis of Exception Handling by Using UML Statecharts. In: Guelfi N; Reggio G; Romanovsky A (eds.), Scientific Engineering of Distributed Java Applications, Berlin: Springer Verlag, 2005, pp. 58-67, ISBN 3-540-25053-0
3. Pintér G; Majzik I: Error Detection in Control Flow of Event-Driven State Based Applications. In P. Pelliccione, H. Muccini, N. Guelfi, A. Romanovsky (eds.), Software Engineering and Fault Tolerance, World Scientific Publishing Co. Pte. Ltd, Series on Software Engineering and Knowledge Engineering, Accepted; to be published in 2007.

Konferencia kiadványokban megjelent referált publikációk:

4. Pintér G; Majzik I: Impact of Statechart Implementation Techniques on the Effectiveness of Fault Detection Mechanisms. In: 30th EUROMICRO Conference, Component Based Software Engineering Track, Rennes: 2004. pp. 136-143.
5. Pintér G; Majzik I: High-Level Supervision of Program Execution Based on Formal Specification. In: International Conference on Dependable Systems and Networks - Workshop on Architecting Dependable Systems (WADS), Florence: 2004. pp. 292-296.
6. Pintér G; Majzik I: Modeling and Analysis of Exception Handling by Using UML Statecharts. In: International Workshop on Scientific Engineering of Distributed Java Applications (FIDJI), Luxembourg-Kirchberg: 2004. pp. 69-78.
7. Pintér G: Abstract Model-Based Checkpoint and Recovery. In: The Fourth Conference of PhD Students in Computer Science (CSCS), Szeged: 2004.
8. Pintér G; Majzik I: Automatic Generation of Executable Assertions for Runtime Checking Temporal Requirements. In: 9th IEEE Int. Symposium on High Assurance Systems Engineering (HASE ’05), Heidelberg: 2005, pp. 111-120, IEEE CS, ISBN 0-7695-2377-3
9. Micskei Z: Automatikus tesztgenerálás modell ellenőrzővel. In: Fiatal Műszakiak X. Tudományos Ülésszaka, Kolozsvár: 2005, pp. 47-50, ISBN 973-8231-44-2
10. Micskei Z; Majzik I: Model-Based Automatic Test Generation for Event-Driven Embedded Systems Using Model Checkers. In: International Conference on Dependability of Computer Systems (DepCoS – RELCOMEX), Wroclaw, Poland: 2006, IEEE Computer Society, pp 191-198.