X. FIATAL MŰSZAKIAK TUDOMÁNYOS ÜLÉSSZAKA

Kolozsvár, 2005. március 18-19.

FUTÁSI IDEJŰ ELLENŐRZÉS ABSZTRAKT MODELLEK ALAPJÁN

Pintér Gergely

This paper outlines some achievements of our recent research related to run-time verification of UML statechart implementations. We present a verification framework offering a considerable degree of granularity by (i) enabling the specification of key dependability requirements in the early modeling phases by proposing a temporal logic language fitted to statecharts and (ii) by supporting the verification of the final implementation against the fully elaborated statechart model.

Cikkünkben a futási idejű ellenőrzéssel kapcsolatos kutatásaink néhány friss eredményét körvonalazzuk. Bemutatunk egy keretrendszert, amely jelentős rugalmasságot kínál a fejlesztőknek azáltal, hogy (i) lehetővé teszi megbízhatósági követelmények megfogalmazását már a tervezés korai fázisában egy általunk bevezetett temporális logikai nyelven, illetve (ii) támogatja a megvalósítás közvetlen ellenőrzését a teljesen kidolgozott UML állapottérkép modell alapján.

1. Bevezető

Futási idejű ellenőrzés során az a célunk, hogy felderítsük és jelezzük, ha egy szoftver megvalósítás (kész alkalmazás) a *futása közben* megsérti az absztrakt modellel (állapottérképpel, kommunikációs diagrammal stb.) megadott *specifikációját*. Ez a cikk röviden összefoglalja az ebben a témakörben végzett legújabb kutatásainkat és vázol egy *keretrendszert*, amely állapottérkép megvalósítások futási idejű ellenőrzésére szolgál. Megközelítésünk egyrészt támogatja a (i) megbízhatósági követelmények megfogalmazását a tervezés *korai fázisaiban*, mikor még csak *előzetes modellek* állnak rendelkezésünkre másrészt (ii) lehetővé teszi a megvalósítás alapjául szolgáló, tehát csak a fejlesztés *kései fázisaiban* előálló, *teljesen kidolgozott viselkedési modellek* alapján történő ellenőrzést:

- Egy szoftverrendszerrel kapcsolatban felmerülő kulcsfontosságú megbízhatósági követelményeket a *tervezés korai fázisában* szokás megfogalmazni, amikor még csak vázlatosan kidolgozott viselkedési modellek állnak rendelkezésre. Ezen kezdeti modelleket nem lehet közvetlenül referencia specifikációként használni, ezért bevezettünk egy *temporális logikai nyelvet*, amely lehetővé teszi, hogy a kulcsfontosságú ütemezéssel és sorrendiséggel kapcsolatos követelményeket már a korai modelleken megfogalmazhassuk. Megközelítésünk matematikai háttérét a második fejezet vázolja, a nyelvet a harmadik fejezetben mutatjuk be.

- Az egymás után következő modellfinomítási lépések során a fejlesztők elkészítik a rendszer teljesen kidolgozott viselkedési modelljét, amelyet a programozási fázisban végül megvalósítanak. A második fejezetben bemutatunk egy leírásmódot UML állapotterképek formális reprezentációjára, majd erre építve a negyedik fejezetben vázoljuk, hogyan használható egy ilyen modell a megvalósítás közvetlen futási idejű vizsgálatok referenciájaként.

2. Formális modellek

Mára az UML (Unified Modeling Language [1]) vált a szoftver modellezés általánosan elfogadott egységes leíró nyelvén. Gazdag grafikus eszközkészletéből az eseményvezérelt viselkedés modellezésére az *állapotterképek* szolgálnak (egy közlekedési lámpa állapotterképe látható az 1. ábra bal oldalán). Egy állapotterkép *állapotai* olyan szituációkat modelleznek, amikor valamilyen állandó körülmény áll fenn (pl. a közlekedési lámpa zöldet mutat). Az állapotokhoz *belépési* és *kilépési akciót* rendelhetünk, amelyeket az állapot aktívra illetve inaktívvá válásakor kell végrehajtani (pl. fel illetve le kell kapcsolni egy izzót). Az állapotokat irányított *átmenetek* kötik össze. Az átmenetekhez megadhatjuk, hogy milyen típusú *kiváltó esemény* hatására mennek végbe (pl. letelik egy időzítés), illetve közvetlenül az átmenetekhez is rendelhetünk *akciókat*. Az állapotok finomításával állapot hierarchiakat hozhatunk létre (pl. az, hogy a közlekedési lámpa zöldet mutat, az speciális esete annak, hogy be van kapcsolva, vagyis a zöld állapot a bekapcsolt állapot finomítása).

1. Ábra. Egy közlekedési lámpa állapotterképe (balra) és a neki megfelelő EHA (jobbra)

Mivel az UML szabvány [1] alapvetően szöveges formában adja meg az állapotterképek működési szemantikáját, célszerű áttérni egy kevésbé vizuális de formális szemantikával definiált ábrázolásmódra. Keretrendszerünkben az UML állapotterképeket automatikusan *kiterjesztett hierarchikus automatákra* (Extended Hierarchical Automaton, EHA [2]) transzformáljuk. Egy EHA egyszerű *állapotokból* és az őket tartalmazó *szekvenciális automatákból* áll. Az állapotokat tetszőleges számú párhuzamosan futó automatára finomíthatjuk (így adható vissza az UML állapotterképek állapot finomítás koncepciója). Az 1. ábra jobb oldala a közlekedési lámpa állapotterképekből képzett EHA-t mutatja.

A keretrendszerünkben használt harmadik leírásmód a *Kripke-struktúrák* eszközkészlete. Ezzel a matematikai formalizmussal specifikálták a kiterjesztett hierarchikus automaták szemantikáját [2] és ez az alapja a következő fejezetben bemutatott temporális logikai ellenőrzésnek is. Egy adott A atomi állításhalmaz mellett egy Kripke-struktúra matematikai szempontból egy $K=(S, T, L)$ hármas, ahol S

az állapotok halmaza, $T \subseteq S \times S$ az állapotátmenet reláció, $L: S \rightarrow A$ pedig a címkézési függvény, amely bizonyos elemi állításokat (címkéket) rendel a Kripke-struktúra állapotaihoz. Egy $\pi = (s_0, s_1, \dots, s_{n-1})$ állapotszekvenciát, melynek lépéseit a T reláció köti össze ($\forall_{0 \leq i < n-1}: (s_i, s_{i+1}) \in T$) a Kripke-struktúra egy *útvonalának* nevezzük. [4]-ban bemutatunk egy módszert arra, hogyan képezhetők le állapottérképek közvetlenül Kripke-struktúrákra. A megközelítés alapgondolata az, hogy összegyűjtjük az állapottérkép összes lehetséges lépését feljegyezve azt, hogy (i) milyen esemény váltotta ki ezt a lépést, (ii) mely konfigurációból indult, (iii) mely konfigurációba érkezett, illetve, hogy milyen (iv) kilépési, (v) átmenthez rendelt és milyen (vi) állapotba belépési akciót kell ennek során végrehajtani. Ezen információ birtokában felépíthetünk úgy egy Kripke-struktúrát, hogy az állapottérkép lépéseinek megfeleltetjük a Kripke-struktúra állapotait és az (i)-(vi) információkat a címkézési függvény segítségével rendeljük hozzá a lépéseket reprezentáló állapotokhoz. Végül a T átmenet relációt úgy adjuk meg, hogy az tükrözze, hogyan következhetnek egymás után az állapotoknak megfelelő lépések az állapottérképben: azt mondjuk, hogy $(s_i, s_{i+1}) \in T$ ha az s_i által reprezentált lépés célkonfigurációja megegyezik az s_{i+1} -nek megfelelő lépés kiindulási konfigurációjával.

3. Sorrendi követelmények ellenőrzése

Események megengedett egymásutániságát, akciók helyes ütemezését, kommunikációs protokollok működését, vagyis számos *sorrendiséggel kapcsolatos követelményt* hatékonyan megfogalmazhatunk *temporális logikában*. Noha számos temporális logika alapú *modell ellenőrzési* módszert publikáltak már állapottérképekhez, ismereteink szerint mindössze egyetlen kereskedelmi termék (a Temporal Rover a Time Rover cégtől) támogatja annak ellenőrzését, hogy egy állapottérkép *megvalósítás* betartja-e a temporális logikában megfogalmazott követelményeket *futása során*. [4]-ban az volt a célunk, hogy definiáljuk egy temporális logika nyelvtant és szemantikát, amelyet kifejezetten állapottérképekhez illesztünk. A nyelv alapja a fent bevezetett leképezés az állapottérképek és a Kripke-struktúrák között. A nyelv atomi állításai arra vonatkozhatnak, hogy (i) az állapottérkép egy állapota aktív-e, (ii) a legutóbbi lépést mely esemény hatására ment végbe, illetve (iii) milyen akciókat hajtott végre a legutóbbi lépés során. A nyelv összetett kifejezései az atomi állításokból a *Boole logika* két alapvető operátora, a tagadás ($\neg$) és az ÉS művelet ($e_1 \wedge e_2$), illetve a „következő állapotban igaz lesz” (Xe) és az „ e_1 igaz mindaddig, amíg e_2 igaz nem lesz” ($e_1 U e_2$) *temporális operátorok* segítségével képezhetők. Az így összeállított kifejezések az állapottérképet reprezentáló Kripke-struktúra útvonalain értékelhetők ki, pl. egy állapotra vonatkozó atomi állítás akkor igaz, ha az állapot szerepel az útvonal első eleme által reprezentált lépés célkonfigurációjában. A Boole operátorok használata kézenfekvő; az X temporális operátor azt mondja, hogy az argumentumát nem a vizsgált útvonal első állapotán, hanem a következőn kell kiértékelni, az U operátor használata a fenti megfogalmazásból következik: az útvonalon valamikor e_2 -nek igazzá kell válnia, mindaddig e_1 -nek kell teljesülnie minden lépésre. A két itt említett operátoron túl szokás bevezetni pl. a logikai implikációt ($\rightarrow$), a VAGY műveletet ($\vee$) stb., illetve pl. a G temporális operátort, amely azt írja elő, hogy az argumentumának minden lépésben

teljesülnie kell. Ezekkel az eszközökkel azt a követelményt, hogy a példaként használt közlekedési lámpánkat kizárólag a kapcsoló megnyomása kapcsolhatja ki így fogalmazhatjuk meg: $G(s_{Be} \rightarrow X(s_{Be} \vee (s_{Ki} \wedge e_{Kapsolo})))$ vagyis minden lépésben igaz az, hogy ha a lámpa be van kapcsolva, akkor a következő lépés után is be lesz, vagy ha mégis ki lenne kapcsolva, akkor ez a változás a kapcsoló megnyomása miatt történt.

4. Viselkedési követelmények ellenőrzése

A megvalósítási fázisban már rendelkezésre áll a tervezett alkalmazás teljesen kidolgozott állapotterképe. Ezt a specifikációt keretrendszerünkben közvetlenül használjuk a megvalósítás futási idejű ellenőrzésére: az állapotterképet kiterjesztett hierarchikus automatává (EHA) transzformáljuk, majd a vizsgált alkalmazást úgy módosítjuk, hogy az folyamatosan tájékoztasson működéséről egy viselkedési monitor komponensre. A monitor referencia információja természetesen az EHA, ehhez hasonlítja a megvalósítás működését. Ezzel a komponenssel ellenőrizhetjük, hogy az alkalmazás végrehajt-e minden akciót, amit egy-egy lépése során el kell végeznie, illetve azt, hogy az események hatására az állapotterben megtett útja megfelel-e a specifikációnak.

5. Összegzés

Ebben a cikkben vázoltunk egy futási idejű ellenőrző rendszert, amely temporális logikában megfogalmazott, illetve kidolgozott állapotterképpel specifikált viselkedés ellenőrzését teszi lehetővé.

Hivatkozásjegyzék

- [1] Object Management Group (OMG): Unified Modeling Language (UML) Version 1.4. 2001
- [2] Latella, D., I. Majzik, and M. Massink: Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. Formal Aspects of Computing, vol. 11. Springer, pp. 637-664.
- [3] Gergely Pintér and István Majzik: *High-level Supervision of Program Execution Based on Formal Specification*. In Proc. Int. Conf. on Dependable Systems and Networks (DSN), Workshop on Architecting Dependable Systems. June 28-July 1, 2004. Florence, Italy. Supplemental Volume pp. 292-296.
- [4] Gergely Pintér and István Majzik: Run-time Verification of Statechart Implementations. In Third Book on Architecting Dependable Systems. (Bírálati szakaszban levő könyvfejezet).

Pintér Gergely doktorandusz hallgató

Budapesti Műszaki és Gazdaságtudományi Egyetem, H-1117 Budapest, Magyar tudósok körútja 2.

Tel.: +36-1-4633579, Fax: +36-1-4632667, E-mail: pinterg@mit.bme.hu