SDL:
A kommunikációs folyamatmodellek egyik szabványosított implementációs eszköze

MEDVE ANNA

Veszprémi Egyetem, Műszaki Informatikai Kar, Információs Rendszerek Tanszék
medve@almos.vein.hu

Kulcsszavak: SDL, UML2.0, FMT, formális nyelvek, ITU-T szabványok, ITU- nyelvek

Célunk a Specification and Description Language (SDL) formális módszertan modellező képességeinek ismertetése. A cikk motivációjának alapja, hogy az Unified Modeling Language (UML) modellező nyelvnek a '99 óta tartó fokozott közelítése a távközlés világában kifejlesztett SDL nyelvhez oly módon jelenik meg a fejlesztők mindennapjaiban, hogy az UML2.0 szabvány magába emelte a dinamikus viselkedések leírására az SDL nyelv teljes implementációs eszköztárát. Jóllehet a fejlesztőkörnyezetek gyártói közreadják az egyes terület-specifikus UML2.0 nyelvi implementációkba beépített lehetőségek segédletét, az előzőekkel szakító szemléletű UML2.0 verzió megismerésében jól hasznosítható az SDL viselkedés-modellező jellemzése.

1. Bevezetés

A távközlés digitalizálása és a számítástechnikával való konvergenciája, valamint az Internet hatásaként létrejött újszerű üzleti modellek visszahatnak a távközlési rendszerek fejlesztésére: az emelt szintű szolgáltatások (value added services) egyre dinamikusabbak és egyre gyorsabb fejlesztési folyamatot igényelnek, fokozottabb jósággal.

A kommunikációs szolgáltatások sajátossága napjainkban, hogy azok különböző infokommunikációs technológiákon alapulnak: a hagyományos nyilvános kapcsolt és mobil távközlő technológiák, vagy az Internet technológiák, vagy ezek keverékei által. Ugyanakkor nem mindig lehetséges élesen elhatárolni a kommunikációs szolgáltatásokat a kommunikáló protokolloktól, és miközben e téren a fejlesztés több évtizedes tapasztalatai kifejezetten a protokollfejlesztésben jegyzettek, aközben napjainkban az üzleti folyamatok szabályainak tervezésében és az információs rendszerek strukturálásában is a kommunikációs modell kerül előtérbe. Magyarázata lehet a fejlődés törvényszerűsége, oka a szoftverkrízis, amely kihat a globális piacon lévő termékek együttműködésére és átmeneti minőségcsökkenéssel jár. A megoldást várhatóan a szabványosítás és a szoftverfejlesztés kínálja [3,4]. A kommunikáló rendszerek működtetésének eszközei a *kommunikáló protokollok és kommunikációs szolgáltatások* [1,2], melyek modellezésére az International Telecomunication Union – Telecommunications Standardization Sector (ITU-T) a Z. formális nyelvek szabványcsaládját fejlesztette ki.

Annak ellenére, hogy a formális módszereket az iparban még kevésbé használják, a kutatók a jövő elkerülhetetlen szoftverfejlesztő eszközeinek tartják. A tapasztalat szerint az ipar olyan fejlesztési módszertant fogad el, amely a szükségletei szerint alkalmazható, kereskedelemben forgalmazott, fejlesztői környezettel jól támogatott, valamint fontos szempont annak szabványosítása is [9]. Az *UML (Unified Modeling Language)* félformális modellező nyelv az *Object Management Group (OMG)* gondozásában ezt az igényt táplálja és kínál megoldásokat egyben.

A távközlési korlátokat lebontó informatikai megoldások szoftvertechnológiája oly módon hatott az UML fejlődésére, hogy a konkrét fejlesztőkörnyezetű UML profilok terület-specifikus fogalomkategóriáira a fejlesztőkörnyezet-forgalmazók tudásbázis-alapú fejlesztéstámogató módszereket építettek be a távközlés területén bevált eszköztárból, kezdetben az integrációs tesztek támogatására. A felhasználóközpontú értéknövelt szolgáltatásokat támogató gyors szoftverfejlesztés technológiáiként előtérbe kerültek a távközlés területén korábban kifejlesztett komponens-alapú és modell-vezérelt modellező technológiák, az automatikus kódgenerálással együtt. A hazai könyvkiadásban égető hiány pótlásaként jelent meg az UML 2 modellező nyelvi kézikönyv [32], amely a szerzőtől már megszokott módon tárgyalja és segíti hozzá az UML-el ismerkedőket és UML-t már alkalmazókat is a nyelv lényegének megértéséhez és hatékony alkalmazásához.

Mivel az OMG kifejezetten a szoftverfejlesztés folyamatára optimalizálja az objektumtechnológiák fejlesztését, az UML fejlődésében és alkalmazás-támogató technológiáiban egyre hangsúlyosabbak és látványosak a távközlésben kidolgozott gyakorlatban jól bevált módszertanok. A cikkben ezek egyikének jellemzését, a dinamikus viselkedést leíró *Specification and Description Language (SDL)* formális nyelv tulajdonságait ismertetjük, mint a távközlés világában kifejlesztett és automatikus kódgenerálásra alkalmas szabványosított implementációs formális módszertant.

2. Az SDL specifikáló és leíró nyelv

Az SDL nyelvet az ITU-T a Z.100 -as ajánlásaiban dolgozta ki komplex rendszerek specifikálására, oly módon, hogy az eseményvezérelt, valósidejű és interaktív

folyamatok konkurens módon diszkrét jelekkel kommunikálnak [10-13].

Jóllehet manapság az SDL nyelv használható bármely valós idejű rendszer specifikálására és implementálására, gyökerei a távközlésben vannak.

Az SDL fejlesztése egy 1968-as ITU tanulmányból indult ki a programvezérlésű kapcsolórendszerek kezelésére vonatkozóan, aminek eredményeként 1972-ben egyetértés született arról, hogy nyelvek szükségesek a gépek és berendezések interakcióinak leírására és programozására. 1976-ban közzétették a leíró nyelv szabványát, egy alap grafikus leírónyelv fejlesztésével, majd négyévenként további fejlesztéseket jelentettek meg.

1980-ban a processz szemantika meghatározásával, 1984-ben struktúrák és adatok hozzáadásával 1988-ban konkrét eszközként jelentették meg az SDL-88-at, egy jól meghatározott szintaktikájú Meta IV formális jelöléssel támogatva a beszélt nyelvi leírást. Az 1992-ben bevezetett típuskonstrukció az SDL-t objektum-orientáltá tette és az 1996-ban az Addendum szabványkiegészítés pedig a hatékonyabb fejlesztőeszközöket eredményezte.

Az SDL eszközök piaca 1996-2000 között jelentősen növekedett. A felhasználók nyomására az *SDL formális* jellegét fokozták, ami a tervezésben a nagyobb pontosságot, megfelelést, és átláthatóságot biztosítja. A számítógépes grafika és a beépített tudástárak fejlődésével lehetővé vált mára az SDL eszközök nagyfokú funkcionalitása és a bizonyítottan jó, eredményes nyomkövetése. Az ipari alkalmazásának elterjedését legfőképpen a megvalósításnak a specifikációból való generálhatósága fokozta: a fejlesztőkörnyezetek generálnak programozási nyelvekre forráskódokat (általában C/C++), amelyeket be lehet szerkeszteni a valósidejű rendszerek termékgyártásába. A generált C++ kód kezelése közbülső nyelvű elemként történik, mint ahogy a kompájlerek kezelik a gépi kódot. Erőssége az eszközöknek, hogy absztrakt módon is lehet használni az SDL nyelvet a megvalósításhoz szükséges követelmények informális leírására [5,6,29].

Az SDL nyelvet 1996 óta használják a távközlési iparon kívül is, elsősorban az orvosi felszerelések iparágában, az autó- és repülőgépiparban.

A hazai kutatások az SDL alkalmasságát célozzák meg a hardver-szoftver együttes tervezésére, és jelentős eredmények születtek a kódbázisú (újabban specifikáció-bázisú) tesztelés és validálás terén az automatikus tesztsor generálásra és szelektálásra adott számos algoritmussal [7,8], a konformancia teszteléssel összefüggésben, és a formális módszertani összefüggésben [1,9].

2.1. Az SDL alkalmazásának jellemzői

Az SDL elsősorban a reaktív és diszkrét rendszerek esetében alkalmazható:

– *specifikálásra:* rendszer szolgáltatásainak specifikálása, a rendszer belső szerkezete nélkül, főleg az interfészek és a kommunikáció jellemzése fontos,

– *tervezésre:* hardver- és szoftverfejlesztők által (bővítményes folyamat, a rendszerspecifikáció transzformálásával, valamint a belső szerkezetre és működésre vonatkozó információkkal)

– *megvalósításra:* rendszerintegráló mérnökök által, cél: a végső tervezésből futtatható rendszert készíteni (automatikus: a tervezéshez futás-idejű információkat kell hozzáadni),

– *dokumentálásra:* felhasználók és rendszerfenntartók számára (az aktuális működésről).

Az SDL alkalmazását megkönnyíti az egymásnak kölcsönösen megfeleltethető grafikus és szöveges nyelvi implementációja, a jól meghatározott fogalomköre, a tiszta, egyértelmű és pontos specifikálás következetessége. Az SDL egyik legfontosabb jellemzője a formálissága, amelyre alapozható a megvalósítás konformanciája és az automatikus kódgenerálás. Az SDL nyelv hierarchiája és formálissága kínálja a spirális fejlesztés menetét, ezzel áthidalja a hézagot a specifikálás és megvalósítás között, biztosítva az absztrakt szintű modellezést a megvalósításhoz szükséges részletezettséggel.

Az SDL specifikációk jóságához szükséges az *MSC (Message Sequence Charts)* nyelv használata, amely kezdetben az SDL nyelv alkalmazását megadó szabvány része volt, 1992-ben választották külön szabványba, amit az 1996-os négyévenkénti soronkövetkező fejlesztés az in-line kifejezések és adattípus definíciók hozzáadásával a követelménytervezésben sokoldalúan alkalmazhatóvá tett [14,17,18].

Az MSC nyelv szerkezeteivel az eseményekre forgatókönyvet szerkeszthetünk, amelyek sorozatával informálisan leírható a megfigyelt viselkedés üzenetcserék sorozataként. A megfigyelt viselkedések egy halmaza valahány rendszerfolyamatot ábrázol, valamint általuk megjeleníthetők a rendszerentitások közötti kapcsolatok (interakciók), ezáltal a forgatókönyvek sorozatai segítik a rendszerentitások hierarchiába szervezését, valamint a dinamikus viselkedés formális analízisét, ezáltal a két nyelv komplett megoldást szolgáltat a specifikáláshoz és megvalósításhoz.

Az adatok ábrázolására SDL-ben az *ASN.1 (Abstract Syntax Notation One)* adatleíró nyelvet érdemes alkalmazni [15]. Ennek nagy előnye a nyílt rendszerek ábrázolásakor érvényesítődik az együttműködőképesség fokozásával, a zárt kommunikációs rendszerekben az SDL beépített típusgeneráló eszközeivel érdemes élni, főleg ha gyakori a rendszerváltozatok továbbfejlesztése, mivel az absztrakt típusok művelet-definíciós algebrai eszköztára jól hasznosítható az algoritmikus leírások és a kommunikációs folyamatelemek elkülönítésére az implementáció során, ezáltal a fejlesztői változtatások egyértelműsítésére is.

2.2. Az SDL szabványai

Az SDL nemzetközileg szabványosított nyelv. A nyelv fejlesztése az IUT-T ajánlások Z sorozatában közreadott szabványokban követhető, revíziójuk folyamatos. A fő szabvány a Z.100, amely tartalmazza az SDL de-

finícióját precíz és tömör módon. Nyelvi referencia, annak ellenére, hogy nem javasolt a nyelv tanulására. Nem tartalmaz példákat, de jól mutatja a szintaktikai szabályokat [19].

A Z.100-as szabványban a konkrét szintaktikai szabályok BNF dialektusban (MetaIV absztrakt nyelvtanban) definiáltak. Mindegyik szöveges nyelvtani elemnek (SDL/PR) adott a grafikus megfelelője (SDL/GR), formális és absztrakt szintaxisként, amelyhez kapcsolódik egy végrehajtásmodell. A formális szemantikát az F melléklet tartalmazza, amely egyenértékű a Z.100-as szabványban természetes nyelven megadott „Szemantikus leírás és Modellek" fejezettel. További szabványok terjesztik ki a Z.100-as szabványt a nyelv továbbfejlesztéseivel.

A Z.105-ös szabvány megadja az ASN.1 modulok kapcsolását az SDL adatleírásokba direkt módon, azaz a Z.105 megadja az ASN.1 használatát SDL-ben.

A Z.106-os szabványt 1996-ban az egyes fejlesztői környezetek közötti átjárhatóságra kiemelt, SDL/PR CIF néven találjuk meg a hétköznapokban.

A Z.107-es szabvány kiegészíti a Z.100 és Z.105-ös szabványokat, az ASN.1 adattípusok használhatók az SDL adatdefiníciókban, azaz külön szabványban rögzítik az ASN.1 beágyazását az SDL nyelvbe.

A Z.109-es szabvány definiálja az SDL-nek megfelelő UML profilt, így hozzáadja az UML elv szerinti ábrázolást. (A Z.120 hasonló a Z.100-as szabványhoz az MSC definiálására.

A nyelv jellemzése

A nyelv fő szerkezeteit az objektum-orientált *absztrakt típusok* jelentik, oly módon, hogy maga az SDL típusok definíciója tartalmazza a szoftverfejlesztés mérnöki megközelítését [20].

Az *OO-elv az SDL-ben* a típusdeklarációk által érvényesül úgy az öröklődés, mint a specializációk vonatkozásában.

Az öröklődéshez a típusdeklarációk elhelyezhetők a rendszerstruktúra bármely szintjén, ha a fastruktúrában érvényesíttetjük az öröklődést. Abban az esetben, ha más rendszerekkel is megosztjuk a deklarációkat, akkor *package* egységekben a rendszeren kívül helyezzük el a deklarációkat. A specializációk szerkesztéséhez altípusokat szerkeszthetünk a főtípushoz adott új tulajdonságokkal (például újabb állapotátmenetek a `process` típushoz, újabb processzek a `block` típushoz), vagy a főtípusban megadott virtuális típus és virtuális állapotátmenet újradefiniálásával.

Az ábrázolandó rendszer szerkezetének és viselkedésének együttes leírására az SDL szerkezeti (strukturális) típusai adják a nyelvi elemeket. A *szerkezetet* hierarchikusan a *rendszer, blokk, processz* és *eljárás* egységek ilyen sorrendű egymásba ágyazása jelenti a `system type`, `block type`, `process type`, `procedure type` szerkezeti típusok megadásával, amint az *1. ábrán* látható.

Az ábrázolandó rendszer felosztásának célja az *átláthatóság*, a részletek alsóbb szinteken taglalhatóságával; a *funkcióorientáltság*, a természetes funkciók követhetőségével; a *modellezhetőség*, az interfészek és kapcsolatok ábrázolhatóságával; az *újratervezhetőség*, az alrendszerek megvalósításainak különböző hardver és szoftver környezetekbe ágyazhatóságával; az *újrafelhasználhatóság*, a már létező modulok felhasználásával; a *tesztelhetőség*, a szoftverelemek belső kommunikációjának statikus ábrázolásával; a *karbantarthatóság*, a „fentről-le" felbontással valamint az interfészek és kapcsolatok ábrázolhatóságával, amely egyben a fejlesztőcsoport tagjainak együttműködését biztosítja, a fejlesztés alatti karbantarthatóság mai irányzatainak megfelelően; az *üzembiztonság*, a szerkezeti típusok és a többi típusok közötti – főleg a kommunikáció-, kapcsolat- és viselkedéstípusok közötti – formális szemantika támogatásával. SDL nyelven folyamatorientált, időben diszkrét rendszermodellezést végezhetünk [21].

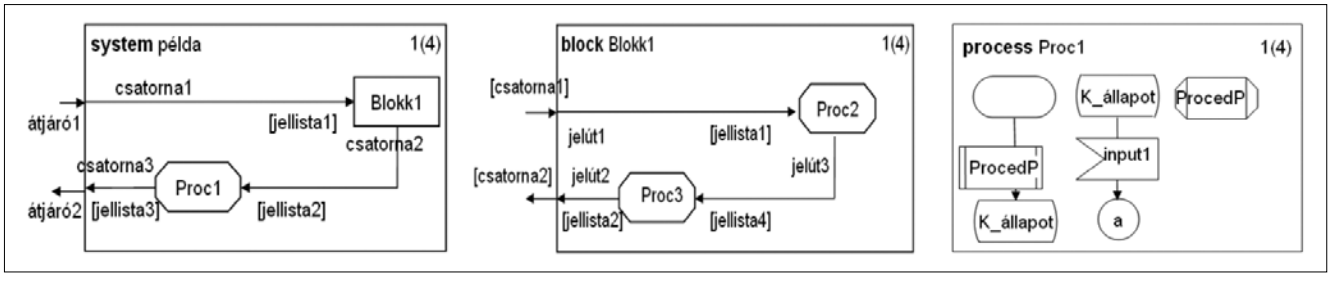
A stabil rendszerarchitektúra megvalósításához az SDL támogatja a távoli eljáráshívást, a rekurzivitást, valamint a típusok lokális és globális hatókörét az egyes állapotgépek külön-külön memóriaterületével is.

Az SDL-ben az elemi működési funkciók láncolatai a kommunikáló automaták állapot-átmenetei, a részfunkciók közötti kölcsönhatások pedig a funkció szerint hierarchiába szervezett automaták kapcsolatai, rendszer-, blokk- és processz szinten. A kommunikáló automatákat a `process` szerkezeti és viselkedési típussal ábrázoljuk. A `process` szerkezeti típus az automatáknak a rendszer egészéhez való kapcsolódásainak jellemzőit rögzíti, míg a *process* viselkedési típus, mint látni fogjuk, az elemi funkciókat ellátó automaták működését ábrázolja.

A rendszer dinamikus *viselkedését* a processzek kommunikációja határozza meg, a *processzek kommunikációját* az üzenetküldések és üzenetfogadások adott módú leírásával adjuk meg.

A jelcserét az üzenetküldések és üzenetfogadások valósítják meg a processz (az automata) erre alkalmas

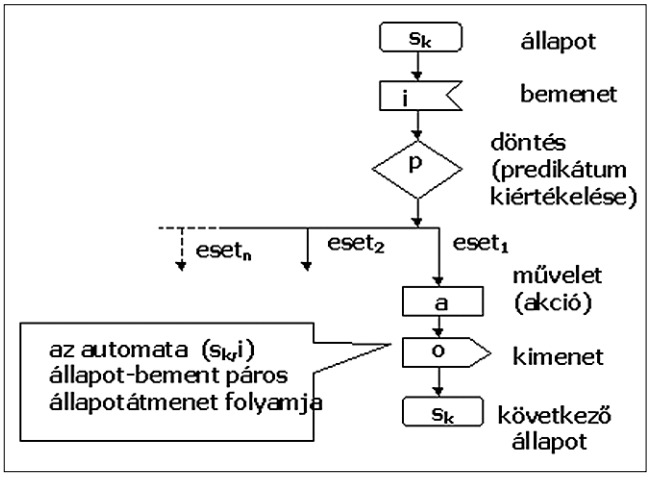
1. ábra Az SDL specifikáció hierarchiája



állapotaiban, ezáltal a *dinamikus jelleg leképezhető az állapotok változásával.*

A kommunikáló automatáknál az állapotok változása gerjesztés-válasz elven történik, és állapotátmenet a neve. Az *állapothoz (state)* kapcsolhatók az adott állapotban fogadott *bemenőjelek (input)* és az adott állapotban a bemenőjelhez társított állapotátmenet leírása. Az *állapotátmenet (transition)* megadja egy `{állapot, bemenôjel}` párosához tartozó műveletsorozatok halmazát és az esetleges kimenőjelet valamint az új állapotot. Az állapotátmenet megadása többféle szerkezetben lehetséges, ez adja egyben a nyelv erejét is. A műveletsorozat neve *akció (action)* és *feltétel (decision)* alapján is megválasztható. Az állapotátmenetben az akciót követheti üzenetküldés is és maradhat az automata ugyanabban az állapotában is *(2. ábra)*:

2. ábra Állapotátmenet



A **kommunikáció eszközeinek leírására** szolgál a *csatorna és jelút,* az *aszinkron jelek* és a *távoli szinkron eljáráshívás* típusai (1. ábra).

A FIFO elven működő csatornákon és jelutakon, a kommunikációs mechanizmus egy vagy kétirányú lehet, és a hordozott paraméterekkel információcsere és szinkronizáció valósul meg az SDL processzek között, valamint az SDL rendszer és környezete között (nem SDL-szerűen viselkedő alkalmazás, vagy másik SDL rendszer).

Az SDL a jelútak és csatornák kombinációjának szerkezetével jólmeghatározott interfészeket definiál a blokkok és processzek között. A kommunikációs típusokhoz nem rendelhető prioritás, ezt a megvalósítás szintjén a fejlesztő teheti meg.

Az SDL gyengesége, hogy a kommunikációs csatornák és jelútak állapotának kezelésére nem ad definíciós eszközt, a megvalósítás szintjén pedig a bejövő jelek feldolgozásánál megengedett jelsorrend csere a SAVE mechanizmussal jelentősen rontja a tesztelhetőséget [22].

Egy jel egyszerre csak egy meghatározott processzpéldánynak küldhető, a többesküldésre nincs szintaxis, a többesküldést mint funkciót a fejlesztőnek kell megszerkesztenie.

Az **adatok leírására az SDL nyelvben** az *absztrakt adattípus definiciót (ADT) és/vagy az ASN.1 adatleíró nyelvet* kell használni, azaz az SDL nem tartalmaz adattípusokat, hanem eszközöket ad a fejlesztőnek bármely adatszerkezet megszerkesztésére.

Az *absztrakt adattípusok (ADT)* olyan típusok, amelyeknek nincs adatszerkezetük, hanem az adat tulajdonságainak a leírására való szerkezetekkel és a fejlesztő kényelmére előredefiniált, a programozási nyelvekből ismert egyszerű és összetett alap-típuskészlettel bármely kommunikált adat leírható. Rendkívül előnyös, mert a fejlesztő specifikálhatja ily módon, az adattípus jelkészletét, műveleteit, függvényeit is, bármely területspecifikus jellemvonás ábrázolható, mi több, az algoritmikus, számításos műveletek típusműveletekbe szerkesztve lerövidülnek, jól áttekinthetővé válnak az állapotátmenet leírások.

A másik lehetőség az adattípusok definiálására, az *ASN.1* absztrakt jelölő nyelv, amely a magas szintű nyelvek adatdefiníciójához ad átjárást és lehetőséget a létező adatszerkezetek újrafelhasználásához. SDL verziótól és fejlesztőkörnyezettől függ, hogy deklarációszinten vagy importálással alkalmazhatók az ASN.1 definíciók [12,15].

2.3. Az SDL jelene és jövője

Manapság két fő irányban végzik az SDL-2000 fejlesztést, éspedig az objektummodellezés és a termékfejlesztés irányában.

Az objektummodellezés jobbító eszközeiként az SDL-2000-be az interfészeket és az ágens elvet vezették be egységesen mindenik SDL architekturális elemre, közelítve ezzel az UML sztereotípusokhoz. A termékfejlesztés jobbításához az adatmodell revíziója, a kivételkezelés és a diagramokon belüli szöveges algoritmusok bevezetése jelentős. Továbbra is fennmaradó gyengesége a nyelvnek az időkezelés és a szinkron műveletek kezelése.

Az SDL nyelv fejlődése a nyelv széleskörű alkalmazásához vezetett az ipari automatizálás területein. A beépített alapműveletek mellé, adatdefiníciós eszközökkel saját típust és műveleteit szerkeszthetjük meg az állapotátmenetek műveletsorozatainak programozásához.

Az SDL nyelv kifejezőereje nagy és rendkívül kényelmes a kommunikáció-vezérelt rendszerek modellezésére, úgy véljük, ez lassítja a matematikai módszerek alkalmazását az újabb verzió formális szemantikájának automatikus helyességbizonyítására és szintézisére (hasonló a jelenség ahhoz, mint amit a VHDL nyelv kialakulásakor tapasztalhattunk) [9].

Az SDL objektum-orientáltsága elérte a napjainkban elvárt objektum-orientált szintet az SDL2000 verzióval, mondhatnánk azt is, hogy a modellfejlesztés általános paradigmái ontológikusan is utolérték az SDL modellezésben de facto létező technikákat, most újabb kihívás az ontológiák alapján kimutatni az SDL-ben eredendően jelenlevő MDA paradigma – megfelelést [23].

2.4. SDL fejlesztőkészletek

Az SDL fejlesztőkörnyezetek legtöbbjének közös funkciói a grafikus szerkesztő, a szöveges és grafikus konverzió, a statikus elemző, a kódgeneráló, a szimuláló és validáló a dinamikus elemzés, az MSC-vel kombinált támogatás.

A Telelogic TAU család egy ipari érvényességűnek (industrial-strength) tartott fejlesztőkészlet, amely a távközlési nyelvek családját támogatja és fejlesztéskövető eszközöket is kínál. Erőssége az MSC, SDL, TTCN modellek környezettámogatásos összeláncolhatósága, ugyanakkor az UML kiindulású folyamatra két fejlesztőeszközt forgalmaz – egyik a rendszerelemzésre, másik az UML2.0 alapon a teljes életciklusra használatos [24].

A Tau SDL Suite 4.6-ös verziónál tart, szolgáltatása a szerkesztés közbeni, tudásmenedzsment alapú szintaktikai ellenőrzés, a szimulációs modellvégrehajtás, a holtpontok és specifikálatlan bemenetek jelzése az állapot-robbanás módszerével, oly módon, hogy a hibakövetéshez MSC nyomvonalat (trace) generál. A TAU család tartalmazza a TTCN fejlesztőkészletet, amellyel SDL modellből is nyerhetünk tesztsorozatokat. A 4.0 verziótól lehetséges az UML modelleknek SDL modellbe transzformálása, majd a validálás, kódgenerálás esetleg tesztsor-generálás SDL alapon. A TAU támogatja a csapatmunkát, akár egyazon időben egyazon modellen is. A TAU család támogatja végrehajtható kód generálását számos valósidejű operációs rendszerben [25].

További SDL szabványt támogató termék a jelenleg Telelogic termék, az ObjectGeode hasonló a Tau SDL-hez, jóval több adattárral és konvertálóval támogatja a valósidejű rendszerek fejlesztését az ipari automatizálásban. A SOLINET cég SDL fejlesztőkészlete valósidejű rendszerek fejlesztéséhez ad támogatást, akadémiai körökben és ipari automatizálásra alkalmazott a Cinderella, demo változata letölthető [26].

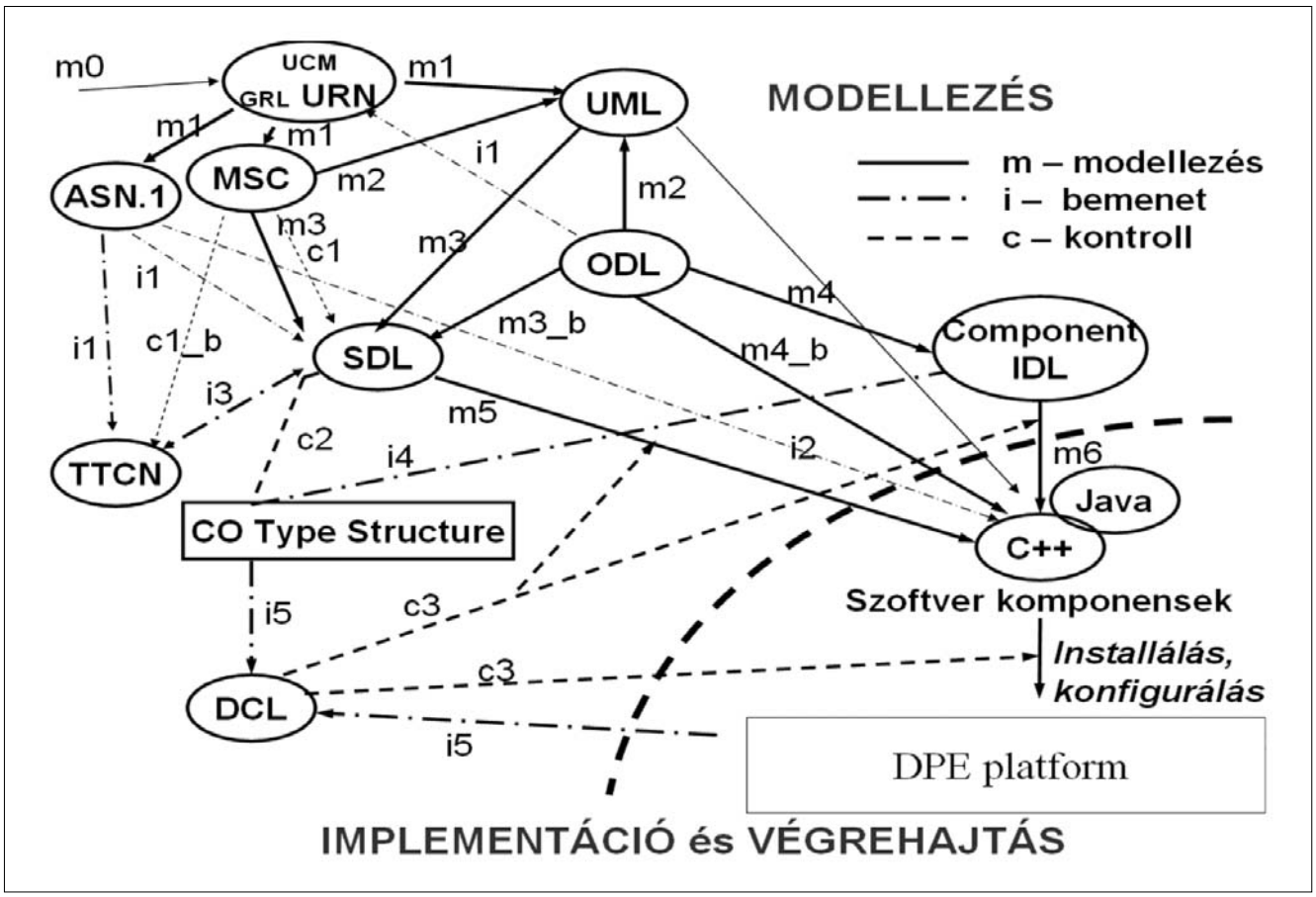
3. A távközlés szabványosított formális nyelvei a fejlesztés fázisaiban

Teret nyertek a fejlesztésben a több paradigma mentén és több nyelvi eszközt használó módszertanok [31]. Az ITU-T és az ETSI munkacsoportjai javasolnak követésre módszertant a távközlési szoftverek fejlesztésére a szabványosított eszközök alapján, de jellemzően a legtöbb módszertan és fejlesztőkészlet ipari környezetben jött létre.

Az ITU-T nyelvek fejlesztőcsoportja meghatározta a System Design Language néven a Z. szabványcsalád azon elemeit, amelyek a teljes fejlesztési folyamatot támogatják.

Az ITU-T szabványkészlete és módszertana az URN-MSC-UML-SDL-TTCN szabványokra épülő inkrementális módszertant ajánlja a munkacsoportjainak eredményeit ismertető konferencianapokon [19].

3. ábra Formális nyelvek egymásra épülése a fejlesztés fázisaiban [20]



A *User Requirements Notation (URN)* szabványa két formális nyelvet, a *Use Case Map (UCM)* és *Goal Requirements Language (GRL)* formális nyelveket tartalmazza, a funkcionális illetve a nem-funkcionális követelmények ábrázolására.

Az UCM leképezhető MSC diagramokra, amelyek többféle módon építhetők be az UML modellbe szekvencia diagramként, az SDL modellbe a statikus és dinamikus leírás során. Az ASN.1 deklarációkat a fordító lefordítja SDL típusra, gyorsítja és egyszerűsíti az átjárást a *Test and Test Control Notation (TTCN-3)* modulokkal a validálás és teszteset generálás folyamataiban.

A rendszer életciklusaiban alkalmazható formális nyelvek kapcsolatát szemlélteti *3. ábra* (az előző oldalon). Tükrözi azt az egységesítési folyamatot, amely jelentősen növeli a fejlesztés hatékonyságát és egyben kiterjeszti a távközlés formális nyelveinek alkalmazását az ipari automatizálás több területére [28].

A rendszerfeltárást URN (UCM és GRL) nyelven ábrázoljuk, az elemzést UML nyelven ábrázoljuk és/vagy elemezzük, fejlesztőeszköztől függően, a koncepciós tervben UCM és MSC diagramokkal ábrázoljuk a rendszer eseményeit, ez konvertálható UML-be és SDL-be is, ASN.1-ben adjuk meg az elemzés során feltárt adatokat, amelyek így beépíthetők az SDL és TTCN specifikációkba.

AZ SDL specifikáció helyességét ellenőrizhetjük MSC idősor diagramokkal. Az SDL speicikációt felhasználjuk a TTCN tesztgenerálás specifikálásához és a forráskód készítéséhez C++ vagy JavaScript nyelven. A *Deployment and Configuration language (DCL)* telepítő és konfigurálás leíró nyelvet az ITU munkacsoportja kidolgozás alatt tartja. Case eszközök automatikusan generálják a kész terméket az SDL specifikációból, és elvégezhetjük a konformancia tesztelést.

4. Összefoglalás

A távközlés eszközeinek széleskörű használatával az elmúlt években a fejlesztői szemlélet is bevonult a különféle heterogén és egymásraépülő rendszerek fejlesztésébe. A több nyelven alapuló technikák nagyon eredményesek lehetnek, mert az összetett rendszerek különféle aspektusainak fejlesztésekor a célnak megfelelőbb eszközt alkalmazhatjuk. Modellelemek újrafelhasználása válik célszerűvé, amely technikáit és munkafolyamatszervezését az automatikus programfejlesztő környezetek és az alkalmazásterület CASE eszközei támogatják.

A távközlési nyelvek néven ismert ITU-T formális módszerek csoportjának és az UML nyelv konvergenciája figyelhető meg az UML2.0 szabvány megjelenésével [30], amely magába integrálta az MSC teljes eszköztárát az interakciók ábrázolására, valamint az SDL nyelv dinamikus viselkedést leíró eszköztárát az implementációs folyamatok állapotgépeinek ábrázolására.

Az SDL mint a formális specifikáció implementációjára jól kidolgozott nyelv, összefogja az ITU-T System Design Language gyűjtőnevű szabványosított formális módszereit. A módszerekben rejlő lehetőségek konkrét hasznosításához továbbra is feladatunk a modellfejlesztés folyamatainak kutatása az UML és a formális modellező nyelvek konvergens elemeire építve az egyes alkalmazás-területek mentén [33,34].

Köszönetnyilvánítás

Köszönettel tartozom Dr. Tarnay Katalinnak, aki a Veszprémi Egyetem Műszaki Informatikai Karán iskolát teremtett a távközlési szoftverek formális módszertanainak kutatására és áldozatos munkájával segített.

A cikkben megjelenített munkát az OTKA 29556 számú kutatási szerződés támogatta.

Irodalom

[1] Tarnay K.:
Kommunikációs protokollok modellezése és konformancia vizsgálata. Doktori értekezés, 1990.
[2] Bőgel Gy.:
Az infokommunikációs hullám sajátosságai, Híradástechnika 2003/5.
[3] Czinkóczky A.:
A távközlési hálózatok fejlődési iránya a következő 5-10 évben, Híradástechnika 2003/10.
[4] Inside the personal communication portal www.intel.com, Intel Glenayre 2002/06.
[5] Z. Mammeri:
SDL modelisation de protocoles et systemes réactifs Hermes 2000.
[6] J. Ellsberger , D. Hogrefe , A. Sarma:
SDL Formal Object-oriented Language for Communicating Systems, Prentice Hall, 1997.
[7] G. Kovács, Z. Pap , D. Le Viet, A. Wu-Hen-Chang, Gy. Csopaki:
Applying Mutation Analysis to SDL Specifications, SDL-Forum, Springer-Verlag. 2003.
[8] Gy. Csopaki, K. J. Turner:
Modelling Digital Logic in SDL.
In Proc. Formal Description Techniques X/Protocol Specification, Testing and Verification XVII, pp.367–382., Chapman and Hall, London, Nov.1997.
[9] Pataricza A.:
Formális módszerek az informatikában, Typotex, Budapest, 2004.
[10] ITU-T Recommendation Z.100 (1995)
"Specification and Description Language (SDL)"
[11] ITU-T Recommendation Z.100 (1999)
"Specification and Description Language (SDL)"
[12] ITU-T Recommendation Z.105 (1995)
"SDL Combined with ASN.1 (SDL/ASN.1)"
[13] ITU-T Recommendation Z.109 (1999)
"SDL Combined with UML (SDL/UML)"
[14] ITU-T Recommendation Z.120 (1999)
"Message Sequence Chart (MSC)"
[15] ITU-T Recommendation X.680 (1994)
"Data Networks and open System communications

– OSI networking and system aspect – Abstract syntax Notation One (ASN.1)"
[16] ITU-T Recommendation X.292 (1998), Z.140-Z.149 (2003): "OSI conformance testing methodologhy and framework for protocol Recommendation for ITU-T applications – The Tree and Tabular Combined Notation (TTCN)"
[17] A. Medve: MSC and the Aspect-Oriented paradigm in protocol engineering, Int. Conf. MicroCAD'2003 Miskolc, pp.71–79.
[18] T. Dulai , A. Medve: The common use of SDL, CSP and MSC in protocol design, Int. Conf. MicroCAD'2003 Miskolc, pp.29–34.
[19] www.itu.int.org/recommendation/zseries/languages
[20] Medve A.: A formális módszerek szerepe a távközlési szoftverek fejlesztésében. Networkshop 2001, www.niif.hu/networkshop
[21] Medve A.: Az SDL nyelv jellemzése. Elektronikus segédlet, VE Információs Rendszerek Tanszék, 2003.
[22] A. Medve: Relations of testability and quality parameters of SDL implementation at the early stage of protocol development life cycle, CSCS'2002, Szeged.
[23] A. Medve: Standardized formal languages for reliable model engineering, Int. Conf. MicroCAD'2005 Miskolc
[24] www.telelogic.com
[25] www.telelogic.com/SDL/default.asp
[26] www.sdl-forum.org/tools
[27] ITU-T Recommendation Z.150-152 (2003) User Requirements Notation – GoalRequirements Language Use Case Map Notation.
[28] Medve A.: A formális nyelvek egymásra épülése a fejlesztés folyamatában, XXIV Neumann Kollokvium, VEAB Veszprém, 2003.
[29] H. Farman: Tau UML Suite Telelogic Szakmai Napok, Inventix Kft.2000.
[30] www.omg.org
[31] V. Vittorini, M. Iacono, N. Mazzocca, G. Franceschinis: TheOsMoSys approach tomulti-formalismmodeling of systems, Software System Modeling 2004/ 3, pp.68–81.
[32] Raffai M: UML 2 Modellező nyelvi kézikönyv, Objektumtechnológia sorozat 4. kötet, Palatia Nyomda és Kiadó, 2005.
[33] A. Medve: Enterprise Modeling with the joint use of User Requirements Notation and UML , Idea Group Publishing, 2005. (fejezet átdolg. alatt).
[34] A. Medve: Process Driven Combination of Scenario-based and State-based Modeling Languages, TSE Special Issue on Interaction and State-Based Modelling, 2005. (beküldve)

Hírek

Az Oracle bejelentette, hogy az **Oracle Database 10g2** új rekordot állított fel a TPC-H két 300 GB-os adattárházas sebességpróbáján is, ami újra igazolja a rendszer kimagasló teljesítményét és a vállalati szintű adatbáziskezelők között kivívott vezető helyét. A négy 3,33 GHz-es Intel Xeon processzorral és 8 MB harmadik szintű gyorsítótárral (EM64T) ellátott két csomópontos Dell PowerEdge 6800 szerverfürtön az Oracle Database 10g2 – Real Application fürtözéssel és Red Hat Enterprise Linux Advanced Server3 operációs rendszeren – világrekordnak számító 22 USD/QphH@300 GB ár-teljesítmény arányt ért el 11.742,8/QphH@300 GB teljesítmény mellett. A második sebességpróbán az Oracle Database 10g2 és az RAC együttesét négy 3,0 GHz-es Intel Xeon processzorral felszerelt, két csomópontos Dell PowerEdge 6600 szerverfürtön, Red Hat Enterprise Linux Advanced Server 3 operációs rendszeren futtatva a 10g első változatához képest 27%-os teljesítménynövekedést értek el. Ennél a benchmarknál 30 USD/QphH@300 GB ár-teljesítmény arány mellett 8604 QphH@300 GB-os teljesítményértéket mértek a 10g előző változatának 6795 QphH@300 GB-os eredményéhez képest.

A Java-fejlesztők már ingyenesen használhatják az **Oracle JDeveloper 10g** fejlesztőrendszert. Az Oracle ezzel a gesztussal a Java-fejlesztők iránti töretlen elkötelezettségét kívánja hangsúlyozni. A cég emellett felajánlotta, hogy az Eclipse Foundation nyílt forráskódú fejlesztői közösségben elvállalja az egységes fejlesztőeszközök kialakítását célzó JavaServer Faces (JSF) projekt vezetését, továbbá bejelentette, hogy fő támogatóként csatlakozik az Apache MyFaces kezdeményezéshez. A J2EE platform népszerűvé válásának egyik feltétele, hogy rendelkezzen olyan eszközökkel, amelyekkel a nagyvállalati Java-alkalmazások egyszerűen fejleszthetők. A Java-alkalmazások felhasználói felületének gyors kialakításához használható JavaServer Faces keretrendszer, valamint az ingyenes Oracle JDeveloper szoftver megkönnyíti az összetett alkalmazások fejlesztését, és így végső soron elősegíti a J2EE platform elterjedését. A webes alkalmazásokat kiszolgáló JSF keretrendszer és az Oracle JDeveloper 10g integrált fejlesztői környezet együttműködésével az összetett alkalmazások gyorsan létrehozhatók és egyszerűen rendszerbe állíthatók.