Az ASN.1 nyelv a protokolltervezésben

POÓS KRISZTIÁN, PAPP ANDRÁS

Veszprémi Egyetem, Műszaki Informatikai Kar, Információs Rendszerek Tanszék

poos.krisztian@irt.vein.hu, papp.andras@irt.vein.hu

Reviewed

Kulcsszavak: kódolási eljárások, formális leíró technikák, mobil adatátvitel

Az ASN.1 nyelv különböző alkalmazások közötti üzenetek leírására szolgál, mint ilyen, magas szintű üzenetleírási formákkal rendelkezik, megkímélve ezzel a protokolltervezőket attól, hogy bit vagy bájt szinten kelljen foglalkozniuk a kommunikációban résztvevő üzenetek felépítésével. Kezdetben e-mail üzenetek leírására használták. Azóta az ASN.1 olyan alkalmazások széles körében is használatossá vált, mint például a hálózat-felügyelet, a biztonságos e-mail, mobil telekommunikáció, légiirányítás, vagy VoIP. Cikkünkben ezt a nyelvet és sokrétű alkalmazhatóságát mutatjuk be.

1. A formális leíró technikák és az ASN.1 kapcsolata

Az ASN.1 nyelv alkalmas adattípusok formális leírására, szabályhalmazokat definiál, amelyekkel bármely adattípus átalakítható egy továbbítható bitfolyammá. A nyelvet (ITU-T X.680 [6], X.691 [8]) alkalmazva a tervezésben időt nyerünk és csökkenthetjük a hibalehetőségeket. A kódolás feladata a modulokkal leírt adatspecifikáció olyan formára hozása, hogy egyértelműen azonosítható legyen a vételi oldalon. Ehhez az ajánlások három szabályhalmazt definiálnak, a típusok és a típusból származtatott értékek reprezentációit, az opcionális mezőt valamint az azonos típusú mezőt. A kódolási szabályokat az X.690-es ajánlás [7] tartalmazza, elnevezésük rendre a következő: BER, CER, DER. Ennek kiegészítése az X.691 és X.693 [9], ami a PER és XER kódolási szabályokat adja a specifikációhoz.

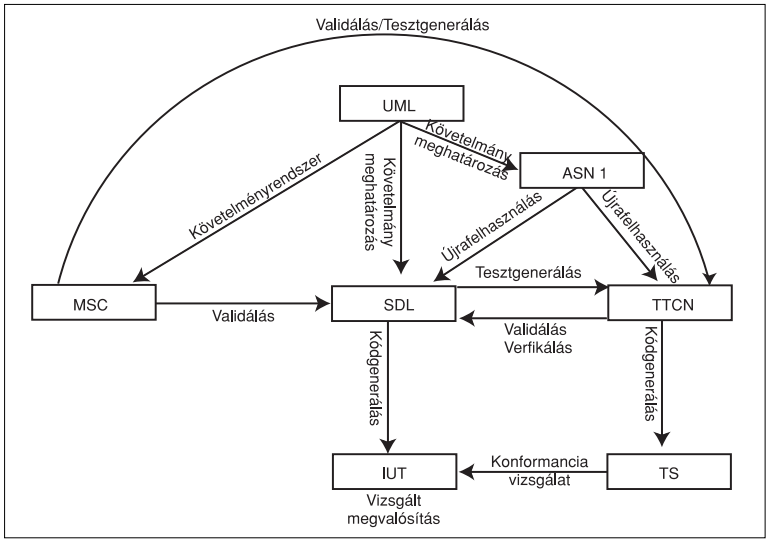
Célszerű egy rendszer viselkedését SDL-ben úgy leírni, hogy az üzenetváltáshoz ASN.1-es adattípusokat használjunk, mert a TTCN nyelv ismeri az ASN.1 adatdefiníciókat, és ez a későbbi a tesztelés során hasznos lehet. Az SDL processzek [3] változókat manipulálnak, amik értékekkel rendelkeznek, melyeket a megfelelő kifejezések kiértékelése adja. Egy változónak csak egy, adott adattípusú értéke lehet. Az adattípust literálok és operátorok összessége együttesen jellemzi. A literálok olyan nevek, amelyek az egyes értékeket jelölik, az operátorok pedig olyan függvények, amelyeket a literálok és a változók fölött alkalmazunk kifejezések szerkesztéséhez. Az operátorok szemantikáját az SDL-ben axiómákkal adjuk meg.

Egy absztrakt adattípus az adatobjektum funkcionális jellemzőit adja meg, tehát a művelet eredményét az adatobjektummal definiálja és azt, hogy megszorítások nélkül miként lehet az adatobjektum által képviselt értékeket megkapni. Az absztrakt adattípus egy vagy több típust (eng. *sort)* definiál, amelyek ismert értékkészlettel és az ezen értelmezett műveletekkel jellemezhetők. Általánosan az absztrakt adattípus egy vagy több osztályt tartalmaz, amelyekre definiálnia kell az operátorokat, amelyek operandusai lehetnek a különböző osztályok, valamint az egyenleteket amelyek eredménye mindig egyetlen osztályt ad.

Tehát az absztrakt adattípus tulajdonképpen osztályok, operátorok és egyenletek összességéből áll.

Az SDL-ben az absztrakt adattípus nincs megnevezve, – csak impliciten létezik – és ennek részeit definiáljuk a különböző adattípus deklarációkkal. Ezt parciális típusdefiníciónak nevezzük. A rendszerspecifikációs fa bármely pontjában egyetlen absztrakt adattípus definíció létezik, amelyet a fa gyökerétől a kérdéses pontig parciális típusdefiníciók alkotnak. Természetesen az absztrakt adattípusok is rendelkeznek öröklődéssel, de nem teljes hierarchia szinten, hanem csak a közvetlen ősöktől van öröklés. A fa egy csomópontjában csak a csomópont direkt őseiben szereplő parciális típusok alkalmazhatóak.

1. ábra Az ASN.1 helye az FDT-k között



Az ASN.1 szabvány [6] megkülönböztet kis- és nagybetűket. Az adattípus kezdőbetűje nagy, a típusból definiált értéké pedig kicsi, valamint a struktúra egy mezője is kisbetűvel kezdődik. A definíció jele '::=', amely egyben a típus és az értékdefiníció jele is.

2. Az ASN.1 története, felhasználhatósága

A számítástechnikai fejlődés kezdetekor a hardvergyártó cégekre nem volt jellemező, hogy a legyártott chipek, processzorok kompatibilisek legyenek egymással. Több cég párhuzamosan fejlesztett és termelt, így változatos architektúrákat készítettek a piac számára, melyeket természetesen specifikusan lehetett csoportosítani.

Ma is sokféle architektúra létezik (x86, Ultrasparc, PowerPC, stb), azonban az informatika hajnalán még több rendszerrel lehetett találkozni. Ilyen különbség például az, hogy nagyon sok rendszer ASCII kódolást, az IBM mainframe-jei EBCDIC kódolást használnak, valamint a PC-k 2-es komplemensű, 16 és 32 bites memória-szavakat, a mainframe-ek 60 bites, egyes komplemensű aritmetikát használnak. Hasonlóképpen felfedezhetünk ábrázolási eltéréseket egy Token-Ring és egy Ethernet hálózat között is. Mindegyik esetben az adatokat más módon kezeli a két különféle architektúra, így szükség van valamilyen közvetítő módszerre, amellyel a kétféle rendszer között adatcserét tudunk végrehajtani.

Amennyiben egy architektúrával dolgozunk, még akkor is felmerülhet adatábrázolási különbség, mert attól függetlenül, hogy az általunk használt eszközök egyazon architektúrára épülnek, még többféle operációs rendszert, és ezen belül sokféle programozási nyelvet használhatunk. Példának vegyünk alapul egy adatstrukúra definiálást egy C és egy Pascal kódrészlettel, *(2. ábra)*.

```
typedef struct header {
   int      mezo1;
   char[8]  mezo2;
   boolean  mezo3;
} header
```

```
Type header = Record
   mezo1 : Integer;
   mezo2 : String[8];
   mezo3 : Boolean;
end;
```

2. ábra C és Pascal kódrészlet

Látható, hogy az adatábrázolás más módon történik a két nyelven. Például egy név tárolására az egyiknél karakterek sorozatát, míg a másiknál string típust használunk.

Ahhoz, hogy egyik architektúráról a másikra, vagy egyik programnyelvről a másik számára érthetővé tegyük a kódot vagy az adatot, valamilyen konverziós eszközre van szükségünk. Definiáljuk ehhez a szükséges szintaxisokat.

Konkrét szintaxisnak nevezzük a küldeni kívánt adatreprezentációkat, egy adott programozási nyelvben. Azért szintaxis, mert figyelembe veszi az adott nyelv lexikai és nyelvtani szabályait, és azért konkrét, mert az alkalmazások kezelik és eleget tesz a gépek architekturális feltételeinek.

Hogy megszabaduljunk a konkrét szintaxisok változatosságától, a továbbítani kívánt adatokat úgy kell leírni, hogy ne legyenek tekintettel a használt programnyelvekre. Ettől függetlenül azonban a leírásnak figyelembe kell vennie egy bizonyos nyelv mind lexikai, mind grammatikai szabályait, azonban mindig függetlennek kell maradnia a programozási nyelvektől és soha nem telepíthető közvetlenül a gépbe. Az ilyen leírást nevezzük *absztrakt szintaxisnak,* Abstract Syntax Notationnek (ASN) pedig a nyelvet, mellyel az absztrakt szintaxis leírható.

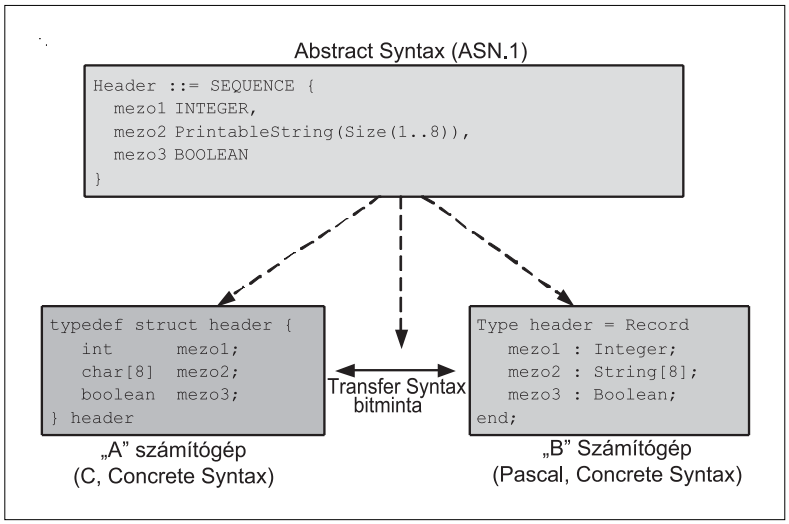
A programozási nyelvektől való függetlenség miatt az absztrakt szintaxisnak legalább olyan erősnek kell lennie, mint bármely nyelv adattípusának, ami tulajdonképpen egy rekurzív jelölés, amely lehetővé teszi komplex adattípusok létrehozását alap adattípusokból (string, int, char stb.) és típuskonstruktorokból (struct, union stb.)

A számítási eszközök általi kezelés és értelmezés alatti bármely félreérthetőség elkerülése végett az absztrakt szintaxisnak formálisnak kell lennie. Az absztrakt szintaxis precízen definiálja az adatot, azonban nincs szemantikai funkciója.

Már csak egyféle szintaxist kell megvizsgálnunk, mégpedig az *átviteli szintaxist.* Ez tulajdonképpen egy félreérthetetlen oktett string halmaz, mely az absztrakt szintaxis egy értékét reprezentálja az átvitel során. Természetesen ez az átviteli szintaxis teljesen az absztrakt szintaxistól függ, csak annyit határoz meg, hogy az adatokat hogyan kell továbbítani az absztrakt szintaxis alapján. Valójában az átviteli szintaxis strukturálja és irányítja a bájtokat, melyeket a másik gépnek küldünk. Az absztrakt szintaxistól eltérően ez egy fizikai mennyiség, és ebből fakadóan számításba kell vennie a bájtok elrendezését, a bitek súlyát stb.

A különböző átviteli szintaxisokat össze lehet kapcsolni egy egyszerű absztrakt szintaxissal. Ez főleg akkor érdekes, amikor megnő az átviendő adat mennyisége, és sokkal bonyolultabb kódolás szükséges: ilyen és ehhez hasonló esetekben lehetőség van az átviteli szintaxis megváltoztatására anélkül, hogy hozzányúlnánk az absztrakt szintaxishoz. Egy egyszerű ASN.1 adatleírásból automatikusan annyi konkrét szintaxist és annyi eljárást tudunk származtatni – ami létrehozza az átviteli szintaxist a kódolókba és dekódolókba –, amennyit csak akarunk.

Az ASN.1 fordító feladata az automatikus generálás végrehajtása, melyet a *3. ábrán* látható szaggatott vonalak mentén haladva végez el. A folyamat során tetemes fáradozástól kíméli meg a felhasználót, miközben lehetővé teszi tetszőleges számú számítógép összekapcsolását. A fordítóba implementálni kell néhány kódolási szabályt, melyek leírják a kapcsolatot az absztrakt és a átviteli szintaxis között.



3. ábra A fenti példára vetített szintaxis hármas

3. ASN.1 az OSI rétegekben

Az ASN.1 [2] felhasználásának bemutatása után térjünk rá használatára az OSI modell rétegeiben. A hét réteg közül csak a két legfelsőre (megjelenítési, alkalmazási réteg) térünk ki, mert csak ezekben jelenik meg az ASN.1.

A megjelenítési réteg az OSI [4] rétegmodell hatodik rétege, és legfőbb feladata biztosítani az adatok kódolását, dekódolását. Ahogy az előző fejezetben láthattuk, az adatábrázolás az architektúrától és a nyelvtől is függhet, ezért egy általános ábrázolás szükséges az adatcsere lebonyolításához. A megjelenítési réteg biztosítja, hogy az adat ebben a formában kerüljön továbbításra, viszont nem törődik az információ jelentésével. Ez gyakorlatilag az, hogy a két rendszernek az adattovábbítás előtt meg kell állapodnia a használni kívánt kódolási szabályban (BER, CER, DER, PER, XER stb.).

Így a megjelenítési réteg az alkalmazási réteg számára biztosított szolgáltatásai a következőek:

– egyezkedés az átviteli szintaxisról
– átviteli szintaxisok egy gyűjteményének ismerete
– fordítás, a konkrét szintaxis kódolási szabályainak használatával az átviteli szintaxisra és vissza
– az egyezkedés során meghatározott átviteli szintaxis összekapcsolása az alkalmazásban elfogadott absztrakt szintaxissal.
– hozzáférés a viszony réteg szolgáltatásaihoz

Az alkalmazási réteg, mint a legfelső (hetedik) réteg feladata az alkalmazások hozzáférése az OSI rétegekhez, továbbá olyan szolgáltatások biztosítása, melyek közvetlenül elérhetőek az alkalmazásból. Egy alkalmazás minden kapcsolati eleme egy-egy alkalmazás-entitás, melyek alkalmazási protokollokat és megjelenítési szolgáltatásokat használnak az információ megosztásához.

Minden egyes alkalmazás adatstruktúrája ASN.1-ben specifikált APDV-ként továbbítódik. Valamennyi esetben, amikor egy alkalmazás adatot kíván küldeni, biztosítja a megfelelő APDV-t; és annak ASN.1 nevét a megjelenítési réteg számára. A megjelenítési réteg ismeri az ASN.1 definícióra vonatkozó adatkomponensek típusát és méretét, valamint kódolásuk, illetve dekódolásuk menetét a továbbításhoz. A túloldalon a megjelenítési réteg analizálja a várt adatstruktúra ASN.1 azonosítóját, miután már tudja, hogy hány bit tartozik az első komponenshez, hány a másodikhoz, etc... Ezzel az információval a megjelenítési réteg végrehajthatja a szükséges konverziókat, hogy biztosítani tudja az adatot a fogadó gép belső felépítésének figyelembe vételével.

Az OSI alkalmazások által használt ASN.1 reprezentáció egyedüli, mióta az ITU javasolta, hogy az összes adatcsere az alkalmazási és a megjelenítési réteg között ASN.1 absztrakt szintaxissal legyen megadva. Az alkalmazási réteg számára azért is szükséges egy ilyen erős és strukturált jelölés, mint az ASN.1, mert itt már nem lehetséges a bitek bájtokban való gyűjtése, mint az alacsonyabb rétegekben. Ezenkívül nem várható el az alkalmazás-fejlesztőktől sem, hogy tökéletesen tudatában legyenek a problémáknak, melyekkel csak akkor találkoznak, ha az üzeneteket bitekké kódolják.

Rövidítések

APDV	Application Protocol Data Value
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation 1
BER	Basic Encoding Rules
CER	Canonical Encoding Rules
DER	Distinguished Encoding Rules
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDGE	Enhanced Data rates for Global Evolution
FDT	Formal Description Techniques
GGSN	Gateway GPRS Support Node
GSN	GPRS Support Node
GTP	GPRS Tunnelling Protocol
ISO	International Standards Organization
ITU-T	ITU, Telecommunication Standardization Sector
MMS	Multimedia Messaging Service
MS	Mobile Station
OSI	Open System Interconnection
PER	Packed Encoding Rules
SDL	Specification and Description Language
SGSN	Serving GPRS Support Node
TCP	Transmission Control Protocol
TTCN	Testing and Test Control Notation
UDP	User Datagram Protocol
UML	Unified Modelling Language
VoIP	Voice over IP
WAP	Wireless Application Protocol
XER	XML Encoding Rules

4. Az ASN.1 szintaxis és jelölésrendszere

Az ASN.1 fő jellemzője, hogy az adatok típusokba vannak sorolva. A típus egy olyan nem üres halmaz, melyet továbbítás előtt kódolhatunk. Az ASN.1 típusoknak [1] a továbbítás miatt speciálisnak kell lenniük, és biztosítaniuk kell a megfelelő funkcionalitásokat.

A főbb ASN.1 típusok a következők: BOOLEAN, NULL, INTEGER, REAL, ENUMERATED, BIT STRING, OCTET STRING, *...String [6], CHOICE, SEQUENCE, SET, SEQUENCE OF, SET OF. Ezen típusok használatával összetett típusokat is készíthetünk.

Amikor egy típust definiálunk, valamilyen nevet kell adnunk neki, hogy hivatkozhassunk rá. A név nagybetűvel kezdődik. Minden ASN.1 hivatkozást a '::=' szimbólum segítségével hozunk létre:

```
Hazas ::= BOOLEAN
```

Az ASN.1 sorok végén nincs pontosvessző.

A SET, SEQUENCE és CHOICE összetett típusok egyes elemei mind egyedi azonosítóval rendelkeznek, mely kisbetűvel kezdődik. Ezen azonosítók segítségével a specifikáció sokkal átláthatóbbá válik és könnyebben olvasható, kezelhető lesz, azonban az adatátvitel során ezek az azonosítók nem továbbítódnak. Így abból a célból, hogy a fogadó gép informálva legyen az értékek típusáról, és hogy az adatot megfelelően tudjuk dekódolni, a továbbító gép kódolója hozzárendel az azonosítóhoz egy 'tag'-et (cédulát). A kódoló alapértelmezés szerint egy 'universal' nevű 'tag'-et használ. Van azonban, amikor az alapértelmezett eset nem elegendő a félreérthetőségek elkerüléséhez, ilyenkor szükséges a „cédulák" határozott jelölése a létrehozandó típusokban a komponensek előtt. A 'tag' (cédula) egy szám szögletes zárójelben, a típus előtt:

```
Koordinatak ::= SET {
          x [ 1]  INTEGER,
          y [ 2]  INTEGER,
          z [ 3]  INTEGER OPTIONAL
}
```

Az ASN.1 megengedi a rekurzív típusok létrehozását is, amennyiben van olyan eleme a rekurzív típusnak, amely véges értékeket tartalmaz:

```
Lottoszam ::= INTEGER (1..49)
Lottohuzas ::= SEQUENCE SIZE (6) OF Lottoszam
```

Amennyiben már megírtuk az egyes ASN.1 jelöléseinket, csak össze kell gyűjtenünk azokat és egy közös specifikációban egyesítenünk, mely leírja az adatátvitel szabályait.

Ez szabálycsoport tulajdonképpen egy protokoll specifikációjának tekinthető. Egy adott specifikáció egy vagy több ASN.1 modult tartalmazhat, ahol minden egyes modul egybefogja a típusokat, értékeket, osztályokat. A modulnevek nagybetűvel kezdődnek, és BEGIN és END kulcsszavak közé fogják a modulban definiált típusokat:

```
Module DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
...
END
```

Az AUTOMATIC TAGS azt jelenti, hogy a specifikáció készítőjének nem kell foglalkozni a szögletes zárójelekbe helyezett 'tag'-ekkel, mert azok automatikusan létrejönnek a fordító által.

5. Az ASN.1 kódolási szabályai

Basic Encoding Rules (BER)

A BER [1] kódolás formátuma minden esetben egy TLV hármas, ahol az egyes elemek jelentése:

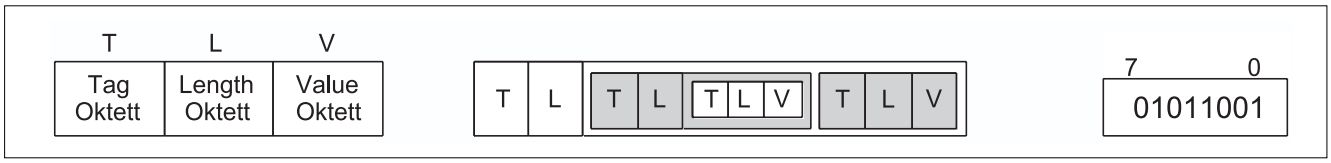
T – type/tag, L – length, V – value.

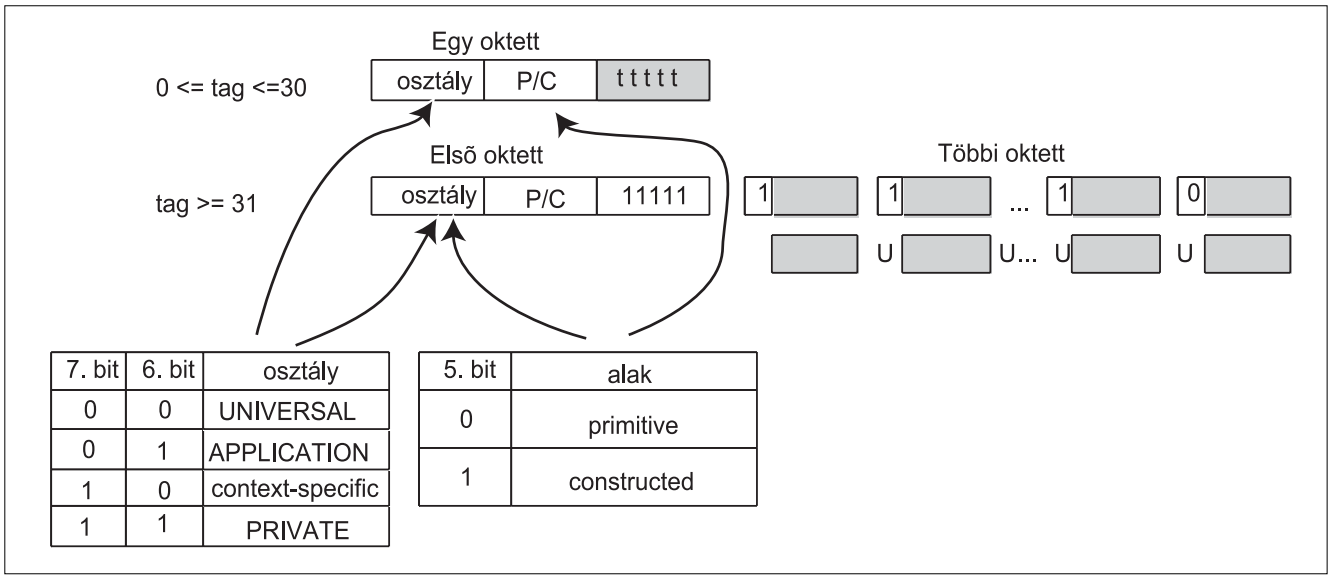
Mindegyik mező oktettek sorozata. Maga a V érték lehet egy új TLV hármas is. A BER kódolás 'Big Endian' kódolás, ugyanis a legmagasabb helyi értékű bit a bal oldalon található. A 'tag' oktettek (általában egy oktett elegendő) megfelelnek az értéktípus kódolt címkéjének. Ha a 'tag' mező hossza kisebb, mint 30, akkor az osztályok és számok kódolt hossza egy oktett lesz. Ha a 'tag' hosszabb 30-nál, akkor a szám a 6-0. sorszámú – ne felejtsük el, hogy itt a bitek sorszámozása 'Big Endian' módszerrel történik, melyet a *4. ábrán* is láthatunk – bitek összefűzéséből épül fel minden oktettben, kivéve az elsőt, ahol az alsó 5 bit mindegyike 1-es értékű lesz. Az utolsó oktettet kivéve mindegyikben a 7. számú bit értéke mindig 1. Az első T mező 5. számú bitje határozza meg, hogy a V csak értéket (primitive) vagy másik TVL hármast (constructed) tartalmaz.

Az L mező tartalmazza az aktuálisan kódolt érték (V) hosszát. Amennyiben az első T mező 5. bitje 'primitive' kódolási formát jelez, az L mezőt határozott alakban kódolja, ellenben ha az 5. bit 'constructed' formát jelez, az L mező kódolási formátumát a küldő fél választhatja meg, hogy határozott vagy határozatlan formában történjen.

A határozott alak lehet rövid (ha az L mező 127-nél kisebb), és lehet hosszú, a küldő döntésétől függően. Ez a szabadság megengedi, hogy a protokoll réteg egy bizonyos számú oktetten kódolja az összes L mezőt, két gép közötti specifikus kommunikációnál.

4. ábra Balról jobbra: A TLV szekvencia primitive és constructed esetben és a 'Big Endian' bitsorrend





5. ábra A T mező két lehetséges formátuma

A hosszú formában az L rész első oktettje a length mező hosszát reprezentálja.

A határozatlan forma kódolása olyan esetekben szükséges, amikor nem a teljes tartalmi rész ismert a küldő számára, így annak hossza nem állapítható meg a kódolás előtt. Ezenfelül másik előnye a határozatlan formának, hogy megvéd minket az értékek kétszeres vizsgálatától – mely először a hossz megállapításánál, majd a tényleges adatkódolásnál történik –, így hatékonyabb kódolókat készíthetünk. Ha az érték határozatlan alakban van kódolva, két zéró oktett zárja le a kódolt adatot. Ez a két utolsó oktett valójában egy TLV hármas, amely egy [UNIVERSAL 0]-val címkézett 0 hosszúságú értéket jelképez.

A BER kódolás architektúra független, hiszen erre az architektúrákban a bitsorrend adott és a kódolási szabályok könnyen konvertálhatóak.

Canonical and Distinguished Encoding Rules (CER/DER)

Az olyan kódolási szabályt, amely semmilyen szabadsági fokot nem hagy, kanonikus kódolási szabálynak nevezzük. A BER-ből két kanonikus kódolási szabályt származtattak, a CER-t [1] és a DER-t [2], amelyek tulajdonképpen a BER specializációi. Ez azt jelenti, hogy egy CER-rel vagy DER-rel kódolt szöveget egy BER dekódolóval tudunk dekódolni. Természetesen ez a másik irányba nem működik.

A két kódolási szabály egy érdekes tulajdonságot, az absztrakt értékek és kódolásuk közötti kétirányúságot adja kezünkbe, aminek segítségével bármely ASN.1 absztrakt értékhez egy oktett stringet tudunk rendelni, és fordítva. Bármely oktett stringhez létezik egy hozzátartozó absztrakt érték.

Ezzel a tulajdonsággal a fogadó alkalmazás összehasonlíthatja a fogadott oktett stringet egy megadott oktett stringgel anélkül, hogy tudná az értéket, amihez az valójában hozzárendelhető.

A kulcsfontosságú különbség a két szabály között az, hogy a CER a 'constructed' alaknál határozatlan, míg a DER határozott alakot használ. Emiatt a CER kódolást olyan alkalmazásoknál használják, amelyeknek nagy mennyiségű adatot kell továbbítani.

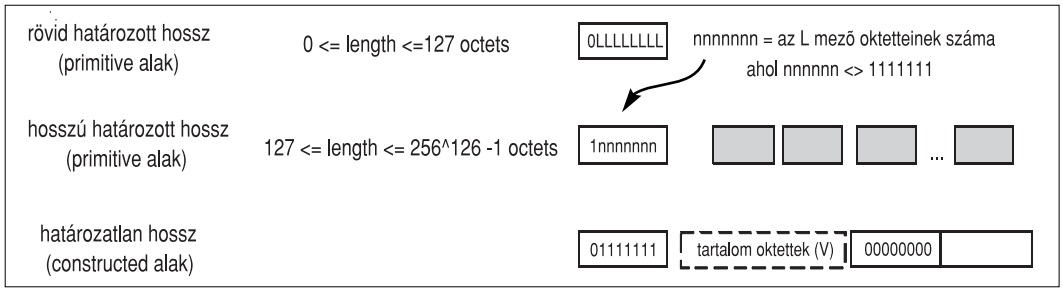
XML Encoding Rules (XER)

A XER kódolási szabály [1] lényege, hogy az ASN.1 értékeket XML nyelvre kódolja át. Az alapvető ötlet, hogy határoljuk az ASN.1 elemeket a következő XML címkékkel: <MARK> ... </MARK>.

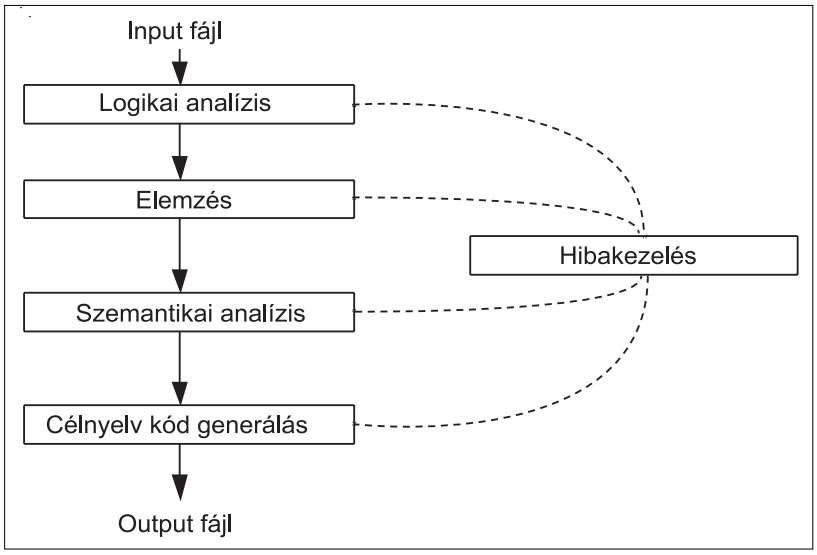
Ez azt jelenti, hogy egy típus értékei a következőképpen kódolhatók:

```
PDU ::= SEQUENCE {  komponens1 SEQUENCE OF T,
                    komponens2 U }

<KOMPONENS1>...</KOMPONENS1>
<KOMPONENS1>...</KOMPONENS1>
<KOMPONENS1>...</KOMPONENS1>
<KOMPONENS2>...</KOMPONENS2>
<KOMPONENS2>...</KOMPONENS2>
```



6. ábra Az L mező lehetséges formái



7. ábra
A fordítás lépései, egy idealizált fordító felépítése

6. Az ASN.1 fordító

Általánosságban a fordító egy olyan számítási eszköz, amely beolvas egy programot mely első nyelven, a forrásnyelven íródott, és lefordítja azt egy második nyelvre, amely a célnyelv, és tulajdonságait tekintve már adott gép architektúrájának megfelelő ábrázolási módot követ. Természetesen minden egyes architektúrához külön, az ahhoz készített fordító szükséges. A mi esetünkben a forrásnyelv az ASN.1, a célnyelv pedig lehet C, C++ és Java, a program pedig egy specifikáció, mely néhány modulból épül fel.

Egy idealizált fordító négy rétegre bontható, melyek mindegyike csak akkor működik, ha a fölötte lévő réteg hibátlanul fejezte be működését. Az elemzési és lexikai hibákat a forráskódban lévő meg nem engedett karakterek, vagy grammatikai struktúrák gerjesztik, míg a szemantikai hibákat az inkoherens specifikációk idézik elő (például egy INTEGER hozzárendelése BOOLEAN-ként deklarált értékhez).

Amennyiben a kód nem tartalmaz sem szintaktikai, sem szemantikai hibákat, a fordító általában a következő fájlokat generálja:

– egy fájlt a konkrét szintaxissal, amely az ASN.1 specifikációban szereplő adattípusok fordítása a megfelelő célnyelvre,

– egy vagy több fájlt, amely tartalmaz egy kódoló és egy dekódoló eljárást az ASN.1 specifikációból minden egyes típusra, amely megvalósítja a kódolási szabályokat, valamint generálják az átviteli szintaxist.

7. A GTP és útprotokollja

Egy teljes ASN.1 specifikáció létrehozását a GTP és annak útprotokollja segítségével próbálunk meg bemutatni, azonban ehhez szükség van a GTP protokoll, és az azt magában foglaló GPRS rövid ismertetésére is. A specifikáció felépítése során a kiindulópont a GPRS hálózat megismerése, ami után már fel tudjuk építeni a függelékben található ASN.1 specifikációt.

A cikkünkben bemutatott példát úgy próbáltuk meg kiválasztani, hogy mindenképen egy viszonylag új technológiát vizsgáljunk meg és ez a mobil távközlés területén használatos rendszer legyen. Így került a GPRS rendszerre és a GTP protokollra a választás.

Mivel a GPRS nemcsak a ma még jóval elterjedtebb GSM, hanem a közeljövőben egyre inkább teret hódító EDGE hálózatokon is használható, érdemes ennek fokozott figyelmet szentelnünk. Napjainkban a legelterjedtebb alkalmazások kapcsán is előtérben van ez a szolgáltatás, hiszen GPRS-t használhatunk WAP és Internet oldalak böngészésekor, vagy MMS üzenetek küldése során. Éppen ezért a GPRS mérföldkőnek számít a GSM hálózatok fejlődésében, a ma rendelkezésre álló hálózati infrastruktúrán, – és ez az az ok, ami miatt ezzel a rendszerrel és protokollal foglalkozunk.

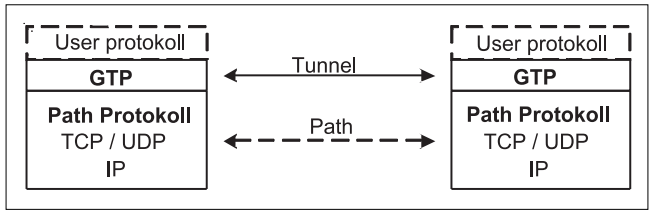
Egy GPRS hálózatban több GSN található, amelyek IP protokollon keresztül tartják egymással a kapcsolatot. Ezek lehetnek GGSN-ek vagy SGSN-ek. Az SGSN kommunikál a mobil készülékkel, a GGSN az átjáró az Internetre. Így, amennyiben egy mobil készülékről például egy www. oldalt böngészünk, az adatok útvonala rendre a következő lesz:

MS ⇒ SGSN ⇒ GGSN ⇒ webszerver.

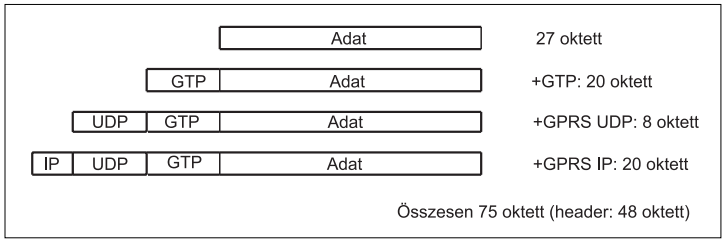
Ez a virtuális adatútvonal.

A GTP alagút [10] ebben a hálózatban tulajdonképpen két GSN között található meg. A két GSN (általában a GGSN és az SGSN) egy virtuális kapcsolaton keresztül, a GTP alagúton tartja egymással a kapcsolatot, amely a *8. ábrán* is látható.

8. ábra A GTP és a GTP útprotokoll szerkezete



9. ábra
Adatcsomaghoz a GTP útprotokoll által hozzáadott fejrészek



A GTP az adatcsomagokat a *9. ábrán* látható módon továbbítja, először egy GTP fejrészt tesz az adat elé, majd ezt egy TCP/UDP [5], végül pedig egy IP [5] fejrésszel egészíti ki. A másik oldalon rendre lebontja ezeket a fejrészeket, és megkapja a szükséges adatot, melyet tovább küldhet a mobil készülék vagy az Internet irányába. Ezt a folyamatot végzi GTP útprotokoll, melynek adatszerkezete a függelék részben az ASN.1 specifikációban található meg, ahol látható a fejrészek elhelyezkedése, illetve az, amint az egyes fejrészek után következő csomagrész újabb fejrészeket tartalmaz, ahogy a GTP csomag egy TCP vagy egy UDP csomagot.

8. Összefoglalás

A formális leíró technikák meglehetősen fontos szerepet töltenek be a protokoll-tervezésben. Ezért nagyon fontos az, hogy biztosítsuk a gyors átjárhatóságot az egyes FDT-k között. Itt kerül a képbe az ASN.1, amely legfőbb tulajdonságának – a hardverfüggetlenségnek – köszönhetően tökéletesen alkalmas erre a feladatra. Úgy is mondhatnánk, hogy az összekötő kapocs szerepét tölti be a formális technikák, az UML, az SDL és a TTCN között.

Az ASN.1 segítségével teljes protokoll-specifikációkat tervezhetünk és magas szintű alkalmazásspecifikációkat írhatunk le. Mindenképpen szükségünk lesz erre a protokolltervezés egyes lépései között, azonban két legfőbb tulajdonsága – olyan érthető, mint amilyen absztrakt – megszabadít minket minden korláttól, mely akadályozhatja tervezői tevékenységünket.

Irodalom

[1] Olivier Dubuission:
ASN.1 – Communication between heterogeneous systems
Morgan Kaufmann Publishers, 2000.

[2] Prof. John Larmouth:
ASN.1 Complete
Morgan Kaufmann Publishers, 1999.

[3] J. Ellsberger, D. Hogrefe, A. Sarma:
SDL Formal Object-oriented Languages for Communicating Systems,
Prentice Hal Europe, 1997.

[4] OSI – A Model for Computer Communications Standards
U. Black, Prentice-Hall, 1994.

[5] Andrew S. Tanenbaum:
Számítógép-hálózatok
Panem-Prentice Hall, 1999.

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS,
OSI networking and system aspects – Abstract Syntax Notation One (ASN.1)

[6] Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation
ITU-T, Rec. X.680, 07/2002.

[7] Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
ITU-T, Rec. X.690, 06/1999.

[8] Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)
ITU-T, Rec. X.691, 06/1999.

[9] Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)
ITU-T, Rec. X.693, 12/2001.

[10] Jyke Jokinen:
GPRS & UMTS Protocols: GTP details,
Tampere University of Technology,
Department of Information Technology,
Advanced Topics in Telecommunications, 2000.
www.cs.tut.fi/kurssit/8309700/reports/gtp-report.pdf

Függelék

A GTP útprotokoll ASN.1 specifikációja:

```
Module-packets DEFINITIONS
AUTOMATIC TAGS ::=
  BEGIN

Ip ::= SEQUENCE {
  ip-header     SEQUENCE {
    ip-version  INTEGER,
    ip-header-length  INTEGER,
    type-of-service   SEQUENCE {
          precedence  BIT STRING,
          delay       BIT STRING,
          throughput  BIT STRING,
          reliability BIT STRING,
          unused      BIT STRING
    },
    full-length       INTEGER,
    identification  OCTET STRING,
    unused      BIT STRING,
    dont-fragment     BIT STRING,
    more-fragment     BIT STRING,
    fragment-offset   BIT STRING,

    life-time   INTEGER,
    protocol-type     BIT STRING,
    header-checksum   BIT STRING,

    source-address OCTET STRING,
    destination-address OCTET STRING,

    options     OCTET STRING
    },
```

```
   ip-packet SEQUENCE {
   }
}

Tcp ::= SEQUENCE {
   tcp-header     SEQUENCE {
      source-port OCTET STRING,
      destination-port  OCTET STRING,
      sequence-number   OCTET STRING,
      acknowledgement-number  OCTET
                              STRING,
      tcp-header-length INTEGER,
      unused       BIT STRING,
      urg-bit      BIT STRING,
      ack-bit      BIT STRING,
      rst-bit      BIT STRING,
      psh-bit      BIT STRING,
      syn-bit      BIT STRING,
      fin-bit      BIT STRING,
      window-size        INTEGER,
      checksum     OCTET STRING,
      urgent       OCTET STRING,

      options      OCTET STRING
      },
   tcp-packet     SEQUENCE {
      ip-packet    Ip
   }
}

Udp ::= SEQUENCE {
   udp-header     SEQUENCE {
source-port       OCTET STRING,
destination-port OCTET STRING,
udp-segment-length INTEGER,
udp-checksum      OCTET STRING
},
   udp-packet     SEQUENCE {
ip-packet  Ip
   }
}

END
```

```
Protocol DEFINITIONS AUTOMATIC
TAGS ::=
     BEGIN
IMPORTS Ip, Tcp, Udp
FROM Module-packets;

PDU ::= CHOICE {
  gtp-header SEQUENCE {
     gtp-version        INTEGER
                        DEFAULT 0,
     pt                 INTEGER
                        DEFAULT 1,
     spare              BIT STRING
                        DEFAULT
                        '111'B,
     snn                INTEGER,
     message-type       OCTET
                        STRING,
     length             OCTET
                        STRING,
     sequence-number    OCTET
                        STRING,
     flow-label  OCTET STRING,
     sndcp-n-pdullc-number   OCTET
                        STRING,
     spare1             BIT STRING
           DEFAULT '11111111'B,
     spare2             BIT STRING
           DEFAULT '11111111'B,
     spare3             BIT STRING
           DEFAULT '11111111'B,
     tid          OCTET STRING}
  },
   gtp-packet CHOICE {
     tcp-packet  Tcp,
     udp-packet  Udp
   }
}

END
```