Debreceni Egyetem

Informatikai Kar

Telekommunikációs hálózatok használata során keletkező adatok feldolgozása

Témavezető:

Espák Miklós

egyetemi tanársegéd

Készítette:

Sikolya Tibor Gábor

programozó matematikus hallgató

Tartalomjegyzék

1 Bevezetés

A telekommunikációs szektor minden bizonnyal leggyorsabban fejlődő ágazata a mobil telekommunikáció. A mobil telekommunikációt megvalósító végfelhasználói eszközök [13] (mobiltelefonok, PDA-k, WLAN interfésszel ellátott laptopok stb.) a mai emberek életének elválaszthatatlan részévé váltak, nagyon sok médium foglalkozik velük.

Ennek ellenére a mobil telekommunikációs hálózatok architektúráját, azaz a mobilszolgáltatók rendszereit és az ott használt technológiákat még informatikus körökben is kevesen ismerik, hazai szakirodalmuk pedig nem túl gazdag.

2005. júliusa óta dolgozom rendszerintegrátorként (a korábban T-Systems Hungary) az IT-Services Hungary kft. SI POP (Systems Integration, Point Of Production) részlegén. A részleg kizárólag német ügyfeleknek nyújt – telekommunikációs területen – rendszerintegrációs szolgáltatásokat.

Az SI POP Mobile Billing csoportján belül működő iBMD projekten a feladatunk a T-Mobile International mobil számlázási rendszerének mediation rétegbeli funkcióit ellátó, Európa-szerte használt iBMD (international Billing Mediation Device) rendszer fejlesztése, tesztelése és karbantartása. Ez alatt az idő alatt volt lehetőségem alaposan megismerni a szolgáltató rendszereit.

Szakdolgozatom első felének célja a mobilszolgáltatók hálózati architektúráinak és számlázási rendszereinek rövid ismertetése. Igyekszem csak a leglényegesebb dolgokra koncentrálni, melyek feltétlenül szükségesek a további fejezetek megértéséhez. E rendszerek mélyebb ismertetése ugyanis akár köteteket is magában foglalna.
A fent említett részben leírtak csak a mobil hálózatokra összpontosítanak, de szoros analógiát mutatnak a vonalas telefonhálózatok jellemzőivel.

A szakdolgozat többi részében egy gyakorlati problémából kiindulva jutok el az azt megoldó alkalmazáshoz, a megoldás értelmezéséhez szükséges ismeretek bemutatásán keresztül.

Az iBMD egy rendszerintegrációs platformra, a Comptel cég EventLink 4 (EL4, régebbi nevén MDS/AMD, [4]) megoldására épül. Az iBMD célja nagyon tömören fájl- és rekordfeldolgozás. Adott egy fájl, melyet valamilyen hálózati elemről tölt le az iBMD (pl.

SMS kapcsolóközpontról). Ezek a fájlok számlázáshoz szükséges információkat tartalmaznak (számlázási fájl). A fájlokban lévő információk valamilyen módon kódolva vannak. Ezekből a fájlokból kell a számlázási rekordokat beolvasni, dekódolni, elemezni, osztályozni, majd a célrendszerek által használt formátumra leképezni és továbbítani. A kimeneti kódolás és az információk jellege nagyban eltér a bemenetitől.

Az egyik ilyen kódolás az ASN.1 BER (Abstract Syntax Notation One, Basic Encoding Rules) [1] [9]. Ha a szolgáltató úgy dönt, hogy új üzletmenetet szeretne megvalósítani az iBMD-n belül, mert például a roaming rekordokat tartalmazó fájlokat feldolgozó alkalmazását szeretné az iBMD-re migrálni [6] [7] [8], akkor egy új Line of Business-t (adott típusú fájlokat feldolgozó munkafolyamatot, röviden LoB-ot) kell implementálnia. Az implementáció egyik legkritikusabb pontja az EL 4 beolvasó alkalmazása, a FileReader [3] által használt inputleíró megírása és tesztelése. Az ASN.1 fájlok formátumát egy ASN.1 specifikáció írja le absztrakt módon.

Mikor egy ilyen LoB inputleírójának megírását kaptam feladatul, akkor vetődött fel bennem a kérdés, hogy lehet-e vajon teljesen automatizálni a két nyelv közötti konverziót. Akkor ez idő hiányában csak részben sikerült. A legnagyobb kihívást, az ASN.1 specifikáció szintaktikai és szemantikai elemzőjét, valamint azt a magas szintű logikát, mely felismeri és kezeli az ASN.1 nyelv mélyebb összefüggéseit nem volt lehetőségem megírni, de most, e szakdolgozat keretei között igen.

A feladat tehát az, hogy ASN.1 absztrakt jelölő nyelven írt specifikációból állítsunk elő a FileReader által használt inputleírót. Egy ASN.1 specifikáció minden eleme egyben típus is, mely újrafelhasználható más elemek típusának definíciójában. Így a specifikáció típusaiból fa építhető. A két nyelv közti legnagyobb különbség az, hogy míg az ASN.1 leíróban minden típus csak egyszer szerepel, a FileReader inputleíró nyelvében ugyanúgy, mint a BER kódolású számlázási fájlban valamennyi helyen ahol csak hivatkoznak rá. Az alkalmazásnak ezt a különbséget is fel kell oldania.

A megfontolás gyakorlati jelentősége az, hogy mivel munkánk során gyakran találkozunk ASN.1 BER fájlokkal és ASN.1 specifikációkkal, szükség van egy olyan alkalmazásra, ami a fent leírt folyamatot emberi beavatkozás nélkül helyesen megoldja, így kizárva a hiba

lehetőségét. Másrészt, mivel a fent említett LoB inputleírójának hossza több mint 3000 sor lett, látható, hogy az alkalmazás használatával rengeteg idő takarítható meg.

A szakdolgozat második felében ismertetem az ASN.1 leíró nyelv általunk használt részhalmazát és a BER kódolást, a FileReader inputleíró nyelvét, majd magát az ASN1BERInputDescGenerator-t.

Az ASN.1-et és a BER-t nem csak mobil hálózatokban használják széles körben, és az EL 4 is alkalmas bármilyen mediációs feladat ellátására, ezért a dolgozat második fele kevésbé mobil telekommunikáció specifikus.

2 Mobil telekommunikációs hálózatok

A *telekommunikáció* információk valamilyen távolságra való továbbítása vagy cseréje elektromos eszköz (vagy eszközök) segítségével. A fogalom magában foglalja az ilyen jellegű kommunikációt, valamint az azt megvalósító eszközöket és rendszereket.

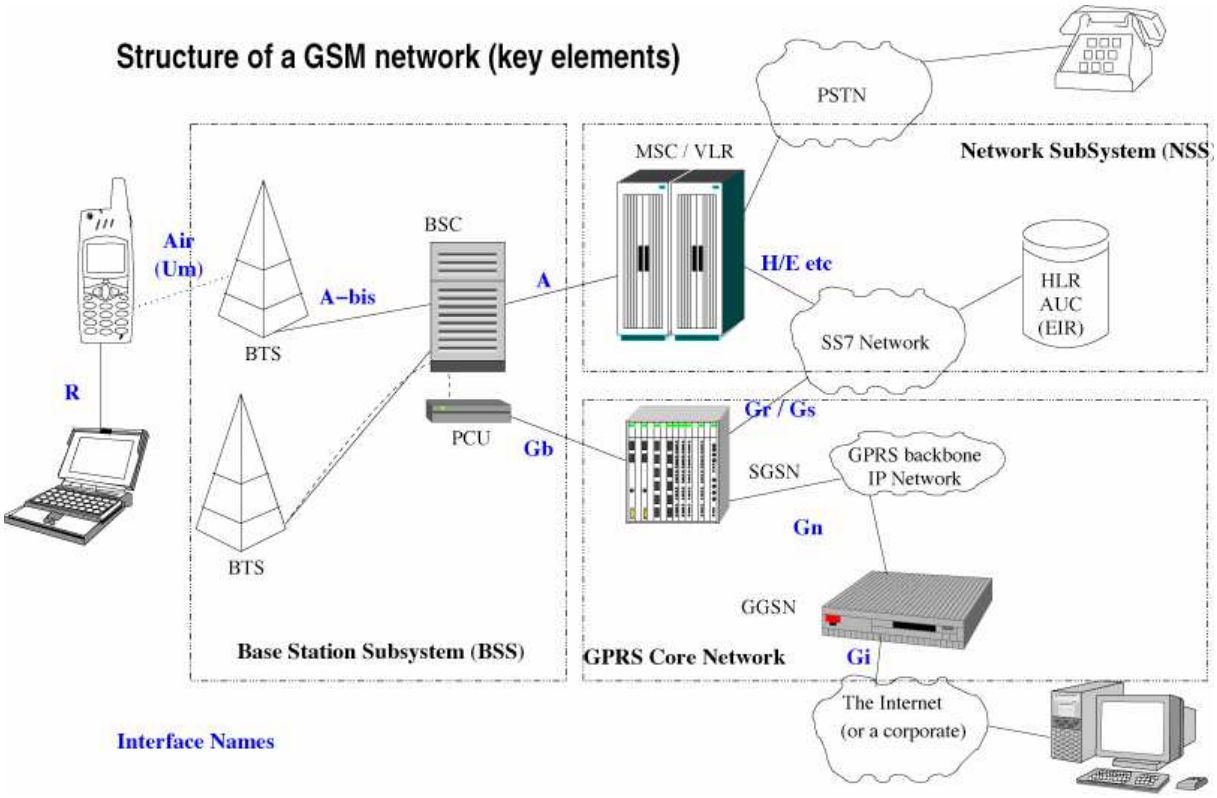
A telekommunikációs rendszerek három fő elemből álnak: az *adó* mely az információt jelekké kódolja, az *átviteli közeg*, amin keresztül a jel továbbítódik, és a *vevő*, ami a jeleket ismét információvá dekódolja. A telekommunikácó lehet *pont-pont* (pl. telefon) vagy *üzenetszórás* jellegű (pl. rádió).

A továbbiakban csak a *digitális jelekkel* való telekommunikációról beszélünk. A telekommunikációs végfelhasználói eszközök és az azokat kiszolgáló rendszerek *telekommunikációs hálózatot* alkotnak, ilyen pl. a vonalas telefonhálózat (*PSTN* – Public Switched Telephone Network). Az adatátviteli réteg jobb kihasználtsága végett a telekommunikációs eszközök valamilyen multiplexeléses közeghozzáférési technikát alkalmaznak, pl. FDMA, WDMA, CDMA, PDMA, TDMA.

Mobil telekommunikációról beszélünk akkor, ha a végfelhasználói eszköz hordozható, azaz mobil.

A szakdolgozat első fejezeteiben telefonhálózatokról, azon belül elsősorban mobiltelefon hálózatokról lesz szó. A leírtak jelentős része ettől függetlenül a telekommunikáció egészére is értelmezhető, de ezeket a hasonlóságokat és eltéréseket nem részletezem.

2.1 A mobilszolgáltatók rendszereinek felépítése



1. ábra A GSM hálózatok legfontosabb elemei

Ebben a fejezetben elsősorban a GSM alapú, azaz a 2G illetve a 2.5G mobil hálózatok felépítését vettem figyelembe [2]. Ezek általános felépítése látható az 1. ábrán.

A mobil hálózatok végfelhasználói eszköze valamilyen *mobil eszköz* (Mobile Station [13]) mely lehet pl. mobiltelefon, PDA, laptop, vagy más hasonló jellegű, például egy mobil internetezésre használt USB portra csatlakoztatható eszköz.

Egy adott szolgáltató által lefedett terület cellákra van osztva. Minden cellában egy-egy BS (Base Station, bázisállomás [14]) van. Egy bázisállomáshoz több TRX, azaz adó-vevő tartozik. A bázisállomások továbbítják az információkat közvetve egymás, a PSTN, vagy egy MSC/VLR felé. Közvetlenül a *bázisállomás-kontrollerekkel* (Base Station Controller) vannak kapcsolatban. Ezek biztosítják a bázisállomásokat irányító magasabb intelligenciát. Egyik legfontosabb feladatuk az MSC/VLR-ek felé irányuló forgalom redukálása, valamint ez az

OSS rendszerek felé irányuló információáram kiinduló forrása is. A BS-ek és a BCS-ek együtt alkotják a BSS-t (Base Station Subsystem, bázisállomás-alrendszer.)

Az információk az MSC-khez (Mobile Switching Centre, mobil kapcsolóközpont, [12]) kerülnek, mely már az NSS (Network SubSytem, hálózati alrendszer) része. Főbb feladatai: autentikáció, helymeghatározás, a hívások regisztrálása és továbbítása. A VLR (Visitor Location Register [15]) az MSC része. Feladata a mobil eszközök cellák és területek közti mozgásának figyelése, mely nélkülözhetetlen a hívások megfelelő helyre való továbbításához és a megfelelő minőségű szolgáltatásokhoz.

Az MSC feladata még a cellaváltások és hívásfelépítések menedzselése a BSS rendszerekkel és az MS-sel folytatott kommunikáció alapján.

Amennyiben más hálózatokkal is kommunikál akkor ez egy Gateway MSC (átjáró). Ezeken keresztül zajlik a kommunikáció például a GPRS alrendszerrel, vagy a PSTN hálózattal (a szolgáltatók a hívásokat általában vonalas telefonhálózaton továbbítják egymás között), vagy valamilyen emeltdíjas szolgáltatást biztosító szolgáltatóval (pl. autópályadíj fizetés mobilról), vagy éppen a szolgáltató hálózatának terheltségét monitorozó-felügyelő rendszerrel.

Az OSS rendszerek számára szükséges hívás-adatokat is az MSC-k hozzák létre az alhálózatukból kapott információk alapján, majd ők tárolják el azokat. Az előbb felsorolt rendszerek (a legfontosabbak) alkotják a szolgáltatók hálózati rétegét.

A következő az ún. Mediation & Provisioning réteg. A Provisioning teszi lehetővé a szolgáltatások és az előfizetők valósidejű aktiválását, erősen kötődik a SAS-hoz (Subscriber Administration System, előfizetőket nyilvántartó rendszer). Itt található még a prepaid (feltöltőkártyás) ügyfelek egyenlegét menedzselő Balance Management is. A legfontosabb pedig a Billing Mediation vagy Mediaton, mely egy interfész a hálózat és az OSS/BSS rendszerek között.

A következő rétegben az OSS/BSS – mely az Operation Support System / Business Support System rövidítése, leginkább működéstámogató/üzletmenet-támogató rendszernek lehet fordítani – rendszerek találhatóak. Ezen rendszerek közül néhány: rater (árazás), billing (számlázás, pre- és postpaid), roaming billing, Data WareHause (adattárház, statisztikák) , NMS (Network Management System, hálózat menedzselő rendszer), Interoperator System

(más szolgáltatók saját hálózaton belüli hívásainak adminisztrációja), Fraud Detection (visszaélés-érzékelés, a hálózat illegális használatát figyeli), cWB (common Wholesale Billing, viszontértékesítők számlázása), legal investigatement system (rendészeti szervek számára biztosított hívás-adatbázis) stb.

2.2 Hálózatok szolgáltatásai és hálózat-típusok

A mára már mindenhol elterjedt 2G (2nd Generation, második generációs) mobil hálózatok legjelentősebb szabványa a GSM [11]. A 2G hálózatok néhány szolgáltatása: Voice, GPRS, SMS, MMS, Value Added Service (emeltdíjas szolgáltatások).

A 3G-t részben megvalósító, 2G-t meghaladó képességű hálózatokat gyakran 2.5G-nek is hívják. Ez már tartalmazza például a PushToTalk (adó-vevő), a Homezone (otthoni mobil), e-mailszolgáltatásokat.

A 3G hálózatok [10] újdonságai: multimédiás adatátvitel (pl. videokonferencia), nagysebességű adatátviteli technológiák (HSDPA, UMTS). A 3G hálózatok szabványa az UMTS.

Létezik már 4G hálózat is, mely még a fejlesztés stádiumában van.

A különböző szolgáltatásokat alhálózatok, hálózat-típusok nyújtják, meghatározott rendszerekkel és hardverelemekkel. Ezek a részhálózatok nem feltétlenül teljesen diszjunktak.

2.3 Billing Mediation

Ebben a fejezetben a számlázási mediációt (*Billing Mediation* [4]) és alapfogalmait ismertetem.

A *mobil operátor* kifejezés a mobil szolgáltatót jelenti.

Hálózati elem (Network Element, NE) a mediation és az OSS/BSS szemszögéből bármely olyan rendszer vagy hardver az operátor hálózati rétegében, mellyel a mediation réteg közvetlenül kapcsolatban áll, pl. Switch, MSC, GGSN vagy SGSN. Ezek a hálózati elemek a hálózat használatából eredő adatokat (Network Usage Data) saját, belső tárolóhelyükön helyezik el. Az így keletkezett fájlokat általában kollektorok gyűjtik össze.

Tágabb értelemben az *NE*, vagy *NETYPE* egy bizonyos típusú szolgáltatáshalmazt nyújtó, azonos interfésszel (gyártóval) rendelkező hálózati elem csoportot jelöl. Például a Nortel GGSN és a Huawei GGSN switchek két különböző NE, míg mind a húsz darab Nokia GSM MSC egy NE. Ez tehát egy logikai kategória. Ezek a hálózati elemek és a velük kapcsolatban lévők egy alhálózatot alkotnak.

A NEID (Network Element IDentifier) egy NE-n belüli konkrét hálózati elemet pl. az UKTCD13_GSM MSC-t jelöli.

A hálózat használatából eredő adatok alapegysége az *ER* (Event Record, esemény-rekord). Egy ER keletkezése egy hálózati elemen valamilyen eseményhez köthető, pl. híváskezdeményezés, cellaváltás, SMS üzenet küldése, stb. Az ER tartalmazza az adott esemény részletes paramétereit. Egy NETYPE többféle altípusú ER-t is generálhat.

A *CDR* (Call Detail Record, hívásrészletező rekord) egy olyan ER, ami egy híváshoz köthető.

Példa az ER- okra: A GSM alhálózatban MobileOriginatedCall (híváskezdeményezés), MobileTerminatedCall (hívásfogadás), CallForwarding (hívásátitányítás), Transit stb. rekordok keletkeznek. A GPRS hálózatok GGSN-jei és SGSN-jei SGSN-CDR-okat, GGSN-CDR-okat és Mobiliy-CDR-okat hoznak létre. Az SGSN szolgálja ki a GPRS kérést, a GGSN a Gateway az internet felé. Az M-CDR akkor keletkezik, ha a GPRS-session alatt az előfizető cellát vált és a session kezlését másik GGSN és SGSN vesz át.

A ER-ok *billing fájlokba* íródnak ki.

A mediation egy számlázási előfeldolgozó rendszer.

A *mediation réteg feladata* a billing fájlok összegyűjtése a hálózati elemekről (collection), azok *előfeldolgozása* (preprocessing) és *konvertálása* (conversion) a célrendszereknek megfelelő rekord- és fájlformátumokra (ami magában foglalja az információk értelmezését és az üzleti szabályoknak megfelelően továbbiak származtatását is), majd a keletkezett kimeneti billing fájlok továbbítása a megfelelő OSS/BSS rendszerek felé (delivery).

A *collection* során történnek a fájlszintű ellenőrzések (file level checks, pl. *duplikáció-ellenőrzés*).

Az *előfeldolgozás* és a *konverzió* rekord szinten történik.

Az *előfeldolgozás* legfontosabb lépései:

A mediation a *konverzió* előtt *validálást* (validation) is végez, ez lehetővé teszi a forrásrendszerek meghibásodásának vagy rossz konfigurálásának felismerését és a hibás rekordok eltvávolítását.

A *validálás* egy speciális formája a *rekord-duplikáció ellenőrzés* (record duplicate checking) valamely kulcs alapján. A duplikált rekordok eltávolításra kerülnek.

A *validálást* követi a *szűrés* (filtering), mely eltávolítja a felesleges (számlázásilag nem releváns) rekordokat.

A konverzió egy speciális formája az *aggregáció* (aggregation), mely azonos típusú rekordok meghatározott kulcs alapján történő egyesítése. Használják pl. hosszú hívások rész-rekordjainak (partial records) egyesítésére, vagy bizonyos mezőinek összegzésére.

Egy mobil operátor OSS/BSS rendszereinek száma meghaladja az ötvenet (alrendszerekkel együtt a százat is). Ezek az rendszerek más és más interfésszel rendelkeznek, mivel a teljes rendszert különböző gyártók különböző alkalmazásai alkotják. Speciális esetben egy OSS/BSS is lehet hálózati elem.

A mediaton feladata tehát a közvetítés (mediáció) megvalósítása a hálózati és az OSS/BSS réteg rendszerei, valamint az OSS/BSS réteg különböző rendszerei között. A mediation elrejti bemeneti interfészészeinek azon sajátosságait melyek a célrendszer számára nem szükségesek, így azok transzparensek lesznek. És fordítva, olyan kimeneti interfészeket implementál, melyek a célrendszerek igényeinek megfelelnek.

Egy adott NE (NETYPE) billing fájljait egy LoB dolgozza fel. A Line of Business angol meghatározása CDR processing workflow, azaz CDR feldolgozó munkafolyamat. Ilyen például az UKSMT, a T-Mobile-UK SMSC-iről érkező adatokat feldolgozó LoB.

3 A Comptel EventLink 4 és az iBMD

A Comptel Corporation nevű finn vállalat telekommunikációs szoftverek fejlesztésével foglalkozik. Komplex mediációs megoldása az EventLink. Jelenleg a 6-os verzió a legújabb, az iBMD-n a 4-es verzió fut. (Röviden csak EL4).

Az EL4 egy rendszerintegrációs platform mely alapvető alkalmazásokat és szolgáltatásokat (pl. API-kat és grafikus felhasználói felületet) nyújt a mediációs feladatok végrahajtásához. Lehetővé teszi, hogy a rendszerintegrátorok egy adott mobil operátor rendszereinek és üzletmenetének megfelelő, komplex üzleti logikát integráljanak a rendszerbe az EL4 által nyújtott programozói interfészeken és a rendszer átlkonfigurálásán keresztül.

A legfontosabb ilyen feladat a *rulesetek* implementálása, melyek egy-egy adott LoB-hoz tartozó üzleti logikát valósítanak meg. A rulesetek nyelve a S-Lang (kiejtve: szleng) melyen az üzleti szabályokat kell implementálni, valamint az inputleíró és az outputleíró nyelvek melyek a be- és kimeneti fájlok és rekordok formátumát írják le.

Maga az alkalmazásplatform (Application Platform) Perl és ksh szkriptek, valamint tetszőleges Oracle adatbázisbeli objektumok (tábla, nézet, tárolt eljárás, trigger stb.) használatával bővíthető, azaz terjeszthető ki.

Az *iBMD* (international Billing Mediation Device) a T-Mobile International, Európa különböző országaiban lévő mobil hálózatainak EL4 alkalmazásplatformra integrált számlázási mediációs rendszere. A következő országokban használják: Németország (DE), Nagy-Britannia (UK), Ausztria (AT), Csehország (CZ). Jelenleg összesen több mint 40 LoB (ruleset) fut a rendszeren.

Az iBMD nem csak ruleseteket implementál. A billing fájlok kezelését és nyilvántartását, a fájlátvitel lehetőségeinek kibővítését és annak validálását (confirmation), a riportokhoz szükséges adatok tárolását és feldolgozását, fájlok újrafeldolgozását (reprocessing), valamint számos más funkciót támogatandó, az iBMD körülbelül duplájára bővíti ki az alap EL4 alkalmazásplatformot

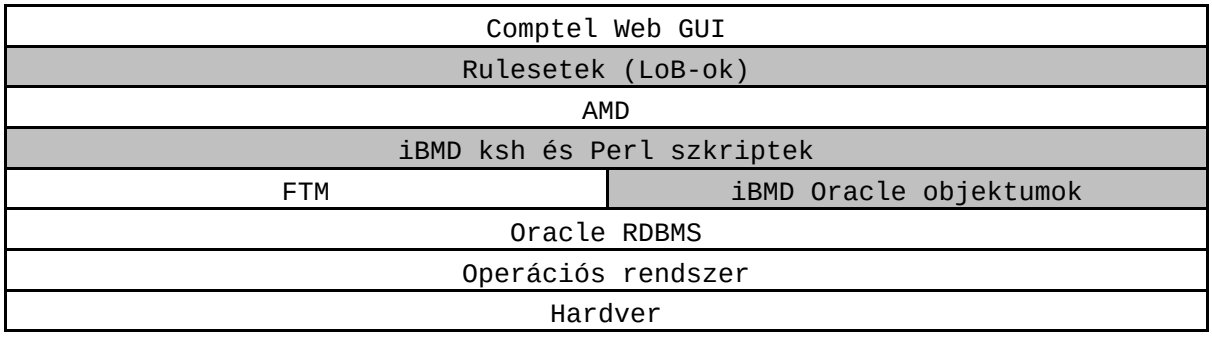
3.1 Az iBMD architektúrája

[5] Az EL4-ben a legkisebb végrehajtási egység a *step*. A step magja egy tetszőleges UNIX shell parancs, pl. egy alkalmazás hívása parancssori argumentumokkal. A stepeket az FTCD felügyeli. Visszatérési értékük a UNIX alapú rendszereknél szokásos. A stepek többek között UNIX környezeti változókkal kommunikálhatnak az adott methodon belül utánuk következő stepek felé.

A stepek valamely *template*-ben szerepelnek, mely mintaként meghatározza a stepek végrehajtási sorrendjét (chain of steps), valamint tartalmazhat feltételes elágazásokat is. A templateknek vannak ún. templateparaméterei. A template nem futtatható.

A *methodok* valamely template-ből származnak és öröklik annak paramétereit és értékeit. Ezek a parméterek a methodparaméterek. Egy template-ből több method is származtatható, más-más néven. Ha egy methodparaméter értéke nincs megadva, az örökölt templateparaméter értéke lesz érvényes. A templateparaméter beállításaiban szabályozhatjuk azt is, kötelező-e az azonos nevű methodparaméternek értéket adni. A methodparaméterekkel szabályozhatjuk a methodok viselkedését amennyiben az adott methodparaméter értelmezve van a method stepjeinek kódjában (ott UNIX környezeti változóként jeleni meg). A leggyakoribb methodok a COLLECTION, CONVERSION, DELIVERY, REPROCESSING, CONFIRMATION.

Az iBMD-ben a methodok is összefűzhetőek, ha egymást automatikusan indítják FTM hívásokon keresztül.



2. ábra Az IBMD architektúrája

Az iBMD architektúráját a 2. ábra mutatja be.

Az *FTM* (File Transfer Manager) az EL4 központi alkalmazása. Alrendszerei segítségével ez irányítja a tljes rendszer működését: környezetet nyújt a stepek (FTCD) és methodok (FTM) futtatásához és ellenőrzi azok státuszát és kilépő kódját (Step Controlling , Method Managing), feladatokat ütemez (Scheduler), összegyűjti a logfájlokhoz szükséges információkat és az adatbázisban tárolja azokat valamint statisztikákat szolgáltat a rendszer állapotáról és a feldolgozott fájlokról (Audit Collector), hiba esetén risztásokat indít el (Alarm Dispatcher).

Az AMD (Account Mediation Device) részei:

- Fájlátvitelt megvalósító alkalmazások (collection , delivery) (FTP, FTAM)
- Beolvasó alkalmazások (FileReader, XMLReader, LogReader)
- CV (Conversion&Validation) A rendszer lényegét adó alkalmazás. Ez végzi a fájlokban lévő rekordok előfeldogozását és konverzióját. Segédalkalmazásai:
    - Lookup Server: Oracle alapú, rulesettől független paraméterek tárolására szolgáló adatbázis. Lookuptáblákat tartalmaz, pl. az országhívószámokat tartalmazó lookuptábla. A LoB- ok ezek segítségével paraméterezhetőek a ruleset megváltoztatása nélkül.
    - Event aggregator: Az aggregációt valósítja meg perzisztens módon.
    - Record Dulpicate Checker: Duplikált rekordok eltávolítására használható.
- Kiíró alkalmazás (FileWriter)

4 Formális jelölésrendszer

Az ASN.1 nyelv-részhalmaz illetve az FR inputleíró nyelv szintakszisának formális leírásához az alábbi jelölésrendszert használom:

Terminális: a nyelv kulcsszavai Courier New betűtípussal, például `EXPLICIT`

Nem terminális: kisbetűs kategórianevek dőlt betűvel szedve, például *tag*

Alternatíva: {a1 | ... | an} , például {`EXPLICIT` | `IMPLICIT`}

Opció: [ ], például [`SIZE`]

Iteráció: ..., az előtte álló szintaktikai elem akárhányszoros ismétlődését jelenti

A szintaktikai szabályok bal oldalán egy nem terminális áll, jobb oldalán pedig egy tetszőleges elemsorozat.A két oldalt kettőspont választja el. Ha a formális leíró karakterek a nyelvnek részei akkor azokat félkövér betűvel szedve jelölöm.

5 Az ASN.1 leíró nyelv és a BER kódolás

Napjaink egyik legkomplexebb rendszere, mely lehetőséget biztosít magasszintű absztrakcióra, az OSI (Open Systems Interconnection – Nyílt Rendszerek Összekapcsolása). Az OSI egy nemzetközi szinten szabványosított architektúra, mely meghatározza a számítógépek és az általuk alkotott rendszerek összekapcsolását egészen a fizikai rétegtől az alkalmazás rétegig.

Ebben az architektúrában a rétegek funkciói és szolgáltatásai absztrakt módon vannak definiálva. Minden réteg csak a szomszédos rétegekkel kommunikál egy jól definiált interfészen keresztül, mely meghatározza, hogy az adott réteg milyen szolgáltatásokat vár el az alatta lévő rétegtől és melyek azok melyeket a felsőbb rétegnek nyújt. Így egy réteget, vagy annak valamely szolgáltatását megvalósító implementációnak csak az adott réteg és a hozzá tartozó interfészek specifikációit kell ismernie. A nem szomszédos rétegek ily módon egymás számára nem láthatóak, azaz transzparensek.

Az OSI azért *nyílt rendszer*, mert minden réteg szolgáltatásainak számos különböző implementációját támogatja.

Az OSI az absztrakt adatstruktúrákat az ASN.1 (Abstract Syntax Notation One, [9]) leíró nyelv segítségével adja meg. Az ASN.1 által leírt struktúrák bináris reprezentációját különböző kódolási szabályrendszerek adják meg. Ilyen például a Basic Encoding Rules (BER), vagy a Distinguished Encoding Rules (DER).

A mobil operátorok rendszerei által a BER a leggyakrabban használt TLV (Type, Length, Value) kódolás, eddigi munkám során a TLV kódolások közül is csak ezzel találkoztam, ezért a későbbiekben csak ezzel foglalkozok majd. Nagy előnye a BER-nek, hogy az információkat a lehető legtömörebb formában kódolja. Az ASN.1 nyelv jellegzetességei más kódolások esetén is a BER-éhez hasonlóan mutatkoznak meg.

Az ASN.1 és a BER a telekommunikációban és a számítógépes hálózatokban széles körben elterjedt, hosszú múlttal rendekező és ezáltal stabil eszközök, melyeket a következő szabványok specifikálnak: [16], [17].

A 7. fejezetben ismertetett általam írt alkalmazás, az *ASN1BERInputDescGenerator* az ASN.1 nyelvnek egy részhalmazát támogatja, egészen pontosan azt a részhalmazt, mellyel az iBMD-n való rendszerintegrátori munkánk során a gyakorlatban találkozunk. Ebben a fejezetben ezt az ASN.1 nyelv-részhalmazt és a BER kódolási szabályrendszert ismertetem. Ahol az ASN.1 nyelv ezen túli – általunk nem, vagy csak ritkán használt - lehetőségeiről írok, ott ezt külön jelölni fogom (NI) formában.

5.1 ASN.1

Ebben a fejezetben az *Abstract Syntax Notation One* [1] [9], vagy röviden ASN.1 absztrakt leírónyelvet ismertetem.

Egy (vagy több különböző) típusú összefüggő adatstruktúrát (vagy struktúrákat), például egy TAP03 típusú CDR-t (vagy a különböző GGSN CDR-okat) a hozzá(juk) tartozó ASN.1 *specifikáció* írja le, mely egy szöveges fájl.

A nyelv a kis- és a nagybetűket különbözteti. A megjegyzések a `--` karakterekkel kezdődnek és a sor végéig tartanak.

Egy specifikáció általános felépítése:

specifikációnév `DEFINITIONS` {`EXPLICIT` | `IMPLICIT`} `TAGS ::=`
`BEGIN`
ASN.1_elem_definíció ...
`END`

Az {`EXPLICIT` | `IMPLICIT`} az alapértelmezett taggelést határozza meg.

(NI) Az explicit tagginget az alkalmazás nem támogatja. Lásd a 7. fejezetet.

Egy *ASN.1_elem_definíció* általános felépítése:

név `::=` [*tag*] *típus*
[**{** *blokk_definíció* **}**]

típus: {*ASN.1_alaptípus* | *definiált_típus_neve* }

A *definiált_típus_neve* egy típusnév ami nem más, mint egy másik ASN.1 elem (definiált típus) neve. Ez esetben a fentiekben leírt elem származtatott típusú lesz, mivel típusát a másik elemből (definiált típusból) származtatjuk. Származtatott típusú elemek definíciója nem tartalmazhat *blokk_definíciót*.

A tag felépítése:

[**[** *osztály azonosító***]**] [{`EXPLICIT` | `IMPLICIT`}]

A név az ASN.1 elem neve. Az osztály és az azonosító (pozitív egész szám), valamint a taggelést meghatározó kulcsszó (mely az alapértelmezést bírálhatja felül) - amennyiben szerepelnek - együtt képezik a taget. Az első kettő azonosítja az ASN.1 elemet az utóbbi pedig meghatározza viselkedését az ASN.1 elemekből építhető fában.

5.1.1 ASN.1 osztályok

Az ASN.1 osztályok a tageket osztályozzák.

Az ASN.1 elemeket (típusokat) ugyanis nem a nevük (típusnév) teszi egyedivé és absztrakttá, a név csak a specifikáción belül egyedi. Ezt a szerepet a tag tölti be, mely a tag osztálya által meghatározott szinten teszi egyedivé az elemet.

Az ASN.1 négy osztályt határoz meg:

(NI) `UNIVERSAL`: Olyan típusokat jelölő tagek, melyek minden alkalmazásban ugyanazt jelentik. Ezek a típusok csak az X.608 [16] specifikációban vannak meghatározva. Az ASN.1 alaptípusok is rendelkeznek ilyen alapértelmezett, `UNIVERSAL` osztálybeli taggel, melyeket pszeudotageknek hívunk.

`APPLICATION`: Olyan típusokat jelölő tagek, melyek egy adott alkalmazáson belül egyediek. Más-más alkalmazásokban ugyanaz a tag más-más típust jelölhet.

`PRIVATE`: Olyan típusokat jelölő tagek, melyek egy adott alkalmazáscsoporton (pl. egy adott cég alkalmazásain) belül egyediek.

(NI) `Context-specific` (környezetfüggő): Olyan tageket jelöl, melyek az előző három osztálytól elérően csak egy adott összetett típuson belül egyediek, de a specifikációban nem. Egy ilyen tag jelentése a környezetétől függ, azaz attól, hogy melyik összetett típuson belül fordul elő és csak az azon belüli gyermek-elemeket különbözteti meg egymástól. Ez a többi három osztályétól eltérő szintakszist eredményez: a tagek a *blokk_definíción* belül helyezkednek el. Például:

```
ERecord ::= CHOICE
{
    chargeInformationRecord    [22] ChargeInformationRecord,
    taxChargeInformationRecord [45] TaxChargeInformationRecord
}

ChargeInformationRecord ::= SEQUENCE
{
    chargedItem         [0] ChargedItem         OPTIONAL,
    exchangeValue       [1] ExchangeValue    OPTIONAL,
    callType            [2] CallType            OPTIONAL,
    callTypeSubtype     [3] CallTypeSubtype     OPTIONAL
}

TaxChargeInformationRecord ::= SEQUENCE
{
    chargedItem         [0] ChargedItem         OPTIONAL,
    exchangeValue       [1] ExchangeValue    OPTIONAL,
    callType            [2] CallType            OPTIONAL,
    callTypeSubtype     [3] CallTypeSubtype     OPTIONAL,
    taxInformation      [4] TaxInformationList
}
```

Alkalmazható együtt más osztálybeli tagekkel is:

```
PrivateKeyInfo ::= [PRIVATE 122] SEQUENCE
{
  version Version,
  privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,
  privateKey PrivateKey,
  attributes [0] IMPLICIT Attributes OPTIONAL
}
```

Általában - az `UNIVERSAL` osztálybeli pszeudotagektől eltekintve - egy specifikáción belül csak egyféle osztályba tartozó tageket használnak.

5.1.2 ASN.1 típusok

Az ASN.1-ben a következő típusosztályok léteznek:

- *ASN.1 alaptípusok (ASN.1 base types)*, más néven *erős típusok:*
    - *Primitív típusok (simple tpyes)*
    - *Összetett típusok (constructed types)*
- *gyenge típusok (weak types)*
    - *definiált típusok (defined types)*
        - *származtatott típusok (derived types)*

Az *ASN.1 alaptípusok* az ASN.1 nyelv által előre definiált típusok.

A *primitív típusok* atomiak. Az alkalmazás által támogatott primitívek: `INTEGER`, `OCTETSTRING`, `REAL`, `Boolean`, `ENUMERATED`, `BitString`. A közvetetten támogatottak (lásd az alkalmazást bemutató rész végét, közvetetten valamennyi primitív ASN.1 típus támogatott) közül a legfontosabbak: `IA5String`, `NULL`, `OBJECT IDENTIFIER`, `PrintableString`, `T61String`, `UTCTime`.

Ezek közül a `BitString` és az `ENUMERATED` esetén a *blokk_definíció* kötelező, a többinél nem szerepelhet.

Maguk a primitív típusok a BER kódolásban általában nem kapnak taget (a kivételeket lásd lentebb), a primitív típusú ASN.1 elemek (lásd lentebb) viszont általában igen.

(NI) (KEC) Az explicit taggel rendelkező primitív típusú ASN.1 elemek és a (context-specific osztályú taggel rendelkező) `SEQUENCE OF` vagy `SET OF` típusú elemek listájának elemei (ha a lista elemei primitív típusúak) egy `UNIVERSAL` osztályú ún. pszeudotaget kapnak. Ez a tag magától a primitív típustól függ, például `INTEGER` ASN.1 típus esetén 0x02 és magát a primitív típust azonosítja. A pszeudotageket illetően lásd még a (PTCS) pontot.

A fenti kivételektől eltekintve a primitív típusok csak valamely ASN.1 elem típusaként jelenhetnek meg. Az ASN.1 elemek maguk is gyenge típusok, ún. *definiált típusok*.

Néhány példa a primitív típusú ASN.1 elemek definíciójára:

```
AiurRequested ::= [APPLICATION 34] INTEGER

Flags :: [APPLICATION 35] BitString
{
flag0 (0),
flag1 (1),
flag2 (2)
}

Colors ::= [PRIVATE 77] ENUMERATED
{
black (0),
red (1),
white (2)
}
```

Az összetett típusok más ASN.1 elemeket is tartalmaznak.

A SEQUENCE OF és a SET OF típusú elemek általános alakja:

név ::= [*tag*] {SEQUENCE OF|SET OF} *egyedüli_gyermek_típusneve*

Az ilyen típusú elemek egy olyan listát definiálnak, melyen a tagok típusa azonos. A lista tagjainak típusát az *egyedüli_gyermek_típusneve* adja meg. Ez a típusnév megegyezik egy ASN.1 elem (definiált típus) nevével (ezt a definiált típust hivatkozza). Ilyen elemekből fog állni a lista. Egy példa:

```
BasicServiceCodeList ::= [APPLICATION 37] SEQUENCE OF BasicServiceCode

BasicServiceCode ::= CHOICE
{
    teleServiceCode      TeleServiceCode,
    bearerServiceCode    BearerServiceCode
}
```

A (KEC) pontban leírt listákat az alkalmazás nem támogatja.

(NI) A SET OF annyiban tér el a SEQUENCE OF-tól, hogy elemei nem rendezettek. BER kódolású fájlokban az listák mindig rendezettek, ezért nincs implementálva.

A CHOICE, SEQUENCE, SET típusú elemek általános alakja:

név ::= [*tag*] {CHOICE | SEQUENCE | SET }
{
gyermek_elem_neve gyermek_elem_típusneve [OPTIONAL]
[, *gyermek_elem_neve gyermek_elem_típusneve* [OPTIONAL]] …
}

A *gyermek_elem_neve* egy egyedi azonosító, a *gyermek_elem_típusneve* egy, a specifikációban definiált elem (azaz egy definiált típus) neve lehet. Innen ered a típusnév elnevezés is. A blokkon belül csak különböző nevű és típusú gyermek szerepelhetnek.

(NI) Context-specific osztályú elemeket tartalmazó specifikációkban a *gyermek_elem_típusneve* helyett szerepelhet bármilyen ASN.1 alaptípus neve is.

A CHOICE egy uniót definiál. A gyermek-elemek közül pontosan egy szerepelhet a BER kódolású fájlban a CHOICE típusú elem gyermekeként. A CHOICE definíciójában az OPTIONAL kulcsszó nem fordulhat elő.

(NI) (PTCS) Context-specific osztályú elemeket tartalmazó specifikációkban a CHOICE automatikusan explicit taggel rendelkezőként viselkedik akkor is, ha implicitként van megjelölve (!). Ebben az esetben ha van saját tagje akkor azzal jelenik meg a BER kódolásban, ha nincs, akkor alapértelmezett pszeudotagje 0x30 lesz. Ez utóbbira jó példa ha egy SEQUENCE OF vagy SET OF típusú elem gyermekeként szerepel. Például:

```
T1 :: SET
{
a     [1]   A,
soc   [2]   SEQUENCE OF T2
}

T2 :: CHOICE
{
a     [1]   A,
soc   [2]   SOC
}
```

A SEQUENCE különböző típusú elemek rendezett kollekciója. A gyermek-elemek a BER kódolásban csak a megadott sorrendben fordulhatnak elő. Az OPTIONAL kulcsszóval jelölt

gyermek-elemek a kódolásból elhagyhatóak, de legalább egynek mindenképp szerepelnie kell, és egy elem csak egyszer szerepelhet.

A `SET` annyiban tér el a `SEQUENCE`-től, hogy nem rendezett, az elemek előfordulási sorrendje tetszőleges.

(NI) Az `ANY` típus definiálatlan, a jövőben meghatározandó típust jelöl. Absztrakt és befejezetlen volta miatt a BER esetén nem értelmezhető.

A *definiált típusok* maguk az ASN.1 elemek. Ezek az elemek így típusként is viselkednek. Egy ASN.1 elem típusa lehet ASN.1 alaptípus, vagy származtatott típus.

A származtatott típusú elemek (*származtatott típusok*) a definiált típusok egy részhalmazát képezik.Olyan típusok, melyek valamely másik típus értelmezési tartományát vagy alkalmazásának körét szűkitik le további információ megadásával. A származtatott típusok *származtatott típusokból álló láncot* képeznek. A láncok legvégén mindig valamilyen ASN.1 alaptípus szerepel. Például:

```
PhoneNumber ::= [APPLICATION 72] IMPLICIT PositiveAsciiNumber
PositiveAsciiNumber ::= [APPLICATION 71] IMPLICIT AsciiNumber
AsciiNumber ::= [APPLICATION 70] IMPLICIT OCTET STRING
```

Az `AsciiNumber`, `PositiveAsciiNumber` és a `PhoneNumber` ASN.1 elemek, melyek egyben (definiált) típusok is.

Az `AsciiNumber` típusa az `OCTET STRING` ASN.1 alaptípus. A `PositiveAsciiNumber` elem származtatott típusú, mert típusát egy másik *definiált típus*, az `AsciiNumber` értelmezési tartományának leszűkítésével kapja (*származtatja*).

Habár az AsciiNumber is szűkíti az `OCTET STRING` alaptípus értelmezési tartományát, típusa mégsem *származtatott típus*, mert nem *definiált típust* szűkít hanem alaptípust. Ahol mégis ki szeretnénk emelni ezt a tulajdonágát, ott *közvetlenül alaptípusból származtatott típusnak* hívjuk. Az ASN.1 maga is meghatároz ilyen típusokat melyek az idők folyamán alaptípusként kerültek be a szabványba.

A `PhoneNumber` is egy származtatott típus, mely a `PositiveAsciiNumber` származtatott típus értelemezési tartományát szűkíti tovább.

A származtatott típusokról lásd még az 5.2 fejezetet.

5.1.3 Implicit tagging és az explicit tagging

A tagging az elemek tagjeit osztályozza és ezzel együtt meghatározza viselkedésüket.

Viselkedés alatt azt értjük, hogy a *származtatott típusokból álló láncban* hogyan viszonyulnak az alattuk lévő elemekhez valamint azok tagjeihez, és hogy ezt figyelembe véve mely tagek jelennek meg a (BER) kódolásban.

Egy tag lehet implicit vagy explicit.

Az implicit taggel rendelkező elemek tagjei közül a láncból csak a legfelső szinten lévő tagje jelenik meg, mintegy helyettesítve az alatta lévő bővebb értelemezési tartományú elemek tagjeit.
Implicit tagging: Valamely ASN.1 kódolásban (pl. a BER) egy implicit taggel ellátott típus egy olyan, más típusból származtatott típus ahol az alant lévő típushoz (amiből származtatunk) képest való megkülönböztetést az alant lévő típus tagjének *elhagyása* mellett egy új tag bevezetése biztosítja.

Az explicit taggel rendelkező elemek tagjei mindenképp megjelennek a kódolásban, beleértve a primitív típusú elemek (és esetenként a CHOICE típusúak, lásd a (PTCS) pontot) pszeudotagjeit is.
(NI) *Explicit tagging:* Valamely ASN.1 kódolásban (pl. a BER) egy *explicit taggel* ellátott típus egy olyan, más típusból származtatott típus ahol az alant lévő típushoz (amiből származtatunk) képest való megkülönböztetést az alant lévő típus tagjének *megtartása* mellett egy *új külső tag* bevezetése biztosítja.

A taggingről és annak implementálásáról további részletek találhatóak a 7. fejezetben, az alkalmazás ismertetésénél.

A kétféle tagging keverhető, azaz egy specifikáció tartalmazhat mind explicit, mind implicit taggel rendelkező elemeket, bár ez elég ritka. Ekkor értelemszerűen minden elemre a tagjének megfelelő tagginget kell alkalmazni.

5.2 BER kódolás

Az *ASN.1 Basic Encoding Rules* (Egyszerű Kódolási Szabályrendszer), röviden BER kódolás meghatározza, hogyan kell az ASN.1 típusok konkrét értékekkel rendelkező példányait binárisan kódolni. A BER valójában oktetsztringekre képezi le ezeket a példányokat, melyek aztán bináris formában íródnak ki az ASN.1 BER kódolású fájlba (egy oktet → egy bájt, például 03 → 0x03).

Az OSI a BER-t határozza meg standardként az ASN.1 példányok binárisra való leképezésére.

A BER egy TLV kódolás, azaz minden BER-beli elem három részből áll: típus, hossz, érték.

Egy ASN.1 példány három módszerrel kódolható. Ez függ a típustól, az értéktől, valamint hogy az érték hossza ismert e. Ezek a kódolási módszerek és alkalmazásaik:

- *Primitív, határozott hosszú (primitive, definite-length)* : Egyszerű típusok, valamint ezekből implicit tagginggel származtatott típusok kódolására használjuk. A hosszt a példány kódolásának kezdetekor ismerni kell.

- *Összetett, határozott hosszú (constructed, definite length)* : Egyszerű sztring típusok, összetett típusok, valamint mindezekből implicit tagginggel származtatott típusok, bármely típusból explicit tagginggel származtatott típusok típusok kódolására használjuk. A hosszt a példány kódolásának kezdetekor ismerni kell.

- *Összetett, határozatlan hosszú (constructed, indefinite length)* : Az előzővel azonos típusok kódolására használjuk. A hosszt a példány kódolásának kezdetekor nem ismerjük.

Implicit tagginggel származtatott típusok az alant lévő típus módszerét, az explicit tagginggel származtatott típusok valamelyik összetett módszert alkalmazzák.

5.2.1 Típus

A BER típust a BER tag határozza meg és a következő információkat kódolja: az ASN.1 tag osztálya, azonosítója (tag number), valamint hogy az alkalmazott módszer primitív vagy öszetett e.
Kétféleképpen kódolható:

- *Ha a tag number 0 és 30 között van*, akkor egy oktet hosszú:
    - 8-7 bitek: osztály:
        - universal : 00
        - application : 01
        - context-specific : 10
        - private : 11
    - 6. bit: 0 – primitív módszer, 1 – összetett módszer
    - 5-1 bitek: tag number.
- *Ha a tag number 31 felett van*, akkor kettő vagy több oktet hosszú:
    - első oktet : ugyanaz mint fent, kivéve, hogy az 5-1 bitek értéke 1111
    - a második és további oktetek tartalmazzák a tag number-t (azonosító) 128-as alapon, így az utolsó oktetet kivéve valamennyi 8. bitje mindig 1.

5.2.2 Hossz

- *Határozott hossz*: ez adja meg az érték oktetjeinek számát.Két formája van.
    - *Rövid forma* (ha az érték hossza 0 és 127 között van) : Egy oktet. A 8. bit éréke 0, a 7-1 bitek adják meg a hosszat
    - *Hosszú forma* (ha az érték hossza 0 és 2^{1008}-1 között van) : A hossz kódolása 2-127 oktet hosszú.
        - első oktet:
            - 8.bit: 1
            - 7-1 bitek: az érték hosszát megadó oktetek száma
        - a második és további oktetek tartalmazzák a hosszat

- *Határozatlan hossz*: Egy oktet hosszan van megadva a 0x80 értékkel. Az érték-oktetek kiírásának végén vége-jelet kell alkalmazni ami a két oktet hosszú 0x0000. (Valójában ez egy universal osztályú 0 azonosítójú 0 hosszú pszeudoelem primitív, határozott hosszú kódolása)

5.2.3 Érték-oktetek

A primitív, határozott hosszú módszernél a példány értékének konkrét reprezentációját adja meg.

Összetett módszereknél a komponens példányok BER kódolásának konkatenációja.

6 Az EventLink 4 beolvasást végző alkalmazásai

Az XML fájlok beolvasására az EL4 az *XMLReader*-t használja.

LogReader bármilyen bináris vagy szöveges fájl beolvasására alkalmas. Lassúsága miatt a gyakorlatban csak CSV és hasonló formátumú szöveges fájlok beolvasására használják, mivel a hosszinformációk hiánya miatt ezek beolvasására az FR nem képes. A fejlesztőnek egy Perl rulesetet kell implementálnia az adott fáljltípus beolvasására.

A beolvasó alkalmazások részletes ismertetése megtalálható az alábbi dokumentumban: [3].

6.1 FileReader

A FileReader, röviden FR a Comptel EventLink egyik fontos alkalmazása, mely tetszőleges formátumú bináris – köztük ASN.1 BER kódolású - fájlok beolvasására szolgál.

A leggyakoribb fájlformátumok: fix mező és rekordhosszú formátum, részben fix mezőhosszú formátum esetenkénti mező- és/vagy blokk-hossz kódolásokkal, részben fix mezőhosszú formátum a rekordhossz megadásával és a rekord végén lévő opcionális vagy változó hosszú mezőkkel, BER.

Saját – a billing fájlok struktúráját meghatározó - leíró nyelvvel rendelkezik, melynek neve *inputleíró*. A beolvasott adatokat egy *message API* nevű belső formátumban tárolja, melyet a többi EL alkalmazás is olvasni tud.

Az *inputleíró* nyelv a kis- és nagybetűket megkülönbözteti. A megjegyzéseket a sor első karaktereként használt # karakter jelöli és a sor végéig tartanak.

Nem ismertetem a nyelv azon elemeit, melyeket a gyakorlatban nem használunk, vagy az alkalmazás szempontjából nem lényegesek. Ezek a [1] [16] helyeken találhatóak részletesen leírva.

A nyelv formális leírása és annak rövid magyarázata:

[*direktívák*]
blokk_típusú_elem

direktívák:
[*d_bájtsorrend*]
[*d_teljesen_struktúrált_nevek*]
[*d_include*]…

d_bájtsorrend: {`#byteorder MSB/LSB`|`#byteorder LSB/MSB`}
d_teljesen_struktúrált_nevek: `#fully_structured_names`
d_include: `include` ’*inputleíró_fájlnév*’

blokk_típusú_elem:
{
`Set` [*elemszám*] *blokk_definíció*
|
[**(**]`Block`[**)**] [{`+`|`.`}]*blokknév* *id* *hossz* *recovery_level*
blokk_definíció
}

blokk_definíció:
{
{ *blokk_típusú_elem* | *mező* } …
}

mező:
[**(**]`Field`[**)**] *mezőnév*[**(***referencia_név***)**] *id* *hossz* *recovery_level* *mező_típus*

Ha a `Field` vagy `Block` zárójelben van, az FR nem olvassa be ténylegesen, hanem átugorja. A *mezőnév* egy azonosító. A *referencia_név* megadásával egy *bin* típusú mező értéke hivatkozható lesz a leíró további részében.

Az *id* és a *hossz* egész típusú kifejezések melyek megadják az elem egyedi, bináris azonosítóját és hosszát. Pl. 0x23 8

A `*`*offszet*`:`*hossz* formában hivatkozhatnak egy pozitív relatív offszetre és hosszra a bináris fájlban, ahol ezek az információk megtalálhatóak pozitív egészként. Vagy a `Ref`**[***referencia_név***]** formában egy *referencia_névre*. A kapott értékekkel elemi aritmetikai

műveletek végezhetőek. Ha nincs ilyen információ akkor azt a `*` jelöli. BER esetén `BER` –t kell megadni mindkét helyen.

A *mező_típus* a következők valamelyike lehet : `ASCII` (nyomtatható ascii karakterek), `EBCDIC`, `BCD`, `BCDR`, `BCDS`, `BIN` (pozitív egész), `OCTSTR` (oktetsztring), `OCTSTRR`, `OCTSTRS`.

A `Block` tetszőleges elemeket, azaz `Block` - ot, `Set` – et, vagy `Field`-et tartalmazhat. A blokknév előtt lévő `+` arra utal, hogy azt a CV-ben *rekordként* fog megjelenni, a `.` pedig arra, hogy *alrekordként*.

A `Block`-ok, különösen a rekordok hosszát vagy meg kell adni, vagy a tartalmazott elemek hosszából kiszámítható kell, hogy legyen. Az FR akkor képes kezelni egy indefinit hosszú mezőt, ha annak van `id`-je és egy ismert hosszú blokk utolsó eleme.

A `Set`-ben minden elem opcionális és bárhányszor előfordulhat. Ha meg van adva az *elemszám*, mely vagy egy pozitív egész, vagy egy `referencia_név` lehet, akkor legfeljebb annyi darab előfordulás megengedett. A `Set`-ben minden elemnek kell legyen `id`-je. `Set` közvetlenül egy `Set`-en belül nem fordulhat elő.

7 ASN1BERInputDescGenerator

A bevezetésben hivatkozott, roaming adatokat feldolgozó LoB-ot, a TMA_TAP03 – at és bemenetét (ASN.1 BER kódolású TAP03 fájl) a következő dokumentumok írják le [6] [7] [8].

Az alkalmazás célja ASN.1 absztrakt jelölő nyelven írt specifikációk értelmezése - beleértve a beolvasást, szintaktikai és szemantikai elemzést -, a beolvasott ASN.1 elemekből egy fa felépítése, a fa további ellenőrzése és átalakítása az ASN.1 és a FileReader inputleíró nyelveknek megfelelően, majd a fa segítségével egy FileReader inputleíró fájl generálása. Az így kapott inputleíró fájlt felhasználva a FileReader-nek képesnek kell lennie beolvasni bármilyen, a megadott ASN.1 specifikációnak megfelelő BER kódolású fájlt.

Nem célja az alkalmazásnak a teljes ASN.1 nyelv támogatása. Általános célú ASN.1 szintaktikai elemzők és egyéb eszközök már nagy számban rendelkezésre állnak mind ingyenes, mind megvásárolható formában. Csak azokra a részekre/feladatokra koncentráltam, melyek az iBMD projecten való munkavégzés során a gyakorlatban előfordulnak, és amely problémára eddig – legjobb tudomásom szerint - nem született azt megoldó komplex, publikus alkalmazás. Az alkalmazás hiánypótló jellegű, ami véleményem szerint jól ellensúlyozza azt, hogy nem általános célú, hiszen egy jól meghatározott speciális feladat megoldására szolgál. Számos szakmai szempontot vettem figyelembe annak meghatározására, az ASN.1 nyelv mely elemeit szükséges vagy nem szükséges támogatnia az alkalmazásnak.

Mivel az iBMD a TLV kódolásúakat tekintve kizárólag ASN.1 BER kódolású fájlokat használ, csak az ilyen fájlokat leíró ASN.1 specifikációkat vettem figyelembe.

- Nem támogatja az alkalmazás az egy szintnél mélyebben egymásba ágyazott ASN.1 elemeket leíró szintakszist.

- Az ASN.1 elemek (típusok) definícióján túl az ASN.1 nyelvben lehetőség van arra, hogy az adott típussal rendelkező rendelkező konkrét példányokat deifináljunk. Ez BER kódolás esetén kizárt, hiszen a példányokat (tényleges információt) a BER kódolású fájlt tartalmazza.

- Nem támogatja az alkalmazás a `Context-specific` és az `UNIVERSAL` ASN.1 osztályokat (ezekkel később bővíthető)

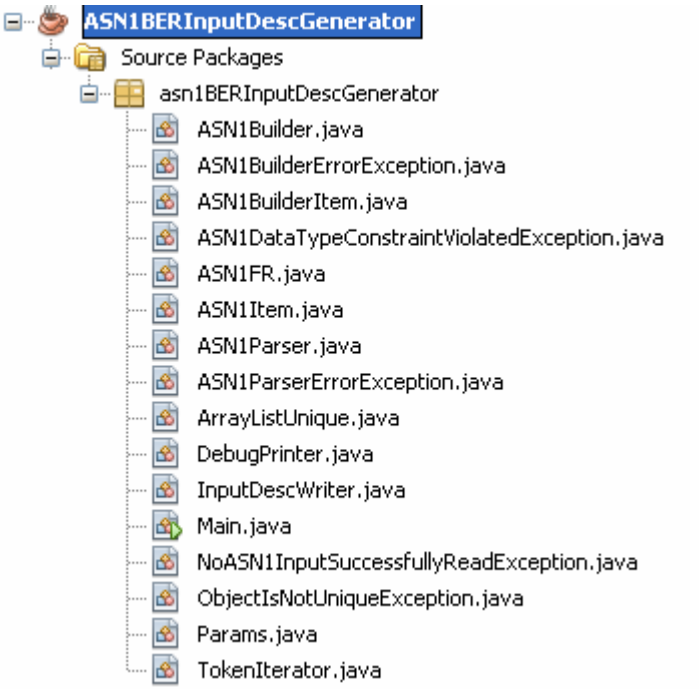
- ASN.1 elem gyermekeinek felsorolása esetén a gyermek típusaként (közvetlenül) ASN.1 alaptípus nem adható meg, - mivel az szoros összefüggésben van a `Context-specific` illetve az `UNIVERSAL` ASN.1 osztályokkal melyek nem kerültek implementálásra. Csak nem-ASN.1 alaptípus neve használható.

- Nem támogatja az alkalmazás az explicit tagginget. Csak az implicit tagginget támogatja. Ilyenkor egymásra láncszerűen hivatkozó ASN.1 elemek (származtatott típusok) közül csak a legfelső szinten lévő elem fog szerepelni a BER kódolásban, a közbensők (beleértve a legalsó szinten lévőt) nem. Ez utóbbiak explicit tagging esetén redundánsak lennének, hiszen a láncban a legfelső szinten lévő elem tagje (néhány ritka kivételtől eltekintve) egyértelműen meghatározza az alatta lévőket is. Ily módon, ha pl. a lánc 4 hosszú, implicit tagging esetén elég egy elemet kódolni BER-ben, míg explicit esetben mind a 4-nek szerepelnie kell egymást tartalmazva. Ez utóbbiak feleslegesen növelnék a billing fájl méretét, ezért telekommunikációs hálózatokban az explicit tagginget nem, vagy csak nagyon ritkán alkalmazzák.

- Az `ANY` típus definiálatlan, a jövőben meghatározandó típust jelöl. Absztrakt és befejezetlen volta miatt a BER esetén nem értelmezhető. A `SET OF` egy olyan, azonos típusú elemekből álló listát határoz meg, melyen az elemek sorrendje tetszőleges. Ilyen ’laza’ típusú elemek a billing fájlokban nem megengedettek és nem is szükségesek, a lista elemeinek sorrendje mindig fontos. Ezért ezek a típusok nincsenek implementálva. A típusokat illető egyéb megszorításokkal kapcsolatban lásd még a következő fejezet `InputDescWriter` - t tárgyaló részét.

Az implementáció nyelvének a Java-t választottam. Fontos megemlíteni még a bővíthetőséget is. Ahol csak lehetőségét láttam, igyekeztem útmutatást, ötleteket felvázolni az alkalmazás bővítését illetően.

7.1 Az alkalmazás ismertetése



3. ábra Az alkalmazás osztályai

Az `ASN1BERInputDescGenerator` alkalmazás egyetlen Java csomagból áll, melynek osztályait (lásd a 3. ábrán) az alkalmazás működésén keresztül mutatom be. A program két parancssori argumentumot vár el, az első a feldolgozandó ASN.1 specifikációt tartalmazó szöveges fájl neve, a második pedig az ebből készítendő új, FileReader inputleíró fájl neve.

Első lépés az ASN.1 fájl *beolvasása* és *tokenekre bontása*. Ezekért két osztály felelős: a `TokenIterator` és az `ASN.1FR`.

A `TokenIterator` absztrakt osztály az Iterator interfészt implementálja egy tetszőleges nyelven írt szöveges fájlban lévő tokenek felett. Metódusai között csak egy van, ami absztrakt, a `read()`, mely a fájl beolvasását és tokenizálását az őt kiterjesztő osztályra bízza. A tokeneket az `ArrayList<ArrayList> lines` kollekcióban tárolja, ahol a külső `ArrayList` a fájl sorait, míg a belső `ArrayList` egy adott sorban lévő tokeneket tartalmazza `Stringek` formájában. Valamennyi, a fájlból beolvasott token egyetlen iterációban fog megjelenni, azaz a sorok közti határ az osztály használója számára nem lényeges. Mindemellett az osztály számos metódust kínál a tokenekhez kapcsolódó egyéb információ kinyerésére, melyek : az aktuális token sorának sorszáma, hányadik az aktuális

token a fájlban, hányadik az aktuális token a sorában, valamint a teljes sor mely az aktuális tokent tartalmazza. Ezek az információk jórészt a parszer által talált esetleges szintaktikai/szemantikai hibák kezeléséhez szükségesek.

Az `ASN.1FR` a `TokenIterator` absztrakt `read()` metódusát implementálja. Beolvassa a fájl sorait, az üres sorokat kihagyja. A nem üres sorokat tokenekre bontja, a megjegyzéseket (konfigurálható, ASN.1-ben -- jelöli a megjegyzések kezdetét) figyelmen kívül hagyja, valamint összefűzi a két szóból álló fenntartott szavakat egy aláhúzásjel segítségével. A beolvasott tokeneket a `TokenIterator lines` attribútumában tárolja. Végül beállítja a sikeres beolvasást jelentő flaget.

Ezután következik a szintaktikai és szemantikai elemzés, melyet az `ASN.1Parser` osztály (`parse()` metódusa) valósít meg. Az osztály konstruktorában megkapja a tokeneket tartalmazó `TokenIterator` példányt. A tokeneken végighaladva ellenőrzi, hogy az ASN.1 specifikáció megfelel-e a szintaktikai és szemantikai szabályoknak. Hiba esetén egy `ASN.1ParserErrorException`-t dob, melynek üzenetében értelemszerűen közli a hibát, hogy mit várt el, valamint a hiba helyét és a sort melyben a hibát okozó tokent találta.

A sikeresen feldolgozott ASN.1 elemeket, illetve azok jellemzőit a `parse()` metódus `ASN.1Item` példányokban tárolja el, az `ASN.1Parser` osztály `ArrayListUnique<ASN.1Item> items` listáján.

Az `ArrayListUnique` osztály a `java.util.ArrayList` kiterjesztése olyan módon, hogy nem engedi meg duplikált elemek felvételét a listára, ezzel `Set` - szerű funkcionalitást biztosítva. Magát a `Set` - et azért nem volt célszerű használnom, mert az nincs tekintettel az elemek sorrendjére, aminek pedig nagy jelentősége van. Az osztály `AddU(E o)` metódusa a kapott `E` típusú példány `equals(Object o)` metódusát használja fel a duplikáció felismerésére. Ha ez alapján a hozzáadandó elem már szerepel a listán, akkor egy `ObjectIsNotUniqueException` kerül kiváltásra és továbbadásra, melyet így a metódus hívójának kell kezelnie.

Az `ASN.1Parser` items listája is ilyen típusú. Az `ASN.1Item` osztály `equals(Object o)` metódusa az ASN.1 elemeket a nevük (`name` attribútumuk) alapján hasonlítja össze, így

felismerhető, ha az ASN.1 specifikációban két elem azonos névvel rendelkezik, ami persze hiba.

Az `ASN.1Item` osztály tartalmaz egy statikus beágyazott osztályt, az `ASN.1ItemProperties`-t. Ez az osztály az ASN.1 elemek bizonyos tulajdonságait sorolja fel `enum` típusok segítségével. Ezek:

- `TagTypes` : A támogatott ASN.1 osztályok
- `DataTypes` : Az ASN.1 alaptípusokat (pl. `CHOICE`) definiálja `enum` példányokként. Az `enum` konstansok neve megegyezik az ASN.1 típusnok nevével. A `DataTypes enum` típusnak vannak attribútumai is, melyek az adott ASN.1 típus tulajdonságait határozzák meg. Ezek az attribútomok is `enum` típusúak, melyek:
  - `PrimConstr` (`PRIMITIVE` ,`CONSTRUCTED`) - egyszerű vagy összetett típus
  - `CanAppearWithoutId` (`YES` , `NO` , `ALWAYS`) - előfordulhat e az adott típusú elem ASN.1 azonosító nélkül
  - `ChildTypeNameRequired` (`YES` , `NO`) : Csak a `SEQUENCE OF` típusnál `YES`. Ekkor a parszer elvárja, hogy az ilyen alaptípusú elemeknél legyen megadva, hogy egyedüli gyermek-elemként (azaz blokk nélkül) definiálva milyen típust tartalmaz az elemhez tartozó lista. Pl.:
    ```
    A ::= [PRIVATE 34] SEQUENCE OF B
    ```
    Ekkor az `A` elem a `PRIVATE 34` taggel egy `B` típusú elemeket tartalmazó listát azonosít. Itt a `B` a gyermek típusneve (`Childtypename`).
  - BlockRequired (`YES` , `NO`) : Tartozik e az adott elemhez blokk. (`SEQUENCE, SET, CHOICE, ENUMERATED, BitString` esetén `YES`)

Az ASN.1 specifikációból beolvasott elemeket az `ASN.1Item` osztály példányai reprezentálják. Az `ASN.1Item` konstruktorában megkapja az elem nevét, valamint hogy az illető elem - amennyiben összetett ASN.1 típusú - az inputleíróban rekordként, alrekordként fog e esetleg szerepelni. Ezen információk - az ASN.1 nyelvet kiterjesztve - az ASN.1

specifikáció módosításával adhatóak meg. Egy adott elem neve előtt elhelyezett ’+’ rekordra utal, míg a ’.’ egy alrekordra. Ezek az inputleíróban is hasonló módon kerülnek jelölésre.

Egy-egy elemhez a következő információk kapcsolódnak : rekord vagy alrekord-e (vagy egyik sem), az elem neve , ha létezik akkor az elem azonosítója és ASN.1 osztálya, ASN.1 típusa vagy ha származtatott típus (`boolean isDerived`) akkor az őt definiáló típus neve, valamint a gyermekei. Ezen kívül további (`boolean` típusú) változók is szerepelnek az attribútumok között, melyek a fa felépítéséhez és kezeléséhez szükségesek a későbbiekben, ezek a szintaktikai és szemantikai elemzés után még nem tartalmaznak érvényes információt : (`choiceWithId , built`).

Az `ASN.1Item` osztály tagosztálya (member class) a `Child` osztály, melynek példányai az adott összetett ASN.1 típussal - `SET,SEQUENCE,CHOICE`, - a `SEQUENCE OF` egyedüli gyermeke egyébként más módon is jelölésre kerül, lásd `ASN.1Item.ASN.1ItemProperties.DataTypes.ChildTypenameRequired enum` típus - rendelkező elem gyermekét/gyermekeit reprezentálják. Fontos megjegyeznem, hogy ezek a `Child` osztálybeli példányok nem tartalmaznak másik `ASN.1Item` példányra mutató referenciát, csak a gyermek opcionális nevét és típusát, azaz az `ASN.1Item` osztály példányaiból fa közvetlenül nem építhető.

Egy adott `ASN.1Item` példány által reprezentált (összetett típusú) ASN.1 elem gyermekei az `ASN.1Item` osztály `ArrayListUnique<Child> Children` listáján kerülnek tárolásra. Egy adott `ASN.1Item` példány gyermekeinek duplikációja (a gyermekek típusneve, azaz a `typename` attribútumuk alapján) itt sem megengedett.

Az ASN.1 jelölési konvenciói szerint egy elem gyermekeinek felsorolásakor általában az adott gyermek neve megegyezik annak típusának nevével, az első betűt kivéve: A név kis betűvel kezdődik, míg a típus naggyal. Ha ez teljesül a `Child` osztály (`name`, `type`) attribútumai közül csak a `type` kerül kitöltésre. Ha nem teljesül, azaz az gyermeknek van típusnevétől különböző, opcionális neve akkor a `Child` osztály (`name`, `type`) attribútumai közül mindkettő kitöltésre kerül. Az opcionális `name` attribútum fogja jelölni az elem nevét mind a fában mind az inputleíróban, de a `type` attribútumára is szükség lesz, hiszen az hivatkozza azt az ASN.1 elemet, mely a gyermek típusát definiálja.

Például:

```
CamelDestination ::= [APPLICATION 51] SEQUENCE
{
    typeOfNumber                TypeOfNumber        OPTIONAL,
    camelDestinationNumber      AddressStringDigits OPTIONAL
}
```

Az első gyermek esetén (name,type) = (null , „TypeOfNumber”) , míg a második esetén („CamelDestinationNumber” , „AddressStringDigits”) lesz. (Az inputleíró konvenciói miatt a nevek nagy kezdőbetűvel írandók, ezért a ’c’ helyett a ’C’.) A fent leírt logikát az ASN.1Item osztály túlterhelt nevű addChild(…) publikus metódusai biztosítják, ezek hívják meg a Child osztály private konstruktorát is. Ily módon ezek a megszorítások kikényszeríthetőek.

A Child osztály optional boolean típusú attribútuma jelöli, hogy a gyermek opcionális e.

Az ASN.1Parser parse() metódusa a szintaktikai és szemantikai elemzés befejezte után, felépít egy HashMapet mely kulcsként az elemek nevét (String), értékként a megfelelő ASN.1Item példány referenciáját tartalmazza. Erre a fa felépítése során lesz szükség.

Egy adott name attribútumú ASN.1Item referenciáját az ASN.1Parser getItemByName(String nam) metódusa adja vissza, - mely az imént említett HashMapet használja. Ha nem találja a nevet, az hibára utal és kivétel keletkezik Ekkor olyan típusnévre való hivatkozás van az ASN.1 specifikációban melyhez nem tartozik megfelelő nevű ASN.1 elem, azaz a hivatkozott típus definíciója nem létezik.

Ezt követően az ASN.1Builder veszi át a szerepet, konstruktorában egy ASN.1Parser példányt kap mely tartalmazza az ASN.1 specifikációból beolvasott elemek nyers információit (ASN.1Item példányok). Az osztály buildAll() metódusa hivatott felépíteni a fát. Ehhez szükség van egy olyan osztályra mely alkalmas faelemek reprezentálására.

Ezt a szerepet az ASN.1BuilderItem osztály tölti be. Ez már tartalmaz más, azonos osztálybeli példányokra mutató referenciákat (Children), valamint a fent leírt logikának megfelelően name és typename attribútumokat. (Figyeljük meg, hogy az ASN.1Item

osztály példányainak csak `name` attribútuma volt, `name` és `type(name)` attribútumai eddig csak az `ASN.1Item` osztály által tartalmazott `Child` osztály példányainak voltak). Azaz az `ASN.1BuilderItem` példányok már alkalmasak faelemek reprezentálására.

Minden `ASN.1BuilderItem` példány egy `ASN.1Item` példányt tartalmaz az `it` nevű referencia által. Az osztály attribútumai tárolják továbbá a : gyermek elemeket, a szülőt , a típusnevet és az opcionális nevet , az elem teljesen minősített nevét és szintjét a fában, valamint azt az információt (`boolean putSetAround`), hogy az illető elem `Set` blokkban legyen e az inputleíróban vagy nem (ez később kerül beállításra). Ezentúl az osztály tagként csak példányszintű metódusokat tartalmaz, melyek szabályozott és további logikát nyújtó hozzáférést biztosítanak a tartalmazott `ASN.1Item` példány nyers információihoz. Ilyen módon az `ASN.1BuilderItem` példányok jól használható felületet képeznek az `ASN.1Item` példányok felett azok faelemként való kezeléséhez, valamint későbbiekben az inputleíró generálásához szükséges információk kiolvasásához. Ezek a metódusok : `int tag(), boolean : primitive(), ascii(), record(), subrecord(), choice(), set(), sequenceOf(), hasPrimitiveChild(), choiceWithId()` (ez arra utal, hogy az adott elem egy `CHOICE` és szülője egy `SEQUENCE OF` ASN.1 típus, azaz az adott elem gyermekeiből lista képezhető oly módon, hogy a listán bármely gyermeke előfordulhat.) , `getName()`.

Az osztály `getTagBytesAsHexString ()` nevű metódusa szolgáltatja - a `tag()` , `primitive()` valamint a `tagType()` példányszintű metódusok által kapott adatokból - az aktuális ASN.1 elem osztályához és azonosítójához (tag) tartozó BER taget. Ha `(tag() == -1 || tagType() == null)` , az hibát jelez és a program futása kivétellel befejeződik. Az alkalmazás által biztosított szintaktikai és szemantikai megszorítások biztosítják, hogy ilyen hiba ne forduljon elő, célja az esetleges későbbi módosítások utáni önellenőrzés. Maga a hiba arra utal, hogy olyan `ASN.1BuilderItem` példányt próbálunk az inputleíróba kiíratni amely példány mögött lévő `ASN.1Item` példánynak nincs érvényes tagje illetve osztálya, holott az inputleíró szabályai az adott elem BER taggel való azonosítását megkövetelik. A jelenlegi implementációban ez csak `CHOICE` esetén lehetséges, de erre a típusra semmiképp sem hívódik meg ez a metódus.

Az ASN.1Builder tartalmaz egy roots nevű HashMapet. Ez tárolja a fa építése folyamán a részfák gyökerét (egy-egy ASN.1BuilderItem) valamint typename (String) attribútumuk alapján ezek az elemek kereshetőek is. Fontos itt megjegyezni, hogy a fát felépítő algoritmus az összetett típusú ASN.1 elemek gyermekeinek mindvégig a típusnevét (typename attribútumát) használja, hiszen ez határozza meg az őt definiáló ASN.1 elem nevét. A faelemként szolgáló ASN.1BuilderItem példányok inputleíróbeli nevét viszont már a getName() metódus hívásával kapjuk meg, melynek visszatérési értéke alapértelmezésben a példány típusneve (typename), illetve ha létezik opcionális neve (name != null), akkor az.

builAll() metódusa végighalad az ASN.1Item példányokon (ASN.1Parser ArrayListUnique<ASN.1Item> items). Ha az aktuális példányhoz (*) tartozó - mely akár egyelemű is lehet - részfa már fel lett építve (built) folytatja a következővel. Ha nem, akkor meghívja a build(ASN.1Item ai, String optionalName) metódust az aktuális ASN.1Item példánnyal.

Ez létrehoz egy új ASN.1BuilderItem példányt, mely tartalmazni fogja - az ASN.1Item it referencia típusú attribútum formájában - a paraméterként kapott feldolgozandó ASN.1Item példányt. A metódus az optionalName nevű paraméterét a korábban leírt (típusnév/opcionális név) logikának megfelelően kezeli.

Ha vannak a kapott ASN.1Item példány által reprezentált ASN.1 elemeknek gyermekei végighalad rajtuk : ha az adott gyermek típusneve szerepel a roots-ban, akkor az ahhoz tartozó részfa már korábban felépítésre került. Ekkor a részfa gyökere - ami egy ASN.1BuilderItemre mutató referencia - eltávolításra kerül a HashMapből és ez lesz a gyermek referenciája. Ily módon a korábban felépített részfa hozzácsatolásra kerül a teljes fához. Ha gyermek típusneve nem szerepel a roots-ban akkor fel kell építeni a hozzátartozó részfát. Ekkor a gyermek elem típusneve alapján meg kell keresni a típust definiáló adott nevű ASN.1 elemet reprezentáló ASN.1Item példányt (ha nincs, akkor az hibára utal az ASN.1 specifikációban, lásd ASN.1Item.getItemByName(String nam)). A keresésből kapott ASN.1Item referenciára ekkor rekurzívan meghívódik a build(…) metódus, melynek visszatérési értéke fogja szolgáltatni az aktuális gyermek referenciáját, valamint az

aktuális gyermek szülője (`parent`) is értelemszerűen beállításra kerül. Miután az éppen feldolgozott `ASN.1Item` valamennyi gyermekéhez tartozó részfa felépült az `ASN.1Item` példány built attribútuma `true`-ra állítódik, majd a `build(…)` metódus visszatér az újonnan létrehozott `ASN.1BuilderItem` példánnyal.

Végül a `build(…)` a felépített részfa gyökerével visszatér a `buildAll()` metódusba.

A részfa gyökere a fentebb (*)-gal jelölt, aktuális `ASN.1Item` példányt tartalmazó `ASN.1BuilderItem` példány referenciája. Ez utóbbi az `ArrayList<ASN.1BuilderItem> Children` attribútumán keresztül rekurzívan tartalmazza az aktuális elemnek (*) az ASN.1 specifikációban meghatározott gyermekeit is - innen a (rész)fa. A gyermekkel nem rendelkező ASN.1 elemeket egyelemű, csak gyökérrel rendelkező részfaként értelmezzük.

A kapott gyökér elhelyezésre kerül a `roots`-ban, az iteráció pedig folytatódik a következő `ASN.1Item` példánnyal. Az algoritmus végén a `roots`-nak pontosan egy elemet kell tartalmaznia, mely a teljes ASN.1 specifikációt reprezentáló fa gyökere. Ha nem egy elemet tartalmaz, az arra utal, hogy a felépített részfák nem összefüggőek, a specifikáció hibás. Ekkor a program futása kivétellel véget ér.

Siker esetén az `ASN.1Builder` osztály `ASN.1BuilderItem root` attribútuma a `roots` egyetlen elemének referenciáját fogja tartalmazni. Az algoritmus tehát egyszerre kétféle szinten (`buildAll()` és `build(…)` metódusok) építi fel a fát. Ennek jelentősége az, hogy az elemek a specifikációban nem feltétlenül a megfelelő sorrendben fordulnak elő. A specifikáció elemeit akár össze is keverhetjük, az algoritmus akkor is ugyanazt a fát fogja - helyesen - felépíteni.

Faelemek alatt a továbbiakban `ASN.1BuilderItem` példányokat értünk, míg *elemek* alatt ASN.1 specifikációbeli elemeket, illetve az őket reprezentáló `ASN.1Item` példányokat, melyek ugyanakkor a faelemek által tartalmazott ’beágyazott információforrások’ is - ez utóbbit használjuk akkor is, amikor ezt a tulajdonságukat szeretnénk kihangsúlyozni.

A fában minden ASN.1 elem annyiszor fordul elő, ahányszor a specifikációban nevére - típusnév formájában - hivatkozás történik (illetve a gyökérelem egyszer). De valamennyi

előfordulási helyén *különböző* `ASN.1BuilderItem` példányok formájában. Ezzel ellentétben valamennyi ilyen `ASN.1BuilderItem` *ugyanazt* az `ASN.1Item` példányt hivatkozza mint információforrást. Ez utóbbi magyarázata az ASN.1 specifikáció absztrakt és rekurzív jellege, azaz, hogy az egyszer és csak egyszer definiált elem - mely egyben *gyenge típus* is -, a nevére típusnévként való hivatkozás formájában újrafelhasználható, és a specifikációban valamennyi előfordulási helyén ugyanazt az adatstruktúrát jelenti. Ezzel ellentétben az `ASN.1BuilderItem` példányok sokszorozott voltát az indokolja, hogy ugyanaz az ASN.1 elem, a szülőjétől és gyermekeitől - azaz a fabéli környezetétől - függően többféleképpen viselkedhet : Egyrészt az implicit tagging, másrészt az inputleíró generálásának speciális kívánalmai miatt. Ennek okán szükség lehet egy adott `ASN.1BuilderItem` által tartalmazott `ASN.1Item` tulajdonságainak megváltoztatására is.

Ha ezt minden körültekintés nélkül tennénk akkor ezzel egyszerre valamennyi azonos névvel rendelkező fabéli elem információit felülírnánk, ezért ilyen indokolt esetben szükségünk lesz egy adott `ASN.1BuilderItem` példány által tartalmazott `ASN.1Item` példány klónozására és ilyen módon a klónozott példány tulajdonságainak biztonságos, a fának csak egy adott pontjára vonatkozó megváltoztatására. Az `ASN.1Item` és belső osztálya a `Child` ennek megfelelően implementálják a `clone()` metódust. Erre a `reduct(root)` metódusban lesz szükségünk.

A kész fán, illetve annak elemein még néhány módosítást és ellenőrzést végre kell hajtanunk. Valamennyi ilyen műveletet egy-egy metódus hajt végre, melyek mindegyike *a fa teljes bejárásával* valósítja meg feladatát.

A `fill(…)` metódus a fát bejárva feltölti az elemek `level` (az elem szintje a fában, ahol a 0 értéket a gyökérelem kapja) illetve `fqn` (teljesen minősített név) attribútumait.

A következő (1) , (2) pontokban az alkalmazás által támogatott ún. *implicit tagging* tekintendő alapértelmezettnek. A (3) pontban, az *explicit tagging* tárgyalásánál részletezem majd a explicit taggel ellátott elemek kezelésének eltérő szabályait.

(1) A `reduct(…)` metódus feladata a fa *redukciója*. Ennek folyamán először felismerésre kerülnek az ún. *származtatott típusokból álló láncok* (melybe beleértjük a kételemű láncot is).

Ez egy E_1, … , E_n alakú struktúra, ahol E_k egy ASN.1 elem definíciója. Az E_k elem típusnév - mint *egyetlen* gyermek - formájában hivatkozza az E_{k+1} elem nevét, mely gyenge típusként az E_k elemet definiálja. Az E_1, … , E_{n-1} elemek származtatott típusúak, míg az E_n ASN.1 alaptípusú. Például :

```
PhoneNumber ::= [APPLICATION 72] IMPLICIT PositiveAsciiNumber
PositiveAsciiNumber ::= [APPLICATION 71] IMPLICIT AsciiNumber
AsciiNumber ::= [APPLICATION 70] IMPLICIT OCTET STRING
```

A fenti példában n=3. E_1 pedig a `PhoneNumber` (telefonszám 3-9 mérettel), E_2 a `PositiveAsciiNumber` (pozitív egész számot tartalmazó numerikus sztring), E_3 az `AsciiNumber` (numerikus sztring) definíciója.

A származtatott típusok által hordozott többlet típusinformáció az inputleíró generálásának szempontjából nem szükséges, a FileReader által beolvasott mezők esetleges szintaktikai/szemantikai validálása a ruleset illetve az azt futtató CV dolga. A FileReader nem is képes ilyen ellenőrzésekre, például jelen esetben nem tudná ellenőrizni, hogy a beolvasott ASCII típusú mező pontosan milyen értéket tartalmaz, valóban pozitív egész-e. Másrészt a származtatott típusok az inputleíróban nem jelennének meg konkrét elemként, például `Field` vagy `Block` formájában, ahogy a BER kódolású fájlban sem jelennek meg. Így a mi szempontunkból teljesen feleslegesek.

A redukció első lépése a láncok felismerése. Ha a fa bejárása során származtatott típusú faelemet találunk (az elem `isDerived` attribútuma `true`), akkor legyen ez az elem az E_1. Majd a fában lefelé haladva keressük meg E_n-t mely az első olyan elem melynek típusa már nem származtatott (`dataType != null`), hanem ASN.1 alaptípus. Ha a fa legalsó szintjét elérve sem találunk ilyet, az hiba, az alkalmazás kivétellel befejezi futását.

Ezután - továbbra is implicit tagginget feltételezve - az E_1, … , E_n láncot E_1’ - re egyszerűsítjük, melyben már nem szerepelnek származtatott típusnevek. Az E_1’ felveszi az E_n típusát (ami immár egy használható ASN.1 alaptípus), valamint ha vannak, akkor gyermekeit is.

(2) Az előző pontban ismertettem a fa redukciójának szükségességét az inputleíró generálásának és a (BER) kódolás szempontjából. Most az ASN.1 nyelv sajátosságai felől megközelítve folytatom ismertetését.

Fontos célja és része a redukciónak az ASN.1 által is ismert *implicit tagging* implementálása. Valamely ASN.1 kódolásban (pl. a BER) egy implicit taggel ellátott típus egy olyan, más típusból származtatott típus ahol az alant lévő típushoz (amiből származtatunk) képest való megkülönböztetést az alant lévő típus tagjének *elhagyása* mellett egy új tag bevezetése biztosítja. A származtatott típusokból álló láncok elemei közül több is tartalmazhat taget, más néven azonosítót, de nem feltétlenül kell. Implicit tagging esetén egy ilyen láncban lévő elemek közül csak a legfelső szintűnek kell előfordulnia a (BER) kódolásban (és ezzel együtt az inputleíróban is), ennek azonosítója pedig a láncban az első azonosító lesz. Azaz E_1' azonosítója az E_k azonosítója lesz, ahol k=1…n, és E_k a legkisebb sorszámú elem amely rendelkezik azonosítóval. Az előző példa redukált változata:

```
PhoneNumber ::= [APPLICATION 72] IMPLICIT OCTET STRING
```

Ezt szokták még úgy is említeni, hogy az implicit tageket 'valami helyett' (instead of) kell érteni. Ez alapján a példánk:

```
PhoneNumber ::= [APPLICATION 72] instead of [APPLICATION 71] instead of
[APPLICATION 70] instead of OCTET STRING
```

(3) Az e*xplicit tagginget* az alkalmazás *nem támogatja*, de ismertetését - a későbbi bővítést lehetővé teendő - szükségesnek tartom. Valamely ASN.1 kódolásban (pl. a BER) egy *explicit taggel* ellátott típus egy olyan, más típusból származtatott típus ahol az alant lévő típushoz (amiből származtatunk) képest való megkülönböztetést az alant lévő típus tagjének *megtartása* mellett egy *új külső tag* bevezetése biztosítja. Ez esetben tehát valamennyi, taggel rendelkező elem tagjének meg kell jelennie a (BER) kódolásban, csak a taggel nem rendelkezők hagyhatók el (ez alól kivételek az ASN.1 alaptípusok, ezek tagje kötelezően meg kell, hogy jelenjen). Ez alapján a fenti példa (IMPLICIT helyett EXPLICIT-et értve) nem redukálható. De következő már igen:

```
(E1)PhoneNumber ::= EXPLICIT PositiveAsciiNumber
(E2)PositiveAsciiNumber ::= [APPLICATION 71] EXPLICIT AsciiNumber
(E3)AsciiNumber ::= [APPLICATION 70] EXPLICIT OCTET STRING
```

Az eredmény:

```
(E1')PhoneNumber ::= [APPLICATION 71] EXPLICIT AsciiNumber
(E3)AsciiNumber ::= [APPLICATION 70] EXPLICIT OCTET STRING
```

Fontos még megjegyezni, hogy explicit tagging esetén az ASN.1 alaptípusok is rendelkeznek alapértelmezett `UNIVERSAL` osztálybeli taggel, melyek kötelezően meg kell, hogy jelenjenek a (BER) kódolásban.

Explicit tagging esetén, tehát a redukáló algoritmus a következőképpen módosul: A lánc felismerésénél az E_n keresését az *első* taggel rendelkező elem megállítja (így elképzelhetőek akár egy elemű láncok is) - ez lesz E_n. A lánc definíciója tehát : $E_1, \ldots, E_n$ ahol $E_1, \ldots, E_{n-1}$ származtatott típusú és egyikük sem rendelkezik taggel, E_n pedig rendelkezik taggel és ha levélelem a fában akkor ASN.1 alaptípusú, ha pedig közbenső elem, akkor nem. Ha levélelem és nem ASN.1 alaptípusú, a specifikáció nem helyes. Ha egy elem sem rendelkezik taggel a láncban, akkor még mindig használható a levélelem ASN.1 alaptípusának megfelelő alapértelmezett, `UNIVERSAL` osztályú tag. A fenti példában n=2, E_3 nem eleme a láncnak. Az explicit tagging tehát megengedi, hogy származtatott típusú elemek maradjanak a fában, amennyiben van tagjük. Az inputleíróban az ilyen struktúrákat egymásba ágyazott `Block` típusokkal reprezentálhatjuk. Az előzőektől eltekintve a redukció algoritmusa változatlan explicit tagging esetén is.

Az implicit tagging 'instead of' analógiájára explicit tagging esetén az 'in addition to' (körülbelül : ehhez még jön ez is) értendő.

(cwi) `A reduct(…)` metódus másik feladata már nem annyira az ASN.1 nyelv, mint inkább az inputleíró sajátosságaihoz kötődik. Ha egy adott elem egy `CHOICE` és szülője egy `SEQUENCE OF` ASN.1 típus, akkor az adott elem gyermekeiből lista képezhető oly módon, hogy a listán bármely gyermeke előfordulhat. Mindkét összetett típus inputleíróbeli típusa egy `Set` típussal kezdődik. Azaz az inputleíróban ez szerepelne: `Set{ Set{ … } }`. Azonban a FileReader egy ilyen konstrukció hatására végtelen ciklusba futna. Ezt kell kiküszöbölnünk.

A megoldás az, hogy a két elemet egyesítjük és megjelöljük az `ASN.1Item.choiceWithId` attribútum segítségével (később a faelem azonos nevű metódusa adja vissza ennek értékét). Az elnevezés onnan ered, hogy az itt leírttól eltérő esetben a `CHOICE` ASN.1 összetett típusú elemeknek az alkalmazás által lefedett ASN.1 nyelv-részhalmazban nem lehet tagjük. Ez esetben viszont lesz. Ezek az faelemek az

inputleíró generálásakor speciális módon kerülnek kezelésre. Lásd még a `notPutSetAround(…)` metódust.

Az egyesítés algoritmusa hasonló az (1) és (2) pontokban leírtakhoz, oly módon, hogy az E_1 lesz a `SEQUENCE OF`, míg az E_2 (n=2) a `CHOICE` típusú elem. Egy konkrét példa, az algoritmus előtt:

```
DetailList ::=  [APPLICATION 3] SEQUENCE OF Detail
Detail ::= CHOICE
{
    mobileOriginatedCall   MobileOriginatedCall,
    mobileTerminatedCall   MobileTerminatedCall,
    valueAddedService      ValueAddedService,
    gprsCall               GprsCall
}
```

És utána:

```
DetailList ::=  [APPLICATION 3] CHOICE
{
    mobileOriginatedCall   MobileOriginatedCall,
    mobileTerminatedCall   MobileTerminatedCall,
    valueAddedService      ValueAddedService,
    gprsCall               GprsCall
}
```

A `reduct(…)` metódus az (1) , (2) , és a (cwi) pontokban leírt algoritmust valósítja meg. Amennyiben egy faelemhez tartozó `ASN.1Item` példányt módosítani kell, előbb klónozza azt, a másolatot módosítja és a faelembeli eredeti referenciát (`ASN.1Item` it) cseréli ki az újra. Ily módon csak az aktuális faelemhez tartozó `ASN.1Item` példány kerül megváltoztatásra. A redukció után valamennyi elem típusa valamely ASN.1 alaptípus lesz. (Explicit tagging esetén ez nem lenne biztosított, lásd a (3) pontot.)

A `checkPrimitiveIDs(…)` metódus ellenőrzi, hogy a redukció után valamennyi primitív ASN.1 típusú elem rendelkezik e taggel. Ha nem, az implicit tagging esetén hiba, mert az adott elem nem lesz azonosítható. Ekkor az alkalmazás kivétellel befejezi futását. Összetett típusok esetén ez az ellenőrzés már a szintaktikai és szemantikai elemzés folyamán megtörténik (Beleértve azt is, hogy a `CHOICE` ASN.1 összetett típusú elemeknek az alkalmazás által lefedett ASN.1 nyelv-részhalmazban nem lehet tagjük. `Context-`

`specific` osztályú elemek esetén például lehetne, sőt biztosan lenne, ugyanis az ilyen osztályba tartozó `CHOICE` típusú elemek tagje automatikusan explicitként értelmezendő, mivel akként viselkedik.). A primitív típusokat azért kell itt ellenőrizni, mert a származtatott típusok miatt a redukció előtt nem egyértelműen eldönthető a tagek szükségessége. Ezen a ponton ellenőrizhető a korábbi algoritmusok, például a szintaktikai és szemantikai elemzés valamint a redukció helyessége.
A bővíthetőséget illetően a `Context-specific` osztályú taggel rendelkező, tag nélküli primitív típusú elemek - amennyiben szükséges - itt kaphatják meg alapértelmezett `UNIVERSAL` osztálybeli tagjüket. Hasonlóan az explicit taggel (lásd (3) pont - explicit tagging) rendelkező primitív elemek itt kaphatják meg a primitív ASN.1 típusokat reprezentáló alapértelmezett `UNIVERSAL` pszeudotageket - egy további gyermekelem formájában.

Ezután még egyszer lefut a `fill(…)` metódus, hogy immár a végleges formájú fán állítsa be a `level` és `fqn` attribútumokat. A korábbi (redukció előtti) hívás eredménye elemzés céljából nagyon hasznos, ezért nem távolítottam el a kódból.

Végül a `notPutSetAround(… )` kezeli azokat az (összetett) elemeket, melyek köré - például a (cwi) pontban már ismertetett `Set` a `Set`-ben probléma elkerülése végett - az inputleíróban nem szükséges `Set { … }` blokkot helyezni. Ezek az elemek vagy ASN.1 `SET` típusúak, vagy a `choiceWithId() == true` tulajdonsággal rendelkeznek. Ez esetben a faelem `putSetAround` attribútuma az alapértelmezett `true` helyett `false` lesz.

Ezzel az `ASN.1Builder buildAll()` metódusa végére ért a fa építésének és a rajta végzett átalakításoknak.

Az `InputDescWriter` osztály konstruktora egy `ASN.1Builder` példányt - mely tartalmazza az `ASN.1BuilderItem` példányokból álló fa gyökerét (`root` attribútum) - valamint az kimeneti fájl nevét kapja meg paraméterként. Túlterhelt nevű, paraméter nélküli `printInputDescr()` metódusát meghívva indíthatjuk el a fa leképezését.

Ez meghívja az azonos nevű `printInputDescr(ASN.1BuilderItem a)` szignatúrájú metódust a fa gyökerével, mely aztán rekurzív módon létrehozza az inputleírót.
A leíró generálásának bonyolult szabályrendszerét itt nem kívánom teljes részletességgel

bemutatni, magát a kódot olvasva az algoritmus működési elve jól nyomonkövethető. A metódus az aktuális `ASN.1BuilderItem` példány következő attribútumai/metódusai által meghatározott tulajdonságok alapján hozza létre a hozzá tartozó leíró-részt : `getName(), level, choiceWithId(), hasPrimitiveChild(), root, primitive(), parent, sequenceOf(), getTagBytesAsHexString(), putSetAround, choice(), Children, record(), set()`.

Ezen a ponton térnék ki az alkalmazás ASN.1- és inputleíróbeli primitív típusokkal kapcsolatos megkötéseire:

Az `ascii()` metódus jelenleg mindig `false` értékkel tér vissza. Eredeti szerepe annak meghatározása, hogy ha az adott elem primitív ASN.1 típus, akkor az `ascii` vagy `octstr` típusú legyen e az inputleíróban. Ennek beállítása teljes biztonsággal azonban csak, utólag kézzel lehetséges, valamennyi ASN.1 elem esetén egyenként meghatározva azt. Bár esetenként, a származtatott típusok információit - még eldobásuk (redukció) előtt - felhasználva - például a `NumberString`, `AsciiString` ASN.1 típusok esetén - lehetséges lenne megállapítani, hogy az ilyen típussal rendelkező elemeket a BER fájlból `ascii` típusként kell beolvasni, de ez a megoldás nem lenne *egyértelmű és teljes*.

Egyértelmű azért nem, mert a mobil operátor a hálózati elemek fel-/átkonfigurálása során dönthet úgy, hogy egy adott mező típusát részben megváltoztatja, például eddig csak alfanumerikus karaktereket tartalmazó mezőbe ezentúl néhány nem nyomtatható karaktert írását is engedélyezi. Mivel a Comptel FileReader inputleírójának `ascii` típusa csak a nyomtatható karaktereket képes beolvasni, ez nem várt hibákat okozna akár egy kisebb változás esetén is. A FileReader célja elsősorban az, hogy ha a rekordformátum helyes, akkor mindent amit csak lehetséges olvasson be. A típussal és tartalommal kapcsolatos ellenőrzéseket a rulesetekben kell implementálni, ahol a fejlesztőnek sokkal nagyobb mozgástere van megfelelő szabályok felállítására.

Ebből a megfontolásból elterjedt gyakorlat, hogy szinte minden mezőt `octstr`-ként olvasnak be (`octstr`-ként bármit be lehet), még az eredetileg `ascii` jellegűeket is, majd ez utóbbiakat például egy `octstr2ascii()` függvény hívásával alakítják át sztringgé. Ez a függvény már képes megfelelően kezelni a nem nyomtatható karaktereket is.

Hasonló a helyzet az egész számokkal is. Az inputleíró `bin` típusa csak előjel nélküli számok beolvasását teszi lehetővé. A kettes komplemenssel ábrázolt egészeket először `octstr`-ként olvassák be, majd megfelelő függvény(ek) hívásával alakítják át előjeles egésszé.

Az előzőekre visszatérve a teljesség pedig azért sérülne, mert tisztán az ASN.1 specifikáció alapján további gépi algoritmusokkal sem lehetne *minden* mezőről eldönteni, pontosan milyen értékeket tartalmazhat - akár a specifikációnak ellentmondva, azt felülbírálva - , csakis speciális esetekben, melyek nem fedik le a teljességet. Ezt csak a teljes interfész-specifikáció ismeretében lehet biztosan megállapítani.

Szintén hasonló okokból a jelenleg támogatott legfontosabb `INTEGER, OCTET_STRING, Boolean, REAL, ENUMERATED, BitString` típusukon kívüli primitív ASN.1 típusokat az alkalmazás felhasználójának kell definiálnia. Ez kétféle módon történhet : vagy a `DataTypes enum` osztályba kell új `enum` példányként felvenni őket, vagy az ASN.1 specifikációba kell beírni őket pszeudoelemekként. Ily módon tetszőleges módon definiálhatjuk tulajdonságaikat.

Az alkalmazás által használt kivételosztályok:

`NoASN.1InputSuccessfullyReadException` - A `TokenIterator` osztály `init()` metódusa dobja, ha úgy próbáljuk meg inicializálni a token iterátort, hogy nem lett semmilyen bemeneti fájl beolvasva.

`ASN.1ParserErrorException` - Az `ASN.1Parser` osztály ezzel jelzi az ASN.1 specifikáció szintaktikai és szemantikai elemzése során észlelt hibákat. Lásd még `ASN.1Parser. getItemByName(…)`.

`ASN.1BuilderErrorException` - Az `ASN.1Builder` osztály metódusai használják a fa felépítése és átalakítása során bekövetkező hibák jelzésére.

`ASN.1DataTypeConstraintViolatedException` - Az `ASN.1Parser` osztály használja a `DataTypes enum` példányok inkonzisztenciáinak jelzésére.

`ObjectIsNotUniqueException` - Az `ArrayListUnique` osztály `addU` metódusa jelzi vele, hogy az aktuálisan a listához adandó példány az `equals(Object o)` metódusa alapján már szerepel a listán, azaz duplikált.

Az `ASN1BERInputDescGenerator` működésének nyomon követését - azaz a debug információk osztályonként, esetenként metódusonként való kiíratását, valamint ezek szabályozását - a `DebugPrinter` osztály - azaz annak statikus beágyazott `D` osztálybeli `enum` típusú példányai és `p(…)` metódusai - teszik lehetővé.

8 Összefoglalás

Az elméleti célkitűzéseket sikerült elérnem. Utólag a választott témakörök közül néhány is elég lett volna a dolgozatba, de ettől függetlenül a mennyiség nem ment a minőség rovására, minden egyes témakört sikerült kellően kifejtenem és lezárnom.

Az alkalmazás eredeti tervéből csak a context-specific és az universal ASN.1 osztályok implementálására nem jutott időm, de az alkalmazás így is megállja a helyét. Sikerült vele legenerálnom ugyanazt az inputleírót, amit annak idején kézzel írt szöveges fájlból, Perl szkript által létrehozott C++ kóddal generáltam. A különbég az, hogy akkor a szintaktikai és szemantikai elemzést, valamint többek között a redukciót manuálisan kellett elvégeznem. Az alkalmazás mindezt teljesen automatizálja és önállóan képes bonyolult döntéseket hozni hibátlan végeredménnyel. Megbízható, az általa implementált ASN.1 nyelv részhalmaz valamennyi szintaktikai és szemantikai hibáját képes felismerni és arra értelemszerű hibaüzenetet kiírni.

Az ASN.1 szintakszisának és szemantikájának azt a részét melyet a jövőben az alkalmazásnak még érdemes implementálnia (pl. explicit tagging) maradéktalanul sikerült specifikálnom.

9 Irodalomjegyzék

[1] Olivier Dubuisson: *ASN.1 - Communication between heterogeneous systems,* Morgan Kaufmann Publishers, 2000.

[2] Yi-Bing Lin, Imrich Chlamtac: *Wireless and Mobile Network Architectures*, John Wiley & Sons, Inc., 2001.

[3] *MDS ARM Programmer's Reference Manual*, Comptel Corporation, 2004.

[4] *MDS/AMD*, Comptel Corporation, 2001.

[5] *iBMD_Revenue_Assurance_Requirements_8.2_v4*, T-Mobile International, 2008.

[6] *iBMD_TAP03_interface_6.1_v2*, T-Mobile International, 2008.

[7] *Req_TAP_TMA_ibmd_6.1_v5*, T-Mobile International, 2008.

[8] *TD57362 (Trasnsferred Account Procedure Data Record Format Specification Version Number 3.6.2)* , GSM Association Classifications, 2000.

[9] Burton S. Kaliski Jr.: *A Layman's Guide to a Subset of ASN.1, BER, and DER,* http://luca.ntop.org/Teaching/Appunti/ASN.1.html, 2004.

[10] *3G & UMTS*, http://www.freewebs.com/telecomm/3g.html, 2005.

[11] *GSM*, http://www.freewebs.com/telecomm/gsm.html, 2005.

[12] *Mobile Switchin Centre*, http://www.mobilephonedirectory.org/Glossary/M/Mobile_Switching_Centre.html, 2005.

[13] *Mobile Station*, http://www.mobile-phone-directory.org/Glossary/M/Mobile_Station.html, 2005.

[14] *Base Station*, http://www.mobile-phone-directory.org/Glossary/B/BS.html, 2005.

[15] *VLR*, http://www.mobile-phone-directory.org/Glossary/V/VLR.html, 2005.

[16] INTERNATIONAL TELECOMMUNICATION UNION: *ITU-T X.680 (Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation)*, 2002.

[17] INTERNATIONAL TELECOMMUNICATION UNION: *ITU-T X.690 (Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)), 2002.*