VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

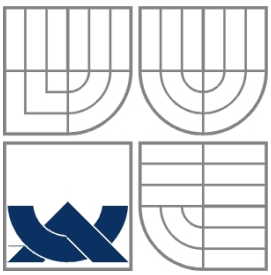
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALYZÁTOR PROTOKOLŮ ŘÍZENÝ PRAVIDLY

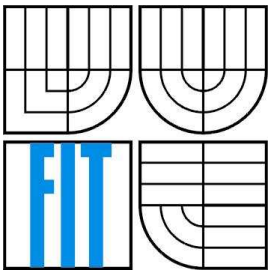
DIPLOMOVÁ PRÁCE
MASTER THESSIS

AUTOR PRÁCE Bc. PETER JURNEČKA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALYZÁTOR PROTOKOLŮ ŘÍZENÝ PRAVIDLY

RULE-DRIVEN PROTOCOL ANALYZER

DIPLOMOVÁ PRÁCE
MASTER THESSIS

AUTOR PRÁCE
AUTHOR
Bc. PETER JURNEČKA

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Dr. Ing. PETR HANÁČEK

BRNO 2009

Abstrakt

Práce se zaobírá přehledem problematiky potřebné na návrh síťového analyzátoru. Popisuje existující řešení, poskytuje teoretické východiska potřebné pro návrh vlastního analyzátoru. V části popisující návrh se specifikuje struktura aplikace jako celku. Samostatná část práce je věnovaná specifikaci metamodelu určeného na popis protokolů a kompilátoru modelů vytvořených v daném metamodelu.

Klíčová slova

Analýza, pakety, počítačové síte, GSM, protokoly

Abstract

The master Thesis deals with an overview of issues necessary for design of custom network analyzer. Describes the existing solutions, provides the theoretical background needed for design of custom analyzer and describes the structure of implemented system. A separate part of the work is devoted to the specification of Metamodel used to modeling of communication protocols and compiler of models modeled with the Metamodel.

Keywords

Analysis, packets, computer networks, GSM, protocols

Citace

Jurnečka Peter: Analyzátor protokolů řízený pravidly. Brno, 2009, diplomová práce, FIT VUT v Brně.

Analyzátor protokolů řízený pravidly

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Dr. Ing. Petra Hanáčka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Peter Jurnečka
24.5.2009

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Doc. Dr. Ing. Petrovi Hanáčkovi za dobré vedení a podnětné návrhy v průběhu konzultací.

Obsah

1 Úvod

Táto diplomová práca sa zaoberá tematikou automatickej analýzy komunikačných protokolov. Úlohou je zoznámiť sa s možnosťami popisu protokolov, zoznámiť s tematikou analýzy komunikačných protokolov, identifikovať moduly potrebné pre zostavenie vlastného analyzátora protokolov a navrhnúť, špecifikovať a implementovať systém slúžiaci na automatickú analýzu dát a komunikačných protokolov. Ďalšou úlohou je otestovať navrhnutý a implmenetovaný systém.

Práca je rozdelená na päť hlavných kapitol. Prvá kapitola uvádza prehľad existujúcich riešení. Opisuje jednotlivé riešenia ich silné a slabé stránky. Keďže navrhovaný analyzátor má zvládať aj analýzu GSM sietí práca sa zameriava sa aj na špecifické analyzátory GSM sietí.

Druhá kapitola uvádza teoretický úvod do problematiky. Obsahuje odpoveď na otázky čo je to protokol a aké sú možnosti popisu protokolov. Opisuje základné prístupy k definícii štruktúry dát a komunikačných protokolov.

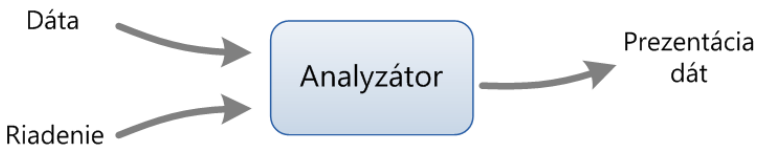
Tretia a štvrtá kapitola sa venujú samostatnej návrhovej práci v tomto diplomovom projekte, podrobne opisuje navrhnutú architektúru analyzátora, použitý systém popisu protokolov, detailne rozpracováva hlavný prínos tejto práce, ktorým je vytvorenie metamodelu a príslušného kompilátora slúžiaceho na popis komunikačných protokolov. Štvrtá kapitola takisto opisuje problémy zistené počas implmenetácie a testovania systému.

Keďže má byť navrhovaný analyzátor otestovaný na skupine GSM protokolov, tak posledná kapitola obsahuje detailnejší úvod do problematiky GSM sietí, príklady protokolov používaných v GSM spoločne s odkazom na kompletný model pravidiel definujúcich vybraný protokol.

2 Prehľad problematiky

S vývojom nových technológii sa počítačové siete stávajú čím ďalej dôležitejšie. Je potrebné zaistiť výkonnosť a bezpečnosť siete, predchádzať problémom, používať efektívne riešenia problémov a prijímať opatrenia ktoré rýchlo vyriešia prípadné problémy. Potrebujeme poznať využitie šírky pásma a ostatných zdrojov na fakturáciu, kontrolu a plánovanie rozvoja siete. Potrebujeme sledovať prevádzku v sieti aby sme sa ubezpečili, že sieť je dobre zabezpečená a prípadné útoky sú včas zaznamenané a zastavené. Môžeme mať problémy v našich novo vyvíjaných aplikáciách a potrebujeme poznať všetky dáta posielané do siete. Vo všetkých týchto prípadoch potrebujeme sieťový analyzátor.

Úlohou tejto práce je vytvoriť pravidlami riadený sieťový analyzátor. Schéma takéhoto analyzátora je na nasledujúcom obrázku.



Obrázok 1. Schéma sieťového analyzátora

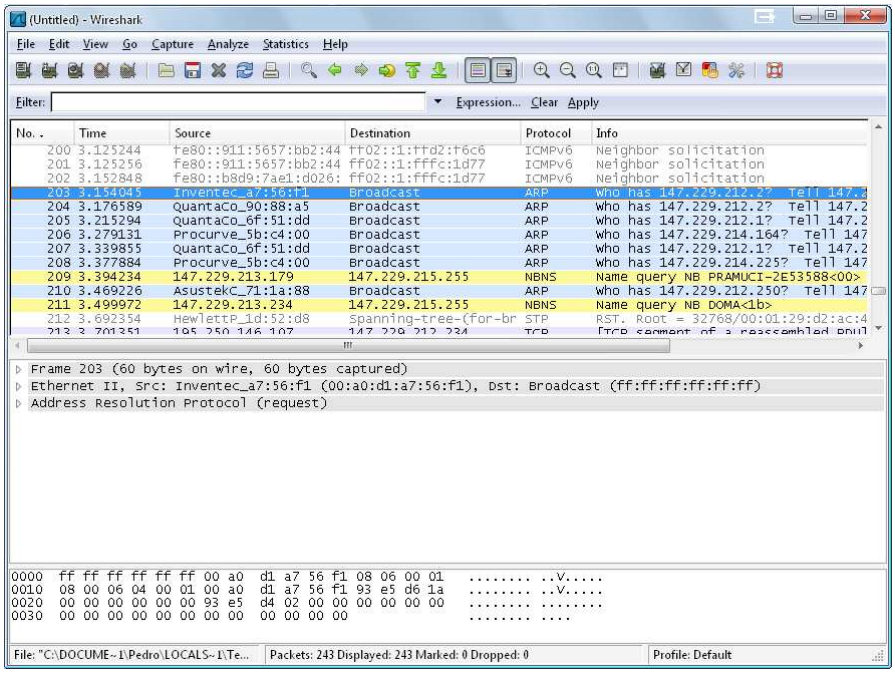
Analyzátor má na vstupe analyzované dáta a riadenie od užívateľa. Výstupom analyzátora je prezentácia dát pomocou užívateľského rozhrania, alebo pomocou vygenerovaného súboru. (napríklad HTML) Vo väčšine súčasných analyzátorov sa používa priame riadenie od užívateľa pomocou príkazov alebo pomocou ovládania cez grafické užívateľské rozhranie. V rámci tejto práce mám vytvoriť analyzátor riaditeľný pomocou príkazov užívateľa, ale aj pomocou vopred definovaných pravidiel.

2.1 Existujúce riešenia

Nasledujúca kapitola uvádza stručný prehľad vybratých sieťových analyzátorov. Opisuje všeobecné softvérové analyzátory ako aj konkrétne jednoúčelové analyzátory GSM protokolov.

2.1.1 Wireshark

Vo svete asi najznámejší sieťový analyzátor. V mnohých inštitúciách a na mnohých školách sa používa ako štandardné vybavenie sieťových laboratórií. Funkcionalita Wiresharku je dosť podobná tcpdump, ale Wireshark poskytuje viac možností filtrácie a triedenia zobrazovaných dát. Umožňuje užívateľovi všetku prevádzku tečúcu cez príslušný segment siete. Stále vyvíjaný produkt, dokáže analyzovať stovky protokolov. Multiplatformový projekt od roku 1998, vydaný pod GNU GPL licenciou, pôvodne pomenovaný Ethereal.

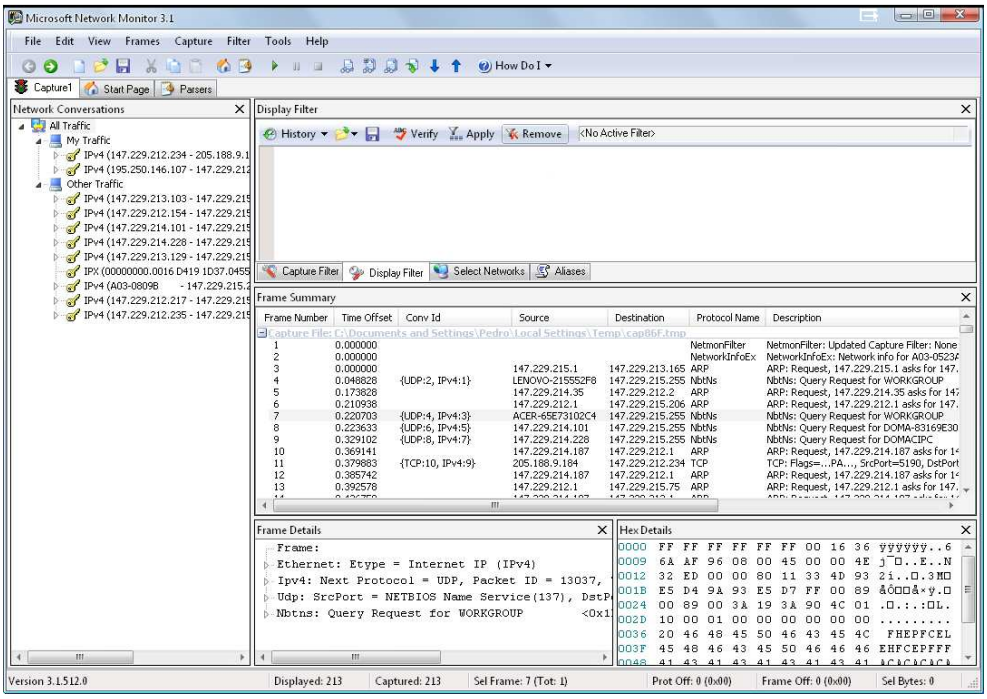


Obrázok 2. Hlavné okno wireshark

2.1.2 Microsoft Network Monitor

Riešenie podobné analyzátoru Wireshark. Softvérový analyzátor bežiaci na OS Windows. Nový živý projekt spoločnosti Microsoft. Zaujímavosťou tohto riešenia je možnosť použitia vlastných definícií protokolov pomocou proprietárneho skriptu.

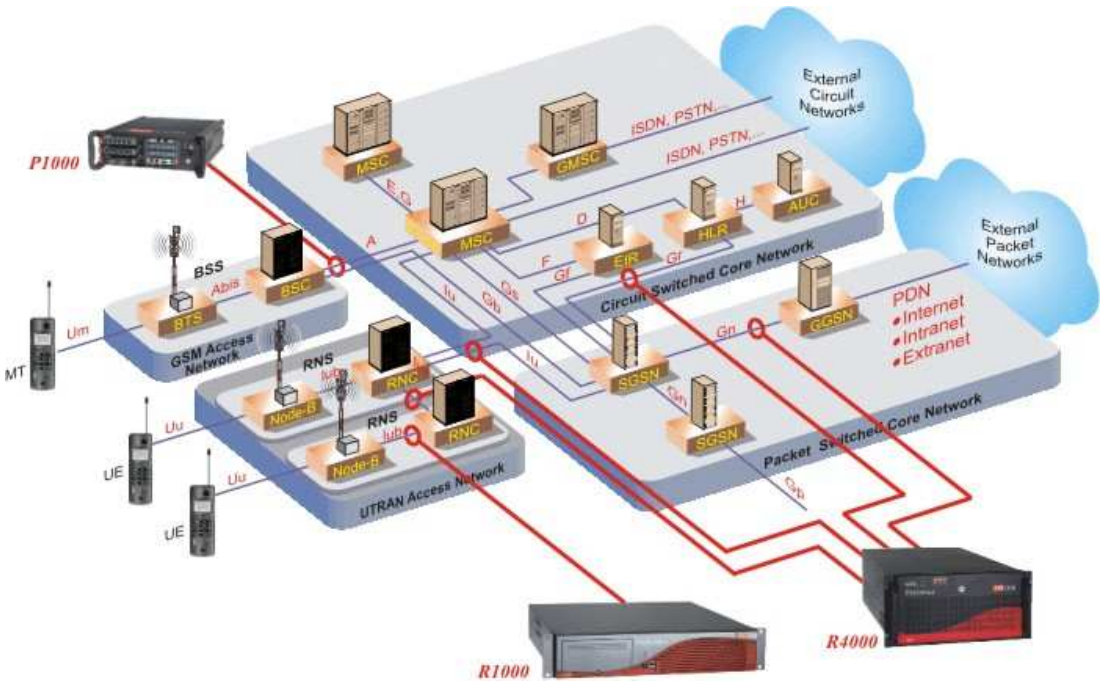
Výhodou tohto systému je dobrá integrácia so systémom, príjemné užívateľské rozhranie a možnosť definície vlastných protokolov. Systém taktiež dokáže prehľadne zoradiť jednotlivé pakety podľa konverzácií. Na druhej strane je veľkou nevýhodou obmedzenosť na jeden operačný systém.



Obrázok 3. Microsoft Network Monitor

2.1.3 RADCOM Protocol Analyzer

Spoločnosť radcom sa zaoberá tvorbou systémov určených na dohľad nad GSM sieťami. Súčasťou ich komplexného dohľadového riešenia je aj analyzátor protokolov určený na presné a detailné merania. K zariadeniu sa je dostupný data sheet, popisujúci celkové riešenie. Nasledujúci obrázok ukazuje pripojenie riešenia od firmy RADCOM ku GSM sieti.



Obrázok 4. HW riešenie firmy RADCOM

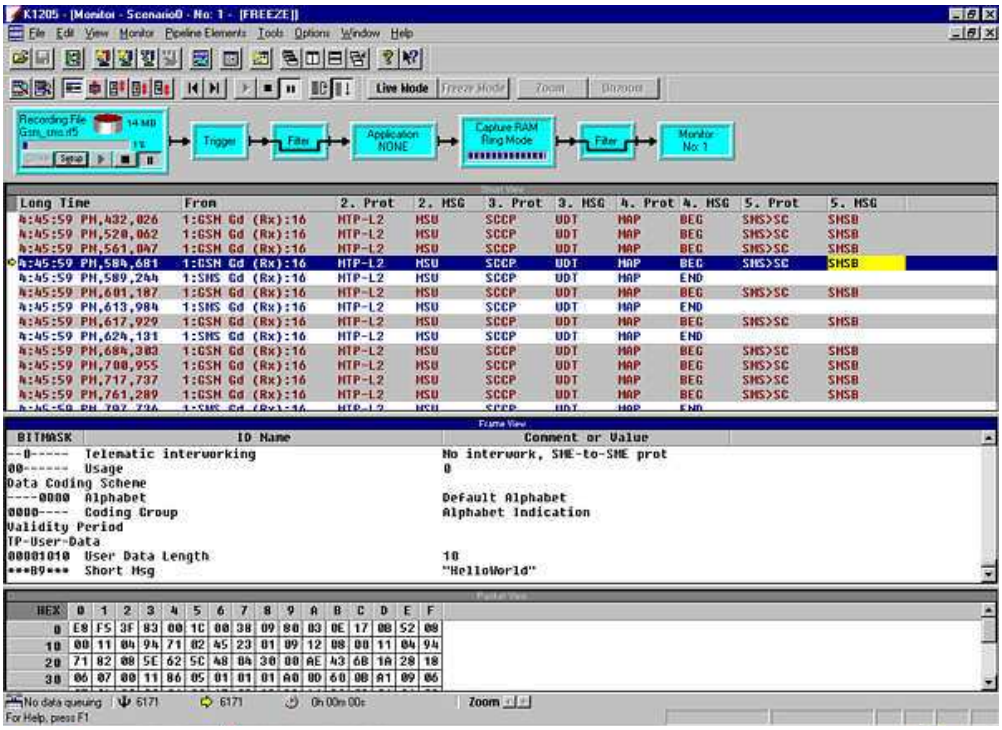
2.1.4 Tektronix K1297-G35 Protocol Analyzer

Samostatne predávaný protokolový analyzátor určený hlavne pre vývojárov mobilných zariadení. Primárne slúži na testovanie kompatibility vyvíjaných riešení. Medzi jeho vlastnosti patrí kompletná analýza, simulácia, a emulácia záťaže siete. Podpora aj pre 3G (UMTS) a 4G (CDMS, WiMAX, LTE) siete. Možnosť tvorby testovacích scenárov.

Obrázok 5. Tektronix K1297-G35

Existuje mobilná ako aj v racku vstavaná verzia určená na priebežné analýzy. Pre príklad systém dokáže analyzovať dáta na nasledujúcich komunikačných rozhraniach:

- **UTRAN:** Iub / Iur Interface
- **2G/3G Core Network:** A, C, D, E, H, Gb(IP), Gb(FR), Gn, Gi, Gs, Iu-CS (ATM/IP), Iu-PS, (ATM/IP), Nb, Nc, Mc
- **CDMA, CDMA2000:** A1, A3, A7, A8, A9, A11, A13
- **IMS:** Gm, Go, Gq, Cx, Dx, Mb, Mg, Mm, Mr, Mw, Mn, Ut
- **WiMAX:** R1, R3, R4, R6, R8
- **LTE:** Uu, S1, X2



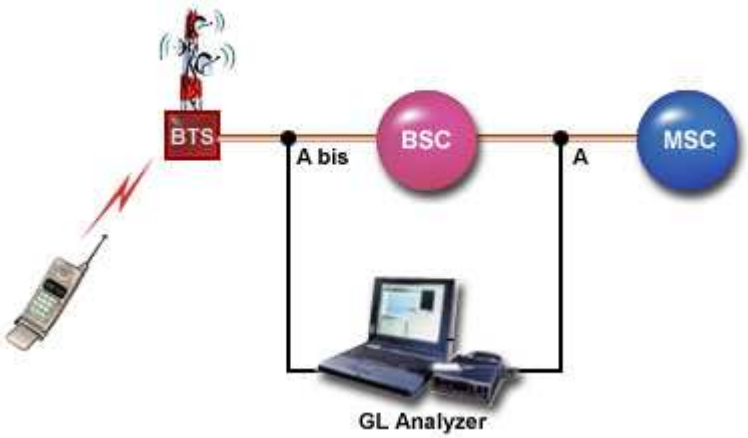
Obrázok 6. Tektronix K1297 monitor

Medzi výhody tohto riešenia patrí jeho podpora viacerých štandardov. Z datasheetu vidno prepracovanosť daného systému (možno je to len šikovnosť marketingového oddelenia).

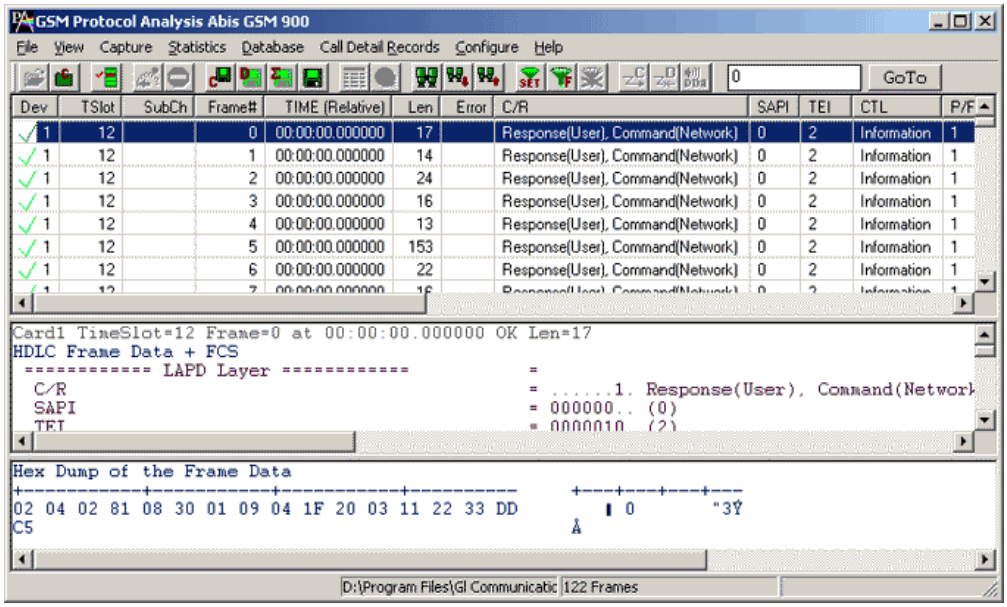
Nevýhodou tohto riešenia môže byť veľká robustnosť. V prípade ak niekto chce testovať iba GSM zariadenia, nepotrebuje podporu pre ostatné rozhrania.

2.1.5 GL Comunications riešenia

Firma GL Communications sa zaoberá tvorbou riešení určených pre komplexný dohľad nad sieťami. Jednotlivé produkty má rozdelené do samostatných modulov, vďaka čomu daný systém dokáže poskytnúť riešenie na mieru pre každého zákazníka. Ako príklad môžeme použiť GSM Protocol Analyzer.



Obrázok 7. Zapojenie GL GSM Protocol Analyzer



Obrázok 8. Obrazovka GL GSM Protocol Analyzer

Analyzátor sa pripája na rozhranie Abis a A. Dokáže analyzovať nasledujúce protokoly:

- **A Interface** (varianty GSM 900, DCS 1800, PCS 1900) – MTP2, MTP3, SCCP, BSSMAP, MM, CC, SMS
- **Abis Interface** (varianty GSM 900, DCS 1800, PCS 1900) - LAPD, BTSM, RR, MM, CC, SMS
- **Gs Interface** (varianty GSM 900, DCS 1800, PCS 1900) – MTP2, MTP3, SCCP, BSSAP+, RRLP, BSSLAP, SMLCPP, LLP, BSSAP-LE (BSSMAP-LE/DTAP-LE), SCCP, MTP3, MTP2
- **Up Interface** – UMA, TCP, UDP, IP, MAC

Z výpisu protokolov vidno že sa jedná len o GSM protokoly. Modul GSM analyzátora obsahuje len najnutnejšie protokoly a algoritmy čo isto znižuje cenu daného modulu. Medzi výhody riešenia patrí dobré rozdelenie na dostatočne malé moduly. Medzi nevýhody môže patriť obmedzená prepojiteľnosť analyzátora iba na linky typu T1/E1/Ethernet.

3 Spôsoby definície protokolov

Sieťový protokol definuje pravidlá a konvencie pre komunikáciu medzi sieťovými zariadeniami. Vo všeobecnosti, protokoly pre počítačové siete využívajú techniky prepínania paketov na odosielanie a prijímanie správ vo forme dátových jednotiek paketov.

Sieťové protokoly zahŕňajú mechanizmy na identifikáciu zariadení v sieti, identifikáciu spojení, rovnako ako aj formálne pravidlá, ktoré určujú, ako budú dáta zabalené do odosielaných a prijímaných správ. Niektoré protokoly tiež podporujú kompresiu a potvrdzovanie správ určené pre spoľahlivú a/alebo vysoko výkonnú sieťovú komunikáciu. Existujú stovky sieťových protokolov, ktoré boli vyvinuté na rôzne účely v špecifických prostrediach.

Zariadenia v sieti vzájomne komunikujú pomocou definovaných správ cez komunikačný kanál. Povaha tejto výmeny definuje sieťový protokol, ktorého dôležitou časťou je definícia formátu prenášaných správ. Pravidlá definujúce špecifikáciu sieťového protokolu sú implementované v komunikujúcich zariadeniach. Čiastkovou úlohou tento práce je definovanie abstraktných pravidiel definujúcich formalizmus určený na definovanie analyzovaných protokolov.

V súčasnej dobe sa sieťové protokoly definujú viacerými metódami. Internetové protokoly sú definované pomocou RFC dokumentov, ďalšie protokoly sú definované štandardom ASN.1 alebo vlastnými štandardmi ako napríklad protokoly GSM rodiny.

3.1 RFC dokumenty

Dokumenty RFC (Request for comment) sú základné technické dokumenty ktoré sa zabývajú najrôznejšími aspektmi fingovania sieťovej komunikácie vrátane definície časti sieťových protokolov. Nemajú formu záväzných noriem ale doporučení. Niektoré z nich, hlavne tie ktoré definujú protokoly, majú povahu skutočných a záväzných štandardov. Pretože ako v prostredí internetu, tak i mimo neho sú všeobecne uznávané, dodržované a rešpektované. Už vydaný dokument sa nikdy nemení, v prípade zmeny protokolu sa vydá nový dokument, ktorý označí ten predchádzajúci za zastaraný.

Sú to:

- Oficiálne dokumenty ktoré definujú fungovanie internetu
- Niektoré popisujú sieťové štandardy, niektoré sú informatívne
- Sú to dokumenty vydávané pre potreby ďalšieho rozvoja internetu
- Dokumenty písané odbornou angličtinou

3.1.1 IP protokol

Príkladom protokolu špecifikovaného pomocou RFC dokumentu je IP protokol. Bol vyvinutý v 70. rokoch pre vládnu vojenskú agentúru USA (DARPA). Cieľom bolo prepojiť rôznorodé počítače vojenských, obranných a výskumných pracovísk a univerzít. Architektúra TCP/IP bola odvodená z

prvej paketovej siete na svete z r. 1969 ARPANET. Tato architektúra dosiahla takej obľuby, že aj keď nie je priemyselným štandardom, sa rýchlo rozšírila do celého sveta a stala sa základom celosvetovej siete INTERNET.

Každý IP paket sa skladá z hlavičky a dát. Všetky súčasti siete - ako sú servery, pracovné stanice a sieťové zariadenia - musia mať aspoň jeden jednoznačný identifikátor, IP adresu. V terajšej prevažujúcej verzii 4 protokolu IP (IPv4) sú IP adresy 32bitové celé čísla. Kvôli lepšej čitateľnosti sú IP adresy zapisované do formátu s desatinou bodkou, tj. ako skupiny štvor čísiel, z ktorých každé môže mať hodnotu od 0 do 255, ktoré sú oddelené bodkami, napríklad 127.0.0.1. Polia v hlavičke IP, používané pre určenie zdroja a cieľa, obsahujú IP adresy.

RFC dokument definujúci IP protokol, ďalej obsahuje presne definície a významy obsahu voliteľných polí. Nie je cieľom tejto práce špecifikovať nejaké protokoly, IP protokol som vybral ako príklad protokolu špecifikovaného pomocou RFC dokumentov.

3.2 Štandard ASN.1

Štandard ASN.1 umožňuje definovať komplexné dátové štruktúry nezávisle na programovacom jazyku. Štandard poskytuje rad pravidiel na popis štruktúry objektov, ktorý je nezávislý na použitej platforme a kódovaní.

3.2.1 X.509

X.509 je štandard definujúci infraštruktúru verejného kľúča (PKI). Špecifikuje formáty pre certifikát verejného kľúča, zoznam zrušených certifikátov, atribúty certifikátov a cestu overenia certifikátu. Digitálne certifikáty slúžia na overovanie identity vlastníkov šifrovacích kľúčov. X.509 je v súčasnej dobe jedným z najpoužívanejších typov certifikátov. X.509 je definovaný dokumentom RFC 3280, avšak na popis štruktúry dát, je použitá notácia ASN.1.

ASN.1 popis certifikátu X.509:

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  EXPLICIT Extensions OPTIONAL
                          -- If present, version MUST be v3
```

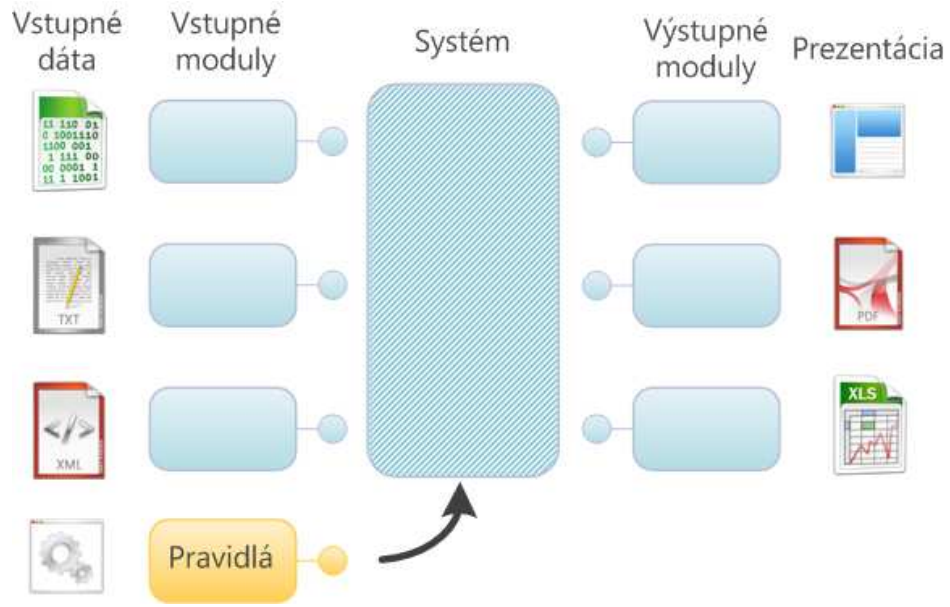
```
        }

    Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

    CertificateSerialNumber  ::=  INTEGER

    Validity ::= SEQUENCE {
         notBefore      Time,
         notAfter       Time }

    Time ::= CHOICE {
         utcTime        UTCTime,
         generalTime    GeneralizedTime }

    UniqueIdentifier  ::=  BIT STRING

    SubjectPublicKeyInfo  ::=  SEQUENCE  {
         algorithm            AlgorithmIdentifier,
         subjectPublicKey     BIT STRING  }

    Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

    Extension  ::=  SEQUENCE  {
         extnID      OBJECT IDENTIFIER,
         critical    BOOLEAN DEFAULT FALSE,
         extnValue   OCTET STRING  }
```

3.3 Definícia vlastným štandardom

Ďalšou z možností definície protokolu je definícia v samostatnom štandarde. Príkladom takejto definície je GSM štandard. Podobne ako IP protokol v RFC je písaný odbornou angličtinou. Príklad protokolu definovaného GSM štandardom uvediem v samostatnej kapitole o GSM.

4 Špecifikácia systému

Navrhovaný systém má slúžiť na všeobecnú analýzu dát rôzneho druhu. Je treba zabezpečiť jednoduchú rozšíriteľnosť analyzovaných dát a databázy známych protokolov. Systém má podporovať pravidlá určené na definíciu štruktúry prezentovaných dát. Nasledujúci obrázok ukazuje základný prehľad systému.



Obrázok 9. Prehľad systému

Načítanie vstupných dát je oddelené do zásuvných modulov z dôvodu kompatibility s rôznymi formátmi analyzovaných dát. Príkladom môže byť textový alebo xml súbor s predspracovanými dátami, načítavanie dát priamo zo sieťovej karty počítača a iné. Keďže je predmetom tejto práce vytvorenie všeobecného analyzátora, je načítavanie dát riešené formou ľahko programovateľných zásuvných modulov.

Modul označený „.pravidlá“ obsahuje objektový model generovaný z XML popisu štruktúry jednotlivých protokolov. Vyčlenenie popisov jednotlivých protokolov do externého XML súboru umožňuje užívateľovi jednoducho dynamicky za behu programu zmeniť definičnú sadu analyzovatelných dát. To znamená, že na pridanie nového protokolu do sady rozpoznávaných, netreba prekompilovať celý analyzátor, ale stačí predefinovať sadu analyzovatelných protokolov v XML. Dané riešenie zlepšuje rozšíriteľnosť a používateľnosť systému.

Výstup systému je taktiež riešený pomocou zásuvných výstupných modulov ktoré sú volané z renderovacieho jadra. Systém zásuvných modulov poskytuje možnosť definovať exportné moduly pre ľubovoľné prezentačné formáty (TXT, HTML, XML ...).

4.1 Prípady a scenáre použitia

Navrhnutý systém slúži na prezentáciu dát v ľudsky čitateľnejšej a prehľadnejšej forme. Prezentácia prebieha v programovom prehliadači, alebo v užívateľom definovaných exportných moduloch. Použitie systému je rozdelené do troch krokov: výber zdroja dát, definícia pravidiel zobrazovaných dát, výber výstupu prezentujúceho zvolené dáta.

Systém primárne slúži na prezentáciu dát prenášaných po počítačovej sieti, paketov. Pakety sú význačné použitím vrstvového modelu, ktorý je špecifický tým, že protokol jednej vrstvy môže obsahovať vnorený protokol vyššej vrstvy.

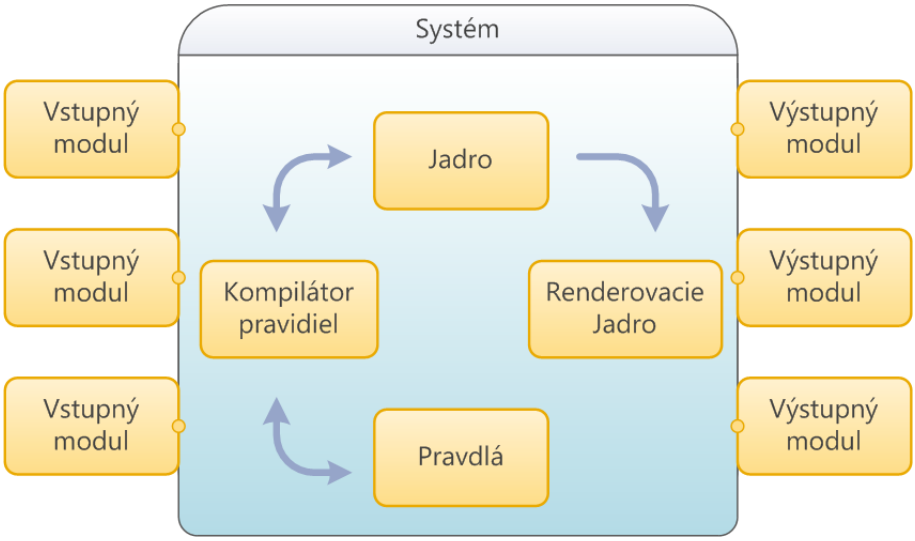
Ďalším využitím systému je prezentácia ľubovoľnej štruktúrovanej sady dát, ktorou môže byť napríklad výpis z behu určitého programu, alebo systémový log.

Oproti existujúcim riešenia navrhnutý systém prináša nasledujúce výhody:

- Dynamická a počas behu programu meniteľná štruktúra pravidiel
- Rozšíriteľnosť zdrojov vstupných dát pomocou zásuvným modulov
- Rozšíriteľnosť generovaných výstupov pomocou zásuvných modulov
- Rýchlosť práce s dátami dosiahnutá dynamicky generovaným DOM rozpoznávacím modulom

5 Vlastný návrh a implementácia

Systém sa skladá zo šiestich základných modulov. Ktorými sú Jadro, Kompilátor pravidiel, Rozhodovacie pravidlá, Renderovacie jadro, Systém vstupných modulov a systém výstupných modulov. Nasledujúci obrázok ukazuje detail modulárneho riešenia systému. Realizácia zdrojov dát a modulov výstupu dát formou zásuvných modulov umožňuje jednoduché rozšírenie množiny podporovaných formátov vstupných a generovaných dát.



Obrázok 10. Detail modulárneho riešenia systému

Jadro systému obsahuje základné funkcie viažuce systém dokopy. Stará sa o načítanie vstupných a výstupných modulov do pamäte, volanie kompilátora pravidiel a volanie renderovacieho jadra. Jadro taktiež obsahuje objektový model zobrazovaných dát.

Pravidlá definujú štruktúru rozpoznávaných protokolov. V systéme je použitá metóda dynamicky generovaného objektového modelu, ktorá zvyšuje výkonnosť systému. Jazyk pravidiel definuje META model popísaný v kapitole 5.1. Kapitola 0 detailne opisuje návrh a implementáciu kompilátora pravidiel.

Riešenie vstupných modulov pomocou zásuvných modulov poskytuje možnosť čítania dát z ľubovoľného zdroja pre ktorý existuje vstupný modul. Systém vstupných modulov je bližšie opísaný v kapitole 5.3 .

Renderovacie jadro a výstupné moduly umožňujú export výstupov systému do ľubovolných formátov pre ktoré existujú exportné moduly. Systém je navrhnutý tak, aby nebolo moc zložité doprogramovať ďalší výstupný modul. Renderovacie jadro a systém výstupných modulov je bližšie popísaný v kapitole 5.4.

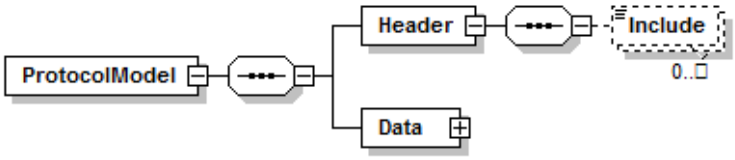
5.1 Metamodel protokolov

Táto kapitola sa popisuje systém pravidiel a metamodel slúžiaci na modelovanie štruktúry spracovávaných dát, protokolov. Metamodel vychádza zo štandardu ASN.1, avšak nejedná sa o štandard ASN.1. Nevýhodou vlastného metamodelu je vlastná implementácia a nemožnosť použiť už existujúce modely, avšak výhodou vlastného metamodelu je jednoduchšia možnosť rozširovania a úprav modelovaných dát.

Užívateľ pomocou vstavaného editora protokolov edituje súbory definícií protokolov (modely). Jednotlivé modely sú uložené vo formáte XML a sú po editácií alebo pri každom spustení programu použité na vygenerovanie modulu systému označeného „.pravidlá“. Nasleduje popis jednotlivých komponentov metamodelu.

5.1.1 ProtocolModel

Root elementom celého metamodelu je element ProtocolModel. ProtocolModel obsahuje graf modelu. V hlavičke modelu (Header) sa môžu nachádzať odkazy na vložené modely (Include). Odkaz na vložený model má formu absolútnej adresy súboru alebo relatívnej adresy k modelu od programového adresára.

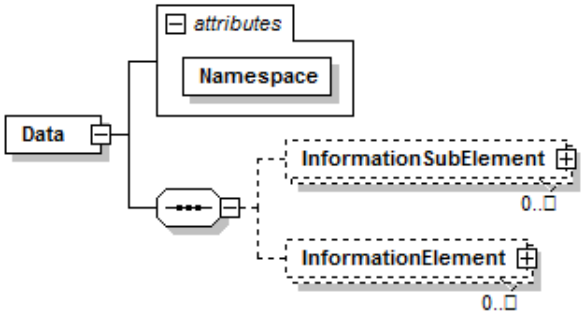


Obrázok 11. ProtocolModel

Ako vidno na obrázku model môže odkazovať na 0 až N ďalších modelov. Odkazovanie na existujúce modely umožňuje znovu použite existujúcich definícii z vložených modelov.

5.1.2 Data

Element obsahujúci dátovú časť modelu sa nazýva Data. Aby sa zamedzilo konfliktom v názvoch generovaných tried, tak každý model sa zaraďuje do menného priestoru (Namespace). Každý model môže obsahovať iba jeden menný priestor, kombinácia viacerých menných priestorov sa dá vytvoriť pomocou vložených odkazov na ďalšie modely v elemente ProtocolModel/Header

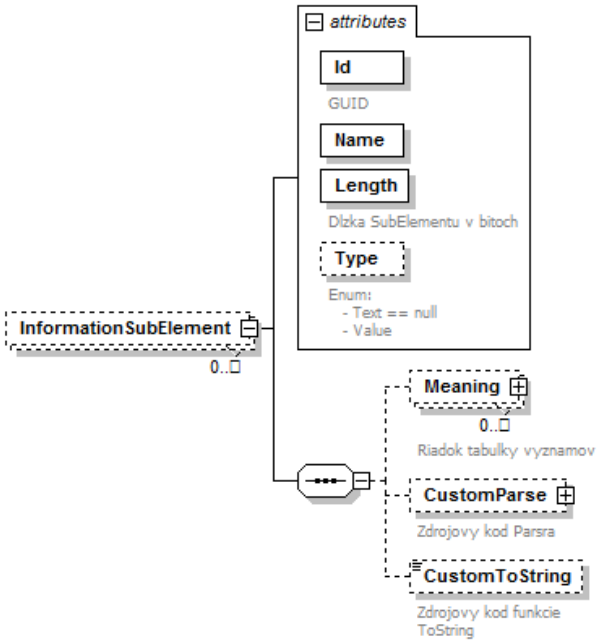


Obrázok 12. Data

Obrázok 16 ukazuje, že Data môžu obsahovať sekvenciu elementov typu InformationSubElement a InformationElement. InformationSubElement a InformationElement sú pomocou kompilátora skompilované do tried použitých v module „Pravidlá“. InformationSubElement opisuje základnú nedeliteľnú jednotku popisu štruktúry dát / protokolu a je popísaný v kapitole 5.1.3. InformationElement definuje vyššiu, štruktúrovanú jednotku popisu štruktúry dát / protokolu a je popísaný v kapitole 5.1.4.

5.1.3 InformationSubElement

InformationSubElement definuje základnú nedeliteľnú jednotku popisu štruktúry dát / protokolu opisujúcu význam dát. Generované triedy priraďujú textový význam príslušným dátam. Štruktúra InformationSubElementu je na nasledujúcom obrázku.



Obrázok 13. InformationSubElement

Každý InformationSubElement obsahuje jedinečný Guid identifikátor `Id`, ktorý slúži na referencovanie objektu počas kompilácie. Dĺžka v bitoch ktorá môže byť aj nulová je uložená v premennej `Length`. Podľa spôsobu prezentácie sa InformationSubElementy delia na dve skupiny a to na textové a hodnotové. Textový InformationSubElement na svoju prezentáciu využíva text priradený k hodnote pomocou `Meaning` elementov, hodnotový InformationSubElement využíva priamo hodnotu príslušných dát. InformationSubElement môže obsahovať 0 až N významov (`Meaning`) z ktorých sa vygeneruje parsovacia funkcia. Ak sa jedná o InformationSubElement so zložitejšou štruktúrou je možné nadefinovať vlastnú parsovaciu funkciu `CustomParse` a vlastnú funkciu definujúcu význam `CustomToString`, V tomto prípade sa neuvádzajú jednotlivé významy pomocou `Meaning` elementov.

Počas kompilácie sa každý InformationSubElement skompiluje na samostatnú triedu s menom definovaným v atribúte `Name`. `Name` je taktiež použité na pomenovanie pomocných premenných vo vygenerovaných InformationElementoch.

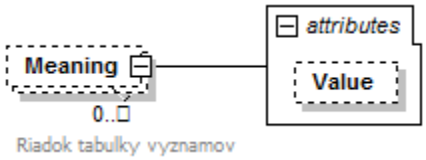
```
<InformationSubElement Id="{80d53397-8975-4e45-8e15-156943d3707f}"
Name="RR Message Type" Length="8">
      <Meaning Value="0x0000003C">RR INITIALISATION REQUEST</Meaning>
      <Meaning Value="0x0000003B">ADDITIONAL ASSIGNMENT</Meaning>
      <Meaning Value="0x0000003F">IMMEDIATE ASSIGNMENT</Meaning>
      <Meaning Value="0x00000039">IMMEDIATE ASSIGNMENT EXTENDED</Meaning>
      <Meaning Value="0x0000003A">IMMEDIATE ASSIGNMENT REJECT</Meaning>
      <Meaning>other RR message</Meaning>
</InformationSubElement>
```

Obrázok 14. XML popis InformationSubElement

Na obrázku 14 je príklad XML popisu InformationSubElementu RR Message Type. Význam a použitie tohto InformationSubElementu sú bližšie opísané v kapitole 6.1.

5.1.3.1 Meaning

Meaning definuje význam hodnoty InformationSubElementu. Jeden InformationSubElement môže obsahovať 0 až N významov. Atribút `Value` definuje hodnotu priradenú k textovému významu. Ak Meaning neobsahuje atribút `Value`, tak sa jedná o default Meaning ktorý je priradený InformationElementu ak nie je priradený žiaden z iných významov.



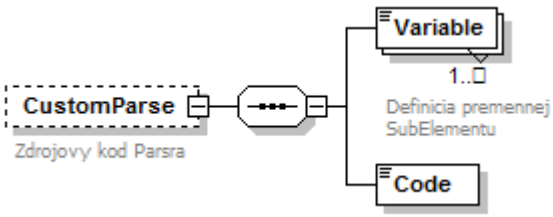
Obrázok 15. Meaning

Všetky Meaningy InformationSubElementu sa skompilujú do metódy `ParseData()`. Pred kompiláciou sa jednotlivé významy zoradia podľa hodnoty atribútu `Value`, tak aby sa implicitný default Meaning nastavil iba v prípade nenastavenie žiadneho iného významu. Počas kompilácie sa pre každý jeden Meaning vytvorí podmienka testujúca hodnotu InformationSubElementu.

Príklad XML popisu meaningu na obrázku 14 ukazuje použitie Meaningu s priradenou hodnotou Value, ako aj default Meaning bez priradenej hodnoty.

5.1.3.2 CustomParse

Druhou možnosťou definície významu InformationSubElementu je použitie elementu CustomParse. Element CustomParse obsahuje skupinu elementov Variable, a Element Code. Všetky elementy sú povinné.



Obrázok 16. CustomParse

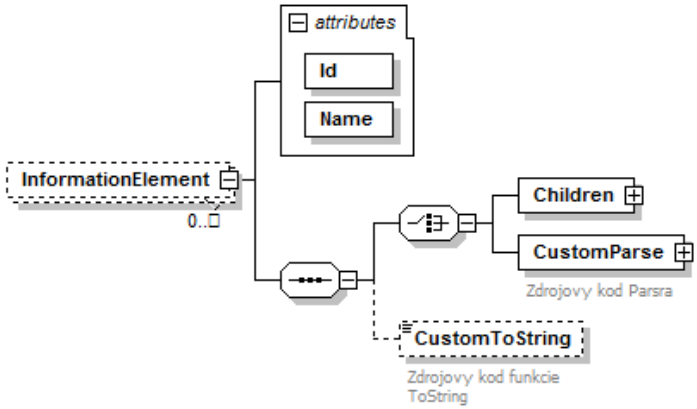
Ako vidno na obrázku, `CustomParse` musí obsahovať aspoň jeden element `Variable`. Z elementov `Variable` sa počas kompilácie vygenerujú lokálne premenné a verejné parametre s typom a menom definovaným v príslušnom `Variable` elemente. Element `Code` obsahuje zdrojový kód funkcie `CustomParse`, ktorý sa počas kompilácie nakopíruje do tela generovanej funkcie `ParseData`(). Kód sa ukladá vo formáte nepoškodzujúcom formátovanie XML .

5.1.3.3 CustomToString

Element CustomToString obsahuje zdrojový kód funkcie `ToString()` použitej vo vygenerovanej triede. Ak sa daný element nenachádza v InformationSubElemente tak sa použije základná ToString funkcia.

5.1.4 InformationElement

InformationElement definuje vyššiu zloženú jednotku popisu štruktúry dát / protokolu opisujúcu štruktúru dát. InformationElement sa skladá z InformationSubElementov alebo InformationElementov. Generované triedy definujú štruktúru analyzovaných dát. Štruktúra InformationElementu je na nasledujúcom obrázku.



Obrázok 17. InformationElement

Ako vidno na obrázku InformationElement obsahuje jedinečný Guid identifikátor `Id`, ktorý slúži na referencovanie objektu počas kompilácie kedy sa každý InformationElement skompiluje na samostatnú triedu s menom definovaným v atribúte `Name`. `Name` je taktiež použité na pomenovanie pomocných premenných vo vygenerovaných InformationElementoch obsahujúcich daný InformationElement ako vnorený element.

Štruktúra popisovaného InformationElementu je definovaná buď pomocu elementu Children ktorý nižšie popísaným spôsobom definuje postup konštrukcie významu každého InformationElementu, alebo pomocou elementu CustomParse, ktorý pomocou vlastného zdrojového kódu definuje štruktúru InformationElementu. Vždy musí byť použitý len jeden zo spôsobov. Element CustomToString umožňuje zmenu obsahu funkcie ToString daného InformationElementu.

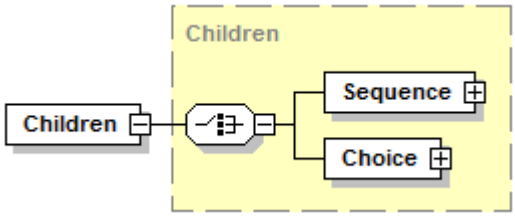
```
<InformationElement Id="{733aaef8-4590-4966-8ff8-8d7b02df0052}" Name="RR">
  <Children>
    <Sequence>
      <SequenceChild SequenceNumber="0">
        <Item Id="{b0629596-046a-4980-bad4-e2c274b7141e}"
ObjectReference="{5c3c42b3-33b6-48d8-988b-26898cd7a2df}" Name="Protocol
Discriminator" />
      </SequenceChild>
    ...
```

Obrázok 18. Príklad začiatku XML popisu InformationElementu

Obrázok 18 zobrazuje začiatok XML popisu InformationElementu RR. Celý príklad protokolu RR aj s detailným popisom významu je uvedený v samostatnej kapitole nižšie.

5.1.4.1 Children

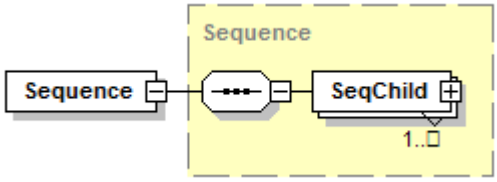
Element Children slúži na definíciu štruktúry dát, ktorá sa používa na automatické generovanie rozpoznávacích funkcií. Na definíciu štruktúry dát sú potrebné dva druhy elementov Sequence a Choice. Sequence definuje sériu za sebou idúcich elementov, Choice definuje výber. Sequence aj Choice sú bližšie opísané v nasledujúcich kapitolách. Ako ukazuje nasledujúci diagram Children element môže obsahovať vždy len jednu z možností - Sequence alebo Choice.



Obrázok 19. Children

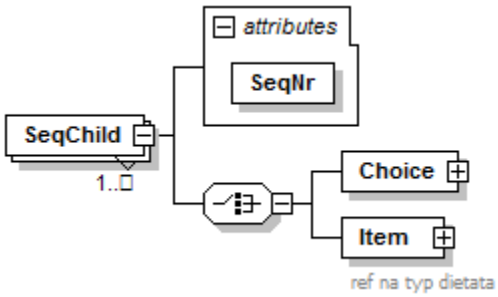
5.1.4.2 Sequence

Sequence definuje sériu za sebou idúcich SequenceChildov ktoré sa skladajú z Choice alebo Itemov odkazujúcich sa na InformationSubElementy a InformationElementy. Nasledujúci obrázok ukazuje štruktúru Sequence elementu.



Obrázok 20. Sequence

Počas kompilácie sa každá sequence najskôr usporiada a vďalšom kroku sa pre každý SequenceChild zavolá príslušná kompilačná funkcia. Keďže pri kompilácii záleží na poradí, SequenceChild obsahuje atribút SeqNr definujúci usporiadanie. Ako vidno na nasledujúcom obrázku SequenceChild môže vždy obsahovať iba Choice alebo Item.

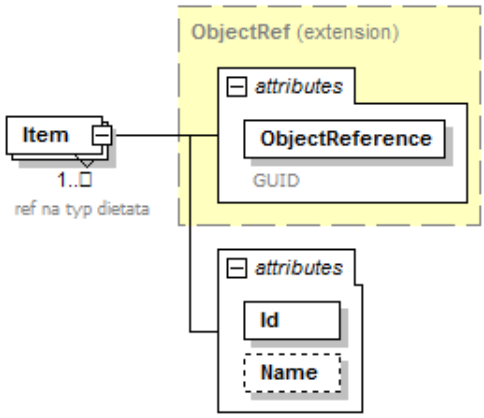


Obrázok 21. SequenceChild

Obrázok 18 ukazuje začiatok XML popisu sekvencie s prvým SequenceChild objektom. Zobrazený SequenceChild obsahuje vnorený Item Protocol Discriminator.

5.1.4.3 Item

Item je základnou stavebnou jednotkou InformationElementu. Obsahuje dva povinné atribúty Id a ObjectReference. Id slúži na referencovanie Itemu pri kompilácii Choice elementov, ObjectReference referencuje InformationElement alebo InformationSubElement definujúci typ daného Itemu.



Obrázok 22. Item

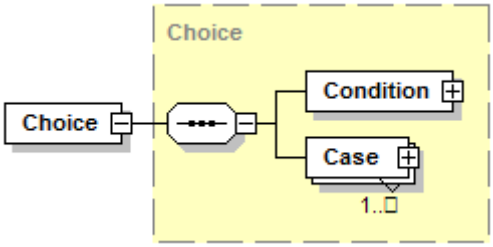
Hore uvedený obrázok ukazuje štruktúru Item elementu. Nižšie uvedený obrázok ukazuje príklad XML popisu Item elementu s menom Message type. ObjectReference ukazuje na InformationSubElement RR Message Type. Proces kompilácie Itemu je popísaný v kapitole 0.

```
<Item Id="{47fec4ef-5ba2-46d9-b4a3-d3526072ad0a}"
ObjectReference="{80d53397-8975-4e45-8e15-156943d3707f}" Name="Message
type" />
```

Obrázok 23. XML popis Item

5.1.4.4 Choice

Choice slúži na definíciu vetvenia zloženého z viacerých možností. Choice element sa skladá z podmienky Condition a množiny 1 až N možností Case. Platný Choice element musí obsahovať práve jednu podmienku Condition a minimálne jednu možnosť Case. Nasledujúci obrázok ukazuje štruktúru elementu Choice.



Obrázok 24. Choice

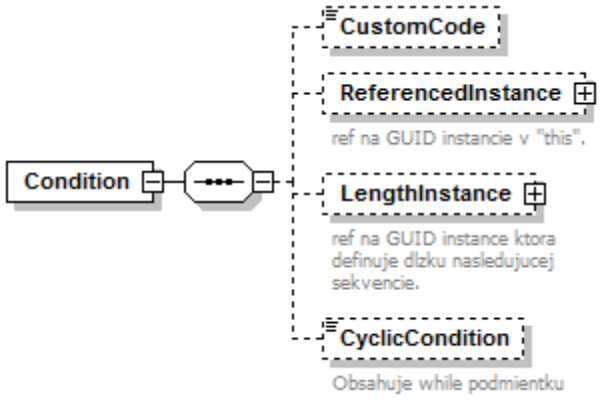
Obrázok 25 zobrazuje časť XML popisu podmienky Choice. Na obrázku je jasne vidno príklad štruktúry Choice skladajúcej sa z podmienky Condition a možnosti Case. Podmienka Condition definuje podmienku pomocou predo definovaného Itemu s Id `{47fec4ef-5ba2-46d9-b4a3-d3526072ad0a}` v rámci ktorého sa kontroluje hodnota vlastnosti Data.

```
<Choice>
  <Condition>
    <ReferencedInstance ObjectReference="{47fec4ef-5ba2-46d9-b4a3-
d3526072ad0a}">
      <SubReference>.Data</SubReference>
    </ReferencedInstance>
  </Condition>
  <Case Value="06">
    <Children>
      <Sequence>
        ...
```

Obrázok 25. Časť XML popisu podmienky Choice

5.1.4.5 Condition

Ako bolo uvedené v predošlej kapitole, každý platný Choice musí obsahovať podmienku Condition. Nasledujúci obrázok ukazuje štruktúru podmienky Condition. Z obrázku sa dá vyčítať, že Condition sa môže skladať z viacerých podmienok. Obrázok 26 ukazuje štruktúru elementu Choice.



Obrázok 26. Condition

Condition môže obsahovať tieto štyri typy podmienok:

- ReferencedInstance
- lengthInstance
- CyclicConditon
- CustomCode

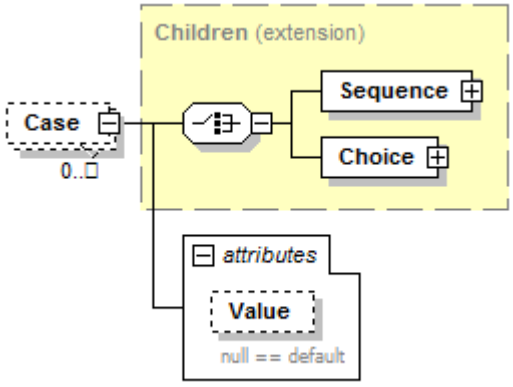
ReferencedInstance definuje podmienku pomocou hodnoty určeného skôr definovaného Itemu. ReferencedInstance obsahuje Guid Itemu ktorý sa má referencovať. Pri kompilovaní sa z ReferencedInstance vytvorí podmienka ktorá sa doplní hodnotami jednotlivých možností (Case).

LengthInstance je špeciálnym prípadom podmienky pri ktorej sa porovnáva dĺžka vloženej sekvencie a hodnota určeného, skôr definovaného Itemu. Počas kompilácie sa využíva iba jedna možnosť Case ktorá definuje štruktúru vložených Itemov.

CyclicCondition a **CustomCode** poskytujú možnosť definovať podmienky pomocou vlastného zdrojového kódu. CyclicCondition je ekvivalent LengthInstance a CustomCode je ekvivalent ReferencedInstance.

5.1.4.6 Case

Case definuje vetvu rozhodovacieho stromu určeného podmienkou Condition. Obrázok 27 znázorňuje štruktúru elementu Case.

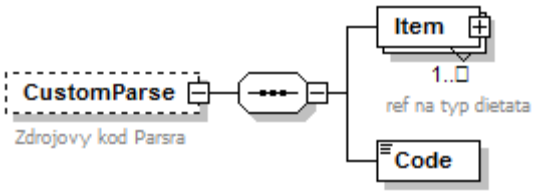


Obrázok 27. Case

Z obrázku vyplýva, že Case môže obsahovať hodnotu a musí obsahovať dieťa (children devinované vyššie). Ak Case neobsahuje hodnotu, tak sa jedná o default Case ktorý je pri kompilácii skompilovaný ako else možnosť, alebo sa jedná o LengthInstance Condition ktorá sa kompiluje podla špeciálnych pravidiel popísaných nižšie.

5.1.4.7 CustomParse

Druhou možnosťou definície významu InformationElementu je použitie elementu CustomParse. Element CustomParse obsahuje skupinu elementov Item, a Element Code. Všetky elementy sú povinné.



Obrázok 28. Custom Parse

Ako vidno na obrázku, `CustomParse` musí obsahovať aspoň jeden element `Item`. Z elementov `item` sa počas kompilácie vygenerujú lokálne premenné a verejné parametre s typom a menom definovaným v príslušnom `Item` elemente. Element `Code` obsahuje zdrojový kód funkcie `CustomParse`, ktorý sa počas kompilácie nakopíruje do tela generovanej funkcie `ParseData`(). Kód sa ukladá vo formáte nepoškodzujúcom formátovanie XML .

5.1.4.8 CustomToString

Element CustomToString obsahuje zdrojový kód funkcie `ToString()` použitej vo vygenerovanej triede. Ak sa daný element nenachádza v InformationElemente tak sa použije základná ToString funkcia.

5.2 Kompilátor

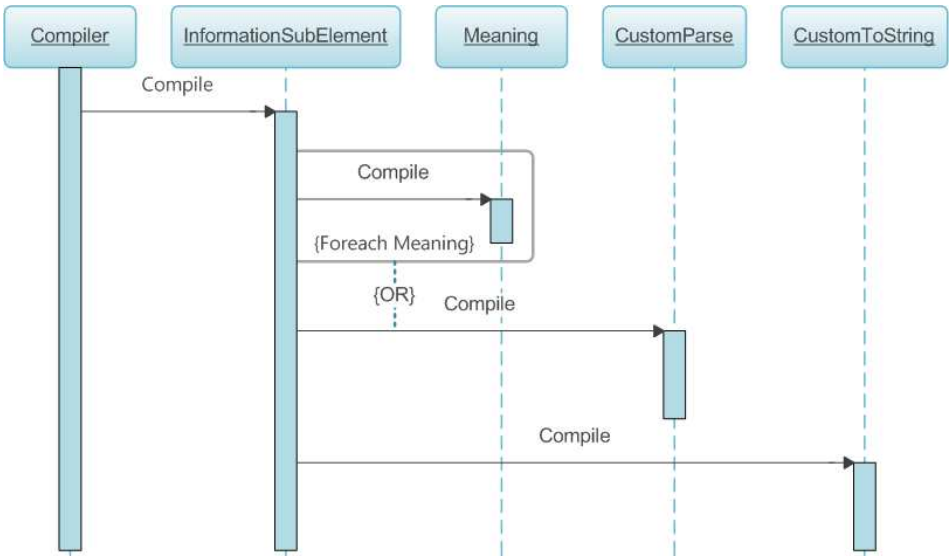
Táto kapitola opisuje kompilátor, slúžiaci na kompiláciu modelov vymodelovaných pomocou vyššie opísaného metamodelu do modulu nazvaného pravidlá, ktorý sa po skompilovaní pripojí k programu ako zásuvný modul znázornený na obrázku 10.

Proces kompilácie pozostáva z vygenerovania zdrojového kódu z modelu popísaného metamodelom z predchádzajúcej kapitoly. Zdrojový kód je generovaný pomocou tried popisujúcich zdrojový kód z menného priestoru `System.CodeDom`. V ďalšom kroku je vygenerovaný zdrojový kód skompilovaný pomocou triedy `Microsoft.CSharp.CSharpCodeProvider` na dynamicky linkovatelnú knižnicu pravidla.dll, ktorá sa po úspešnom skompilovaní pripojí k aktuálne bežiacej inštancii analyzátora.

Nasledujúce podkapitoly podrobne opisujú implementáciu metamodelu a postup kompilácie užívateľom definovaného modelu do dynamicky linkovatelnej knižnice pravidla.dll.

5.2.1 InformationSubElement

InformationSubElement definuje základnú nedeliteľnú jednotku popisu štruktúry dát / protokolu opisujúcu význam dát. Generované triedy priraďujú textový význam príslušným dátam. Nasledujúci diagram zobrazuje postup kompilácie InformationSubElementu.



Obrázok 29. Postup kompilácie InformationSubElement

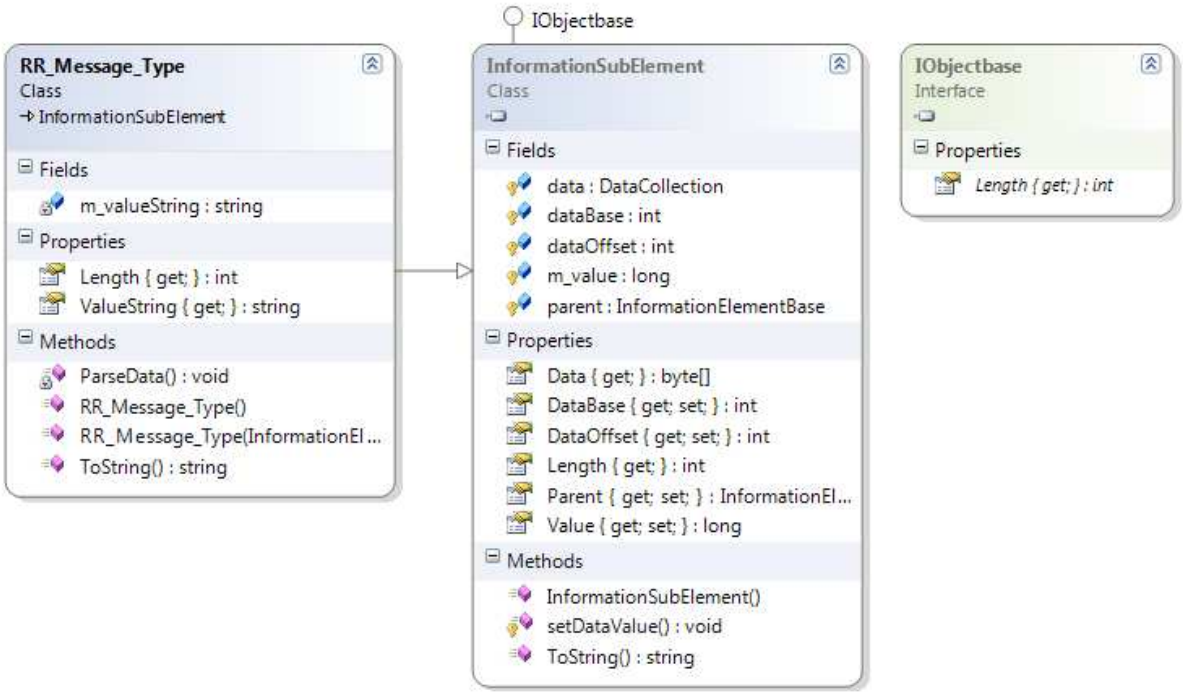
Ako bolo spomenuté skôr, za celú kompiláciu je zodpovedná trieda Compiler. Ak namodelovaný InformationSubElement obsahuje implementáciu CustomParse tak sa použije ona, v opačnom prípade sa metóda ParseData() vygeneruje z nadefinovaných významov (Meaning). Ak namodelovaný informationSubElememt obsahuje implementáciu metódy ToString tak sa použije.

Nasledujúce dva zdrojové kódy ukazujú definíciu významu InformationSubelementu RR MessageType a vygenerovaný zdrojový kód.

```
<InformationSubElement Id="{80d53397-8975-4e45-8e15-156943d3707f}"
Name="RR Message Type" Length="8">
      <Meaning Value="0x0000003C">RR INITIALISATION REQUEST</Meaning>
      <Meaning Value="0x0000003B">ADDITIONAL ASSIGNMENT</Meaning>
      <Meaning Value="0x0000003F">IMMEDIATE ASSIGNMENT</Meaning>
      <Meaning Value="0x00000039">IMMEDIATE ASSIGNMENT EXTENDED</Meaning>
      <Meaning Value="0x0000003A">IMMEDIATE ASSIGNMENT REJECT</Meaning>
      <Meaning>other RR message</Meaning>
</InformationSubElement>
```

Obrázok 30. XML popis RR Message Type

Z vyššie uvedeného XML popisu InformationSubElementu sa vygeneruje trieda `public class RR_Message_Type` dediaca vlastnosti od triedy `InformationSubElement` nachádzajúcej sa v mennom priestore `ObjectModel.InformationElements`. Vygenerovanej triede sa vygeneruje taktiež základný bezparametrický konštruktor, a taktiež konštruktor `public RR_Message_Type(InformationElementBase parent_, int dataBase_, int dataOffset_)` ktorý je používaný v analyzátore. Popis použitia a funkčnosti tohto konštruktora je popísaný v kapitole 5.3 a 5.4 . Nasledujúci diagram ukazuje vygenerovanú triedu v úplnom kontexte dedených tried a rozhraní.



Obrázok 31. Class Diagram vygenerovanej triedy spolu s dedenými triedami

Keďže sa jedná o textový InformationSubElement tak sa po z prvého významu (meaning) vygeneruje nasledujúci kód ktorý priradí text významu do lokálnej premennej m_valueString.

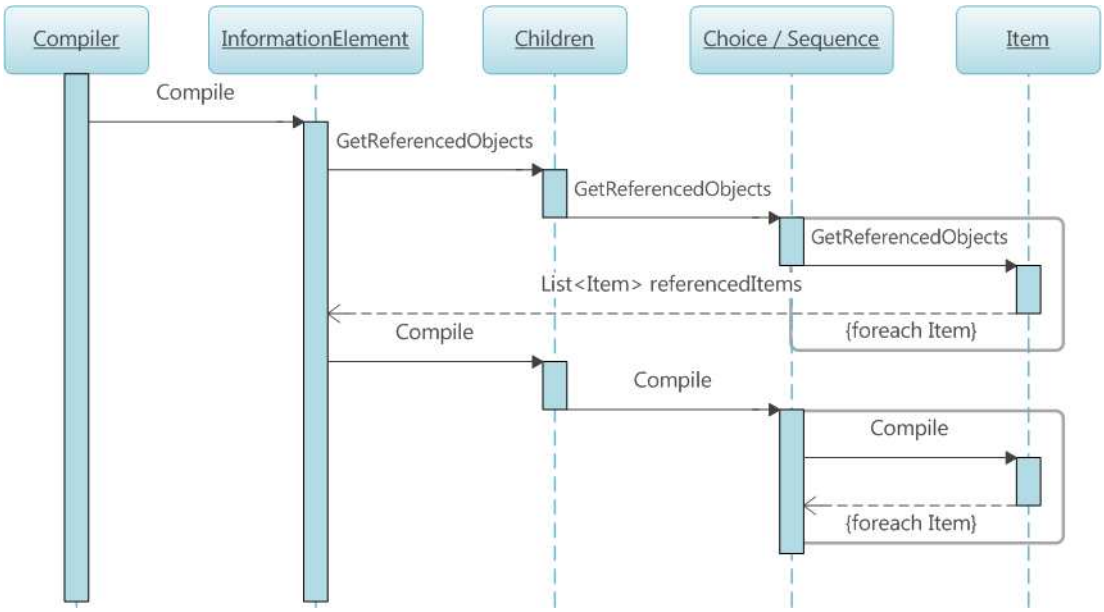
```
if (this.m_value == 57)
{
  this.m_valueString = "IMMEDIATE ASSIGNMENT EXTENDED";
  return;
}
```

Obrázok 32. Vygenerovaný zdrojový kód významu

Hodnotový InformationSubElement neobsahuje lokálnu premennú m_valueString, ale pomocou zmenenej funkcie `public override string ToString()` vracia rovno hodnotu m_value získanú z dát.

5.2.2 InformationElement

InformationElement definuje vyššiu zloženú jednotku popisu štruktúry dát / protokolu opisujúcu štruktúru dát. Kompilácia InformationElementov pozostáva z dvoch krokov. V prvom kroku sa zistia referencované Itemy, v druhom kroku sa spustí kompilácia štruktúry InformationElementu. Nasledujúci diagram znázorňuje postupné volanie oboch krokov kompilácie InformationElementu.



Obrázok 33. Postup kompilácie InformationElement

Ako bolo spomenuté vyššie Children je rekurzívne definované štruktúra ktorej štruktúru ukazuje nasledujúci obrázok. Children môžu obsahovať Sequence alebo Choice. Sequence môže obsahovať Item alebo Choice, Choice môže obsahovať Children (Sequence alebo Choice). Touto rekurziou je ovplyvnená aj kompilácia, pri ktorej sa vytvára jeden súbor so zdrojovým kódom.

Obrázok 34. Štruktúra Children

V prvom kroku kompilácie sa zozbierajú všetky referencované Itemy. To znamená, že sa zozbierajú všetky Itemy na ktoré ukazuje ľubovoľná podmienka Choice.Condition. Zoznam referencovaných Itemov sa využije v druhom kroku kompilácie pri tvorbe lokálnych premenných a to tao, že pre každý referencovaný item sa vygeneruje lokálna premenná v tvare typ lokalnej premennej m_ meno premennej definovane v Item.Name.

Poradie Sequence a Choice elementov určuje štruktúru vygenerovaného kódu. Zo Sequence sa vygeneruje rad za sebou idúcich príkazov vygenerovaných z vnorených objektov Choice alebo Item. Z Choice sa vygeneruje rad podmienok definovaných pomocou podmienky zloženej z Choice.Condition a každého jedného Choice.Case. Výkonnú časť generovaného kódu zabezpečujú skompilované Itemy. Nasleduje príklad XML popisu InformationElementu a príslušného vygenerovaného zdrojového kódu.

```
<InformationElement Id="{733aaef8-4590-4966-8ff8-8d7b02df0052}" Name="RR">
  <Children>
    <Sequence>
      <SequenceChild SequenceNumber="0">
        <Item Id="{b0629596-046a-4980-bad4-e2c274b7141e}"
ObjectReference="{5c3c42b3-33b6-48d8-988b-26898cd7a2df}" Name="Protocol
Discriminator" />
      </SequenceChild>
      ...
```

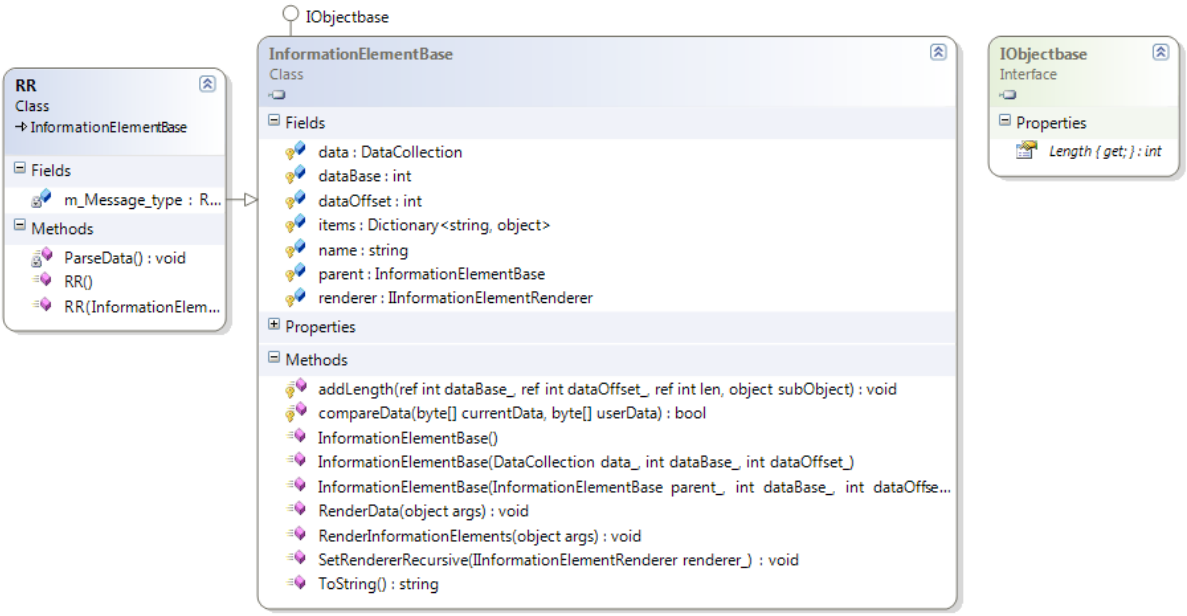
Obrázok 35. Časť XML popisu RR InformationElementu

Z XML popisu uvedeného na obrázku 35 sa vygeneruje trieda `public class RR` dediaca od triedy `InformationElementBase` skompilovanej v projekte ObjectModel. Vo vygenerovanom zdrojovom kóde sa nachádza blok lokálnych premenných, základný bezparametrický konštruktor a taktiež parametrický konštruktor `public RR(InformationElementBase parent_, int dataBase_, int dataOffset_)` používaný v analyzátore, ktorého použitie je opísané v kapitole 5.3 a funkcia ParseData zodpovedná za anayzu dát. Celý XML popis triedy RR aj s celým vygenerovaným zdrojovým kódom sa nachádza v prílohe. Nasledujúci obrázok ukazuje časť vygenerovaného zdrojového kódu obsahujúcu začiatok funkcie Parsedata a skompilovaný Item Protocol Discriminator s Id {b0629596-046a-4980-bad4-e2c274b7141e}. Kompilácia Itemu pozostáva z vygenerovania príkazov vytvorenia pomocnej premennej, naplnenia pomocnej premennej novým objekotm, posunutia pomocného ukazatela aktuálnej pozície a pridania vytvoreného objektu do tabulky lokálnych objektov.

```
private void ParseData()
{
  int tmpBase = this.dataBase;
  int tmpOffset = this.dataOffset;
  int tmpLength = 0;
  // sequence .... .... .... .... ....
  Protocols.GsmTest.Protocol_Discriminator tmp_Protocol_Discriminator;
  tmp_Protocol_Discriminator = new
    Protocols.GsmTest.Protocol_Discriminator(this, tmpBase, tmpOffset);
  this.addLength(ref tmpBase, ref tmpOffset, ref tmpLength,
    tmp_Protocol_Discriminator);
  this.items.Add("Protocol Discriminator", tmp_Protocol_Discriminator);
```

Obrázok 36. Časť vygenerovaného zdrojového kódu

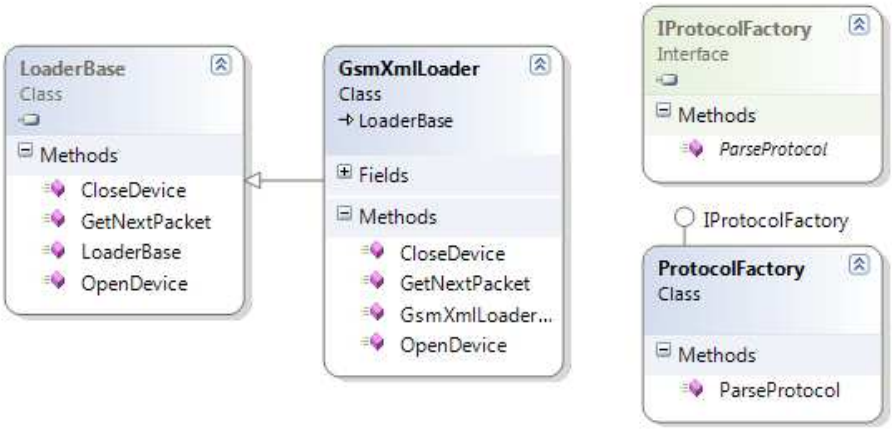
Nasledujúci obrázok ukazuje class diagram vygenerovanej triedy RR spolu s príslušným kontextom dedených tried. Vygenerovaná trieda RR obsahuje lokálnu premennú `private Protocols.GsmTest.RR_Message_Type m_Message_type` použitú pri rozhodovaní o vnorenom protokole. Trieda `InformationElementBase` obsahuje lokálne premenné `DataCollection data` slúžiacu na držanie dátového kontextu t.j. celého spracovávaného bloku dát / paketu, `Dictionary<string, Object> items` slúžiacu na držanie vnorených elementov a protokolov, `Int32 dataBase, dataOffset` slúžiace ako ukazatele do pola dát, ukazujúce na začiatok daného elementu,



Obrázok 37. Class Diagram vygenerovanej triedy RR

5.3 Systém vstupných modulov a analýza dát

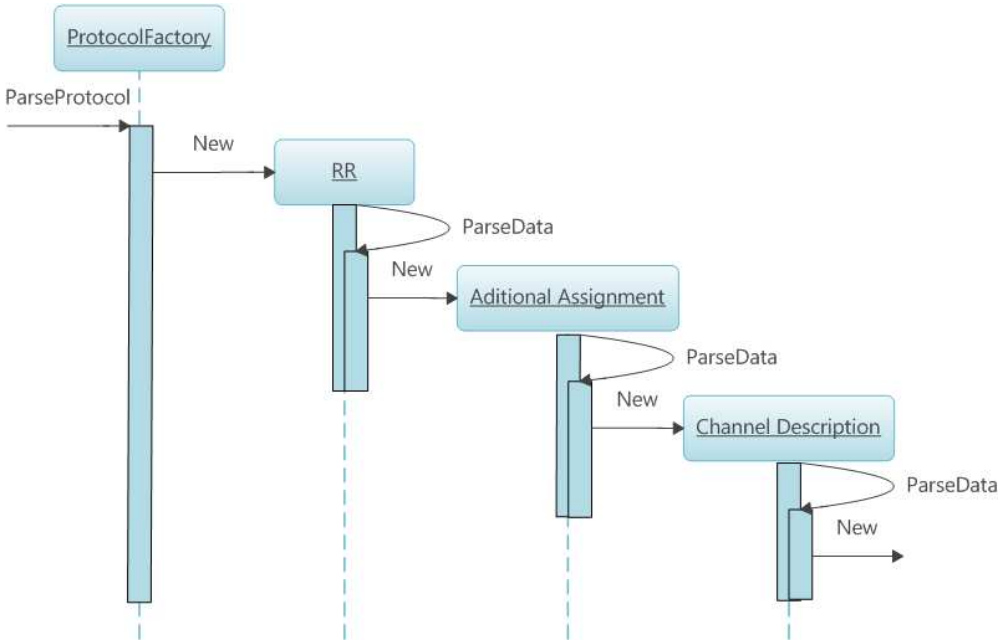
Riešenie vstupných modulov pomocou zásuvných modulov poskytuje možnosť čítania dát z ľubovoľného zdroja pre ktorý existuje vstupný modul. Každý vstupný modul musí obsahovať minimálne dve triedy, jednu slúžiacu na samotné načítavanie dát dediacu od vstupnej triedy LoaderBase a druhú dediacu od rozhrania IProtocolactory slúžiacu na spustenie spracovania dát.



Obrázok 38. Class diagram GsmXmlLoader

Ako vidno na vyššie uvedenom príklade vstupného modulu načítavajúceho dáta uloženej Gsm komunikácie každý vstupný modul musí implementovať tieto funkcie: `public override bool OpenDevice()` slúžiacu na vytvorenie kontextu vstupného modulu, v našom prípade otvorenie spracovávaného súboru, `public override void CloseDevice()` slúžiacu na zatvorenie kontextu vstupného modulu, v našom prípade zatvorenie otvoreného súboru, `public override InformationElementBase GetNextPacket()` slúžiacu na postupné sekvenčné spracovanie celého dátového súboru.

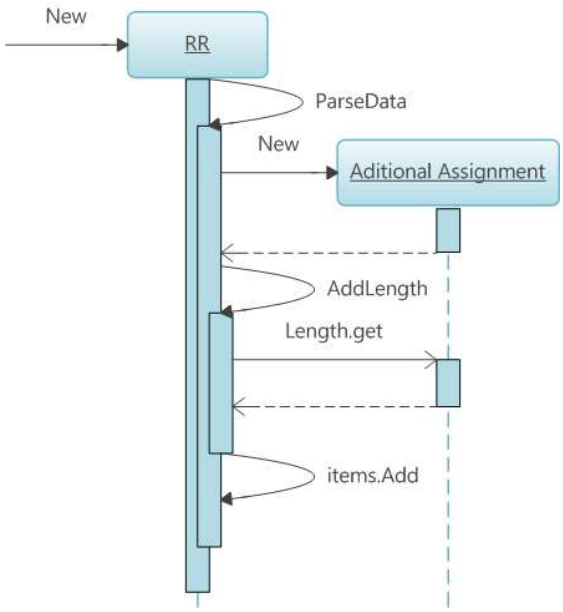
Spracovanie dát z raw formátu do objektového modelu rieši samostatná trieda ProtocolFactory. Keďže rovnaké dáta môžeme dostať z roznych zdrojov, je táto trieda nezávislá od dátového objektu GsmXmlLoader. Trieda ProtocolFactory slúži na priradenie protokolu najnižšej vrstvy k dátam. Celý strom ďalších protokolov zložený z vygenerovaných InformationElementov a InformationSubElememntov sa spracuje automaticky tak ako je nakreslený na nasledujúcom diagrame.



Obrázok 39. Spracovanie dát do objektového modelu

Ako vidno na obrázku 39, ktorý ukazuje príklad spracovania dát do objektového modelu. Trieda ProtocolFactory po rozhodutí, že sa jedná o protokol RR, pomocou svojej funkcie `public void ParseProtocol(InformationElementBase rootProtocol)` vytvorí novú inštanciu triedy RR, ktorá v rámci konštruktora zavolá vygenerovanú funkciu `private void ParseData()` ktorá zavolá konštruktor triedy `new Protocols.GsmTest.Aditional_Assignment(this, tmpBase, tmpOffset).` Parametre konštruktora `tmpBase` a `tmpOffset` ukazujú absolútnu polohu konštruovaného elementu Aditional_Assingnment v poli dát `RR.data`.

Obrázok 40 ukazuje detail volaní vygenerovaného kódu počas konštrukcie objektového modelu analyzovaných dát. Po vytvorení objektu RR sa z konštruktora zavolá funkcia `private void ParseData()` ktorá má na starosti vygenerovanie ďalšej úrovne stromu objektového modelu analyzovaných dát. Analýza dát prebieha v nasledujúcich krokoch. Na začiatku analýzy sa v každom vytvorenom objekte vytvoria pomocne premenné `tmpBase` a `tmpOffset` ukazujúce na aktuálnu polohu „kurzora“ . Vytvorí sa vnorený objekt, v našom príklade objekt Aditional Assignment. Zistí sa dĺžka vytvoreného objektu, a pomocou funkcie AddLength táto dĺžka zmení polohu „kurzora“ `tmpBase` a `tmpOffset`. Nakoniec sa novovytvorený objekt vloží do zoznamu elementov RR.items, ktorý sa využíva na prezentovanie dát.

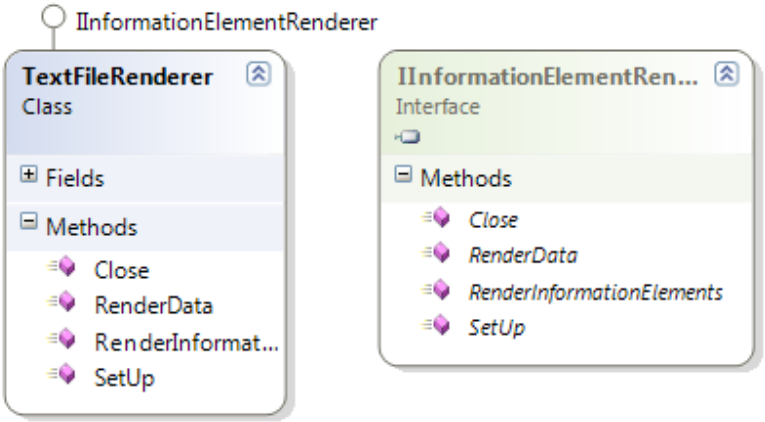


Obrázok 40. Detail vytvorenia nového objektu

5.4 Systém Rendererov a výstupných modulov

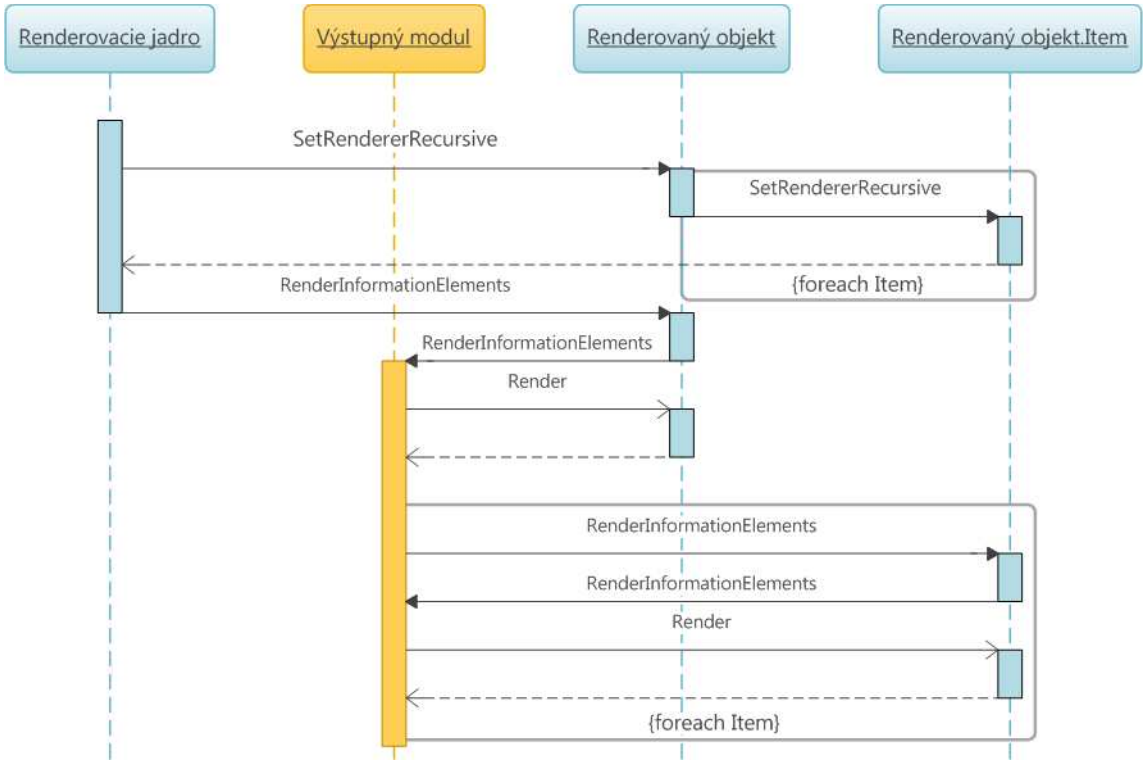
Renderovacie jadro a výstupné moduly umožňujú export výstupov systému do ľubovolných formátov pre ktoré existujú výstupné moduly. Systém je navrhnutý tak, aby nebolo moc zložité doprogramovať ďalší výstupný modul.

Pre príklad si zoberme implementovaný výstupný modul TextFileRenderer. Ako zobrazuje obrázok 41, každý výstupný modul musí implementovať rozhranie `IInformationElementRenderer` definujúce potrebné funkcie. Funkcie `public void SetUp()` a `public void Close()` slúžia na vytvorenie a zrušenie kontextu výstupného modulu, čo v tomto prípade znamená otvorenie a zatvorenie výstupného súboru. Funkcia `public void RenderData(InformationElementBase renderedObject, Object args)` slúži na samostatný hex výstup dát. Funkcia `public void RenderInformationElements (InformationElementBase renderedObject, Object args)` slúži na samotný výstup, volania a logika tejto funkcie sú popísane v nasledujúcom odstavci.



Obrázok 41. Class diagram triedy TextFileRenderer

Nasledujúci obrázok zobrazuje postupnosť volaní jednotlivých funkcií a rozhraní používaných pri výstupe.



Obrázok 42. Výstup rát pomocou výstupných modulov

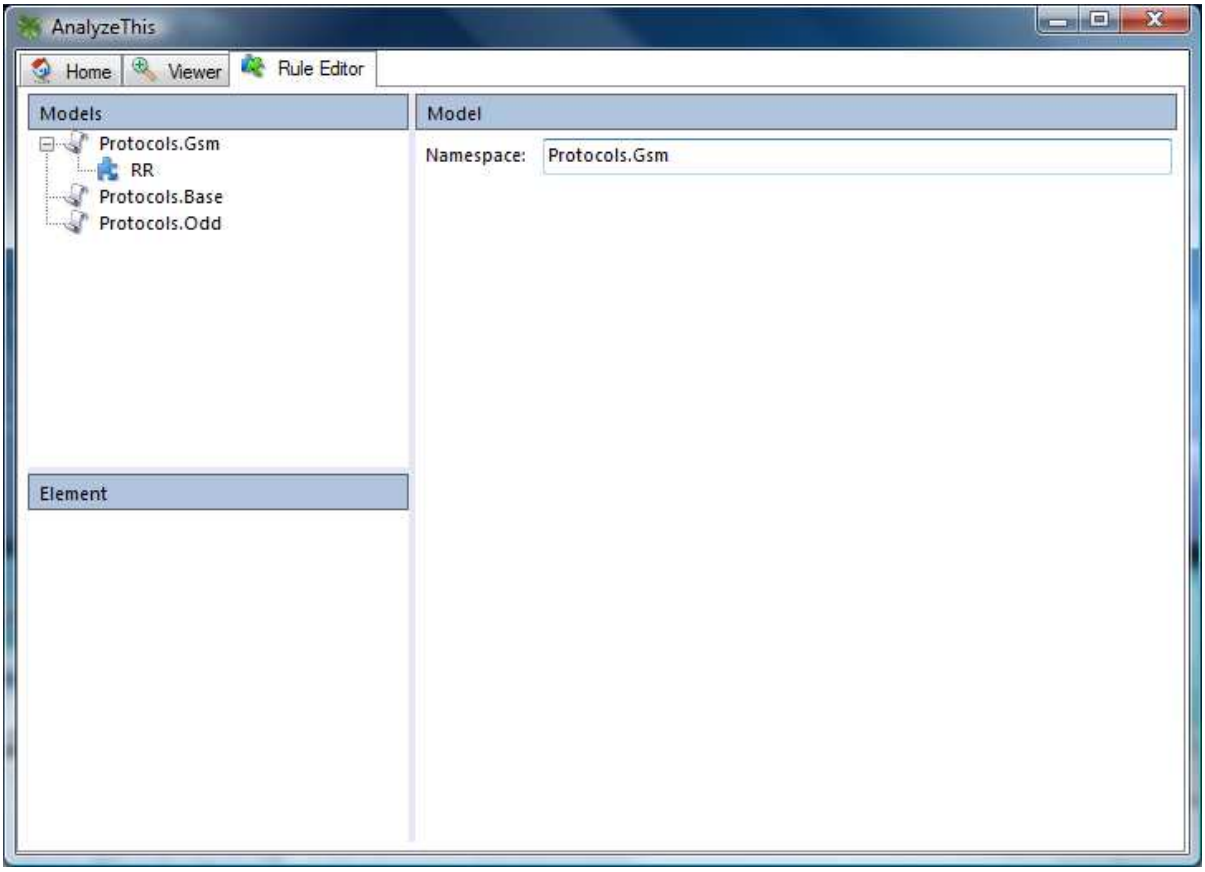
Ako vidno na obrázku 42 výstupný modul sa stará len o spôsob a formát vykresľovania prezentovaných dát. Aby to mohlo celé fungovať tak samotný proces výstupu je rozdelený do dvoch fáz. V prvej fáze, za zaregistruje zvolený výstupný modul do aktuálneho objektového modelu prezentovaných dát.

Samotné vykresľovanie dát prebieha až v druhej fáze. Renderovacie jadro zavolá prezentačnú funkciu `public void RenderInformationElements (Object args)` na každom objekte reprezentujúcom protokol najnižšej vrstvy. Keďže je vykresľovanie oddelené od prezentovaných dát, každý vykresľovaný objekt zavolá vykreslovaciu funkciu zaregistrovaného výstupného modulu `public void RenderInformationElements (InformationElementBase renderedObject, Object args)` ktorá sa postará o vykreslenie daného objektu. Ak vykreslovaný objekt obsahuje vnorené objekty, vykreslovanie jadro zavolá prezentačnú funkciu `public void RenderInformationElements (Object args)` na každom z nich a takto rekurzívne sa vykresli celý objektový model analyzovaných dát.

5.5 Implementácia editora XML pravidiel

Editor XML pravidiel je implementovaný formou WinForms .NET aplikácie, bežiacej na .NET 3.5 frameworku. Celá aplikácia vrátane editora je riešená formou vlastných ovládacích prvkov - výmenných panelov. Každý panel je zodpovedný za modelovanie časti modelu. Po vymodelovaní je možné zmenený model prekompilovať na záložke home.

Na obrázku nižšie je ukážka panelov zobrazených pri editácii elementu Data, ktorý je zodpovedný za definovanie menného priestoru modelu.

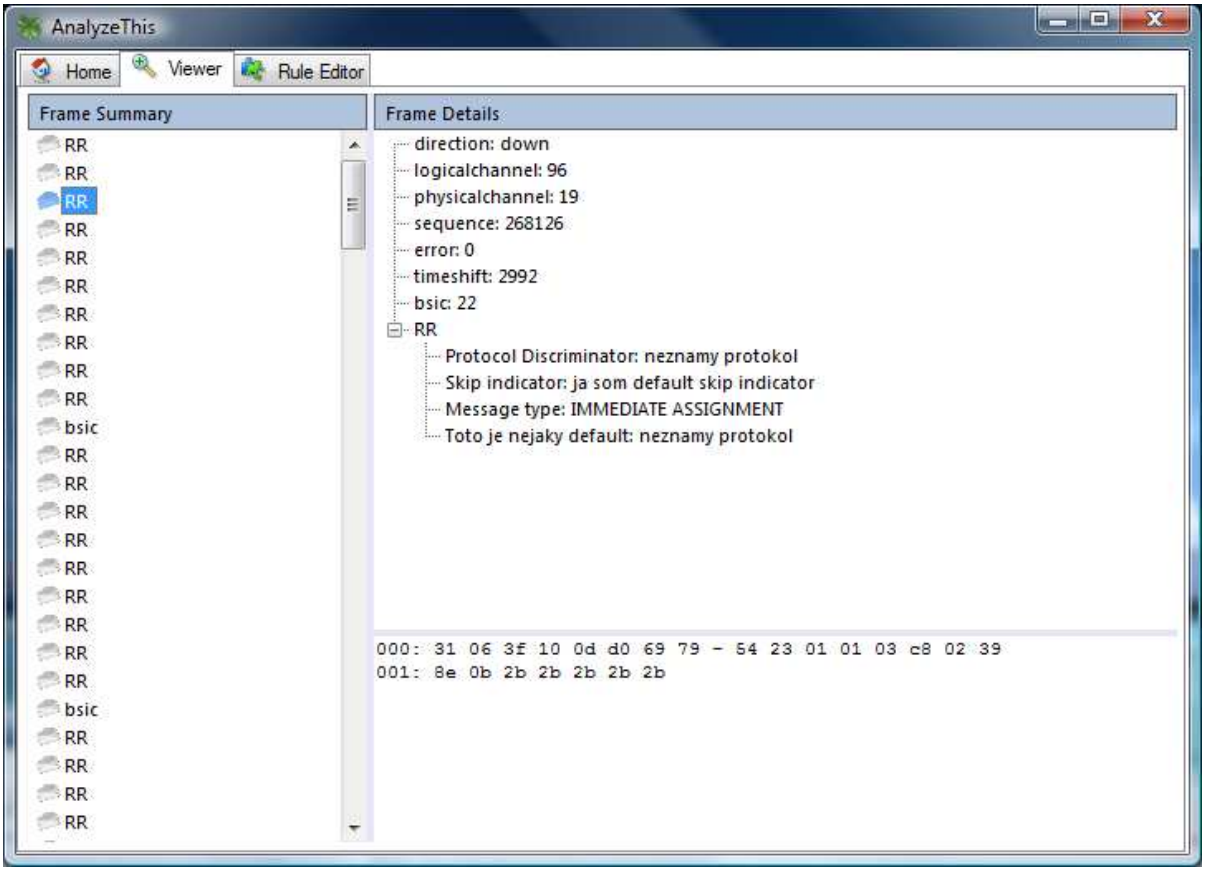


Obrázok 43. Editor pravidiel

5.6 Implementácia prehliadača dát

Prehliadač dát je rovnako ako editor pravidiel implementovaný formou WinForms .NET 3.5 aplikácie používajúcej vymeniteľné panely. Zaujímavosťou je, že panel Frame details zodpovedný za samotné prehliadanie je implementovaný ako výstupný modul volaný z renderovacieho jadra.

Nasledujúci obrázok ukazuje obrazovku prehliadača dát. Panel Frame Details spomenutý v predošlom odstavci je umiestnený v pravej strane obrazovky.



Obrázok 44. Prehliadač dát

6 Zhodnotenie výsledkov práce

Táto kapitola podáva ucelený pohľad na systém ako celok a uvádza kompletný príklad analýzy dát spoločne aj s teoretickým základom potrebným na komplexné pochopenie tématiky.

6.1 GSM

Pojem GSM je dnes už všeobecné známy, veď mobilne siete zaznamenali od svojho vzniku prudký rozvoj. GSM (Global System for Mobile Communications) je v súčasnosti najpopulárnejší svetový štandard pre mobilné telefóny, veď tvorí až 70% svetového trhu v danej oblasti. Telefóny, ktoré využívajú GSM používa viac ako miliarda ľudí a to vo viac ako 200 krajinách na celom svete.

GSM sa od svojich predchodcov výrazne líši v tom, že obidva kanály, signalizačný aj hlasový, sú digitálne, vďaka čomu dostal označenie mobilný systém druhej generácie čiže 2G. Táto technológia pracuje na viacerých frekvenciách. V Európe sú vyčlenené pásma 900MHz a 1800 MHz. V USA a Kanade sa používajú na GSM frekvencie 850MHz a 1900 MHz. V prípade GSM sa jedna o bunkovú sieť, čo znamená, že jednotlivé mobily sa pripájajú na sieť prostredníctvom najbližšej bunky. GSM je otvorený štandard, ktorý vyvíja 3GPP.

Mobilný systém GSM používal v pôvodnej klasické modifikácií frekvenčné pásmo 890 MHz - 960 MHz. Tento variant systému sa označoval ako PGSM (Primary GSM). Neskôr vznikla kvôli stále zvyšujúcim nárokom na počet mobilných účastníkov ďalší variant systému EGSM (Extended GSM), ktorý pre komunikáciu medzi mobilnými a základňovými stanicami používal síce rovnaké frekvenčné pásmo ale mal väčší počet užívateľských kanálov. Oba systémy patrili do skupiny GSM 900. Iné frekvenčné pásmo, ktoré pracuje medzi 1710 MHz až 1880 MHz je GSM 1800.

6.1.1 Architektúra siete GSM

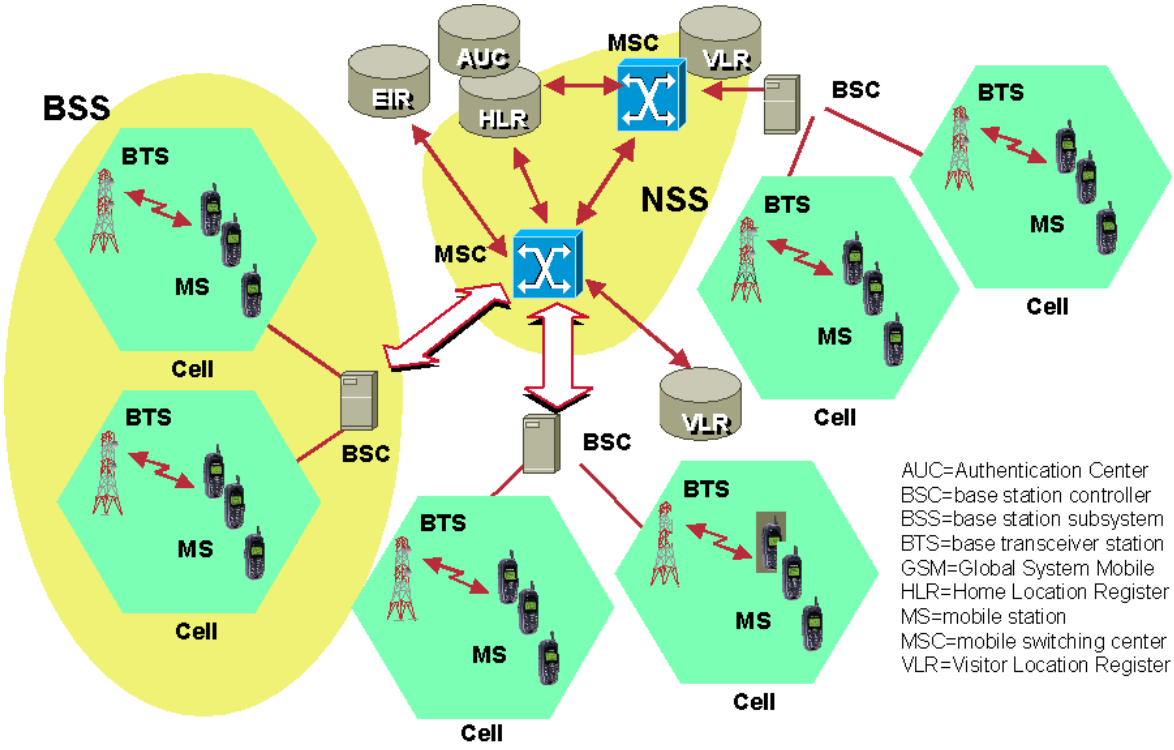
Sieť GSM využíva celulárny čiže bunkový princíp. Pokryté územie je rozdelené do jednotlivých buniek. V centre každej bunky sa nachádza základná stanica označovaná ako BTS (Base Transceiver Station) a práve ona zabezpečuje spojenie s mobilný telefónom a teda umožňuje s ním komunikovať. Mobilný telefón sa pritom musí nachádzať s príslušnej bunke. Niekoľko BTS je riadene základňovou riadiacou jednotkou BSC (Base Station Controller). BTS a BSC tvoria subsystém základňových staníc BSS (Base Station Subsystem).

Sústava všetkých základňových staníc mobilnej siete je cez svoje riadiace jednotky napojená na centrálnu ústredňu (MSC, Mobile Services Switching Centre), ktorú si môžeme predstaviť ako analógiu klasickej telefónnej ústredný z pevnej siete. Tiež slúži k smerovaniu jednotlivých hovorov k ich príjemcom alebo inej mobilnej siete. Tato štruktúra tvorí administratívni región. Spojenie MSC s inými sieťami (pevná linka) je cez bránu GMSC, kde G znamená Gateway.

MSC využíva pre smerovanie niekoľko identifikačných báz: HLR (Home Location Register) - informácie o účastníkoch, ktorí patria do administratívneho regiónu centrálnej ústrední MSC. Je dôležitá pri pohybe z účastníka z jednej stanice do druhej . VLR (Visited Location Register) - dočasné

informácie získané z HLR o účastníkoch, ktorí sa aktuálne nachádzajú v danom administratívnom regióne. AUC (Authenticatiion Center) - chránené dáta, EIR (Equipment Identity Register) - údaje o mobilných staniciach.

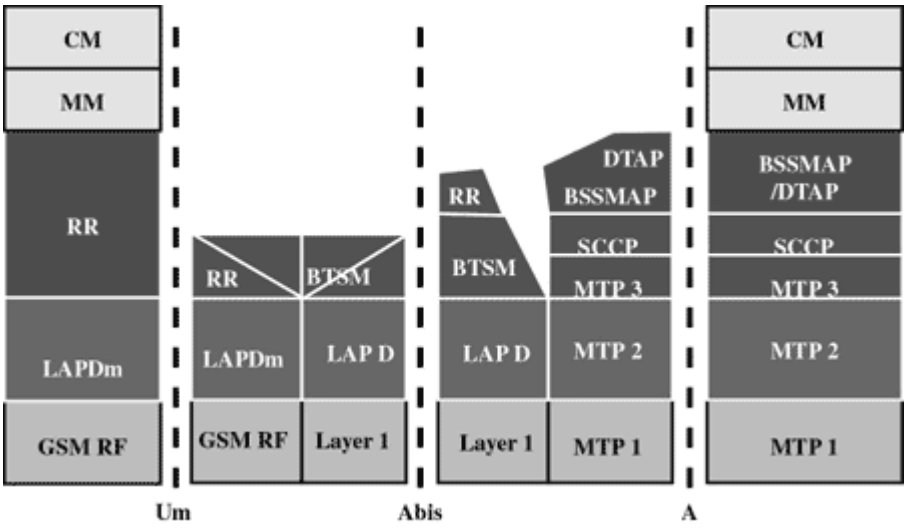
Keď dôjde k premiestneniu z pásma jednej bunky do druhej, základné stanice to hneď spoznajú a komunikáciu s daným zariadením si medzi sebou predajú. Používateľ daného mobilného telefónu by to vôbec nemal zaznamenať. Jednotlivé základňové stanice (BS, resp. BTS) samozrejmé musia byť medzi sebou prepojené a musia mať spoločné riadenie.



Obrázok 45. Štruktúra siete GSM

6.1.2 Protokolový zásobník GSM

Vrstvový model architektúry GSM integruje a spája peer-to-peer komunikáciu medzi dvoma odlišnými systémami. Protokoly nižších vrstiev poskytujú nesú protokoly vyšších vrstiev. Výmena informácii nastáva vždy iba medzi dvoma susednými vrstvami aby bolo zaistené správne formátovanie, prenos a prijatie informácie. Nasledujúci obrázok ukazuje model GSM protokolového zásobníka.



Obrázok 46. GSM protokolový zásobník

6.1.2.1 MS protokoly

GSM signalizačné protokoly sú rozdelené do troch hlavných vrstiev, závisiacich na rozhraní:

Layer 1: Fyzická vrstva ktorá moduluje a prenáša jednotlivé komunikačné kanály cez vzduchové rozhranie.

Layer 2: Data-link vrstva. Na **Um** rozhraní sa používa modifikovaná verzia Link Access protocol for the D channel (LAP-D) protokolu používaného v ISDN, nazvaná Link Access protocol on the Dm channel (LAP-Dm). Na **A** rozhraní je použíta Message Transfer Part (MTP) časť protokolu SS7.

Layer 3: Tretia vrstna GSM signalizačného protokolu je rozdelená do troch subvrstiev.

- (RR) Radio Resource management
- (MM) Mobility Management
- (CM) Connection Management

6.1.2.2 MS – BTS protokoly

RR vrstva sa stará o zriadenie linky, a to ako rádiovej tak aj pevnej, medzi MS a MSC. Hlavné funkčné súčasti pracujúce s RR vrstvou sú MS, BSS a MSC. RR vrstva sa stará o RR-relácie v dedikovanom móde, a taktiež o konfiguráciu rádiových kanálov, vrátane rozdelenia špecializovaných kanálov.

MM vrstva je postavená v hornej časti RR vrstvy a zvláda funkcie, ktoré vznikajú v súvislosti s mobilitou účastníka, rovnako ako o autentizačné a bezpečnostné aspekty. Lacation management sa zaoberá postupmi, ktoré umožnia, aby systém poznal aktuálnu polohu zapnutej MS tak, aby mohli byť uskutočňované prichádzajúce volania.

CM vrstva je zodpovedná za CC, doplnkové služby, manažment SMS a riadenie. Každá služba pod CM môže byť považovaná za samostatnú podvrstvu vrstvy CM. CC vrstva taktiež zahŕňa vytvorenie hovoru, výber typu služieb (vrátane prechodu medzi službami počas hovoru) a ukončenie hovoru.

6.1.2.3 BSC protokoly

Po odoslaní informácii z BTS na BSC je použitý rad rôznych rozhraní. Prenos informácii medzi BTS a BSC používa rozhranie Abis. Na tejto úrovni rádiové zdroje nižšej vrstvy L3 sú menené z RR na Base Transciever Station Management (BTSM). Manažment BTS využíva samostatný kanál medzi BTS a MSC. Tento kanál sa neuvádza medzi GSM protokolmi.

Protokoly RR vrstvy sú zodpovedné za alokáciu a realokáciu hovorových kanálov medzi MS a BTS. Tieto služby obsahujú správu a počiatočný prístup k systému, vyvolávanie MT hovorov, handover hovorov medzi bunkami, správu výkonu prenášaného signálu a ukončenie hovorov. Protokoly RR vrstvy poskytujú procedúry určené na používanie, alokáciu, realokáciu a uvolňovanie GSM kanálov. BSC sa stará o časť správy rádiových zdrojov určenú na koordináciu frekvencií, alokáciu frekvencií, a manažment vrstvy L2.

Na komunikáciu medzi BSC a MSC sa používajú protokoly rodiny SS7. MTP 1-3 sú použité v podpornej architektúre a na vyšších vrstvách je použitý BSS MAP protokol.

6.1.2.4 MSC protokoly

Do MSC prichádzajú informácie z BSC pomocou rozhrania **A** a MTP vrstiev 1-3. Ekvivalent rádiových zdrojov je nazvaný BSS MAP. BSS MAP / DTAP a MM a CM sú vrchné vrstvy protokolov tretej vrstvy L3. Toto ukončuje proces prenosu. Cez ovládaco-signalizačnú sieť (controll-signaling network) jednotlivé MSC navzájom komunikujú za účelom vyhľadania a prepojenia užívateľov pomocou mobilnej siete. Každá MSC obsahuje lokalizačné registre (VLR/HLR), slúžiaci na vyhľadanie a určenie parametrov pripojenia MS a roamujúcich užívateľov.

Každý užívateľ GSM siete má priradený záznam v HLR obsahujúci lokáciu a povolené služby. Separovaný register VLR slúži na vyhľadanie lokácie užívateľa. AK sa užívateľ vzdiali z dosahu danej MSC a jej VLR, pomocou protokolu SS7 si MSC vymenia informácie o zmene lokácie užívateľa.

Nasledujúca kapitola uvádza príklad GSM RR protokolu slúžiaceho na správu hovorov.

6.1.2.5 Protokol RR

Protokol RR najdeme v štandarde GSM 04.08 www.etsi.fr .

8	7	6	5	4	3	2	1	Octet
Protocol discriminator				Skip indicator				1
Message type								2
Information elements								3 - n
RR structure								

Obrázok 47. Štruktúra protokolu RR

Protocol discriminator: slúži na rozpoznanie RR protokolu, pri CC ma hodnotu 0110.

Skip indicator: slúži na určenie zlých paketov.

Message type: pole slúžiace na rozlíšenie vnorených protokolov:

```
00111- - -        Channel establishment messages:
      011         ADDITIONAL ASSIGNMENT
      111         IMMEDIATE ASSIGNMENT
      001         IMMEDIATE ASSIGNMENT EXTENDED
      010         IMMEDIATE ASSIGNMENT REJECT
00110- - -        Ciphering messages:
      101         CIPHERING MODE COMMAND
      010         CIPHERING MODE COMPLETE
00101- - -        Handover messages:
      110          ASSIGNMENT COMMAND
      001         ASSIGNMENT COMPLETE
      111         ASSIGNMENT FAILURE
      011         HANDOVER COMMAND
      100         HANDOVER COMPLETE
      000         HANDOVER FAILURE
      101         PHYSICAL INFORMATION
00001- - -        Channel release messages:
      101         CHANNEL RELEASE
      010         PARTIAL RELEASE
      111         PARTIAL RELEASE COMPLETE
      ...
```

Obrázok 48. Časť tabuľky významov RR message type

Vyššie uvedený protokol RR som si vybral do príkladu pre jeho relatívnu jednoduchosť. Úplný XML popis protokolu RR vrátane automaticky vygenerovaného zdrojového kódu sa nachádza v prílohe 1 a 2. Postup analýzy a generovania zdrojového kódu protokolu RR a vnorených protokolov / InformationElementov je rozpísaný v príslušných kapitolách.

7 Záver

Úlohou tejto diplomovej práce bolo preštudovať spôsoby automatickej analýzy komunikačných protokolov, navrhnúť štruktúru analyzátora protokolov riadeného pravidlami, vytvoriť testovaciu databázu pravidiel pre rodinu protokolov GSM, a vytvorený analyzátor otestovať. Zadanie bolo splnené v celom rozsahu.

Úvod práce uvádza prehľad existujúcich riešení. Opisuje existujúce riešenia ich silné a slabé stránky. Keďže navrhovaný analyzátor má zvládať aj analýzu GSM sietí práca sa zameriava sa aj na špecifické analyzátory GSM sietí. Ďalej sa práca venuje teoretickému úvodu do problematiky v ktorom podáva odpovede na otázky čo je to protokol a aké sú možnosti popisu protokolov.

Jadro práce tvorí samostatná návrhová práca, ktorá opisuje návrh štruktúry, špecifikáciu a implementáciu analyzátora sietí. Najdôležitejšou časťou systému je vlastný navrhnutý metamodel slúžiaci na tvorbu modelov popisujúcich komunikačné protokoly a kompilátor slúžiaci na prevod užívateľom definovaných modelov do dynamicky linkovaného modulu pravidlá.dll.

Posledná časť tejto práce podáva podrobnejší úvod do GSM technológie a použitých protokolov na ktorých je aj celý analyzátor otestovaný.

Testami sa preukázalo že navrhnutý analyzátor a metamodel dokáže popísať celú rodinu GSM protokolov pre ktoré je primárne určený. Navrhnutý analyzátor taktiež spĺňa rozšírené požiadavky na systém ktorí definujú analyzátor riadený pravidlami ako trojstupňovú aplikáciu ktorá dokáže vybrať ľubovoľný zdroj dát, definovať pravidlá určené na analýzu dát a nakoniec definovať požadovaný formát výstupu. Navrhnutý analyzátor toto dosahuje pomocou riešenia formou zásuvných modulov, ktoré dávajú možnosť ďalšieho rozširovania tohto systému.

Literatúra

[1] Rahnema, M.: Overview Of The GSM Systém and Protocol Architecture, IEEE Communications Magazine, April 1993

[2] 3GPP: Mobile radio interface layer 3 specification, 3GPP TS 04.08, 1998 [online] Verzia 7.21.0 [cit. 2008-12-30]. Dostupné na URL: < http://portal.etsi.org/Portal_Common/home.asp >

[3] Razavi, A: ProtoTalk: An Object-Oriented Framework for Implemeting Telecomunication Protocols, University of Waterloo 2007

[4] Hanáček P.: BMS - Bezdrátové a mobilní sítě, FIT VUT Brno 2008 [online] Aktualizované 2008-12-08 [cit. 2008-12-20]. Dostupné na URL: < https://www.fit.vutbr.cz/study/courses/BMS/public/bms.htm >

[5] Abdullah, I: Protocol specification and automatic implementation using XML and CBSE, CIIT, 2003

[6] Wireshark Developer's Guide [online] Verzia 28551 [cit. 2009-06-01]. Dostupné na URL: < http://www.wireshark.org/docs/wsdg_html_chunked/ >

[7] Radcom – Network Testing and Monitoring [online] Aktualizované 2008-12-20 [cit. 2008-12-20]. Dostupné na URL: < http://www.radcom.com/Products.aspx?boneId=374 >

[8] Tektronics – Protocols Analyzers [online] Aktualizované 2008-12-30 [cit. 2008-12-30]. Dostupné na URL: < http://www.tektronixcommunications.com/modules/communications >

[9] Špecifikácia IP protokolu RFC 791 [online] Aktualizované 1981-09-01 [cit. 2008-12-30]. Dostupné na URL: < http://www.faqs.org/rfcs/rfc791.html >

[10] ASN.1 & OID Project [online] Aktualizované 2008-12-20 [cit. 2008-12-20]. Dostupné na URL: < http://www.itu.int/ITU-T/asn1/ >

[11] 3GPP: Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface , 3GPP TS 04.11, 1996 [online] Verzia 5.1.0 [cit. 2008-12-30]. Dostupné na URL: < http://portal.etsi.org/Portal_Common/home.asp >

[12] Hillebrand F.: GSM and UMTS, The Creation of Global Mobile Communications, John Wiley & Sons, 2002

[13] Dynamically creating Applications using System.CodeDom [online] Aktualizované 2005-05-21 [cit. 2009-03-17]. Dostupné na URL: <http://www.c-sharpcorner.com/UploadFile/skarthikeyan/DynamicCreatingApplication06022005064351AM/DynamicCreatingApplication.aspx >

[14] System.CodeDom Namespace () [online] Aktualizované 2009-03-18 [cit. 2009-03-18]. Dostupné na URL: < http://msdn.microsoft.com/en-us/library/system.codedom.aspx >

[15] C# CodeDOM introduction [online]
Aktualizované 2002-06-27 [cit. 2009-03-18]. Dostupné na URL:
< http://www.codeproject.com/KB/recipes/codedomparser.aspx >

[16] I know the answer (it's 42) : C# 2.0: Loading plugins at run-time [online]
Aktualizované 2009-04-13 [cit. 2009-04-13]. Dostupné na URL:
< http://blogs.msdn.com/abhinaba/archive/2005/11/14/492458.aspx >

[17] C#: Runtime Assembly Loading, Reflection and Polymorphic Casting [online]
Aktualizované 2009-04-13 [cit. 2009-04-13]. Dostupné na URL:
<http://social.msdn.microsoft.com/Forums/en-US/netfxbcl/thread/81b05943-cb17-4388-945a-b5cec2b98f23 >

[18] Using Reflection to load unreferenced assemblies at runtime [online]
Aktualizované 2009-04-13 [cit. 2009-04-13]. Dostupné na URL:
< http://www.codeproject.com/KB/cs/csharpreflection.aspx >

Prílohy

1) XML popis InformationElementu RR
2) Vygenerovaný zdrojový kód InformationElementu RR
3) Diagram metamodelu

1) XML popis InformationElementu RR

```
<InformationElement Id="{733aaef8-4590-4966-8ff8-8d7b02df0052}" Name="RR">
  <Children>
    <Sequence>
      <SequenceChild SequenceNumber="0">
        <Item Id="{b0629596-046a-4980-bad4-e2c274b7141e}"
ObjectReference="{5c3c42b3-33b6-48d8-988b-26898cd7a2df}" Name="Protocol
Discriminator" />
      </SequenceChild>
      <SequenceChild SequenceNumber="1">
        <Item Id="{d2539858-93f3-4942-8615-eace2ae0aa69}"
ObjectReference="{273d3d10-718c-4e62-8ddd-2021471f68a3}" Name="Skip indicator" />
      </SequenceChild>
      <SequenceChild SequenceNumber="2">
        <Item Id="{47fec4ef-5ba2-46d9-b4a3-d3526072ad0a}"
ObjectReference="{80d53397-8975-4e45-8e15-156943d3707f}" Name="Message type" />
      </SequenceChild>
      <SequenceChild SequenceNumber="3">
        <Choice>
          <Condition>
            <ReferencedInstance ObjectReference="{47fec4ef-5ba2-46d9-b4a3-
d3526072ad0a}">
              <SubReference>.Data</SubReference>
            </ReferencedInstance>
          </Condition>
          <Case Value="06">
            <Children>
              <Sequence>
                <SequenceChild SequenceNumber="0">
                  <Item Id="{799e21ec-54e0-4ec4-98a8-87b2b8999980}"
ObjectReference="{904a5f2a-b321-42e1-b8e2-b2bf1d35f27b}" Name="Aditional
Assignment" />
                </SequenceChild>
              </Sequence>
            </Children>
          </Case>
          <Case>
            <Children>
              <Sequence>
                <SequenceChild SequenceNumber="0">
                  <Item Id="{8679950e-3b49-445a-a4d1-221128288881}"
ObjectReference="{5c3c42b3-33b6-48d8-988b-26898cd7a2df}" Name="Toto je nejaky
default" />
                </SequenceChild>
              </Sequence>
            </Children>
          </Case>
        </Choice>
      </SequenceChild>
    </Sequence>
  </Children>
</InformationElement>
```

2) Vygeneovaný zdrojový kód InformationElementu RR

```
namespace Protocols.GsmTest
{
  using ObjectModel.InformationElements;
  using System;
  using System.Collections.Generic;
  using System.Text;

  public class RR : InformationElementBase
  {
    #region Local Variables
    private Protocols.GsmTest.RR_Message_Type m_Message_type;
    #endregion
    public RR()
    {
    }
    public RR(InformationElementBase parent_, int dataBase_, int dataOffset_)
    {
      this.parent = parent_;
      this.data = parent_.DataColl;
      this.dataBase = dataBase_;
      this.dataOffset = dataOffset_;
      this.ParseData();
    }
    private void ParseData()
    {
      int tmpBase = this.dataBase;
      int tmpOffset = this.dataOffset;
      int tmpLength = 0;
      // sequence .... .... .... .... ....
      Protocols.GsmTest.Protocol_Discriminator tmp_Protocol_Discriminator;
      tmp_Protocol_Discriminator = new Protocols.GsmTest.Protocol_Discriminator(this,
      tmpBase, tmpOffset);
      this.addLength(ref tmpBase, ref tmpOffset, ref tmpLength, tmp_Protocol_Discriminator);
      this.items.Add("Protocol Discriminator", tmp_Protocol_Discriminator);
      // ...
      Protocols.GsmTest.SkipIndicator tmp_Skip_indicator;
      tmp_Skip_indicator = new Protocols.GsmTest.SkipIndicator(this, tmpBase, tmpOffset);
      this.addLength(ref tmpBase, ref tmpOffset, ref tmpLength, tmp_Skip_indicator);
      this.items.Add("Skip indicator", tmp_Skip_indicator);
      // ...
      Protocols.GsmTest.RR_Message_Type tmp_Message_type;
      tmp_Message_type = new Protocols.GsmTest.RR_Message_Type(this, tmpBase, tmpOffset);
      this.addLength(ref tmpBase, ref tmpOffset, ref tmpLength, tmp_Message_type);
      this.items.Add("Message type", tmp_Message_type);
      this.m_Message_type = tmp_Message_type;
      // ...
      // choice .... .... .... ....
      if (compareData(this.m_Message_type.Data, new byte[] { 6 }))
      {
        // sequence .... .... .... .... ....
        Protocols.GsmTest.Aditional_Assignment tmp_Aditional_Assignment;
        tmp_Aditional_Assignment = new Protocols.GsmTest.Aditional_Assignment(this, tmpBase,
        tmpOffset);
        this.addLength(ref tmpBase, ref tmpOffset, ref tmpLength, tmp_Aditional_Assignment);
        this.items.Add("Aditional Assignment", tmp_Aditional_Assignment);
        // ...
      }
      else
      {
        // sequence .... .... .... .... ....
        Protocols.GsmTest.Protocol_Discriminator tmp_Toto_je_nejaky_default;
        tmp_Toto_je_nejaky_default = new Protocols.GsmTest.Protocol_Discriminator(this,
        tmpBase, tmpOffset);
        this.addLength(ref tmpBase, ref tmpOffset, ref tmpLength,
        tmp_Toto_je_nejaky_default);
        this.items.Add("Toto je nejaky default", tmp_Toto_je_nejaky_default);
        // ...
      }
    }
  }
}
```

3) Diagram metamodelu