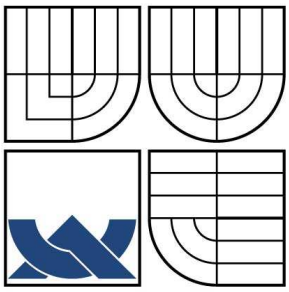
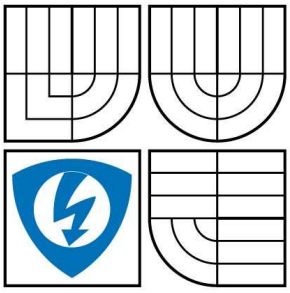


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE OBSLUŽNÝCH FUNKCÍ PRO PRÁCI S DATABÁZÍ MIB V SIMULACNÍM PROSTREDÍ OPNET MODELER

IMPLEMENTATION OF SERVICE FUNCTIONS FOR WORK WITH MIB DATABASE IN SIMULATION ENVIRONMENT OPNET MODELER

BAKALÁRSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN RUML
AUTHOR

VEDOUCÍ PRÁCE — Ing. KAROL MOLNÁR, Ph. D.
SUPERVISOR

BRNO 2008

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ
Fakulta elektrotechniky
a komunikačních technologií
Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Ruml Martin
Ročník: 3

ID: 77784
Akademický rok: 2007/08

NÁZEV TÉMATU:

Implementace obslužných funkcí pro práci s databází MIB v simulačním prostředí OPNET Modeler

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se se standardy zabývající se strukturou systémové databáze MIB a s obslužnými funkcemi této databáze. Identifikujte základní požadavky na agenta, který se bude starat o modelovou databázi MIB. Při specifikaci požadovaných funkcí zohledněte všechny možné způsoby získání a zápisu údajů do MIB. Výsledky analýzy zdokumentujte.
Na základě definovaných požadavků na obslužný systém databáze MIB navrhněte modul agenta SNMP, který bude mít na starosti správu databáze MIB. Implementujte také funkce, které zajišťují práci s tabulkami v MIB. Po návrhu proveďte implementaci jednotlivých obslužných funkcí. Pro implementaci zvolte jazyk C.

DOPORUČENÁ LITERATURA:

[1] CASE, J., McCLOGHRIE, K., ROSE, M., WALDBUSSER, S. Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2) RFC 1442, Internet Engineering Task Force, 1993
[2] CASE, J., McCLOGHRIE, K., ROSE, M., WALDBUSSER, S. Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2) RFC 1450, , Internet Engineering Task Force,1993
[3] MILLER, A.M., Managing Internetworks with SNMP, Wiley, 1999

Termín zadání: 11.2.2008 ***Termín odevzdání:*** 4.6.2008

Vedoucí projektu: Ing. Karol Molnár, Ph.D.

prof. Ing. Kamil Vrba, CSc.
předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Martin Ruml

Bytem: Sušilova 460/3, 67801, Blansko

Narozen/a (datum a místo): 1.9.1985, Brno

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

prof. Ing. Kamil Vrba, CSc.

(dále jen „nabyvatel“)

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: Implementace obslužných funkcí pro práci s databází MIB v simulacním prostredí OPNET Modeler

Vedoucí/ školitel VŠKP: Ing. Karol Molnár, Ph.D.

Ústav: Ústav telekomunikací

Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů 1
- □ elektronické formě – počet exemplářů 1

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….. ……

Nabyvatel Autor

Anotace

Předmětem této bakalářské práce je implementace obslužných funkcí pro práci s databází MIB. Databáze MIB je systémová databáze, která slouží k uchovávání řídících informací v síťových uzlech. První kapitola slouží jako lehký úvod do řízení počítačových sítí. Druhá kapitola popisuje základní rysy protokolu SNMP. Protokol SNMP je standardizovaný aplikační protokol, který slouží k přenosu řídících informací z a do systémové databáze MIB. Protokol je založen na konceptu manager – agent, kdy aktivní je až na výjimky vždy manager, který se dotazuje agenta na hodnoty konkrétních položek. Strukturou a standardy pro systémové databáze MIB se zabývá další kapitola. Popisuje základní větvení mezi správními organizacemi.

Hlavní část práce je věnována návrhu a implementaci obslužných funkcí databáze MIB. V této kapitole jsou identifikovány hlavní požadavky na databázi MIB z hlediska spolupráce s protokolem SNMP. Tyto procesy jsou dále zapracovány do experimentální implementace databáze MIB. Implementace je napsána v jazyce C++. Z návrhu vznikl hlavičkový soubor tříd a podpůrných funkcí, jež implementují základní mechanizmy databáze MIB a jejích metod pro obsluhu této databáze. Hlavičkový soubor obsahuje též několik procesů pro ulehčení spolupráce databáze MIB s aplikačním protokolem SNMP. Závěrečná kapitola ukazuje používání navržených tříd. Vytvořený program je vybaven grafickým rozhranním, které využívá funkcí určených pro místní přístup. Mezi další ukázané funkce program patří velmi jednoduchý modul pro komunikaci s databází prostřednictvím zpráv SNMP.

Klíčová slova

MIB, Management Information Base, SNMP, Simple Network Management Protocol, obslužné funkce, databáze, implementace

Abstract

Implementation of service for work with MIB database is subjeckt the bachelor's thesis. MIB database is system database which is instrumental for storage of control informations in network elements. A first chapter is instrumental like a introduction to the control computer networks. A second chapter describes the basic features of protocol SNMP. The protocol SNMP is a standardized application protocol which uses for a transmission control informacions from and to the system database MIB. The protocol is based on a draft manager-agent when the manager is mostly active who enquire of the agent for the values of concrete items. A next chapter deals with a structure and standards for the MIB database. It describes a basic enbranchment among the administrative organizations.

A main part of working is devoted to a proposal and a implementation of servece functions MIB database. The main requirements for the MIB database in light of a cooperation with the protocol SNMP are identified in this chapter. These processes are trained-in to the experiment implementation of MIB database. The implementation is written in C++. A header file of the classes and supportive functions who implement basic mechanisms of MIB database and its methods for a tender the database created from the proposal. The header file contains the several processes for an easement of cooperation MIB database with the application protocol SNMP, too. A final chapter shows the using of designed classes. The created programe is equipped with a graphics interface who uses the functions for a local access. Very easy modul for the comunication with the database through the messages of SNMP is among next shown functions in the programe.

Keywords

MIB, Management Information Base, SNMP, Simple Network Management Protocol, service functions, database, implementation

RUML, M. *Implementace obslužných funkcí pro práci s databází MIB v simulačním prostředí OPNET Modeler* . Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 56 s. Vedoucí bakalářské práce Ing. Karol Molnár, Ph.D.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Implementace obslužných funkcí pro práci s databází MIB v simulačním prostředí OPNET Modeler jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Karolovi Molnárovi, Ph.D., za cenné rady, připomínky a průběžné konzultace při zpracování bakalářské práce.

V Brně dne …… ……..

podpis autora

Seznam použitých zkratek

API	- Application Programming Interface
ASN.1	- Abstract Syntax Notation One
C++	- Objektově orientovaný programovací jazyk
CMIP	- Common Management Information Protocol
IETF	- Internet Engineering Task Force
IP	- Internet Protocol
MIB	- Management Information Base
OID	- Object Identifier
RFC	- Request For Comments
SGMP	- Simple Gateway Monitoring Protocol
SNMP	- Simple Network Management Protocol
UDP	- User Datagram Protocol
VCL	- Visual Component Library

Obsah

Seznam obrázků

1. Úvod

Tato práce se věnuje nejprve lehkému úvodu do problematiky komunikačních sítí z hlediska nutnosti řízení. Následuje seznámení se základními rysy protokolu SNMP, jež standardizuje přenos řídících informací mezi řízeným zařízením (většinou síťovým uzlem) a manažerem. Dále tato práce analyzuje strukturu systémové databáze MIB a obslužných funkcím, jež jsou zapotřebí pro práci s touto databází. Provádí uvedení a rozbor základní problematiky řízení komunikačních sítí a identifikuje základní požadavky na agenta, jehož funkcí bude starat se o modelovou databázi MIB. Na základě rozboru struktury systémové databáze MIB je prováděn návrh modulu agenta, jež by měl zprostředkovat získávání a zápis údajů do MIB. Tento agent bude sloužit jako podpůrný modul vyšší vrstvě, jež představuje agent SNMP.

Praktická část této práce je zaměřena na implementaci struktury databáze MIB a navazujících obslužných funkcí, jež využívá agenta SNMP k zápisu a čtení řídících dat do databáze MIB. Implementace je prováděna v programovacím jazyce C++.

2. Řízení komunikačních sítí

Řízení komunikačních sítí je podle [4] v nejobecnějším pojetí zajištění komplexních služeb pro koncové spotřebitele. Tuto problematiku můžeme rozdělit do dvou základních částí. První z nich je řízení vlastní komunikace v síti, což je problematika struktury řízení komunikace v sítích, o což se stará komunikační protokol jednotlivých síťovích architektur. Druhou problematikou je aplikační řízení sítí obvykle nazývaný Network Management, jež tvoří prostředky a služby pro dohled, koordinaci a spravování komunikačních zdrojů. Právě potřeba managementu sítí vyvolala vznik paralelní architektury v síťových prvcích, jež by mohla přijímat a odesílat pakety zajišťující řízení sítě.

Sjednocením způsobu řízení a jednotný řídící protokol, nezávislý na použitém síťovém hardwaru, umožňuje jednotnou reakci řízených síťových komponent na stejný řídící příkaz. Dále masivním rozvojem síťových technologií a vzrůstajícím počtem síťových komponent v těchto sítích, posiluje značná potřeba dálkové správy a neustálého dohledu nad komponentami sítě. Proto na základě iniciativy výrobců síťových komponent byly definovány standardy, jež umožňují vzdálenou správu síťových prvků. Jedním z těchto protokolů je standard z názvem Simple Network Management Protocol zkráceně SNMP, který je definován dokumentem RFC 1157. Tento standard je určen pro řízení dnes tak hojně využívaných TCP/IP sítí.

Provádění monitorování a řízení homogenní sítě je relativně jednoduché, avšak u heterogenních sítí je nutno sjednotit jak protokoly, tak i řídící příkazy. Proto se provádí návrh systémů řízení sítě na vysoké stupni abstrakce. Díky koncové orientovanosti transportních protokolů je možno přenášet řídící informace izolovaně od ostatních protokolů, což je pro implementaci řídících systémů žádoucí.

Systém řízení sítě jsou komunikační systémy, které dokáží řídit všechny parametry služeb, které poskytuje komunikační síť. Pro tyto účely musí zajistit sledování síťovích zdrojů, archivovat získaná data, přesně definovat množinu monitorovaných informací. Na základě takto shromážděných informací systém provádí diagnostiku a následně rekonfiguraci sítě s ohledem na zachování kvality koncové služby. V neposlední řadě musí systém nabízet prostředky pro analýzu a tvorbu výkonnostních a provozních charakteristik pro správce sítě.

3. Protokol SNMP

S rozvojem síťových technologií a vzrůstajícím počtem síťových komponent v těchto sítích vzniká značná potřeba dálkové správy a neustálého dohledu nad komponentami sítě. Proto byl v rove 1988 organizací Internet Engineering Task Force (IETF) uveden standard s názvem Simple Network Management Protocol (SNMP), do češtiny přeložitelné jako jednoduchý protokol pro správu sítě. Tento standard vznikl na základě iniciativy uživatelů a výrobců síťových prvků. Jedná se o aplikační protokol, jež je založen na modelu klient-server a nabízí službu správy sítí TCP/IP.

3.1. Vznik SNMP

Podle [2] se vznik protokolu SNMP datuje do období 80 let. Na počátku stál protokol Simple Gateway Monitoring Protocol (SGMP), který sloužil k výměně informací mezi směrovači a bránami tehdy ještě akademické sítě. Tento protokol byl navržen koncem roku 1987. Jednou z variant protokolu SGMP byl protokol SNMP, jako protokol pro správu směrovačů v internetu. Druhým standardem, jež vznikl z protokolu SGMP je protokol Common Management Information Protocol (CMIP), jež pochází od organizace ISO. Zpočátku byla snaha o společný vývoj obou variant, alespoň na úrovni společné struktury spravovaných informací (SMI) a informační databáze (MIB), avšak i toto řešení se ukázalo jako nepraktické a to převážně z důvodu objektové orientace CMIP.

3.2. Princip funkce

Podle [2] klientská aplikace nazývaná Manager vytvoří virtuální spojení se serverem, jež se nazývá SNMP agent a který běží na sledovaném zařízení. Agent monitoruje stav zařízení na kterém běží a poskytuje o něm informace manageru. Tyto informace mohou být například počet zpracovaných paketů za jednotku času nebo počet chybných paketů za jednotku času. Administrátor přes managere takto spravuje jednotlivá zařízení na dálku a může na základě získaných informací snadněji provádět správu sítě, identifikovat a odstraňovat vzniklé závady. Informace jež poskytuje agent vyplívají ze struktury daného zařízení a jsou uspořádány v databázi MIB.

Protokol SNMP využívá dvou základních komponent:

- Agent – tuto část obsahuje spravované zařízení
- Manager – jedná se o konzoly pro správu přes níž se vyčítají nebo zapisují data spravovaná agentem.

Agent je proces běžící ve spravovaném zařízení. Je mu vyčleněna část procesorového času a operační paměti, v této paměti udržuje právě aktuální parametry sloužící k řízení chování daného zařízení. Obhospodařované parametry jsou uchovávány v přesně definovaných strukturách specifikující konkrétní datové typy uchovávaných dat. Nad těmito strukturami jsou podle instrukcí prováděny operace čtení a zápisu. Tato databáze má název Management Information Base (MIB).

Manager je proces běžící na pracovní stanici nebo serveru, přes nějž je správa prováděna. K provádění správy je manager vybaven znalostí struktury databáze MIB, jež se nachází ve spravovaném Agentovy. Je schopen komunikovat s agentem a je vybaven nástroji umožňujícími čtení a zápis z databáze MIB.

Agent a Manager spolu komunikují prostřednictvím protokolu SNMP. Aby mohlo spolupracovat i zařízení od různých výrobců, musí protokol SNMP přesně definovat způsob výměny informací a to jak komunikační protokol, tak popis přenášených dat. Protokol SNMP patří do rodiny protokolů IP a ke komunikaci používá přenosový protokol UDP (User Datagram Protocol), jež poskytuje nezaručený přenos dat pomocí přenosu datagramů mezi počítači v síti.

Protokol SNMP prokázal vekou životaschopnost. Díky univerzálnosti a jednoduchosti tohoto protokolu dochází k jeho implementaci skoro do každého sofistikovanějšího síťového zařízení, čímž získává tento standard klíčové postavení mezi ostatními standardy pro řízení sítí.

Vývoj protokolu SNMP pokračuje dále a přináší nové možnosti. Jedním z významných vylepšení je standard RMON (Remote Monitoring), neboli vzdálené monitoroví, jež se zaměřuje na změnu komunikace mezi spravovaným zařízením a administrátorským terminálem. Dochází z přemístění větší části činnosti na agenta a tím nižšímu zatížení sítě.

3.2.1. Správy SNMP v1

Pro výměnu řídících informací protokol SNMPv1 definuje následující typy správ:

- GetRequest
- GetNextRequest
- GetRespons
- SetRequest

- Trap

GetRequest

Je to správa, kterou odesílá managerem a je určená agentovy. Jejím účelem je žádat o informaci, přesněji hodnotu položky databáze MIB. Jedná se o operaci čtení určitého objektu, který je přesně určen pomocí OID, jež je ve správě obsaženo.

GetNextRequest

Jde o žádost o další informace. Správu též generuje manager a příjemcem je agent. Jedná se taktéž o operaci čtení, ale čtená informace je v následujícím uzlu. Díky tomu nemusíme posílat v specifikovat uzel stromu. Tento dotaz se používá se spojení s dotazem GetRequest.

GetRespons

Je to odpověď od agenta a je adresovaná managerovy jako odpověď na jeho předešlou žádost GetRequest. Přenáší hodnotu objektu, kterou si manager vyžádal.

SetRequest

Správa je odeslána managerem a je určena agentovi. Slouží k nastavení hodnoty objektu databáze MIB. Jde tedy o operaci zápisu. Pokud tuto zprávu agent nezvládá, neprovádí se vlastně na daném zařízení management, ale jen sledování.

Trap

Jedná se o speciální typ správy, který je odesílána agentem a to bez předchozího přijetí jakéhokoli požadavku. Správa je odeslána jako reakce na určenou událost, ke které na spravovaném zařízení došlo.

3.2.2. Správy SNMP v2

Tato druhá verze standardu SNMP obohacuje výčet typů správ ze standardu SNMPv1 následujícími novými typy správ:

- GetBulkRequest - Tato správa slouží k získání více hodnot i přes více uzlů. Je určena především pro čtení rozsáhlých tabulek.
- InformRequest - Jedná se o správu vhodnou pro rozsáhlejší sítě. Jejím účelem je informovat o určitých událostech v spravovaných oblastech.

4. Databáze MIB

Vzhledem k tomu, že protokol SNMP slouží pouze pro přenos řídících informací, bylo nutné tyto řídící informace někde uchovat v jednotlivých spravovaných zařízeních. Proto vzniklo strukturované úložiště informací s názvem Management Information Base (MIB). Tato databáze slouží pro uložení informací o parametrech uzlu samotného a o provozu přes tento uzel procházejícího.

Podle [2] MIB je standard, který popisuje sadu objektů, které jsou předmětem správy. Struktura databáze MIB se odvíjí od funkcí spravovaného zařízení, proto jednotlivá zařízení implementují jednu nebo více větví MIB v závislosti na svých funkcích. Databáze MIB jako každá databáze specifikuje struktura a formát dat, které může obsahovat. Popisy databáze MIB jsou vytvářeny podle pravidel Structure of Management Information (SMI), jež jsou dány dokumenty RFC1155, RFC1212 a RFC1215.

Podle [1] jsou statické i dynamické parametry síťových prvků a síťového provozu jím protékajícího modelovány pomocí řízených objektů. A právě popis těchto objektů a jejich chování je standardizováno strukturou řídících informací SMI.

4.1. Popis spravovaných objektů

Podle [1] se popis spravovaných objektů skládá ze tři části:

- Název
- Typ a syntaxe
- Kódování

4.1.1. Název objektu

Slouží k jednoznačné identifikaci daného objektu. Nazývá se identifikátor objektu, což je z anglického Object Identifier (OID). Nabývá dvou forem a to číselné a textové. Číselný se využívá při automatizovaném zpracování, naproti tomu textový je vhodnější pro orientaci člověka.

4.1.2. Typ a syntaxe objektu

Definici typu a syntaxe se provádí v jazyce Abstract Syntax Notation One (ASN.1). Tento jazyk dovoluje definovat formu reprezentace dat, která jsou následně přenášena mezi agentem a managerem.

Podle [3] jazyk ASN.1 definuje tři typy objektů:

- Type – slouží pro definici nových datových struktur
- Value – neboli hodnoty a můžeme též říkat proměnný.
- Macro – definuje rozšíření jazyka

Uzly stromové struktury můžeme adresovat za požití identifikačních číslic oddělených tečkou a nebo pomocí textového názvu. Stromová struktura se skládá z uzlů jejichž definice může vypadat například takto:

```
mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }
```

Zde vidíme, že důležitým klíčovým slovem je OBJECT IDENTIFIER, před kterým se nalézá textový název uzlu. Za tímto klíčovým slovem je textový název nadřazeného uzlu a číslo identifikující definovaný uzel. Dále může definice obsahovat další klíčová slova, jež slouží ke specifikaci datového typu SNMP jak ukazuje následující definice:

```
sysName OBJECT-TYPE
      SYNTAX  DisplayString (SIZE (0..255))
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
              "An administratively-assigned name for this
              managed node.  By convention, this is the
              node's fully-qualified domain name."
      ::= { system 5 }
```

Uvedený příklad ukazuje definici položky stromové struktury, která nese skalární hodnotu. Pro definici tabulky se využívá klíčových slov SEQUENCE a SEQUENCE OF. Nejprve je definován kořenový uzel tabulky, kde klíčové slovo SEQUENCE OF znamená, že uzel bude obsahovat neznámý počet záznamů, jež budou typu uvedeném za klíčovým slovem.

```
ipRouteTable OBJECT-TYPE
      SYNTAX  SEQUENCE OF IpRouteEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
              "This entity's IP Routing table."
      ::= { ip 21 }
```

Klíčového slova SEQUENCE je použito pro definici schématu tabulky, kdy tento záznam udává, jaké sloupce bude tabulka obsahovat. Jsou zde definovány textové názvy sloupců a datové typy jednotlivých sloupců.

```
IpRouteEntry ::=
      SEQUENCE {
          ipRouteDest
              IpAddress,
          ipRouteIfIndex
              INTEGER,
          ipRouteMetric1
              INTEGER,
          ipRouteMetric2
              INTEGER,
          ipRouteMetric3
              INTEGER,
          ipRouteMetric4
              INTEGER,
          ipRouteNextHop
              IpAddress,
          ipRouteType
              INTEGER,
          ipRouteProto
              INTEGER,
          ipRouteAge
              INTEGER,
          ipRouteMask
              IpAddress,
          ipRouteMetric5
              INTEGER,
          ipRouteInfo
              OBJECT IDENTIFIER
      }
```

Pro určení primárního klíče, jež se používá k indexování tabulky slouží klíčové slovo INDEX, za nímž se nachází výčet textových názvů sloupců, přes které je index vytvořen a tudíž takto vzniká složený primární klíč.

```
ipRouteEntry OBJECT-TYPE
      SYNTAX  IpRouteEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
              "A route to a particular destination."
      INDEX   { ipRouteDest }
```

::= { ipRouteTable 1 }

Datové typy jednotlivých objektů, jež popisuje standardu ASN.1 se dělí na datové typy univerzální, aplikační, kontextově závislé a privátní.

Univerzální datové typy:

- Integer - je to celé číslo se znaménkem
- Octet String - řetězec znaků
- Object Identifier – objektový identifikátor
- Null – datový typ, který nemá hodnotu.

Aplikační datové typy:

- IpAddress
- Counter
- Gauge
- TimeTicks

4.1.3. Kódování objektu

Určuje fyzickou formu instance daného objektu. Pro přenos dat přes síť bylo zvoleno kódování Basic Encoding Rules (BER). Toto kódování udává přesná fyzická vyjádření dat, která jsou následně přepravována sítí.

4.2. Struktura MIB

Databáze MIB má tvar hierarchické stromové struktury a její tvar odpovídá danému zařízení. Databáze je objektově orientována a je složena z objektů. Stromová struktura databáze se promítá i do názvů jednotlivých objektů. Název objektu, neboli OID je tvořen celými čísly oddělenými tečkou. Tato celá čísla jsou uzly ve stromové struktuře a posloupnost těchto čísel definuje cestu skrz strom, do míst, kde se objekt nachází. Pro lepší orientaci lidí ve stromové struktuře databáze má OID nejen číselnou podobu, ale i textovou, kde jsou řetězce znaků odděleny tečnou a tedy funguje stejně jako číselné OID.

Objekty, ze kterých je stromová struktura složena jsou podle nejhrubšího dělení dva typy. První z nich je uzel stromu. Tento typ objektu slouží jako základní stavební prvek pro budování stromové struktury databáze. Druhým typem objektu je koncový uzel, neboli objekt, který obsahuje data.

Kořenový uzel stromu databáze MIB je bez jména. Pod tímto kořenovým uzlem se nachází důležité uzly, které představují tři oblasti, jež každá z nich má jinou spravující organizaci:

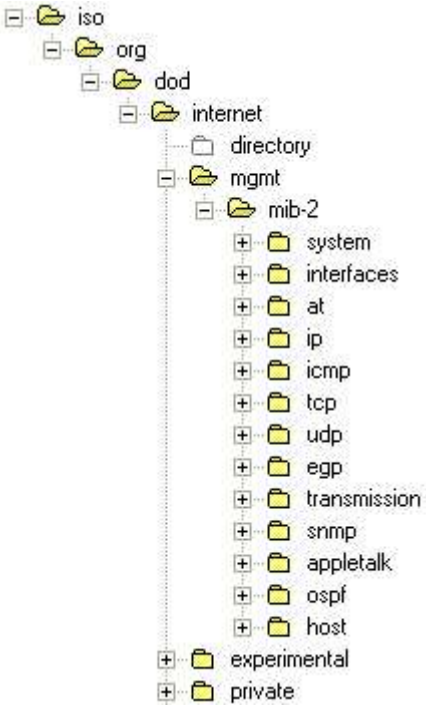
- ccitt(0) – organizace ISO
- iso(1) – organizace ITU-T
- joint-iso-ccitt(2) – společně spravováno organizací ISO a organizací ITU-T

Pro správu sítí na bázi protokolu IP je určena část podstromu databáze MIB s názvem iso(1).org(3).dod(6).internet(1). V této větvi se nachází objekty, jež odpovídají vlastnostem protokolů TCP/IP a mezi hlavní zanořené objekty patří větve:

- mgmt(2)
- private(4)

Ve větvi mgnt se nachází pouze jedna podvětev a tou je větev mib(1). V této větvi se nachází obecné vlastnosti, jež jsou dostatečné pro popis zařízení od různých výrobců. Tato obecnost zaručuje, že zařízení od různých výrobců tuto strukturu obsahující budou moci být spravovány jen jedním nástrojem.

Větev private(4) je určena pro jednotlivé výrobce, kteří mají zájem definovat vlastní strukturu databáze MIB. Takto je každému výrobci přidělena větev, v které má volnost tvorby vlastní struktury jak co do šířky tak i hloubky. Příkladem takové větve může být například cisco(9). OID pro tuto větev potom vypadá následovně 1.3.6.1.4.1.9



Obr 1.: Ukázka části stromu systémové databáze MIB

5. Návrh a implementace obslužných funkcí databáze MIB

Při návrhu databáze MIB, bylo zapotřebí vytvořit stromovou strukturu, kterou tato databáze má. Pro tento účel byl zvolen princip, kdy každý uzel stromu obsahuje ukazatel na uzel stromu, který leží pod ním a na uzel stromu, který leží vedle něj na stejné úrovni ve stromu. Taktéž pro zpětný chod o úroveň výše byl do návrhu přidán ukazatel na nadřazený uzel stromu. Tyto tři ukazatele dovolují plný průchod stromem. Díky tomu, že součástí nebyl ukazatel na předešlou položku na stejné úrovni, je tato zpětná část chodu obcházena vystoupáním na nadřazený uzel a dopředeným chodem po jemu podřízené nižší úrovni.

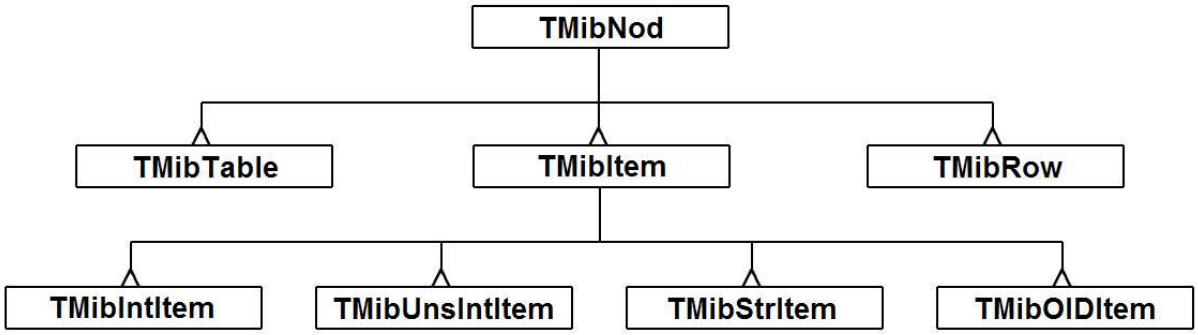
Vzhledem k nutnosti jednoznačné adresovatelnosti jednotlivých uzlů stromu bylo zapotřebí, aby uzly stromu uchovávaly v sobě nějakou formu identifikačních dat, jež by vyhověly potřebám adresace OID. Díky stromovému uspořádání databáze a úrovňovému členění OID, je možno v každém uzlu uchovávat pouze část, identifikující uzel jen na jeho konečné úrovni. Zbývající část OID je součástí nadřazených uzlů a stromového uspořádání. Pro tyto všechny výše jmenované účely byla navržena základní stavební jednotka stromu, kterou je třída TMibNod.

Toto všechno však neřeší samotné uložení informace, k čemuž hlavně má databáze MIB sloužít, proto na koncích stromu je nutné uložit hodnotu. V databázi MIB byly identifikovány tři základní datové typy pro uložení hodnoty. Těmi jsou Integer, neboli celé číslo se znaménkem, dále Unsigned Integer, neboli celé čísle beze znaménka. Oboje v 32 bitové reprezentaci. A jako poslední řetězec znaků.

Vzhledem k stromové struktuře uzlů, na níž jsou koncové položky s hodnotou napojovány a s přihlédnutím k potřebě jednotného přístupu k hodnotám položek, bylo navrženo rozhraní pro práci s položkami ve formě třídy TMibItem, jako potomek třídy TMibNod. Třída jako taková reprezentuje položku s hodnotou. Pro tento účel disponuje ukazatelem na proměnou daného typu a vlastní proměnnou, která specifikuje na jaký typ proměnné ukazatel směřuje. Tento přístup je však pro přístup k hodnotám velmi obecný, proto byly od třídy TMibItem odvozeny třídy TMibIntItem, TMibUnsItem a TMibStrItem, které již obsahují koncově orientované metody pro práci s konkrétním datovým typem jimi reprezentovaným.

Vzhledem k možnosti konverze obou číselných datových typů na řetězec je součástí návrhu třídy TMibItem specifikace rozhraní metod pro jednotné čtení i zápis hodnot do všech tří datových typů. K tomuto účelu je využito virtuálních metod, jež specifikují volání metod,

které jsou implementovány v třídách potomků pro konkrétní datové typy. Kompletní odvozování tříd, z kterých se strom skládá je znázorněno na obr. 1.



Obr 2.: Model dědičnosti tříd tvořících strom databáze MIB

Pro práci s celou stromovou strukturou jako celkem byla navržena třída TMibDatabase. Bylo potřeba koncentrovat schopnost čtení a zápisu skrz celý strom, proto tato třída obstarává implementaci funkcí sloužících jak k průchodu celým stromem, tak i tvorbu kořenového uzlu stromu. Do třídy byly i koncentrovány metody pro přidávání uzlů a položek do stromu jím spravovaným.

5.1. Implementace

5.1.1. Třída TMibDatabase

Představuje celou databázi MIB a zaštituje metody pro přístup k uzlům a položkám. K tomuto účelu je vybavena ukazatelem na kořenový uzel celého stromu. Tento kořenový uzel je vytvořen při zavolání konstruktoru třídy TMibDatabase. Je schopen zprostředkovat operace smazání celého stromu, přidání uzlu, přidání položky s hodnotou, čtení a zápisu hodnot položek podle zadaného objektového identifikátoru. K těmto účelům je vybaven následujícími metodami:

GetRootNod

Slouží k získání ukazatele na kořenový uzel stromu. Vrací proměnnou typu ukazatel na objekt TMibNod.

```
TMibNod* GetRootNod();
```

Clear

Slouží ke smazání všech položek a uzlů stromu.

```
void Clear();
```

AddSubNod

Slouží k přidání uzlu do stromové struktury. Přidávaný uzel je umístěn hned pod kořenový uzel stromu. Metoda vyžaduje dva vstupní parametry. První je identifikátor uzlu typu integer a druhý parametr je název uzlu typu ukazatel na nulou ukončený řetězec. Návratovou hodnotou je ukazatel na vytvořený uzel typu TMibNod.

```
TMibNod* AddSubNod(int ASubOID, char* AName);
```

GetOID

Slouží k získání ukazatele na položku ze stromu, která je specifikována objektovým identifikátorem. Jako vstupní parametr se zadává ukazatel na nulou ukončený řetězec, který musí mít tvar objektového identifikátoru a přesně určuje žádanou položku. Návratová hodnota je typu ukazatel na objekt typu TMibNod. Pokud je návratová hodnota 0, tak funkci se nepodařilo nalézt zadaný uzel.

```
TMibNod* GetOID(char* AOID);
```

AddNod

Slouží k přidání uzlu do stromové struktury. Pozice přidaného uzlu je zadána objektovým identifikátorem. Chybějící uzly stromu vedoucí k přidávanému uzlu jsou také vytvořeny. Jako vstupní parametry se zadává objektový identifikátor, jako ukazatel na nulou ukončený řetězec a název přidávaného uzlu.

```
TMibNod* AddNod(char* FullOID, char* AName);
```

AddItemInt

Slouží k přidání položky, která bude schopna uchovávat hodnotu datového typu integer. Jako vstupní hodnoty je třeba zadat objektový identifikátor a název položky. Umístění položky je dáno objektovým identifikátorem. Chybějící uzly vedoucí k vytvářené položce jsou vytvořeny. Vrácená hodnota je ukazatel na vytvořenou položku, ale přetypovaná na objekt TMibNod nebo hodnota 0, která oznamuje selhání funkce.

```
TMibNod* AddItemInt(char* FullOID, char* AName, int AValue);
```

AddItemUnsInt

Tato metoda funguje stejně jako metoda AddItemInt, ale s tím rozdílem, že uchovávaná hodnota je datového typu unsignet integer.

```
TMibNod* AddItemUnsInt(char* FullOID, char* AName, unsigned int
                       AValue);
```

AddItemStr

Slouží k přidání položky, která bude schopna uchovávat řetězec. Chování, vstupní parametry a navrácená hodnota jest stejná jako u metod AddItemInt a AddItemUnsInt.

```
TMibNod* AddItemStr(char* FullOID, char* AName, char* AStr);
```

ReadItem

Slouží k vyčtení hodnoty položky určené objektovým identifikátorem, jež je vyžadován jako jediný vstupní parametr a je datového typu ukazatel na nulou ukončený řetězec. Jako vrácená hodnota ukazatel na nulou ukončený řetězec, který představuje hodnotu položky převedenou na řetězec a to z datových typů integer, unsignet integer a string. Pokud položka určená objektovým identifikátorem nebyla nalezena je vrácena hodnota 0.

```
char* ReadItem(char* FullOID);
```

WriteItem

Slouží k zápisu hodnoty položky určené objektovým identifikátorem, jež je předán jako vstupní parametr ve formě ukazatele na nulou ukončený řetězec. Pokud je položka nalezena a hodnotu se podaří úspěšně převést na vyžadovaný datový typ, je hodnota zapsána a funkce vrací hodnotu 1. V opačném případě je vrácena hodnota 0.

```
int WriteItem(char* FullOID, char* NewValue);
```

GetVarOID

Metoda slouží k vyhledání uzlu ve stromové struktuře databáze za využití pokročilého hledání, jež je schopno indexovat i položky v tabulce. Jako vstupní parametr je předává ukazatel na instanci třídy TIntList, jež obsahuje objektový identifikátor hledaného uzlu. Metoda vrací ukazatel na hledanou instanci třídy TMibNod. Pokud daný objekt není nalezen metody vrací hodnotu 0.

```
TMibNod* GetVarOID(TIntList* AIntList);
```

GetStrOID

Metoda zaobaluje volání metody GetVarOID() pro zadávání identifikátoru objektu v textové podobě jako nulou ukončený řetězec.

```
TMibNod* GetStrOID(char* AStrOID);
```

GetBerOID

Metoda zaobaluje volání metody GetVarOID() pro zadávání identifikátoru objektu v kódování BER. Metoda vyžaduje zadání délky v bajtech, jež objektový identifikátor zabírá.

```
TMibNod* GetBerOID(unsigned char* ASrcData, int ASrcOffset, int
ACount);
```

AddTable

Metoda slouží k přidání tabulky do stromové struktury databáze MIB. Tato metoda vytvoří instanci třídy TMibTable v umístění, jež je zadáno objektovým identifikátorem. Takto vytvořený kořenový uzel tabulky, by měl být použit nejprve k určení schématu tabulky a následnému určení sloupců pro indexování řádků tabulky. Až po těchto úkonech může následovat přidávání řádků a plnění buněk tabulky daty.

```
TMibNod* AddTable(char* FullOID, char* AName);
```

AddItemOID

Metoda slouží k vytvoření hodnotové položky v hierarchii databáze MIB, jež bude sloužit k uchování Objektového identifikátoru. Toho je docíleno vytvořením instance třídy TMibOIDItem. Hodnota položky je předávána jako instance třídy typu TIntList.

```
TMibNod* AddItemOID(char* FullOID, char* AName, int ASnmpType,
                    TIntList* AIntList);
```

AddItemStrOID

Metoda zaobaluje volání metody AddItemOID(). U toho volání může být hodnota OID vytvářené položky předávána ve formě textového řetězce, jež je teprve převeden na instanci třídy TIntList.

```
TMibNod* AddItemStrOID(char* FullOID, char* AName, int ASnmpType,
                       char* AStr);
```

FindNextItem

Funkcí této metody je vyhovět požadavkům kladených na databázi ze strany protokolu SNMP přesněji žádosti typu GetNext, když je tato metoda volána s parametrem existujícího uzlu databáze a od něho se postupuje skrz databázi a je hledán následující hodnotová položka.

```
TMibNod* FindNextItem(TMibNod* ANod);
```

5.1.2. Třída TMibNod

Úkolem třídy TMibNod je společně z dalšími třídami stejného typu, či odvozeného, tvořit hierarchickou strukturu složenou s uzlů. Tyto uzly jsou právě představovány instancemi třídami TMibNod. Proto, aby právě mohla vzniknout hierarchická struktura, třída disponuje vždy ukazatelem na následující uzel stromu, který leží na stejné úrovni a ukazatel na první položku úrovně o jedno nižší, jež tvoří jemu příslušející větev stromu. Pro identifikaci příslušnosti stromu ke správnímu objektu nejvyšší úrovně součástí třídy i ukazatel na třídu TMibDatabase. Třída taktéž slouží jako bázová třída pro odvození tříd s rozšířenými schopnostmi, kterými jsou například položky stromu, které obsahují i hodnotu. Při volání

konstruktoru třída požaduje jako vstupní parametry ukazatel na objekt TmibDatabase, v jehož stromě se bude objekt nacházet, dále ukazatel na uzel nadřazené úrovně třídy TMibNod a jako poslední identifikátor objektu jak v číselné formě, tak i textové jako ukazatel na nulou ukončený řetězec znaků.

```
TMibNod(TMibDatabase* AMibDB, TMibNod* AParent, int AOID, char*
                                AName);
```

Třída disponuje následujícími metodami:

SetNextNod

Metoda slouží k nastavení ukazatele na další uzel stromu na stejné úrovni. Jako vstupní parametr přijímá ukazatel na objekt TMibNod.

```
void SetNextNod(TMibNod* ANextNod);
```

SetFirstSubNod

Metoda provádí nastavení ukazatele na první podřízený uzel, který se nachází o jednu úroveň níže. Jako vstupní parametr metoda vyžaduje ukazatel na daný uzel typu TMibNod.

```
void SetFirstSubNod(TMibNod* AFirstSubNod);
```

GetLastSubNod

Metoda slouží ke zjištění posledního uzlu nebo položky na úrovni o jedno nižší. Jako návratová hodnota je ukazatel na objekt typu TMibNod.

```
TMibNod* GetLastSubNod();
```

GetMibDatabase

Metoda slouží k získání ukazatele na třídu TMibDatabase, které je podřízena tato celá stromová struktura a která disponuje obslužnými funkcemi pro přístup k celému stromu.

```
TMibDatabase* GetMibDatabase();
```

GetParent

Metoda slouží k získání ukazatele na nadřazený uzel stromu, kterým je objekt typu TMibNod.

```
TMibNod* GetParent();
```

GetName

Metoda vrací textový název objektu jako ukazatel na nulou ukončený řetězec znaků.

```
char* GetName();
```

GetOID

Metoda vrací celočíselný identifikátor objektu pro úroveň na níž se nachází.

```
int GetOID();
```

GetNextNod

Metoda vrací ukazatel na následující uzel stromové struktury, který se nachází na stejné úrovni. Vrácený ukazatel směřuje na objekt typu TMibNod.

```
TMibNod* GetNextNod();
```

GetFirstSubNod

Metoda vrací ukazatel na objekt typu TMibNod, který představuje první uzel na nižší úrovni, jež je tomuto objektu podřízen.

```
TMibNod* GetFirstSubNod();
```

AddSubNod

Metoda přidává uzel na nižší úroveň, jako podřízený uzel tomuto uzlu. Jako vstupní parametr přijímá celočíselný identifikátor uzlu a textový identifikátor uzlu v podobě ukazatele na nulou ukončený řetězec znaků.

```
TMibNod* AddSubNod(int ASubOID, char* AName);
```

AddNod

Metoda provádí přidání nového uzlu na stejnou úroveň, jako je tento objekt. Nový uzel je přímo umístěn za stávající a případný už existující uzel je posunut o jedno vzad. Metoda přijímá jako vstupní parametr celočíselný identifikátor uzlu a textový identifikátor uzlu v podobě ukazatele na nulou ukončený řetězec znaků.

```
TMibNod* AddNod(int ASubOID, char* AName);
```

GetLastNod

Metoda vrací ukazatel na následující uzel nebo položku na stejné úrovni, jako je tato třída. V případě, že již další uzel není je vrácena hodnota 0.

```
TMibNod* GetLastNod();
```

AddExtSubNod

Metoda slouží pro přidání existující položky do stromové struktury. Jako vstupní parametr přijímá celočíselný identifikátor položky a ukazatel na přidávanou položku přetypovanou na třídu TMibNod.

```
TMibNod* AddExtSubNod(int ASubOID, TMibNod* ASubNod);
```

GetSubNodOID

Metoda slouží k získání ukazatele na uzel či položku, která je přímo podřízena tomuto objektu. Jako vstupní parametry metoda přebírá celočíselný identifikátor položky či uzlu a

proměnnou typu integer, do které je uložena pozice předcházejícího objektu z prohledávané úrovně. Toho lze využít například při operacích mazání. Hodnota 0 je nadřazený objekt a hodnoty 1 a větší je pořadí objektu na stejné úrovni jako je žádaný uzel.

```
TMibNod* GetSubNodOID(int OID, int &Index);
```

GetSubNodCount

Vrací počet uzlů a položek podřízených tomuto objektu a nacházejících se o úroveň níže.

```
int GetSubNodCount();
```

GetSubNod

Vrací ukazatel na položku nebo uzel, jež je podřízen přímo tomuto objektu a je určen indexem, jež tato metoda přijímá jako vstupní parametr.

```
TMibNod* GetSubNod(int Index);
```

IsRootNod

Metoda provádí ověření zda je daný uzel i kořenovým uzlem stromové struktury. Pokud je objekt kořenovým uzlem stromu vrací metoda hodnotu 1, v opačném případě je vrácena hodnota 0.

```
int IsRootNod();
```

GetRootNod

Metoda slouží pro získání ukazatele na kořenový uzel celé stromové struktury. Vracený ukazatel na objekt je typu TMibNod.

```
TMibNod* GetRootNod();
```

DeleteNextNod

Metoda smaže uzel nebo položku, která se nachází za tímto objektem na stejné úrovni. V případě úspěchu je vrácena hodnota 1, v opačném případě hodnota 0.

```
int DeleteNextNod();
```

DeleteFirstSubNod

Metoda smaže první uzel nebo položku, která je o úroveň níže a je tomuto objektu podřízena.

```
int DeleteFirstSubNod();
```

DeleteSubNod

Metoda slouží ke smazání některého z uzlů nebo položek umístěných pod tímto objektem na nižší úrovni. Mazaný objekt je specifikován indexem, jež metoda přijímá jako

vstupní parametr. V případě úspěšného dokončení operace je vrácena hodnota 1 a pokud není položka nalezena je vrácena hodnota 0;

```
int DeleteSubNod(int Index);
```

DestroyAllNextNod

Metoda slouží ke zrušení všech položek a uzlů, které jsou na stejné úrovni, jako tento objekt a úroveň následují za tímto objektem.

```
void DestroyAllNextNod();
```

DestroyAllSubNod

Metoda slouží ke zrušení všech položek a uzlů, které se nacházejí pod tímto objektem ve stromové struktuře.

```
void DestroyAllSubNod();
```

SetObjType

Metoda slouží k nastavení typu objektu. Blíže viz. metoda GetObjType.

```
void SetObjType(int AObjType);
```

GetObjType

Metoda vrací typ objektu. Tato hodnota specifikuje úlohu objektu ve stromové struktuře. Vrácený typ může nabývat hodnot následujících konstant: otNod pokud je objekt uzlem nebo otItem pokud je objekt položkou s hodnotou.

```
int GetObjType();
```

IsNod

Metoda vrací hodnotu 1 pokud daný objekt vystupuje jako uzel stromové struktury. V opačném případě vrací hodnotu 0.

```
int IsNod();
```

IsItem

Metoda vrací hodnotu 1, pokud je daný objekt položka a tudíž má i hodnotu. Pro práci s položkami se využívá rozhraní TMibItem, na což je možno objekt korektně přetypovat. Pokud objekt není položka je vrácena hodnota 0.

```
int IsItem();
```

IsTable

Metoda slouží k ověření jestli je daný uzel kořenovým uzlem tabulky. Pokud ano tak vrací hodnotu 1 jinak je vrácena hodnota 0.

```
int IsTable();
```

IsTableItem

Metoda slouží k ověření jestli daný uzel stromu není hodnotovou položkou v tabulce. Pokud ano, vrací hodnotu 1, jinak je navrácena hodnota 0.

```
int IsTableItem();
```

IsTableRow

Metoda slouží k ověření, jestli je daný uzel stromu kořenovým uzlem jednoho z řádků v tabulce. Pokud je objekt typu TMibRow, tak funkce vrací hodnotu 1 a v opačném případě hodnotu 0.

```
int IsTableRow();
```

GetPreNod

Účelem této metody je zjistit, která položka leží na stejné úrovni stromu a ve stejné větvi stromu před touto položkou. Metoda nepřijímá žádné vstupní parametry a navrací odkaz na předchozí položku. Pokud předchozí položka není, vrací hodnotu 0.

```
TMibNod* GetPreNod();
```

IsFirstNodOnLevel

Metoda slouží k ověření, jestli je daná položka první ve své větvi stromu a na úrovni které se nachází. Jako návratová hodnota se vrací hodnota 1, pokud je položka první a pokud není tak hodnota 0.

```
int IsFirstNodOnLevel();
```

ChangeWithNextNode

Metoda slouží k prohození pořadí dvou uzlů ve stromu, které jsou hned za sebou na stejné úrovni a ve stejné větvi stromu. Metoda se volá na prvním z prohazovaných uzlů. Pokud již za tímto uzlem není žádný uzel umístěn metoda selže a vrátí hodnotu 0. V případě úspěchu je návratová hodnota různá od nuly.

```
int ChangeWithNextNode();
```

SetSnmpType

Metoda slouží ke změně hodnoty určující jaký datový typ bude daná položka mít z hlediska protokolu SNMP. Hodnoty by měly být nastavovány na hodnoty konstant snmpXXX. Funkce přijímá hodnoty těchto konstant jako jediný vstupní parametr.

```
void SetSnmpType(int ASnmpType);
```

GetSnmpType

Metoda slouží ke zjištění typu položky z hlediska protokolu SNMP. Tento typ určuje jak bude s daty zacházeno z hlediska zakódování informací do BER a také jak z daty bude vypadat indexování položek z tabulky.

```
int GetSnmpType();
```

CreTableItemFullOID

Tato metoda je využívána k vytvoření indexu řádku tabulky, který je předán jako parametr funkci. Tato metoda je volána mechanizmy pro zjišťování objektových identifikátorů v tabulce a provádí vytvoření závěrečné části, která identifikuje řádek. Používá k navrácení hodnoty instanci třídy TIntList, která se předává při volání metody. Přidávání jednotlivých částí identifikátoru probíhá reverzně. Konkrétní implementace tvorby identifikátoru je obsažena ve třídě TMibTable.

```
virtual int CreTableItemFullOID(TIntList* AIntList, TMibRow*
AMibRow);
```

CreFullOID

Metoda vytváří rozhranní pro vytváření objektového identifikátoru po částech. Metoda postupně prochází od zadané položky strom směrem vzhůru a postupně reverzně přidává identifikátor sebe sama a následně zavolá stejnou metodu u rodičovského uzlu. V případě, že daný uzel je v tabulce je volána metoda CreTableItemFullOID() na kořenovém uzlu tabulky. Po dokončení průchodu vzhůru se výsledný identifikátor nachází v instanci třídy TIntList, jež byla předána při volání metody.

```
virtual int CreFullOID(TIntList* AIntList);
```

GetFullOID

Metody slouží k získání identifikátoru položky ve formě třídy TIntList, na níž je vrácen ukazatel jako návratová hodnota. Pokud funkce selže, vrátí hodnotu 0. Funkce k vytvoření objektového identifikátoru využívá volání metody CreFullOID().

```
virtual TIntList* GetFullOID();
```

GetParentTable

Metoda je určena pro objekty odvozené od třídy TIntItem, jež tvoří buňky tabulky. Tato metoda slouží ke zjištění ukazatele na kořenový uzel tabulky, ve které se položka nachází.

```
TMibNod* GetParentTable();
```

GetParentTableRow

Metoda je určena pro objekty odvozené od třídy TIntItem, jež tvoří buňky tabulky. Tato metoda slouží ke zjištění ukazatele na kořenový uzel řádku tabulky, na kterém se daná položka nachází.

```
TMibNod* GetParentTableRow();
```

GetLengthBerObjectID

Metoda slouží ke zjištění počtu bajtů, které budou potřeba na zakódování objektového identifikátoru hodnotové položky za použití kódování BER. Výsledná hodnota by měla být použita k alokování patřičné velikosti paměti, jež bude předána metodě GetBerObjectID().

```
virtual int GetLengthBerObjectID();
```

GetBerObjectID

Metoda slouží k zakódování objektového identifikátoru dané položky za použití kódován BER do předaného úseku paměti (ABuffer). Zápis může být posunut za použití parametru (AOffset). Metoda vrací počet zapsaných bajtů.

```
virtual int GetBerObjectID(unsigned char* ABuffer, int AOffset);
```

GetSubNodVarOID

Metoda slouží k vyhledání instance podřízeného uzlu ve stromové struktuře na základě objektového identifikátoru proměnné délky, jež je předán jako instance třídy typu TIntList. Konkrétní zpracovávaný úsek objektového identifikátoru je specifikován za použití vlastnosti offset. Metoda může použít více hodnot nežli jednu pokud to je zapotřebí a podle počtu použitých částí inkrementuje vlastnost offset. Díky tomu je možno stejnou instanci třídy TIntList okamžitě znovu použít pro hledání na další nižší úrovni.

```
virtual TMibNod* GetSubNodVarOID(TIntList* AIntList);
```

5.1.3. Třída TMibItem

Představuje rozhraní pro jednotný přístup k položkám stromu, jež slouží k uchování hodnot různých datových typů. Její bázovou třídou je TMibNod. Díky tomu mohou položky a uzly navzájem existovat v jednom stromu. Tato třída definuje rozhraní pro přístup k paměťovým položkám, které budou implementovány jako třída, jež bude potomkem této třídy. Díky tomu bude možno využít virtuálních metod, které tato třída zavádí a jež budou na úrovni nižších tříd předefinovány funkcemi, které budou specifické pro každý datový typ. Příkladem využití tohoto principu je zjištění uchované hodnoty z výstupem ve formě řetězce a nebo nastavení hodnoty. V konstruktoru třída přijímá ukazatel na nadřazený uzel stromu typu

TMibNod, číselný identifikátor položky, textový identifikátor položky a číselný identifikátor reprezentující typ uchovávané proměnné. Třída má konstruktor s touto hlavičkou:

```
TMibItem(TMibNod* AParent, int AOID, char* AName, int ALinkType);
```

GetLinkTyp

Slouží pro získání číselného identifikátoru, který určuje datový typ proměnné, slouží k uchování hodnot položky. Jako výstup metoda vrací datový typ integer.

```
int GetLinkTyp();
```

SetLinkType

Metoda umožňuje změnu datového typu uchovávané informace dané položky. Jako vstupní parametr metoda přijímá celé číslo reprezentující datový typ a ukazatel na proměnnou, jež bude sloužit k uchování hodnoty položky.

```
void SetLinkType(int ALinkType, void* ALink);
```

GetLink

Účelem této metody je získání ukazatele na prostor paměti, v němž je uložena aktuální hodnota položky. Při zavolání metody vrací obecný ukazatel. Ve spojení s metodou GetLinkType, lze získaný obecný ukazatel přetypovat na ukazatel na konkrétní datový typ a je možno s danou hodnotou libovolně pracovat.

```
void* GetLink();
```

SetLink

Slouží ke změně adresy, která ukazuje na prostor v paměti, kde je uložena aktuální hodnota položky. Jako vstupní parametr se zadává obecný ukazatel na místo v paměti. Při použití této metody je nutno pamatovat na uvolnění starého paměťového prostoru a zároveň nové umístění musí být legitimně alokováno. Je nutné zachovat velikost vyhrazené paměti pro hodnotu, jež je specifikována vlastností LinkTyp, jež lze získat metodou GetLinkTyp.

```
void SetLink(void* ALink);
```

GetValueToStr

Metoda slouží k získání hodnoty položky. Návratovou hodnotou je ukazatel na nulou ukončený řetězec a proto součástí implementace této metody v potomcích musí být konverzní algoritmus pro hodnoty jiných datových typů, než je návratová hodnota. Tato metoda je virtuální a jelikož tato třída není určena k přímému využití, ale jen jako definice rozhranní pro potomky této třídy, je nutné konkrétní implementaci této metody hledat v třídách potomků.

```
virtual char* GetValueToStr();
```

SetStrToValue

Metoda slouží k nastavení nové hodnoty položky. Vstupním parametrem je nová hodnota v textové reprezentaci, předávaná jako ukazatel na nulou ukončený řetězec. Jako návratová hodnota je vrácen datový typ integer, který nabývá hodnot 0 a 1. Hodnota 0 znamená selhání při metodě a hodnota 1 značí úspěšné nastavení nové hodnoty. Tato metoda je rovněž virtuální a proto pro ni platí stejná pravidla jako pro metodu GetValueToStr.

```
virtual int SetStrToValue(char* NewValue);
```

CompareValue

Metoda je abstraktním rozhranním zprostředkovávající možnost porovnat dvě hodnotové položky stejného typu,jež jsou uloženy na stejné úrovni ve stejné větvi stromu. Konkrétní mechanizmy porovnání jsou implementovány v odvozených třídách.

```
virtual int CompareValue(TMibItem* AMibItem);
```

CreValueOID

Metoda je abstraktním rozhranním zprostředkovávající přístup pro získání části objektového identifikátoru z hodnoty položky. Této metody se využívá při indexování tabulky přes hodnoty sloupců. Konkrétní mechanizmy k vytvoření identifikátoru z hodnoty jsou implementovány v odvozených třídách.

```
virtual int CreValueOID(TIntList* AIntList);
```

GetLengthBerValue

Metoda je abstraktním rozhranním zprostředkovávající počet bajtů, jež bude zabírat konkrétní hodnoty položky v kódování BER. Této metody se využívá při alokaci paměti, jež bude předána při volání metody GetBerValue(). Konkrétní mechanizmy k výpočtu jsou implementovány v odvozených třídách.

```
virtual int GetLengthBerValue();
```

GetLengthBerVarbind

Metoda je abstraktním rozhranním zprostředkovávající počet bajtů, jež bude zabírat konkrétní položka včetně objektového identifikátoru v kódování BER. Této metody se využívá při alokaci paměti, jež bude předána při volání metody GetBerVarbind(). Konkrétní mechanizmy k výpočtu jsou implementovány v odvozených třídách.

```
virtual int GetLengthBerVarbind();
```

GetBerValue

Metoda je abstraktním rozhranním zprostředkovávající přístup pro získání hodnoty položky databáze MIB v kódování BER. Konkrétní mechanizmy k zakódování jednotlivých datových typů jsou implementovány v odvozených třídách.

```
virtual int GetBerValue(unsigned char* ABuffer, int AOffset);
```

GetBerVarbind

Metoda je abstraktním rozhranním zprostředkovávající přístup pro získání hodnotové položky včetně objektového identifikátoru v kódování BER. Konkrétní mechanizmy k zakódování jednotlivých datových typů jsou implementovány v odvozených třídách.

```
virtual int GetBerVarbind(unsigned char* ABuffer, int AOffset);
```

5.1.4. Třída TMibIntItem

Slouží jako položka ve stromu, která uchovává hodnotu v datovém typu Integer. Jejím předkem je třída TMibItem, díky čemuž můžme objekt zařadit do stromové struktury a můžeme přistupovat k hodnotě položky přes univerzální rozhraní třídy TMibItem, pro něž je důležité, aby třída implementovala virtuální metody GetValueToStr a SetStrToValue, jež jsou pro správnou funkčnost nutné. Třída přijímá při volání konstruktoru, jako vstupní parametry ukazatel na nadřazený objekt stromu TMibNod, číselný identifikátor a název objektu. Tyto parametry jsou předány konstruktoru nadřazené třídy TMibItem.

```
TMibIntItem(TMibNod* AParent, int AOID, char* AName);
```

Pro práci s hodnotami položky třída disponuje metodami:

GetValue

Metoda slouží k získání hodnoty položky, která je vrácena, jako návratová hodnota datového typu integer.

```
int GetValue();
```

SetValue

Metoda je určena k nastavení nové hodnoty položky. Jako vstupní parametr přijímá zmíněnou novou hodnotu datového typu integer.

```
void SetValue(int AValue);
```

GetValueToStr

Metoda je virtuální a je určena k převodu aktuální hodnoty položky z datového typu integer na řetězec. K převodu je použita funkce MibIntToStr.

```
virtual char* GetValueToStr();
```

SetStrToValue

Metoda je virtuální a je určena k nastavení hodnoty položky. Jako vstupní parametr přijímá ukazatel na nulou ukončený řetězec, který musí být formátem celé číslo se znaménkem, které svým rozsahem nepřekračuje datový typ integer. Pro převod se používá funkce MibStrToInt. V případě selhání funkce je vrácena hodnota 0. Pokud byly hodnoty úspěšně nastaveny, je vrácena hodnota 1.

```
virtual int SetStrToValue(char* NewValue);
```

Metody implementující abstraktní rozhranní třídy TMibItem

- `virtual int CompareValue(TMibItem* AMibItem);`
- `virtual int CreValueOID(TIntList* AIntList);`
- `virtual int GetLengthBerValue();`
- `virtual int GetBerValue(unsigned char* ABuffer, int AOffset);`

5.1.5. Třída TMibUnsIntItem

Je určena k uchování hodnoty datového typu unsigned integer, jako součást stromu. Třída funguje stejně jako třída TmibIntItem. Z důvodu jiného datového typu jsou použity jiné převodní funkce v implementaci metod GetValueToStr a SetStrToValue. U první jmenované to je funkce MibUnsIntToStr a druhá jmenovaná využívá funkce MibStrToUnsInt.

```
TMibUnsIntItem(TMibNod* AParent, int AOID, char* AName);
```

Třída disponuje těmito metodami:

- GetValue
- SetValue
- GetValueToStr
- SetStrToValue

5.1.6. Třída TMibStrItem

Slouží k uchování textu v položce stromu. Stejně jako třída TMibIntItem a třída TMibUnsIntItem je potomkem třídy TMibItem. Třída je velmi podobná oběma předchozím třídám. Text uchovává jako ukazatel na nulou ukončený řetězec znaků, který je skladován ve vyhrazené části paměti konkrétní instance objektu. Hlavička konstruktoru vypadá následovně:

```
TMibStrItem(TMibNod* AParent, int AOID, char* AName);
```

Třída nabízí následující metody:

- GetStr
- SetStr

- GetValueToStr
- SetStrToValue

GetStr

Metoda slouží k získání textové informace, kterou tato třída zpřístupňuje. K tomuto účelu alokuje paměť o velikosti daného řetězce a udělá do vyhrazeného prostoru paměti kopii textového řetězce. Následně, jako návratovou hodnotu vrací ukazatel na onu kopii nulou ukončeného řetězec znaků. Z toho plyne nutnost, aby se příjemce návratové hodnoty postaral o uvolnění paměti, až řetězec nebude potřebovat.

```
char* GetStr();
```

SetStr

Účelem metody je nastavit textové hodnoty položky. Metoda provádí alokaci paměti o velikosti předaného řetězce. Následně do takto vytvořeného prostoru si udělá vlastní kopii řetězce. Jako vstupní parametr je přijímám ukazatel na nulou ukončený řetězec.

```
void SetStr(char* AStr);
```

GetValueToStr

Metoda je virtuální a slouží k získání textu z položky. Její implementace je nutná pro spolupráci s rozhraním TMibItem. K implementaci je použita metoda GetStr, která navrací požadovanou hodnotu položky v textové podobě.

```
virtual char* GetValueToStr();
```

SetStrToValue

Metoda je taktéž virtuální a slouží ke spolupráci s rozhraním TMibItem. Provádí nastavení uchovaného textu na novou hodnotu. Pro implementaci metody je využito služeb metody SetStr. Metoda přijímá jako vstupní parametr ukazatel na nulou ukončený řetězec. Jako návratovou hodnotu vždy vrací hodnotu 1.

```
virtual int SetStrToValue(char* NewValue);
```

Metody implementující abstraktní rozhranní třídy TMibItem

- `virtual int CompareValue(TMibItem* AMibItem);`
- `virtual int CreValueOID(TIntList* AIntList);`
- `virtual int GetLengthBerValue();`
- `virtual int GetBerValue(unsigned char* ABuffer, int AOffset);`

5.1.7. Třída TMibOIDItem

Účelem této třídy je uchovávat hodnotu objektového identifikátoru jako položku ve stromu systémové databáze MIB. Třída je potom odvozena od třídy TMibItem a proto implementuje všechny povinné metody, jež jsou definovány abstraktním rozhranním třídy TMibItem. K uchovávání objektového identifikátoru si třída vytváří privátní instanci třídy TIntList. Díky tomu je velmi snadná spolupráce s celou databází. Třída jako parametry konstruktoru přijímá odkaz na nadřazený uzel stromu (AParent). Poslední část vlastního objektového identifikátoru (AOID), svůj název v textové formě (AName). A také číslo reprezentující daný datový typ pro protokol SNMP jako je například konstanta snmpOID.

```
TMibOIDItem(TMibNod* AParent, int AOID, char* AName, int ASnmpType);
```

Metoda GetValueOID

Funkce je určená ke zjištění hodnoty objektového identifikátoru, který je touto položkou uchováván. Hodnota je navrácena jako řada celých čísel uložených v instanci třídy TIntList, na nějž funkce vrací ukazatel.

```
TIntList* GetValueOID();
```

Metoda SetValueOID

Funkce je určena k nastavení hodnoty objektového identifikátoru, který tato položka stromu uchovává. Hodnota objektového identifikátoru musí být předána jako ukazatel na instanci třídy typu TIntList.

```
void SetValueOID(TIntList* AIntList);
```

Metoda GetValueToStr

Tato metoda provádí převod aktuální hodnoty objektového identifikátoru položky na textovou reprezentaci. K tomuto účelu využívá funkce IntListToStr(). Jako návratovou hodnotu vrací ukazatel na nulou ukončený řetězec znaků, který je ve formátu řady číslic oddělených tečkou.

```
virtual char* GetValueToStr();
```

Metoda SetStrToValue

Účelem této metody je implementovat abstraktní rozhranní rodičovské třídy TMibItem na konkrétní datový typ, v tomto případě objektový identifikátor a provést nastavení hodnoty položky na hodnotu předanou ve formě speciálně formátovaného řetězce. V případě neúspěchu převodu funkce vrací hodnotu 0 a v případě úspěšného nastavení vrací funkce hodnotu 1.

```
virtual int SetStrToValue(char* NewValue);
```

Metoda CompareValue

Účelem metody je implementovat porovnání hodnot dvou položek stromu, jež jsou stejného datového typu. Jedná se o konkrétní implementaci abstraktního rozhranní rodičovské třídy TMibItem. Metody přijímá jako vstupní parametr ukazatel na položku třídy, se kterou má být hodnota porovnána. Pokud položky nabudou stejného datového typu funkce, navrátí hodnotu 0. Pokud je tato položka, u které metodu voláme menší, než položka předaná jako parametr volání, návratová hodnota bude 2, v opačném případě bude návratová hodnota 3 a nebo pokud budou mít položky stejnou hodnotu bude návratová hodnota 1.

```
virtual int CompareValue(TMibItem* AMibItem);
```

Metoda CreValueOID

Metoda zkouší k získání objektového identifikátoru z hodnoty položky, čehož se využívá u položek v tabulce, jež jsou ve sloupečcích přes které je vytvořen index tabulky. Tato metoda pochází z abstraktního rozhranní třídy TMibItem. Vytvořená hodnota se reverzně vloží do instance třídy TIntList, na kterou se předává ukazatel při volání metody. Metoda vrací nenulovou hodnotu v případě úspěchu a v případě neúspěchu vrací hodnotu 0.

```
virtual int CreValueOID(TIntList* AIntList);
```

Metoda GetLengthBerValue

Metoda je součástí rozhranní třídy TMibNod a implementuje zjištění velikosti paměti, jež zabere hodnota této třídy v kódování BER. Velikost je vrácena jako počet bajtů paměti, které by měly být alokovány před voláním funkce GetBerValue().

```
virtual int GetLengthBerValue();
```

Metoda GetBerValue

Metoda je součástí rozhranní třídy TMibNod a implementuje zakódování hodnoty této třídy v kódování BER do určitého paměťového prostoru, jež je předáván jako parametr ABuffer. Jako druhý parametr se zadává posun v paměti, do které se bude kopírovat. Jako návratová hodnota je vrácen počet zapsaných bajtů.

```
virtual int GetBerValue(unsigned char* ABuffer, int AOffset);
```

Metody implementující abstraktní rozhranní třídy TMibItem

- `virtual int CompareValue(TMibItem* AMibItem);`
- `virtual int CreValueOID(TIntList* AIntList);`
- `virtual int GetLengthBerValue();`
- `virtual int GetBerValue(unsigned char* ABuffer, int AOffset);`

5.1.8. Třída TMibRow

Třída slouží k jednoduššímu uspořádání paměťových položek v tabulce, když položky, jež leží na jednom řádku jsou umístěny právě pod tuto třídu, čímž je s nimi možno lépe manipulovat. Hlavním úkolem třídy je usnadnit přemísťování většího množství položek, což jsou v tomto případě položky na jednom řádku při změně uspořádání řádku v tabulce, jež je vynuceno změnou parametrů majících vliv na seřazení řádků v tabulce. Takovým důvodem může být změna hodnoty buňky, jež je součástí indexování tabulky nebo například přidání nového řádku. Třída je odvozena z třídy typu TMibNod. Tato třída je vždy podřízenou položkou třídy TMibTable ve výsledné stromové struktuře.

```
TMibRow(TMibNod* AParent, int AOID, char* AName);
```

Třída přináší navíc k rozhranní třídy TmibNod, které zdědila jen dvě další metody. Jednu pro zjištění počtu položek na řádku, což je metoda GetColumnCount() a jako další metodu GetColumnItem(), jež vrací konkrétní hodnotovou položku z řádku, jež je specifikována indexem předaným jako vstupní parametr metodě. Index může nabývat hodnoty 0 až GetColumnItem () – 1.

5.1.9. Struktura STabSchItem

Jejím účelem je popisovat položku schématu tabulky. Určuje jaký bude mít daný sloupec název a jaký bude mít datový typ a typ SNMP.

5.1.10. Třída TMibTable

Tato třída se umisťuje do stromové struktury databáze MIB na místo položky s hodnotou a reprezentuje tabulku v systémové databázi. Třída spravuje schéma celé tabulky. Toto schéma musí být nastaveno na začátku před započetím vkládání hodnot a dále by nemělo být měněno, aby nedošlo ke vzniku nekorektních dat. Na základě tohoto schématu potom tato třída pomocí metody AddRow() umožňuje do tabulky přidávat řádky, jež jsou tvořeny instancí třídy TMibRow, který se přímo naváže na tuto třídu a následně se podle schématu vytvoří instance hodnotových položek (např. TMibIntItem atd…), jež se napojí jako podpoložky na třídu TMibRow. Pro správu schématu je třída vybavena metodami GetSchemaCount() pro zjištění počtu schématických položek a GetSchemaItem() pro zjištění konkrétní položky schématu na využití struktury STabSchItem. Tato třída má na starosti i správu indexování tabulky a tudíž je vybavena metodami pro určení indexu jako je AddSchemaItem() . Index se ukládá jako indexy odpovídajících schématických položek. Takto je možno index vytvořit nad více sloupci tabulky, čímž vznikne složený primární klíč.

Třída je odvozena z třídy TMibNod, tudíž rozšiřuje toto rozhranní a upravuje metody specifiké pro práci s tabulami, jako je například metoda GetSubNodVarOID(), jež je určena k hledání podřízených položek ve stromové hierarchii. Tato metoda v případě tabulek volá metodu CreTableItemFullOID(), jež vytvoří část objektového identifikátoru, pro konkrétní záznam v tabulce za pomocí informací o indexování tabulky a o sloupcích jakými tabulka disponuje. Konstruktor třídy je stejný jako má rodičovská třída TMibNod.

```
TMibTable(TMibNod* AParent, int AOID, char* AName);
```

Metody správy schématu tabulky

- `int GetSchemaCount();`
- `STabSchItem GetSchemaItem(int Index);`
- `int AddSchemaItem(STabSchItem AItem);`
- `int AddSchemaItem(int Atyp_int, int Atype_ext, char* AName);`

Metody správy indexování tabulky

- `int GetIndexCount();`
- `int GetIndexItem(int Index);`
- `int AddIndexItem(int Index);`

Metoda AddRow

Metoda vytvoří nový řádek v tabulce. Operace se skládá s vytvoření instance TMibRow a zařazení ji jako podřízenou položku tohoto uzlu. Následně se pod touto instancí vytvoří podle schématu instance odpovídajících tříd a nastaví se na výchozí hodnoty. Tudíž je nutné řádek následně naplnit patřičnými daty přes přímé přistupování k jednotlivým buňkám na řádku.

```
TMibRow* AddRow();
```

Metoda SortRow

Tato metoda se musí volat vždy po provedení změn v sloupcích, jež slouží jako index tabulky. Těmito situacemi je změna hodnoty buňky a nebo při přidání nového řádku je potřeba po provedení naplnění řádku hodnotami zavolat tuto metodu. Metoda provede průchod informacemi indexů a podle nich provede seřazení řádků od nejmenšího indexu po největší.

```
void SortRow();
```

Metoda GetRowItem

Metoda je určená k získání odkazu na instanci třídy konkrétního řádku tabulky. Index se zadává jako vstupní parament při volání a jedná se o pořadový index.

```
TMibRow* GetRowItem(int Index);
```

Metoda GetRowCount

Slouží k získání počtu řádků v tabulce. K tomuto účelu využívá informací ze schématu. Za tímto účelem využívá metod třídy TmibNod, kdy projde všechny potomky tohoto uzlu a testuje je zda-li jsou to třídy typu TMibRow.

```
int GetRowCount();
```

Metoda GetColumnItem

Metoda slouží k získání objektu konkrétní buňky tabulky. Metoda přijímá pořadový index řádku a pořadový index sloupce. Pro zjištění položky z řádku volá metodu GetColumnItem jež náleží třídě TMibRow konkrétního řádku.

```
TMibItem* GetColumnItem(int RowIndex, int ColIndex);
```

Metoda DeleteRow

Metoda slouží ke smazání celého řádku tabulky se všemi hodnotami. Řádek je určen pořadovým indexem, jež se předává jako jediný vstupní parametr.

```
int DeleteRow(int Index);
```

Metoda CreTableItemFullOID

Jejím účelem je vytvořit index pro zadaný řádek tabulky, jež je specifikován odkazem AMibRow. Index je tvořen na základě indexovacích a schématických informací o tabulce. Tvořený index řádku je pouze částečný a je reverzně přidávám do třídy AintList, jež je předána jako vstupní parametr. Jako návratová hodnota je vraceno nenulové číslo v případě úspěchu a v případě chyby je hodnota 0.

```
virtual int CreTableItemFullOID(TIntList* AIntList, TMibRow*
                                AMibRow);
```

Metoda GetSubNodVarOID

Metoda upravuje rozhranní rodičovské třídy TMibNod z důvodu specifických vlastností tabulek. Tato třída proto volá metodu CreTableItemFullOID(). Účelem metody je kompletování objektového identifikátoru z údajů uložených v jednotlivých uzlech, jež jsou po cestě k hodnotovým položkám.

```
virtual TMibNod* GetSubNodVarOID(TIntList* AIntList);
```

5.1.11. Převodní funkce, funkce podpory kódování a podpůrné třídy

Funkce MibIntToStr

Slouží k převodu celého čísla se znaménkem na řetězec. Jako vstupní parametr přijímá proměnnou typu integer, jež převeden na řetězec ukončený nulou a vrátí ukazatel na první znak.

```
char* MibIntToStr(int AValue)
```

Funkce MibUnsIntToStr

Slouží k převodu neznaménkového celého čísla na řetězec. Jako vstupní parametr přijímá proměnnou typu unsigned integer, jež převede na řetězec ukončený nulou a vrátí ukazatel na první znak.

```
char* MibUnsIntToStr(unsigned int AValue)
```

Funkce MibStrToInt

Slouží k převodu řetězce na celé číslo se znaménkem. Jako vstupní parametr přijímá proměnnou typu ukazatel na nulou ukončený řetězec a existující proměnnou tepu integer, do níž bude převedené číslo uloženo. Funkce vrací proměnnou typu integer a to nabývající hodnoty 1, pokud se převod zdařil nebo hodnoty 0 pokud převod selhal.

```
int MibStrToInt(char* Str, int &Value)
```

Funkce MibStrToUnsInt

Slouží k převodu řetězce na neznaménkové celé číslo. Jako vstupní parametr přijímá proměnnou typu ukazatel na nulou ukončený řetězec a existující proměnnou typu unsigned integer, do níž bude převedené číslo uloženo. Funkce vrací proměnnou typu integer, a to nabývající hodnoty 1, pokud se převod zdařil, nebo hodnoty 0 pokud převod selhal.

```
int MibStrToUnsInt(char* Str, unsigned int &Value)
```

Třída TIntList

Slouží k uchování posloupnosti celých čísel. Třída jako vstupní parametr konstruktoru přijímá celé číslo udávající počet celých čísel, které bude třída uchovávat. Všechny takto alokované paměťové bloky jsou nastaveny na výchozí hodnotu 0. Pro práci s tímto seznamem čísel je možno použít 3 metod:

- GetCount
- GetValue
- SeValue

Metoda GetCount vrací celé číslo, jež udává délku seznamu. Pro přímou práci s hodnotami uchovaných celých čísel souží metoda GetValue, která přijímá celé číslo udávající pořadí položky v seznamu a vrací hodnotu dané položky. Pro nastavení vybrané položky slouží metoda SetValue, jež jako vstupní parametr přijímá celé číslo udávající pořadí položky v seznamu a jako druhý parametr se zadává nová hodnota položky. První položka je indexována hodnotou 0.

Vzhledem k tomu, že je třída využívána k uchovávání objektových identifikátorů, jak dlouhodobě, tak jako mezistupeň při předávání mezi jednotlivými funkčními bloky zpracování dat, je do této třídy přidána stavová proměnná pro uchování aktuálního posunutí. Tato proměnná se obsluhuje metodami GetOffset a SetOffset. Účelem je ukazovat na určité číslo v řadě uchovávaných hodnot, aby bylo možno předat dalším funkčním blokům informaci o hodnotě indexu, kde mají zahájit svoji činnost. Hodnota tohoto posunutí se smí pohybovat od hodnoty 0 až do hodnoty počtu prvků jež lze zjistit voláním metody GetCount(). Platné hodnoty posunutí jsou do hodnoty o jedno menší než je počet prvků. Pokud je hodnota stejná jako je počet prvků uchovávaných, znamená to, že ukazatel offsetu nabyl maximální možné hodnoty a paměť již nejde s jeho hodnotou dále indexovat. K usnadnění testování vzniku tohoto stavu slouží metoda IsEndOffset(), která vrací hodnotu 1 nebo 0 podle toho, zda je či není ukazatel posunutí ve své maximální hodnotě.

Pro možnost postupného rozšiřování délky řady uchovávaných celých čísel byla třída vybavena možností dynamického navyšování své velikosti. Tato možnost je přístupná přes metodu AddValue() a AddValueRev(). Metoda AddValue() slouží k přidání jedné hodnoty na konec řady. Metoda AddValueRev() slouží k přidání jedné hodnoty na začátek řady.

Funkce StrToIntList

Slouží k převodu řetězce, který má formát objektového identifikátoru databáze MIB na seznam proměnných typu integer, jež je v tomto případě reprezentován objektem TIntList. Jako vstupní parametr přijímá proměnnou typu ukazatel na nulou ukončený řetězec a navrací hodnotu typu ukazatel na objekt TintList. V případě selhání převodu vrací hodnotu 0.

```
TIntList*  StrToIntList(char* S);
```

Funkce IntListToStr

Slouží k převodu posloupnosti hodnot uložených ve třídě TIntList na textovou reprezentaci v podání nulou ukončeného řetězce. Navrácený textový řetězec má formu řady číslic, jež jsou odděleny tečkou. Jako vstupní parametr funkce přijímá ukazatel na objekt typu

TIntList, jehož hodnoty požadujeme převést. Jako návratová hodnota je předávám ukazatel na nulou ukončený řetězec. V případě selhání převodu funkce vrací hodnotu 0.

```
char* IntListToStr(TIntList* AIntList);
```

Funkce MibStrComp

Slouží k porovnání dvou nulou ukončených řetězců. Tyto vstupní řetězce jsou předány ve formě odkazů. Porovnání je realizováno přes postupné porovnávání hodnot odpovídajících si oktetů řetězců. Funkce vrací hodnotu nula pokud jsou si řetězce rovny. Pokud je řetězec A větší než řetězec B je vrácena kladná návratová hodnota a pokud je řetězec B větší než řetězec A je vrácena záporná hodnota.

```
int MibStrComp(char* StrA, char* StrB);
```

Funkce GetLengthInt

Funkce slouží k výpočtu, kolik bajtů bude potřebných k vyjádření určité hodnoty. Funkce jako vstupní parametr přijímá celočíselnou hodnotu, pro níž má být potřebný počet bajtů zjištěn a je vrácen jako návratová hodnota.

```
int GetLengthInt(unsigned int AValue);
```

Funkce SaveInt

Slouží s zápisu celočíselné hodnoty do paměťového prostoru proměnné velikosti, a to podle potřebného počtu bajtů k vyjádření dané hodnoty. Funkce přijímá jako vstupní parametr Avalue, což je hodnota zapisovaného čísla, ASrcOffsett jež je celočíselnou hodnotou specifikující posunutí adresy zápisu a jako ABuffer adresu paměťového prostoru, ke které bude přičten právě offset a následně od takto získané adresy bude proveden zápis požadované hodnoty. Návratová hodnota funkce udává kolik paměťových bajtů bylo zápisem obsazeno a lze ji využít k inkrementaci hodnoty offsetu pro další zápis dat.

```
int SaveInt(unsigned int AValue, int ASrcOffset, unsigned char*
                                                    ABuffer);
```

Funkce GetLengthBerInt

Funkce slouží ke zjištění, kolik bajtů bude potřeba k vyjádření celočíselné hodnoty v kódování BER. Této funkce je využíváno při implementaci metod podporujících spolupráci MIB s protokolem SNMP. Jejím účelem je zjistit potřebné množství paměti, která se musí alokovat a předat při volání funkce SaveBerInt(), jež provede fyzický zápis dané hodnoty do tohoto alokovaného paměťového prostoru.

```
int GetLengthBerInt(unsigned int AValue);
```

Funkce SaveBerInt

Slouží s zápisu celočíselné hodnoty do paměťového prostoru v kódování BER. Funkce přijímá jako vstupní parametr AValue hodnotu zapisovaného čísla. Jako ASrcOffsett celočíselnou hodnotu specifikující posunutí adresy zápisu a jako ABuffer adresu paměťového prostoru, kam bude zápis hodnoty v kódování BER proveden. Návratová hodnota funkce udává kolik paměťových bajtů bylo zápisem obsazeno a lze ji využít k inkrementaci hodnoty offsetu pro další zápis hodnot.

```
int SaveBerInt(unsigned int AValue, int ASrcOffset, unsigned char*
                                ABuffer);
```

Funkce LoadBerInt

Funkce je určená k načítání hodnoty celých čísel v kódování BER. Jako vstupní parametry vyžaduje adresu paměti odkud bude načítáno (ASrcData). Jako další parametr je zadávána hodnota posunutí adresy čtení (ASrcOffset). A jako poslední je vyžadován ukazatel na proměnnou typu integer, do které bude dekódovaná hodnota zapsána.

```
int LoadBerInt(unsigned char* ASrcData, int ASrcOffset, int*
                                AValue);
```

Funkce GetLengthBerOID

Funkce slouží ke zjištění délky paměti ,jež bude potřeba k reprezentaci určité hodnoty objektového identifikátoru v kódování BER. Jako vstupní parametr se předává konkrétní objektový identifikátor, jež je uložen v objektu typu TintList, jehož ukazatel se funkci předá. Funkce vrací počet bajtů, který daný objektový identifikátor v kódování BER potřebuje.

```
int GetLengthBerOID(TIntList* AIntList);
```

Funkce SaveBerOID

Funkce slouží k uložení hodnoty objektového identifikátoru v kódování BER. Jako vstupní parametry přijímá jako AintList ukazatel na třídu typu TintList, ve které je daný objektorý identifikátor uložen. Dalším parametrem je hodnota posunutí adresy zápisu (AOffset) a ukazatel na paměť, do níž bude zapisováno (ABuffer). Paměť musí být předem alokována v potřebné velikosti, jež lze zjistit voláním funkce GetLengthBerOID().

```
int SaveBerOID(TIntList* AIntList, int AOffset, unsigned char*
                                ABuffer);
```

Funkce LoadVarOID

Funkce provádí načtení objektového identifikátoru, jež je uložen v kódování BER a následně takto vyčtenou hodnotu zapíše do třídy typu TIntList, jejíž instanci k tomuto účelu vytvoří a vrací nám ukazatel na tuto třídu jako návratovou hodnotu. Pokud při provádění

funkce dojde k chybě, například čtením nekorektních dat, funkce vrací hodnotu 0. Jako vstupní parametry se předává ukazatel na čtenou paměť (ASrcData), posunutí začátku čtené adresy (ASrcOffset) a počet bajtů, které budou čteny.

```
TIntList* LoadVarOID(unsigned char* ASrcData, int ASrcOffset, int
                                                    ACount);
```

Funkce CopyBuf

Funkce je určena k vytváření kopií sekvencí paměti do jiné části paměti. Jako vstupní parametry přijímá adresu zdrojových dat (ASrcData), posunutí začátku kopírování (ASrcOffset), adresu cílového umístění kopie (ADstData) a posunutí zápisu do cílového umístění (ADstOffset). Jako poslední je vyžadován počet bajtů, jež se mají zkopírovat. Funkce vrací jako návratovou hodnotu počet zkopírovaných bajtů. Tuto hodnotu lze využít k inkrementaci offsetu pro další zápis.

```
int CopyBuf(char* ASrcData, char* ADstData, int ASrcOffset, int
                                    ADstOffset, int ACount);
```

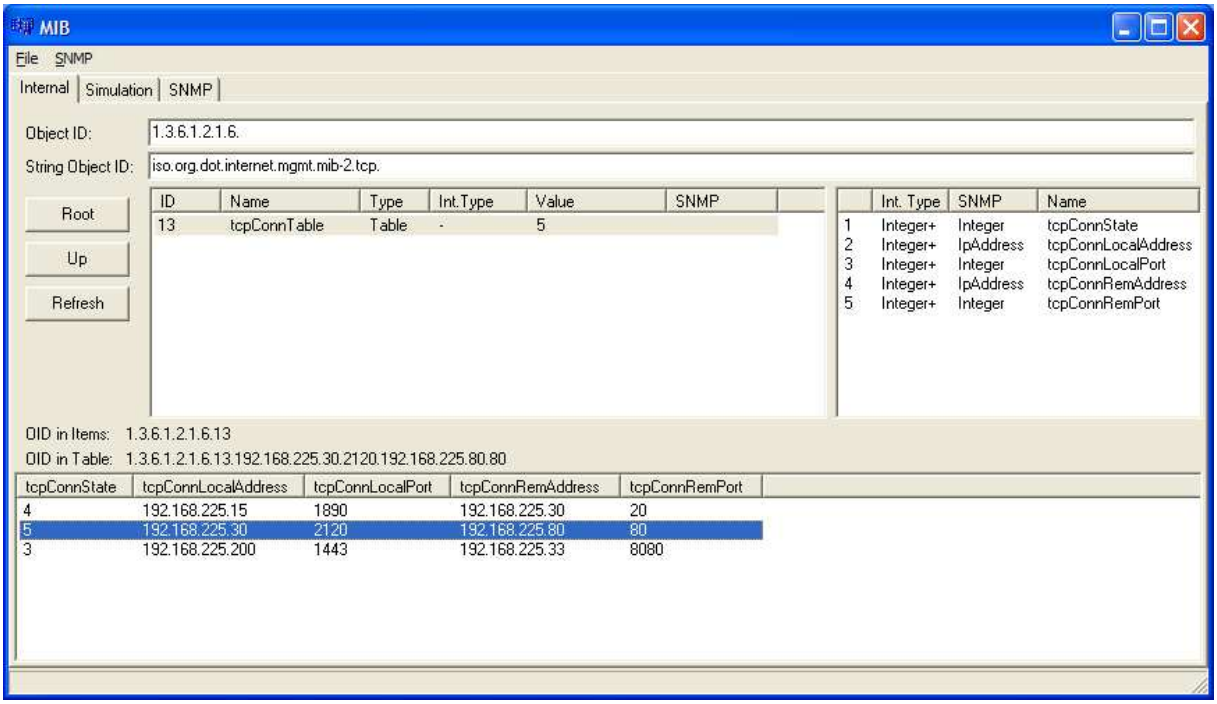
Funkce GetObjectTypeName a GetSnmpTypeName

Funkce slouží pro získání textové reprezentace hodnoty konstant datových typů se kterými jednotlivé třídy pracují. Jako vstupní parametr přijímají hodnotu datového typu a vrací ukazatel na nulou ukončený řetězec.

- `char* GetObjectTypeName(int ASnmpType);`
- `char* GetSnmpTypeName(int ASnmpType);`

6. Ukázková aplikace

Ukázková aplikace, jež je postavena na základě vytvořených tříd pro databázi MIB byla naprogramována ve vývojovém prostředí C++ Builder 6 a pro grafické prostředí využívá VCL (Visual Component Library). Tato aplikace vznikla jako vedlejší produkt při tvorbě implementace databáze MIB a sloužila ke zjednodušení testování vytvářených tříd a k vizualizaci stromové struktury, jež celá databáze nabývá. Postupným vývojem byla nakonec provedena úprava aplikace i pro možnost uživatelsky přehlednějšího a příjemnějšího přístupu. Aplikace si v současné době klade hlavně za cíl ukázat, jakým způsobem se pracuje s vytvořenými třídami a jak realizovat různé druhy přístupu do databáze.

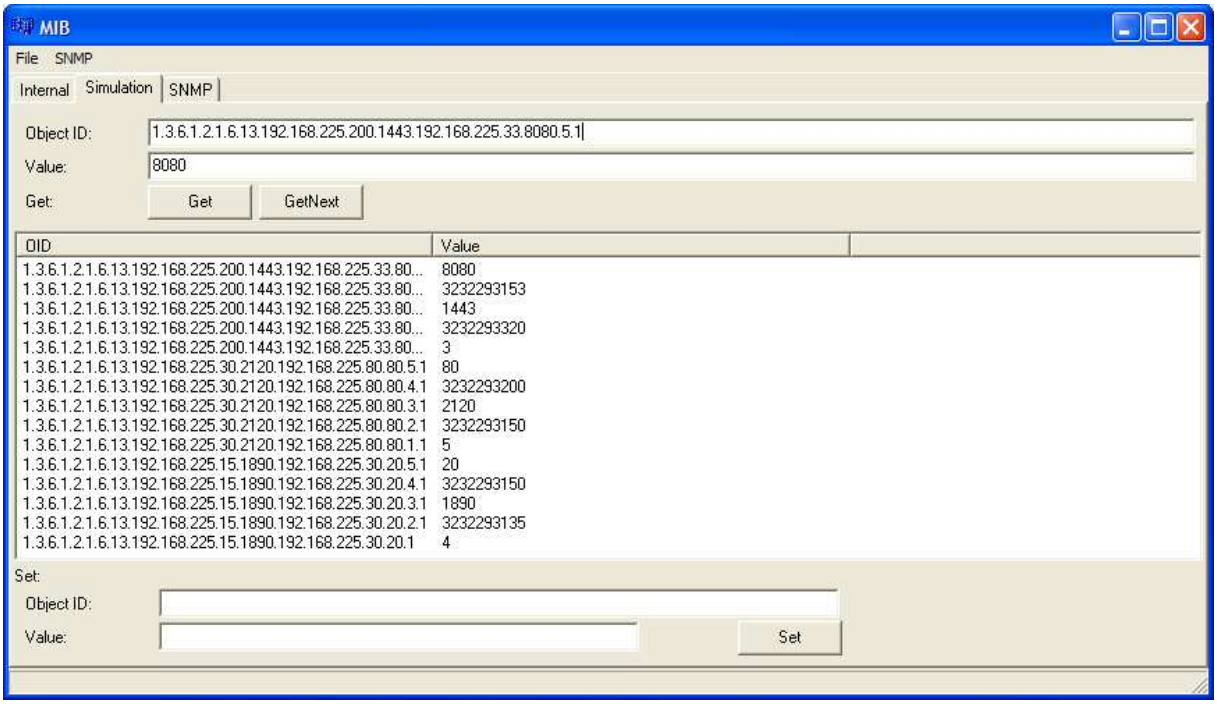


Obr 3.: Ukázková aplikace – záložka „Internal“

Vizuální prostředí programu je členěno do tří částí, což je reprezentováno třemi záložkami. První z nich, jež je se nazývá „Internal“ ukazuje rozhranní místního přístupu, kde můžeme procházet stromovou strukturu po jednotlivých větvích a úrovních. Ve výpisu se nám zobrazuje aktuální část cesty do uzlu, v němž se nacházíme a také obsah daného uzlu. Ten obsahuje informace o identifikátorech podřízených položek, jak textově, tak i číselně. Dále jsou zde informace o typu daných objektů a o hodnotách kterých nabývají. Při označení konkrétní položky z výpisu se zobrazí ve střední části obrazovky plný objektový identifikátor vybrané položky. Pokud by daná položka byla kořenovým uzlem tabulky, tak v pravé části programu dojde k zobrazení schématu tabulky, kde leze vyčíst pořadí sloupců a jejich název a též datové typy jednotlivých sloupců. Informace jež jsou uloženy v tabulkách, se zobrazí ve

spodní části. Pokud označíme konkrétní řádek tabulky, ve střední části programu se zobrazí plný objektový identifikátor vybraného řádku, který je vytvořen na základě informací o indexování tabulek a může být vytvořen i jako složený klíč nad několika sloupci tabulky.

Druhá záložka z názvem „Simulation“ slouží jako ukázka k simulování vzdáleného přístupu za použití objektových identifikátorů. Tato část programu umožňuje zjišťovat hodnoty jednotlivých položek, zjišťovat hodnoty následujících položek za využití adresy, již známé položky a následně také nastavování hodnot. Je to nazváno jako simulace, protože nedochází zde ke kódování či dekódování zpráv, k jakému by při reálném přístupu docházelo, ale přístup je skoro přímý, jen s tím rozdílem, že se používá objektového adresování, jež je zadáváno ve formě speciálně formátovaných řetězců. Pro přístup je třeba znát strukturu realizované databáze. Ve střední části programu se nachází jednoduchá historie naposledy zjištěných hodnot položek.

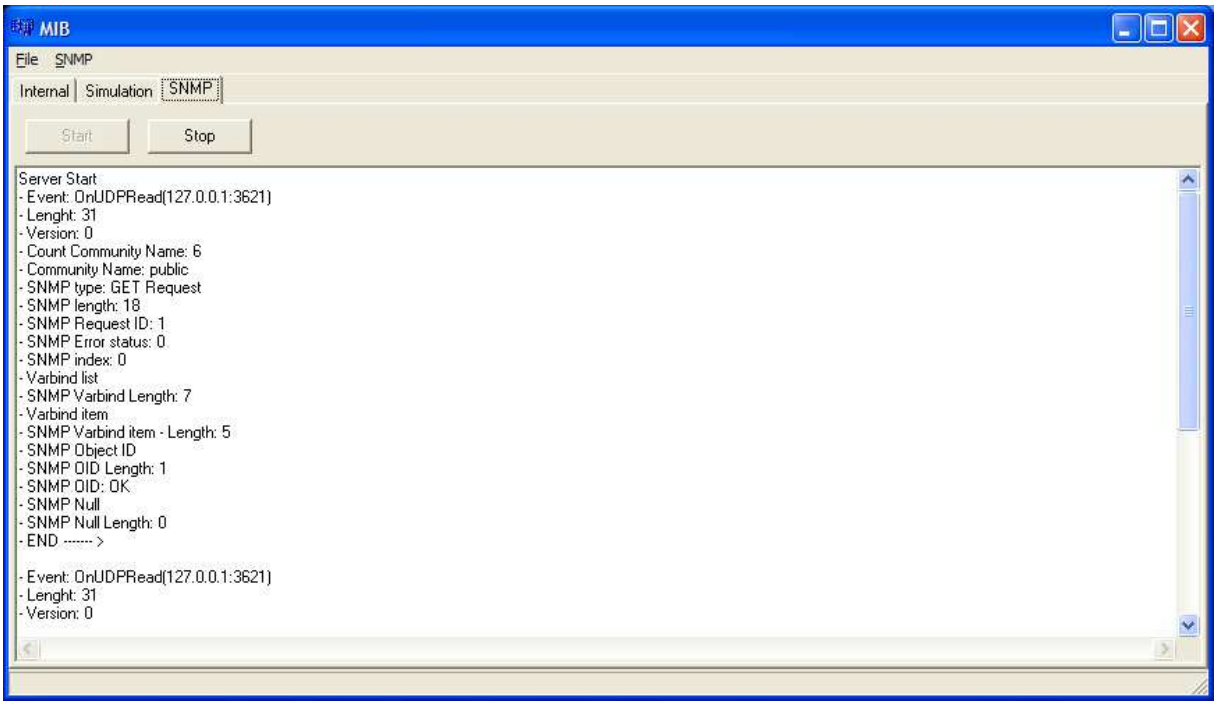


Obr 4.: Ukázková aplikace – záložka „Simulation“

Třetí a tudíž poslední záložka s názvem „SNMP“ obsahuje ovládací prvky určené k ovládání velmi jednoduchého agenta pro protokol SNMP. Zde je realizována i možnost vzdáleného přístupu do databáze MIB. Zde najdeme dvě tlačítka, která slouží ke spuštění a zastavení příjmu zpráv a dále komponentu s informacemi o naposledy přijatých zprávách a jejich rozkladu. Realizovaný agent SNMP byl vytvořen hlavně z důvodů lepšího přizpůsobení databáze MIB na požadavky protokolu SNMP. Díky tomu byl lépe vyladěn způsob adresace přes objektové identifikátory a byly přidány metody do rozhranní třídy TMibItem, jež nyní umožňují získávání dat z databáze již v kódování BER. Protože agent SNMP nebyl hlavním

cílem práce, tak jeho implementace je velmi jednoduchá a tudíž i nedokonalá. Díky tomu je s jeho provozem spojeno značné množství omezení.

Hlavní částí agenta je komponenta TIdUDPServer, což je komponenta zaobalující volání rozhraní API operačního systému rodiny Windows a slouží k příjmu a vysílání dat prostřednictvím protokolu UDP. V obslužné funkci události OnUDPRead, jež se volá při příchodu UDP datagramu je provedeno dekódování přijatých dat. Parser prozatím zvládá jen zprávy protokolu SNMP z verze 1 a to přesněji zprávu typu GetRequest. I zde jsou však omezení ještě v délce přijímaných dat. Po dekódování zprávy se provede vyhledání patřičného uzlu z databáze, jež je realizováno metodou GetVarOID() třídy TMibDatabase. Takto je získán uzel z hierarchické struktury, u kterého je provedena kontrola, zda-li se jedná o uzel z hodnotou. Pokud ano, je získaný uzel přetypován na objekt TMibItem a pomocí metod GetLengthBerVarbind() a GetBerVarbind() jsou vytvořeny data pro zprávu GetRespons, která bude odeslána zpět managerovi SNMP.



Obr 5.: Ukázková aplikace – záložka „SNMP“

Hlavní částí programu je instance třídy TMibDatabase, která je naplněna testovacími daty v inicializační proceduře hlavního formuláře. Kořenový uzel databáze je zpřístupněn pomocí metody GetRootNod(). Metoda vrací ukazatel na instanci třídy TMibNod, která je kořenovým uzlem celé vytvořené stromové struktury databáze MIB. Za použití metody GetSubNod() třídy TMibNod a jednoduchého pořadového indexu je možno databázi lehce procházet směrem do hloubky stromu. Pro získávání nadřazených uzlů se využívá metody GetParent() třídy TMibNod.

Pro získání detailních informací o uzlech a položkách stromu se využívají metody jako GetName() třídy TMibNod pro zjištění názvu uzlu a GetOID() pro zjištění části objektového identifikátoru, jež identifikuje objekt v dané větvi a na dané úrovni,, kde právě je. Ke zjištění typu objektu pro možnost přetypovat jej na rozhranní umožňující konkrétnější přístup slouží metody GetObjType() třídy TMibNod. Návratová hodnota má hodnoty odpovídajících konstantám (otNod, otItem, otTable, otTabRow). Zde již můžeme získat u položek nesoucích hodnoty databáze MIB přístup k rozhranní TMibItem, jež zaobaluje přístup k různým datovým typům prostřednictvím konverze hodnot do formy řetězců (metody GetValueToStr(), SetStrToValue()) a nebo za využití kódování BER (metody GetLengthBerValue(), GetLengthBerVarbind(), GetBerValue(), GetBerVarbind()). Ke zjištění typu hodnotových objektů pro možnost přetypovat je na jejich pravý datový typ, slouží metoda GetLinkType(). Navrácená hodnota nabývá hodnot konstant (typInt, typUnsInt, typChar, typOID). Po přetypování na pravý datový typ objektu je možno použít metody pro práci s konkrétními datovými typy. Např. metody třídy TMibIntItem jako GetValue() a SetValue() a nebo pro třídu TMibOIDItem jako GetValueOID() a SetValueOID(). Pro předávání objektových identifikátorů se využívá pomocná třída TIntList.

K vyhledávání objektů v databázi je ve třídě TMibDatabase určena metoda GetVarOID(), jež je pro snadnější použití ještě zjednodušena metodami GetStrOID() a GetBerOID(). Metoda GetVarOID využívá při své činnosti volání metody GetSubNodVarOID() třídy TMibNod, díky níž jsou vyhledávány na jednotlivých úrovních stromu uzly, které odpovídají cestě k hledanému objektu.

7. Závěr

Cílem práce je provedení implementace jednotlivých obslužných funkcí databáze MIB. Vzhledem k tomu, že databáze MIB slouží k uchování informací sloužících k řízení komunikačních sítí obsahuje práce nejprve lehký úvod do řízení sítí. Následně se práce zabývá seznámením se základy managementu síťových uzlů s využitím protokolu SNMP, na které úzce navazuje databáze MIB. V části zabývající se strukturou systémové databáze MIB a obslužnými funkcemi je nastíněna kořenová struktura databáze MIB pro protokol IP a základní datové typy, se kterými databáze pracuje. S části zabývající se protokolem SNMP vyplívají základní metody používané k vyčítání a změně dat v řídící databázi a na základě těchto poznatků je provedena analýza za účelem stanovení vhodné implementace obslužných funkcí nad databází MIB. Část práce zabývající se implementací databáze a jejich obslužných funkcí popisuje implementované řešení v jazyce C++ s využitím objektově orientovaného jazyka. Základem řešení je třída TMibDatabase, která uchovává celou strukturu databáze a definuje metody sloužící k práci s databází. Pro uchování samotné stromové struktury systémové databáze MIB bylo zvoleno řešení s navzájem na sebe odkazujícími objekty, jež jsou představovány instancemi třídy TMibNod. Jednotlivé skalární hodnoty jsou v databázi představovány třídami, které jsou odvozeny od třídy TMibItem. Tato třída je odvozena od třídy TMibNod, čímž je dána kompatibilita třídy TMibItem se strukturou stromu. Navržené řešení zatím implementuje tři základní datové typy pro uchování skalárních hodnot a těmi jsou datový typ integer a unsigned integer oba v 32-bitové formě. A jako třetí je objekt pro uskladnění řetězce znaků. Každý tento primitivní datový typ má vlastní třídu, která je s ním schopna pracovat. Těmito třídami jsou TMibIntItem, TMibUnsIntItem, TMibStrItem. Další třídou pro ukládání hodnot je třída TMibOIDItem, která uchovává jako hodnotu objektový identifikátor, jež ukazuje na jiný uzel databáze MIB. Díky tomu je možno realizovat relace uvnitř databáze MIB.

Při implementaci databáze MIB byly navrženy třídy TMibTable a TMibRow. Tyto třídy slouží k realizaci tabulek. Vzhledem k tomu, že tabulky používají jiné mechanizmy pro adresování než uzly se skalárními hodnotami, je v těchto třídách implementováno přizpůsobení procesních pochodů tabulek pro rozhranní třídy TMibNod, z kterého jsou obě třídy odvozeny. Pro zachování jednotnosti práce s daty bylo k realizaci tabulek použito pro uložení hodnot buněk opět tříd odvozených z třídy TMibItem. Důležitým faktem ale zůstává, že i tabulky jsou neustále i vnitřním uspořádáním stromového charakteru.

Poslední část práce popisuje program, který ukazuje, jak by měly být vytvořené třídy používány. V programu je realizován jak interní přístup k databázi, tak i simulace vzdáleného přístupu skrze objektové identifikátory. Poslední část programu ukazuje realizaci velmi jednoduchého agenta SNMP, ve kterém je představeno používání rozhranní, jež jsou určeny pro spolupráci s protokolem SNMP a používají kódování BER.

8. Literatura

[1] MOLNÁR, K. Uživatelem ovladatelný mechanismus diferencovaného zajištění kvality služeb, Habilitační práce, VUT v Brně, 2007

[2] KLAŠKA, L. Správa počítačových sítí, Tutorial, [on-line], 2000, Dostupné na WWW: <http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&temaID=23&clanekID=24>

[3] ODVÁRKA, P. Struktura MIB a přístup k jejím informacím, Tutorial, [on-line], 2001, Dostupné na WWW: <http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&temaID=129&clanekID=133>

[4] HERMAN, I. Komunikační technologie, Skripta, VUT v Brně, 2007

[5] ŠELIGA, M. Implementace databáze MIB v simulačním prostředí OPNET Modeler, Bakalářská práce, VUT v Brně, 2007

[6] CASE, J., McCLOGHRIE, K., ROSE, M., WALDBUSSER, S. Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2), RFC 1442, Internet Engineering Task Force, 1993

[7] CASE, J., McCLOGHRIE, K., ROSE, M., WALDBUSSER, S. Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2), RFC 1450, Internet Engineering Task Force,1993

[8] Carpenter, G., Wijnen, B. Simple Network Management Protocol Distributed Program Interface, RFC 1228, 1991

[9] K. McCloghrie, Rose, M. Management Information Base for Network Management of TCP/IP-based internets: MIB-II , RFC 1213, 1991