České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídicí techniky

Diplomová práce

Podpůrné nástroje pro RTPS komunikaci

Praha, 2005 Lukáš Pokorný

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne: ....

Podpis

Poděkování

Tímto bych chtěl poděkovat všem lidem, kteří mi pomohli při vytváření této práce. Velký dík patří mému vedoucímu diplomové práce Ing. Petru Smolíkovi za trpělivost a podporu během řešení všech problémů z oboru Real-Time komunikací a programování v jazyce C. Další velký dík patří Ing. Františku Vackovi za pomoc při vytváření Java rozhraní.

Největší dík patří mé rodině, hlavně mým rodičům, kteří mě po celou dobu studií podporovali a měli se mnou neskutečnou trpělivost. Děkuji mami, děkuji tati. Dík také patří milým lidičkám z pekárny ve Václavské pasáži. Tuto práci bych chtěl věnovat svým rodičům, bratrům, babičce, dědovi a Káje.

Abstrakt

RTPS protokol slouží k Real-Time komunikaci po Ethernetu. Projekt ORTE je otevřená implementace RTPS protokolu. Tato práce se zabývá vytvořením podpůrných nástrojů RTPS protokolu pro projekt ORTE. Prvním řešeným úkolem bylo vytvoření a začlenění podpory RTPS prototokolu do programu Ethereal. Druhý úkol spočíval ve vytvoření API rozhranní v jazyce Java pro knihovnu liborte z projektu ORTE. Při řešení tohoto úkolu bylo použito JNI rozhraní umožňjící zdrojovému kódu Javy kooperaci s programovacím jazykem C. Poslední úkol spočíval v otestování nově vytvořeného Java API rozhraní vytvořením aplikace pro monitorování síťových aplikací využívajících RTPS protokolu.

Abstract

RTPS communication protocol is used to have Real-Time communication over Ethernet. Project ORTE is a free open-source implementation of RTPS protocol. This work is all about developing support tools for RTPS protocol covered by ORTE project. The first goal was create the support of RTPS protocol to program Ethereal. Second goal was create Java API to liborte library from project ORTE. JNI was use to solve the problem how can Java cooperates with language C. Third and the last goal was to use new Java API and create an aplication to monitor other network applications, which use RTPS protocol.

Obsah

Seznam obrázků **4**

Seznam tabulek **5**

Úvod **6**

1 Síťová kom unikace **7**

1.1 Komunikace 7

1.2 Sítě 7

1.2.1 Síťové topologie 8

1.2.2 Modely distribuce dat 8

1.2.3 ISO/OSI model 9

1.2.4 Zapouzdření dat 9

1.2.5 Protokoly 10

1.2.6 Způsoby adresace 11

2 Ethernet **13**

2.1 Technologie Ethernet 13

2.2 Historie Ethernetu 13

2.3 Důležité vlastnosti 14

2.4 Ethernet a ISO/OSI model 14

2.5 Typy Ethernetu 15

3 Real-Time Ethernet **16**

3.1 Definice Real-Time systému 17

3.2 Real-Time Ethernet 17

3.3 Architektury Real-Time Ethernetu 18

3.4 RTPS protokol 18

3.4.1 Publish-Subscribe protokolem 19

3.4.2 Composite State protokol 20

4 Programovací jazyk C, Java a JNI rozhraní **23**

4.1 Jazyk C 23

4.2 Jazyk Java 24

4.3 Java Native Interface 25

4.3.1 Zpracování programu 26

4.3.2 Praktický příklad vytvoření programu pomocí JNI 26

5 Projekt ORTE **34**
5.1 Představení projektu ORTE 34
5.2 Knihovna liborte 34
5.3 API rozhraní 35
5.4 Režimy komunikace 35
5.5 Serializace a deserializace dat 35
5.6 Příklad užití ORTE API 36
6 Popis řešení úloh **37**
6.1 Hlavní úkoly a postup řešení 37
6.2 Podobné projekty 38
7 Java rozhraní pro projekt ORTE **39**
7.1 Základní koncepce 39
7.2 Balíček org.ocera.orte 39
7.2.1 Návrh systému balíků 39
7.2.2 Návrh tříd objektů 40
7.2.3 Návrh metod Java API 43
7.2.4 Překlad balíků 44
7.3 Knihovna libjorte 44
7.3.1 Návrh nativních funkcí 44
7.3.2 Překlad knihovny 45
7.4 Příklad použití ORTE Java API 45
8 RtpSeYe **46**
8.1 Základní požadavky 46
8.2 Návrh aplikace 46
8.2.1 Návrh GUI 46
8.2.2 Návrh datových modelů 47
8.3 Balíkorg.ocera.orte.rtpseye 49
8.4 Režimy aplikace RTPSeYe 50
8.5 Testování funkčnosti 50
9 Ethereal **52**
9.1 O programu 52
9.2 Zpracování zachycených paketů 53

9.3 Tvorba podpory RTPS protokolu 54
9.3.1 Soubor packet-rtps.c 54
9.3.2 Soubor packet-rtps.h 55
9.4 Začlenění podpory RTPS do Etherealu 55
Závěr **56**
Seznam použitých zdrojů **57**
A Obrázková příloha **59**
B Použité zkratky **62**
C Adresářová s truktura přiloženého CD **64**

Seznam obrázků

Obr. 1 Typy síťových topologií
Obr. 2 Modely disribuce dat
Obr. 3 Vrstvové modely
Obr. 4 Proces zapouzdřování dat
Obr. 5 Podniková pyramida a užití Ethernetu
Obr. 6 Ethernet podle ISO/OSI modelu
Obr. 7 IEEE 802.3 frame
Obr. 8 Typy real-time systémů
Obr. 9 Typy Real-Time Ethernetu
Obr. 10 Grafické znázornění parametrů persistence a strength
Obr. 11 Grafické znázornění parametrů minimum separation a deadline
Obr. 12 Objekty Manager a ManagedApplication
Obr. 13 Zpracování programu v jazyce C
Obr. 14 Zpracování programu v jazyce Java
Obr. 15 Vytváření programu pomocí Javy, JNI a jazyku C
Obr. 16 Zdrojový kód jazyka Java (ImaginaryObject.java)
Obr. 17 Zdrojový kód vygenerovaného hlavičkového souboru (my_jni_test_ImaginaryObject.h)
Obr. 18 Zdrojový kód s implementací těla nativní funkce (printWord.c)
Obr. 19 Zdrojový kód s implementací těla nativní funkce (printNumber.c)
Obr. 20 Zdrojový kód s implementací těla nativní funkce (setNumber.c)
Obr. 21 Příklad užití API rozhraní knihovny liborte
Obr. 22 Struktura balíků ORTE Java API
Obr. 23 Propojení metody Java API s původní liborte funkcí
Obr. 24 Příklad užití Java API rozhraní
Obr. 25 Návrh GUI applikace RTPSeYe
Obr. 26 Struktura třídy NodeInfo
Obr. 27 Datový model pro uložení objektů třídy NodeInfo
Obr. 28 Datové modely grafických komponent (JTree a JTable)
Obr. 28 Zpracování dat v Etherealu
Obr. 30 Hierarchie Java 2 Platform, Standart Edition v 1.4
Obr. 31 Registrace RTPSeYe k RTPS Managerovi
Obr. 32 Selhání registrace RTPSeYe k RTPS Managerovi
Obr. 33 Pracovní režim RTPSeYe
Obr. 34 Ethereal zachytávající RTPS komunikaci

Seznam tabulek

Tab. 1 Třídy IP adres
Tab. 2 Rezervované IP adresy
Tab. 3 Deadline průmyslových zařízení
Tab. 4 Parametry Publishera
Tab. 5 Parametry Subscribera
Tab. 6 Primitivní datové typy Javy a odpovídající JNI typy
Tab. 7 Referenční datové typy Javy a odpovídající JNI typy
Tab. 8 Pomocné datové typy JNI
Tab. 9 Porovnání NDDS a ORTE

Úvod

Evropský projekt OCERA (*Open Components for Real-Time Applications*) si klade za cíl vytvořit integrované prostředí pro embedded real-time aplikace. Samotný projekt je velice rozsáhlý a byl proto rozčleněn na menší projekty. Jednotlivé projekty si rozebraly evropské technické univerzity, které se přihlásily ke spolupráci na hlavním projektu.

České vysoké učení technické v Praze (ČVUT) v projektu OCERA zastupuje katedra řídicí techniky. Oblast hlavního úkolu, který katedra obdržela k vyřešení lze souhrně nazvat *komunikace*. Oblast komunikace byla opět rozdělena na menší části zabývající se technologií CAN, OpenCAN a Real-Time Ethernet. Projekt zabývající se Real-Time Ethernetem dostal název ORTE *(OCERA Real-Time Ethernet).*

Výsledkem řešení projektu ORTE je implementace RTPS (*Real-Time Publish-Subscribe*) komunikačního protokolu. Komunikační protokol je postaven na distribučním modelu Publish-Subscribe, který definuje dva druhy komunikačních uzlů (nodů). Producenta dat - Publishera a odběratele dat – Subscribera. RTPS protokol používá klasický komunikační model založený na transportních protokolech IP/UDP. Nad těmito protokoly je vytvořena softwarová mezivrstva, která se stará o řízení datového toku. Mezivrstva poskytuje rozhraní pro výše položené uživatelské aplikace. Mezi výhody zvolené architektury patří fakt, že není třeba provádět změny stávajícího komunikačního modelu (IP/UDP). Nevýhoda zvolené architektury spočívá v přenosu real-time dat spolu s klasickými daty a problém jednotného času (synchronizace). Právě to řadí RTPS protokol do kategorie tzv. soft-real-time komunikace.

Cílem této práce je vytvořit v rámci projektu ORTE podpůrné nástroje pro RTPS komunikaci. Hlavním úkolem je navrhout a realizovat aplikační rozhraní (API) pro knihovnu projektu ORTE. Knihovna poskytuje API rozhraní v jazyce C, úkolem je doplnit knihovnu o API rozhraní v jazyce Java. Na základě nového rozhraní, pak navrhnout a realizovat aplikaci s grafickým rozhraním sloužící pro monitorování síťových aplikací, které využívají RTPS komunikačního protokolu. Poslední úkol spočívá ve vytvoření podpory RTPS protokolu pro monitorovací program Ethereal.

Tento dokument popisuje způsoby řešení jednotlivých úkolů. První polovina dokumentu má ryze teoretický charakter, slouží ke shrnutí základních poznatků a představení technologií potřebných pro řešení zadaných úkolů. Druhá polovina slouží k popisu návrhu řešení a vlastní realizaci zadaných úkolů.

Kapitola 1

Síťová komunikace

Kapitola věnovaná popisu síťové komunikace. V podkapitolách jsou vysvětleny základní principy komunikace a základní terminologie.

1.1 Komunikace

Pod pojmem komunikace si můžeme obecně představit přenos informace, sdělování myšlenek a poznatků. Lidé mezi sebou komunikují již od pradávna. Ať už se jednalo o komunikaci neverbální (bubny, posunky, obrázky, písmo atd.) či verbální (řeč), vždy bylo cílem něco sdělit druhé straně. K tomu, aby se mohli dva a více jedinců mezi sebou domluvit a předat si tak informace, musí obě zúčastněné strany používat stejný jazyk. Jazyk je soubor výrazů (slov) a pravidel, jak s výrazy nakládat.

Díky rychlému rozvoji techniky v posledních dvou stoletích došlo k vytvoření zařízení, která mohou komunikovat mezi sebou a umožňovat tak lidem např. komunikovat na velké vzdálenosti (např. e-mail), řídit technologické procesy (např. HMI). Aby si dvě zařízení rozuměla, musí opět používat stejný jazyk. V síťové terminologii nazýváme tento jazyk komunikačním protokolem. Síťové zařízení budeme v dalším textu nazývat *nodem*.

1.2 Sítě

Aby mohlo docházet mezi zařízeními k přenosu informací, potřebujeme zařízení nějakým způsobem propojit a vytvořit tak komunikační kanál. V praxi se využívá přenosového média, po kterém jsou posílána kvanta informací ve formě signálů. Typ přenosového média je vybírán podle různých kriterií např. podle délky komunikačního kanálu, požadované rychlosti komunikace, ceny přenosového média a k němu náležejícího vybavení, vlivů vnějšího okolí, možnosti budoucího rozšíření a dalších.

V současné době používáme 3 typy přenosových médií:

- metalické kabely,
- optické kabely,
- elektromagnetické vlny.

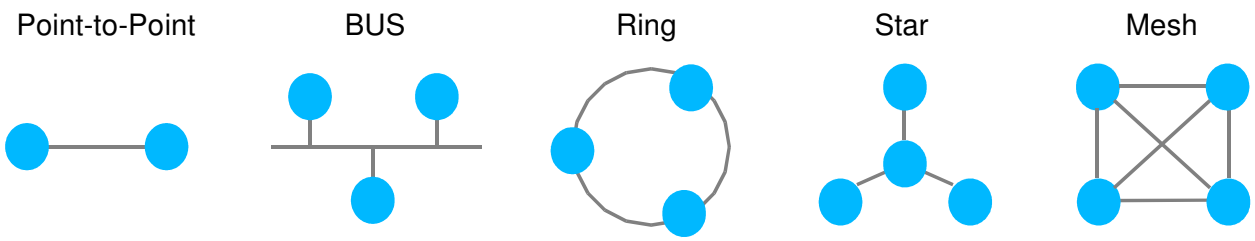
Nejběžnějším přenosovým médiem jsou *metalické kabely*. Používáme je na krátké vzdálenosti a

pro přenosové rychlosti do 1 Gb/s. Výhodou metalických kabelů je hlavně nízká cena ve srovnání s ostatními přenosovými médii. Tyto kabely jsou však náchylné na různé formy elektromagnetického rušení a lze je použít na krátké vzdálenosti (maximálně stovky metrů). Použitím *optických kabelů* získáme komunikační kanál na dlouhé vzdálenosti (řádově km) s možností vysokých přenosových rychlostí (10 Gb/s), který není náchylný na žádnou formu elektromagnetického rušení. Elektromagnetické vlny používáme pro bezdrátové přenosy na krátké i dlouhé vzdálenosti s různými přenosovými rychlostmi.

1.2.1 Síťové topologie

Topologie sítě popisuje strukturu sítě neboli způsob propojení jednotlivých nodů.

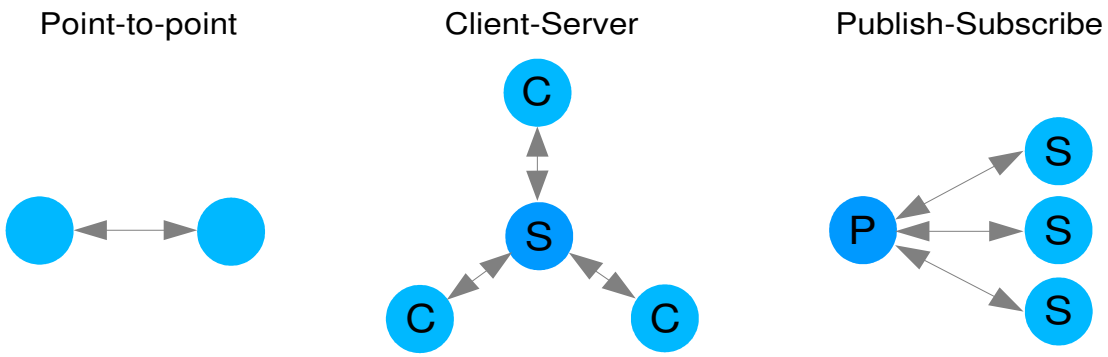
1. ***Point-to-point (přímé spojení).*** Nody jsou přímo propojeny přenosovým médiem.
2. ***Bus (sběrnice).*** Jednotlivé komunikační nody jsou připojeny ke společnému přenosovému médiu. Každý nod, který chce komunikovat s dalším, musí nejprve získat právo přístupu na sběrnici.
3. ***Ring (kruh).*** Všechny nody jsou zapojeny do kruhu. Po síti je posílána dokola zpráva tzv. token. nod který obdrží token má právo přístupu na sběrnici.
4. ***Star (hvězda)***. Všechny nody jsou připojeny k jednomu centrálnímu nodu. V případě selhání centrálního nodu je nefunkční celá síť. Lze vytvořit i rozšíření tzv. ***Extended star*** – nody připojené k hlavnímu centrálnímu nodu tvoří centrální body pro další nody.
5. ***Mesh.*** Každý nod je propojen s každým nodem. Jedná o topologii s maximální propojeností nodsů. Jsou zde záměrně vytvářeny redundantní spojení za účelem dosažení co největší spolehlivosti sítě (příklad užití – atomová elektrárna).



Obr .1 Typy síťových topologií

1.2.2 Modely distribuce dat

- ***Point To Point.*** Model zajišťující přímé připojení účastníků. Není vhodný pro více nodů.
- ***Client – Server.*** Model s centralizovaným prvkem – *serverem*, k němuž jsou přímo připojeni *klienti*. Praktické pro centralizaci dat, nevýhodné pro komunikaci klient-klient. V případě selhání centrálního prvku dojde ke kolapsu veškeré komunikace.
- ***Publish – Subscribe.*** Specifikace typu nodů na *producenty* dat (*Publisher*) a *konzumenty* dat (*Subscriber*). Producent může dodávat data více konzumentům. Může existovat záložní producent, který v případě selhání prvního producenta převezme jeho funkci. Systém je snadno rekonfigurovatelný. V případě odebrání nodu nehrozí zhroucení komunikace. Nový nod může být kdykoliv přidán. Jedná se o typ komunikace *one-to-many* (1-pro více).

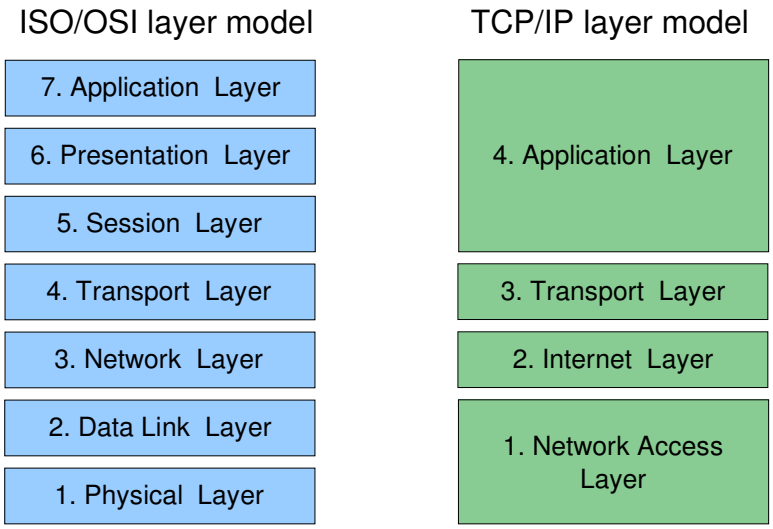


Obr. 2 Modely disribuce dat

1.2.3 ISO/OSI model

ISO/OSI referenční vrstvový model byl vyvinut z potřeby sjednocení skupiny síťových standardů pro výrobce síťových technologií, což vedlo ke zvýšení kompatibility síťových technologií. Model se stal populární pro znázorňování datové komunikace po síti a předlohou pro další vrstvové modely např. TCP/IP model.

Referenční ISO/OSI model se skládá ze 7 vrstev, TCP/IP ze 4 vrstev:



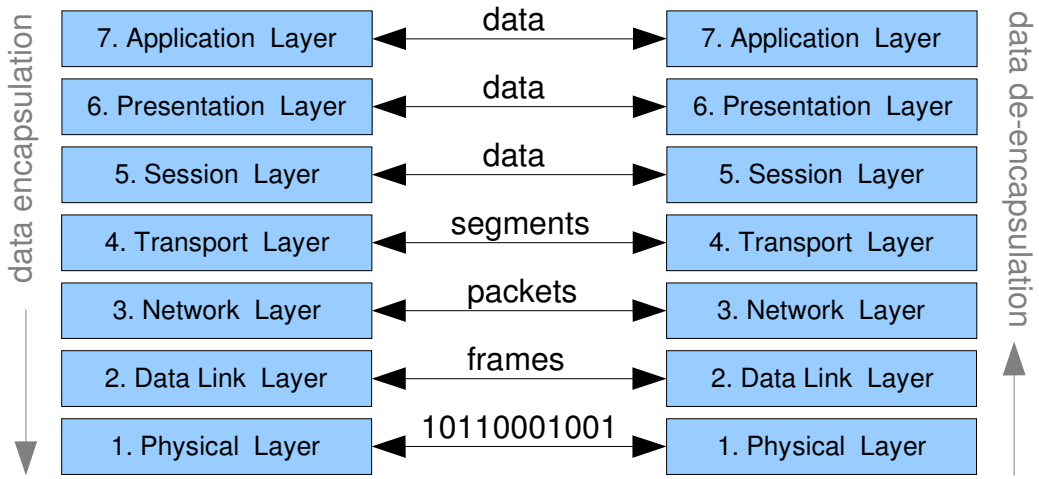
Obr. 3 Vrstvové modely

Každá vrstva řeší vymezenou množinu problémů, nakládá určitým způsobem s původními poslanými daty a používá vlastní způsob adresace svých zpráv. Jednotlivé vrstvy mezi sebou komunikují a předávají si data. Spodní vrstvy poskytují služby vyšším vrstvám. Komunikační protokoly zajišťují logické spojení stejných vrstev účastníků komunikace.

1.2.4 Zapouzdření dat

Zapouzdření dat (*data encapsulation*) lze dobře popsat na ISO/OSI modelu. K zapouzdření dat

dochází na straně odesílatele. Princip zapouzdření spočívá v postupném předávání dat z vyšších vrstev všem vrstvám nižším, přičemž 1. fyzická vrstva data odešle. Každá vrstva provede se získanými daty určitou operaci, většinou rozdělí data na menší části, k nimž přidá hlavičku svého přenosového protokolu. Pak pošle data vrstvě pod sebou. Datové jednotky různých vrstev mají své vlastní názvy (framy, pakety, segmenty). Cílová statnice po přijetí dat provádí opačný proces (*de-encapsulation*).



Obr. 4 Proces zapouzdřování dat

1.2.5 Protokoly

Komunikační protokol je soubor pravidel pro přenos dat. Pravidla určují, jakým způsobem navázat kounikaci, jak vyrobit zprávu, jak do zprávy vložit data a následně ji odeslat, co dělat při doručení, nedoručení a poškození zprávy a jiné.

Před vlasním přenosem zprávy je třeba navázat spojení mezi účastníky. Dělení protokolů podle typu navazování spojení:

- ***Connection-less.*** Příjemce není předem upozorněn, že zanedlouho obdrží data. V běžném životě můžeme tento způsob přirovnat k doručování dopisů. Poštovní doručovatelka vhodí dopis do schránky, bez předchozího upozornění adresáta. Příkladem takového protokolu je *IP (Internet Protocol)*.
- ***Connection-oriented.*** Příjemce je předem upozorněn, že zanedlouho obdrží data. Před přenosem je navazováno spojení. V běžném životě můžeme tento způsob přirovnat k telefonování. Telefon příjemce předem upozorní zvoněním na příchozí hovor. Příkladem takového protokolu je *TCP (Transfer Control Protocol)*.

Podle způsobu doručování dat můžeme protokoly dělit do dalších kategorií:

- ***Reliable.*** Data jsou vždy doručena. Protokol vlastní mechanismy na kontrolu správného doručení dat. V případě nedoručení dat jsou data opětovně přeposlána. Kontrola doručování je na úkor rychlosti doručování. Příkladem takového protokolu je TCP.
- ***Unreliable.*** Doručení dat není kontrolováno. Protokol má vyšší doručovací rychlost. Příkladem takového protokolu je *UDP (User Datagram Protocol)*.

Důležité protokoly:

- ***Protokoly 2. vrstvy.*** Mezi tyto protokoly se řadí tzv. *routing protokoly*, které slouží k vyhledávání cest mezi jednotlivými zařízeními připojenými do sítě. Např. protokoly *RIP*, *IGRP*, *RIPv2* a další. Pro svou adresaci používají *MAC adresu*.
- ***IP (Internet Protocol).*** Connection-less unreliable protokol 3. síťové vrstvy. Slouží ke směrování datových jednotek – paketů. Z množiny možných cest odesílatel-příjemce se snaží vybrat tu nejlepší a po ní směruje pakety. Pro svou adresaci používá tzv. IP adresu.
- ***UDP (User Datagram Protocol).*** Connection-oriented, unreliable protokol 4. síťové vrstvy. Slouží pro posílání dat vyšších vrstev bez kontroly doručování zpráv. Protokoly vyšší vrstvy využívající služeb UDP jsou nap
- ř. *TFTP*, *DNS*, *DHCP*, *RTPS* a další.
- ***TCP (Transfer Control Protocol).*** Connection-oriented, reliable protokol 4. síťové vrstvy. Slouží pro posílání dat vyšších vrstev s kontrolou doručování zpráv. Protokoly vyšší vrstvy využívající služeb TCP/IP jsou např. *HTTP*, *FTP*, *Telnet* a další.

1.2.6 Způsoby adresace

Základní způsoby adresace jsou:

- ***Unicast.*** Zpráva je adresována právě jednomu příjemci.
- ***Multicast.*** Zpráva je adresována více příjemcům.
- ***Broadcast.*** Zpráva je adresována všem příjemcům připojeným k síti.

Typy adres:

- ***MAC Adresa.*** Jedinečná 6 bytová adresa vypálená do paměti ROM síťové karty. První tři byty identifikují výrobce karty a zbylé tři byty představují sériové číslo karty. Adresa je používána k adresaci v protokolech nižších vrstev.
- ***IP Adresa.*** Adresa tvořená 4 byty (32bit), skložená ze dvou částí:
  - *síťová část* - identifikuje síť,
  - *systémová část* – identifikuje hosta v síti.

Podle toho, kolik bitů přísluší síťové části, dělíme IP adresy do následujících tříd:

třída	*rozsah*	*byty síťové části*	*užití*
A	0 - 127	1	velké sítě
B	128 - 191	2	středně velké sítě
C	192 - 223	3	malé sítě
D	224 - 239	1	multicasting

Tab. 1 Třídy IP adres

IP adresy s pevně definovaným významem:

adresa	***význam***
127.0.0.1	localhost – adresa vlastní NIC
192.168.1.10/24	běžná adresa zařízení
192.168.1.**0**	adresa sítě
192.168.1.**255**	broadcast adresa

Tab. 2 Rezervované IP adresy

Díky velkému rozvoji síťových technologií a neustálému narůstání počtu připojených zařízení k síti, dochází k vyčerpávání stávajících 4 bytových IP adres. Jedno z řešení tohoto problému je zavedení nového typu adresy. *IPv6 adresa* je složená ze 16 bytů (128 bitů), které představují celkem $3{,}4 \cdot 10^{38}$ adres.

Kapitola 2

Ethernet

V kapitole jsou shrnuty základní poznatky týkající se technologie Ethernet. Je popsán historický vývoj i současné trendy a vysvětlen princip fungování Ethernetu.

2.1 Technologie Ethernet

Ethernet je jedna z nejrozšířenějších technologií určená pro budování LAN sítí. V současné době pojem Ethernet představuje standart IEEE 802.3.

Základní myšlenka Ethernetu spočívá v možnosti komunikace zařízení po sdíleném médiu, aniž by se signály jednotlivých zařízení rušily. Tento přístup se v počátcích stal základem pro technologii *CSMA/CD*. CSMA/CD technologie zajišťující přístup k médiu a řešení kolizí byla potřebná hlavně v začátcích Ethernetu, kdy přenosové médium tvořil koaxiální kabel a režim vysílání byl *half duplex* (stanice mohla buď vysílat nebo přijímat, nikdy současně). Postupným vývojem a nástupem nových přenosových médií přibyly *UTP* kabely a optické kabely. Nová přenosová média umožnily Ethernetu pracovat v režimu *full duplex* (stanice vysílá a přijímá současně). Použití full duplex režimu vede k odstranění problému kolizí.

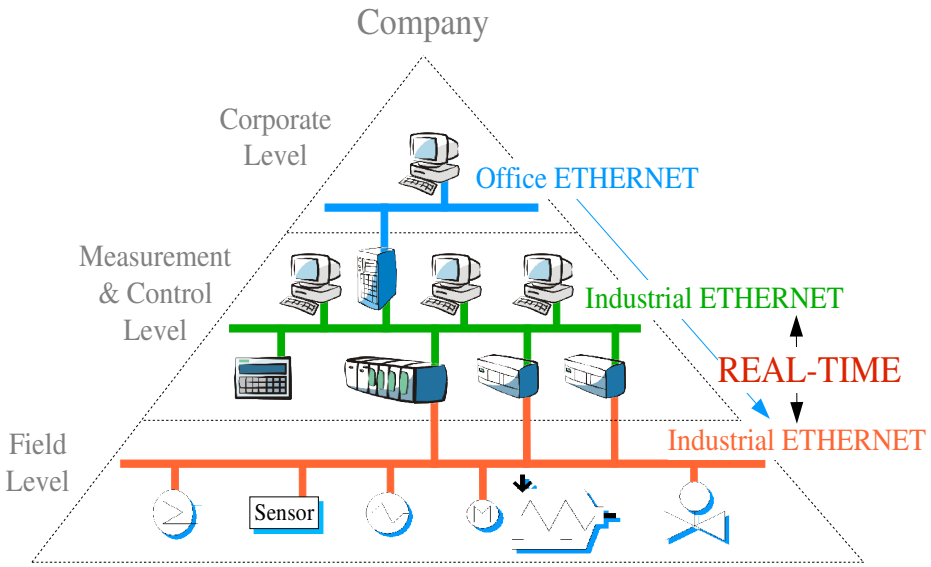
2.2 Historie Ethernetu

Historie této technologie sahá do roku 1972, kdy Robert Metcalfe a jeho spolupracovníci ze společnosti Xerox Corporation navrhli technologii zvanou *Alto Aloha Network*. O rok později byla technologie přejmenována na Ethernet. V roce 1976 byla technologie Ethernet rozšířena o tzv. *carrier sense metodu* a byla vybudována první síť k propojení počítačů. V roce 1980 byl vydán první standard pro 10Mbit Ethernet tzv. *Blue Book Standard (DIX Ethernet)* konsorcia DIX (Digital, Intel, Xerox).

V roce 1985 dochází k dalšímu rozšíření Ethernetu a zařazení pod nový standard IEEE 802.3. V témže roce se Ethernet poprvé dostává do průmyslu pod jménem *SINEC H1*, slouží pro komunikaci mezi PLC fy. Siemens. Největší rozšíření Ethernetu je však v kancelářské sféře. Ethernet se drží na hranici 10 Mbit/s.

V roce 1995 dochází k vylepšení Ethernetu a rychlost je zvýšena na 100Mbit/s. Tento typ Ethernetu se začal nazývat *Fast Ethernet*. V témže roce se jako přenosové médium začíná používat UPT kabel, který umožňuje full duplex režim. V roce 1998 je rychlost opět zvýšena na 1Gbit/s a Ethernet dostal název *Gigabit Ethernet*. Od roku 2000 začíná prudký nárůst průmyslového Ethernetu s označním *Idustrial Ethernet*.

V současné době, rok 2005, probíhá vytváření standardů pro 10Gbit Ethernet. Postupné zvyšování rychlostí vede k novým možnostem použití Ethernetu např. pro přenos videa. Pronikání Ethernetu do oblasti průmyslu přináší myšlenku sjednocení kancelářských sítí (*Administrative/ Office Ethernet*) a průmyslových sítí (*Idustrial Ethernet*). Výsledkem procesu je jednotná síťová technologie viz. Obr.5. V průmyslové sféře rostou požadavky na *Real-Time Ethernet*, který by nahradil sběrnice typu *fieldbus*.



Obr. 5 Podniková pyramida a užití Ethernetu

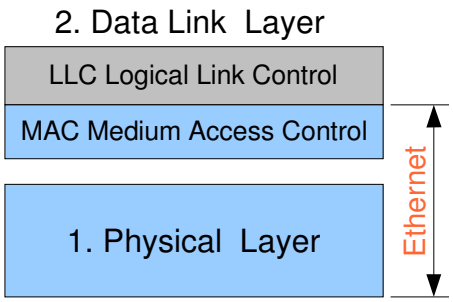
2.2 Důležité vlastnosti Ethernetu

Ethernet za svůj prudký rozvoj vděčí svým nadčasovým vlastnostem:

- **Flexibilní technologie**, možnost spolupracovat s novými techologiemi při zachování zpětné vazby na staré technologie, díky stejnému *frame formátu* (2.3).
- **Snadná implementace a instalace.**
- **Spolehlivost.**
- **Cenová dostupnost**, cena je odvislá od typu použitého přenosového média.

2.3 Ethernet a ISO/OSI model

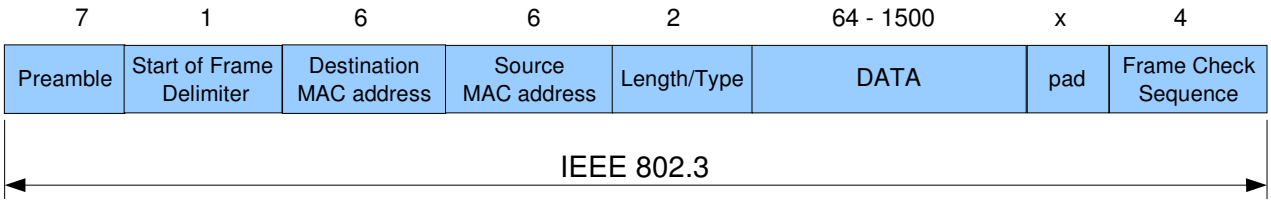
Ethernet pracuje na prvních dvou vrstvách modelu, respektive na první a polovině druhé vrstvy.



Obr. 6 Ethernet podle ISO/OSI modelu

- ***Medium Acces Control (MAC).*** LLC (Logical Link Control) získá data z vyšších vrstev v podobě paketů. Tato data jsou předána MAC ke zpracování. MAC zapouzdří data do tzv. framu (Obr.7). Zapouzdření se provede následovně:
  - *Preamble*. Zajišťuje synchronizaci pro 10Mbit Ethernet a méně.
  - *Start of Frame Delimiter.*Ukončuje synchronizační údaje. Obsahuje vždy posloupnost 10101011.
  - *Destination Address*. MAC adresa příjemce framu.
  - *Source Address*. MAC adresa odesílatele framu.
  - *Length/Type.* Pokud je hodnota menší než 1536, jedná se o délku dat framu. Při hodnotě větší se jedná o identifikaci protokolu vyšší vrstvy, kterému náleží data ve framu.
  - *Data.* Přenášená data paketu 3. vrstvy. Minimální délka dat 64, maximální 1500 bytů.
  - *Pad*. "Prázdná data". Užívají se k doplnění chybějících bytů na požadovanou nejmenší délku dat framu.
  - *Frame Check Sequence (FCS)*. Kontrolní součet. Pokud neodpovídá hodnota FCS nově spočtenému součtu, je frame zničen.

- ***Physical Layer.*** Převádí frame na signál, který odešle po přenosovém médiu. Fyzická vrstva zahrnuje různé typy přenosových médií, konektory, úrovně signálů a další.



Obr. 7 IEEE 802.3 frame

Pozn. Frame formát Ethernetu a IEEE 802.3 se drobně liší užitím políček *Preamble* a *Length/Type*.

2.4 Typy Ethernetu

Jednotlivé typy Ethernetu se od sebe liší použitým přenosovým médiem, přístupem na médium a rychlostí přenosu. Díky jednotnému frame formátu, uvedenému výše, je zachována propojitelnost všech typů Ethernetu, které jsou:

- ***Legacy Ethernet.*** 10 Mbit Ethernet [1] (10Base5, 10Base2, 10Base-T).
- ***Fast Ethernet.*** 100 Mbit Ethernet [1] (100Base-TX, 100Base-FX).
- ***Gigabit Ethernet.*** 1000 Mbit Ethernet [1] (1000Base-T, 1000Base-SX, 1000Base-LX).
- ***10Gigabit Ethernet.*** 10 Gbit Ethernet [1].

Kapitola 3

Real-Time Ethernet

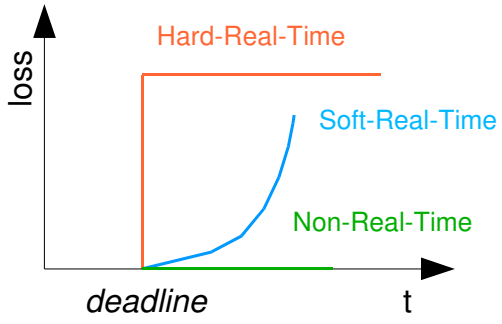
Ethernet v kancelářské sféře (Administrative/Office Ethernet) v současné době splňuje všechny základní požadavky na něj kladené. Postupné pronikání Ethernetu do oblasti průmyslu přineslo požadavek komunikace v reálném čase. Následující kapitola vysvětlí pojem reálného času a požadavky kladné na real-time komunikaci. Konec kapitoly je věnován RTPS protokolu.

3.1 Definice Real-Time systémů

Definice real-time systému: *"Systém reálného času je takový systém, který je schopen reagovat na vstupní události do předem stanoveného pevného časového okamžiku"*, Prof. Zezulka [2]. Podle definice záleží na výběru pevného časového okamžitu, nazývaného také *deadline* nebo *latency*.

Podle rychlosti reakce dělíme systémy do kategorií:

- ***Non-Real-Time systém.*** Nedodržení deadline nevadí, nejsou ohroženy lidské životy ani řízený proces.
- ***Soft-Real-Time systém.*** Toleruje se občasné nedodržení deadline. Nedodržením deadline nejsou ohroženy lidské životy ani řízený proces. S časem rostoucím od deadline narůstají ztráty. Příkladem ze života je např. člověk, který čeká na autobus. Odjezd autobusu je deadline. Občas se stává, že autobus nedodrží svůj deadline, což nikterak neohrožuje cestující na životech, ale zvyšuje jim tím časovou ztrátu.
- ***Hard-Real-Time systém.*** Systémy ve kterých na dodržení deadline závisí lidské životy i řízená technologie. Často se jedná o velice malý časový interval deadline. Příkladem ze života můžou být brzdy autobusu. V případě selhání brzd, tj. nedodržení deadline reakce brzd, jsou ohroženy lidské životy cestujících i autobus.



Obr. 8 Typy real-time systémů

Real-Time systémy musí splňovat tři základní parametry:

- ***Deadline (Latency).*** Dodržování nastaveného deadline (3.1).
- ***Synchronizace (Synchronicity).*** Synchronizace reakcí jednotlivých účastníků na vstupní události. Parametr určuje největší možnou vzájemnou odchylku reakcí mezi sebou, tzv. *jitter*.
- ***Výkon (Throughput).*** Fyzické omezení dané vlastnostmi každého systému (např. rychlost vzorkování dat, rychlost zpracování operací a jiné). Parametr také ovlivňuje aktuální zatížení systému.

3.2 Real-Time Ethernet

Ethernet byl zpočátku pro real-time komunikaci zcela nevhodný díky možnosti vzniku kolizí při režimu přijímání a vysílání half-duplex (2.1). Nedeterminismus Ethernetu představovaly náhodně vzniklé kolize, které ošetřovala metoda CSMA/CD. S příchodem nových přenosových médií (UTP kabel, optický kabel) bylo umožněno vysílání a přijímání v režimu full-duplex (2.1), při kterém již kolize nevznikají.

V současnosti problém nedeterminismu spočívá v nasazení přepínacích zařízení, *switchů* [1]. Zařízení pracuje ve druhé vrstvě ISO/OSI modelu a zajišťuje přepínání framů mezi jednotlivými úseky sítě. Nedeterminismus způsobuje zpracování framů v paměti switche. V případě zahlcení paměti switche vzniká zpoždění, které se může projevit nedodržením dealine doručení framu.

Real-Time Ethernet musí dodržet základní real-time parametry:

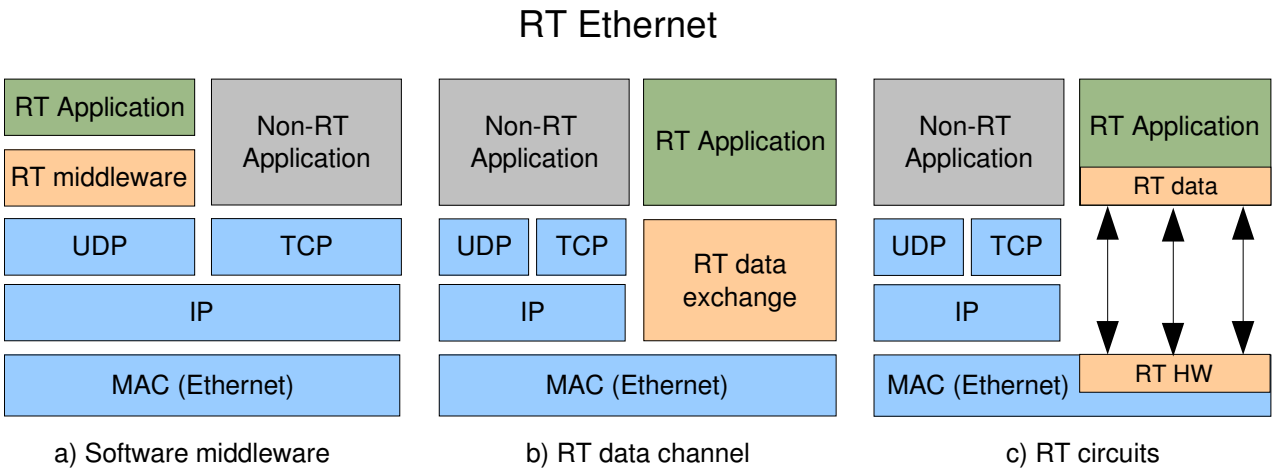
- Deadline. V průmyslu se pohybuje v řádech milisekud. Následující tabulka shrnuje deadline průmyslových zařízení:

zařízení	*deadline*
PLC, HMI	100 ms
fieldbus	10ms
motion control	100us - 1ms

Tab. 3 Deadline průmyslových zařízení

- Synchronizace. zajišťuje je *PTP protokol* (IEEE 1588), který posílá tzv. časové značky (*time stamps*) a tím umožňuje sjednocení systémového času podle hlavních hodin.
- Výkon. Použití 100 Mbit Ethernetu představuje v porovnání s rychlostmi stávajících sběrnic dostatečný výkon.

3.3 Architektury Real-Time Ethernetu



Obr. 9 Typy Real-Time Ethernetu

- ***Software middleware***. Architektura využívá maximálně stávající technologii IP/UDP. Ryze softwarové řešení. Datový tok řídí mezivrstva, nevadí tedy nespolehlivé doručování přes UDP. Problém s UDP – v případě přetížení sítě jsou první zahazovány UDP pakety. Klasická i RT data jsou distribuovány stejnou cestou. Bez dodatečné synchronizace se hodí pouze pro soft-real-time komunikaci.
- ***RT Data Channel.*** Architektura softwarově obcházející stávající technologii TCP/UDP/IP. Není třeba speciální hardware. Deterministická komunikace zajištěna dělením časových rámců na RT a non-RT část.
- ***RT Circuits.*** Architektura využívající hardwarové úpravy. Nevýhodné právě z důvodu hardwarových změn klasických Ethernetových rozhraní (NIC).

3.4 RTPS protokol

RTPS protokol vyvinula a zpřístupnila firma *RTI (Real-Time Innovation, Inc.)* [3]. Základní požadavky při návrhu protokolu byly tyto:

- Schopnost zavedení časového modelu a schopnost časování zpráv.
- Možnost volby režimu doručování zpráv podle kritérií:
  - maximální rychlost doručování (*best-effort*),
  - maximální spolehlivost doručování (*reliable*).
- Schopnost kontrolovat využívanou paměť.
- Schopnost fungování v prostředí RTOS.

Vytvořený protokol má dva základní rysy:

- využívá architekturu modelu softwarové mezivrstvy (3.2),
- využívá distribučního modelu Publish-Subscribe (1.2.2).

Základní vlastnosti protokolu se odvíjejí od předchozích dvou bodů. Architektura softwarové mezivrstvy řadí protokol do oblasti soft-real-time komunikace, pokud nebude vyřešen problém synchronizace (3.1). Softwarová mezivrstva využívá klasických transportních protokolů IP/UDP.

Mezivrstva sama zajišťuje řízení datové komunikace. RTPS protokol proto zahrnuje dva další protokoly:

- ***Publish-Subscribe Protocol***, výměna dat (publikací),
- ***Composite State Protocol (CST),*** výměna stavových informací.

3.4.1 Publish-Subscribe protokol

Klasický Publish-Subscribe model (1.2.2) byl protokolem RTPS doplněn o časové parametry a parametr určující režim doručování dat. Distribuční model Publish-Subscribe využívá mezivrstvu k vytvoření komunikačních nodů:

- ***Publisher***, nod, který se stará o produkci dat. Publisher pomocí mezivrstvy registruje název (*topic*) a type (*type name*) dat, která bude publikovat (*issue*). Data jsou produkována tak rychle, jak to umožňuje technologie použitá k jejich získávání (např. čidlo vydává data každých 10ms). Neexistuje žádný parametr nastavující rychlost publikování dat. Na síti se můžou vyskytovat Publisheři, kteří produkují shodná data (např. data záložních čidel). Je třeba tyto Publishery mezi sebou odlišit. Publisher vlastní dva parametry *strength* (síla, váha) a *persistence* (platnost, trvanlivost). V případě publikace shodných dat od různých Publisherů, jsou publikována data Publisheru s vyšší hodnotou parametru strength. V případě selhání Publishera nebo překročení parametru persistence jsou publikovaná data nahrazena daty Publishera s druhým nejvyšším parametrem strength. Po obnově prvního Publishera dojde ke změně platných dat na data obnoveného Publishera. Tímto mechanismem je zaručeno plynulé zálohování bez přerušení komunikace na síti.

parametr	***význam***
topic	jméno publikovaných dat
type name	jméno datového typu
strength	váha Publishera
persistence	doba platnosti dat

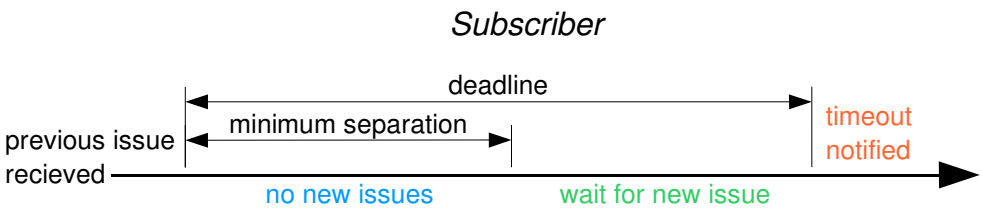
Tab. 4 Parametry Publishera

Obr. 10 Grafické znázornění parametrů persistence a strength

- ***Subscriber***, nod, který odebírá data. Subscriber pomocí mezivrstvy registruje název (*topic*) a type (*type name*) dat vždy jediné publikace, kterou chce odebírat. Subscriber definuje dva časové parametry: *minimum separation* a *deadline*. Minimum separation je časový interval měřený od poslední přijaté zprávy. Po dobu trvání intervalu nejsou přijímána nová data. Tento časový úsek slouží Subscriberovi ke zpracování předešlé přijaté publikace. Po uplynutí této doby Subscriber začne opět přijímat data. Nedorazí-li data před uplynutím druhého časového intervalu - deadline, je ohlášena chyba.

parametr	***význam***
topic	jméno přijímaných dat
type name	jméno datového typu
minimum separation	doba nepřijímání nových zpráv
deadline	deadline

Tab. 5 Parametry Subscribera



Obr. 11 Grafické znázornění parametrů minimum separation a deadline

3.4.2 Composite State Protocol

Composite State Protocol (dále CST) definuje speciální síťové objekty. Samotný protokol slouží k výměně stavových informací mezi objekty.

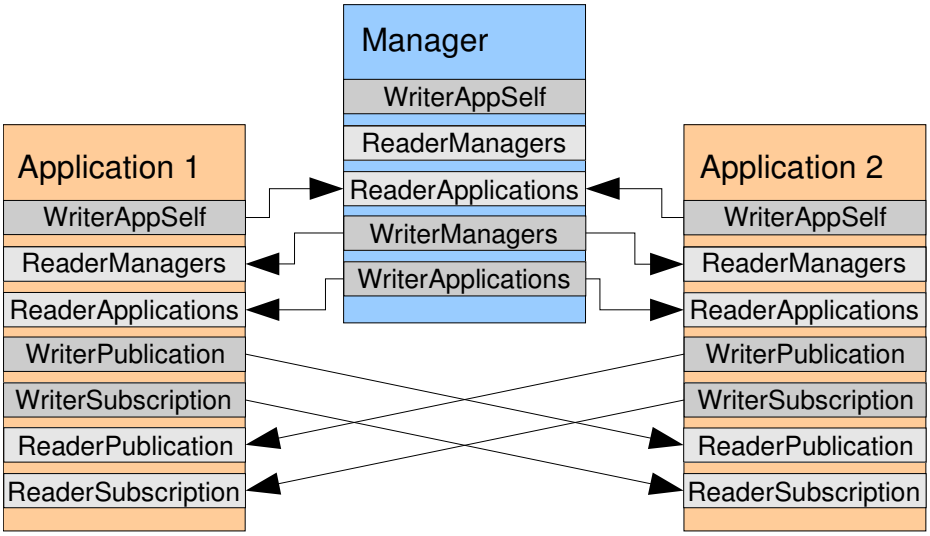
Existují dva druhy síťových aplikací:

- ***Manager***, speciální aplikace, která slouží k vyhledávání a informování ostatních aplikací. Manager vede záznamy o svých spravovaných aplikacích a jejich parametrech. Manager komunikuje s dalšími managery a vyměňují si seznamy spravovaných aplikací. Kdykoliv se k manageru přihlásí nová aplikace, zjistí její atributy a rozešle je všem známým nodům v síti.
- ***Managed Application***, aplikace spravovaná jedním i více Managery. Obsahuje jednoho i více Publisherů a Subscriberů.

Atributy síťových objektů (*network objects* [3]) reprezentuje tzv. *composite state* [3]. Jednotlivé síťové objekty si mezi sebou posílají RTPS zprávy a vyměňují si tak nejen informace o svých atributech, ale i publikovaná data.

Základní objekty pro výměnu composite state jsou:

- ***Writer***, zapisuje a odesílá aktuální stav svého objektu.
  - *CST Writer, z*ápisuje stav síťového objektu.
  - *Publication,* přenos publikace pro Subscriber.
- ***Reader***, přijímá stav zapsaný Writerem a provádí jeho rekonstukci.
  - *CST Reader*, příjem stavu síťového objektu.
  - *Subscription*, odběr publikací.



Obr. 12 Objekty Manager a ManagedApplication

Každý *Manager* se skládá z pěti objektů:

- ***WriterApplicationSelf.*** CSTWriter, kterým Manager poskytuje informace o sobě ostatním Managerům.
- ***ReaderManagers.*** CSTReader, kterým Manager získává informace o ostatních Managerech.
- ***WriterManagers.*** CSTWriter, kterým Manager poskytuje svým aplikacím informace o ostatních Managerech.
- ***ReaderApplications.*** CSTReader, pro registraci aplikace (lokální, vzdálené) k Manageru.
- ***WriterApplications.*** CSTWriter, kterým Manager posílá informace o svých registrovaných aplikacích ostatním Managerům.

Každá *ManagedApplication* se skládá ze sedmi objektů:

- ***WriterApplicationSelf.*** CSTWriter, kterým aplikace registruje k Managerovi.
- ***ReaderManagers***. CSTReader, kterým aplikace získává informace o Managerech.
- ***ReaderApplications***. CSTReader, kterým aplikace získává informace o ostatních aplikacích.

- ***WriterPublications***. Writer, kterým aplikace poskytuje jednotlivá vydání publikace jedné nebo více instancím objektu Subscription. K předání publikace je použito Publish-Subscribe protokolu (3.3.1).
- ***WriterSubscriptions.*** Writer, kterým aplikace poskytuje informace o vlastních objektech typu Subscription.
- ***ReaderPublications.*** Reader, kterým aplikace přijímá informace o objektech typu Publication.
- ***ReaderSubscriptions.*** Reader, kterým aplikace přijímá jednotlivá vydání publikace od objektu Publication. K předání publikace je použito Publish-Subscribe protokolu (3.3.1).

Kapitola 4.

Programovací jazyk C, Java a JNI rozhraní

Tato kapitola se zabývá popisem vlastností a způsobu interpretace programů ve dvou odlišných programovacích jazycích C a Java. Dále je představeno rozhraní *JNI (Java Native Interface)*, které umožňuje kooperaci obou programovacích jazyků. Podkapitola o JNI rozhraní je navíc doplněna o příklad vytvoření jednoduchého programu, ve kterém jsou popsány základní způsoby práce s JNI rozhraním.

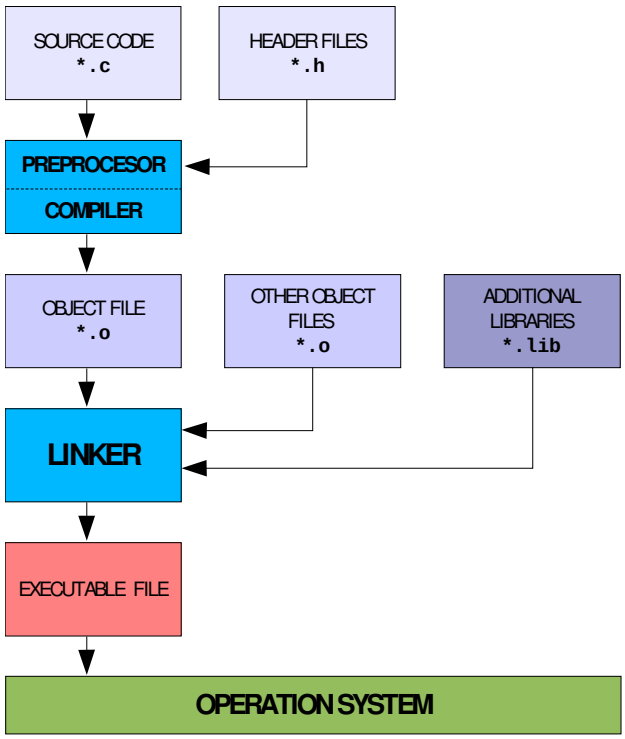
4.1 Jazyk C

Jazyk C [4] patří do kategorie procedurálních programovacích jazyků. Instrukce zdrojového kódu jsou vykonávány sekvenčně. Hodnoty uložené v paměti počítače reprezentují proměnné. Procedury slouží k vykonávání předdefinovaných operací nad hodnotami proměnných. Uvolňování paměti je u normáních proměnných provedeno po skončení bloku ve kterém byly definovány. Paměť, kterou programátor sám alokoval, musí také sám uvolnit, jinak může dojít k jejímu vyčerpání.

Zdrojový kód programu ve formě textového souboru je nejprve zpracován *preprocesorem*, který upraví zdrojový kód pro *překladač (compiler)*. Preprocesor vyjme komentáře, vloží hlavičkové soubory (*header files*), rozvine makra a další. Výsledkem je opět textový soubor, který je následně zpracován překladačem do relativního - objektového kódu počítače. Slovo relativní znamená, že ještě nejsou určeny adresy proměnných. Výsledkem překladu je objektový soubor. Ještě však zbývá určit absolutní adresy pro proměnné. To provede v sestavovací program tzv. *linker*. Vstupem linkeru jsou objektové soubory spolu s přídavnými knihovnami, jejichž funkce v našem programu využíváme. Po sestavení programu již máme k dispozici spustitelný soubor, který obsahuje strojový kód přímo pro procesor počítače. Celý postup je znázorněn na obrázku Obr. 13.

Stručný popis zpracování programu

- *Preprocesor* zpracuje zdrojový kód a hlavičkové soubory.
- *Compiler* přeloží výstup preprocesoru do objektového souboru, relativní adresace.
- *Linker* spojí objektové soubory spolu s přídavnými knihovnami do spustitelného souboru.
- *Operační systém* poskytne prostředí pro vykonání spustitelného souboru.



Obr. 13 Zpracování programu v jazyce C

Výhodou jazyka C je přístup k hardwaru počítače a rychlost programu. Nevýhodou jazyka je nepřenositelnost přeloženého programu, způsobená závislostí překladače na architektuře a operačním systému (*platformě*) počítače. Po překladu získáme spustitelný soubor se strojovými instrukcemi přímo pro danou platformu. Důsledkem toho je, že program přeložený na jednom počítači nebude fungovat na počítači s jinou platformou. Abychom mohli program spustit, je nutno jej přeložit pro danou platformu. To přináší nové problémy např. změny zdrojového kódu, nemožnost použití knihoven pod novou platformou a jiné. Přenášení *(portace)* programu na další platformu stojí čas strávený programováním a odlišné zdrojové kódy stejného programu pro různé platformy.

4.2 Jazyk Java

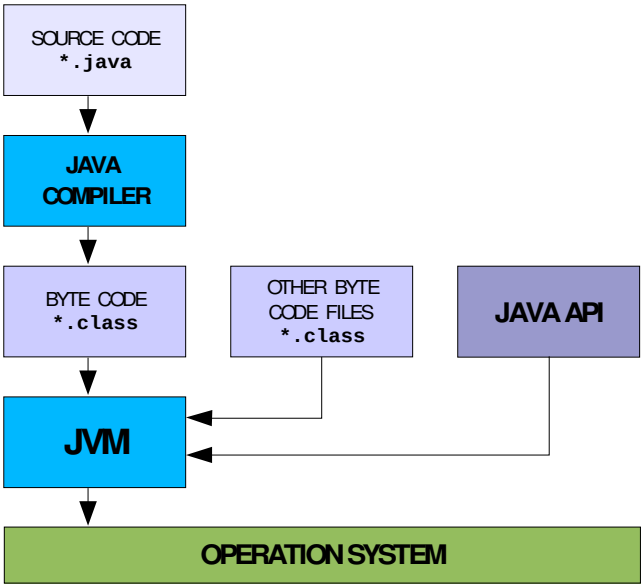
Jazyk Java [5] patří do kategorie objektově orientovaných programovacích jazyků (*OOP*). Základní jednotkou je *objekt*, který se skládá z datových položek (*fields*) a metod pro manipulaci s nimi. Vlastnosti objektu, jako např. počet a typ datových položek a metod určuje *třída* (*class*). Třídu si můžeme představit jako šablonu objektu. Instance tříd tj. objekty jsou vytvářeny dynamicky. Rušení objektu probíhá dynamicky pomocí vlastního mechanismu pro uvolňování alokované paměti tzv. *garbage collectoru (GC).* GC zjistí, na které objekty neexistuje odkaz a ty v případě vyčerpávání paměti zruší.

Způsob zpracování programu jazyk Java řadí mazi *interpretované* programovací jazyky. Zdrojový kód programu *překladač (compiler)* přeloží na optimalizovanou sadu instrukcí tzv. *byte code* [5], který je nezávislý na platformě i architektuře stroje. Program přeložený na počítači s platformou Linux lze spustit na počítači s Unixem, MacOS či Windows. Interpretaci byte code provede virtuální stroj *Java Virtual Machine (JVM)* [5].

Virtuální stroj JVM je software vytvořený pro danou platformu. Je to ta část Javy, která řeší rozdíly mezi jednotlivými počítačovými architekturami a operačními systémy. JVM zaručuje jednotnou interpretaci byte codu, tím je zaručena přenositelnost programů Javy za cenu jejich pomalejšího vykonávání.. Pro přístup k operačnímu systému dané platformy JVM využívá tzv. *nativních metod*.

Stručný popis zpracování programu

- Compiler přeloží zdrojový kód do byte kódu.
- JVM interpretuje byte kód.
- JVM se s operačním systémem dorozumívá přes nativních metody.



Obr. 14 Zpracování programu v jazyce Java

4.3 Java Native Interface

V předchozí kapitole bylo zmíněno, že virtuální stroj JVM užívá nativních metod . Tyto metody mohou být zpřístupněny i programátorovi pomocí tzv. *Java Native Interface (JNI)* [6] (Příloha A, Obr. 30). JNI je speciální API rozhraní umožňující spolupráci Javy s jiným programovacím jazykem (např. C, C++).

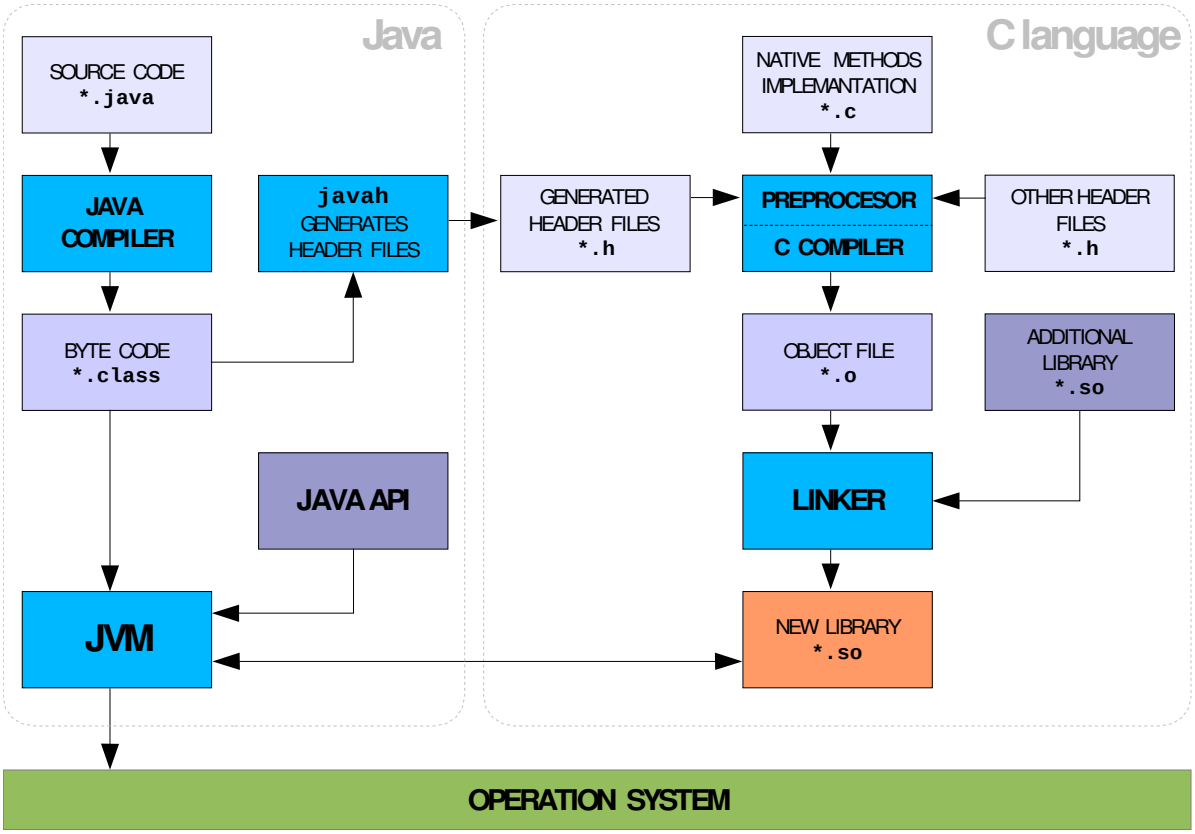
Důvod používání JNI rozhraní tkví v tom, že pomocí nativního jazyka dokážeme vykonat operace, na které samotný jazyk Java nestačí (např. real-time aplikace, aplikace přímo přistupující do paměti). JNI rozhraní umožňuje volat nativní metody z kódu Javy a naopak, z nativního kódu volat metody Javy.

Cena, která se platí za použití JNI rozhraní je ztráta přenositelnosti. Kód vytvořený v Javě zůstává nadále přenositelný. Část kódu vytvořená v nativním jazyce je platformě závislá. Proto je vždy dobré zvážit nutnost použití tohoto rozhraní.

4.3.1 Zpracování programu

Stručný popis vytváření programu pomocí JNI:

- vytvořit a zkompilovat program v jazyce Java,
- vygenerovat hlavičkové soubory
- vytvořit a zkompilovat zdrojové kódy v jazyce C do dynamické knihovny,
- spustit program vytvořený v Javě (dynamická knihovna se nahraje automaticky).



Obr. 15 Vytváření programu pomocí Javy, JNI a jazyku C

4.3.2 Praktický příklad vytváření programu pomocí JNI

Následující zdrojové kódy slouží pouze k demostraci základních principů vytváření programu pomocí Javy, JNI rozhraní a jazyku C. Těchto principů bylo dále využito při řešení úkolu vytvoření rozhraní jazyka Java pro knihovnu `liborte` (kapitola 7).

Uváděné příkazy jsou příkazové řádky jsou příkazy operačního systému Linux. V ostatních operačních systémech jsou příkazy obdobné.

Program v jazyce Java

Nejdříve vytvoříme část programu v jazyce Java. Klasický zdrojový kód Javy je doplněn o nativní metody, které obsahují ve své deklaraci klíčové slovo `native` (zelená oblast na Obr. 16). Metoda pouze deklaruje návratový typ a může obsahovat argumenty, nemá však definováno tělo, to dodefinuje nativní jazyk. Do zdrojového kódu programu je třeba uvést příkaz pro načtení dynamické knihovny, která bude obsahovat těla nativních metod (modrá oblast na Obr. 16). Soubor knihovny se jmenuje `libtest.so`. Neuvádí se celý název knihovny, ale vypouští se předpona `lib` a koncovka (`.so` nebo `.dll`). Jméno balíčku (šedivá oblast na Obr. 16) bude použito při pojmenování vygenerovaného hlavičkového souboru a nativních funkcí.

```
/* ImaginaryObject.java */

package my.jni.test;

public class ImaginaryObject {

    /** constructor */
    public ImaginaryObject() {
        number = 1;
    }

    // load library
    static {
        System.loadLibrary("test");
    }

    // main method
    public static void main(String[] args) {

        ImaginaryObject iobject = new ImaginaryObject();

        ImaginaryObject.printWord("Hi all from Java!");
        iobject.printNumber();
        iobject.setNumber(2);
        iobject.printNumber();
    }

    // methods
    public void printHello() {
        System.out.println("Hello from Java!");
    }

    // native methods
    public native static void printWord(String word); // static method!
    public native void printNumber();
    public native void setNumber(int newValue);

    // data fields
    public int number;
}
```

Obr. 16 Zdrojový kód jazyka Java (ImaginaryObject.class)

Zdrojový kód přeložíme příkazem:

```
javac -sourcepath ./ my/jni/test/ImaginaryObject.java
```

Získáme soubor s byte kódem, `ImaginaryObject.class`.

Generování hlavičkových souborů

Po přeložení zdrojového kódu můžeme vygenerovat hlavičkové soubory. Ty tvoří spojnici mezi nativní metodou jazyka Java a budoucí funkcí jazyka C. Hlavičkový soubor získáme užitím příkazu `javah` s parametrem jména třídy, která obsahuje nativní metody:

```
javah -jni -classpath ./ my.jni.test.ImaginaryObject
```

Ve vygenerovaném hlavičkovém souboru nalezneme deklarace všech nativních metod dané třídy (zelená oblast na Obr. 17). Název souboru je odvozen podle názvu třídy a názvu balíčku, ve kterém se třída nachází. V našem případě `my_jni_test_ImaginaryObject.h`.

```
 * DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class my_jni_test_ImaginaryObject */

#ifndef _Included_my_jni_test_ImaginaryObject
#define _Included_my_jni_test_ImaginaryObject
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     my_jni_test_ImaginaryObject
 * Method:    printWord
 * Signature: (Ljava/lang/String;)V
 */
JNIEXPORT void JNICALL Java_my_jni_test_ImaginaryObject_printWord
  (JNIEnv *, jclass, jstring);

/*
 * Class:     my_jni_test_ImaginaryObject
 * Method:    printNumber
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_my_jni_test_ImaginaryObject_printNumber
  (JNIEnv *, jobject);

/*
 * Class:     my_jni_test_ImaginaryObject
 * Method:    setNumber
 * Signature: (I)V
 */
JNIEXPORT void JNICALL Java_my_jni_test_ImaginaryObject_setNumber
  (JNIEnv *, jobject, jint);

#ifdef __cplusplus
}
#endif
#endif
```

Obr. 17 Zdrojový kód vygenerovaného hlavičkového souboru (my_jni_test_ImaginaryObject.h)

V oranžové oblasti (Obr. 17) je znázoněno vložení důležitého hlavičkového souboru `jni.h`. V tomto souboru jsou mimo jiné definovány datové typy JNI a makra uvedené kolem návratového typu nativní funkce, `JNIEXPORT` a `JNICALL`. Makra zajišťují nativní funkci správné propojení a volací konvence při exportování z dynamické knihovny.

Vytváření zdrojových kódů v nativním jazyce

Z hlavičkového souboru si zkopírujeme protopyp nativní funkce (záměrně je použit termín *funkce*, neboť v terminologii jazyka C se výraz *metoda* neužívá) a vložíme ho do souboru, kde doplníme tělo metody v jazyce C. Jméno funkce je vytvořeno z klíčového slova `Java`, názvu balíčku (`my.jni.test`), názvu své rodičovské třídy (`ImaginaryObject`) a jména nativní metody v rodičovské třídě:

```
Java_my_jni_test_ImaginaryObject_printWord()
Java_my_jni_test_ImaginaryObject_printNumber()
Java_my_jni_test_ImaginaryObject_setNumber()
```

Ještě než popíšeme způsob zacházení s argumenty nativní funkce, je potřeba vysvětlit, jak se v JNI pracuje s datovými typy.

Datové typy JNI

V jazyce Java jsou známy dvě skupiny datových typů:

- ***primitivní datové typy*** (boolean, byte, int, ...),
- ***referenční datové typy*** (pole a objekty).

Primitivní datové typy mají v Javě pevně určenou velikost (z důvodu přenostitelnosti byte kódu). Proto se při generování prototypu nativní funkce s argumenty nepoužívají datové typy jazyka C (datové typy jsou platformě závislé). JNI v hlavičkovém souboru `jni.h` definuje vlastní datové typy. Tabulka Tab.6 a Tab.7 představuje datové typy Javy a jim odpovídající datové JNI typy.

primitivní datový typ v jazyce Java	*datový typ pro jazyk C*	*deskriptor datové složky*
`boolean`	`jboolean`	`Z`
`byte`	`jbyte`	`B`
`char`	`jchar`	`C`
`short`	`jshort`	`S`
`int`	`jint`	`I`
`long`	`jlong`	`J`
`float`	`jfloat`	`F`
`double`	`jdouble`	`D`

Tab. 6 Primitivní datové typy Javy a odpovídající JNI typy

referenční datové typy -Java	*datové typy-C*	*deskriptor datové složky*
`string`	`jobject`	`Ljava/lang/String;`
`pole` např. `int[]`	`jintArray`	`[I`
libovolnýobjekt např. `org.ocera.orte.AppInfo`	`jobject`	`Lorg/ocera/orte/types/AppInfo;`

Tab. 7 Referenční datové typy Javy a odpovídající JNI typy

Vedle primitivních a referenčních datový typů definuje JNI ještě pomocné datové typy.

pomocný JNI datový typ	*popis*
`JavaVM`	ukazatel na virtuální stroj Java
`JNIEnv`	ukazatel na prostředí Javy
`jclass`	typ zastupující třídu objektu
`jmethodId`	identifikátor metody objektu
`jfieldId`	identifikátor datové položky objektu

Tab. 8 Pomocné datové typy JNI

Argumenty nativní metody

Nativní metoda může a nemusí obsahovat argumenty. Vygenerovaný prototyp nativní funkce obsahuje datové typy JNI odpovídající datovým typům Javy (podle Tab.6 a Tab.7). Dále jsou přidány dva pomocné argumenty datových typů JNI:

- **`JNIEnv`**, který zajišťuje odkaz na prostředí Javy,
- **`jclass`**, je-li nativní metoda Javy statická, zajišťuje odkaz na svou třídu,
- **`jobject`**, není-li nativní metoda Javy statická, zajišťuje odkaz na svůj objekt.

nativní metody:

```
public native static void printWord(String word); // ! static method
public native void printNumber(); // nonstatic method
public native void setNumber(int newValue); // nonstatic method
```

prototypy nativních funkcí:

```
JNIEXPORT void JNICALL Java_my_jni_test_ImaginaryObject_printWord
  (JNIEnv *env, jclass cls, jstring java_word);
JNIEXPORT void JNICALL Java_my_jni_test_ImaginaryObject_printNumber
  (JNIEnv *env, jobject obj);
JNIEXPORT void JNICALL Java_my_jni_test_ImaginaryObject_setNumber
  (JNIEnv *env, jobject obj, jint new_value);
```

Druhý přidávaný argument rozlišuje `static/nonstatic` nativní metodu. Kopíruje tak chování Javy. Statickou metodu lze volat, i když neexistuje žádná instance třídy (tj. objekt), ve kterém je tato metoda definována. Volání statické metody je provedeno pomocí názvu její třídy, například:

```
ImaginaryObject.printWord("Hi all from Java!");.
```

U nonstatic metody je třeba funkci volat pomocí názvu instance třídy (tj. objektu), například:

```
iobject.printNumber();.
```

Nativní funkci je třeba sdělit, kdo danou metodu volá jestli třída nebo instance třídy, a proto jsou tyto argumenty závislé na typu nativní metody (`static/nonstatic`).

Často se stává, že v nonstatic metodě potřebujeme získat ukazatel na třídu rodičovského objektu typu `jclass`. Tento ukazatel vyžadují některé metody používané pro práci s datovými položkami. Jsou dva způsoby, jak získat odkaz na třídu:

```
jclass cls;
cls = (*env)->FindClass(env, "my/jni/test/ImaginaryObject");
```

nebo

```
cls = (*env)->GetObjectClass(env, obj);.
```

V prvním způsobu jsou argumenty ukazatel na prostředí Javy (`env`) a celý název třídy (včetně jména balíčku). Ve druhém způsobu je, kromě ukazatele na prostředí Javy (`env`), znát ukazatel na instanci dané třídy (`obj`).

Zdrojové kódy v nativním jazyce

Následující tři obrázky představují zdrojové kódy implementující těla nativních funkcí. Oranžově vyznačená oblast zvýrazňuje vložení vygenerovaného hlavičkového souboru (obr.17), zelená oblast zvýrazňuje vygenerovanou deklaraci nativní funkce a konečně modré oblasti zvýrazňují způsoby práce s datovými položkami a metodami z prostředí Javy.

```
/* printWord.c */

// add pregenerated header file
#include "my_jni_test_ImaginaryObject.h"

JNIEXPORT void JNICALL
Java_my_jni_test_ImaginaryObject_printWord
(JNIEnv *env, jclass cls, jstring java_word)
{
  const char *word;
  jmethodID mid;

  word = (*env)->GetStringUTFChars(env, java_word, 0);
  if(word == 0) return;
  printf("printed from C: word = %s \n",word);
  // free memory
  (*env)->ReleaseStringUTFChars(env, java_word, word);

  // call printHelloWorld method ////////////////////////////
  // get methodId
  mid = (*env)->GetMethodID(env, cls, "printHello", "()V");
  if(mid == 0) return;
  // call method
  (*env)->CallStaticVoidMethod(env, cls, mid);
  return;
}
```

Obr. 18 Zdrojový kód s implementací těla nativní funkce (printWord.c)

```c
/* printNumber.c */

// add pregenerated header file
#include "my_jni_test_ImaginaryObject.h"

JNIEXPORT void JNICALL
Java_my_jni_test_ImaginaryObject_printNumber
(JNIEnv *env, jobject obj)
{
  jclass cls;
  jfieldID fid;
  jint number = 0;

  // get class of main object
  cls = (*env)->FindClass(env, "my/jni/test/ImaginaryObject");
  if(cls == 0)  return;
  // get fieldID of number
  fid = (*env)->GetFieldID(env,
                           cls,
                           "number", // name of java field
                           "I");     // type descriptor
  if(fid == 0) return;
  // get java field's value
  number = (*env)->GetIntField(env,obj,fid);
  printf("printed from C: number = %d \n", number);
  return;
}
```

Obr. 19 Zdrojový kód s implementací těla nativní funkce (printNumber.c)

```c
/* setNumber.c */

// add pregenerated header file
#include "my_jni_test_ImaginaryObject.h"

JNIEXPORT void JNICALL
Java_my_jni_test_ImaginaryObject_setNumber
(JNIEnv *env, jobject obj, jint new_value)
{
  jclass cls;
  jfieldID fid;

  // get class of main object
  cls = (*env)->FindClass(env, "my/jni/test/ImaginaryObject");
  if(cls == 0)  return;
  // get fieldID of number
  fid = (*env)->GetFieldID(env,
                           cls,
                           "number", // name of java field
                           "I");     // type descriptor
  if(fid == 0)  return;
  // set new value
  (*env)->SetIntField(env, obj, fid, new_value);
  return;
}
```

Obr. 20 Zdrojový kód s implementací těla nativní funkce (setNumber.c)

Přístup k datovým položkám Javy

Může se stát, že potřebujeme znát nebo změnit stav datové položky z prostředí Javy. Abychom mohli k dané položce přistoupit, musíme získat její indetifikátor.

```
jfieldID fid;
 fid = (*env)->GetFieldID(env,
                          cls,
                          "number", // name of java field
                          "I");     // type descriptor for int
```

Po získání identifikátoru stačí zavolat funkce Get<Type>Field a Set<Type>Field.

```
 number = (*env)->GetIntField(env,obj,fid); // Get
 (*env)->SetIntField(env, obj, fid, new_value); // Set
```

Volání metody z prostředí Javy

Může se stát, že potřebujeme zavolat metodu z prostředí Javy. Abychom mohli k dané metodě přistoupit, musíme získat její indetifikátor.

```
jmethodID mid;
 mid = (*env)->GetMethodID(env, cls, "printHello", "()V");
```

Volání metody provedeme následujícími funkcemi (podle typu metody static/nostatic)

`CallStatic<Type>Method() nebo Call<Type>Method()`:

např.

```
 (*env)->CallStaticVoidMethod(env, cls, mid);
```

Kapitola 5.
Projekt ORTE

Tato kapitola je věnována projektu ORTE. V úvodu jsou zmíněny cíle projektu a dále jsou popsány jeho podčásti. Největší důraz je kladen na knihovnu `liborte` poskytující programátorům API rozhraní pro vytvoření soft-real-time aplikací.

5.1 Představení projektu ORTE

Projekt *ORTE (OCERA Real-Time Ethernet)* [7] je součástí evropského open source projektu *OCERA (Open Components for Real-Time Applications)* [8] s označením IST-2001 35102. Cílem projektu OCERA je vytvořit integrované prostředí pro embedded real-time aplikace. Celý projekt je rozdělen na více podčástí, které řeší své dílčí problémy.

ORTE spadá do skupiny řešící komunikaci, na kterou jsou v případě real-time aplikací kladeny přísné nároky. ORTE je volně šiřitelná multiplatformní implementace RTPS komunikačního protokolu (3.4). Podporovány jsou architektury typu little endian i big endian [9], s operačními systémy Linux, RTLinux, Unix, RTAI, FreeBSD, MacOS, Windows.

ORTE je distribuováno ve formě zdrojových kódů vytvořených v jazyce C. Zdrojové kódy jsou uzpůsobeny pro multiplatformní použití. Zdrojové kódy si uživatel přeloží přímo pro svou architekturu a platformu (obě musí být podporovány). Výsledkem překladu je knihovna a pomocné nástroje (ukázkové aplikace (demo), IDL compiler [10], dokumentace). Knihovna uživateli poskytuje aplikační rozhraní (API). Uživatel si může vybrat, zda pro tvorbu vlastní aplikace využije knihovny s API rozhraním v jazyce C nebo Java (7).

5.2 Knihovna *liborte*

Knihovna `liborte` (dále knihovna) byla vytvořena s úmyslem usnadnit práci programátorům tvořícím síťové aplikace. Knihovna poskytuje API rozhraní v jazyce C. Zdrojové kódy knihovny mohou být přeloženy do podoby statické nebo dynamické knihovny [11].

- **`orte-X.X.X/orte/include/orte/`**, adresář obsahující hlavičkovými soubory.
- **`orte-X.X.X/orte/liborte/`**, adresář obsahující zdrojovými kódy.
- **`orte-X.X.X/_compiled/lib/`**, adresář obsahující vytvořenou knihovnu.

5.3 API rozhraní

Nabízené funkce [7] můžeme rozdělit do několika základních bloků:

- ***Domain.*** Doména tvoří základ každé ORTE aplikace, funkce tohoto bloku zajišťují vytvoření, ovládání a rušení domény.

  Existují dva typy domén:

  - doména typu *ManagerDomain,*
  - doména typu *ApplicationDomain.*

- ***Registration.*** Soubor funkcí určených pro registraci datových typů. Publisher registruje své datové typy, které bude publikovat. Subscriber registruje datové typy, které chce přijímat.
- ***Publication.*** Soubor funkcí sloužící k vytvoření, ovládání a rušení Publishera.
- ***Subscription.*** Soubor funkcí sloužící k vytvoření, ovládání a rušení Subscribera.
- ***Pomocné funkce.*** Inicializace, uspání aplikace, podrobné informativní výpisy.

Rozhraní definuje i datové typy [10]:

- ***Datové struktury.*** Tvoří datové typy argumentů výše uvedených funkcí. Nejčastěji nesou informace a aktuálním nebo požadovaném stavu ORTE objektů.
- ***Ukazatele na funkce.*** Tělo funkce si vytvoří uživatel na základě vlastních požadavků. Příkladem jsou funkce reagující na událost. ORTE tak poskytuje možnost reakce na události:
  - *Interní událost*, ošetřena tzv. *callback* funkcí. V případě Publishera je callback volán po odeslání dat, v případě Subscribera po příjmu nových dat.
  - *Externí událost,* ošetřena tzv. *event* funkcemi. Události jsou vyvolány přítomností jiných aplikací s RTPS protokolem (např. událost vyvolaná při objevení nového Publishera nebo Subscribera).

5.4 Režimy komunikace

ORTE umožňuje zasílání zpráv ve dvou režimech:

- ***Best-effort.*** Zprávu (*issue*) doručit co nejdříve, nevadí případné ztráty některých zpráv – za krátký okamžik přijdou nová data.
- ***Strict reliability.*** Zajišťuje spolehlivé doručení všech zpráv v daném pořadí tzv. potvrzované doručování (např. pro posílání řídicích sekvencí).

5.5 Serializace a deserializace dat

Serializace slouží k přípravě dat určených pro odeslání. Data, které chceme publikovat jsou serializační funkcí vložena do bufferu, ze kterého jsou posléze odeslána. Deserializační funkce slouží k získání dat z bufferu. Programátor potřebuje vytvořit příslušné serializační a deserializační funkce pro všechny datové typy proměnných, jejichž hodnoty jsou publikovány.

Projekt ORTE obsahuje pomocný nástroj IDL compiler, který umožňuje automatické generování serializačních a deserializačních funkcí na základě seznamu jmen a datových typů proměnných. IDL compiler umožňuje vygenerovat zdrojové kódy pro jazyk C i Java.

5.6 Příklad užití *liborte*

Hlavním cílem `liborte` je ulehčit práci programátorům. Jednoduché aplikace lze vytvořit pomocí několika příkazů. Příklady jsou uvedeny hlavně z důvodu pozdějšího srovnání s Java API rozhraním knihovny `libjorte`. Barevně jsou vyznačeny stejné nebo podobné části obou aplikací, Publishera a Subscribera.

```
/* Publisher.c */

#include "orte.h"

send_callback(...) {
  // user defined
}

int main(int argc, char *args[]) {
  ORTEDomain domain;
  ORTEPublication publisher;

  ORTEInit();

  // create app domain
  domain = ORTEDomainAppCreate(...);

  // register data type to send
  ORTETypeRegisterAdd(domain,...);

  // create pulisher
  publisher = ORTEPublicationCreate(domain,
                                    send_callback,
                                    ...);
  // send data
  ORTEPublicationSend(publisher);

  // destroy publisher
  ORTEPublicationDestroy(publisher);

  // destroy domain
  ORTEDomainAppDestroy(domain);

}
```

```
/* Subscriber.c */

#include "orte.h"

recv_callback(...) {
  // user defined
}

int main(int argc, char *args[]) {
  ORTEDomain domain;
  ORTESubscription subscriber;

  ORTEInit();

  // create app domain
  domain = ORTEDomainAppCreate(domain,...);

  // register data type to send
  ORTETypeRegisterAdd(domain,...);

  // create subscriber
  subscriber = ORTESubscriptionCreate(domain,
                                      recv_callback,
                                      ...);

  // destroy subscriber
  ORTESubscriptionDestroy(subscriber);

  // destroy domain
  ORTEDomainAppDestroy(domain);

}
```

Obr. 21 Příklad užití API rozhraní knihovny liborte

Kapitola 6.
Popis řešených úloh

Tato kapitola popisuje, základní úlohy této práce. Způsoby návrhu řešení a předpokládané výsledky. Ve druhé části kapitoly jsou představeny a porovnány projekty zabývající se stejnými úlohami.

6.1 Hlavní úkoly a postup řešení

1. Vytvořit Java rozhraní pro knihovnu projektu ORTE

Knihovna projektu ORTE (5) poskytuje API rozhraní v jazyce C. Úkolem je navrhnout a vytvořit nové rozhraní pro jazyk Java. Základní koncepce spočívá v propojení stávajícího API rozhraní s nově vytvořeným rozhraním v jazyce Java. Kooperace jazyka C s jazykem Java je vyřešena pomocí JNI rozhraní (4.3). Výsledkem práce bude Java API rozhraní ekvivalentní k původnímu API rozhraní.

2. Otestovat a aplikovat Java rozhraní v praxi

Po vytvoření Java API rozhraní bude potřeba ověřit jeho funkčnost. Zároveň je potřeba pro projekt ORTE vytvořit monitorovací program s grafickým rozhraním (dále GUI). Program bude mít za úkol automaticky sledovat síťové aplikace využívající RTPS protokolu a monitorovat jejich parametry. Pro vytvoření grafického rozhraní byla vybrána komponenta Swing poskytovaná jazykem Java. Tato komponenta obsahuje všechny potřebné prvky pro navržené GUI (8.2.1). Výsledkem práce bude aplikace splňující výše uvedené požadavky.

3. Podpora RTPS protokolu pro program Ethereal

RTPS protokol definuje 10 typů zpráv [3]. Je třeba vytvořit vhodný nástroj pro monitorování zpráv a zobrazování hodnot jejich parametrů. Úkolem je vytvořit podporu RTPS protokolu pro monitorovací program Ethereal. Důvody, proč byl vybrán právě program Ethereal jsou tyto:

- open-source program (GPL licence),
- použitelný na většině platforem,

- podpora vývoje nástrojů pro nové protokoly.

Výsledkem práce bude začlenění zdrojových kódů, umožňující monitorování RTPS komunikace, do stabilní verze programu.

6.2 Podobné projekty

Ekvivalentní projekt k projektu ORTE je *NDDS (Network Data Delivery Service)* [12] od firmy RTI, která protokol RTPS vyvinula. Jedná se o velice propracovanou komerční implementaci RTPS protokolu, která byla nasazena do provozu. NDDS s úspěchem vyžívají někteří výrobci PLC pro distribuci řídících dat po Ethernetu nebo pro komunikaci elektronických systémů vojenských plavidel a další.

NDDS poskytuje programátorům API rozhraní pro programovací jazyky C, C++ a Java. Serializaci a deserializaci dat provádí nástroj *nddsgen*. ORTE pro stejné účely používá ekvivalentní program IDL compiler (5.5). Oba nástroje užívají CDR reprezentaci datových typů [3].

RTI poskytuje komerční balík programů *WaveWorks* určení k monitorování a vizualizaci RTPS komunikace. Balík obsahuje programy *WaveSurf* a *WaveScope*.

Program WaveSurf, jehož ekvivalentem z projektu ORTE je program RTPSeYe (8), slouží k objevování aplikací používajících RTPS protokolu, jejich hierarchickému zobrazení a zobrazení parametrů vybrané aplikace. WaveSurf je v porovnání s RTPSeYe ve stabilní verzi, má více režimů zobrazování nalezených aplikací a hlavně možnost měnit na dálku některé parametry nalezených aplikací.

Program WaveScope slouží k důkladnému monitorování vlastností/parametrů RTPS komunikace na daném komunikačním nodu. Částečným ekvivalentem z projektu ORTE je program Ethereal. Zatímco Ethereal je univerzální program umožňující monitorovat komunikaci více protokolů (v době psaní tohoto textu 683 protokolů), WaveScope je program vytvořený a optimalizovaný přímo pro monitorování RTPS protokolu.

Tabulka Tab.9 shrnuje vlastnosti základní NDDS a ORTE:

	NDDS	***ORTE***
producent	RTI	OCERA
licence	komerční	GPL (open-source)
status	stabilní	beta verze
platforma	multiplatformní	multiplatformní
API rozhraní	C, C++, Java	C, C++, Java
podpůrné nástroje	nddsgen WaveScope WaveSurf demoaplikace	IDL Ethereal RTPSeYe demoaplikace

Tab.9 Porovnání NDDS a ORTE

Kapitola 7.
Java rozhraní pro projekt ORTE

Následující kapitola popisuje způsob návrhu a realizace balíčku `org.ocera.orte` a knihovny `libjorte`. Použití obou částí poskytuje programátorům ORTE API rozhraní v jazyce Java. Na konci kapitoly je uveden krátký příklad použití.

7.1 Základní koncepce

Kapitola 4.3 se zabývala JNI rozhraním, které umožňuje kooperaci programovacího jazyka Java s dalším programovacím jazykem. Základní myšlenkou použití je vytvořit zdrojový kód v jazyce Java, který obsahuje speciální nativní metody, které nemají definované tělo. Těla nativních metod jsou dodefinována až ve zdrojovém kódu druhého tzv. nativní programovacího jazyka. Důvody pro použití JNI rozhraní jsou nejčastěji v tom, že Java se nehodí pro řešení nebo neumí vyřešit některé problémy, které zase umí vyřešit jiné programovací jazyky.

7.2 Balíček *org.ocera.orte*

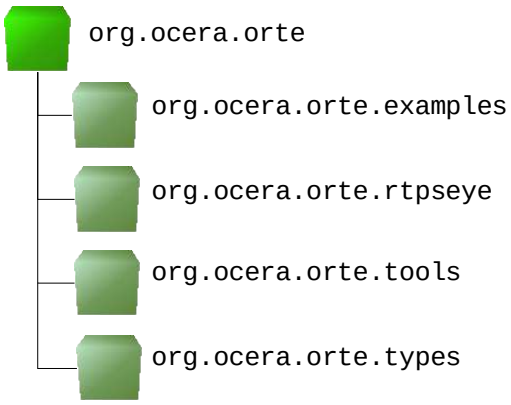
Balíky (*packages*) [5] v Javě představují ekvivalent ke knihovnám ostatních programovacích jazyků. Balík je reprezentován adresářem, který obsahuje přeložené zdrojové kódy rozhraní (*interfaces*) a tříd (*classes*). Systém balíčků tvoří hierarchie takovýchto adresářů. Jména balíků odpovídají jménům jednotlivých adresářů. Aby se zabránilo shodnému pojmenování balíků, je dobré dodržovat celosvětovou konvenci pojmenování balíků, podle které je jméno hlavního balíku odvozeno od jména domény Internetu. Jméno domény je zapsáno tečkovou notací a pozpátku.

Do balíku by se měli dávat třídy a rozhraní, které spolu souvisí např. mají podobnou funkci, navzájem na sobě závisí, jedná se o části jednoho projektu atd. Systém balíků slouží pro zpřehlednění a zatřídění zdrojových kódů a udržení této hierarchie po přeložení.

7.2.1 Návrh systémů balíků

Projekt OCERA vlastní doménové jméno `ocera.org`. Aby bylo zabráněno možné záměně mezi jednotlivými projekty OCERA, bylo jméno hlavního balíčku upraveno na `org.ocera.orte`.

Obrázek Obr. 22 představuje navrženou strukturu balíčků:



Obr. 22 Struktura balíků ORTE Java API

Hlavní balíček `org.ocera.orte` je v distribuci ORTE umístněn v adresáři:

- `orte-X.X.X/orte/java/`

Po vytvoření struktury podle Obr. 22 získáme následující adresářovou hierarchii:

- `orte-X.X.X/orte/java/org/ocera/orte/`
- `orte-X.X.X/orte/java/org/ocera/orte/examples/`
- `orte-X.X.X/orte/java/org/ocera/orte/rtpseye/`
- `orte-X.X.X/orte/java/org/ocera/tools/`
- `orte-X.X.X/orte/java/org/ocera/orte/types/`

Přeložené zdrojové kódy jsou uloženy do adresáře:

- `orte-X.X.X/orte/_compiled/`

7.2.2 Návrh tříd objektů

Cílem práce je vytvořit API rozhraní pro jazyk Java ekvivalentní s rozhraním v jazyce C. Základem nového Java rozhraní je původní rozhraní obsažené v knihovně `liborte`. V kapitole 5.3 jsou popsány bloky funkcí zaměřené na konkrétní oblast působení (např. správa Publishera). Samotný jazyk C neumí pracovat s objekty, kterým by funkce přiřadil. Jazyk Java je již objektově orientovaný jazyk, ve kterém jsou objektu přiřazeny vlastní datové položky a metody.

Následující kapitoly popisují balíčky a třídy objektů, které obsahují. Podrobný popis tříd lze nalézt ve vygenerované dokumentaci projektu ORTE:

- `orte-X.X.X/doc/jorte/index.html`

Balíček *org.ocera.orte*

Hlavní balíček obsahuje základní třídy potřebné pro vytvoření aplikace.

- ***Class Domain.*** Rodičovská třída tříd *DomainApp* a *DomainMgr*. Obsahuje základní metody pro manipulaci s doménou.
- ***Class DomainApp.*** Třída určená k vytvoření a správu aplikační domény. Obsahuje metody pro vytvoření *Publishera* a *Subcribera*.
- ***Class DomainMgr.*** Třída určená k vytvoření a správě domény managera.
- ***Class JOrte.*** Třída s pomocnými metodami. Inicializace ORTE vrstvy, uspání vlákna aplikace, kontrolní výpisy.
- ***Class OrteCallback.*** Abstraktní třída (*abstratct class* [5]) s jedinou metodou `callback()`.
- ***Class Publication.*** Třída pro aplikaci typu *Publisher*, obsahuje potřebné metody pro správu *Publishera*.
- ***Class Subscription.*** Třída pro aplikaci typu *Subscriber*, obsahuje potřebné metody pro správu *Subscribera*.
- ***Class SubscriptionCallback.*** Rodičovská třída pro callback objekt *Subscribera*. Obsahuje jedinou metodou `callback()`.

Balíček org.ocera.orte.examples

Balíček obsahuje další dva (v době psaní zprávy) balíky:

- `org.ocera.orte.examples.guiapps` – zatím prázdný balíček připravený pro demoaplikaci *Publishera* a *Subscribera* s grafickým rozhraním.
- `org.ocera.orte.examples.hello` – balíček s funkční demoaplikací *Publisher* a *Subscriber* pro příkazový řádek.

Balíček *org.ocera.orte.rtpseye*

Balíček aplikace *RTPSeYe* sloužící k monitorování ostatních aplikací užívajcích ke komunikaci RTPS protokolu. Aplikace disponuje grafickým rozhraním. Aplikaci je věnována samostatná kapitola 8.

Balíček org.ocera.orte.tools

Balíček s pomocnými nástroji a konstantami.

- ***Class Conversions.*** Třída obsahuje pouze 2 metody pro konverzi IP adresy z číselné podoby do tečkové notace a naopak. Třída bude v příští verzi nejpíš odstraněna, protože pro reprezentaci IP adresy bude použito třídy `java.net.Inet4Address`, která tyto konverzní metody obsahuje.

- ***Class ORTEConstant.*** Třída určená pouze pro konstanty. Obsahuje základní konstanty `liborte` a konstanty s velikostí datových typů Javy. Z důvodu změny základní konstanty není dobré mít její definici na dvou různých místech (v balíčku Javy a v knihovně `liborte`). Byla zvažována možnost získávání hodnot konstant přímo z knihovny `liborte`, ale toto bylo nakonec zavrhnuto z důvodu zbytečného zatěžování JVM. Velikosti datových typů Javy jsou připraveny pro IDL compiler [10], který definuje serializační a deserializační metody. Metody využívají pro práci s bufferem právě velikosti jednotlivých datových typů.

Balíček org.ocera.orte.types

Balíček obsahuje objekty používané jako argumenty metod ostatních tříd. Třídy tohoto balíčku jsou odvozeny od datových struktur knihovny `liborte`. Objekty slouží k uchovávání hodnot parametrů RTPS komunikace. Třída objektu se většinou skládá z datových položek a metody pro jejich získání (*get method*) a nastavení (*set method*). Někdy je přítomna i speciální metoda `toString()` [13] sloužící k definovanému výpisu hodnot datových proměnných objektu.

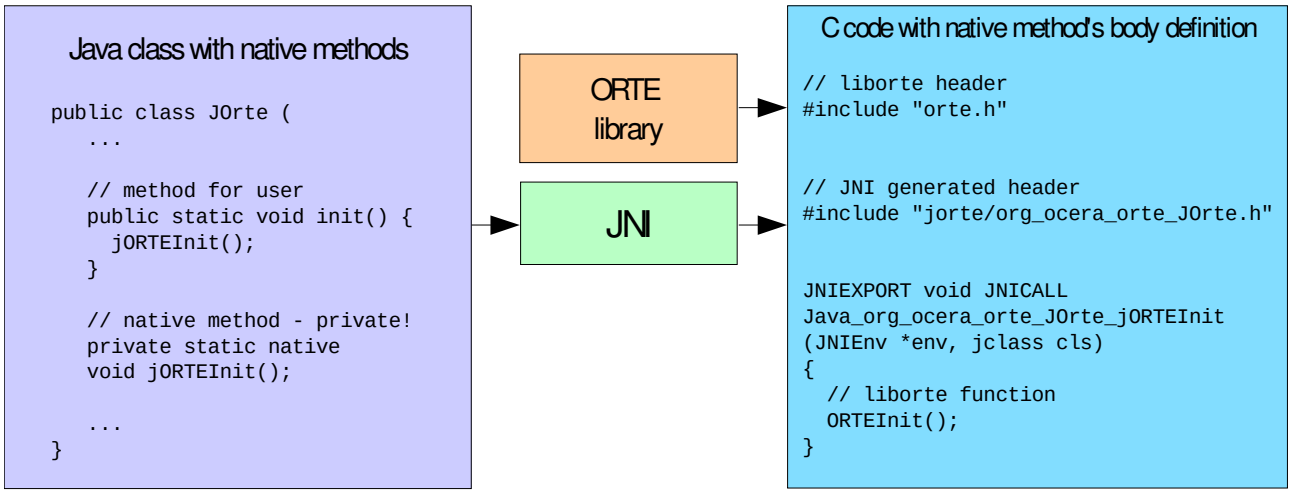
- ***Class AppInfo.*** Třída objektu obsahující datové položky se základními údaji o aplikaci. Objekt této třídy tvoří typ argumentu metod pro ošetření událostí (*event methods*).
- ***Class CommonPubSubInfo.*** Rodičovská třída tříd *PubInfo* a *SubInfo*. Obsahuje datové položky *topic* a *typeName* a metody pro jejich získání (*get methods*).
- ***Class DomainEvents.*** Rodičovská třída určená k dědění. Obsahuje prázdné metody k ošetření událostí (*event methods*). Uživatel musí ve své třídě překrýt původní (prázdné) metody pro ošetření událostí svými metodami.
- ***Class DomainProp.*** Datové položky třídy objektu uchovávají parametry domény.
- ***Class GUIDRtps.*** Třída objektu obsahující tři hlavní identfikátory RTPS aplikace: *hostId*, *appId* a *objectId* a příslušní *get/set metody*.
- ***Class MassageData.*** Rodičovská třída určená k dědění pro třídu, kterou si uživatel vygeneruje na základě svých požadavků pomocí IDL compileru. Rodičovská třída obsahuje pouze datovou položku `topic` a metody pro její získání a nastavení. Dále obsahuje metodu `getType()`, která jako jméno parametru `type` vrací automaticky jméno své třídy.
- ***Class NtpTime.*** Třída objektu substituující původní strukturu *NtpTime*. Objekt této třídy je používán jako typ argumentu metod.
- ***Class ProtocolVersion.*** Třída objektu sloužící k identifikaci verze RTPS protokolu.
- ***Class PubInfo.*** Třída objektu obsahující datové položky s údaji o *Publisherovi*. Objekt této třídy tvoří typ argumentu metod pro ošetření událostí (*event methods*).
- ***Class PublProp.*** Datové položky třídy objektu slouží k uchování parametrů *Publishera*.
- ***Class RecvInfo.*** Třída objektu obsahující datové položky s údaji o doručených datech. Objekt využívají hlavně *callback* metody.
- ***Class RecvStatus.*** Třída objektu definující přijímání dat (`NEED_DATA` nebo `DEADLINE`).
- ***Class SendInfo.*** Třída objektu obsahující datové položky s údaji o odeslaných datech.

Objekt využívají hlavně *callback* metody.

- ***Class SendStatus.*** Třída objektu definující typ odesílání dat (`NEED_DATA` nebo `CQL`).
- ***Class SequenceNumber.*** Třída objektu substituující původní strukturu *SequenceNumber*.
- ***Class Status.*** Třída objektu definující režim doručování zpráv (`BEST_EFFORT` nebo `STRICT`).
- ***Class SubInfo.*** Třída objektu obsahující datové položky s údaji o *Subscriberovi*. Objekt této třídy tvoří typ argumentu metod pro ošetření událostí (*event methods*).
- ***Class SubsProp.*** Datové položky třídy objektu slouží k uchování parametrů *Subscribera*.
- ***Class VendorId.*** Třída objektu sloužící k identifikaci dodavatele implementace RTPS protokolu.

7.2.3 Návrh metod Java API rozhraní

Pojmem *metoda Java API rozhraní* je myšlena taková metoda, která přes nativní metodu využívá ekvivalentní funkci z knihovny `liborte`. Pouze třídy hlavního balíčku `org.ocera.orte` a některé třídy balíčku `org.ocera.orte.tools` obsahují nativní metody. Na obrázku (Obr.23) je znázorněno propojení metody rozhraní v jazyce Java s funkcí z knihovny `liborte` (dále jen `liborte` funkce).



Obr. 23 Propojení metody Java API s původní liborte funkcí

Ve třídě objektu se nachází dvě metody:

- *metoda pro uživatele API rozhraní*,
- *nativní metoda*.

Uživateli API rozhraní není nativní metoda poskytnuta přímo, ale je zapouzdřena do další metody, která nativní metodu volá např. na obrázku metoda API rozhraní `init()` volá privátní (tj. pro uživatele API rozhraní nedostupnou) nativní metodu `jORTEInit()`. Toto je z důvodu bezpečnosti a praktičnosti. Může se stát, že nativní metoda způsobí vyjímku (*Exception*) [5], která by měla být ošetřena. Ošetření vyjímky a další případné operace mohou být pohodlně vloženy do těla metody API rozhraní.

7.2.4 Překlad balíčků

Překlad zdrojových kódů výše uvedených balíčků projektu ORTE provádí sestavovací nástroj ANT [14] (obdoba sestavovacího nástroje *make*). Pravidla překladu jednotlivých zdrojových kódů jsou uvedena v souboru `orte-X.X.X/orte/java/build.xml`. ANT neobsahuje Java překladač. Pro překlad balíčků je nutno mít nainstalované následující programy:

- *Java 2 Standart Edition JDK 1.4.2 (J2SE) a vyšší.*
- *Apache ANT.*

Systémové požadavky se odvíjí od požadavků J2SE JDK.

7.3 Knihovna *libjorte*

Zdrojové kódy v jazyce C slouží k vytvoření dynamické knihovny `libjorte`. Knihovna obsahuje jak původní API rozhraní v jazyce C, tak těla nativních funkcí. Při vytváření objektů z balíku `org.ocera.orte`, JVM provede načtení `libjorte` a propojení nativních metod s jejich těly.

7.3.1 Návrh nativních funkcí

Tělo nativní funkce, které je vytvořeno v jazyce C můžeme rozdělit na tři části:

- příprava argumentů pro funkci `liborte`,
- volaní funkce `liborte`,
- import navrácených hodnot do prostředí Javy.

Přípravou argumentů myslíme import příslušných datových položek, případně volání pomocných metod, z prostředí Javy. Tím získáme proměnné JNI datových typů (4.3.2). JNI datové typy přetypujeme na datové typy argumentů funkce `liborte`. Způsoby importu datových položek, volání metod z prostření Javy a základní JNI datové typy jsou uvedeny v (4.3.2).

Jakmile jsou připraveny všechny argumenty příslušné `liborte` funkce, dojde k jejímu zavolání. Návratová hodnota je po úspěšném vykonání funkce upravena a předána prostředí Java.

Při pojmenovávání nativních metod (ne metod Java API rozhraní) bylo jméno nativní metody odvozeno od jména `liborte` funkce, ke kterému bylo jako předpona přidáno písmeno “j” např. podle Obr. 23 se `liborte` funkce jmenuje `ORTEInit()` a příslušná nativní metoda `jORTEInit()`.

Zdrojový kód každé nativní funkce je uložen do vlastního souboru s názvem podle jména nativní funkce např. `jORTEInit.c`.

Zdrojové kódy nativní části Java API rozhraní jsou uloženy do adresářů:

- `orte-X.X.X/orte/libjorte/` – zdrojové kódy
- `orte-X.X.X/orte/include/jorte/` – vygenerované hlavičkové soubory

7.3.2 Překlad knihovny

Aby byla knihovna `libjorte` vytvořena při překladu projektu ORTE, je nutno změnit v konfiguračním souboru `orte-X.X.X/config.omk-default` hodnotu proměnné:

`CONFIG_OC_JAVA_ORTE=n` na `CONFIG_OC_JAVA_ORTE=y`.

7.4 Příklad použití ORTE Java API

Snahou při tvorbě API rozhraní pro jazyk Java bylo ulehčit práci programátorům zavedením rozhraní pro další programovací jazyk. Dalším cílem bylo zachovat jednoduchost použití rozhraní. Jednoduché aplikace lze opět vytvořit pomocí několika příkazů. Lze porovnat zdrojové kódy aplikací vytvořených pomocí API jazyka C (Obr.21) se zdrojovými kódy stejných aplikací vytvořených pomocí API jazyka Java. Barevně jsou na obrázcích vyznačeny stejné nebo podobné části obou aplikací, Publishera a Subscribera.

```
/* Publisher.java */

public class Publisher {

   // load native library 'libjorte.so'
   static { System.loadLibrary("jorte");}

  public static void main(String[] args) {

    // create events
    MyEvents events = new MyEvents();

    // create application domain
    DomainApp appDomain = new DomainApp(...);

    // create message instance
    HelloMsg hellomsg = new HelloMsg(...);

    // publisher properties
    PublProp publProp = new PublProp(...);

    // publisher parameters
    ...



    // create publication
    Publication pub =
        appDomain.createPublication(...);

    // send data
    pub.send(hellomsg);

    // kill & clean publisher – call GC
    pub = null;
    System.gc();
  }
}
```

```
/* Subscriber.java */

public class Subscriber {

   // load native library 'libjorte.so'
   static { System.loadLibrary("jorte");}

  public static void main(String[] args) {

    // create events
    MyEvents events = new MyEvents();

    // create application domain
    DomainApp appDomain = new DomainApp(...);

    // create message instance
    HelloMsg hellomsg = new HelloMsg(...);

    // subscription properties
    SubsProp subProps = new SubsProp(...);

    // subscriber's parameters
    ...

    // create callback instance
    MyCallback mycallback = new MyCallback();

    // create subscription
    Subscription sub =
         appDomain.createSubscription(...);



    // kill & clean subscriber - call GC
    sub = null;
    System.gc();
  }
}
```

Obr. 24 Příklad užití Java API rozhraní

Kapitola 8.
RTPSeYe

Kapitola je věnována popisu aplikace *RTPSeYe*. Aplikace je učena k objevování síťových aplikací používajících RTPS protokol a monitorování jejich parametrů. V následujících podkapitolách je popsán návrh a realizace aplikace.

8.1 Základní požadavky

Hlavním cílem bylo navrhnout a vytvořit jednouduchou aplikaci pro monitorování síťových aplikací využívajících RTPS protokolu.

Na aplikaci jsou kladeny tyto požadavky:

1. Aplikace bude uživateli poskytovat grafické rozhraní (GUI).
2. Při vytváření aplikace bude použito Java API rozhraní projektu ORTE a tím bude zároveň ověřena funkčnost rozhraní.
3. Síťové aplikace s RTPS protokolem budou automaticky vyhledávány a posléze monitorovány.
4. Objevené aplikace budou zobrazeny hierarchicky, podle identifikačních parametrů (*hostId*, *appId*).
5. Možnost zobrazit parametry vybrané aplikace.

8.2 Návrh aplikace

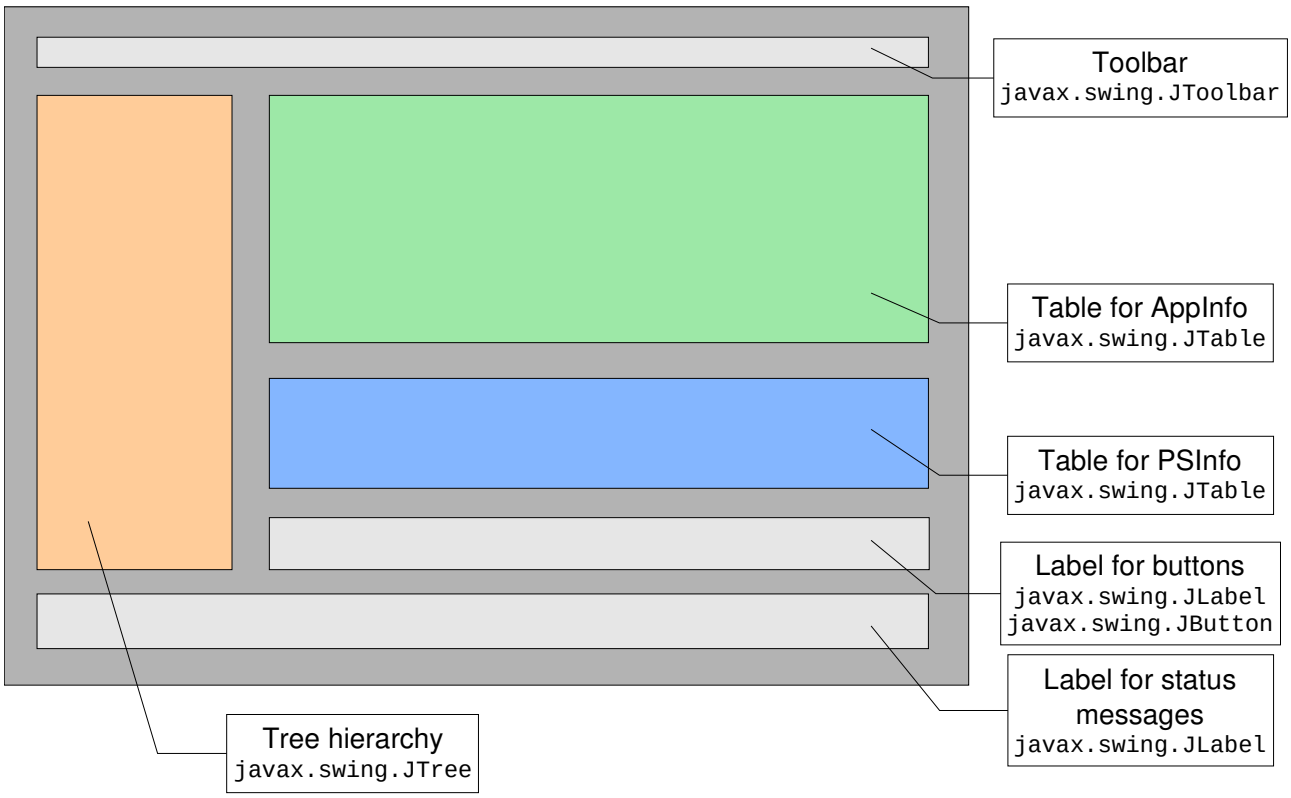
8.2.1 Návrh GUI

Java poskytuje dva základní balíky *Swing* a *AWT* (Příloha A, Obr. 30) obsahující grafické komponenty. K vytvoření grafického rozhraní byly vybrány komponenty balíku Swing:

- `javax.swing.JFrame` – Okno grafické aplikace.
- `javax.swing.JScrollPane` – Panel s posuvníky. Slouží jako podlad pro tabulku.
- `javax.swing.JPanel` – Panel bez posuvníků. Slouží jako podlad pro strom a tlačítka.
- `javax.swing.JSplitPane` – Oddělovač výše uvedených panelů.
- `javax.swing.JToolBar` – Panel nástrojů připravený pro menu (zatím nevyužit).

- `javax.swing.jLabel` – Objekt použitý pro textové výpisy o stavu aplikace.
- `javax.swing.jTree` – Strom, určen k hierarchickému zobrazování.
- `javax.swing.jTable` – Tabulka.
- `javax.swing.jButton` – Tlačítko.

Následující obrázek znázorňuje rozložení výše uvedených komponent.



Obr. 25 Návrh GUI applikace RTPSeYe

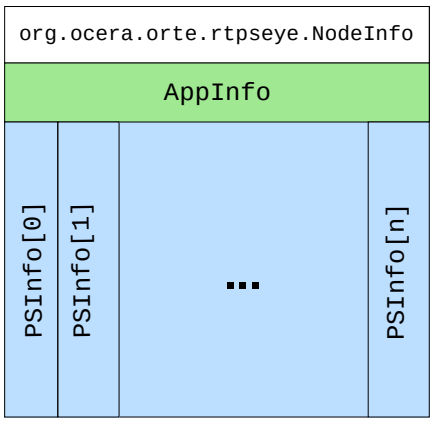
8.2.2 Návrh datových modelů

Manipulace s daty o aplikaci

Data určená k zobrazení jsou získána díky metodám třídy `org.ocera.orte.rtpseye.Events`. Při vyvolání události je zavolána příslušná event metoda a jsou naplněny datové položky objektu třídy *AppInfo*, případně *PubInfo* nebo *SubInfo*. Každé síťové aplikaci náleží vlastní objekt *AppInfo*. Síťová aplikace může obsahovat několik Publisherů a Subscriberů. Každý Publisher/Subscriber vlastní objekt třídy *PubInfo/SubInfo* se svými základními parametry.

Pro uchování všech údajů o aplikaci byla navrhnuta třída `org.ocera.orte.rtpseye.NodeInfo`. Obsahuje dvě datové položky (Obr. 26):

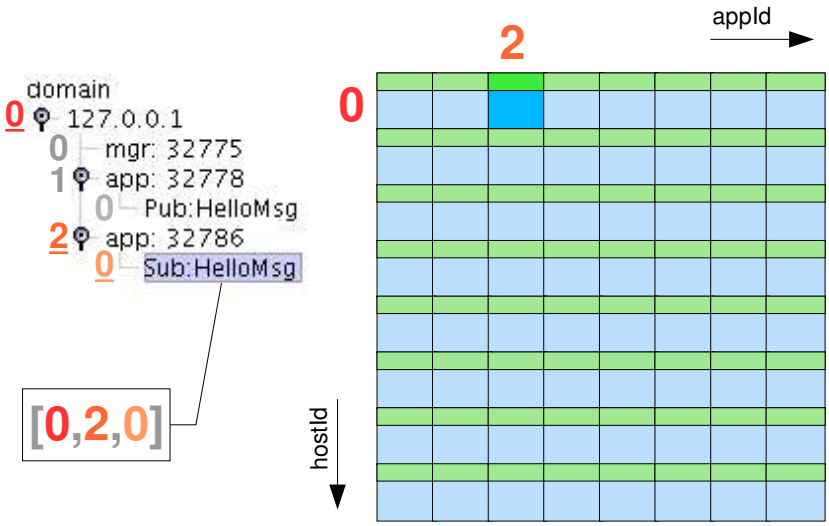
- `org.ocera.orte.types.AppInfo` – nese základní údaje o aplikaci,
- `org.ocera.orte.types.CommonPubSubInfo[]` – pole pro uložení příslušného počtu objektů typu `org.ocera.orte.types.PubInfo` a `org.ocera.orte.types.SubInfo`.



Obr. 26 Struktura třídy NodeInfo

Je třeba vyřešit problém, jak ukládat objekty třídy *NodeInfo*. Nalezené síťové aplikace jsou hierarchicky zobrazovány pomocí komponenty `javax.swing.JTree`. Pozice uzlu (*node*) v hierarchii se určuje vzdáleností od rodičovského uzlu. Celý RTPS strom se může zanořit maximálně do tří úrovní (až k uzlům Publisher/Subscriber). Během vnořování můžeme ukládat indexy jednotlivých rodičovských uzlů (Obr. 27). Indexy jsou ukládány pouze když je vybraný uzel typu *appId* nebo Pub/Sub uzel. Při vybrání kořenového uzlu (*root node*) nebo uzlu *hostId* není totiž zřejmé, které *AppInfo*, případně *Pub/SubInfo,* by se mělo zobrazit v tabulce.

Základní myšlenka uchovávání objektů třídy *NodeInfo* je vytvořit dvourozměrné dynamické pole, do jehož buněk se přistupuje za pomoci získaných indexů. První dva indexy slouží pro nalezení příslušné buňky tj. objektu třídy *NodeInfo*. Tím jsme získali přístup k datové položce *AppInfo* a existuje-li třetí index, z pole `CommonPubSubInfo[]` získáme *Pub/SubInfo*. V následující kapitole je vysvětleno, jak získaná data zobrazit.



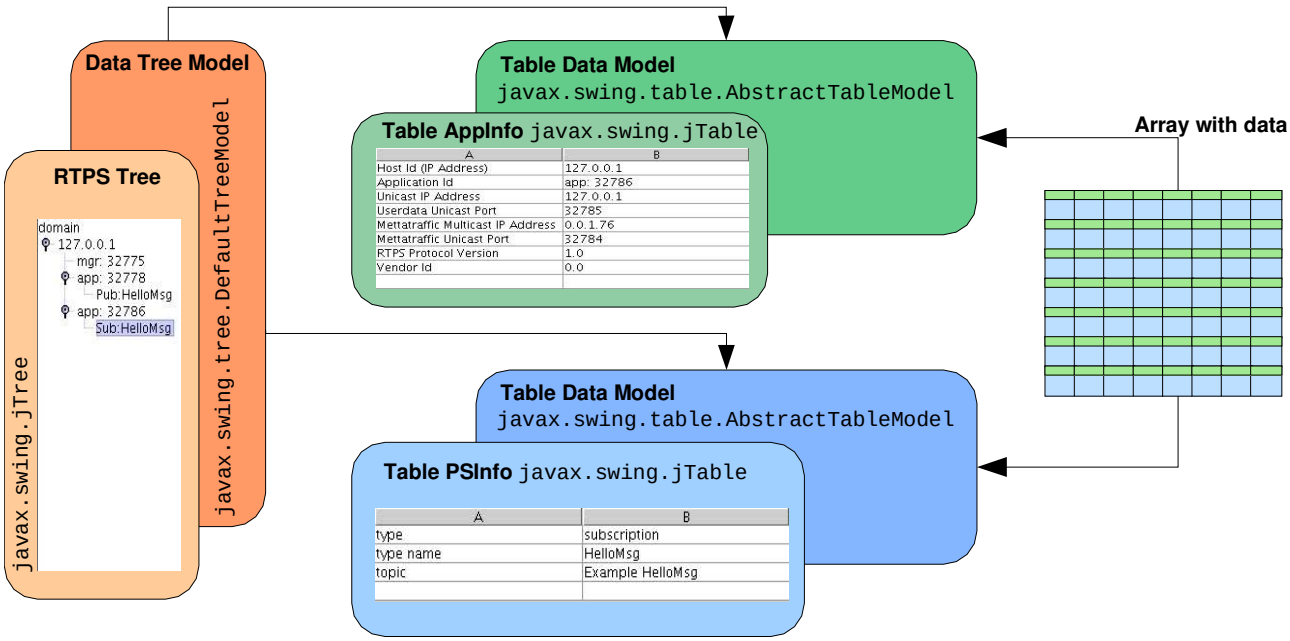
Obr. 27 Datový model pro uložení objektů třídy NodeInfo

Zobrazování pomocí datových modelů

Grafické komponenty `javax.swing.JTree` a `javax.swing.JTable` slouží k zobrazování dat. Komponenty však zbrazovaná data nedokáží sama uchovat, a proto ke své činnosti potřebují tzv. datové modely:

- `javax.swing.DefaultTreeModel`,
- `javax.swing.DefaultMutableTreeNode`,
- `javax.swing.AbstractTableModel`.

Datové modely jsou v pozadí grafické komponenty a starají se o udržování prvků obou komponent. Na obrázku je návrh datových modelů grafických komponent, provázání modelů, včetně pole s daty.



Obr. 28 Datové modely grafických komponent (JTree a JTable)

8.3 Balík *org.ocera.orte.rtpseye*

- ***Class AppInfoTableModel.*** Třída zajišťující datový model pro tabulku *AppInfo*. Třída je zděděná od třídy `javax.swing.table.AbstractTableModel`.
- ***Class Constant.*** Třída s konstantami určenými pro zdrojové kódy aplikace RTPSeYe.
- ***Class Events.*** Třída objektu definuje tzv. *event metody*, které obsahují kód pro obsloužení vzniklé události. V aplikaci RTPSeYe se event metodami ošetřuje přidávání a odebírání uzlů (aplikací) stromu, případná změna jejich paramatrů v tabulkách.
- ***Class ManageTree.*** Třída objektu, obsahuje pomocné metody pro manipulaci se stromem.
- ***Class NodeInfo.*** Třída jejíž datové položky (*AppInfo* a pole) slouží k uchovávání parametrů aplikací (Obr.).
- ***Class PubSubInfoTableModel.*** Třída zajišťující datový model pro tabulku *PSInfo*. Třída je zděděná od třídy `javax.swing.table.AbstractTableModel`.

- ***Class RtpsEye.*** Hlavní třída programu RTPSeYe. Poděděná od třídy `javax.swing.JFrame`. Obsahuje všechny grafické komponenty (8.2.1).
- ***Class RtpsTree.*** Třída objektu, obsahuje hlavní metody pro manipulaci s datovým modelem stromu.

8.4 Režimy aplikace *RTPSeYe*

Aplikace RTPSeYe pracuje ve čtyřech základních režimech:

- Start aplikace, hledání managera a registrace.
- Selhání registrace k managerovi.
- Úspěšná registrace k managerovi, normální provoz aplikace.
- Ukončení aplikace.

Po spuštění aplikace je nejprve inicializováno grafické rozhraní. Poté je vytvořena aplikační doména a aplikace se snaží registrovat k některému manageru (příloha A, Obr. 31).

V případě nenalezení manageru, se aplikace dostane do chybového stavu. V tomto stavu nedokáže vyhledávat a zobrazovat informace o síťových aplikacích užívajících RTPS protokol (příloha A, Obr. 32).

V případě nalezení manageru a registrace sama sebe, začne aplikace vyhledávat síťové aplikace užívající RTPS protokol. Nalezené aplikace jsou hierarchicky zobrazeny ve stromové struktuře a vybrané aplikaci jsou v tabulkách zobrazeny parametry (příloha A, Obr. 33).

Kliknutím na tlačítko `exit`. se aplikace RTPSeYe ukončí. Nejdříve se provede odhlášení aplikace od managera a poté je zrušena aplikační doména a grafické rozhraní.

8.5 Testování funkčnosti

Funkčnost aplikace RTPSeYe byla testována aplikacemi typu Publisher a Subscriber od více dodavatelů. Bylo testováno, zda je aplikace objevena a jsou korektně zobrazeny její parametry. Pro testy byly použity aplikace:

- **demoaplikace vytvořené pomocí Java API:**
  `org.ocera.orte.examples.hello.Publisher`
  `org.ocera.orte.examples.hello.Subscriber`
- **demoaplikace projektu ORTE (původní C API):**
  `orte-0.3.0/_compiled/bin-utils/h_publisher`
  `orte-0.3.0/_compiled/bin-utils/h_subscriber`
- **demoaplikace projektu NDDS:**
  `ndds.3.0i/bin/i86Linux2.4/nddsPing -p` (Publisher)
  `ndds.3.0i/bin/i86Linux2.4/nddsPing -s` (Subscriber)

Všechny síťové aplikace byly pomocí RTPSeYe nalezeny a zobrazeny jejich parametry. Tím jsme ověřili funkčnost RTPSeYe.

Kapitola 9.
Ethereal

Následující kapitola je věnována programu Ethereal, který slouží k monitorování síťové komunikace. Cílem práce bylo vyvinout a zaimplementovat do Etherealu podporu komunikačního protokolu RTPS.

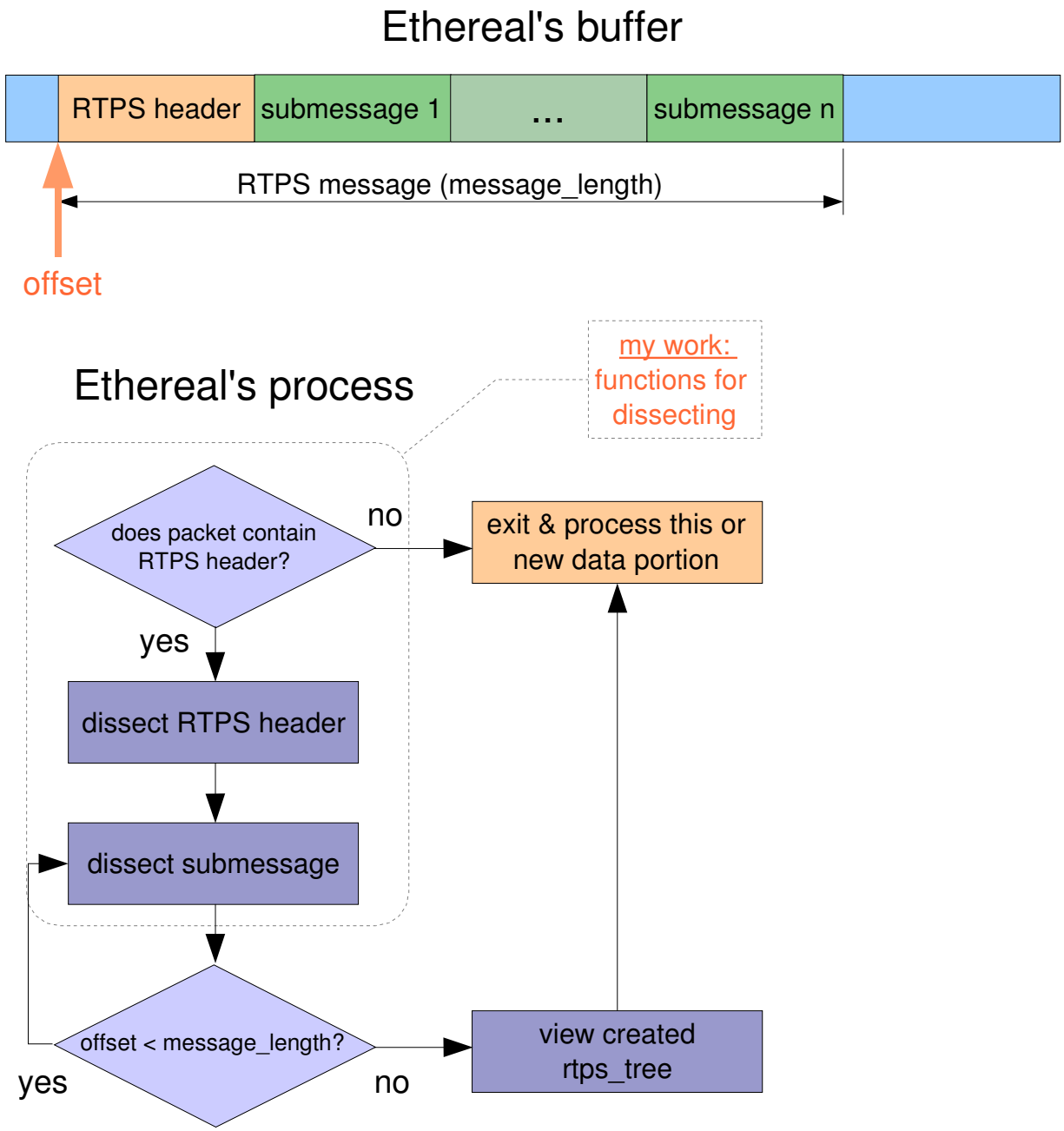
9.1 O programu

Autorem programu Ethereal [15] je pan Gerald Combs. Ethereal je program sloužící k monitorování síťové komunikace. Hlavní výhody programu jsou:

- ***Cena.*** Program je volně šiřitelný pod *GPL licencí* [16].
- ***Rozšiřitelnost.*** Program typu *open-source*, je možno upravovat zdrojové kódy a přidávat podporu nových protokolů.
- ***Multiplatformnost.*** Program lze nainstalovat na různých OS (Linux, UNIX, Windows, MacOS,...). Program je základní součástí některých Linuxových distribucí (Fedora, SuSE).
- ***Podpora protokolů.*** Aktuální verze (v době psaní zprávy 0.9.14) obsahuje podporu 683 protokolů.
- ***Režimy zachytávání paketů.*** Program umožňuje pracovat ve dvou základních režimech:
  - *Online.* Pakety zachytávány v reálném čase. Ethereal umí uložit do souboru zachycené pakety pro pozdější zpracování.
  - *Offline.* Dříve stažené pakety jsou uloženy do souboru a zpracovány později. Ethereal umí pracovat i se soubory jiných monitorovacích programů (MS Network Analyzer, Novell's LANanalyzer a další).
- ***Režimy zobrazování.*** S Etherealem můžeme pracovat ve dvou základních zobrazovacích režimech:
  - *GUI - Grafické rozhraní* (příloha A, Obr. 34).
  - *Režim pro příkazový řádek.*

9.2 Zpracování zachycených paketů

Na následujícím obrázku (Obr. 29) je zobrazen princip fungování Etherealu. Program má svůj buffer, do kterého ukládá zachycené pakety. Pomocí heuristik se snaží určit, kterému protokolu zprávy z paketů náleží. Po určení typu protokolu jsou data z bufferu rozebrána podle specifikace daného protokolu. Rozebraná data jsou vkládána do stromové struktury. Jakmile je zpráva celá zpracovaná, dojde k předání stromové struktury grafickému rozhraní, které zobrazí získané údaje.



Obr. 29 Zpracování dat v Etherealu

9.3 Tvorba podpory RTPS protokolu

Tvůrci programu Ethereal přesně vymezili pravidla návrhu podprory nového protokolu. Program Ethereal je vytvořen v programovacím jazyce C. Přidání podpory nového protokolu spočívá ve vytvoření funkcí, které zajistí správné rozebrání dat bufferu Etherealu. Funkce pro rozebírání dat jsou vytvořeny podle specifikace daného protokolu [3].

Nově vytvořené soubory se zdrojovým kódem jsou vloženy mezi zdrojové kódy Etherealu. Jména souborů je třeba připsat do seznamu zdrojových kódů pro sestavovací program *Make* [17]. Do souboru `/ethereal/epan/Makefile.common` přidáme do příslušných seznamů jména souborů `packet-rtps.c` a `packet-rtps.h`. Program poté znovu sestavíme příkazem `make`.

9.3.1 Soubor *packet-rtps.c*

Soubor obsahuje definice funkcí:

- funkce předdefinované tvůrci Etherealu,
- funkce pro rozebírání RPTS zpráv,
- ostatní pomocné funkce.

Předdefinované funkce

Funkce, které musí být přítomny ve zdrojovém kódu (nejsou použity beze změn, uživatel v nich provádí úpravy):

- `dissect_rtps()`. Hlavní funkce pro rozebírání dat. Provádí kontrolu hlavičky zprávy (*message header*) z paketu. V případě ověření a zpracování hlavičky dochází ke spuštění cyklu, ve kterém jsou rozebírány RTPS zprávy, souhrně označované jako *submessages*. Při zpracovávání jsou volány funkce pro rozebírání jednotlivých typů submessage zpráv, popřípadě další pomocné funkce.
- `proto_register_rtps()`. Funkce pro registraci protokolu do Etherealu.
- `proto_reg_handoff_rtps()`. Funkce pro registraci protokolu do Etherealu.

Funkce pro rozebírání RTPS zpráv

Funkce rozebere data bufferu podle specifikace [3] pro daný typ RTPS zprávy. Získané údaje jsou vkládány do hlavního stromu, který umožní zobrazení údajů uživateli.

- `dissect_PAD()`.
- `dissect_VAR()`.
- `dissect_ISSUE()`.
- `dissect_ACK()`.
- `dissect_HEARTBEAT()`.
- `dissect_GAP()`.
- `dissect_INFO_TS()`.

- `dissect_INFO_SRC()`.
- `dissect_INFO_REPLY()`.
- `dissect_INFO_DST()`.

Pomocné funkce

Pomocné funkce jsou používány výše uvedenými funkcemi. Slouží k předdefinovanému zpracování části dat. Jejich argumenty jsou ukazatel na buffer Etherealu a offset. Funkce získá z bufferu potřebná data, která vrací upravené. Pomocné funkce provádějí konverzi pořadí bytů tzv. *endianing* [9].

Funkce pro získání datového typu, případně bloků dat definované RTPS specifikací [3]:

- `get_guint16()`. Vrací 16 načtených bitů z bufferu.
- `get_guint32()`. Vrací 32 načtených bitů z bufferu.
- `get_NtpTime()`. Vrací datovou strukturu *NtpTime* s načtenými hodnotami.
- `get_bitmap()`. Provede načtení a zpracování datového bloku *Bitmap* [3].
- `get_parameter()`. Provede načtení a zpracování datového bloku *Parameter* [3].

Funkce pro převod hodnot z číselné reprezentce na znaky tzv. *to_string* funkce:

- `protocol_version_to_string()`. Verze protokolu.
- `vendor_id_to_string()`. Identifikace dodavetele RTPS implementace.
- `IP_to_string()`. Převod IP adresy z číselného formátu do tečkové notace.
- `port_to_string()`. Číslo portu.
- `host_id_to_string()`. Převod *HostId*.
- `app_id_to_string()`. Převod *AppId*.
- `object_id_to_string()`. Převod *ObjectId*.
- `seq_nr_to_string()`. Převod *SequenceNumber*.

9.3.2 Soubor *packet-rtps.h*

Hlavičkový soubor `packet-rtps.h` obsahuje definice konstant a vybraných datových typů přejatých z projektu ORTE. Části zdrojového kódu byly přejaty ze souborů:

- `orte-0.2.0/orte/include/orte/typedefs_api.h` – definice datových typů,ů
- `orte-0.2.0/orte/include/orte/typedefs_defines.h` – definice konstant.

9.4 Začlenění podpory RTPS do Etherealu

Vytvořené zdrojové kódy byly zaslány tvůrcům programu Ethereal, kteří zkontrolovali jejich funkčnost a provedli drobné úpravy. V květnu roku 2004 byla podpora RTPS protokolu zahrnuta do stabilní verze programu.

Závěr

Projekt ORTE prostřednictvím knihovny *liborte* nabízí API rozhraní (dále rozhraní) v jazyce C. Hlavním úkolem práce bylo navrhnout a vytvořit nové rozhraní v jazyce Java. Navrhnuté Java rozhraní bylo třeba napojit na stávající rozhraní jazyka C. K propojení obou rozhraní bylo použito technologie JNI, která umožňuje kooperaci obou programovacích jazyků. JNI umožňuje Javě používat tzv. nativní metody. Těla nativních metod nejsou definována v jazyce Java, ale jsou dodefinována pomocí tzv. nativního jazyka, v našem případě jazyk C.

Zdrojové kódy v jazyce C byly přidány k původní knihovně *liborte*. Rozšířením knihovny vznikla nová knihovna, která byla pojmenována *libjorte*. Java rozhraní bylo uloženo do balíčku *org.ocera.orte*. Snahou při řešení úkolu bylo vytvořit nové rozhraní plně ekvivalentní s původním rozhraním. V současné době je implementována většina funkcí rozhraní.

Nové rozhraní bylo otestováno v praxi vytvořením jednoduché aplikace pojmenované *RTPSeYe*. Aplikace slouží k monitorování síťových aplikací využívajích RTPS protokolu a zobrazování jejich parametrů. Pro realizaci aplikace bylo využito prvků grafického rozhraní, které poskytuje komponenta jazyka Java – *Swing*. Na vytvořené aplikaci byla ověřena funkčnost základních funkcí Java rozhraní. Z důvodu časové náročnosti řešení předchozích úkolů nezbyl čas na praktické ověření funkčnosti všech funkcí rozhraní. Ověření probíhalo pouze kontrolou výpisu atributů poskytovaných původním i novým rozhraním. Také nezbyl čas na vytvoření jednoduchých grafických aplikací typu Publisher a Subscriber.

Posledním řešeným úkolem bylo zabudovat do monitorovacího programu Ethereal podporu RTPS protokolu. Program pak umožní detailně sledovat RTPS komunikaci. Vytvoření podpory se sestávalo z návrhu a realizace funkcí, které podle normy RTPS protokolu rozeberou data bufferu aplikace. Rozebraná data pak převedou do zobrazitelné podoby a předají aplikaci k zobrazení.

Vytvořené zdrojové kódy byly zaslány tvůrcům Etherealu k otestování a schválení. Po několika dodatečných opravách byly zdrojové kódy schváleny a staly se součástí stabilní verze programu. Od června roku 2004 je program Ethereal distribuován s podporou RTPS protokolu.

Seznam použitých zdrojů

[1] *Cisco Network Academy* [online], 2005
http://cnap.comtel.cz/users/semestry/

[2] Zezulka F., *Požadavky DCS na real-time Ethernet* [conference Indetcon], May 18, 2005, Praha.

[3] RTI, *Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification v 1.17* [PDF], February 2002.

[4] Herout P., *Učebnice jazyk*a *C*, 3. vyd., Kopp České Budějovice, 2001.

[5] Spell B., *Java Programujeme profesionálně*, 1. vyd., Computer Press Praha, 2002, ISBN 80-7226-667-5.

[6] Liang S., *The Java Native Interface Programmer's Guide an Specific* [PDF], 1. vyd., Paolo Alto, 1999, ISBN 0-201-32577-2.

[7] Smolik P. a kol., *D 7.4_rep – Communication Components* [PDF], OCERA, February, 2004.

[8] *OCERA Project* [online], 2005
http://ocera.org/

[9] *Little Endian vs. Big Endian* [online], 2005
http://www.noveltheory.com/techpapers/endian.asp

[10] *OMG IDL* [online], 2005
http://www.omg.org/gettingstarted/omg_idl.htm

[11] *Using and Building Static and Shared Libraries* [online], 2005
http://users.actcom.co.il/~choo/lupg/tutorials/libraries/unix-c-libraries.html

[12] RTI, *NDDS Network Middleware for Distributed Real-Time Applications* [PDF], 2002.

[13] Eckel B., *Myslíme v jazyku Java*, 1. vyd., Grada Praha, 2001, ISBN 80-247-9010-6.

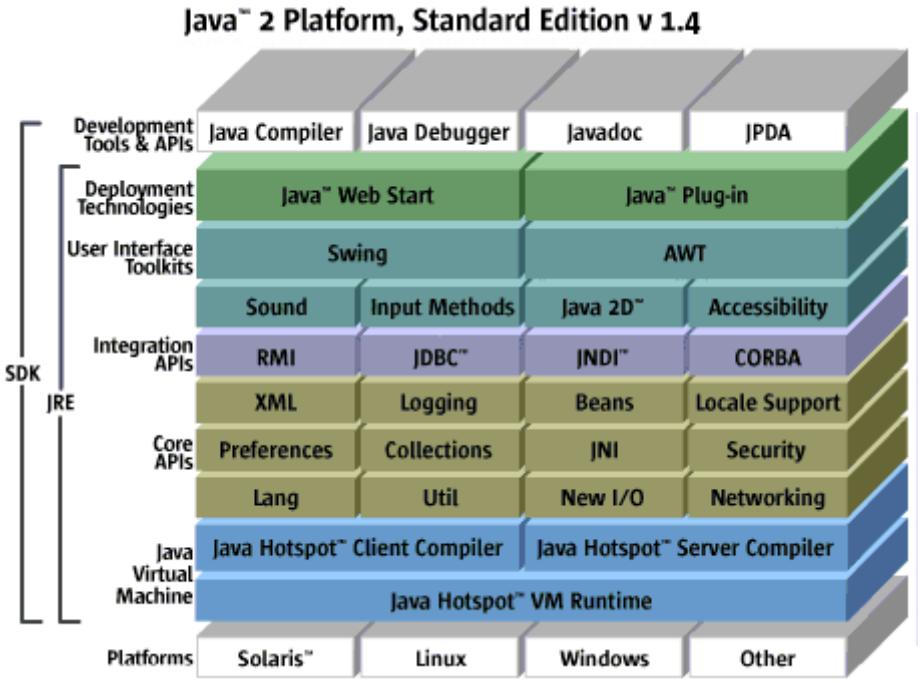
[14] *Apache Ant* [online], 2005
http://ant.apache.org/

[15] *Ethereal* [online], 2005
http://www.ethereal.com/

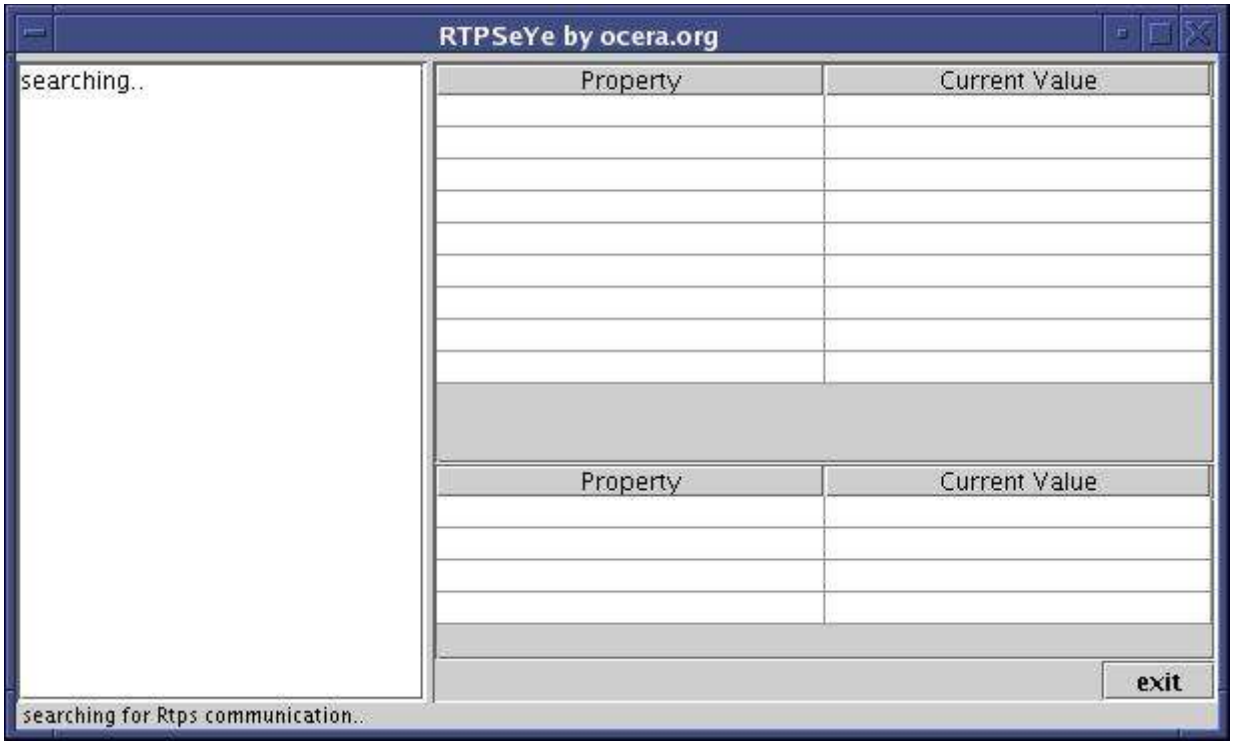
[16] *GNU GPL Licence* [online], 2005
http://www.gnu.org/copyleft/gpl.html

[17] *Make* [online], 2005
http://www.gnu.org/software/make/

A Obrázková příloha

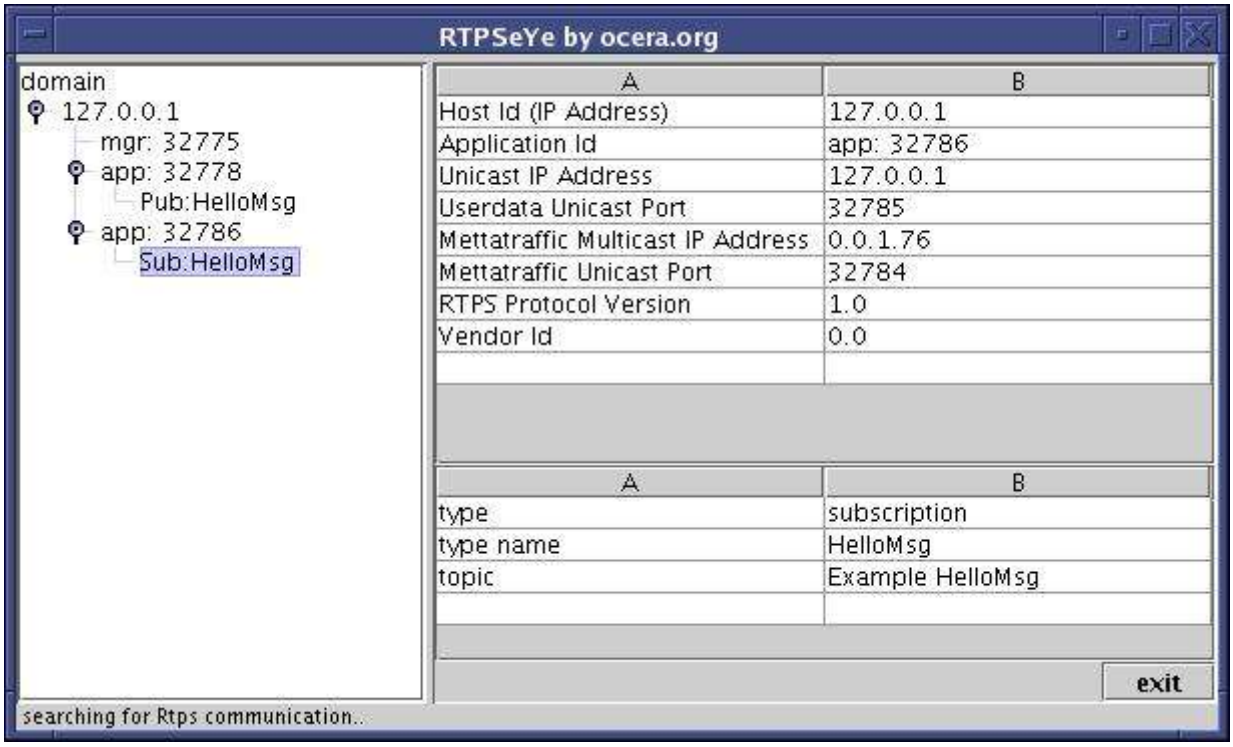


Obr.30 Hierarchie Java 2 Platform, Standart Edition v 1.4

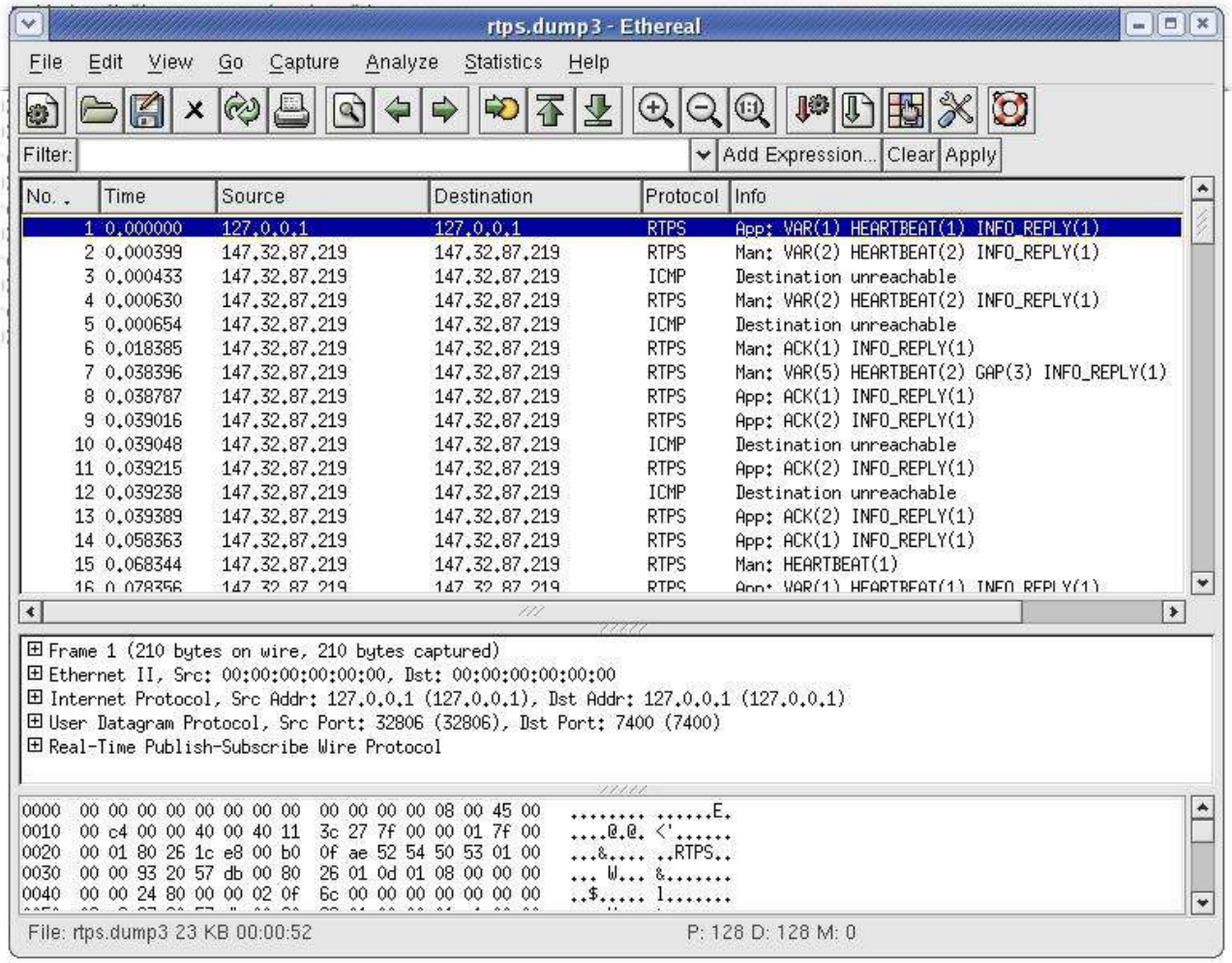


Obr. 31 Registrace RTPSeYe k RTPS Managerovi

Obr. 32 Selhání registrace RTPSeYe k RTPS Managerovi



Obr. 33 Pracovní režim RTPSeYe



Obr. 34 Ethereal zachytávající RTPS komunikaci

B Použité zkratky

ANT	Another Neat Tool
API	Application Program Interface aplikační programové rozhraní
CDR	Common Data Representation reprezentace základních datových typů
CSMA/CD	Carrier Sense Multiple Access with Collision Detection vícenásobný přístup s doposloucháváním nosného kmitočtu s detekcí kolizí
CST	CST – Composite State Protocol popis stavu síťového objektu
FTP	File Tranfer Protocol
GC	Garbage Collector
GNU	GNU is Not Unix
GPL	General Public Licence
GUI	Graphical Unit Interface grafické uživatelské rozhraní
HMI	Human Machine Interface
HTTP	Hyper Text Tranfer Protocol
HW	Hardware
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization mezinárodní organizace pro tvorbu standartů
J2SE	Java Second Platform, Standard Edition platforma jazyka Java
JNI	Java Native Interface nativní rozhraní jazyka Java
JVM	Java Virtual Machine virtuální stroj Java
LAN	Local Area Network

NDDS	Network Data Delivery Service
	komerční implementace RTPS protokolu
NIC	Network Interface Card
	síťová karta
OCERA	Open Components for Embedded Real-Time Applications
OOP	Object Oriented Programing
ORTE	OCERA Real-Time Ethernet
	volně šiřitelná implmentace RTPS protokolu
OS	Operating System
	operační systém
OSI model	Open System Interconnection reference model
PTP	Precision Time Protocol
	pomocný synchronizační protokol pro real-time komunikaci
RT	Real-Time
RTC	Real-Time Category
RTI	Real Time Inovation, Inc.
RTOS	Real Time Operation System
RTPS	Real-Time Publish-Subscribe protocol
	real-time komunikační protokol
SW	Software
TFPT	Tiny File Tranfer Protocol
UTP cable	Unschield Twisted Pair cable
UDP	User Datagram Protocol

C Adresářová struktura přiloženého CD

`/docs`

Offline materiály využité v této práci.

`/dp`

`/dp/pict`

Obrázky tohoto dokumentu ve formátu OpenOffice Drawing nebo JPEG.

`dp_2005_lukas_pokorny.pdf`

Tento dokument ve formátu PDF.

`/ethereal`

`/rtps-sources`

Adresář se zdrojovými kódy pro podporu RTPS protokolu.

Adresář pro zabalenou aktuální verzi programu Ethereal.

`/jni-example`

`/my/jni/test`

Adresář se zdrojovými kódy příkladu užití JNI.

`/orte`

Adresář pro zabalenou aktuální verzi projektu ORTE (ver. 0.3.0).