

**Technická univerzita v Košiciach**  
**Fakulta elektrotechniky a informatiky**  
**Katedra počítačov a informatiky**

**Aplikačný rámec pre sprostredkovanie IPFIX  
správ v nástroji SLAmeter**

Diplomová práca

**Príloha A**

**SYSTÉMOVÁ PRÍRUČKA Mediator v1.0**

Študijný program: Informatika  
Študijný odbor: Informatika  
Školiace pracovisko: Katedra počítačov a informatiky (KPI)  
Vedúci práce: Ing. Peter Fecilak, PhD.  
Konzultant: Ing. Adrián Pekár

**Košice 2013**

**Bc. Rastislav Kudla**

Copyright © 2013 Rastislav Kudla. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Text. A copy of the license can be found at <http://www.gnu.org/licenses/fdl.html>.

# Obsah

<b>1</b>	<b>Funkcia programu</b>	<b>6</b>
<b>2</b>	<b>Analýza problému</b>	<b>7</b>
2.1	Vybrané príklady použitia sprostredkovania správ . . . . .	7
2.2	Analýza aplikačného rámca pre IPFIX Mediátor . . . . .	8
<b>3</b>	<b>Popis programu</b>	<b>11</b>
3.1	Popis riešenia . . . . .	11
<b>4</b>	<b>Popis tried, členských premenných a metód</b>	<b>12</b>
4.1	Balík sk.tuke.cnl.bm.Mediator . . . . .	12
4.1.1	Trieda Default . . . . .	12
4.1.2	Trieda DropsCounter . . . . .	12
4.1.3	Trieda FlowRecordDispatcher . . . . .	13
4.1.4	Trieda IPLoader . . . . .	15
4.1.5	Trieda Mediator . . . . .	16
4.1.6	Trieda Support . . . . .	17
4.2	Balík sk.tuke.cnl.bm.Mediator.IPFIX . . . . .	19
4.2.1	Trieda IPFIXEncoder . . . . .	19
4.2.2	Trieda IPFIXFlowRecord . . . . .	24
4.2.3	Trieda IPFIXMessageHeader . . . . .	25
4.3	Balík sk.tuke.cnl.bm.Mediator.IntermediateProcesses . . . . .	25
4.3.1	Trieda AIntermediateProcess . . . . .	25
4.3.2	Trieda ExampleProcess . . . . .	29
4.3.3	Trieda IPInputBuffer . . . . .	30
4.4	Balik sk.tuke.cnl.bm.Mediator.collecting . . . . .	31
4.4.1	Trieda UDPServer . . . . .	32
4.4.2	Trieda IpfixedParser . . . . .	33
4.5	Balik sk.tuke.cnl.bm.Mediator.exporting . . . . .	34

---

4.5.1	Trieda ExportCache . . . . .	34
4.5.2	Trieda MessageEncoder . . . . .	35
4.5.3	Trieda UDPExporter . . . . .	38
4.6	Balik sk.tuke.cnl.bm.exceptions . . . . .	39
<b>5</b>	<b>Preklad programu</b>	<b>40</b>
5.1	Zoznam zdrojových textov . . . . .	40
5.2	Požiadavky na technické prostriedky pri preklade . . . . .	41
5.3	Požiadavky na programové prostriedky pri preklade . . . . .	41
5.4	Náväznosť na iné programové produkty . . . . .	41
5.5	Vlastný preklad . . . . .	42
5.6	Vytvorenie inštalačného DEB súboru . . . . .	42
5.7	Opis známych chýb . . . . .	43
<b>6</b>	<b>Zhodnotenie riešenia</b>	<b>44</b>
	<b>Zoznam použitej literatúry</b>	<b>45</b>

## Zoznam obrázkov

2–1 Referenčný model sprostredkovania správ v IPFIX . . . . .	9
2–2 Zjednodušený model komponentov IPFIX Mediátora . . . . .	10
4–1 Diagram tried rozhrania pre sprostredkovateľské procesy . . . . .	26
4–2 Diagram tried prvej fázy zhromažďovacieho procesu . . . . .	32
4–3 Diagram tried druhej fázy zhromažďovacieho procesu . . . . .	33
4–4 Diagram tried exportovacieho procesu . . . . .	34

# 1 Funkcia programu

Program Mediátor je implementáciou aplikačného rámca pre problém sprostredkovania správ v protokole IPFIX (*IP Flow Information Export (IPFIX) Mediation Problem*) vyvíjaný výskumnou skupinou MONICA sídliacou v Laboratóriu počítačových sietí (CNL) na Technickej Univerzite v Košiciach. Je súčasťou meracej architektúry SLAmeter, ktorej úlohou je pasívne meranie parametrov sieťovej prevádzky na báze tokov. Na základe nameraných hodnôt určuje triedu kvality služieb a Internetového pripojenia poskytovateľov Internetu. Trieda kvality vypovedá o dodržiavaní zmluvy o úrovni poskytovanej služby - *SLA*.

Komponentmi architektúry IPFIX (IP Flow Information Export) podľa RFC 5470 [3] sú exportéry a kolektory komunikujúce protokolom IPFIX. Vzhľadom k trvalému rastu IP prevádzky v heterogénnych sieťových prostrediach, tieto exportér-kolektor systémy môžu viesť k problémom škálovateľnosti. Navyiac, neposkytujú flexibilitu potrebnú pre široký rad meracích aplikácií.

Sprostredkovateľské moduly Mediátora môžu z pohľadu manipulácie s dátami poskytovať agregáciu, koreláciu, filtrovanie, anonymizáciu a iné úpravy záznamov o tokoch za účelom šetrenia výpočtových zdrojov meracieho systému a vykonávania predspracovania úloh pre kolektor. Z hľadiska interoperability nástrojov rôznych vývojárov, môžu poskytovať konverziu z iných protokolov na IPFIX, respektíve zvyšovať spoľahlivosť exportov napríklad prevodom z nespoľahlivého, bezspojoovo orientovaného protokolu UDP na spoľahlivý SCTP.

Program bol v roku 2013 vytvorený Rastislavom Kudlom v rámci jeho diplomovej práce.

## 2 Analýza problému

Problematika sprostredkovania IPFIX správ je podrobne spracovaná v [1]. Hovorí o tom, že sieťoví administrátori často celia problémom týkajúcim sa škálovateľnosti meracieho systému, flexibility monitorovania na základe tokov, alebo aj spoľahlivosti exportovania. Napriek tomu, že sa vyvinuli známe techniky ako *vzorkovanie a filtrovanie paketov*, *zokupovanie dátových záznamov*, alebo *replikácia exportu*, tieto problémy nevymizli. Pozostávajú z prispôsobovania niektorých parametrov meracích nástrojov zdrojom meracieho systému zatiaľ čo musia naplniť patričné podmienky ako sú *presnosť nameraných dát*, *granularita toku*, či *spoľahlivosť exportu*. Tieto okolnosti závisia na dvoch faktoroch:

1. **Kapacita meracieho systému** - pozostáva zo šírky pásma spravovanej siete, kapacity úložiska a výkonu exportovacích a zhromažďovacích nástrojov
2. **Požiadavky aplikácie** - rôzne aplikácie vyžadujú rôznu zrnitosť záznamov o tokoch a presnosť dát.

### 2.1 Vybrané príklady použitia sprostredkovania správ

RFC 5982 [1] uvádza viacero príkladov zaradenia IPFIX Mediátora do klasickej exportér - kolektor architektúry. Uvedme aspoň niektoré:

- prispôsobovanie granularity tokov,
- distribuovaná zhromažďovacia infraštruktúra,
- spájanie času,
- spájanie priestoru,
  - spájanie priestoru v rámci jednej pozorovacej domény,
  - spájanie priestoru viacerých pozorovacích domén jedného exportéra,

- spájanie priestoru niekoľkých exportérov,
- spájanie priestoru administratívnych domén,
- anonymizácia dátových záznamov,
- distribúcia dátových záznamov,
- konverzia z protokolu nižšej verzie na IPFIX,

## 2.2 Analýza aplikačného rámca pre IPFIX Mediátor

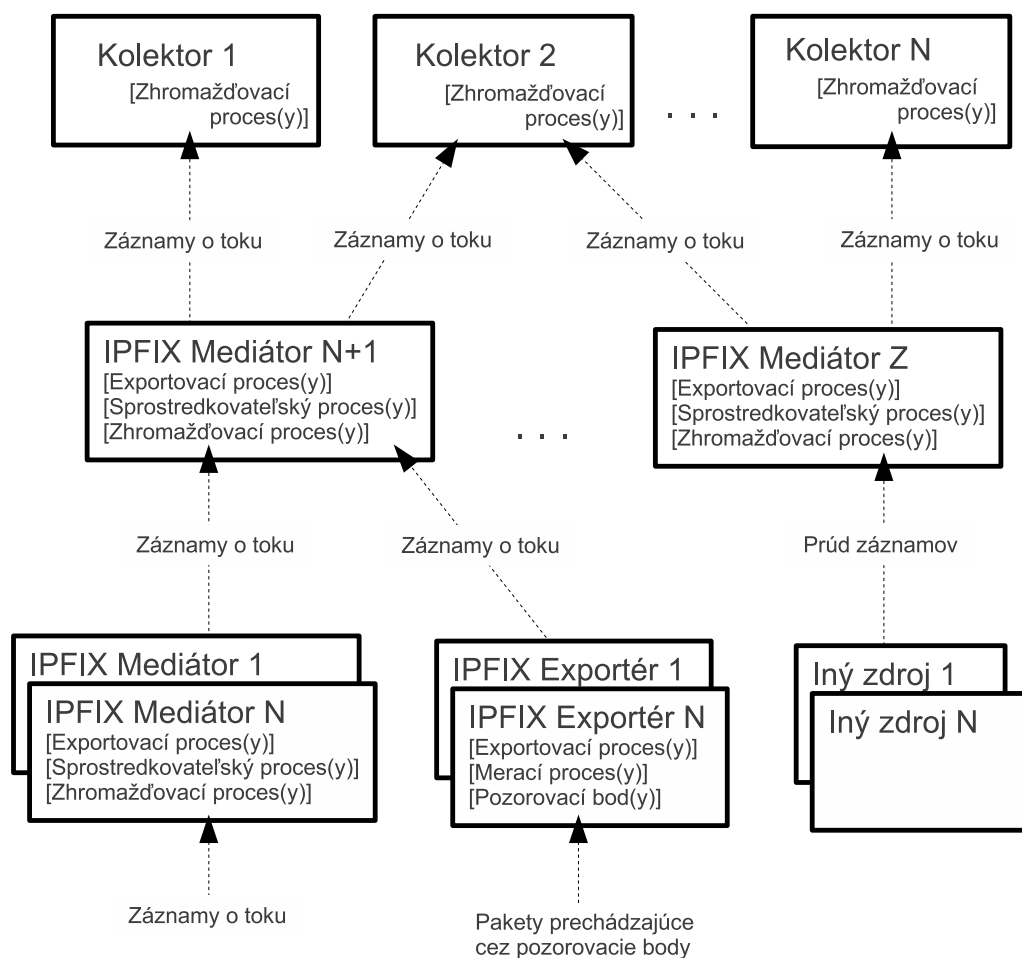
Analýze aplikačného rámca pre sprostredkovanie správ v IPFIX sa venuje RFC 6183 [2]. Na Obrázku 2–1 je zobrazený referenčný model sprostredkovania správ v IPFIX ako rozšírenie referenčného modelu IPFIX, popísaného v *Architecture for IP Flow Information Export* [3]. Táto schéma zobrazuje možné scenáre, ktoré môžu existovať v meracej architektúre.

Funkčné komponenty v rámci každej entity sú ohraničené zátvorkami []. Mediátor môže prijímať záznamy o toku od iných mediátorov a exportérov a prúd záznamov z iných zdrojov. Za iné zdroje sa považujú nástroje iných protokolov, ako napríklad NetFlow exportéry [4]. Spracovane dáta vo forme záznamov o toku potom exportuje jednému alebo viacerým kolektorom a mediátorom.

Zjednodušený model komponentov IPFIX mediátora je predstavuje Obrázok 2–2. Mediátor obsahuje jeden alebo viac sprostredkovateľských procesov, hierarchicky uložených medzi jedným alebo viacerými exportovacími a zhromažďovacími procesmi. Tento model sa týka najbežnejšieho prípadu, kedy mediátor prijíma dátové záznamy od exportéra, alebo iného mediátora.

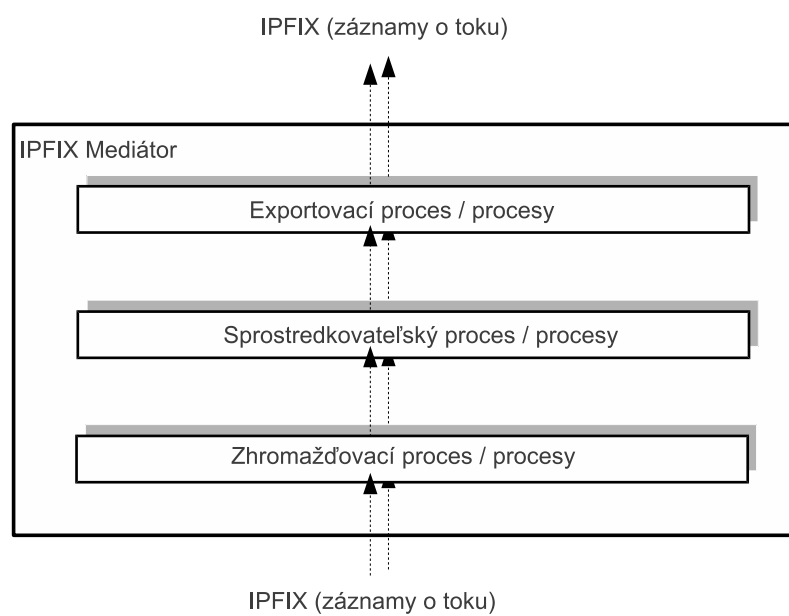
Sprostredkovateľské procesy sú kľúčovými funkčnými blokmi sprostredkovania správ v IPFIX. Musia pokryť každý príklad použitia sprostredkovania správ z kapitoly 2.1. Mediátor je schopný súčasne podporovať viac ako jeden sprostredkovateľský proces.





Obr. 2 – 1 Referenčný model sprostredkovania správ v IPFIX

- **Paralelné spracovanie** - Prúd záznamov je spracovaný viacerými sprostredkovateľskými procesmi paralelne. V tomto scenári, každý sprostredkovateľský proces dostáva kópiu celého prúdu záznamov ako vstup.
- **Sériové spracovanie** - Sprostredkovateľské procesy sú zapojené sériovo. Výstupný prúd záznamov jedného procesu je vstupným prúdom nasledujúceho.



**Obr. 2 – 2** Zjednodušený model komponentov IPFIX Mediátora

## 3 Popis programu

Jednotlivé časti programu sú umiestnené v nasledujúcich balíkoch:

- `sk.tuke.cnl.bm.Mediator.collecting` - implementácia zhromažďovacieho procesu
- `sk.tuke.cnl.bm.Mediator.exporting` - implementácia exportovacieho procesu
- `sk.tuke.cnl.bm.Mediator.IntermediateProcesses` - triedy tvoriace podporu pre sprostredkovateľské procesy. Hlavne triedy nových modulov musia byť umiestnené v tomto balíku!
- `sk.tuke.cnl.bm.Mediator.IPFIX` - triedy s manuálnou implementáciou protokolu IPFIX
- `sk.tuke.cnl.bm.Mediator.exceptions` - vlastné výnimky aplikácie
- `sk.tuke.cnl.bm.Mediator` - hlavné triedy samotného programu

### 3.1 Popis riešenia

## 4 Popis tried, členských premenných a metód

Keďže niektoré triedy Mediátora sú kvôli jednotnosti riešení v rámci výskumnej skupiny MONICA totožné s triedami nástroja JXColl, v nasledujúcich častiach budú uvedené len tie, ktoré sa týkajú výhradne Mediátora. Popis ostatných tried a metód je uvedený v systémovej príručke programu JXColl [5].

### 4.1 Balík `sk.tuke.cnl.bm.Mediator`

#### 4.1.1 Trieda `Default`

Trieda predstavuje rozhranie obsahujúce východiskové hodnoty konfiguračného súboru. Neobsahuje konštruktor ani žiadne metódy, iba verejne prístupné statické konštanty.

#### 4.1.2 Trieda `DropsCounter`

Slúži na výpočet štatistiky zahodených entít. Pod entitou sa myslia záznamy o tokoch, dátové záznamy, alebo IPFIX pakety. Obsahuje len statické metódy.

#### Metódy

*public static void **inputBufferDropsUP()***

Zvyšuje počet strát spôsobených preplnením vstupného bufferu sprostredkovateľských procesov o jeden.

#### Parametre:

*String* processName - meno procesu

*public static void **exportCacheDropsUP()***

Zvyšuje počet strát spôsobených `ExportCache` o jeden.

*public static void **encodingDropsUp()***

Zvyšuje počet strát spôsobených chybou pri kódovaní o jeden.

*public static void **decodingDropsUp()***

Zvyšuje počet strát spôsobených chybou pri dekódovaní o jeden.

*public static void **packetDropsUp()***

Zvyšuje počet IPFIX paketov zahodených UDP serverom o jeden.

*public static void **printStats()***

Vypíše štatistiku všetkých zahodených entít.

#### 4.1.3 Trieda `FlowRecordDispatcher`

Úlohou tejto triedy je na základe konfiguračného súboru distribuovať prijaté záznamy o toku sprostredkovateľským procesom (sériovo alebo paralelne) a exportovaciemu procesu. Distribúcia prebieha v súlade s IPFIX Mediator Framework (RFC 6183). Táto trieda je implementovaná podľa návrhového vzoru *Singleton*.

##### Metódy

*public static `FlowRecordDispatcher` **getInstance()***

Implementuje vzor *Singleton*. Vytvorí a vráti jedinečnú inštanciu v prípade že neexistuje, v opačnom prípade ju iba vráti.

**Návratová hodnota:**

Jedinečný objekt typu `FlowRecordDispatcher`.

*public synchronized void **dispatchFlowRecord**(IPFIXFlowRecord flowRecord, String inputProcess)*

Posiela prijaté záznamy o tokoch príslušným sprostredkovateľským procesom, alebo exportovaciemu procesu podľa konfigurácie. Najprv získa zoznam prijímateľov toku na základe mena pôvodcu. Ak je zoznam prázdny - záznam o toku je určený na export, preto ho zapíše do `ExportCache`. Ak zoznam nie je prázdny, získa si inštancie prijímateľov toku a záznam im zapíše do vstupného buffera. Táto metóda je synchronizovaná, lebo je prístupná viacerým vláknam.

**Parametre:**

*IPFIXFlowRecord flowRecord* - záznam o toku, ktorý sa má distribuovať ďalej

*String inputProcess* - meno pôvodcu záznamu o toku.

*private void **fillInputBuffer**(AIntermediateProcess process, IPFIXFlowRecord flowRecord)*

Metóda, ktorá zapisuje záznamy o tokoch do vstupného buffera sprostredkovateľských procesov. V prípade neúspechu sa zvýši počítadlo v `DropsCounter` a vypíše error.

**Parametre:**

*AIntermediateProcess process* - inštancia sprostredkovateľského procesu.

*IPFIXFlowRecord flowRecord* - záznam o toku, ktorý sa má zapísať.

*private ArrayList<String> **getReceiversList**(String inputDevice)*

Získava zoznam príjemcov záznamu o toku od zadaného pôvodcu toku.

**Parametre:**

*String inputDevice* - pôvodca záznamu o toku

**Návratová hodnota:**

Zoznam príjemcov toku, typ `ArrayList`.

**4.1.4 Trieda IPLoader**

Trieda je zodpovedná za dynamické načítavanie sprostredkovateľských procesov definovaných v konfiguračnom súbore. Implementuje návrhový vzor *Singleton*.

**Metódy**

*public static IPLoader **getInstance()***

Implementuje vzor *Singleton*. Vytvorí a vráti jedinečnú inštanciu v prípade že neexistuje, v opačnom prípade ju iba vráti.

*public void **loadProcesses()***

Hlavná metóda triedy, dynamicky načítava sprostredkovateľské moduly definované v konfiguračnom súbore. Najprv získa systémový *ClassLoader*. V cykle prechádza zoznam sprostredkovateľských modulov. Každý reťazec obsahujúci meno prevedie na binárne meno (meno triedy vrátane balíčkov) a pomocou *ClassLoader*-a získa jeho `Class` objekt. Na základe tohto objektu získa jedinečnú inštanciu modulu a keďže sa jedná o vlákno, spustí ho tak, že zavolá jeho metódu `start()`.

**Hádže:**

`IPLoaderException` - V prípade akejkoľvek chyby, ktorá môže nastať pri vykonávaní metódy. Chyby, ktoré sú zachytávané sú typov:

- `SecurityException`
- `ClassNotFoundException`
- `IllegalAccessException`
- `NoSuchMethodException`

- `InvocationTargetException`

#### 4.1.5 Trieda Mediator

Úlohou hlavnej triedy Mediátora je postupne spustiť všetky vlákna a procesy potrebné pre beh programu. Najprv sa prečítajú a spracujú argumenty príkazového riadku. Program vie rozpoznávať dva druhy argumentov. Prvým je cesta ku konfiguračnému súboru. Ak nie je zadaná, používa sa východiskový konfiguračný súbor. Druhým argumentom môže byť zadaná možnosť `--logtofile`. Vtedy sú všetky logovacie výstupy presmerované zo štandardného výstupu do súboru.

Potom ako program načíta všetky nastavenia z konfiguračného súboru, spustí všetky svoje moduly - sprostredkovateľské procesy pomocou triedy `IPLoader`. Nasleduje spustenie vlákna, ktoré prijíma IPFIX pakety prostredníctvom protokolu UDP a vlákna, ktoré ich spracováva. Hovoríme o `UDPServer` a `UDPProcessor`. Nakoniec je spustené exportovacie vlákno - `UDPExporter`. Kedykoľvek keď nastane chyba je Mediátor korektne ukončený a to tak, že uvoľní všetku pamäť a zastaví bežiacie vlákna. Rovnako je Mediátor zastavený po stlačení kombinácie kláves `Ctrl+c`.

#### Metódy

*public static void **main**(String[] args)*

Hlavná metóda triedy.

##### **Parametre:**

*String[]* args - argumenty príkazového riadku.

*public static void **stopMediator**()*

Metóda, ktorá korektne ukončuje beh programu. Zastaví všetky spustené vlákna a uvoľní všetky druhy pamäte.



*public static void **interruptThread()***

Preruší vykonávanie vlákna.

**Parametre:**

*Thread* thread - objekt vlákna, ktoré sa má zastaviť.

*private static void **loggingToFile()***

Metóda, ktorá vykonáva logovanie do súboru namiesto štandardného výstupu.

#### 4.1.6 Trieda Support

Podporná trieda, ktorá obsahuje pomocné metódy potrebné pri de(kódovaní) a pri validácii formátu dát. Uvádzam iba vlastné metódy.

##### Metódy

*public static byte[] **byteToByteArray**(byte x)*

Konvertuje primitívny typ *byte* na pole bytov v usporiadaní bytov Big Endian.

**Parametre:**

*byte* x - hodnota, ktorá sa má zakódovať.

**Návratová hodnota:**

Pole bytov, typ *byte[]*.

*public static byte[] **shortToByteArray**(short x)*

Konvertuje primitívny typ *short* na pole bytov v usporiadaní bytov Big Endian.

**Parametre:**

*short* x - hodnota, ktorá sa má zakódovať.

**Návratová hodnota:**

Pole bytov, typ *byte[]*.

*public static byte[] **intToByteArray**(int x)*

Konvertuje primitívny typ *int* na pole bytov v usporiadaní bytov Big Endian.

**Parametre:**

*int* x - hodnota, ktorá sa má zakódovať.

**Návratová hodnota:**

Pole bytov, typ *byte[]*.

*public static byte[] **longToByteArray**(long x)*

Konvertuje primitívny typ *long* na pole bytov v usporiadaní bytov Big Endian.

**Parametre:**

*long* x - hodnota, ktorá sa má zakódovať.

**Návratová hodnota:**

Pole bytov, typ *byte[]*.

*public static byte[] **floatToByteArray**(float x)*

Konvertuje primitívny typ *float* na pole bytov v usporiadaní bytov Big Endian.

**Parametre:**

*float* x - hodnota, ktorá sa má zakódovať.

**Návratová hodnota:**

Pole bytov, typ *byte[]*.

*public static byte[] **doubleToByteArray**(double x)*

Konvertuje primitívny typ *double* na pole bytov v usporiadaní bytov Big Endian.

**Parametre:**

*double* x - hodnota, ktorá sa má zakódovať.

**Návratová hodnota:**

Pole bytov, typ *byte[]*.

```
public static boolean validateMAC(String macAddress)
```

Validuje formát MAC adresy.

**Parametre:**

*String* macAddress - adresa, ktorej formát sa má overiť.

**Návratová hodnota:**

*true* - v prípade, že je adresa v správnom formáte.

*false* - opačne.

## 4.2 Balík *sk.tuke.cnl.bm.Mediator.IPFIX*

### 4.2.1 Trieda *IPFIXEncoder*

Trieda so statickými metódami slúžiacimi na zakódovanie reťazcovej reprezentácie hodnôt informačných elementov na abstraktne dátové typy podľa RFC 5101 a RFC 5102. Je presným opakom triedy *IPFIXDecoder*.

**Metódy**

```
public static byte[] encode(String dataType, String value)
```

Zakóduje hodnotu dátového typu do poľa bytov podľa špecifikácie IPFIX. Priamo nevykonáva zakódovanie, volá konkrétne metódy podľa kategórie dátového typu.

**Parametre:**

*String* dataType - reťazec definujúci dátový typ obsiahnutý v bufferi.

*String* value - samotné dáta, ktoré sú predmetom zakódovania

**Návratová hodnota:**

Pole bytov reprezentujúce interpretovanú reťazcovú hodnotu na základe predaného typu.

**Hádže:**

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`OutOfBoundsException` - Ak je hodnota mimo povoleného rozsahu

`UnknownHostException` - Ak program nevie rozpoznať host, alebo IP adresu

`DataException` - Ak je chyba v formáte kódovanej hodnoty vzhľadom na dátový typ

*public static byte[] **encodeUnsignedIntegralType**(String dataType, String value)*

Zakóduje celočíselné bezznamiekové dátové typy `unsigned8`, `unsigned16`, `unsigned32`, `unsigned64` a `unsigned128`.

**Parametre:**

*String* dataType - reťazec definujúci dátový typ obsiahnutý v bufferi.

*String* value - samotné dáta, ktoré sú predmetom zakódovania

**Návratová hodnota:**

Pole bytov reprezentujúce interpretovanú reťazcovú hodnotu na základe predaného typu.

**Hádže:**

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formáte kódovanej hodnoty vzhľadom na dátový typ

`OutOfBoundsException` - Ak je hodnota mimo povoleného rozsahu

*public static byte[] **encodeSignedIntegralType**(String dataType, String value)*

Zakóduje celočíselné znamienkové dátové typy `signed8`, `signed16`, `signed32` a `signed64`.

**Parametre:**

*String* dataType - reťazec definujúci dátový typ obsiahnutý v bufferi.

*String* value - samotné dáta, ktoré sú predmetom zakódovania

**Návratová hodnota:**

Pole bytov reprezentujúce interpretovanú reťazcovú hodnotu na základe predaného typu.

**Hádže:**

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formáte kódovanej hodnoty vzhľadom na dátový typ

*public static byte[]* **encodeFloatType**(*String* dataType, *String* value)

Zakóduje desatinné dátové typy float32 a float64.

**Parametre:**

*String* dataType - reťazec definujúci dátový typ obsiahnutý v bufferi.

*String* value - samotné dáta, ktoré sú predmetom zakódovania

**Návratová hodnota:**

Pole bytov reprezentujúce interpretovanú reťazcovú hodnotu na základe predaného typu.

**Hádže:**

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formáte kódovanej hodnoty vzhľadom na dátový typ

*public static byte[]* **encodeAddressType**(*String* dataType, *String* value)

Zakóduje dátové typy obsahujúce adresy: ipv4Address, ipv6Address a macAddress.

**Parametre:**

*String* dataType - reťazec definujúci dátový typ obsiahnutý v bufferi.

*String* value - samotné dáta, ktoré sú predmetom zakódovania

**Návratová hodnota:**

Pole bytov reprezentujúce interpretovanú reťazcovú hodnotu na základe predaného typu.

**Hádže:**

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`UnknownHostException` - Ak program nevie rozpoznať host, alebo IP adresu

`DataException` - Ak je chyba v formáte kódovanej hodnoty vzhľadom na dátový typ

```
public static byte[] encodeBooleanType(String value)
```

Zakóduje boolean reprezentujúci pravdivostnú hodnotu.

**Návratová hodnota:**

Pole bytov reprezentujúci pravdivostnú hodnotu, "true" alebo "false".

**Hádže:**

`OutOfBoundsException` - Ak je hodnota mimo povoleného rozsahu

```
public static byte[] encodeStringType(String value)
```

Zakóduje reťazec v kódovaní UTF-8 do pole bytov.

**Návratová hodnota:**

Pole bytov reťazca v kódovaní UTF-8.

```
public static byte[] encodeOctetArrayType(String value)
```

Reťazec v kódu Base64 prevedie na pole bytov.

**Návratová hodnota:**

Pole bytov predstavujúce binárne dáta zakódované v Base64.

```
public static byte[] encodeDateTimeType(String dataType, String value)
```

Zakóduje dátové typy časových známkok: `dateTimeSeconds`, `dateTimeMilliseconds`, `dateTimeMicroseconds` a `dateTimeNanoseconds` do poľa bytov.

**Parametre:**

*String* `dataType` - reťazec definujúci dátový typ obsiahnutý v bufferi.

*String* `value` - samotné dáta, ktoré sú predmetom zakódovania

**Návratová hodnota:**

Pole bytov reprezentujúce interpretovanú hodnotu reťazca na základe predaného typu. Dátové typy `dateTimeSeconds` a `dateTimeMilliseconds` predstavujú počet sekúnd, resp. milisekúnd od Unix epochy (00:00 1.1.1970 UTC). Dátové typy `dateTimeMicroseconds` a `dateTimeNanoseconds` sú zakódované vo formáte časovej známky NTP Timestamp.

**Hádže:**

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formáte kódovanej hodnoty vzhľadom na dátový typ

`OutOfBoundsException` - Ak je hodnota mimo povoleného rozsahu

```
public static void checkStringNumbersRange(String min, String max, String value, String dataType)
```

Overí, či číselná hodnota v reťazci spadá do rozsahu daného dátovým typom.

**Parametre:**

*String* `min` - dolná hranica rozsahu

*String* `max` - horná hranica rozsahu

*String* `value` - hodnota, ktorá sa má overiť

*String* `dataType` - dátový typ hodnoty

**Hádže:**

`OutOfBoundsException` - Ak je hodnota mimo povoleného rozsahu

### 4.2.2 Trieda **IPFIXFlowRecord**

Táto trieda je reprezentáciou IPFIX Flow record-u, teda záznamu o toku.

#### **Konštruktor**

```
public IPFIXFlowRecord(IPFIXTemplateRecord referencedTemplate, ArrayList  
<IPFIXDataRecord> dataRecords, IPFIXMessage.IPFIXMessageHeader message-  
Header)
```

Konštruktor prostredníctvom predaných parametrov inicializuje členské premenné.

#### **Parametre:**

*IPFIXTemplateRecord* referencedTemplate - šablóna patriaca dátovým záznamom

*ArrayList<IPFIXDataRecord>* dataRecords - pole dátových záznamov

*IPFIXMessage.IPFIXMessageHeader* messageHeader - hlavička IPFIX správy, ktorá obsahovala tento záznam o toku

```
public IPFIXFlowRecord()
```

Bezparametrický konštruktor. Iba inicializuje prázdne pole dátových záznamov.

#### **Metódy:**

Metódy, ktoré tu nie sú spomenuté, sú klasické gettery a settery.

```
public int getReferencedTemplateID()
```

#### **Návratová hodnota:**

Vracia ID šablóny z IPFIX správy, ktorá obsahovala tento záznam o toku.



```
public void addDataRecord(IPFIXDataRecord dataRecord)
```

Pridá dátový záznam do poľa dátových záznamom flow record-u.

**Parametre:**

*IPFIXDataRecord dataRecord* - dátový záznam, ktorý sa priradí záznamu o toku.

#### 4.2.3 Trieda **IPFIXMessageHeader**

Trieda reprezentujúca hlavičku IPFIX správy. Implementuje návrhový vzor Singleton.

**Metody:**

Metódy, ktoré tu nie sú spomenuté, sú klasické gettery a settery.

```
public static IPFIXMessageHeader getInstance()
```

**Návratová hodnota:**

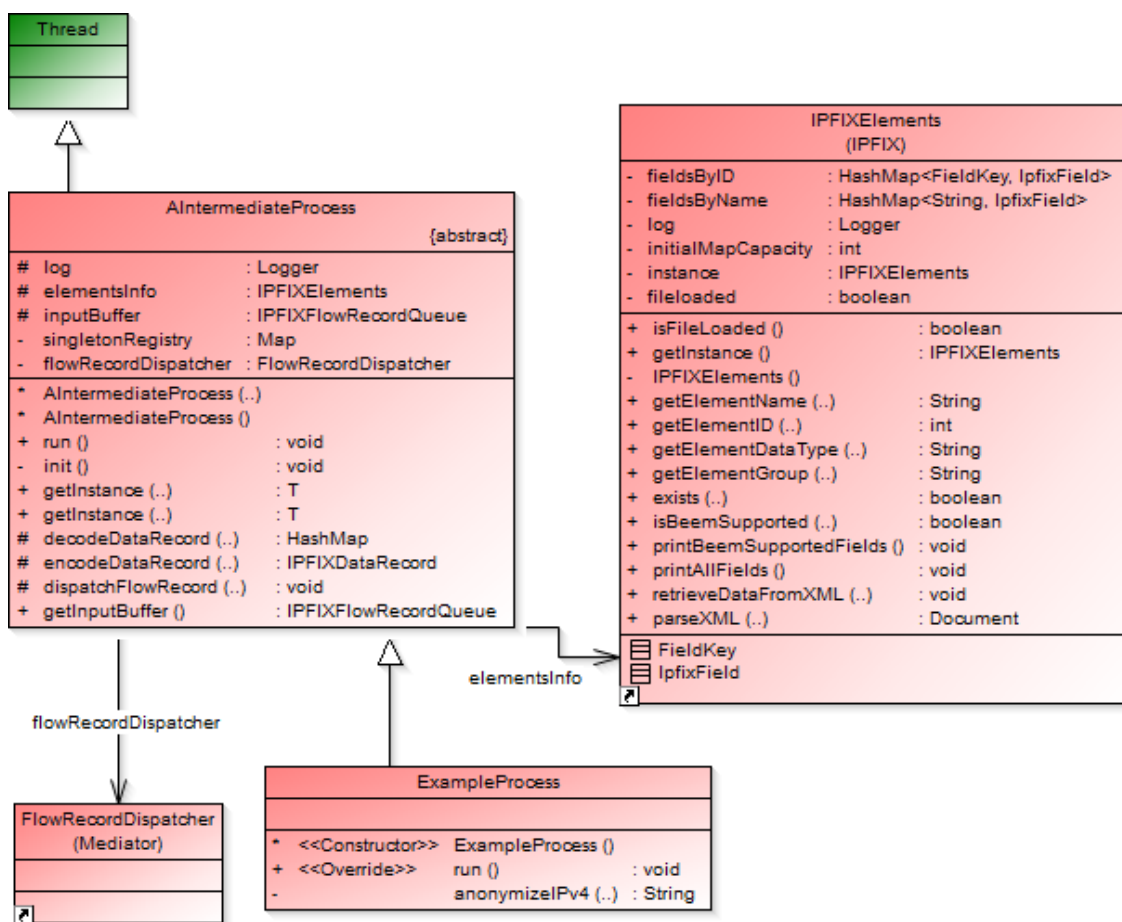
Vracia jedinečnú inštanciu objektu.

### 4.3 Balík **sk.tuke.cnl.bm.Mediator.IntermediateProcesses**

Diagram tried rozhrania pre sprostredkovateľské procesy, vrátane triedy **ExampleProcess** je znázornený na obrázku 4–1.

#### 4.3.1 Trieda **AIntermediateProcess**

Abstraktná trieda, poskytuje východiskové metódy sprostredkovateľským procesom a tvorí akési rozhranie medzi modulmi a aplikačným rámcom. Dedí od triedy **Thread**.



Obr. 4–1 Diagram tried rozhrania pre sprostredkovateľské procesy

## Konštruktor

### *AIntermediateProcess(String childName)*

Nastaví meno procesu, podľa prijatého parametra. Inicializuje vstupný buffer a získa jedinečnú inštanciu triedy IPFIXElements.

#### Parametre:

*String* childName - meno triedy potomka

### *AIntermediateProcess()*

Bezparametrický konštruktor. Inicializuje vstupný buffer a získa jedinečnú inštanciu triedy IPFIXElements.

**Metody:**

```
public static final synchronized <T extends AIntermediateProcess> T getInstance  
(Class clazz)
```

Toto riešenie je hybridom viacerých prístupov, ktoré sa diskutujú na Internete, no vychádza z návrhového vzoru *Factory method*. Výsledkom je abstraktná trieda, slúžiaca ako továreň na podtriedy tým, že volá jej statickú metódu *getInstance(Class clazz)*. Ak sú splnené podmienky, že konkrétna trieda, napr. **SelectionProcess** je definovaná v rovnakom balíčku ako **AIntermediateProcess** a ich konštruktory nemajú explicitne nastavený prístup (predvoleným prístupom je „privátny v rámci balíčka“), tak jediným spôsobom ako získať inštanciu podtriedy mimo balíčka je cez konštrukciu:

```
SelectionProcess instance =  
AIntermediateProcess.getInstance(SelectionProcess.class);
```

Dalo by sa vyčítať, že vytváranie inšancií používa reflexiu, ktorá je pomalá. Avšak, keďže vytvárame Singletony, volanie *newInstance()* sa vykoná pre každý modul práve raz.

**Parametre:**

*Class clazz* = class objekt požadovanej triedy

**Návratová hodnota:**

Objekt typu *T*, pričom *T* dedí od **AIntermediateProcess**.

```
public static final synchronized <T extends AIntermediateProcess> T getInstance  
(String processName)
```

Aby bolo možné získavať inšancie modulov aj na základe mena triedy a nie len cez class objekty, vytvoril som túto metódu. Premennú *processName* prevedie na binárne meno procesu, podľa špecifikácie jazyka Java, teda názov triedy vrátane balíčkov,

napr. `sk.tuke.cnl.Mediator.SelectionProcess`. Táto metóda načíta *class* objekt sprostredkovateľského procesu cez systémový class loader, tak ako to bolo vyššie spomínané. Potom zavolá pôvodnú metódu *getInstance(Class clazz)* a vráti inštanciu procesu.

**Parametre:**

*String* processName = meno požadovanej triedy

**Návratová hodnota:**

Objekt typu T, pričom T dedí od `AIntermediateProcess`.

*protected final HashMap* **decodeDataRecord**(*IPFIXTemplateRecord* template, *IPFIXDataRecord* dataRecord)

Pri prvom prechode funkciou sa generuje pamäťový záznam o informačných elementoch (ie) z XML súboru. Vytiahnu sa informácie o ie, ktoré sa nachádzajú v šablóne, dekodujú sa ich dátové typy a príslušnosť k skupine.

**Parametre:**

*IPFIXTemplateRecord* template - šablóna dát

*IPFIXDataRecord* dataRecord - dátový záznam

**Návratová hodnota:**

Dekódované dáta ako objekt typu `HashMap`.

*protected final IPFIXDataRecord* **encodeDataRecord**(*IPFIXTemplateRecord* template, *HashMap<String, String>* dataMap)

Zakóduje všetky hodnoty z hashmappy obsahujúcej hodnoty informačných elementov podľa šablóny do dátového záznamu.

**Parametre:**

*IPFIXTemplateRecord* template - šablóna dát

*HashMap<String, String>* dataMap - hodnoty informačných elementov v hashmape,

ktorá sa má zakódovať

**Návratová hodnota:**

Vracia objekt dátového záznamu - `IPFIXDataRecord`.

**Hádže:**

`EncodingException` - Ak nastane chyba pri kódovaní.

*protected final void **dispatchFlowRecord**(IPFIXFlowRecord flowRecord, String inputProcess)*

Vytvára rozhranie pre prístup k metóde aplikačného rámca.

**Parametre:**

*IPFIXFlowRecord* flowRecord - záznam o toku, ktorý sa má posunúť ďalej

*String* inputProcess - pôvodca záznamu o toku

#### 4.3.2 Trieda `ExampleProcess`

Táto trieda je vzorovým riešením jednoduchého sprostredkovateľského procesu, vykonávajúceho anonymizáciu. Účelom triedy je pomoc ďalšej generácii riešiteľov.

**Konštruktor**

***ExampleProcess**()*

Vola rodičovský konštruktor a predáva mu svoje meno ako parameter.

**Metody:**

*public void **run**()*

Hlavná metóda vlákna. V cykle čaká na záznamy o tokoch vo svojom vstupnom bufferi (*inputBuffer*) a postupne ich odtiaľ číta a odstraňuje. Nazvime ich *vstupne záznamy*. Vstupný buffer jej naplňa trieda `FlowRecordDispatcher`. Po prečítaní

vstupného záznamu vytvorí a inicializuje *výstupný záznam*. Následne prechádza všetky dátové záznamy vstupného záznamu, dekoduje ich, anonymizuje zdrojovú a cieľovú IP adresu a naspať zakóduje. Ak všetko prebehlo bez problémov, tak dátový záznam priradí výstupnému záznamu. Napokon výstupný záznam o toku posunie distribútorovi záznamov, ktorý ho bude prepošle nasledujúcemu sprostredkovateľskému procesu, alebo pripraví na export.

*private String **anonymizeIPv4**(String address)*

Metóda na veľmi jednoduchú anonymizáciu IP adresy, číslo v poslednom oktete zmení na 0.

**Parametre:**

*String* address - IP adresa, ktorá sa anonymizovať

**Návratová hodnota:**

Anonymizovaná IP adresa, vrátená ako reťazec.

### 4.3.3 Trieda **IPInputBuffer**

Reprezentuje vstupný buffer sprostredkovateľských modulov. Táto trieda je cache pre záznamy o tokoch. Jej použitie je kritické vo vysokorýchlostných sieťach, pretože udržiava elementy a tým pádom môže vyrovnávať nárazový nápor. Je synchronizovaná a jej implementácia je FIFO front typu **ArrayBlockingQueue**.

**Metódy:**

*public boolean **write**(IPFIXFlowRecord flowRecord)*

Zapisuje záznamy o tokoch do frontu. Ak je front plný, záznam sa zahadzuje.

**Parametre:**

*IPFIXFlowRecord* flowRecord - záznam o toku

**Návratová hodnota:**

Pravdivostná hodnota podľa toho, či záznam bol, alebo nebol zapísaný do cache.

```
public void write(IPFIXTemplateRecord template, ArrayList<IPFIXDataRecord>  
dataRecords, IPFIXMessage.IPFIXMessageHeader messageHeader)
```

Metóda obalí prijaté parametre do objektu triedy `IPFIXFlowRecord` a zavolá predchádzajúcu metódu.

**Parametre:**

*IPFIXTemplateRecord* template - šablóna

*ArrayList*<*IPFIXDataRecord*> dataRecords - pole dátových záznamov

*IPFIXMessage.IPFIXMessageHeader* messageHeader - hlavička IPFIX správy, z ktorej tento záznam o toku pochádza

```
public IPFIXFlowRecord read()
```

Prečíta a zmaže vrchol frontu. Ak je front prázdny čaká dokiaľ sa tam nejaký element nepridá.

**Návratová hodnota:**

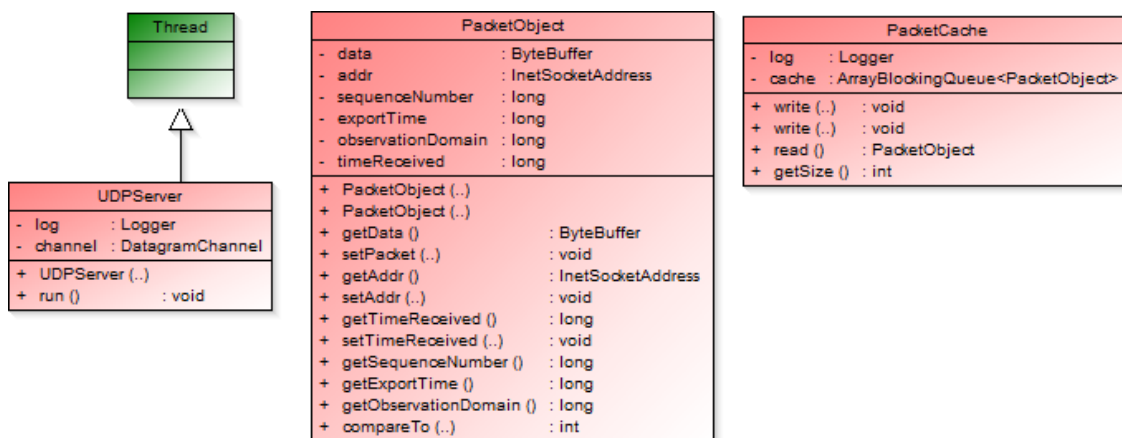
Objekt triedy `IPFIXFlowRecord`.

**Hádže:**

`InterruptedException` - Ak nastala chyba pri synchronizovaní vlákien, alebo ak bolo vlákno prerušené počas čakania.

## 4.4 Balík `sk.tuke.cnl.bm.Mediator.collecting`

Zjednodušený diagram tried tohto balíka môžeme vidieť na Obr. 4–2 a na Obr. 4–3.



Obr. 4–2 Diagram tried prvej fázy zhromažďovacieho procesu

#### 4.4.1 Trieda UDPSever

Slúži ako UDP server. Prijíma UDP datagramy cez `DatagramChannel` a ukladá ich do `PacketCache`.

##### Konštruktor

```
public UDPSever(int port)
```

Konštruktor inicializuje `DatagramChannel`, nastaví mu blokovací režim a priviaže ho k portu definovanom v konfiguračnom súbore, ktorý mu je predaný ako parameter. Nastaví meno vlákna.

##### Parametre:

`int` port - číslo portu

##### Metódy

```
public void run()
```

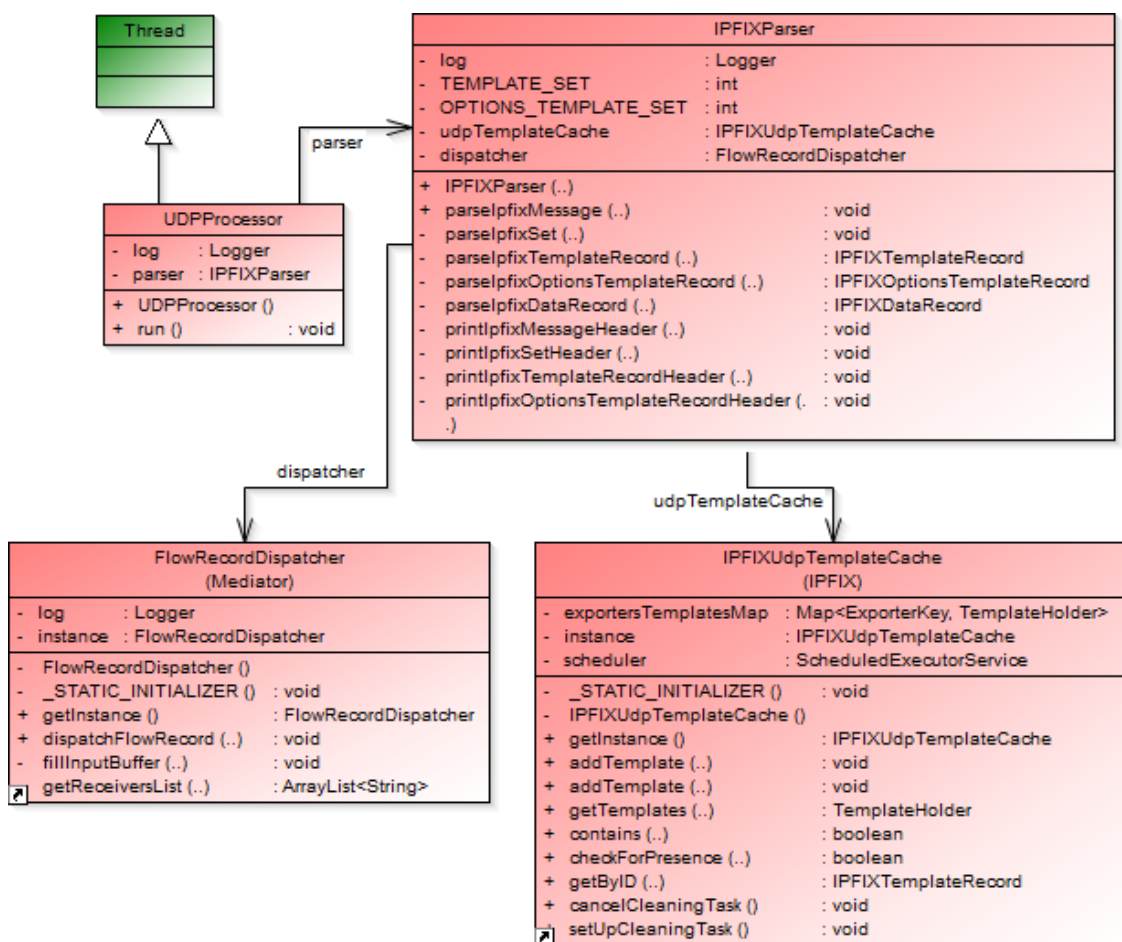
Hlavná metóda vlákna. Pokiaľ nedôjde k prerušeniu, prijíma cez vytvorený kanál dáta od exportéra. Prijaté dáta obalí do objektu `ByteBuffer` a predá ich spolu s časom prijatia a IP adresou a portom exportéra metóde `write()`, ktorá ich zapíše do



PacketCache.

```
public void cleanUp()
```

Táto metóda zruší čistiace vlákno pre UDP Template Cache. Je volaná pri prerušení tohto vlákna.



Obr. 4 – 3 Diagram tried druhej fázy zhromažďovacieho procesu

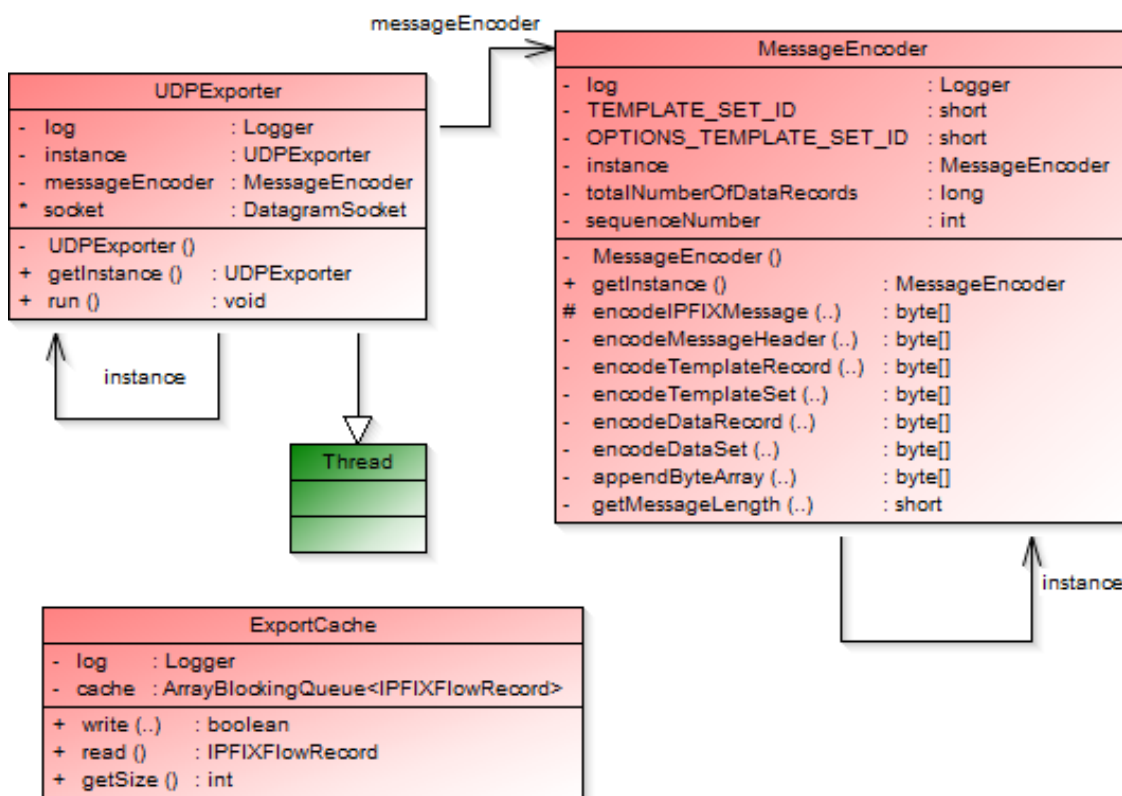
#### 4.4.2 Trieda IpfixParser

Táto trieda sa používa na parsovanie IPFIX správ a ich spracovanie. V porovnaní s verziou v aplikácii JXColl bola prečistená. Boli vypustené sekcie spracovávajúce TCP a SCTP spojenia. Zásadnejšia zmena prišla na výstupe z triedy. Sparované

dátové záznamy sú zabalené do vytvoreného objektu triedy `IPFIXFlowRecord`, spolu s príslušnou šablónou a hlavičkou prijatej IPFIX správy. Vytvorený záznam o toku je spolu s reťazcom predstavujúcim zdroj záznamu (v tomto prípade „exportér“) posunutý triede `FlowRecordDispatcher`.

## 4.5 Balík `sk.tuke.cnl.bm.Mediator.exporting`

Diagram tried exportovacieho procesu je na Obr. 4–4.



Obr. 4–4 Diagram tried exportovacieho procesu

### 4.5.1 Trieda `ExportCache`

Analógia k triede `IPInputBuffer`. Reprezentuje exportovaciu cache, ktorá je synchronizovaná a jej implementácia je FIFO front typu `ArrayBlockingQueue`.

**Metódy:**

*public static boolean* **write**(*IPFIXFlowRecord flowRecord*)

Zapisuje záznamy o tokoch do frontu. Ak je front plný, záznam sa zahadzuje.

**Parametre:**

*IPFIXFlowRecord flowRecord* - záznam o toku

**Návratová hodnota:**

Pravdivostná hodnota podľa toho, či záznam bol, alebo nebol zapísaný do cache.

*public static IPFIXFlowRecord* **read**()

Prečíta a zmaže vrchol frontu. Ak je front prázdny čaká dokiaľ sa tam nejaký element nepridá.

**Návratová hodnota:**

Objekt triedy *IPFIXFlowRecord*.

**Hádže:**

*InterruptedException* - Ak nastala chyba pri synchronizovaní vlákien, alebo ak bolo vlákno prerušené počas čakania.

*public static int* **getSize**()

**Návratová hodnota:**

Vracia počet elementov v cache.

#### 4.5.2 Trieda *MessageEncoder*

Trieda slúži na zakódovanie resp. zabalenie záznamu o toku do IPFIX paketu podľa špecifikácie v RFC 5101 a RFC 5102.

**Metódy:**

```
public static MessageEncoder getInstance()
```

**Návratová hodnota:**

Metóda vracia jedinečnú inštanciu objektu triedy podľa návrhového vzoru *Singleton*.

```
protected byte[] createIPFIXMessage(IPFIXFlowRecord flowRecord)
```

Na základe záznamu o toku vytvára prúd bytov z IPFIX správy. Vola jednotlivé metódy, ktoré robia čiastkové úlohy ako zakódovanie sád, hlavičky a podobne.

**Parametre:**

*IPFIXFlowRecord* flowRecord - záznam o toku

**Návratová hodnota:**

Pole bytov prúdu IPFIX správy.

```
private byte[] encodeMessageHeader(IPFIXMessage.IPFIXMessageHeader header, short length)
```

Zakóduje hlavičku IPFIX správy.

**Parametre:**

*IPFIXMessage.IPFIXMessageHeader* header - hlavička, ktorá sa má zakódovať

*short* length - celková dĺžka IPFIX správy

**Návratová hodnota:**

Pole bytov prúdu hlavičky IPFIX správy.

```
private byte[] encodeTemplateRecord(IPFIXTemplateRecord templateRecord)
```

Zakóduje záznam šablóny.

**Parametre:**

*IPFIXTemplateRecord* templateRecord - záznam šablóny, ktorý sa má zakódovať

**Návratová hodnota:**

Pole bytov prúdu záznamu šablóny.

```
private byte[] encodeTemplateSet(byte[] templateRecordBytes)
```

Zakóduje sadu šablón.

**Parametre:**

*byte[]* templateRecordBytes - pole bytov prúdu záznamov šablóny, ktoré obsahuje sadu šablóny

**Návratová hodnota:**

Pole bytov prúdu sady šablón.

```
private byte[] encodeDataRecord(IPFIXDataRecord dataRecord)
```

Zakóduje dátový záznam.

**Parametre:**

*IPFIXDataRecord* dataRecord - dátový záznam, ktorý sa má zakódovať

**Návratová hodnota:**

Pole bytov prúdu dátového záznamu.

```
private byte[] encodeDataSet(ByteArrayOutputStream dataRecordsStream, int templateID)
```

Zakóduje dátovú sadu.

**Parametre:**

*ByteArrayOutputStream* dataRecordsStream - prúd bytov dátových záznamov, ktoré obsahuje sada šablóny

*int* templateID - ID prislúchajúcej šablóny

**Návratová hodnota:**

Pole bytov prúdu dátovej sady.

---

```
private byte[] appendByteArray(byte[] first, byte[] second)
```

Pomocná metóda, ktorá na koniec prvého poľa bytov pripojí druhé pole bytov .

**Parametre:**

*byte[]* first - prve pole bytov

*byte[]* second - druhe pole bytov

**Návratová hodnota:**

Pole bytov výsledného poľa.

```
private short getMessageLength(byte[] templateSetBytes, byte[] dataSetBytes, byte[] optionsTemplateSetStream)
```

Vracia celkovú dĺžku IPFIX správy.

**Parametre:**

*byte[]* templateSetBytes - pole bytov sady šablón

*byte[]* dataSetBytes - pole bytov dátovej sady

*byte[]* optionsTemplateSetStream - pole bytov sady šablón možností

**Návratová hodnota:**

Celková dĺžka IPFIX správy, vrátane hlavičky. Návratový typ je *short*.

### 4.5.3 Trieda UDPExporter

Trieda je samostatným vláknom, záznamy o toku číta z ExportCache. Tie potom pošle triede MessageEncoder, ktorá ich zabalí do IPFIX správ. Zakódované správy exportuje kolektoru na IP adresu a port, ktoré sú definované v konfiguračnom súbore.

**Konštruktor:**

```
private UDPExporter()
```

Bezparametrický konštruktor. Vola rodičovský konštruktor s parametrom svojho

mena. Potom sa pokúsi vytvoriť socket pomocou triedy `DatagramSocket`.

### Metódy:

*public static MessageEncoder **getInstance()***

### Návratová hodnota:

Metóda vracia jedinečnú inštanciu objektu triedy podľa návrhového vzoru *Singleton*.

*public void **run()***

Hlavná metóda vlákna. Dokiaľ nie je vlákno prerušené, cyklicky číta záznamy o toku z `ExportCache`. Prostredníctvom triedy `MessageEncoder` vytvorí zo záznamu o toku výstupný prúd bytov. Ten spolu s dĺžkou prúdu, IP adresou a portom zabalí do paketu - objektu triedy `DatagramPacket`. Vzniknutý paket odošle.

## 4.6 Balík `sk.tuke.cnl.bm.exceptions`

Balík obsahuje triedy výnimiek špecifických pre Mediátor. Všetky výnimky dedia od hlavnej triedy výnimiek - `MediatorException`.

## 5 Preklad programu

### 5.1 Zoznam zdrojových textov

Zdrojové texty sú k dispozícii v prílohe diplomovej práce.

Sú k dispozícii tieto zdrojové texty:

- balík sk.tuke.cnl.bm.Mediator:
  - Config.java
  - Default.java
  - DropsCounter.java
  - FlowRecordDispatcher.java
  - IPLoader.java
  - Mediator.java
  - Support.java
- balík sk.tuke.cnl.bm.Mediator.IPFIX:
  - ExporterKey.java
  - FieldSpecifier.java
  - IPFIXDataRecord.java
  - IPFIXDecoder.java
  - IPFIXElements.java
  - IPFIXEncoder.java
  - IPFIXFlowRecord.java
  - IPFIXMessage.java
  - IPFIXOptionsTemplateRecord.java
  - IPFIXSet.java
  - IPFIXTemplateRecord.java
  - IpfixedTemplateCache.java
  - TemplateHolder.java
- balík sk.tuke.cnl.bm.Mediator.IntermediateProcesses:
  - AIntermediateProcess.java
  - ExampleProcess.java
  - IPInputBuffer.java
- balík sk.tuke.cnl.bm.Mediator.collecting:
  - IPFIXParser.java
  - PacketCache.java
  - PacketObject.java
  - UDPProcessor.java
  - UDPServer.java
- balík sk.tuke.cnl.bm.Mediator.exporting:
  - ExportCache.java



```
MessageEncoder.java
UDPExporter.java
- balík sk.tuke.cnl.bm.exceptions:
  DataException.java
  DataFormatException.java
  EncodingException.java
  IPLoaderException.java
  MediatorException.java
  OutOfBoundsException.java
  TemplateException.java
```

## 5.2 Požiadavky na technické prostriedky pri preklade

Preklad programu si vyžaduje nasledovnú hardvérovú konfiguráciu:

- CPU Intel Pentium III 1Ghz alebo ekvivalent
- operačná pamäť 512MB
- pevný disk s 1GB voľného miesta
- sieťová karta 100Mb/s

## 5.3 Požiadavky na programové prostriedky pri preklade

- operačný systém GNU/Linux s verziou jadra 2.6 a vyššou
- Java Runtime Environment (JRE) verzie 1.7.0\_03 a vyššej
- knižnice dodávané na inštalačnom médiu

## 5.4 Náväznosť na iné programové produkty

Program umožňuje sprostredkovanie správ medzi meracím/exportovacím procesom a zhromažďovacím procesom IPFIX architektúry, ktoré budú následne vyhodnotené príslušnými aplikáciami. Je implementáciou (*IP Flow Information Export (IPFIX)*)

*Mediation Problem*) v architektúre SLAmeter. Z toho vyplýva jeho náväznosť na merací/exportovací proces - BEEM a zhromažďovací proces - JXColl (alebo iné implementácie týchto procesov).

## 5.5 Vlastný preklad

Preklad programu spočíva v nakopírovaní zdrojových súborov a spustení kompilátora jazyka Java s potrebnými parametrami a parametrom classpath nastaveným na prídavné knižnice. Odporúča sa použiť váš obľúbený java IDE, kde stačí jednoducho nastaviť verziu JDK na 7.0 alebo vyššie a do cesty classpath pridať cesty ku všetkým potrebným knižniciam. V prostredí NetBeans IDE potom stačí kliknúť na tlačidlo *Clean and Build*.

## 5.6 Vytvorenie inštalačného DEB súboru

Stačí spustiť skript `buildDeb.sh`, ktorý sa nachádza v priečinku `/deb`.

```
sh buildDeb.sh
```

Výstupom tohto skriptu je súbor s názvom `debian.deb`, ktorý môžeme následne premenovať podľa verzie Mediátora (napríklad na `mediator_1.0_i386.deb`). Tento skript vykonáva nasledovné:

1. v prípade, ak neexistuje priečinok `debian`, extrahuje ho z archívu `debian.tar.gz`, inak tento krok preskočí
2. skopíruje binárny súbor z projektu do DEB balíčka (predpokladá sa, že bol program kompilovaný v Netbeans IDE pomocou Clean and Build tlačidla)
3. skopíruje konfiguračný súbor z projektu do DEB balíčka
4. skopíruje IPFIX definičný súbor z projektu do DEB balíčka

5. vymaže prípadné dočasné súbory z DEB balíčka
6. vygeneruje MD5 kontrolné súčty pre všetky súbory DEB balíčka
7. zabezpečí maximálnu kompresiu manuálových stránok a changelog súborov
8. skopíruje binárny súbor z projektu do DEB balíčka a nastaví mu práva na vykonávanie
9. vytvorí samotný DEB balíček
10. overí ho pomocou programu lintian - ten vypíše prípadne varovania a/alebo chyby
11. archivuje vytvorený DEB balíček do archívu debian.tar.gz

Pred spustením skriptu je nutné skompilovať Mediátor pomocou Netbeans IDE tlačidlom *Clean and Build*. Prípadné zmeny control alebo changelog súboru, manuálových stránok je nutné vykonať ručne. Manuálové stránky je vhodné upraviť pomocou programu *GmanEdit* . Po spustení skriptu sa vytvorí DEB balíček s názvom `debian.deb`. Ten je vhodné premenovať podľa aktuálnej verzie. Vytvorí sa aj archív `debian.tar.gz`, ktorý obsahuje najaktuálnejšiu adresárovú štruktúru DEB balíčka pre budúce využitie (ak neexistuje priečinok `debian`, vytvorí sa extrakciou z tohto archívu). Ak je potrebné len aktualizovať kód, stačí spustiť skript a ten sa o všetko postará, pričom vytvorí aj adresár `debian`. Súbory možno v ňom upravovať až kým nie je všetko podľa predstáv. Ak je všetko hotové, v Netbeans IDE je potrebné vymazať priečinok `debian` (vykoná sa SVN DELETE, namiesto obyčajného odstránenia zo súborového systému) a projekt "commitnúť". [5, 7]

## 5.7 Opis známych chýb

V súčasnosti nie sú známe žiadne vážne chyby.

## 6 Zhodnotenie riešenia

Hlavným cieľom práce bolo zvýšiť interoperabilitu s inými IPFIX riešeniami pomocou zvýšenia konformity so štandardom IPFIX. V práci boli vyriešené problémy, ktoré doteraz znemožňovali dekódovanie viacerých záznamov sade, informačných elementov s variabilnou dĺžkou, informačných elementov s redukovaným kódovaním alebo niektorých predtým neimplementovaných dátových typov.

Súčasťou práce bolo rozšírenie podpory prenosu údajov o tokoch prostredníctvom transportných protokolov TCP a SCTP, čo zvyšuje možnosti nasadenia nástroja BasicMeter aj v podmienkach s vyššou náchylnosťou na preťaženie v sieti.

Možnosti budúceho vývoja zhromažďovacieho procesu nástroja BasicMeter predstavuje implementácia podpory pre dátové typy umožňujúce prenos štruktúrovaných dát a podpora pre zabezpečené pripojenia od exportérov.

## Literatúra

- [1] KOBAYASHI, A. – CLAISE, B. et al.: *IP Flow Information Export (IPFIX) Mediation: Problem Statement*. RFC 5982. 2010
- [2] KOBAYASHI, A. et al.: *IP Flow Information Export (IPFIX) Mediation: Framework*. RFC 6183. 2011
- [3] SADASIVAN, G. et al.: *Architecture for IP Flow Information Export* RFC 5470. 2009
- [4] CLAISE, B.: *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. 2004
- [5] VEREŠČÁK, T.: Optimalizácia zhromažďovacieho procesu nástroja BasicMeter, 2012, Diplomová práca, Príloha A, Systémová príručka JXColl v3.9, KPI FEI TU, Košice
- [6] PEKÁR, A.: Meranie prevádzkových parametrov siete v reálnom čase, 2009, Bakalárska práca, KPI FEI TU, Košice
- [7] PEKÁR, A.: Optimalizácia zhromažďovacieho procesu nástroja BasicMeter, 2011, Diplomová práca, KPI FEI TU, Košice