VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

BRÁNA PRO PŘEKLAD SIGNALIZAČNÍCH ZPRÁV PRO VIDEOKONFERENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

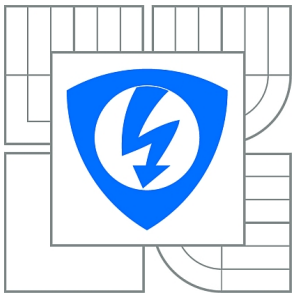
AUTOR PRÁCE
AUTHOR

Bc. DAVID SEIFERT

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

BRÁNA PRO PŘEKLAD SIGNALIZAČNÍCH ZPRÁV PRO VIDEOKONFERENCE

SIGNALLING GATEWAY FOR VIDEOCONFERENCING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

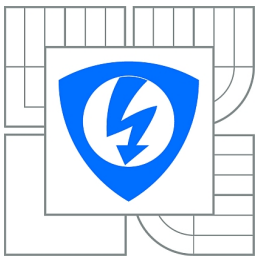
AUTOR PRÁCE — Bc. DAVID SEIFERT

AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN ZUKAL

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. David Seifert ***ID:*** 106765
Ročník: 2 ***Akademický rok:*** 2012/2013

NÁZEV TÉMATU:

Brána pro překlad signalizačních zpráv pro videokonference

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a zrealizujte vlastní řešení pro překlad SIP a WebSocket zpráv. Aplikace bude přijímat SIP zprávy zabalené ve zprávách protokolu WebSocket a bude je dále posílat jako SIP zprávy a naopak. Dbejte na efektivní implementaci tak, aby byla aplikace schopna obsloužit i větší počet spojení. Funkčnost aplikace ověřte testováním.

DOPORUČENÁ LITERATURA:

[1] JOHNSTON, A. B. SIP: Understanding the Session Initiation Protocol, Second Edition. London: Artech House Publishers, 2004. ISBN 978-1580536554
[2] FETTE, I., MELNIKOV, A. The WebSocket Protocol [online]. December 2011, [cit. 2012-10-09]. Dostupný z WWW: <http://tools.ietf.org/html/rfc6455>
[3] CASTILLO, I. B., VILLEGAS, J.M., PASCUAL V. The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP) [online]. September 7 2012, [cit. 2012-10-09]. Dostupný z WWW: <http://tools.ietf.org/html/draft-ietf-sipcore-sip-websocket-03>

Termín zadání: 11.2.2013 ***Termín odevzdání:*** 29.5.2013

Vedoucí práce: Ing. Martin Zukal
Konzultanti diplomové práce:

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

UPOZORNĚNÍ:

Abstrakt

Obecně se pod pojmem brána (Gateway) v počítačových sítích myslí uzel, který spojuje dvě sítě s odlišnými protokoly. Tato diplomová práce se zabývá překladem mezi 2 protokoly, protokolem SIP a protokolem WebSocket. Tento překlad je v práci nejprve popsán teoreticky a následně se práce zabývá způsobem řešení tohoto překladu s pomocí dostupných nástrojů na Internetu.

Klíčová slova

protokol, SIP, WebSocket, Brána, Java, jWebSocket, JAIN-SIP, třída

Keywords

protocol, SIP, WebSocket, Gateway, Java, jWebSocket, JAIN-SIP, class

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma " *Brána pro překlad signalizačních zpráv pro videokonference* " jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne …… ……..
(podpis autora)

Poděkování

Děkuji vedoucímu práce panu Ing. Martinu Zukalovi za velmi užitečnou metodickou pomoc, cenné rady, trpělivost a lidský přístup při zpracování diplomové práce.

V Brně dne …….. .....

podpis autora

Bibliografická citace mé práce:

SEIFERT, D. *Brána pro překlad signalizačních zpráv pro videokonference.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 60 s. Vedoucí semestrální práce Ing. Martin Zukal.

OBSAH

ÚVOD

Protokol SIP podporuje relace v rámci různých aplikací na Internetu. Mezi ně patří například VOIP telefonie či konference. Pokud ale máme webový prohlížeč, který není schopen posílat přímo SIP zprávy, je možným řešením posílat tyto SIP zprávy zabalené ve WebSocket paketech dle standardu [12] aplikaci, která je přeloží na běžné SIP zprávy. Aby bylo možné uskutečnit VOIP hovor přímo z prohlížeče je nutné mít k dispozici: webový prohlížeč s podporou technologie WebRTC (http://www.webrtc.org/), SIP klienta implementovaného v JavaScriptu (např. s využitím Jssip - http://www.jssip.net/), bránu pro překlad WebSocket zpráv na SIP zprávy a vhodný Registrar server s proxy serverem – např. Asterisk.
Tato diplomová práce se zabývá způsobem překladu zpráv protokolu WebSocket na zprávy protokolu SIP a naopak dle standardu [12], zabývá se tedy realizací brány. První kapitola uvádí základní popis protokolu SIP – k čemu je protokol určen, jeho syntaxe, metody a základní pole záhlaví. Druhá kapitola se věnuje WebSocket protokolu. Podobně jako první kapitola se zabývá jeho základním popisem - účelem protokolu, syntaxí, způsobem navázání spojení, zprávami tohoto protokolu. Další kapitola se zabývá překladem SIP zpráv na WebSocket zprávy. Popisuje, jakým způsobem je možné využít zprávy WebSocket protokolu pro přenos SIP zpráv. Čtvrtá kapitola pojednává o samotném řešení tohoto překladu. První podkapitola popisuje porovnání a výběr dostupných knihoven. Druhá podkapitola se věnuje popisu návrhu řešení pomocí diagramu tříd a jeho rozboru. Na základě tohoto diagramu je aplikace implementována v jazyce Java. Způsob implementace je popsaný ve třetí podkapitole. Třetí podkapitola popisuje účel a funkci jednotlivých rozhraní a tříd aplikace. Aby bylo možné vytvořenou aplikaci zvanou brána otestovat, je implementována testovací aplikace. Pátá kapitola se zabývá testovací aplikací. Podobně jako kapitola čtvrtá, popisuje význam jednotlivých, funkčně významných částí tříd, tvořící testovací aplikaci. Po popisu testovací aplikace se pátá kapitola věnuje samotným testům brány.

1 SIP protokol

SIP protokol [1] je signalizační protokol, který je textově orientovaný. Slouží pro řízení navázání, modifikaci a ukončení spojení s jedním nebo více účastníky v IP sítích [7]. Účastníci ve spojení spolu komunikují skrze skupinové vysílání, tzv. multicast (speciální forma všesměrového vysílání, kdy kopie paketu jsou doručeny vymezené skupině koncových uzlů), nebo více spojení typu bod-bod, tzv. unicast. Je možná i kombinace obou uvedených možností.
SIP protokol pracuje jako součást tzv. multimediální architektury [1]. Ta zahrnuje více protokolů, které poskytují služby tvořící komunikační systém umožňující komunikaci v reálném čase poskytující uživateli kompletní služby. Mezi tyto protokoly patří např.Real-time Transport Protocol (RTP) [9] pro přenos multimediálních dat a poskytování QoS zpětné vazby, Real-time Streaming Protocol (RTSP) [10] pro ovládání proudového vysílání multimediálních dat, Session Description Protocol (SDP) [11] pro popis multimediálních relací apod.

Protokol poskytuje primitiva, které lze použít pro poskytnutí různých služeb:

- lokalizaci účastníka – nalezení spojení s koncovou stanicí,
- zjištění stavu účastníka – zjištění, jestli je účastník schopen relaci navázat (může mít obsazeno, zapnutou funkci přesměrování apod.),
- zjištění možností účastníka – zjištění jaké jsou možnosti účastníka (typ kodeku, max. přenosová rychlost, audio/video atd.),
- vlastní navázání spojení – zde vstupuje do hry také protokol SDP, který popisuje navázané spojení a odkazuje na RTP datový tok,
- řízení probíhajícího spojení – případné změny vlastností v průběhu relace a činnosti spojené s jejím ukončováním.

V současnosti je protokol SIP využíván především pro IP telefonii, lze jej použít pro hlasové hovory, video-hovory, tak i video-konference. Hojně se používá pro službu zasílání zpráv (instant messaging) a v aplikacích tohoto typu.

Typické scénáře včetně sestavení spojení je možné nalézt v [1].

1.1 SIP zprávy

Jak bylo řečeno výše, SIP je textově-orientovaný protokol a používá znakovou sadu UTF-8. SIP zprávou je kterákoliv žádost poslaná od klienta k serveru nebo odpověď poslaná ze serveru na klienta. Tedy zprávy protokolu SIP jsou dvojího druhu – žádosti a odpovědi. Oba typy zpráv, tj. žádost a odpověď, používají formát předepsaný v RFC 2822. Oba typy zpráv se skládají ze „startovního řádku" – v angličtině start-line, jednoho nebo více polí záhlaví (hlavičky) a prázdného řádku signalizující konec polí záhlaví a volitelného těla zprávy [1].
Na pozici prvního řádku, tedy „startovního řádku" se nachází buď řádek zvaný „řádek žádosti" (v angličtině „Request-line"), to pokud se jedná o žádost a nebo řádek zvaný „stavový řádek" („Status-line"), pokud se jedná o odpověď.
Složení těchto dvou typů řádků je následující [1]:

Řádek žádosti = metoda M URI žádosti M verze protokolu SIP CRLF
Startovní řádek = verze protokolu SIP M návratový kód M významová fráze CRLF

, kde

- metoda – představuje požadavek, který vznáší klient,
- M (anglický termín je single space) – mezera,
- URI žádosti (Request-URI) – označení klienta nebo serveru, kterému je požadavek zaslán,
- verze protokolu SIP (SIP-Version) – použitá verze protokolu SIP,
- návratový kód (Status-Code) – také označovaný jako stavový kód, je to 3-místné číslo, které ukazuje výsledek při pokusu o porozumění a uspokojení žádosti,
- významová fráze (Reason-Phrase) – používá se ke krátkému textovému popisu návratovému kódu, představuje „lidský pohled" na návratový kód.

Každý řádek hlavičky zprávy a prázdný řádek musí být ukončeny posloupností tzv. carrige-return line-feed (CRLF) – tj. znak vložený, když se stiskne např. klávesa „Enter" na počítači.

1.1.1 Žádosti

Každá žádost nese jednu metodu. Ta vyvolává na serveru určitou funkci:

- INVITE: Metoda INVITE slouží k přizvání uživatele nebo služby k podílení se na relaci. Tělo zprávy obsahuje popis relace (spojení).

- ACK: Metoda ACK potvrzuje, že klient v pořádku přijal odpověď na INVITE dotaz.

- BYE: Klient používá metodu BYE k oznámení protistraně, že hodlá ukončit hovor. Metoda BYE může být vyslána jak volaným tak volajícím.

- CANCEL: Metoda CANCEL ukončuje nevyřízený požadavek se stejnou identifikací, tedy položkami Call-ID, To, From a pořadovým číslem požadavku Cseq. Vyřízené požadavky již metoda CANCEL neovlivní (požadavek je vyřízen pokud byla odeslána konečná odpověď, např. 200 OK).

- REGISTER: Metoda REGISTER je používána k registraci současné adresy klienta u SIP serveru, který jí předá lokalizační službě.

Příklad žádosti:

INVITE sip:user2@server2.com SIP/2.0
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: user2 <sip:user2@server2.com>
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 314159 INVITE
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142
Pozn. : Tělo zprávy bylo vypuštěno.

1.1.2 Odpovědi

Odpovědi se „dělí" do šesti tříd. Tuto třídu určuje návratový kód, který označuje výsledek žádosti obdobně, jak je tomu například v protokolu HTTP. Rozdělení do tříd odpovědí se děje podle první číslice:

1*xx* - žádost přijata, pokračuji ve zpracování žádosti,
2*xx* - znamená úspěšné provedení žádosti,
3*xx* - označuje přesměrování (odpověď od přesměrovacího serveru),
4*xx* - chyba způsobená klientem (chybný formát žádosti),
5*xx* - chyba způsobená serverem,
6*xx* - obecná chyba, žádost nemůže být akceptována ani jiným serverem.
Nejběžnější odpovědi jsou "200 OK" (úspěšné provedení žádosti), "302 Moved Temporarily" (běžné přesměrování) a "404 Not Found" (volaný uživatel se nenachází na daném serveru).

příklad odpovědi:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP site4.server2.com;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP site3.server1.com;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds;received=192.0.2.1
Max-Forwards: 70
To: user2 <sip:user2@server2.com>;tag=a6c85cf
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 314159 INVITE
Contact: <sip:user2@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
```

Pozn. : Tělo zprávy bylo vypuštěno.

Nejdůležitějšími poli záhlaví zprávy jsou [3]:

- Via: obsahuje lokální adresu uživatele, který vysílá požadavek, kde se očekává, že přijde odpověď nebo se jedná o adresu serverů, přes které požadavek prošel a přes které bude opačným směrem procházet odpověď. Dále obsahuje parametr branch, který identifikuje SIP transakci žádosti. Používá jej server i klient, navíc umožňuje detekovat případné smyčky.
- Max-Forwards: představuje maximální počet skoků, které může zpráva absolvovat před dosažením příjemce, zabraňuje tak případným smyčkám, tj. její existenci "napořád".
- URI žádosti (Request-URI): je součástí prvního řádku – „startovního řádku", její výchozí – počáteční hodnota by měla být nastavena na hodnotu v poli To, představuje aktuálního adresáta.
- To : adresa volaného, určuje požadovaného "logického" příjemce žádosti, obsahuje "display name"(pole umožňuje zobrazit jméno) a SIP nebo SIPS URI, např. To: user2 <user2@server2.com> nebo <sip:operator@cs.columbia.edu> .
- From: adresa volajícího, označuje logickou identitu iniciátora žádosti.
- Contact : obsahuje SIP nebo SIPS URI, které představuje přímou cestu k odesílajícímu-volajícímu, tomu, co posílá žádost. Obsahuje uživatelské jméno a plně kvalifikovaný název domény (volajícího) (FQDN). Může také obsahovat IP adresu.
- Call-Id : obsahuje globálně jedinečný identifikátor pro toto volání, generované kombinací náhodného řetězce a doménového jména uživatele nebo IP adresy, unikátní identifikace volání.
- CSeq : v žádosti obsahuje jedno decimální pořadové číslo a metodu žádosti. Pořadové číslo musí být vyjádřeno jako 32-bitové nezáporné číslo. Toto pole slouží k seřazení transakcí v rámci dialogu, poskytuje prostředek pro jednoznačnou identifikaci transakce, rozlišuje mezi novými žádostmi a žádostmi opakovaných přenosů.

Informace k dalším záhlavím lze nalézt v [1].

2 WebSocket protokol

WebSocket je webová technologie poskytující plně duplexní komunikační kanály přes jedno TCP spojení. WebSocket tak nahrazuje dotazový model HTTP, který nevyužívá možnosti obousměrné komunikace přes jedno TCP spojení. API WebSocketu je standardizováno W3C, WebSocket protokol je standardizován IETF jako RFC6455. WS je navržen k implementaci ve webových prohlížečích a webových serverech, ale může být používán jakoukoliv klientskou nebo serverovou aplikací. WebSocket protokol je nezávislý na TCP protokolu.

Mezi výhody WebSocket protokolu patří mimo jiné komunikace v reálném čase a menší zatížení sítě. WebSocket protokol se díky svým dobrým vlastnostem používá v řadě webových aplikací (zejména ty, pracující v reálném čase): hry, burzy, víceuživatelské aplikace se simultánní editací, uživatelské rozhraní poskytující na straně serveru služby v reálném čase.

WebSocket protokol využívá stávajících dostupných webových technologií [4]. V rámci tohoto konstrukčního principu specifikace protokolu definuje, že WebSocket spojení začíná svůj "život" jako HTTP spojení zaručující plnou zpětnou kompatibilitu s "před-WebSocketským" světem.

2.1 Sestavení WebSocket spojení

Handshake je obecně automatizovaný proces vyjednávání, který dynamicky nastaví parametry komunikačního kanálu zřízeného mezi dvěmi entitami před začátkem klasické komunikace.
V našem případě před začátkem WebSocket komunikace.
Právě tímto procesem začíná každé WebSocket spojení (pomocí tohoto procesu je sestaveno). Jeho účelem je dosažení kompatibility se serverem a mezilehlými zařízeními, které podporují HTTP. Jeden port pak může být použitý pro HTTP klienty "mluvící" (provozující hovor) se serverem a zároveň pro WS klienty "mluvící" se stejným serverem [5]. Takže WebSocket spojení používá stejné porty jako http (80) a HTTPS (443).
Sestavení spojení začíná HTTP žádostí klienta:

```
GET ws://echo.example.com/ HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Část URI žádost (Request-URI) metody GET, které se nachází na prvním řádku, výše uvedeného příkladu za názvem této metody, se používá k identifikaci koncového bodu WebSocket spojení. Identifikuje zdroj, pro který je žádost určena.
Klientova žádost obsahuje jméno hostitele (hostname) v poli záhlaví jeho žádosti, takže oba - klient a server - se mohou „dohodnout"- zjistit, který hostitel je používán.
Další pole záhlaví jsou používané k výběru nějaké z možností, které protokol WebSocket nabízí.

Pole "Sec-WebSocket-Protokol" se používá při zahajování WebSocket spojení. Je posíláno od klienta k serveru a zpět ze serveru ke klientovi k potvrzení tzv. podprotokolu (anglicky sub-protocol) spojení (tedy protokolu, který budou používat v rámci WebSocket zpráv - ty jej budou obsahovat). To umožňuje skriptům jak výběr podprotokolu (tedy protokolu, který bude použit při komunikaci) tak jistotu, že server souhlasí s použitím tohoto podprotokolu [5]. Podprotokol představuje protokol aplikační vrstvy. Pro tuto práci je klíčovým polem, protože představuje možnost, jak lze přenášet SIP zprávy pomocí WebSocket zpráv [6].
Dále lze použít seznam klientem podporovaných rozšíření – "Sec-WebSocket-Extensions" – ,ten umožňuje dohodnout se na parametrech protokolu po dobu spojení [5].

Pole záhlaví Upgrade poskytuje mechanismus jednoduchého přepnutí protokolu na protokol, který je nekompatibilní s HTTP. Klient se v podstatě táže serveru, jestli je možné používat místo HTTP, protokol, který je uvedený v poli Upgrade. Pokud jej server ovládá, přepne protokol na protokol uvedený v poli Upgrade. V našem případě WebSocket protokol.

K prokázání, že byla přijata žádost od klienta, server potřebuje 2 druhy informací [5] a jejich kombinací vytváří odpověď. První druh informace pochází z pole záhlaví "Sec-Websocket-Key" v klientově žádosti:

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Z tohoto pole záhlaví, server musí vzít hodnotu (-tak jak je přítomná v záhlaví pole, t.j. base64-zakódovaná, mínus vedoucí a koncový bílý znak) a spojit ji s Globally Unique Identifier (- globálně jedinečný identifikátor - GUID - RFC 4122) - s globálně jedinečným identifikátorem "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" ve formátu řetězce. Je nepravděpodobné, že takový řetězec bude používaný síťovými koncovými body, které nerozumí WebSocket protokolu. Následně je použit hash typu SHA-1 (160 bitů) a nakonec zakódování na bázi base64 (RFC 4648), a z tohoto zřetězení je pak vrácena odpověď serveru na přijatou žádost [5].

Tato výsledná hodnota by se pak měla objevit ve vráceném poli záhlaví "Sec-Websocket-Accept".

Odpověď od serveru je většinou jednoduší než klientova žádost. První řádek je HTTP status-line, se "status" kódem-návratovým kódem 101:

HTTP/1.1 101 Switching Protocols

Každý jiný návratový kód než 101 signalizuje, proces vyjednávání WebSocket spojení není kompletní a že stále platí sémantika HTTP.
Pole záhlaví "Connection" a "Upgrade" dokončují HTTP Upgrade. Pole "Sec-WebSocket-Accept" dává na vědomí, zda je server ochoten přijmout spojení. Toto pole musí obsahovat hash klientova nonce (řetězec uvedený výše) poslaného v Sec-Websocket-Key spolu s předdefinovaným GUID. Žádná jiná hodnota nesmí být interpretována jako přijetí spojení serverem.

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

Tyto pole jsou následně kontrolovány WebSocket klientem. Pokud obsažená hodnota pole "Sec-WebSocket-Accept" neodpovídá očekávané hodnotě nebo pokud chybí nebo dokonce HTTP návratový kód není 101, spojení nebude sestaveno a WebSocket rámce nebudou odeslány.

2.2 Zprávy ve WebSocket protokolu

Po úspěšném navázání spojení, tedy po úspěšném vyjednávání spojení lze posílat data. To lze až do doby, než je poslán řídicí rámec Close.

Zprávy ve WebSocket protokolu jsou definovány v [5]. Data jsou přenášena pomocí sekvence rámců. Z důvodu bezpečnosti jsou rámce ve směru od klienta k serveru maskovány. Maskování se provádí bez ohledu na to, zda-li běží nebo neběží přes TLS spojení. Server nesmí povolit spojení, kde by byly rámce posílány nemaskované. Takové spojení server musí uzavřít. Naopak server své rámce nemaskuje a klient je naopak nucen uzavřít spojení v případě, že obdrží maskovaný rámec od serveru. Pro detaily o maskování viz. kap.5 [5].

Struktura rámce je popsána na obr.1 [5].

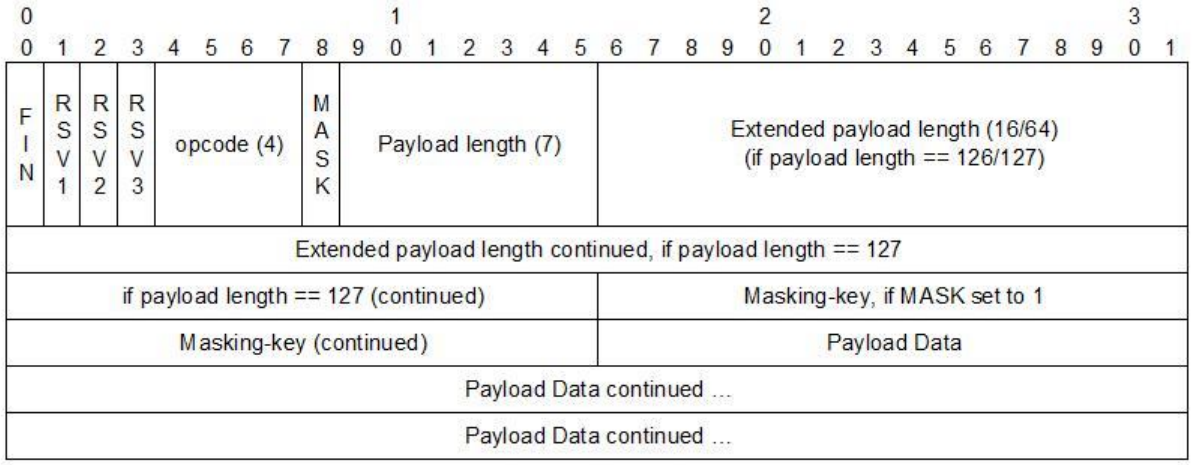
FIN: 1 bit

Značí, že se jedná o poslední fragment ve zprávě. První fragment může být také posledním fragmentem.

RSV1, RSV2, RSV3: každý 1 bit

Souvisí s možným rozšířením. Tyto bity musí být rovny 0, pokud rozšíření, dle dohody definuje význam pro nenulové hodnoty (je definováno jen pro nenulové hodnoty). Pokud je přijata nenulová hodnota a žádné z vyjednaných rozšíření nedefinuje význam takové nenulové

hodnoty, koncový bod, který takový rámec obdrží, to pak považuje za selhání WebSocket spojení.



Obr. 1 Struktura rámce WebSocket protokolu

Opcode: 4 bity

Definuje význam přenášených dat – „Payload data". Pokud je na jeho místě obdržena neznámá kombinace těchto 4 bitů, přijímající koncový bod tuto situaci musí považovat za selhání WebSocket spojení. Jsou definovány následující hodnoty:

- %x0 označuje pokračování rámce
- %x1 označuje textový rámec
- %x2 označuje binární rámec
- %x3-7 jsou rezervovány pro další neřídicí rámce
- %x8 označuje uzavření spojení
- %x9 označuje ping
- %xA označuje pong
- %xB-F jsou rezervovány pro další řídící(ovládací) rámce

Mask: 1 bit

Říká, zda přenášená data (Payload data) jsou maskována. Pokud je nastaven na 1, je přítomen maskovací klíč v poli „masking-key", a ten je používán k odmaskování užitečných dat dle [5] Všechny rámce poslané od klienta k serveru mají tento bit nastaven na 1.

Payload length: 7 bitů, 7+16 bitů, nebo 7+64 bitů

Délka přenášených dat v bytech: pokud je hodnota v rozmezí 0-125, pak se jedná o délku přenášených dat. Jestliže je 126, následující 2 byty interpretované jako 16-bitové celé číslo bez znaménka představující délku užitečné zátěže. Jestliže je 127, následujících 8 bytů je interpretováno jako 64-bitové celé číslo bez znaménka (nejvýznamnější bit musí být 0) a představují délkou užitečné zátěže.
Přenášené data představují součet dat rozšíření (anglický termín je Extension data) a dat aplikačních (Application data). Délka dat rozšíření může být nulová – nemusí se ve zprávě objevit, v takovém případě je délka přenášených dat rovna délce dat aplikačních.

Masking-key: 0 nebo 4 byty

Všechny rámce poslané od klienta k serveru jsou maskovány pomocí 32-bitové hodnoty, která je obsažena v rámu. Toto pole je přítomné, jestliže mask bit je nastaven na jedna. Chybí v případě, že mask bit je nastaven na 0. Viz. kap. 5.3 [5] pro více informací o maskování ve směru klient-server.

Payload data: (x+y) bytů

Přenášená data jsou definovány jako data rozšíření spojené s daty aplikačními (jejich součet).

Extension data: x bytů

Každé rozšíření musí být specifikované délkou dat rozšíření nebo jak může být tato délka vypočtena, a jak je třeba rozšíření použití se sjednávají během sestavování spojení. Jestliže je přítomno, data rozšíření jsou obsažena v celkové délce přenášených dat.

Application data: y bytů

Libovolné data aplikační nastupují do zbytku rámce po jakýkoliv datech rozšíření. Délka dat aplikačních je rovna délce přenášených dat mínus data rozšíření.

3 Překlad WebSocket protokolu na SIP

3.1. DEFINICE

SIP WebSocket klient: SIP entita schopná otevření odchozích připojení k WebSocket serverům a komunikující s nimi, použitím WebSocket podprotokolu (anglicky sub-protocol) SIP jak je definováno v - dokumentu [6].

SIP WebSocket Server: SIP entita schopná naslouchání příchozích připojení z WebSocket klientů a komunikující s nimi použitím WebSocket podprotokolu SIP jak je definováno v [6].

3.2. WebSocket pro přenos SIP zpráv

Jak bylo popsáno výše, WebSocket spojení začíná, dohodnutím pravidel sestavovaného spojení, využívající již existující HTTP strukturu. V průběhu ustanovování WebSocket spojení se mimo jiné klient a server domlouvají na tzv. podprotokolu (sub-protocol). Ten je aplikačním protokolem, který využívá WebSocket spojení [6].
V následující zprávě (žádosti) je oproti dříve zobrazené žádosti navíc obsaženo pole záhlaví "Sec-WebSocket-Protokol". To říká, o jaký protokol žádá klient server, aby byl používán .
V našem případě je to právě SIP protokol neboť ten bude přenášen pomocí WebSocket rámců.

```
GET / HTTP/1.1
Host: sip-ws.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

Server ve své odpovědi potvrzuje používání aplikačního protokolu sip.

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip

Jednotky WebSocket zpráv mohou obsahovat buď text v UTF-8 kódování nebo binární data a mohou být rozděleny do několika textových/binárních rámců dle potřeby.

Po případném úspěšném sestavení WebSocket spojení, může být toto spojení použito pro přenos SIP žádostí a SIP odpovědí. WebSocket zprávy předané přes toto spojení se musí přizpůsobit vyjednávanému WebSocket podprotokolu.

U SIP protokolu pole záhlaví Via v SIP zprávách nese identifikátor transportního protokolu.

Např.

Via: SIP/2.0/**UDP** pc33.server1.com;branch=z9hG4bK776asdhds Max-Forwards: 32

Dokument [6] definuje novou hodnotu "WS", která se má používat na žádosti přes klasické (nezabezpečené) WebSocket spojení a "WSS" na žádosti přes zabezpečené WebSocket spojení (zde je WebSocket spojení ustanoveno na základě TLS s TCP přenosem).

Parametr Via se také u klasického SIP spojení používá k záznamu cesty – servery do něj zapisují parametr "received"(obdržel), aby odpověď mohla projít stejnou cestou, jakou prošla žádost. Protože SIP odpovědi nyní mohou být poslány pouze přes existující WebSocket spojení, parametr Via se nepoužívá – nemá smysl. Dalším důvodem je bezpečnost – skrývání citlivých informací o síti SIP WebSocket serverů. Dokument [6] říká, že tato volba zápisu parametru "received" je na rozhodnutí jednotlivých SIP WebSocket serverů.

Dokument definuje hodnotu "ws" jako hodnotu přenosového parametru pro SIP URI (RFC3986), aby informovala o použití SIP WebSocket podprotokolu pro přenos.

```
transport-param  =  "transport="
                    ( "udp" / "tcp" / "sctp" / "tls" / "ws"
                    / other-transport )
```

Standardní port pro používání SIP URI schématu “sip” a přenosového parametru “ws” je 80, a standardní port pro používání SIP URI schématu “sips” a přenosového parametru “wss” je 443.

4. Návrh vlastního řešení

4.1. Průzkum, porovnání a výběr z dostupných knihoven

WebSocket je efektivní a rychlá technologie a tak existuje spousta knihoven (též implementací), které jsou součástí prohlížečů, anebo přímo určené pro vývojáře vlastních aplikací. Problémem je, že většina z nich není volně dostupná a nepřestavují svobodný software.
Mezi licence, které za určitých podmínek (někdy malých podmínek) umožňují zasahovat do kódu, využívat jej nebo dále upravovat patří například licence GNU LGPL nebo GPL, BSD, MIT, EPL. GNU GPL patří mezi copyleftové licence [8], což znamená, že kód je možné svobodně upravovat i šířit. Podmínkou je, že právě tuto svobodu musím poskytovat dále(zachovat ji). GNU LGPL také patří mezi licence svobodného softwaru, ale představuje kompromis mezi silně copyleftovými licencemi a permisivními licencemi. Dílo, které je vytvořeno pod licencí LGPL lze využívat (linkovat) programem, který nemá licenci (L)GPL, a který může být jak svobodný software, tak software proprietární (komerční). Pod permisivní licence spadá BSD a MIT. Např. BSD má nízké požadavky, dovoluje komerční použití a dokonce jej lze použít pro komerční licenci. Lze jej využít v propietárním softwaru.
EPL představuje slabší copyleftovou licenci. Zde je možné některé díla licencovat.

Přehled projektů a knihoven pro WebSocket protokol ukazuje následující tabulka.

projekt	Knihovna (klient)	Knihovna (server)	Licence	Jazyk implementace	Jazyk API
Realtime.co	Ano	Ano	komerční	C	mimo jiné JavaScript / .NET / Java / PHP
lightstreamer	Ano	Ano	komerční, freeware	Java	mimo jiné JavaScript / Java / Android
jetty	Ano	Ano	EPL, Apache 2	Java / Android	Java / javax.servlet
resin	Ne	Ano	GPL	Java / C	Java
libwebsocket	Ano	Ano	LGPL(v2)	C / Sokets	C
jWebSocket	Ano	Ano	LGPL	Java	Java
Kaazing	Ano	Ano	komerční	Java	JavaScript / .NET

Tab. 1 Výpis některých dostupných WebSocket knihoven na internetu

Protože je záměrem implementace v jazyku Java, byla pro realizaci řešení zvolena knihovna projektu jWebSocket [14]. Tento projekt nabízí API klienta i serveru v Javě a hlavně je pod licencí svobodného softwaru.
Pro realizaci SIP části byl zvolen projekt JAIN-SIP [15]. Jedná se o projekt, který poskytuje API v Javě. Vlastností tohoto projektu je, že představuje referenční implementaci dle RFC 3261. Jeho velkou výhodou je velmi dobrá dokumentace.

4.2 Struktura aplikace

Aplikace se skládá z několika tříd, z nichž některé implementují určitá rozhraní. Implementace těchto rozhraní je nutná z důvodu umožnění komunikace a předávání dat mezi jednotlivými instancemi daných tříd. Celkovou strukturu aplikace ukazuje obr.2. Samotnou bránu, jejímž úkolem je překlad signalizačních zpráv, tvoří tyto třídy:

- ControlLayer (řídící vrstva)

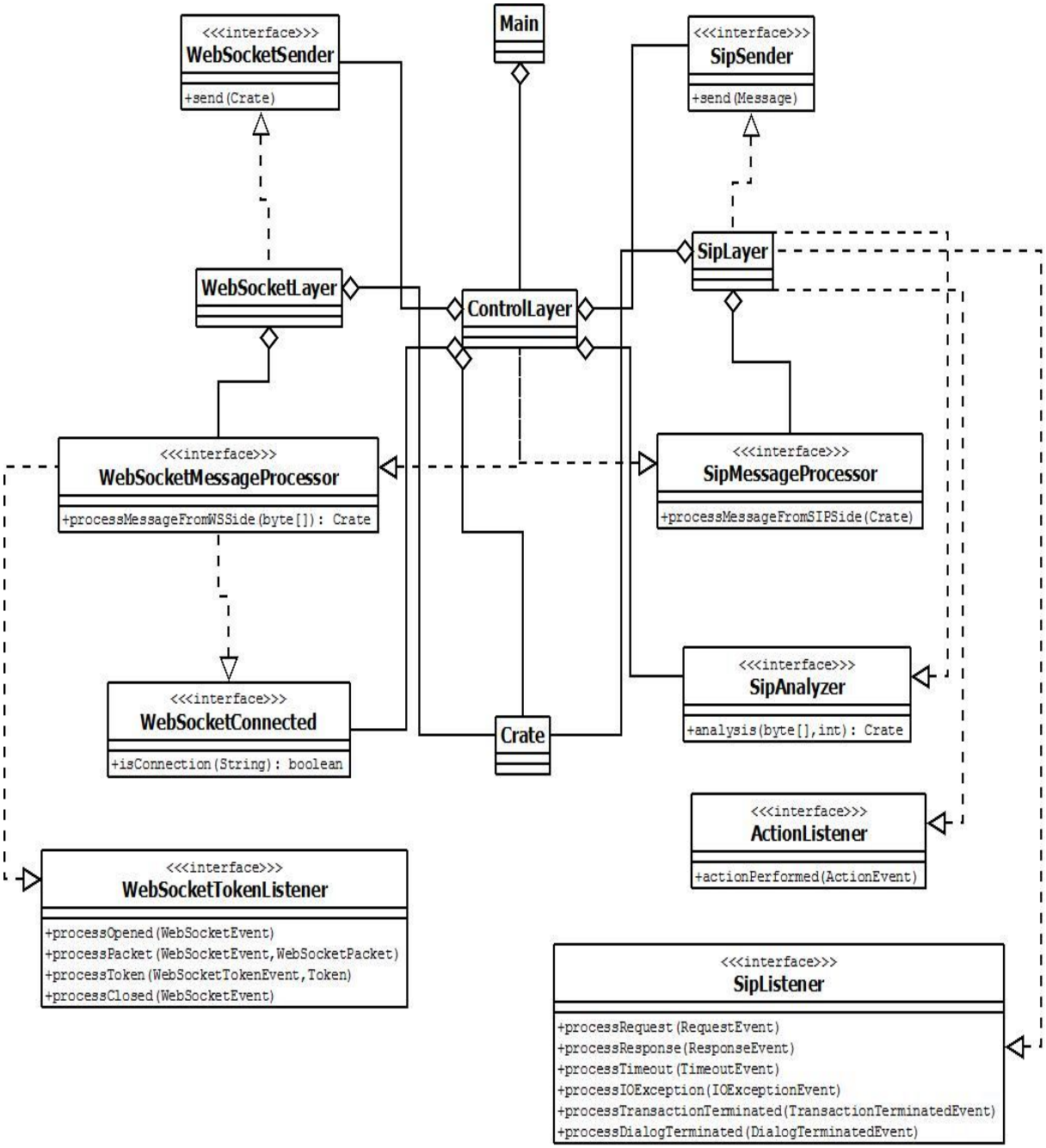
- SipLayer (SIP vrstva)
- WebSocketLayer (WebSocket vrstva)
- Main (hlavní)
- Crate („přepravka“, bedna)

Main je spouštěcí třída. Obsahuje potřebnou statickou metodu main() pro spuštění aplikace. Také se v ní nachází potřebný kód, který umožňuje samotné spuštění implementace jWebSocket. Tento kód umožní mimo jiné použití důležité třídy – JWebSocketFactory, která poskytuje WebSocket server, který je nutný pro obsluhu WebSocket (WS) klientů. Posledním úkolem spouštěcí třídy Main je vytvoření instance třídy ControlLayer.
Třída ControlLayer je prostředníkem mezi třídami SipLayer a WebSocketLayer. Instance této třídy, kterou vytváří třída Main, vytváří instance tříd SipLayer a WebSocketLayer. Jejím hlavním úkolem je předávání SIP zpráv od instance třídy SipLayer instanci třídy WebSocketLayer a předávání SIP zpráv, které jsou obsaženy ve WebSocket paketech předané od instance třídy WebSocketLayer instanci třídy SipLayer.
Třída WebSocketLayer má na starosti komunikaci brány založenou na WebSocket protokolu a obsluhuje WS klienty, kteří posílají WS pakety, obsahující SIP zprávy, jež jsou určené pro SIP klienty. Stejně tak posílá těmto WS klientů přeložené SIP zprávy od SIP klientů.
Třída SipLayer zajišťuje komunikaci založenou na SIP protokolu a obsluhuje SIP klienty, kteří posílají SIP zprávy, jenž jsou určené pro překlad na WS pakety, které budou poslány WS klientům. Stejně tak posílá těmto SIP klientům zprávy poslané WS klienty (ale již přeložené na SIP zprávy).
Třída Crate(„bedna“, „přepravka“) slouží k předávání potřebných dat a informací mezi jednotlivými třídy tvořící bránu.

Mezi rozhraní, která tvoří bránu patří:

- SipAnalyzer
- SipMessageProcessor
- SipSender
- WebSocketConnected
- WebSocketMessageProcessor
- WebSocketSender

Implementaci rozhraní SipAnalyzer požaduje třída ControlLayer po třídě tvořící SIP část brány (v našem případě třída SipLayer). Myšlenkou tohoto rozhraní je umožnění analýzy přijatého toku dat (pole bytů), které by měly představovat SIP zprávu. Jedná se tedy o důležitou část překladu.
Implementaci rozhraní SipMessageProcessor požaduje třída SipLayer. Ta slouží k umožnění komunikace ve směru od třídy SipLayer k třídě ControlLayer. Tedy, aby bylo možné přijatou zprávu od SIP klienta předat dále.
Podmínku pro opačnou komunikaci tedy ve směru od třídy ControlLayer ke třídě SipLayer v sobě obsahuje rozhraní SipSender. Slouží k odeslání přeložené SIP zprávy přijaté od nějakého WS klienta. Tato zpráva je pak zaslána požadovanému SIP klientovi.
Pokud byla přijata SIP zpráva od SIP klienta, která byla zanalyzována a předána třídě ControlLayer, pak je potřeba její předání třídě WebSocketLayer, za účelem jejího poslání WS klientovi. Protože tento klient nemusí mít navázané spojení s bránou, a protože tento klient se u brány ještě nemusel registrovat vzniká zde přirozený požadavek zjistit, zda-li je možné zprávu vůbec předat (zda-li má její předávka třídě WebSocketLayer nějaký smysl). Z tohoto důvodu je zde rozhraní WebSocketConnected. Implementaci požaduje třída ControlLayer po WebSocket části brány (v našem případě po třídě WebSocketLayer).
Pokud spojení existuje (brána má v paměti spojení na daného WS klienta), potom je potřeba tuto zprávu poslat. To zajišťuje rozhraní WebSocketSender. Implementaci tohoto rozhraní požaduje třída ControlLayer po třídě WebSocketLayer, která poslání přeložené zprávy, zajišťuje.
Komunikaci v opačném směru (tj. od třídy WebSocketLayer k třídě ControlLayer) umožňuje rozhraní WebSocketMessageProcessor. Třída, která implementuje toto rozhraní (v našem případě ControlLayer) bude informována o příchodu WS zprávy, jež by v sobě měla obsahovat SIP zprávu.



Obr. 2 Struktura aplikace

4. 3 Implementace brány

4.3.1 Třída WebSocketLayer

Bránu lze rozdělit do 3 větších celků, jež zajišťují postupný překlad. Tyto 3 celky tvoří třídy SipLayer, ControlLayer a právě třída WebSocketLayer. Celky jež představují třída SipLayer a WebSocketLayer popisují příjem a obsluhu na daném „rozhraní“. Tímto „rozhraním“ je myšlena komunikace pomocí jednotlivých protokolů, mezi kterými probíhá překlad. Tyto 2 třídy tedy představují pomyslné vstupy a výstupy brány, mezi nimiž leží výše zmiňovaný „prostředník“ a třetí celek - třída „ControlLayer“. Tato podkapitola popisuje realizaci „WS rozhraní“, které představuje právě třída WebSocketLayer.

4.3.1.1 Hlavička třídy

Třída WebSocketLayer implementuje 3 rozraní, a to:

- WebSocketServerTokenListener
- WebSocketConnected
- WebSocketSender

Splnění požadavku rozhraní WebSocketConnected zajišťuje implementace metody boolean isConnection(String). Tato metoda stejně jako ostatní metody je popsána v podkapitole metody. Rozhraní WebSocketSender představuje implementace metody public void send(Crate). Datový typ Crate je detailně popsán v podkapitole Třída Crate. Rozhraní, které tato třída implementuje a které není přímo součástí samotné brány, avšak její implementace představuje jednu z důležitých částí brány je WebSocketServerTokenListener. Díky implementaci tohoto rozhraní je brána od Tokenového serveru informována mimo jiné o příchodu WS paketů a informacích týkajících se jednotlivých WS spojení. Tokenový server představuje jednu z komponent knihovny jWebSocket. Umožňuje oznámení o příchodu datové struktury zvané token, anebo oznámení o příchodu samotného WS paketu, jež může v sobě nést právě token. Tokeny nejsou v této práci používány. Tedy implementaci rozhraní WebSocketServerTokenListener požaduje právě Tokenový server, který požaduje implementaci metod:

- void processClosed(WebSocketServerEvent)

- void processOpened(WebSocketServerEvent)
- void processPacket(WebSocketServerEvent, WebSocketPacket)
- void processToken(WebSocketServerTokenEvent, Token)

Metoda processToken(WebSocketServerTokenEvent, Token) není v bráně využívána. Ta představuje WebSocket komunikaci pomocí tokenů. Těchto tokenů je několik druhů a ani jeden z nich není v bráně využíván. Proto zde nejsou popisovány. Podrobné informace lze nalézt v [13] a případné zájemce je možné odkázat na stránky projektu jWebSocket[14].

4.3.1.2 Atributy třídy

Atribut je členská proměnná dané třídy. Atributy Představují v podstatě paměť objektu. Atribut logger slouží pouze pro informační výpis do konzole. Pro zpřehlednění výpisu v konzoly se k němu používá atribut clas. Tyto atributy figurují i v ostatních třídách brány.

- controlL představuje odkaz na instanci třídy ControlLayer, která instanci třídy WebSocketLayer vytvořila.
- serverT v sobě obsahuje odkaz na Tokenový server. Je to výchozí server, který brána používá ke komunikaci s WS klienty. Brána využívá pouze jeho „paketové“ části. „Tokenová“ část není využívána.
- connectIdentifWS je mapa s typovými parametry String a WebSocketConnector, slouží k zapamatování si spojení brány s daným WS klientem. Potřebné údaje o tomto spojení v sobě obsahuje právě datový typ zde označovaný jako WebSocketConnector. String zde představuje odpovídající identifikátor tohoto spojení, podle kterého lze určit komu směrovat danou odpověď nebo případnou žádost.
- connectIdentifWSInfoOfSecurity. Brána do příchozích zpráv přidává pole(své záhlaví) Via. Záhlaví Via se používá pro pozdější směrování odpovědi. Součástí tohoto záhlaví Via je i tzv. transportní protokol-tj. protokol, který je užit pro přenos SIP zpráv. V našem případě se jedná o spojení, které je typu WS nebo WSS. WSS představuje šifrovanou formu (SSL, TLS) websocket spojení. Aby brána věděla, jaký typ spojení se používá a mohla jej do záhlaví Via doplnit, je tato informace pro každé spojení uložena právě v tomto atributu.

4.3.1.3 Metody třídy

Úlohou konstruktoru je inicializace části brány, která má za úkol obsluhu a komunikaci s WS klienty. Spouští se zde sada nástrojů podporující WS komunikaci. Tyto nástroje jsou spuštěny pomocí příkazu JWebSocketFactory.start(). Pomocí této třídy je získán odkaz na Tokenový server a třída WebSocketLayer je přidána jako „posluchač“ (listener) serveru, který bude tuto třídu informovat (její instance) o příchozích událostech. Následně je nutné server spustit.

Metody:

- void boolean isConnection(String). Úkolem metody je zjištění existence daného WS spojení podle identifikátoru, který je metodě předán v jejím argumentu. Pokud spojení existuje, vrací hodnotu true.
- void send(Crate). Umožňuje poslat SIP zprávu adresovanému WS klientovi. Vstupním argumentem je datový typ (DT) Crate. Ten mimo jiné obsahuje úložiště pro obecný typ SIP zprávy označované jako Message a potřebné gettery (metody sloužící pro vyčtení hodnot jednotlivých atributů třídy). Tato SIP zpráva, která zahrnuje jak typ žádost (Request) tak odpověď (Response), je zprávou určenou k poslání danému WS klientovi. Zpráva je nejprve převedena na pole bytů. V aplikaci je použito kódování UTF-8, ale je možné i kódování ASCII. Následně je získán z DT Crate potřebný identifikátor spojení, podle kterého je nalezen potřebný WebSocketConnector, který obsahuje veškeré informace o daném WS spojení. Z pole bytů je vytvořen WS paket, zde DT RawPacket. Tento paket je následně pomocí informací o daném WS spojení Tokenovým serverem poslán adresovanému WS klientovi.
- void processClosed(WebSocketServerEvent). Prostřednictvím této metody Tokenový server oznamuje bráně ukončení daného spojení s určitým WS klientem. Proto úlohou této metody je z paměti brány, kterou představuje atribut connectIdentifWS, vymazat záznam o daném spojení.
- void processOpened(WebSocketServerEvent. Tokenový server oznamuje navázání nového spojení s určitým WS klientem. Protože nelze v této fázi zjistit, jaký SIP identifikátor danému WS klientovi náleží nemá metoda v bráně žádnou důležitou funkci.
- void processPacket(WebSocketServerEvent, WebSocketPacket). Je společně s metodou send(Crate) nejdůležitější metodou této části brány. Pomocí této metody je Tokenovým serverem předán bráně došlý paket od konkrétního WS klienta. Ten je identifikován WS spojením, které je dané právě WebSocketConnector-em. Nejprve

jsou získána užitečná data z WS paketu, která jsou ve formě bytového pole. Tato data by po překladu měla tvořit SIP zprávu. Dané pole bytů je předáno třídě ControlLayer (jako vstupní argument metody Crate processMessageFromWSSide) a ta se postará o zajištění analýzy tohoto pole bytů, představující data. Metoda vrací DT Crate. Pokud tento datový typ (v Javě odkaz) neobsahuje null (odkaz neodkazující nikam) a zároveň informace uložená v tomto DT říká, že se jedná o zprávu typu odpověď znamená to, že není potřeba ukládat informace o daném WS spojení. To je dáno tím, že zpráva typu odpověď (Reponse) přichází pouze v reakci na žádost. A protože tato žádost byla úspěšně předána (došla odpověď) znamená to již existující spojení a tedy uložený potřebný záznam. V takovém případě metoda končí svou funkci. Další situace nastává, jestliže dané pole bytů nepředstavovalo SIP zprávu. V tomto případě úložiště (Crate) pro identifikátor bude obsahovat null (nebude nikam odkazovat). Potom je odeslán danému WS klientovi paket informující jej o „špatné“ zprávě. V této situaci metoda opět končí svou funkci. Jedná-li se o zprávu typu žádost, potom se zjišťuje, jestli už má brána uložené potřebné informace o tomto WS spojení, pokud ano, metoda končí. Pokud ne je do atributu connectIdentifWS uložen WebSocketConnector na daného WS klienta, který je identifikován daným identifikátorem (fromIdentif). Následně se ještě uloží informace o jaký typ spojení se jedná do atributu connectIdentifWSInfoSecurity.

- void processToken(WebSocketServerTokenEvent, Token). Tato metoda není bránou používána.
- boolean isSecurity(String). Metoda slouží ke zjištění o jaký typ spojení se jedná. Vrací true v případě, že spojení je typu wss (SSL, TLS).

4.3.2 Třída ControlLayer

Úkolem třídy ControlLayer je předávání zpráv mezi WS stranou brány a SIP stranou brány. Poskytuje služby třídě WebSocketLayer a SipLayer.

4.3.2.1 Hlavička třídy

Tato třída implementuje 2 rozhraní - WebSocketMessageProcessor a SipMessageProcessor. Toho dosahuje pomocí metod Crate processMessageFromWSSide(byte[]) a void processMessageFromSiPSide(Crate).

4.3.2.2 Atributy třídy

Mimo informačního a ladicího výpisu, třída obsahuje odkazy na třídy WebSocketLayer a SipLayer, s kterými úzce spolupracuje, a které při spuštění programu vytváří.

4.3.2.3 Metody třídy

Úkolem konstruktoru, který volá spouštěcí metoda main() je zajištění veškeré potřebné inicializace brány. Takže se zde právě vytváří instance obsluhující WS stranu a SIP stranu brány. Při tomto procesu je jim v argumentech jejich konstruktorů předán odkaz na instanci třídy ControlLayer. Skončením metod main() ze třídy Main a konstruktoru třídy ControlLayer je brána připravena k činnosti.
Metody:

- Crate processMessageFromWSSide(byte). Tato metoda převezme pole bytů od třídy WebSocketLayer. Toto pole bytů se nechá podrobit analýze, o kterou se stará třída SipLayer, neboť ta má ve své kompetenci v podstatě vše, co souvisí se SIP protokolem. Tuto analýzu volá pomoci metody třídy SipLayer (Crate analysis(byte[], int)). Výsledkem je vrácená „přepravka“ - DT Crate. Pokud tato přepravka není vrácena (vrácen null), znamená to, že pole bytů představovalo špatný formát SIP zprávy, anebo se vůbec o SIP zprávu nejednalo. V takovém případě vrací tato metoda null. Pokud se jednalo o řádnou SIP zprávu, pak je tato zpráva předána k odeslání třídě SipLayer a metoda vrací DT Crate.
- Void processMessageFromSIPSide(Crate). Podobně jako metoda processMessageFromWSSide(byte[]), která je volána třídou WebSocketLayer, je tato metoda volána třídou SipLayer. Oznamuje tím, že má SIP zprávu, která je určena k překladu. Nejprve se zjistí, zda-li se jedná o zprávu typu odpověď nebo žádost. Pokud se jedná o odpověď a je zjištěno, že spojení na daného WS klienta existuje je zpráva předána třídě WebSocketLayer k poslání. Pokud dané spojení neexistuje brána nedělá nic a zpráva je zahozena. Jedná-li se o žádost, potom se nejprve opět zjišťuje

existence daného WS spojení, existuje-li je zpráva opět předána k poslání. Pokud neexistuje brána pošle danému SIP klientovi odpověď s kódem 404, ve které klienta informuje o nenalezeném klientovi.

- boolean isSecurity(String). Slouží ke zjištění, zda-li se jedná o ws nebo wss spojení. Je zde z důvodu pole Via, které je přidáváno ve třídě SipLayer. Vrací true v případě wss, false i v případě, že spojení neexistuje – zpráva je stejně zahozena.

4.3.3 Třída SipLayer

4.3.3.1 Hlavička třídy

Třída SipLayer implementuje rozhraní SipListener, ParseExceptionListener, SipAnalyzer a SipSender. Rozhraní SipAnalyzer a SipSender implementuje pomocí metod Crate analysis(byte[], int) resp. void send(Message). Rozhraní ParseExceptionListener není přímou součástí brány. Jeho implementace je zde z důvodu použití parsovacího nástroje projektu JAIN-SIP. Také rozhraní SipListener není přímou součástí brány, ale pro samotnou bránu se jedná o důležité rozhraní. Tím, že třída SipLayer implementuje toto rozhraní, lze potom tuto třídu pokládat za posluchače a má mimo jiné možnost býti informovována o přijetí SIP zpráv od SIP klientů.

4.3.3.2 Atributy třídy

Třída má samozřejmě odkaz na třídu ControlLayer, se kterou při překladu neustále spolupracuje. Obsahuje také atributy pro výpis do konzole, ať už se jedná o výpis ladící nebo informační.

- identifOfSIPC. Pokud SIP klient posílá žádost, která je určena k překladu, je s daným SIP klientem provozována transakce. Tato transakce je z pohledu brány serverovského typu, z pohledu klienta klientského typu. Serverovská transakce (ServerTransaction – ST) obsahuje potřebné informace k poslání případné zpětné SIP zprávy – prozatímní odpověď (provisional Response), odpověď. Transakce má jedinečný identifikátor (napříč prostorem a časem pro všechny žádosti poslané tzv. User Agent, který je obsažen v daném SIP klientovi), který se jmenuje „BranchID“. Protože SIP klient může provozovat více spojení – posílat zprávy více WS klientům – je potřeba si jeho

jednotlivé ST pamatovat. Z tohoto důvodu je zde atribut identifOfSIPC. Ten je složen ze dvou map. První mapa, která tvoří tento atribut obsahuje klíč-identifikátor SIP klienta(fromIdentif) a jemu přiřazenou(mapovanou) hodnotu-druhou mapu(mapu transakcí). Tato druhá mapa obsahuje jednotlivé transakce daného SIP klienta, kde jako klíč je branchId(tedy identifikátor dané serverovské transakce) a jemu přiřazenou hodnotu –konkrétní serverovskou transakci (DT ServerTransaction).

- ipAddress a port. Jedná se o IP adresu počítače, na kterém brána běží. Port představuje výchozí port SIP protokolu – tedy 5060. Brána pro SIP klienty naslouchá na portu 5060.
- SF, SS, vlastnostiSS. DT SipFactory představuje jednotný přístup k nástrojům implementace JAIN-SIP. DT SipStack(SS) reprezentuje danou implementaci. Pomocí DT Properties(poslední z těchto třech atributů) jsou nastavovány vlastnosti SS. SS musí mít minimálně jméno.
- udpLP. DT ListeningPoint představuje javovskou reprezentaci soketu(ip adresa + port). Dle názvu reprezentuje „místo naslouchání".
- SP. DT SipProvider. Tato třída má v knihovně JAIN-SIP na starost zprávy (příjem, posílání). Právě u něj se třída SipLayer registruje jako posluchač. Třída SipProvider informuje třídu SipLayer o příjmu SIP zpráv nebo stavech transakcí.
- creatorMessage, creatorHeader, creatorAddress. Slouží pro vytváření SIP zpráv, jejich jednotlivých záhlaví a adres, které určují příjemce nebo odesílatele.
- viaHeaderGateway a viaHeaderGatewayBack. Jedná se o pole záhlaví Via, které reprezentuje bránu. Výběr jednoho z těchto dvou typů polí závisí na směru, ze kterého přišla žádost(tj. zda-li žádost poslal WS klient nebo SIP klient). Doplnění pole Via brány je závislé na tom, zda-li byl použit jako přenosový protokol SIPu protokol UDP nebo WS (resp. WSS).
- uncompleteST. V praxi může nastat situace, že na danou žádost, kterou posílá určitý SIP klient, nepřijde okamžitě konečná odpověď (např. 200 OK). Častým případem jsou prozatímní odpovědi (např. 180 Riging). Nelze vyloučit ani případ, kdy konečná odpověď nepřijde vůbec. V bráně ale stále existují záznamy o nevyřízených serverovských transakcích. Tyto transakce je zapotřebí po určité době vymazat.
  V bráně běží neustále časovač. Jakmile časovač vyprší, brána se podívá do záznamu (atribut identifOfSIPC) a zaznamená si do atributu uncompleteST momentálně nevyřízené transakce. Tyto transakce mohly vzniknou v době začátku intervalu běhu časovače, anebo na jeho konci. Proto nejsou dané transakce ještě vymazány. Než

znovu vyprší časovač, mohou být tyto transakce úspěšně vyřízeny. Pokud časovač vyprší znovu. Transakce jsou přesunuty do atributu toRemoveST.

- toRemoveST. Představuje záznamy o nevyřízených transakcí, které při dalším vypršení časovače budou ukončeny a vymazány. Před jejich odstraněním jsou vymazány záznamy z atributů identifOfSIPC a uncompleteST. Na závěr jsou identifikátory jednotlivých transakcí vymazány i z atributu toRemoveST.

4.3.3.3 Metody třídy

Souhrnně lze říci, že úkolem konstruktoru je definice a inicializace jednotlivých atributů třídy. Jsou vytvořeny nebo získány jednotlivé potřebné nástroje implementace JAIN-SIP. Tyto nástroje představují atributy třídy.

Metody:

- Crate analysis(byte[], int). Úkolem metody je analýza bytového pole dat, které by měly představovat SIP zprávu. Druhý argument metody určuje část brány, která si nechává analýzu provádět. Možnosti jsou 2, buď se jedná o analýzu-parsování pole bytů dat, které byl obsažen ve WS zprávě, anebo se jedná o kontrolu zprávy přijatou od SIP klienta. Nejprve je vytvořena instance parseru. Následuje samotné parsování. Pokud je parsování neúspěšné, nejedná se o SIP zprávu, nebo SIP zpráva neobsahuje všechny potřebné náležitosti. Je-li parsování úspěšné, výsledkem je zpráva DT Message. DT Message je obecná SIP zpráva, ze které dědí zprávy typu žádost a odpověď. Ze zprávy je získáno pole From, z kterého se vybírá tvz. Sip-URI. Jedná se o identifikátor identifikující odesílatele. Tento identifikátor převedený na řetězec využívá brána při identifikaci jednotlivých klientů. Spolu s identifikátorem To (také ve tvaru Sip-URI) identifikující adresáta, jsou v bráně určováni adresování klienti, jednotlivé serverovské transakce i jednotlivá WS spojení. Tyto identifikátory jsou tak jednou z klíčových položek brány. V metodě analysis(byte, int) se následně zjistí zda-li se jedná o zprávu typu odpověď nebo žádost. Tato skutečnost je pak předána v DT Crate. V případě, že se jedná o zprávu typu žádost, která je směřována od WS klienta k SIP klientovi, potom probíhá proces přidání položky Via brány. Při tomto procesu je generován náhodný řetězec reprezentující branchId.
- void send(Message). Účelem metody je poslání SIP zprávy adresovanému SIP klientovi. Podle toho, zda-li se jedná o SIP zprávu typu žádost nebo odpověď je buď

přidáno pole Via brány, nebo je pole Via brány odebráno. Jedná-li se o odpověď, znamená to, že jsou zpáteční cestou postupně odstraňovány jednotlivé pole Via, včetně té, které patří bráně. Metoda nejprve zjistí, zda-li dotyčná zpráva pole brány skutečně obsahuje. Pokud ne, není tato zpráva určena bráně a je zahozena-ignorována. Pokud pole Via brány obsahuje, je odstraněno. Následně metoda zjistí podle identifikátoru (který je tvořen záhlavím From) adresovaného SIP klienta a jeho serverovskou transakci-spojení brány s ním. Danou serverovskou transakci rozpozná podle parametru branch (branchId). Ten je získán z nyní nejvýše umístěného pole Via. Podle toho je nalezena v mapě právě odpovídající serverovská transakce. Odpověď je poslána přes danou serverovskou transakci. Jedná-li se o zprávu typu žádost, vytváří brána novou klientskou transakci(KT). Ta je vytvořena z dané žádosti a přes tuto KT je daná žádost poslána.

Metoda void send(Message) řeší mazání úspěšně vyřízených serverovských transakcí (zde je potřeba rozumět pod pojmem úspěšných ty, u kterých dojde nějaká konečná odpověď). Pokud je daná ST v některém ze stavů, z nichž může přejít do stavu ukončeného (Terminated) , anebo je přímo v ukončeném stavu vymaže daný záznam z atributu identifOfSIPC. Nepředstavuje-li přijatá zpráva konečnou odpověď je záznam ST uložen do atributu uncompleteST.

- void processRequest(RequestEvent). Metoda je volána, když třída SipProvider přijmula příchozí zprávu, která je typu žádost a předává ji svému posluchači. V reakci na tuto žádost je vytvořena nová serverovská transakce (ST) s daným SIP klientem. Brána přidává parametr received do nejvyššího pole Via. To je z toho důvodu, aby bylo jasné, kam se bude posílat odpověď. Parametr received brána přidává pokaždé, i v případě, že host obsahuje ip adresu. Přes vytvořenou ST je klientovi poslána prozatímní (informační) odpověď – kód zprávy 100 - Trying. Metoda kontroluje náležitosti této žádosti pomocí metody Crate analysis(byte[], int). Výhodou této kontroly je, že vzniklá ST se neřídí následným přidáváním pole Via brány a neovlivňuje tak tuto ST, která nesmí vědět o přidaném poli Via, neboť je vedena s daným SIP klientem. Následně jsou získány oba potřebné identifikátory (pole From a To). Identifikátor From potom slouží k identifikování SIP klienta, který čeká na odpověď od WS klienta, kterému posílá zprávu. Identifikátor To slouží k identifikování WS klienta, kterému je zpráva určena. Ze ST je získán parametr (identifikátor) branch. Nyní se zjišťuje, zda-li už SIP klient provozuje nějaké transakce s bránou. Pokud ne, je vytvořen záznam v mapě uchovávající jednotlivé informace

daných SIP klientů – atribut identifOfSIPC. Následně je do této mapy přidána k danému klientovi mapa, která uchovává jeho ST a do ní je vložen odpovídající záznam. Pokud už klient s bránou komunikoval, je přidána pouze nová ST identifikována parametrem branch. Metoda dále vytvoří nový branch (brány), který je potřeba pro přidání pole Via brány do příchozí žádostí. Nyní se využívá metody boolean isSecurity, jenž má své jádro ve třídě WebSocketLayer. Podle toho je do pole Via přidáno odpovídající schéma (ws, wss) o použitém protokolu. Zpráva je potom předána třídě ControlLayer.

- void processResponse(ResponseEvent). Metoda je volána pokud SipProvider předává příchozí odpověď. Následuje kontrola, zda-li se jedná o odpověď určenou pro zpracování bránou. Pokud ano, obsahuje záhlaví Via dané zprávy mimo jiné pole brány. To by mělo býti umístěno nejvýše v daném záhlaví. Pokud ne, zpráva není určena pro bránu a je ignorována. Jedná-li se o zprávu určenou pro bránu k překladu, je odstraněno pole Via odpovídající bráně. Následně je zpráva předána třídě ControlLayer dále ke zpracování.
- Response createResponseFromReq(Message, int). Slouží k vytvoření odpovědi z předané žádosti. Odpověď je typu dle předaného kódu (100, 200 apod.). Brána tuto metodu využívá při posílání prozatímní odpovědi informující o zpracování daného SIP klienta.
- void actionPerformed(ActionEvent). Metoda řeší mazání neúspěšných ST po uplynutí časovače. Povel k mazání představuje záznam v atributu toRemoveST. ST, které se nachází po uplynutí časovače v tomto atributu jsou ukončeny a odstraněny z atributů identifOfSIPC a uncompleteST.

4.3.4 Třída Crate

Tato třída představuje „přepravku“ dat. Slouží k uložení potřebných zpráv, které si jednotlivé třídy brány předávají mezi sebou. K atributům, které představují předávanou zprávu, identifikátor jednotlivých klientů a informaci popisující typ zprávy jsou přidány přístupové metody (tzv. gettery). Z důvodu bezpečnosti (aby nedošlo k nežádanému přepisu) lze data do jednotlivých atributů uložit pouze přes konstruktor (tj. třída nemá settry – nastavovací metody). Jedná se tedy o neměnitelný objekt (immutable object).

5. Testování brány

Aby bylo možné demonstrovat práci brány a zároveň otestovat její správnou činnost, byla vytvořena aplikace zvaná „MultipleOfScenario“. Název naznačuje, že se jedná o více scénářů, které aplikaci testují v různých situacích. Tedy třída MultipleOfScenario obsahuje spouštěcí metodu aplikace. Tento scénář využívá 2 nové třídy. Tyto třídy reprezentují jednolivé WS klienty, kteří si nechávají přeložit WS zprávy na SIP zprávy. A SIP klienty, kteří využívají opačného překladu u brány. WS klienti jsou vytvářeni jako instance třídy GeneratorWS_ActiveSide (resp. GeneratorWSAS_provisional) a SIP klienti jsou vytvářeni jako instance třídy GeneratorSIP_PassiveSide. Ke třídě GeneratorWS_ActiveSide náleží ještě třída NecessarySipForWSKlient. Názvy jsou odvozeny z faktu, že k tomu, aby byla možná vůbec nějaká komunikace, musí se WS klient „registrovat“ (posláním WS zprávy určenou pro překlad a její následné poslání libovolnému SIP klientovi) u brány (proto aktivní strana). Aby byl WS klient schopný vytvořit nějakou SIP zprávu, používá k tomu třídu NecessrySipForWSKlient. Tato testovací aplikace je určena pouze pro ověření práce brány, proto nesplňuje všechny požadavky kladené na SIP klienty.

5.1. Třída GeneratorWS_ActiveSide a GeneratorWSAS_provisional

Mezi atributy třídy patří atribut DT BaseTokenClient. Představuje WS klienta, který navazuje spojení s bránou a tedy jejím WS serverem. Důležitým atributem je také DT NecessarySipForWSKlient, který je popsaný dále. Dále si instance třídy pamatují své „jméno“, které je dáno názvem nickKlient a je zadáváno při volání konstruktoru třídy.

5.1.1 Metody třídy

V konstruktoru probíhá vytvoření BaseTokenClient-a. Instance třídy je přidána jako posluchač toho klienta. Aby mohla být přidána jako posluchač a mohla být informována o příchozích paketech a stavech spojení, musí tato třída implementovat rozhraní WebSocketClientTokenListener. V konstruktoru následuje „otevření“ spojení s WS serverem a tedy bránou. Zde se využívá spojení typu wss a k tomu je zapotřebí port 9797, na kterém naslouchá brána. Pokud by se jednalo o ws spojení, byl by zapotřebí port 8787. Je potřeba dodat, že možnosti typu zprávy jako token, nejsou využívány.
Metody:

- void processSentPacket(String, String, int, String). Představuje prakticky totožnou metodu jako je metoda stejného názvu se vstupními parametry (String, String, String, String, int, String). Jedná se o přetížené metody, které mají stejnou funkci. Liší se pouze možností zadáním přezdívky a domény WS klienta. Doména, která je zadávána představuje v SIP-URI pouze tu část před zavináčem zvanou user, nikoliv host. Jako testovací zprávy se posílají tzv. Instant Message (metoda v SIP žádosti MESSAGE). V metodě se zadává přezdívka SIP klienta, kterému je adresována zpráva, jeho doména (opět část před zavináčem), jako host je zde IP adresa počítače, na kterém testovací aplikace běží, následuje port, na kterém SIP klient naslouchá a text zprávy, který je mu adresován. Metoda potom vytvoří danou SIP zprávu pomocí třídy NecessarySipForWSKlient (NSFWSK). Tato zpráva je převedena na pole bajtů. Z tohoto pole je vytvořen WS paket, zde DT RawPacket. Tento WS paket je následně pomocí WS klienta (BaseTokenClient) poslán.
- void processPacket(WebSocketClientEvent, WebSocketPacket). Metoda je volána v případě, že WS klient přijme paket. Provádí výpisy, které zobrazují výsledky práce překladu, tj. dá se z nich zjistit, zda-li byla daná zpráva dobře přeložena. To, že zpráva byla dobře přeložena se pozná zejména tím, že žádost v sobě bude obsahovat pole Via, které reprezentuje bránu. Navíc bez překladu by příchozí zpráva typu SIP nebyla možná. Metoda provádí převod obsahu WS paketu na SIP zprávu pomocí pomocné třídy NSFWSK. V případě, že se jedná o žádost, vytvoří odpověď, která je poslána autorovi žádosti. Pokud se jedná o odpověď, zahlásí, že nevytvářela odpověď.

Ostatní metody nejsou pro testy brány využívány.

Třída GeneratorWSAS_provisional se od třídy GeneratorWS_ActiveSide liší pouze několika řádky v metodě void processPacket(WebSocketClientEvent, WebSocketPacket). Třída GeneratorWSAS_provisional slouží k otestování zpracování prozatímní odpovědi SIP zprávy od WS klienta.

5.2. Třída GeneratorSIP_PassiveSide

Protože tato třída opět představuje práci se SIP protokolem (resp. jeho implementaci) má podobnou strukturu jako třída SipLayer brány. Například opět musí implementovat rozhraní SipListener. Má podobné atributy např. pro vytvoření zpráv DT MessageFactory nebo

naslouchací bod DT ListeningPoint – ty jsou totožné a jejich funkce je stejná. Cílem této podkapitoly je pouze popsání rozdílných funkcí, resp. odlišných atributů.

5.2.1 Atributy třídy

Atributy, které SIP strana brány neobsahuje jsou counterWSSIPC (DT Integer), memoryWSSIPClient, callIdOfDialog, cSeq_SIPClients, tagFrom a TagToRes (všichni DT ConcurrentHashMap). Všechny tyto atributy souvisejí s vedením dialogu mezi SIP klientem a daným adresovaným WS klientem. Základem je myšlenka, že daný SIP klient může vést více dialogů, tj. může komunikovat s více WS klienty. Jednotlivý WS klienti jsou identifikováni pomocí řetězce, který představuje nickName (přezdívku) daného WS klienta. K této přezdívce je přiřazeno číslo, které je potom použito pro vyhledání jednotlivých parametrů dialogu, který je veden s daným WS klientem. Číslo vzniká počítáním WS klientů. Přezdívky WS klientů musí být jedinečné.

5.2.2. Metody třídy

Struktura konstruktoru je prakticky totožná s konstruktorem třídy brány - SipLayer. Jednou z nejdůležitějších metod představuje metoda vytvářející SIP žádosti (Request createInstantMessaging(String, String, String, String, int, String, String)).
Metody:

- Request createInstantMessaging(String, String, String, String, int, String, String). Metodu lze rozdělit do dvou částí. První se dá označit jako část generující potřebné náhodné parametry a zjišťující náležitosti zprávy resp. její příslušnosti k dialogu. Druhá část představuje samotnou tvorbu zprávy. Nejprve se vygeneruje náhodný řetězec pomocí třídy UUID, jehož část je použita jako identifikátor transakce, tj. představuje parametr branch. Tento parametr se generuje při každém přístupu do metody, neboť transakce má pouze jednu žádost. Pokud je již dialog s daným klientem veden, potom se pomocí přezdívky vyhledá v odpovídající tabulce číslo identifikující daného WS klienta. Z vráceného čísla jsou postupně zjištěny parametry (resp. záhlaví) CallID, Cseq – to je inkrementováno o jedničku (protože identifikuje jednotlivé transakce v rámci dialogu) a tag z pole From, který je v rámci dialogu stejný. Není-li dialog s daným WS klientem ještě veden, potom se nejprve přidá do první z map číslo

identifikující parametry daného dialogu s daným WS klientem. Následně je vygenerován náhodný řetězec pro CallId. Opět CallId tvoří část tohoto řetězce. Část Cseq, které tvoří pořadové číslo je počítána od čísla 1000. Generování parametru tag pole From probíhá stejným způsobem pomocí třídy UUID. Nyní je vytvářena SIP zpráva. K tomu slouží třídy MessageFactory, HeaderFactory a AddressFactory. Pomocí jejich metod jsou vytvořeny povinná záhlaví, které musí SIP zpráva obsahovat. SIP zpráva je typu žádost a metoda je nastavena na MESSAGE. Mimo povinná záhlaví jsou zde ještě záhlaví udávající informace o obsahu zprávy. Posílá se prostý text (text-plain). A samozřejmě samotný text, který zadává daný SIP klient.

- void processSentMessage(String, String, String, String, String). Z předaných argumentů vytváří žádost pomocí metody createInstantMessaging, z dané SIP žádosti vytvoří novou klientskou transakci a přes tuto transakci danou zprávu odešle. Port 5060 je rezervovaný pro bránu. Pokud běží test (či jiná aplikace) na stejném počítači, pro garanci správné funkčnosti, nesmí používat port 5060. Tedy mezi zadávanými porty se nemůže vyskytovat tento port. „Přenosový“ protokol je pevně daný – UDP.
- void processRequest(Request). Metoda slouží k oznámení o příjmu SIP žádosti. Na základě této žádosti generuje odpověď s kódem 200, aby se ověřil opačný směr překladu. Vypisuje jednotlivé informace dávající přehled o zpracovaných zprávách. Zde se uplatňuje poslední daný atribut zvaný TagToRes. Představuje parametr tag u záhlaví To. Ten je doplňován autorem odpovědi na danou žádost na začátku dialogu. Pokud se jedná o „zakládání“ dialogu – žádost představuje první zprávu, je vygenerován nový parametr tag. Pokud už dialog nějakou dobu probíhá je pomocí identifikátoru zjištěn stávající tag. Z tohoto důvodu je v rámci spuštěného programu (konkrétního scénáře) možný pouze jeden dialog. Další je sice možný, ale bude mít stejný tag To.
- void processResponse(Response). Informuje pouze o příchodu odpovědi, čímž překlad končí.

5.3. Třída NecessarySipForWSKlient

Jednotlivé části kódu této třídy se shodují s předchozími popisovanými třídami v podkapitolách 4.4. a 5.2. tj. třídou SipLayer a GeneratorSIP_PassiveSide. Některé metody

jsou prakticky totožné. Třída obsahuje vše potřebné pro to, aby mohl WS klient vytvořit SIP zprávu. Částečně odlišné kódy prakticky představují pouze 2 metody a to metoda Response createResponse(byte[], int, String) a metoda processClosed(). První z obou uvedených metod, jak název napovídá, slouží k tomu, aby mohl daný WS klient odpovědět SIP klientovi. Parametry představují postupně bytové pole určené k analýze, kód odpovědi a text odpovědi. Je zde opět obsažený parser, který zjišťuje o jaký typ zprávy se jedná. Pokud se jedná o odpověď, je jenom tato skutečnost oznámena výpisem do konzole a na odpověď již není odpovězeno. Jedná-li se o žádost, je vytvořena odpověď pro otestování druhého směru překladu. Druhá metoda má pouze za úkol vyčištění použitých proměnných.

5.4. Třída MultipleOfScenario

Tato třída slouží ke spouštění testovací aplikace brány. Třída definuje celkem 6 scénářů, které slouží k ověření práce brány v jednotlivých situacích.
První scénář testuje bránu v případě, že existuje SIP klient a žádný WS klient. Protože žádný WS klient nepošle bráně žádný WS paket obsahující SIP zprávu k překladu, brána o jeho existenci nebude vědět. SIP klient se snaží poslat zprávu WS klientovi s daným SIP-URI v poli záhlaví To. Tímto se ověřuje reakce brány na případ, že uvedený adresát (WS klient) neexistuje.
Druhý scénář vystavuje bránu případu, že existují 3 WS klienti a 2 SIP klienti a 3 WS klienti se snaží posílat zprávy SIP klientům, kteří na ně pouze odpovídají. Takže se jedná o otestování jednoho směru překladu brány, kdy žádost přichází ve směru od WS klienta a SIP klient na ni odpovídá.
Třetí scénář zahrnuje situaci obou-směrné komunikace. Oproti druhému scénáři zde přibývá případ, kdy probíhá i překlad ve druhém směru, tj. žádost přichází i ve směru od SIP klienta. Podle toho, zda-li brána stihne uložit záznam o daném WS klientovi dříve, než dojde zpráva od SIP klienta adresovaná tomuto WS klientovi, bude odpovídat danému SIP klientovi zprávami typu 404, anebo pře-poslanou odpovědí od WS klienta.
Účelem čtvrtého scénáře je otestovat bránu při zvýšeném provozu. Je vytvořeno 8 WS klientů a 12 SIP klientů. Aby brána věděla o daných WS klientech, posílá každý z nich jednu zprávu určenou prvnímu ze SIP klientů. Následně každý SIP klient posílá zprávu každému z 8 WS klientů. Takže by se mělo celkově poslat 104 zpráv určených k překladu (8 od WS klientů, 8*12 od SIP klientů).

Pro první až čtvrtý scénář je nutné zadat jeden vstupní parametr. Ten představuje volený scénář. Pokud se tedy bude jednat například o použití 3.scénáře z příkazové řádky OS Windows, je při spuštění aplikace potřeba zadat: java –jar MultipleOfScenario.jar 3. Konstrukce čtvrtého scénáře, tj. WS klienti posílají zprávu prvnímu z SIP klientů a následně každý z SIP klientů posílá zprávu každému WS klientovi se používá i u scénáře 5 i 6. U pátého scénáře však v porovnání se scénářem č. 4 přibývá možnost volby počtu WS i SIP klientů, kterým se zadávají jména. Jména jsou v tomto případě brány i jako část z domén uživatelů (tedy nejen jako jejich přezdívky). Tento scénář lze spustit zadáním příkazu: java –jar MultipleOfScenario.jar 5 2 3 Alena Bob Cyril Daniel Eva, spustí 5. scénář, vytvoří 2 WS klienty, kterými jsou Alena a Bob a vytvoří 3 SIP klienty, kterými jsou Cyril, Daniel a Eva. Ti potom postupují podle popsané konstrukce posílání zpráv.
Šestý scénář se od pátého liší pouze v možnosti volit porty daných SIP klientů. Je potřeba upozornit, že porty je potřeba volit s ohledem na porty používané v OS a s ohledem na registrované porty či dobře známé. Ke spuštění z příkazové řádky OS Windows je nutné zadat příkaz: java –jar MultipleOfScenario.jar 6 3 4 5061 5062 5063 60101 Jana Katerina Pavlina Jan Jiri Lubomir Pavel, kde čísla 5061, 5062, 5063 a 60101 představují porty SIP klientů.
Sedmý scénář je mimo použití WS klientu vytvořených z GeneratorWSAS_provisional totožný se scénářem šest. U scénáře 7 jsou povoleny konečné odpovědi.
Osmý scénář se shoduje se sedmým, ale konečné odpovědi nejsou povoleny.

5.5. Testování a demonstrace práce brány

Jednotlivé podkapitoly se budou nyní věnovat pozorování funkčnosti brány. Jednotlivé situace budou popisovány z pohledu třídy MultipleOfScenario. Pro provedení testu je potřeba spustit bránu voláním statické metody ve třídě Main a spustit test voláním statické metody ve třídě MultipleOfScenario s odpovídajícími vstupními argumenty.
Zkoušet jednotlivé testy je možné až po inicializaci brány. To, že inicializace brány byla úspěšná, se pozná podle následujícího výpisu:

```
0    [main] DEBUG cz.xseife00.projekty.aplikace.brana.Main  -
=============
TRIDA: Main
METODA: <init>()
jWebSocket Ver. 1.0b8 (nightly build 20507) (32bit)
(c) 2010-2012 Innotrade GmbH (jWebSocket.org), Germany (NRW), Herzogenrath
Distributed under GNU LGPL License Version 3.0
(http://www.gnu.org/licenses/lgpl.html)
```

```
1334 [main] DEBUG cz.xseife00.projekty.aplikace.brana.ControlLayer  -
=============
TRIDA: ControlLayer
METODA: <init>()
1355 [main] DEBUG cz.xseife00.projekty.aplikace.brana.WebSocketLayer  -
=============
.
.
.
KONEC>>>>
=============
TRIDA: SipLayer
METODA: <init>()
<<<<KONEC
5117 [main] DEBUG cz.xseife00.projekty.aplikace.brana.ControlLayer  -
KONEC>>>>
=============
TRIDA: ControlLayer
METODA: <init>()
<<<<KONEC
5117 [main] DEBUG cz.xseife00.projekty.aplikace.brana.Main  -
KONEC>>>>
=============
TRIDA: Main
METODA: <init>()
<<<<KONEC
```

Tento výpis z konzole provedla brána. Je zde zachycen začátek a konec výpisu. Výpis by měl obsahovat informaci o tom, že byly postupně volány konstruktory tříd tvořících bránu (mimo třídu Crate). Tento výpis by měl končit informací, že skončil svou práci konstruktor spouštěcí třídy Main.
Nyní, když je brána připravena k své činnosti je možné spustit první ze série testů (scénářů).

5.5.1. První scénář

Jako vstupní parametr je potřeba zadat 1. Je to jediný potřebný vstupní argument v poli řetězců pro spouštěcí metodu testovací aplikace. Jako příklad se SIP klient přezdívaný „Bob“ naslouchající na portu 5061 snaží poslat SIP zprávu WS klientovi, u kterého uvádí přezdívku „WS_klient“ a zejména uživatelskou část pole To „WS_klient“. Posílaná zpráva z následujícího výpisu obsahuje text „Hello“.

```
SIP klient (Bob) posila zpravu: MESSAGE sip:WS_klient@192.168.56.1:5060 SIP/2.0
Call-ID: 3be88481234a3e93f4e6@192.168.56.1
CSeq: 1000 MESSAGE
From: "Bob" <sip:Bob@192.168.56.1:5061>;tag=b1e5a92bfc9f367d9ddc
To: "WS_klient" <sip:WS_klient@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP Bob.office.quick.cz:5061;branch=z9hG4bK5f1fd53ba2882b4d751a
Max-Forwards: 4
Content-Type: text/plain
```

```
Content-Length: 5

Hello
```

Jedná se o výpis SIP klienta do konzole. Z tohoto výpisu je v našem případě nejdůležitější pole záhlaví To, konkrétně text uvedený v uzavřených závorkách (`<sip:WS_klient@192.168.56.1:5060>`). Ten představuje výše zmíněný identifikátor jednotlivých klientů v bráně. Brána se podle tohoto identifikátoru snaží najít odpovídajícího WS klienta. Protože žádný v našem scénáři neexistuje, pošle SIP klientovi „Bob“ zprávu o nenalezení daného klienta. Tuto situaci znázorňuje další výpis z konzole:

```
DOSTAL JSEM ODPOVED SIP Klient( Bob )
Stav klientske transakce(GeneratorSIP): Completed Transaction
ODPOVED od brany:
SIP/2.0 404 Not found
To: "WS_klient" <sip:WS_klient@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP
Bob.office.quick.cz:5061;branch=z9hG4bK5f1fd53ba2882b4d751a;received=192.168.56.1
CSeq: 1000 MESSAGE
Call-ID: 3be88481234a3e93f4e6@192.168.56.1
From: "Bob" <sip:Bob@192.168.56.1:5061>;tag=b1e5a92bfc9f367d9ddc
Content-Length: 0
```

5.5.2. Druhý scénář

Nyní máme k dispozici 3 WS klienty (pod přezdívkami WS_Klient1, WS_Klient2, WS_Klient3) a 2 SIP klienty (pod přezdívkami „Bob“ a „Hurvinek“). Scénář je takový, že každý z WS klientů posílá zprávu prvnímu ze dvou SIP klientů („Bob“) a první z WS klientů navíc posílá zprávu druhému ze SIP klientů.

Pokud bychom se zaměřili na prvního z WS klientů, pak tento klient odesílá zprávu zaznamenanou následujícím výpisem.
Poznamenejme, že WS_SIP_Client je stejný výraz pro WS klienta, který posílá WS zprávy obsahující SIP zprávy a umí tyto SIP zprávy přijaté ve WS paketech zpracovat.

```
WS_SIP_Client( WS_Klient1) posila zpravu:
MESSAGE sip:Bob@192.168.56.1:5061 SIP/2.0
Call-ID: 63eed9e2f0107a86f5aa@192.168.56.1
CSeq: 1000 MESSAGE
From: "WS_Klient1" <sip:pc1@192.168.56.1:5060>;tag=2bd17d23dc645dd3b0d6
To: "Bob" <sip:Bob@192.168.56.1:5061>;tag=chyba
Via: SIP/2.0/WSS 192.168.56.1:5060;branch=z9hG4bKf15f1d7e03759d6bf1e6
Max-Forwards: 4
Content-Type: text/plain
```

```
Content-Length: 18

Ja jsem WS_Klient1
```

Adresovaný SIP klient („Bob“) přijme následující zprávu:

```
SIP_Client(Bob) obdrzel zpravu:
MESSAGE sip:Bob@192.168.56.1:5061 SIP/2.0
Call-ID: 63eed9e2f0107a86f5aa@192.168.56.1
CSeq: 1000 MESSAGE
From: "WS_Klient1" <sip:pc1@192.168.56.1:5060>;tag=2bd17d23dc645dd3b0d6
To: "Bob" <sip:Bob@192.168.56.1:5061>;tag=chyba
Via: SIP/2.0/UDP
192.168.56.1:5060;branch=z9hG4bK205d7969f66e45979cc76916a37ff465,SIP/2.0/WSS
192.168.56.1:5060;branch=z9hG4bKf15f1d7e03759d6bf1e6
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 18

Ja jsem WS_Klient1
```

Rozdíl mezi odesílanou a přijatou zprávou je poli záhlaví Via. Brána do tohoto záhlaví přidala své pole Via pro případnou odpověď. V našem případě brána naslouchá na IP adrese 192.168.56.1. Port naslouchání je pevně nastaven na port 5060, neboť ten je doporučen pro SIP protokol. Brána nepřidává parametr received, neboť znalost zdrojové IP adresy paketu je daná WS spojením.V následující moment SIP klient posílá odpověď pro otestování opačného směru zpracování bránou. SIP klient doplňuje svůj tag v poli To:

```
SIP_Client(Bob) posila odpoved WS_SIP_Clientovi( WS_Klient1):
SIP/2.0 200 OK
Via: SIP/2.0/UDP
192.168.56.1:5060;branch=z9hG4bK205d7969f66e45979cc76916a37ff465,SIP/2.0/WSS
192.168.56.1:5060;branch=z9hG4bKf15f1d7e03759d6bf1e6
CSeq: 1000 MESSAGE
Call-ID: 63eed9e2f0107a86f5aa@192.168.56.1
From: "WS_Klient1" <sip:pc1@192.168.56.1:5060>;tag=2bd17d23dc645dd3b0d6
Content-Type: text/plain
To: "Bob" <sip:Bob@192.168.56.1:5061>;tag=c7ec95f7ba14bd772c18
Content-Length: 50

Ja jsem  SIP_Client(Bob) a odpovidam Ti na zpravu!
```

Protože při putování odpovědi zpětným směrem jsou jednotlivé pole Via mazány, brána odstraní to své. WS klient přijme následující zprávu:

```
WS_Client( WS_Klient1): Dosel mi paket:
SIP/2.0 200 OK
Via: SIP/2.0/WSS 192.168.56.1:5060;branch=z9hG4bKf15f1d7e03759d6bf1e6
```

```
CSeq: 1000 MESSAGE
Call-ID: 63eed9e2f0107a86f5aa@192.168.56.1
From: "WS_Klient1" <sip:pc1@192.168.56.1:5060>;tag=2bd17d23dc645dd3b0d6
Content-Type: text/plain
To: "Bob" <sip:Bob@192.168.56.1:5061>;tag=c7ec95f7ba14bd772c18
Content-Length: 50

Ja jsem  SIP_Client(Bob) a odpovidam Ti na zpravu!
```

5.5.3. Třetí scénář

První scénář demonstroval případ, kdy brána nic neví o adresovaném WS klientovi a žádost se posílala ve směru od SIP klienta. V druhém scénáři byli naopak „aktivní“ WS klienti, kteří posílali zprávy SIP klientům. Druhý scénář tak testoval „jeden plný směr překladu“. Ten představuje překlad i s odpovědí schéma WS klient → Brána → SIP klient → Brána → WS klient. Tento směr překladu může být také v našem případě nazývaný jako „žádost od WS klienta“. Třetí scénář obsahuje i „druhý plný směr překladu“ v našem případě zvaný „žádost od SIP klienta“ a představuje schéma SIP klient → Brána → WS klient → Brána → SIP klient. Zde mimo jiné posílá jako příklad SIP klient „Hurvinek“ žádost právě v „druhém směru překladu“. Výběr z výpisu do konzole popisující tento směr:

```
SIP klient (Hurvinek) posila zpravu: MESSAGE sip:WS_Klient2@192.168.56.1:5060
SIP/2.0
Call-ID: 9aba9efc74f56615ac03@192.168.56.1
CSeq: 1000 MESSAGE
From: "Hurvinek" <sip:Hurvinek@192.168.56.1:5062>;tag=9e928afc9fd49e802fcb
To: "WS_Klient2" <sip:WS_Klient2@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP Hurvinek.office.quick.cz:5062;branch=z9hG4bK7061b33376c259b656d6
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 43

Ahoj, ja jsem SIP Klient Hurvinek. ZPRAVA_4
.
.
.
DOSTAL JSEM ODPOVED SIP Klient( Hurvinek )
Stav klientske transakce(GeneratorSIP): Proceeding Transaction
ODPOVED od brany:
SIP/2.0 100 Trying
To: "WS_Klient2" <sip:WS_Klient2@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP
Hurvinek.office.quick.cz:5062;branch=z9hG4bK7061b33376c259b656d6;received=192.168.
56.1
CSeq: 1000 MESSAGE
Call-ID: 9aba9efc74f56615ac03@192.168.56.1
From: "Hurvinek" <sip:Hurvinek@192.168.56.1:5062>;tag=9e928afc9fd49e802fcb
Content-Length: 0
.
.
```

```
.
WS_Client( WS_Klient2): Dosel mi paket:
MESSAGE sip:WS_Klient2@192.168.56.1:5060 SIP/2.0
Call-ID: 9aba9efc74f56615ac03@192.168.56.1
CSeq: 1000 MESSAGE
From: "Hurvinek" <sip:Hurvinek@192.168.56.1:5062>;tag=9e928afc9fd49e802fcb
To: "WS_Klient2" <sip:WS_Klient2@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/WSS
192.168.56.1:5060;branch=z9hG4bKef2ba7bd6ca64031b08c005403e4e08f,SIP/2.0/UDP
Hurvinek.office.quick.cz:5062;branch=z9hG4bK7061b33376c259b656d6;received=192.168.
56.1
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 43

Ahoj, ja jsem SIP Klient Hurvinek. ZPRAVA_4
.
.
.
WS KLIENT(WS_Klient2): Vytvoril jsem SIP Response:
SIP/2.0 200 OK
Via: SIP/2.0/WSS
192.168.56.1:5060;branch=z9hG4bKef2ba7bd6ca64031b08c005403e4e08f,SIP/2.0/UDP
Hurvinek.office.quick.cz:5062;branch=z9hG4bK7061b33376c259b656d6;received=192.168.
56.1
CSeq: 1000 MESSAGE
Call-ID: 9aba9efc74f56615ac03@192.168.56.1
From: "Hurvinek" <sip:Hurvinek@192.168.56.1:5062>;tag=9e928afc9fd49e802fcb
Content-Type: text/plain
To: "WS_Klient2" <sip:WS_Klient2@192.168.56.1:5060>;tag=3422a4221f79289c2fa4
Content-Length: 64

Ja jsem WS_klient (WS_Klient2),  odpovidam na zpravu. Dobry den.
.
.
.
DOSTAL JSEM ODPOVED SIP Klient( Hurvinek )
Stav klientske transakce(GeneratorSIP): Completed Transaction
ODPOVED od brany:
SIP/2.0 200 OK
Via: SIP/2.0/UDP
Hurvinek.office.quick.cz:5062;branch=z9hG4bK7061b33376c259b656d6;received=192.168.
56.1
CSeq: 1000 MESSAGE
Call-ID: 9aba9efc74f56615ac03@192.168.56.1
From: "Hurvinek" <sip:Hurvinek@192.168.56.1:5062>;tag=9e928afc9fd49e802fcb
Content-Type: text/plain
To: "WS_Klient2" <sip:WS_Klient2@192.168.56.1:5060>;tag=3422a4221f79289c2fa4
Content-Length: 64

Ja jsem WS_klient (WS_Klient2),  odpovidam na zpravu. Dobry den.
```

V posledním výpisu jsou vyobrazené zprávy týkající se „druhého směru překladu“. Brána informuje SIP klienta o zpracovávání zprávy (odpověď typu 100 Trying). Tato zpráva po svém přeložení je úspěšně přijata adresovaným WS klientem, který následně vytvoří odpověď

určenou k překladu a SIP klientovi, který posílal žádost. Brána v tomto směru přidává do svého pole Via parametr received. Poslední zpráva ukazuje příjem zprávy SIP klientem, která představuje odpověď.

5.5.4. Šestý scénář

Scénář číslo 4 představuje v porovnání se scénářem číslo 3 pouze větší počet obsluhovaných klientů, tedy větší zátěž brány. Počet klientů u scénáře 4 je pevně dán. Scénář 5 je prakticky identický se scénářem 6. Rozdíl tvoří pouze výběr portů. Společné scénářům 5 a 6 je možnost výběru počtu SIP a WS klientů, kterým jsou dávány přezdívky. Maximální počet klientů, které lze zvolit je 100 (50 SIP klientů a 50 WS klientů). Aby výpis nebyl příliš dlouhý, pro jednoduchost uveďme na závěr této kapitoly příklad pro vstupní parametry: 6 1 1 5064 Alena Bob. Protože v našem případě brána zpracovala zprávu od WS klienta Aleny dříve, bylo možné zpracovat zprávu od SIP klienta Boba. Zkrácený výpis do konzole:

```
WS_SIP_Client( Alena) posila zpravu:
MESSAGE sip:Bob@192.168.56.1:5064 SIP/2.0
Call-ID: b9412bb3a85cc51e73f1@192.168.56.1
CSeq: 1000 MESSAGE
From: "Alena" <sip:Alena@192.168.56.1:5060>;tag=1941871f580bbc54ca3a
To: "Bob" <sip:Bob@192.168.56.1:5064>;tag=chyba
Via: SIP/2.0/WSS 192.168.56.1:5060;branch=z9hG4bK943050fb454bbcede768
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 23

Tady je Alena. ZPRAVA_1
.
.
.
SIP klient (Bob) posila zpravu: MESSAGE sip:Alena@192.168.56.1:5060 SIP/2.0
Call-ID: 6c9e0b33a1259f31d868@192.168.56.1
CSeq: 1000 MESSAGE
From: "Bob" <sip:Bob@192.168.56.1:5064>;tag=26df7d71bae33aa835e3
To: "Alena" <sip:Alena@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP Bob.office.quick.cz:5064;branch=z9hG4bKe71d56a4fe17decdcf58
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 26

ZDE se hlasi Bob. ZPRAVA_2
///////// POCET POSLANYCH ZPRAV: 2 //////////////
.
.
.
ODPOVED od brany:
SIP/2.0 100 Trying
To: "Alena" <sip:Alena@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP
Bob.office.quick.cz:5064;branch=z9hG4bKe71d56a4fe17decdcf58;received=192.168.56.1
```

```
CSeq: 1000 MESSAGE
Call-ID: 6c9e0b33a1259f31d868@192.168.56.1
From: "Bob" <sip:Bob@192.168.56.1:5064>;tag=26df7d71bae33aa835e3
Content-Length: 0
.
.
.
DOSTAL JSEM ODPOVED SIP Klient( Bob )
Stav klientske transakce(GeneratorSIP): Completed Transaction
ODPOVED od brany:
SIP/2.0 200 OK
Via: SIP/2.0/UDP
Bob.office.quick.cz:5064;branch=z9hG4bKe71d56a4fe17decdcf58;received=192.168.56.1
CSeq: 1000 MESSAGE
Call-ID: 6c9e0b33a1259f31d868@192.168.56.1
From: "Bob" <sip:Bob@192.168.56.1:5064>;tag=26df7d71bae33aa835e3
Content-Type: text/plain
To: "Alena" <sip:Alena@192.168.56.1:5060>;tag=a327daf12dcdf9df1959
Content-Length: 59

Ja jsem WS_klient (Alena),  odpovidam na zpravu. Dobry den.
.
.
.
WS_Client( Alena): Dosel mi paket:
SIP/2.0 200 OK
Via: SIP/2.0/WSS 192.168.56.1:5060;branch=z9hG4bK943050fb454bbcede768
CSeq: 1000 MESSAGE
Call-ID: b9412bb3a85cc51e73f1@192.168.56.1
From: "Alena" <sip:Alena@192.168.56.1:5060>;tag=1941871f580bbc54ca3a
Content-Type: text/plain
To: "Bob" <sip:Bob@192.168.56.1:5064>;tag=4b315e800d576ffc2a79
Content-Length: 50

Ja jsem  SIP_Client(Bob) a odpovidam Ti na zpravu!
```

5.5.5 Sedmý scénář

V předcházejících scénářích jsme se dívali pouze na výpis testovací aplikace. Nyní se podíváme také na výpis brány. Sedmý scénář se z příkazové řádky OS Windows volá stejným způsobem, jako šestý scénář: java-jar MultipleOfScenario.jar 7 1 1 5061 Adam Bob. Proces posílání zpráv je náhodný, takže může nastat podobně jako v předcházejících scénářích situace, že daný WS klient ještě nebude pro bránu existovat a brána bude nucena posílat SIP klientovi odpověď typu 404 Not Found. V našem případě tato situace nastala, jak označuje výpis přijatý SIP klientem „Bobem“.

```
DOSTAL JSEM ODPOVED SIP Klient( Bob )
Stav klientske transakce(GeneratorSIP): Completed Transaction
ODPOVED od brany:
```

```
SIP/2.0 404 Not found
To: "Adam" <sip:Adam@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP
Bob.office.quick.cz:5061;branch=z9hG4bK331fa455feffb08de160;received=192.168.56.1
CSeq: 1000 MESSAGE
Call-ID: 8c54a4e8c6095185bc17@192.168.56.1
From: "Bob" <sip:Bob@192.168.56.1:5061>;tag=b844ada970c04fa9f1be
Content-Length: 0
```

V bráně vypadá situace zpracování této vzniklé ST následovně :

```
17398 [EventScannerThread] DEBUG cz.xseife00.projekty.aplikace.brana.SipLayer  -
INFO o ST:
17398 [EventScannerThread] DEBUG cz.xseife00.projekty.aplikace.brana.SipLayer  -
BranchId: z9hG4bK331fa455feffb08de160
17400 [EventScannerThread] DEBUG cz.xseife00.projekty.aplikace.brana.SipLayer  -
Stav Serverovske transakce: Completed Transaction
17400 [EventScannerThread] DEBUG cz.xseife00.projekty.aplikace.brana.SipLayer  -
Zpracovali jsme Response od WS_klienta
17400 [EventScannerThread] INFO  cz.xseife00.projekty.aplikace.brana.SipLayer  -
Stav pred standardnim vymazem
:{sip:Bob@192.168.56.1:5061={z9hG4bK331fa455feffb08de160=gov.nist.javax.sip.stack.
SIPServerTransaction@b24b52a6}}
17400 [EventScannerThread] INFO  cz.xseife00.projekty.aplikace.brana.SipLayer  -
MAZEME(standardne) ST.
17401 [EventScannerThread] INFO  cz.xseife00.projekty.aplikace.brana.SipLayer  -
Stav po standardnim vymazu:{}
```

Z posledních dvou výpisů je důležitý parametr branch ST. Parametr branch ST přidělil SIP klient. V našem případě se jednalo o `branch=z9hG4bK331fa455feffb08de160.` Ve výpisu brány je informace o vzniklé ST. Ta má své identifikační číslo (branchId) právě `z9hG4bK331fa455feffb08de160.` Posláním odpovědi 404 Not Found od brány přechází ST do stravu kompletní („Completed“). To je stav, který bráně dovoluje informace o ST vymazat. Záznam o ST před prováděním údržby (vymazání úspěšných transakcí) vypadá následovně: `{sip:Bob@192.168.56.1:5061={z9hG4bK331fa455feffb08de160=gov.nist.javax.sip.stack.SIPServerTransaction@b24b52a6}}`. Právě protože se jedná o úspěšnou transakci probíhá tzv. standardní mazání, ve výpisu `MAZEME(standardne) ST.` Kdyby nepřišla v určitém čase odpověď, nebo kdyby brána tuto konečnou odpověď nevytvořila. Provedlo by se vymazání záznamu o ST tzv. nestandardním způsobem (tím je myšleno vymazání záznamu o neúspěšné transakci).

5.5.6 Osmý scénář

Osmý scénář má zakázanou konečnou odpověď. Pokud se adresovaný WS klient zaregistruje u brány dříve, než jej začnou kontaktovat SIP klienti, bude se opakovat posílání prozatímní odpovědi od brány do té doby, než vyprší Timeout (zabudovaný časovač) dané transakce. Protože brána bude čekat na konečnou odpověď marně, dojde k vymazání záznamu. Příkaz pro spuštění z příkazové řádky OS Windows je následující: java –jar MultipleOfScenario.jar 8 1 3 5061 5062 5063 Adam Bob Bort Black. V našem případě se stihl WS klient zavčas zaregistrovat (dříve než došla žádost od nějakého ze 3 SIP klientů). Brána neměla ze začátku důvod něco nestandardně mazat:

```
22966 [AWT-EventQueue-0] WARN  cz.xseife00.projekty.aplikace.brana.SipLayer  -
ULOZENO K NESTANDARDNIMU VYMAZANI:{}
```

Následovalo spuštění scénáře 8. Žádosti od SIP klientů posílané bráně jsou následující:

```
SIP klient (Bob) posila zpravu: MESSAGE sip:Adam@192.168.56.1:5060 SIP/2.0
Call-ID: 352d36a970e7996d1594@192.168.56.1
CSeq: 1000 MESSAGE
From: "Bob" <sip:Bob@192.168.56.1:5061>;tag=faa6e5df41c763b7aaee
To: "Adam" <sip:Adam@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP Bob.office.quick.cz:5061;branch=z9hG4bKa98288a096fa041e14e0
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 26

ZDE se hlasi Bob. ZPRAVA_2


SIP klient (Bort) posila zpravu: MESSAGE sip:Adam@192.168.56.1:5060 SIP/2.0
Call-ID: 52ac79d86c80e6f018cb@192.168.56.1
CSeq: 1000 MESSAGE
From: "Bort" <sip:Bort@192.168.56.1:5062>;tag=6f460a78e8395e904090
To: "Adam" <sip:Adam@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP Bort.office.quick.cz:5062;branch=z9hG4bK1da4ec3b287bf6a13165
Max-Forwards: 4
Content-Type: text/plain
Content-Length: 27

ZDE se hlasi Bort. ZPRAVA_3


SIP klient (Black) posila zpravu: MESSAGE sip:Adam@192.168.56.1:5060 SIP/2.0
Call-ID: adf321eb9f035166d7bf@192.168.56.1
CSeq: 1000 MESSAGE
From: "Black" <sip:Black@192.168.56.1:5063>;tag=6cb79bcdc631094286fb
To: "Adam" <sip:Adam@192.168.56.1:5060>;tag=chyba
Via: SIP/2.0/UDP Black.office.quick.cz:5063;branch=z9hG4bK91d88b3ae97ae709f4a5
Max-Forwards: 4
```

```
Content-Type: text/plain
Content-Length: 28

ZDE se hlasi Black. ZPRAVA_4
```

WS klient zareaguje posláním všem 3 SIP klientům prozatímní odpovědi. Jedna z nich je poslána klientu „Bob“:

```
WS KLIENT(Adam): Vytvoril jsem SIP Response:
SIP/2.0 180 Ringing
Via: SIP/2.0/WSS
192.168.56.1:5060;branch=z9hG4bK6aee077d1e2c400f99464af4d89caa29,SIP/2.0/UDP
Bob.office.quick.cz:5061;branch=z9hG4bKa98288a096fa041e14e0;received=192.168.56.1
CSeq: 1000 MESSAGE
Call-ID: 352d36a970e7996d1594@192.168.56.1
From: "Bob" <sip:Bob@192.168.56.1:5061>;tag=faa6e5df41c763b7aaee
Content-Type: text/plain
To: "Adam" <sip:Adam@192.168.56.1:5060>;tag=4fb0e0a22e8be0b3c3dd
Content-Length: 23

Ringing WS_klient(Adam)
```

Konkrétní výpis přijetí prozatímní odpovědi od Adama „Bobem“:

```
DOSTAL JSEM ODPOVED SIP Klient( Bob )
Stav klientske transakce(GeneratorSIP): Proceeding Transaction
ODPOVED od brany:
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP
Bob.office.quick.cz:5061;branch=z9hG4bKa98288a096fa041e14e0;received=192.168.56.1
CSeq: 1000 MESSAGE
Call-ID: 352d36a970e7996d1594@192.168.56.1
From: "Bob" <sip:Bob@192.168.56.1:5061>;tag=faa6e5df41c763b7aaee
Content-Type: text/plain
To: "Adam" <sip:Adam@192.168.56.1:5060>;tag=4fb0e0a22e8be0b3c3dd
Content-Length: 23

Ringing WS_klient(Adam)
```

Brána ze začátku nepotřebovala nic nestandardně promazávat proto výpis po uplynutí časovače byl následující:

```
22966 [AWT-EventQueue-0] WARN  cz.xseife00.projekty.aplikace.brana.SipLayer  -
ULOZENO K NESTANDARDNIMU VYMAZANI:{}
```

Po uplynutí doby byl výpis brány následující (zde krácený) :

```
41128 … INFO o ST:
41128 … BranchId: z9hG4bK1da4ec3b287bf6a13165
41128 … Stav Serverovske transakce: Proceeding Transaction
41128 … Zpracovali jsme Response od WS_klienta
```

```
41129 … Stav pred standardnim vymazem
:{sip:Bort@192.168.56.1:5062={z9hG4bK1da4ec3b287bf6a13165=gov.nist.javax.sip.stack
.SIPServerTransaction@c2d2cba0},
sip:Bob@192.168.56.1:5061={z9hG4bKa98288a096fa041e14e0=gov.nist.javax.sip.stack.SI
PServerTransaction@7990941e},
sip:Black@192.168.56.1:5063={z9hG4bK91d88b3ae97ae709f4a5=gov.nist.javax.sip.stack.
SIPServerTransaction@ab3e7b9e}}
41129 … VYTVARIME NOVEHO KLIENTA PRO NEKOMPLETNI ST.
41129 … Stav po standardnim
vymazu:{sip:Bort@192.168.56.1:5062={z9hG4bK1da4ec3b287bf6a13165=gov.nist.javax.sip
.stack.SIPServerTransaction@c2d2cba0},
sip:Bob@192.168.56.1:5061={z9hG4bKa98288a096fa041e14e0=gov.nist.javax.sip.stack.SI
PServerTransaction@7990941e},
sip:Black@192.168.56.1:5063={z9hG4bK91d88b3ae97ae709f4a5=gov.nist.javax.sip.stack.
SIPServerTransaction@ab3e7b9e}}
```

Z přecházejícího výpisu je vidět, že stav serverovské transakce byl „v jednání“ (Proceeding Transaction). Proto brána nemohla nic standardně vymazat. Bylo zapotřebí počkat znovu na uplynutí časovače. Z následujícího výpisu je vidět, že dané 3 ST jsou uloženy k nestandardnímu vymazání (čas 42974ms po spuštění aplikace). Po vypršení časovače (v čase 62966) bylo zahájeno nestandardní odstranění záznamů. V časech 62697 a 62968 je vidět výsledek nestandardního mazání. Poté již brána nemá nic k nestandardnímu vymazání.

```
42974 …ULOZENO K NESTANDARDNIMU
VYMAZANI:{sip:Bort@192.168.56.1:5062=[Ljava.lang.String;@aea5ca,
sip:Bob@192.168.56.1:5061=[Ljava.lang.String;@67226,
sip:Black@192.168.56.1:5063=[Ljava.lang.String;@130291c}
62966 … Nestandardni vymaz.
62967 … Stav pred nestandardnim mazanim:
{sip:Bort@192.168.56.1:5062={z9hG4bK1da4ec3b287bf6a13165=gov.nist.javax.sip.stack.
SIPServerTransaction@c2d2cba0},
sip:Bob@192.168.56.1:5061={z9hG4bKa98288a096fa041e14e0=gov.nist.javax.sip.stack.SI
PServerTransaction@7990941e},
sip:Black@192.168.56.1:5063={z9hG4bK91d88b3ae97ae709f4a5=gov.nist.javax.sip.stack.
SIPServerTransaction@ab3e7b9e}}
62967 …co chceme nestandardne vymazat:
{sip:Bort@192.168.56.1:5062=[Ljava.lang.String;@aea5ca,
sip:Bob@192.168.56.1:5061=[Ljava.lang.String;@67226,
sip:Black@192.168.56.1:5063=[Ljava.lang.String;@130291c}
62967 … po nestandardnim mazani-identifOfSIPC:{}
62968 … po nestandardnim mazani- unCompleteClient:{}
62968 … ULOZENO K NESTANDARDNIMU VYMAZANI:{}
```

6 Závěr

Tato diplomová práce se zabývá překladem zpráv protokolu SIP na zprávy protoku WebSocket a naopak. Byly popsány základy jednotlivých protokolů, k čemu slouží a jejich principy. Byl představen a podrobně popsán návrh řešení tohoto překladu. Návrh byl vytvořen s využitím diagramů modelovacího jazyka UML. Tento návrh byl následně implementován v jazyce Java pomocí dvou dostupných knihoven. K vytvořené aplikaci byla implementována testovací aplikace.

Výstupem této práce jsou tedy dvě aplikace. Aplikace brána, provádějící překlad mezi SIP protokolem a WebSocket protokolem a testovací aplikace, která slouží k otestování brány. S využitím testovací aplikace bylo provedeno několik testovacích scénářů, které vyvinutou aplikaci testují v různých podmínkách. Popis a výsledky scénářů lze nalézt v kapitole 5. Výsledky testů neodhalily žádnou chybu a potvrzují tak správnost návrhu i implementace. Knihovna jWebSocket se ukázala jako velmi dobře funkční implementace protokolu WebSocket. Knihovna ovšem umožňuje pro přenos zpráv volit jen mezi pevně danými podprotokoly. Tento fakt znemožnil posílání SIP zpráv pomocí podprotokolu SIP, jak to vyžaduje standard. Místo toho jsou zprávy posílány pomocí podprotokolu org.jwebsocket.json, což však na správné funkci aplikace nic nemění.

Výhodou realizované aplikace je její jednoduchá přenositelnost mezi různými platformami. Toho je dosaženo díky její implementaci v jazyku Java.

POUŽITÁ LITERATURA:

[1] ROSENBERG, J. et. al. : SIP: Session Initiation Protocol [online]. 2002, [cit. 1.11.2012] Dostupné z: http://tools.ietf.org/html/rfc3261

[2] SIP. Cesnet [online]. 11.3.2012, [cit. 26.11.2012]. Dostupné z: https://sip.cesnet.cz/cs/protokoly/sip

[3] BANERJEE, K. : SIP (Session Initiation Protocol). Introduction to SIP a Beginners' Tutorial as part of Internet Multimedia [online]. 2009, [cit. 22.11.2012] Dostupné z: http://www.siptutorial.net/SIP/

[4] About HTML5 WebSockets, WebSocket.org [online]. 2012, [cit. 10.11.2012]. Dostupné z: http://www.websocket.org/aboutwebsocket.html

[5] MELNIKOV, A. , FETTE, I. : The WebSocket Protocol [online]. [2011], [cit. 5.11.2012]. Dostupné z: http://tools.ietf.org/html/rfc6455

[6] BAZ CASTILLO, I. et. al. : The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP) [online]. [2012], [cit. 20.11.2012] Dostupné z: http://tools.ietf.org/html/draft-ibc-sipcore-sip-websocket-02

[7] SOUMAR, M. : Signalizační protokol pro přenos hlasu přes datové sítě – SIP, Elektrorevue [online]. [2003], [cit. 15.11.2012]. Dostupné z: http://www.elektrorevue.cz/clanky/03003/index.html

[8] KUČERA, F. : Co je to Copyleft, Svobodný software [online]. [2010], [cit. 1.12.2012]. Dostupné z: https://svobodnysoftware.frantovo.cz/drupal/node/7

[9] SCHULZRINNE, H. et. al. : RTP: A Transport Protocol for Real-Time Applications [online]. [1996], [1.11.2012]. Dostupné z: http://www.ietf.org/rfc/rfc1889.txt

[10] SCHULZRINNE, H. et. al. : Real Time Streaming Protocol (RTSP) [online]. [1998], [1.11.2012].
Dostupné z: http://www.ietf.org/rfc/rfc2326.txt

[11] HANDLEY, M. , JACOBSON, V. : SDP: Session Description Protocol [online]. [1998], [1.11.2012].
Dostupné z: http://www.ietf.org/rfc/rfc2327.txt

[12] BAZ, C. et al. : The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP) – draft –08 [online]. [2013], [24.5.2013].
Dostupné z: http://tools.ietf.org/html/draft-ietf-sipcore-sip-websocket-08

[13] Projekt jWebSocket : Tokens [online]. [24.5.2013].
Dostupné z: http://jwebsocket.org/documentation/User-Guide/tokens

[14] Projekt jWebSocket: jWebSocket the open source solution for realtime web developers [online]. [24.5.2013].
Dostupné z: http://jwebsocket.org/

[15] Projekt JAIN-SIP: Java.net – The Source for Java Technology Collaboration (NIST- National Institute of Standards and Technology) [online]. [24.5.2013]
Dostupné z: https://jsip.java.net/

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BSD	Berkeley Software Distribution
CRLF	Carriage Return Line Feed
EPL	Eclipse Public License
FQDN	Fully Qualified Domain Name
GPL GNU	General Public License
GUID	Globally unique identifier
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IP	Internet Protocol
JAIN	Java APIs for Integrated Networks
LGPL GNU	Lesser General Public License
MIT	Massachusetts Institute of Technology
QoS	Quality of Service
RFC	Request For Comments
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SDP	Session Description Protocol
SHA	Secure Hash Algorithm
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform resource identifier
UTF-8	UCS Transformation Format – 8-bit
W3C	World Wide Web Consortium
WS	WebSocket
NSFWSK	NecessarySipForWSKlient

PŘÍLOHA

Na přiloženém CD jsou, mimo samotnou bránu a testovací aplikaci, všechny potřebné soubory pro spuštění brány i testovací aplikace. Aby byla testovací aplikace funkční, je potřeba ji spouštět na stejném počítači, na kterém je pro test spuštěná brána. Popis spuštění je popsán v souboru ReadMe.