VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO TESTOVÁNÍ ODOLNOSTI WEBOVÝCH SLUŽEB

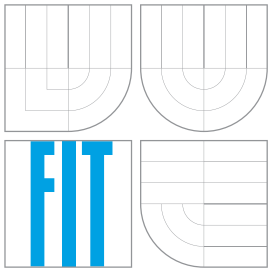
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ ZELINKA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO TESTOVÁNÍ ODOLNOSTI WEBOVÝCH SLUŽEB

A TOOL FOR ROBUSTNESS TESTING OF WEB-SERVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ ZELINKA

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. MAREK RYCHLÝ, Ph.D.

BRNO 2013

Abstrakt

Abstract

Klíčová slova

Keywords

Citace

Tomáš Zelinka: Nástroj pro testování odolnosti webových služeb, diplomová práce, Brno, FIT VUT v Brně, 2013

Nástroj pro testování odolnosti webových služeb

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana RNDr. Marka Rychlého Ph.D. Uvedl jsem všechny literární prameny, ze kterých jsem čerpal.

.....
Tomáš Zelinka
22.května

Poděkování

Tímto bych velice rád poděkoval svému odbornému vedoucímu této práce RNDr. Markovi Rychlému Ph.D. za technickou podporu při vzniku celého projektu a technické zprávy.

Obsah

Kapitola 1

Úvod

Kapitola 2

Základní pojmy

V této kapitole je popsána většina důležitých pojmů a zkratek, které se vyskytují dále v práci nebo jsou důležité pro objasnění některého z použitých principů. Dále jsou vysvětleny principy webových služeb a použitelné přístupy. V další části pak jsou popsány principy testování softwaru. V poslední části základních pojmů jsou popsány součásti platformy programovacího jazyka Java.

Hypermédia

Pojem hypermédia [1] poprvé použil Ted Nelson jako zobecnění pojmu hypertext. Hypertext jsou textové dokumenty propojené pomocí odkazů, zatímco hypermédia rozšiřuje pojem hypertext na jakoukoliv formu média. Tedy jsou to média propojené pomocí odkazů na jiná média.

Serializace a Deserializace

Postup, při kterém je objekt nebo datová struktura převeden do určitého formátu [3]. Tento formát musí umožňovat uložení datové struktury např. na pevný disk nebo jej odeslat pomocí sítě na vzdálený stroj. Deserializace je inverzní postup, při kterém je datová struktura převedena z serializované podoby zpět do své původní.

HTML

HTML je zkratka pro Hypertext Markup Language [19], což je značkovací jazyk pro vytváření strukturovaných dokumentů na webu. HTML umožňuje vkládat sémantické značky, které jsou zpravidla párové. Mezi páry značek tzv. tagů je umístěn text, který je na základě značek interpretován webovým prohlížečem

HTTP

HTTP je zkratka pro Hypertext Transfer Protocol [8]. Je to protokol pro výměnu hypertextových dokumentů ve formátu HTML. Výměna dokumentů probíhá formou dotaz-odpověď.

URI

Universal Resource Identifier zkratka pro jednotný identifikátor zdroje [7]. Je textový řetězec, který má definovanou strukturu. Slouží k identifikaci zdroje v počítačových sítích, zejména internetu. Pod pojmem zdroj je v tomto kontextu myšlen dokument nebo služba.

URI je složeno z řetězce umístění zdroje (URL, Universal Resource Locator) a řetězce názvu zdroje (URN, Univesal Resource Name).

XML

XML je zkratka pro Extensible Markup Language [9]. Je to obecný značkovací jazyk, kterým je možno vytvářet konkrétní značkovací jazyky. Používá se pro přenášení dat mezi webovými aplikacemi a vytváření dokumentů. Při vytváření dokumentů se pomocí značek označuje význam textu, což hraje velkou roli např. při vyhledávání informací v dokumentu.

SOAP

SOAP je původně zkratka pro Simple Object Acsess Protocol, ale od verze 1.2 se používá pouze SOAP jako samostatný pojem [17]. Specifikuje protokol pro výměnu strukturovaných a typovaných informací v decentralizovaných a distribuovaných sítích. Používá XML pro definování sady funkcí pro vytváření a výměnu zpráv nad různými podpůrnými protokoly. SOAP byl vyvinut tak, aby byl nezávislý na transportních protokolech.

REST

Zkratka REST znamená REpresentation State Transfer. Označuje architektonický styl pro hypermediální distribuované systémy. V těchto systémech jsou uloženy dokumenty nebo služby, které jsou identifikovány pomocí URI. REST není přímo svázán s konkrétním komunikačním protokolem. Jelikož HTTP protokol nejlépe vyhovuje principům REST, je s tímto stylem spojován ve většině případů .

Volně svázaný systém

Volně svázané (loosely coupled)[11] systémy obsahují na sobě nezávislé komponenty, které spolu komunikují, a tak vytvářejí požadovaný systém. Nezávislost těchto komponent spočívá v možnosti jejich obměny. Ve volně svázaném systému se mohou jednotlivé komponenty uvnitř měnit (verze, implementace, atd.) aniž by to negativně zasáhlo daný systém.

Těsně svázaný systém

Těsně svázané (tightly coupled)[11] systémy obsahují na sobě závislé komponenty, jejichž implementace je spojuje dohromady. V těsně vázaných systémech si nemůžeme dovolit měnit (implementace, verze, atd.) pouze jednu komponentu, aniž by to mělo negativní následky na chod ostatních komponent nebo systému jako celku.

Otevřený software

Otevřený software (open-source software)[5] je software s otevřeným zdrojovým kódem. Otevřenost znamená možnost zdrojový kód využívat při splnění určitých podmínek.

2.1 Architektura orientovaná na služby

Architektura orientovaná na služby (Services oriented architecture, SOA) [11] je sada principů a postupů pro návrh a vývoj softwaru. SOA se zaměřuje na přesně definované softwarové komponenty (části počítačových systému, datové struktury, apod.), které mohou být použity vícekrát, pro více účelů a v kontextu SOA sem jim říká služby. Nyní si si ukážeme hlavní principy a postupy, které se používají při vytváření služeb v rámci SOA.

Zapouzdření operace

Při vytváření systému obsahujících služby může být rozsah zapouzdření menší i větší. Služba může zapouzdřit jednu operaci v rámci systému nebo operaci, která reprezentuje celý systém.

Vztah mezi službami

Jednotlivé služby mohou být využívány jinými službami nebo programy. Aby mohly služby a programy mezi sebou komunikovat, musí o sobě vědět. Toto povědomí je realizováno pomocí popisu služby. V popisu služby je uveden název služby a druh dat, které služba očekává a vrací.

Komunikace mezi službami

Služby komunikují pomocí zpráv. Jakmile je zpráva odeslána, služba nad ní ztrácí kontrolu. Podle SOA by měla být zpráva nezávislou komunikační jednotkou, a tak zastávat stejnou pozici při komunikaci jako služba. Jinými slovy to znamená, že zpráva si sama určí službu, ke které bude doručena. Abychom toho dosáhli, je nutné vybavit zprávu patřičnou inteligencí. Jedním z možných přístupů SOA, kterým lze SOA implementovat, jsou webové služby. Nutno poznamenat, že ne všechny systémy implementované pomocí webových služeb splňují koncept SOA a zároveň v systému vytvořeném pomocí SOA konceptu nemusí být obsaženy webové služby. Tedy SOA a webové služby nejsou synonyma, ale velmi často spojované pojmy.

SOAP webové služby

Základem pro SOAP webové služeby je SOAP protokol. SOAP je navržen jako bezstavový protokol. Používá jednocestný způsob odesílání zpráv (one-way MEP, Message Exchange Pattern), ale je možno jej přizpůsobit k dalším způsobům odesílání zpráv jako např. žádost/odpověď, žádost/více odpovědí. SOAP protokol je možno používat na libovolné platformě s libovolným programovacím jazykem.

Hlavními specifikacemi pro SOAP webové služby je soubor specifikací s prefixem WS-. Tento soubor specifikací zahrnuje další funkcionality, které byly vyvinuty nad SOAP protokolem. Patří mezi ně např. transakční zpracování, bezpečnost, spolehlivost atd. Dalšími důležitými technologiemi jsou jazyk pro popis definice webové služby (WSDL, Web Service Definition Language) a registr pro popis a lokalizaci webových služeb (UDDI, Universal Description and Discovery Integration). Nejdříve si popíšeme specifikaci SOAP a po té budou představeny technologie WSDL a UDDI.

Specifikace SOAP protokolu je rozdělena na 4 části:

- **Model zpracování**
  Model zpracování představuje jednotlivé části distribuovaného modelu, který je v SOAP použit. Jsou zde popsány druhy uzlů, které využívá distribuovaný model, použité způsoby výměny zpráv mezi uzly a další nezbytnosti, které jsou potřeba pro zpracování SOAP zpráv.

- **Model rozšíření**
  SOAP protokol byl vytvořen jako velmi dobře rozšiřitelný. Tento model popisuje způsob jak mají být zpracovány nové funkcionality a rozšíření SOAP protokolu.

- **Spojení s protokolem pro přenos zpráv**
  SOAP protokol není závislý na přenosovém protokolu a může tak být spjat s různými protokoly. Jako nejvhodnější kandidáti byly vybrány protokoly HTTP a protokol pro elektronickou poštu (SMTP, Simple Mail Transport Protocol). Pro mnohem rozšířenější použití se dnes hlavně používá protokol HTTP. Používá se také pro jeho snadný průchod přes síťová zabezpečení (firewall apod.).

- **Vytváření zpráv**
  Zprávy jsou v SOAP protokolu definovány jako XML dokument. Tento formát má své výhody i nevýhody. XML je výhodné, protože odpadají problémy s přenositelnosti mezi systémy, protože se jedná o textový dokument. Právě zpracování XML dokumentu přináší problémy s výkoností, protože zpracování textových dokumentů přináší svá úskalí.

SOAP zpráva

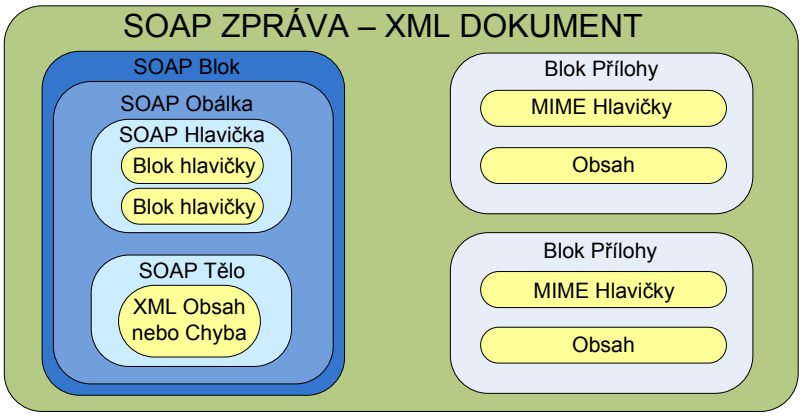
Jelikož služby mezi sebou komunikují pomocí zpráv, je nutné, aby zprávy měly jednotný formát a služby používaly stejný transportní protokol. Formát zprávy musí být flexibilní a rozšiřitelný. Z obrázku 2.1 je patrné, že SOAP zpráva se skládá ze 4 částí:

- **Obálka**
  Jak již název části napovídá, obálka se dá charakterizovat jako kontejner, který volitelně obsahuje hlavičku a povinně musí obsahovat tělo zprávy.

- **Hlavička**
  K dosažení zmíněné inteligence zprávy se využívá obsahu hlavičky SOAP zprávy. Obsah je složen z jednotlivých bloků, které mohou nést různé druhy informace související s doručováním zprávy, zpracování těla zpráv, bezpečností apod. Sémantika jednotlivých bloků není v SOAP specifikována. Každý blok může mít rozdílný význam, a tedy i specifikaci.

- **Tělo**
  Tělo zprávy obsahuje odesílaná data, které jsou ve formátu XML. V případě potřeby může zpráva obsahovat informaci o chybě na straně příjemce. Zpráva s informací o chybě je pak doručena zpět odesilateli.

- **Přílohy**
  SOAP příloha umožňuje posílat zejména data, která nelze popsat XML dokumentem. Obecně se jedná o binární data (multimédia, dokumenty, apod.).

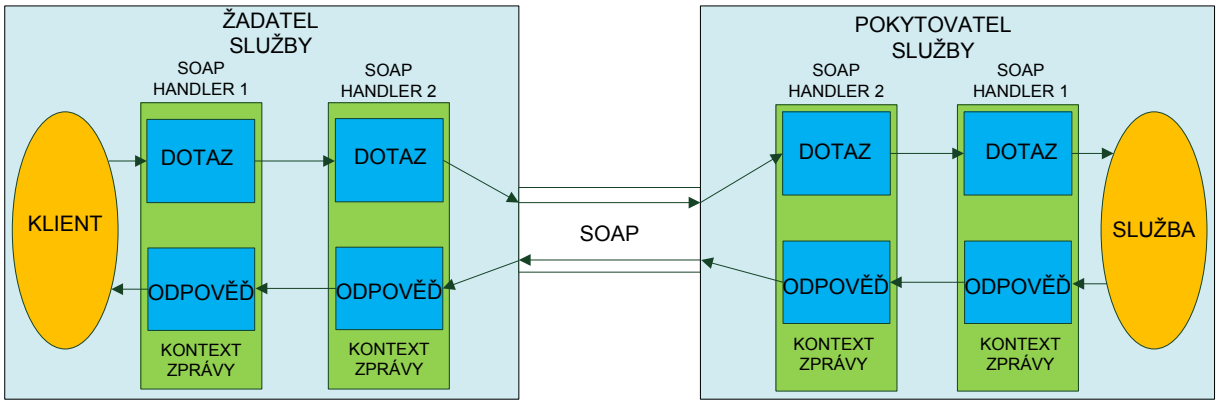


Obrázek 2.1: Obecný formát SOAP zprávy.

SOAP obslužné rutiny

SOAP obslužné rutiny (SOAP handlers)[14] mohou zpracovávat zprávu po příchodu a před odchodem zprávy. Jelikož se v hlavičce zprávy nacházejí různé informace (způsob zpracování těla zprávy, bezpečnostní kontext, transakční kontext, apod.), které je potřeba zpracovat dříve než je služba spuštěna, zpráva je nejdříve zachycena těmito rutinami a ty vykonají příslušné operace před spuštěním služby. SOAP obslužné rutiny se mohou nacházet na klientské straně i na koncové straně služeb.

Na klientské straně zpravidla přidávají informace (bloky hlavičky) do zprávy. Na koncové straně služeb čtou, případně odstraňují informace ze zprávy a vykonávají patřičné operace. Obslužné rutiny nemusejí být přítomny vůbec nebo jich muže být i více najednou.



Obrázek 2.2: Použití SOAP obslužné rutiny.

Jazyk pro popis služeb

Popis služby slouží jako strojově zpracovatelný manuál pro zacházení s webovou službou (WSDL, Web Service Definition Language). WSDL je popisuje webovou službu pomocí XML dokumentu. WSDL patří mezi jazyky definující rozhraní (IDL, Interface Definition Language). IDL jazyky slouží k přesnému popsání rozhraní dané softwarové komponenty tak, aby bylo možné jej strojově zpracovat. Hlavní vlastností IDL jazyků je nezávislost na programovacím jazyku. V praxi to znamená, že pomocí IDL jazyka lze vygenerovat rozhraní pro různé programovací jazyky. V případě WSDL se generují rozhraní pro webové služby. WSDL však neslouží pouze ke generování rozhraní, ale také k vyhledávání a rozpoznání webové služby. V případě vyhledání a rozpoznání je možné WSDL dokument analyzovat a rozhodnout, zda-li daná webová služba odpovídá požadavkům.

WSDL dokument je rozdělen na abstraktní část a konkrétní část. V abstraktní části jsou popsány součásti rozhraní, které nejsou závislé na konkrétních technologiích tzn, datové typy, typy přenášených zpráv. Definice z abstraktní části jsou pak použity pro přesný popis spojení se službou a popis služby samotné.

XML struktura WSDL dokumentu začíná kořenovým elementem *description*, která obsahuje abstraktní součásti *types*, *interfaces*. V konkretní části se pak definice z *types* a *interfaces* použijí pro součásti *binding*a *service*. Toto je popis WSDL podle verze 2.0. Předešlá verze obsahuje odlišnosti v pojmenování a struktuře jednotlivých elementů. WSDL verze 2.0 je také vhodná pro popis REST webových služeb.

Popis jednotlivých součástí WSDL dokumentu:

- **Typy**
  V součásti *types* jsou definovány použité datové typy. Je možné použít různé druhy popisu datových typů, ale je doporučeno a hlavně se používá XML schéma pro definici datových typu (XSD, XML Schema Definition Language). Také je zde popsán formát přenášených zpráv. Je možné tyto definice načíst z externího zdroje pomocí elementu *import*.

- **Rozhraní**
  V součásti *interfaces* jsou popsány operace, které budou použity službou. Zde se jednotlivým operacím přiřadí vstupní a výstupní zprávy, které jsou definovány v části *types*.

- **Spojení**
  Součást *bindings* slouží k definici spojení s danou technologii. Obvykle to jsou SOAP a HTTP protokoly. Dále je pak definováno jaký styl SOAP zprávy má být použit. V tomto případě jsou dvě možnosti, *document* nebo *rpc* styl. Je doporučeno používat *document* styl. Styl *document* nemá strukturální omezení obsahu SOAP zprávy, čili může to být libovolný XML dokument. Naproti tomu *rpc* styl musí odpovídat předepsanému RPC schématu zprávy. Oba styly musí být XML dokumenty, proto mohou být naprosto stejné pouze z výše popsanými omezeními.

- **Služba**
  Součást *service* je použita pro konkrétní definici dané služby pomocí výše zmíněných součástí. Je zde použito definované spojení a adresa, na které je služba dostupná.

WSDL má samozřejmě mnoho dalších možností, ale výše uvedené jsou postačující k vy-

tvoření plnohodnotné služby. Více informací o WSDL je možno získat v příslušných specifikacích.

Registr pro popis a nalezení webových služeb - UDDI

UDDI slouží pro ukládání informací o webových službách. Informace jsou uloženy jako XML dokumenty většinou v SQL databázích. UDDI se hlavně používá pro vyhledávání webových služeb na základě jejich popisu, aby se potom mohly použít. Tento centralizovaný způsob má za cíl zpřehlednit a usnadnit vyhledání a následné použití webové služby.

Svého času se specifikace UDDI těšila velké popularity a byla hojně využívána a rozvíjena. Vývoj specifikace se však zastavil před několika lety na verzi 3.0. V této době velké firmy nabízel veřejné UDDI registry pro zařazení veřejných webových služeb. Avšak z této cesty bylo upuštěno a dnes již veřejných UDDI registrů je velmi málo. UDDI se však stále používá, ale své místo získala tato technologie v privátních sítích a řešeních. Používá se při použití sběrnic služeb (ESB, Enterprise Service Bus) a podobných integračních řešeních, kde se automatizují procesy. Zde právě sběrnice služeb může dynamicky vyhledat v UDDI registrech službu podle požadovaných kritérií, a tak vybrat vhodnou službu k použití.

UDDI registr se skládá ze 3 částí:

- **Bílé stránky**
  V bílých stránkách jsou uloženy informace o poskytovateli služby. Pomocí bílých stránek lze najít službu např. podle jména nebo názvu poskytovatele.
- **Žluté stránky**
  Ve žlutých stránkách jsou uloženy informace o kategorii poskytované služby. Jelikož poskytovatel může poskytovat více kategorií služeb, může se více žlutých stránek vztahovat k jedné bílé.
- **Zelené stránky**
  V zelených stránkách jsou uloženy technické informace o službách. Typické informace v zelených stránkách jsou spojení s webovou službou, adresa služby nebo předávané parametry. Jelikož služba může definovat více druhů spojení, mohou zelené stránky obsahovat více záznamů k jedné službě.

Jednou s velmi populárních implementací UDDI specifikace je jUDDI, která je pod správou Apache Foundation.

SOA pomocí webových služeb

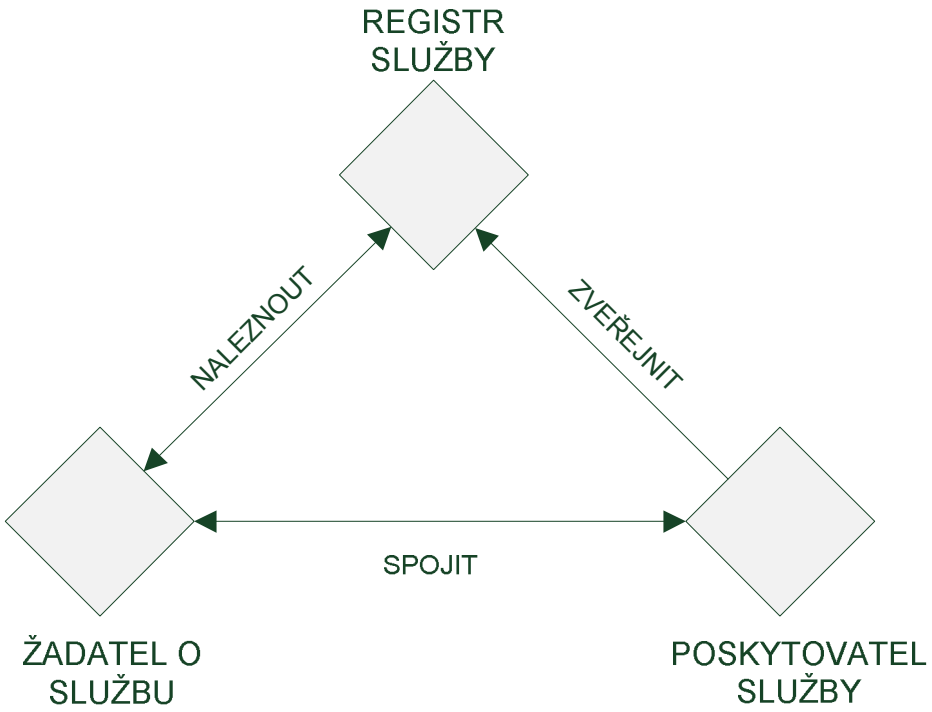
Co jsou to webové služby jsme si popsali již dříve. Nyní se podíváme na jejich strukturu a jak pomocí nich lze implementovat principy SOA.

Zapouzdření operace do služby je zde realizováno právě pomocí webové služby. Popis služby lze realizovat pomocí WSDL. Komunikace je realizována pomocí SOAP protokolu. SOA pomocí webových služeb obsahuje 3 role :

- **Poskytovatel služby - Service provider**
  Je zodpovědný za popis služby a její spuštění na serveru. Pak je služba dostupná ze sítě a je možno zaregistrovat její popis v registrech, kde bude k nalezení pro žadatele o službu. Tuto roli lze připodobnit k serveru v architektuře klient-server.

- **Žadatel o službu - Service requester**
  Vyhledá požadovanou službu v jednom nebo více registrech. Po nalezení popisu dané služby se žadatel spojí se službou u poskytovatele a může ji používat. Tuto roli lze připodobnit ke klientovi v architektuře klient-server.

- **Registr služby - Service registry**
  Registr vystavuje popisy služeb dostupných od poskytovatelů pro žadatele, kteří v těchto registrech vyhledávají požadované služby. Jakmile je jednou služba v registru nalezena, není již tato role potřebná.



Obrázek 2.3: Model SOA pomocí webových služeb.

Tyto role mohou hrát jakékoliv programy nebo uzly v síti. Webové služby také obsahuje tři operace, které definují vztahy mezi jednotlivými rolemi:

- **Zveřejnění**
  Je vztah operace mezi registrem služeb a poskytovatelem, kdy poskytovatel zveřejní popis služby v registru, a tak ho zpřístupní žadatelům. Konkretní mechanizmus zveřejnění závisí na implementaci registru. V určitých případech roli registru přejímá sama síť, kdy je popis služby zveřejněn přímo v přístupné části souborového systému na serveru.

- **Nalezení**
  Žadatel definuje vlastnosti transakce a podle nich je vyhledána služba v registrech.

Výsledkem hledání je seznam služeb, které odpovídají kriteriím. Nejjednodušší vyhledávací dotaz je HTTP GET[1] bez parametrů, který vždy vrátí všechny služby, které jsou k dispozici. Pak je na žadateli, aby vybral správnou službu. Pro komplexní registraci a vyhledání služeb se používá standardu UDDI .

- **Spojení**
  Ustanovuje vztah klient-server mezi žadatelem o službu a poskytovatelem. Tento vztah může být dynamický, kdy mohou být za běhu vybírány jiné implementace dané služby daných kriterii. Také může být statický, kdy programátor přímo implementuje způsob volání dané služby.

2.2 Principy REST

Architektonický styl REST byl navržen Roy Fieldingem v rámci jeho disertační práce v roce 2000 [12]. REST není architekturou ani technologii. REST je sada omezení, které musí dané řešení obsahovat. REST byl vytvořen pro prostředí internetu a webu. REST je orientován na zdroje, kde zdroj představují požadováná data nebo stav daného vzdáleného systému. Tento styl získávání dat je velmi podobný SQL dotazům, kde také požadujeme určitá data (deklarativní programování). V kontrastu s tímto je SOAP protokol, pomocí kterého jsou na vzdálené straně vyvolávány procedury (procedurální programování).

Ačkoliv je možné se dočíst o soupeření a porovnávání mezi SOA a REST, tak hlavní smysl REST je být alternativou k procedurálnímu charakteru SOA architektury.

Nyní se podíváme na základní sadu charakteristik REST:

- **Architektura klient/server**
  Zde je klientem uživatelské rozhraní a server je úložiště dat. Díky oddělení uživatelského rozhraní a serveru má být dosaženo lepší přenositelnosti uživatelského rozhraní a lepší škálovatelnosti serverové strany.

- **Bezstavovost**
  Kontext nesmí být uložen na straně serveru a odeslaná odpověď musí být sebepopisující. Veškeré informace o aktuálním sezení jsou uloženy na straně klienta. Toto omezení má umožňovat lepší viditelnost, spolehlivost a škálovatelnost. Lepší viditelnosti je dosaženo, protože monitorovací systémy mohou monitorovat pouze odesílané požadavky a přijímané odpovědi, ve kterých jsou uvedeny všechny potřebné informace. Lepší spolehlivosti je dosaženo díky snadnější obnově po chybách, protože na serveru nejsou uloženy kontextové informace. Díky neukládání kontextových informací na serveru je dosaženo i lepší škálovatelnosti.

- **Mezi-paměť**
  Pro lepší výkonnost dotazů by měla být použita na straně klienta mezi-paměť (cache), která ukládá výsledky již provedených dotazů. Toto zlepšení výkonnosti je vykoupeno menší spolehlivostí dat, které nemusí být v danou chvíli aktuální.

- **Vrstvy**
  Systém podle REST by měl být sestaven z více vrstev a každá vrstva komunikuje

[1] Hypertext transfer protokol - protokol pro přenášení hypertextových HTML dokumentů
GET - jedna z metod HTTP pro získání webové stránky klientem ze serveru

pouze se sousedními vrstvami. Například webový prohlížeč si vyžádá data od web serveru, ale již neví o databázi, kterou server využívá k získání požadovaných dat.

- **Jednotné komunikační rozhraní**
  Jednotlivé komponenty systému musí mít jednotné rozhraní. Což může mít za následek menší výkon systému, protože rozhraní nebude vytvořeno podle vlastností aplikace. Pro vytvoření jednotného rozhraní jsou vytvořeny omezení, která musí daný systém splňovat. Mezi ně patří identifikace zdrojů, manipulace se zdrojem skrze jeho reprezentaci, zprávy mezi komponentami systému musejí být sebe popisující, hypermédia jsou prostředkem ke stavu aplikace.

- **Kód na vyžádání**
  Nepovinnou funkcionalitou je tzv. kód na vyžádání. Tento pojem souvisí s rozšířeními, které jsou zasílána klientské aplikaci na vyžádání. Jedná se o dodatečný kód (Java applet, Javascript), který se spouští na straně klienta. Kód na vyžádání je nepovinnou charakteristikou, protože vykonávání takovéhoto kódu může být na klientské straně zakázáno.

Tyto vlastnosti nemusí být na první pohled úplně srozumitelné, proto budou dále popsány na konkrétní technologii v následující kapitole.

REST webové služby

Druhou koncepcí webových služeb [16], se kterou se můžeme setkat, jsou webové služby respektující REST architektonický styl. Protože REST stylu vyhovují vlastnosti HTTP protokolu a HTTP protokol je využíván ke konstrukci webových služeb, byly vlastnosti REST použity pro konstrukci webových služeb. Nyní bude popsáno propojení REST vlastností a omezení, která byla představena dříve v textu, s protokolem HTTP a technologii webových služeb.

HTTP protokol splňuje REST vlastnosti a omezení díky následujícím vlastnostem:

- **Architektura klient/server**
  Samotný protokol HTTP je konstruován stylem dotaz/odpověď klienta a serveru.

- **Bezstavovost**
  Tato vlastnost už musí být implementována specificky na dané architektuře, která je postavena nad HTTP protokolem. Pokud je kontext ukládán na serveru, tak se nejedná o aplikaci založenou striktně na principech REST.

- **Mezi-paměť**
  Mezi-paměť může využívat i HTTP protokol pro ukládání odpovědí na klientské straně.

- **Vrstvy**
  Vícevrstvé rozdělení je využito v rámci HTTP protokolu. Když si webový prohlížeč vyžádá od serveru data, není si vědom např. databáze, se kterou server dále komunikuje.

- **Identifikace zdrojů**
  Zdroje v HTTP mohou být identifikovány pomocí URI.

- **Manipulace se zdroji pomocí jejich reprezentace**
  Data, která jsou odesílána pomocí HTTP mohou být přenášená v jiné reprezentaci než jak jsou uložena. Data je možno přenášet ve standardizovaném formátu (JSON, XML apod.).

- **Sebe popisující zprávy**
  Tuto vlastnost HTTP protokol implicitně nesplňuje, ale umožňuje. Tuto vlastnost je třeba doplnit ve vrstvě, která je implementována nad HTTP protokolem.

- **Hypermédia jako prostředek stavu aplikace**
  Toto omezení se vztahuje k využití HTML dokumentů. Klient komunikuje se serverem pomocí hypermédii, které jsou dynamicky zprostředkovány serverem. Klient nemusí nutně být schopen interpretovat každý druh média, který mu je serverem nabídnut. Správnou interpretaci určitých druhů médii je možno dosáhnout kódem na vyžádání, který dodatečně umožní správnou interpretaci.

Nyní jsme si ukázali, že HTTP protokol je možno použít pro aplikace založených na REST principech.

Základní princip webových služeb je zprostředkování aplikačně nezávislého komunikačního rozhraní. Pro vytvoření jednotného a aplikačně nezávislého komunikačního rozhraní je možné použít HTTP protokol, a zprostředkovat tak aplikace, které jsou vytvořeny různými technologiemi. HTTP disponuje sadou příkazů, které je možno využít právě ke konstrukci webových služeb respektující omezení REST. Základní příkazy, které se nejčastěji používají jsou GET, PUT, POST, DELETE. Tato skupina operací se označuje akronymem CRUD (Create, Read, Update, Delete) a je možné se s ní setkat v deklarativních způsobech programování.

- **GET**
  získání stavu zdroje

- **PUT**
  vytvoření nového zdroje

- **POST**
  upravení zdroje

- **DELETE**
  odstranění zdroje

Pro REST charakter webových služeb jsou tyto operace naprosto dostačující. Z akronymu REST - Representative State Transfer (přenos reprezentativního stavu) - je patrné, že se bude přenášet pouze stav. V případě REST se přenáší stav požadovaného zdroje, přesněji jeho reprezentace. Ačkoliv se to na první pohled může zdát velmi limitující, ve skutečnosti může v pozadí těchto příkazů pracovat velmi komplexní logika.

2.3 Testování software

Testování je součástí životního cyklu software [18] [13]. Testování je obecně neodmyslitelnou součástí jakéhokoliv produktu, nejenom z oblasti IT. Přeci jen, pokud chceme zaručit určité vlastnosti produktu, je třeba tyto vlastnosti vyzkoušet - otestovat. Jelikož software vytvářejí lidé a ti nejsou neomylní, je třeba prověřit možné pochybení při vývoji software.

V ideálním případě bychom při testování vyzkoušeli všechny možné způsoby použití, vstupy, prostředí, abychom důkladně otestovali vytvořený program. V reálném světě tento způsob není možný, protože vyzkoušet všechny možné vstupy i jednoduchého programu je prakticky neřešitelný problém. K tomuto tvrzení stačí poukázat na program, který sečte dvě čísla z množiny reálných čísel. Ne vždy je kompletní testování neřešitelný problém, minimálně je však kompletní testování velmi nepraktické a hlavně neekonomické.

Proto je třeba si uvědomit, že nemůžeme testovat všechny možné vstupy. Ačkoliv nemůžeme zaručit absolutní bezchybovost software, můžeme se pokusit snížit počet chyb v software na minimum. Hlavním cílem testování je tedy odhalit co největší počet chyb. Což je paradox testování, protože úspěšné testování znamená odhalení chyby. Aby bylo testování úspěšné a odhalilo co nejvíce chyb, je nutné k němu přistupovat komplexním způsobem a počínat si obdobně jako při vývoji samotného software. Podívejme se tedy na základní techniky testování software.

Základní techniky testování software

K popsání všech důležitých technik testování software by bylo potřeba celá kniha, zde budou popsány pouze ty základní metody nebo ty metody, které souvisejí s touto prací.

Statické a Dynamické testování

Pokud při testování není třeba spustit samotný program, jedná se o statické testování. To může být například procházení zdrojových kódů nebo dokumentace. Důležité je odhalení chyb již ve specifikaci požadavků, protože vytvoření programu podle chybné specifikace patří mezi ty opravdu nákladné chyby.

Dynamické testování vyžaduje spuštění určité verze programu. Testování probíhá vkládáním množiny vstupů a hodnocení dosažených výstupů. Celá tato práce je zaměřena právě na dynamické testování, kam lze zařadit i níže popsané techniky.

Black box a White box testování

Black box (černá krabička) je technika testování, při které neznáme vnitřní strukturu programu. Black box testování se navrhuje hlavně podle specifikace a je třeba vhodně navrhnout vstupní data tak, abychom dosáhli vyváženosti mezi efektivitou a úplností testování. Používají se další podpůrné techniky jako rozdělení dat podle intervalů, rozdělení dat podle ekvivalenčních tříd apod.

White box (bílá krabička) nebo také transparent box (průhledná krabička) je naproti Black box testování zaměřena na vnitřní strukturu programu. Zaměřuje se hlavně na provedení každého příkazu, který je v programu napsán. V tomto směru je lehce podobné black box testování, protože otestování každého příkazu v těle programu může být prakticky nemožné, minimálně neefektivní. Dalším aspektem testování struktury programu je také skutečnost, že ačkoliv otestujeme každý možný příkaz v programu, přesto v něm může být chyba. Příkladem může být nedodržení specifikace nebo chybná specifikace.

Regresní testování

Regresní testování se obvykle provádí po implementaci nové funkcionality. Má za úkol provést testy starších částí programu, které již byly testovány. Tento druh testování je důležitý, protože zavedením nové funkcionality je možné zanést chyby i do starších částí programu.

Výkonnostní testování

Výkonnostní testování se věnuje výkonu programu. Testovat výkon aplikace lze více způsoby. Program můžeme podrobit zátěžovému testování (stress testing), kdy jsou softwaru ztíženy podmínky pro fungování (odebrání paměti, čas procesoru apod.) nebo je vystaven krátkodobému přetížení množstvím vstupních dat.

Pak je možné testovat dlouhodobou zátěž, kdy je program přetížen množstvím vstupních dat po delší dobu. Ještě je možné opakovat jednu a tu samou operaci neustále do kola, což má za cíl testovat zacházení s pamětí.

Bezpečnostní testování

Důležitou součástí testování je i test bezpečnostních opatření. Při bezpečnostních testech se testuje zajištění základních požadavků na bezpečnost, což jsou *dostupnost*, *důvěryhodnost*, *integrita*. Pro bezpečnou aplikaci je nutné zabezpečit všechny tyto tři vlastnosti současně, jinak nelze považovat aplikaci za bezpečnou.

Protože bez dostupnosti uživatel nebude mít přístup k datům, ačkoliv jsou důvěryhodná a bez poškození. Bez integrity jsou data dostupná a nebyly zobrazeny jiným uživatelům, než pro které byly určeny, ale mohou být poškozená. Nakonec pokud data budou nezměněná i dostupná, mohou být vystavena nepatřičným uživatelům.

Základní úrovně testování software

Výše zmíněné techniky testování se mohou provádět v různých úrovních vývoje softwaru. Podle úrovně vývoje jsou pojmenovány i další techniky testování

- **Testování modulů**
  Také známé jako Unit testing neboli testování jednotek. Zde se testují větší celky kódu programu. Při objektově orientovaném programování se často jedná o testování jednotlivých tříd. Obecně se však testují vhodně oddělené úseky kódu, které jsou testovatelné odděleně. Při testování modulů je často nutné simulovat interakci s ostatními moduly.

- **Integrační testování**
  Při integračním testování se provádějí testy spojení jednotlivých modulů. Testuje se rozhraní a interakce mezi jednotlivými moduly. Obecně je předpokládáno, že čím má integrace větší rozsah, tím je obtížnější odhalit rozhraní, ve kterém vznikla chyba. Z tohoto důvodu vzniklo několik přístupů k integračnímu testování.

  První přístup spočívá v kompletní integraci všech modulů, což je jeden pól extrému. Při tomto přístupu se nejdříve sestaví všechny moduly a po té je zahájeno integrační testování. Na první pohled se může takovýto přístup jevit velmi výhodně, pokud naivně předpokládáme, že se během testování neobjeví žádná chyba. Tento přístup má hlavní nevýhodu právě ve výše zmíněném předpokladu. Pokud je nalezena chyba je obtížné a časově náročně ji odstranit.

  Protipólem je testovat spojení každého modulu zvlášť. Tento přístup je výhodný pro nacházení a odstraňování vzniklých chyb, ale zároveň je velmi časově náročný, tedy neefektivní. Proto se používají techniky, které jsou někde mezi těmito dvěma extrémy.

- **Systémové testování**
  Systémové testování se zaměřuje na program nebo produkt jako celek. Může obsahovat testy založené na specifikaci, testování případu užití programu apod. Systémové testování je poslední úroveň vývojového testování, kdy se ověřuje zda program odpovídá specifikaci. Při systémovém testování by se měly testovat funkční požadavky i ty, které přímo s funkčností nesouvisejí. Mezi nefunkcionální testování patří například výkonnostní testy, testy spolehlivosti, bezpečnostní testy apod.

- **Akceptační testování**
  Akceptační testování je záležitostí uživatele, zákazníka nebo je prováděno prostředí téměř identickém produkčnímu, ne-li přímo v produkčním prostředí[2]. Testuje je se hlavně splnění specifikovaných požadavků na software a jestli dodaný software je schopen pracovat v daném prostředí a je vhodný pro daný účel.

  S akceptačním testováním jsou spojené i pojmy Alfa a Beta testování. Alfa testování se provádí ve vývojovém prostředí programátory a patří do něj výše zmíněné úrovně a techniky. Beta testování se provádí před oficiálním vydání softwaru, kdy si daný software nainstalují uživatelé v reálném prostředí a provádějí testy. Beta testování se provádí v případě, že je software určen pro širokou škálu uživatelů.

2.4 Platforma jazyka Java

Java Enterprise edition

Platforma Java poskytuje více druhů použití jazyka Java. Tato práce se zaměřuje na použití podniková edice Java platformy (Java EE, Java Enterprise edition)[15]. Existuje více částí platformy Java, které jsou určené pro různé použití. Java EE se používá pro vytváření rozsáhlých podnikových aplikací a informačních systémů. Základem Java EE je standardní edice jazyka Java (Java SE, Java Standard Edition), která se postupným vývojem rozšířila a musela být rozdělena na více edici včetně Java EE.

Aplikační logika Java EE je rozdělena do komponent podle funkce. Jednotlivé komponenty jsou umístěny v několika aplikačních vrstvách:

- **Klientská vrstva** - umístěna na straně klienta
  Tato vrstva typicky obsahuje uživatelské rozhraní pro ovládání cele aplikace.

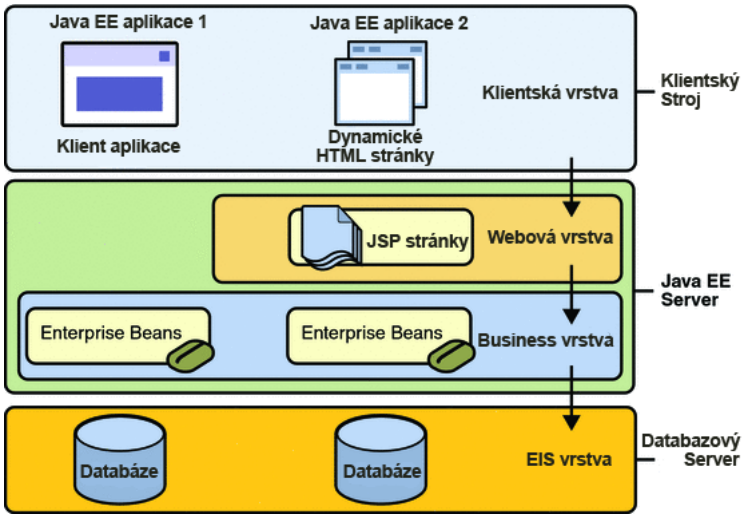
- **Webová vrstva** - umístěna na straně Java EE serveru
  Na této vrstvě jsou umístěny komponenty vytvářející obsah, který je následně odeslán klientovi.

- **Business vrstva** - umístěna na straně Java EE serveru
  Zde je implementována aplikační logika, která přijímá vstupy od klienta, zpracovává je, případně je uloží do databáze.

- **Vrstva Podnikový informační systém (EIS)**
  Komponentami této vrstvy jsou databáze a podobné aplikace pro ukládání dat.

Takto navržené vícevrstvé aplikace je obtížné vytvářet, proto jsou komponenty vytvářeny tak, aby byly znovupoužitelné. Pak je mnohem snadnější vytvářet složité aplikace. Každá

[2]Produkční prostřední nebo produkce označuje prostředí, ve kterém bude program, produkt nasazen a používán



Obrázek 2.4: Jednotlivé vrstvy Java EE architektury. [15]

komponenta musí být umístěna do kontejneru. Kontejner je rozhraní mezi komponentou a funkcemi nižší úrovně, které jsou implementačně závislé. Obecným kontejnerem je aplikační server, který zprostředkovává služby ostatním kontejnerům a komponentám. Na aplikační server je možné nasadit tzv. Enterprise JavaBeans (EJB)[3] a Web kontejner, do kterých lze umisťovat patřičné komponenty. Dále existují klientský a applet kontejner, které jsou vhodné pro klientskou část aplikace.

Pro vytváření vícevrstvých aplikací Java EE obsahuje velké množství technologii a jejich specifikací, které je možno rozdělit do několika kategorii:

- **Webové služby**
  Zahrnuje specifikace technologii pro implementaci SOA pomocí webových služeb. Například obsahuje specifikaci pro tvorbu webových služeb založených na XML (JAX-WS). JAX-WS je možné využít k tvorbě klientských aplikací a webových služeb založených na XML . Mezi klienty a službami lze vytvořit komunikaci založenou na zprávách nebo vzdálené volání procedur (RPC, Remote Procedure Call).

- **Webové aplikace**
  Zde jsou uvedeny technologie pro tvorbu webových aplikací a podporu pro implementaci modelu model-view-controller (MVC). Významnou specifikací z této kategorie je JavaServer Pages (JSP), která se používá k tvorbě dynamických webových stránek. Používá se společně se specifikací Java Servlet. Servlet je komponenta běžící v rámci webového serveru. Přijímá dotazy od klientů a generuje odpovědi ve formě HTML dokumentu. JSP slouží k oddělení uživatelského rozhraní od aplikační logiky. Pomocí JSP se vytváří HTML dokument s vnořenými úseky Java kódu.

- **Podnikové aplikace**
  V této kategorii jsou uvedeny technologie pro implementaci business logiky a různých podpůrných technologii. Pro vytváření podnikových aplikací je vhodná specifikace Enterprise JavaBeans (EJB). Pomocí EJB jsou vytvářeny komponenty, které oddělují

[3] EJB je kontejner, vhodný pro implementaci aplikační logiky

aplikační logiku od prezentační vrstvy. Hlavní vlastnosti EJB komponent by měla být znovupoužitelnost. EJB komponenty je možno využít k zajištění bezpečnosti, transakčního zpracování nebo také k testování. Pro komunikaci mezi jednotlivými komponentami je vhodná specifikace Java Messaging Service (JMS).

- **Správa a zabezpečení**
  Obsahuje technologie pro monitorování a zabezpečení vytvořených aplikací. Pro tvorbu monitorovacích aplikací se používá specifikace Java Management Extensions (JMX). JMX umožňuje vytvářet aplikace s webovým rozhraním, které slouží k monitorování a správě různých zařízení, aplikací nebo sítí se službami. Aplikace v rámci JMX jsou vytvářeny pomocí Managed Bean (MBean) komponent, které zajišťují spojení se sledovaným objektem.

Kapitola 3

Testování webových služeb

Tato část se bude věnovat aspektům testování webových služeb se zaměřením na testování robustnosti a odolnosti [10]. Na testování webových služeb se můžeme dívat jako na testování jakéhokoliv jiného software, avšak musíme vzít v úvahu několik zásadních skutečností. Webové služby, ať už s použitím SOAP protokolu nebo REST principů, jsou vzdálené aplikace, které se mohou skládat z několika vrstev. Dále pak webové služby mohou být kompletně bez grafického uživatelského rozhraní, což znesnadňuje manuální testování, na druhou stranu je zde větší prostor pro automatizaci testování.

3.1 Různé pohledy na webové služby

Při testování webových služeb se můžeme setkat s různými aspekty, které vyplývají z jejich použití. Webové služby jsou velice často implementovány s typem komunikace dotaz/odpověď nebo alespoň odvozeným způsobem. Tedy lze rozdělit účastníky v komunikaci na klienty a server, kdy klient odešle dotaz a server (služba) odešle odpověď. Zde je nutné si uvědomit, že pokud testujeme klient/server aplikace, je nutné otestovat tyto části odděleně. Musíme si být jistí, že server dává správně odpovědi a klient odesílá správně dotazy. Protože pokud jedna část (např. klient) odesílá chybná data, druhá část pak nemusí odesílat data adekvátním způsobem.

Server

Pro webové služby je toto základní pohled, protože služba je umístěna na vzdáleném stroji a plní tak roli serveru v architektuře klient/server. Z tohoto pohledu se testují reakce webové služby na zaslané dotazy a analyzují se přijaté odpovědi. Pro tento typ testování je nutno vytvořit klientskou část aplikace. Klientskou část aplikace pro testování je možno vytvořit automatizovaně. Existují nástroje pro generování klientských rozhraní jak pro SOAP webové služby, tak i pro webové služby založené na REST principech. Příkladem takové technologie je JBoss Wise.

Klient

Pokud využíváme webových služeb většinou se nespokojíme s vygenerovanou klientskou částí aplikace a je třeba ji rozšířit. Proto je třeba klientskou část aplikace také podrobit testování. Nejdříve otestujeme klientskou část s tzv. falešným serverem (stub server). Stub

server má pouze shodné rozhraní se skutečně použitým. Což umožňuje velmi rychle vytvořit testovací server, se kterým je možno vyhodnotit vlastnosti odesílaných dotazů.

3.2 Různé umístění webových služeb

Dále pak je nutné uvažovat umístění webových služeb. Webové služby mohou být dostupné z intranetu, kdy jsou služby umístěny v uvnitř podnikové sítě. Přístup k nim mají pouze zaměstnanci nebo uživatelé s přístupem do dané uzavřené sítě. Pak existují webové služby dostupné z internetu, které jsou veřejně dostupné.

Intranet

Pokud je služba dostupná pouze z vnitřní sítě, je možné přibližně odhadnou kolik uživatelů bude danou webovou službu využívat, protože přístup do vnitřní sítě obvykle má omezený počet uživatelů. Pokud uživatelé přistupují ke službě ve vnitřní síti z vnější sítě, může simulaci tohoto případu užití komplikovat použitá bezpečnostní opatření (firewall, autentizace, VPN apod.).

Internet

U veřejně dostupných webových služeb je obtížnější odhadnout kolik uživatelů bude tuto službu využívat. V tomto případě nemusejí být tak vysoké nároky na zabezpečení jako na webové služby dostupné z intranetu.

3.3 Izolace webových služeb

Při testování webových služeb je v některých případech potřeba cílovou webovou službu izolovat. Izolace webové služby se využívá v případech, kdy je potřeba simulovat propojení s dalšími službami nebo systémy. Tyto služby jsou simulovány tzv. *Mock* službami. *Mock* služba disponuje shodným rozhraním jako její vzorová služba, avšak její funkcionalita není implementována a je omezena na konstantní odpověď. Toto řešení je výhodné v testovacích prostředích z více důvodů. Vzorové služby mohou být placené, jejich implementace nemusí být veřejná, mají delší čas odezvy než Mock služby, Mock služby dovolují testovat jednotlivé celky systému odděleně apod.

Mock služeb se využívá také při zátěžovém testování. Mock služby zatěžují cílový systém minimálně, a tak je možné izolovat cílovou službu a zaměřit se pouze na testování této služby.

3.4 Požadavky na webové služby

Webové služby vyžadují pro správné fungovaní specifické požadavky. Těchto požadavků je několik skupin a obecně se dají shrnout do jednoho pojmu - kvalita služeb (QoS, Quality of Service). Pojem kvalita služeb souvisí s více vrstvami TCP/IP modelu, protože informace proudí skrze internet. V této práci se však hlavně budeme zabývat kvalitou služeb z pohledu aplikační vrstvy[1].

[1] Pojem QoS je také spojován s protokoly nižších vrstev, kde zajišťuje speciální zacházení s určitými typy informací (video, hlas apod.)

Základními pojmy kvality služeb jsou spolehlivost, výkon, robustnost, bezpečnost a ostatní jsou spíše odvozeny od těchto základních požadavků. Pokud je třeba formální a přesné rozlišení jednotlivých pojmů je vytvořena tzv. dohoda o úrovni poskytování služby (SLA, Service Level Agreement).

- **Spolehlivost**
  Spolehlivost u webových služeb souvisí s plněním definovaných požadavků na službu. Pokud služba splňuje definované požadavky jako čas odezvy, dostupnost, schopnost zpracovat určitý počet transakcí za daný časový okamžik apod., je označena za spolehlivou. Spolehlivost tak může být a často také je provázána s ostatními požadavky.

- **Výkon**
  Výkon webové služby vyjadřuje jak rychle dokáže webová služba plnit požadované operace. Výkon lze dále rozdělit na propustnost, doba odezvy a čas provedení. S výkonností souvisí i škálovatelnost, protože webová služba musí být škálovatelná. Webové služby by měly být připraveny na případné rozšíření v případě nedostatečného výkonu.

- **Robustnost**
  Robustnost je schopnost služby fungovat předvídatelným způsobem při nesprávných, chybných, neúplných vstupech.

- **Bezpečnost**
  Jelikož jsou webové služby často exponovány velkému počtu uživatelů, je nutné se zaručit její bezpečnost. Mezi základní bezpečnostní vlastnosti patří dostupnost, integrita a důvěrnost.

3.5 Způsoby testování webových služeb

Jelikož webové služby patří mezi distribuované aplikace, je nutné nejdříve testovat dostupnost dané služby. Pokud služba není z nějakého důvodu dostupná, další druhy testování (rozhraní, zátěžové, bezpečnostní) ztrácejí na významu. Po té co zjistíme, že služba je dostupná, je vhodné ověřit její správné fungování funkcionálními testy. Pokud webová služba vykazuje správnou funkcionalitu, je vhodné otestovat další vlastnosti jako například bezpečnost nebo výkonost služby.

Funkcionální testování

Cílem funkcionálního testování v rámci webových služeb je zjištění, zda-li služba je dostupná a její funkcionalita odpovídá její specifikaci. V první řadě se testuje spojení. Ve většině případů se jedná o HTTP požadavek. Přijaté odpovědi jsou pak podrobeny patřičné analýze.

Testování odolnosti

Testování odolnosti software dříve zmíněno nebylo, protože bývá zahrnuta do obvyklého testování. V případě webových služeb však nabývá zvláštního významu, protože se jedná o distribuovanou technologii. Testování odolnosti se zabývá simulováním prostředí, kde se vyskytuje větší či menší množství chyb. Chyby jsou vkládány do dat, se kterými aplikace pracuje, do komunikace mezi jednotlivými částmi aplikace apod. Do kategorie testování odolnosti spadá i technika injekce chyb, kdy se testuje reakce webové služby na nevhodný nebo poškozený vstup.

Injekce chyb představuje techniku testování odolnosti aplikace. Při injekci chyb se záměrně vkládají chyby buď do zdrojového kódu před kompilací, nebo za běhu programu. Při vkládání chyb do zdrojového kódu programu před kompilací se jedná o tzv. Compile-Time Injection. Tuto techniku je možno použít jak pro simulování softwarových chyb, tak i hardwarových. Technika založená na vkládání chyb za běhu programu se nazývá Run-Time Injection.Pro injekci chyb do webových služeb lze využít techniku vkládání chyb za běhu programu. Více informací obsahuje diplomová práce Martina Žouželky, na kterou tato práce navazuje.

Zátěžové testování

Webová služba je druh webové aplikace. Může k ní přistupovat velké množství uživatelů. Proto je nutné vědět kolik uživatelů je daná webová služba schopna obsloužit za určitý časový okamžik. V tomto směru existuje více způsobu jak pohlížet na výkonnost webové služby. Je možné měřit počet zpracovaných požadavků za sekundu, počet uživatelů při zachování stejné latence. Zátěžové testy se však musí provádět automatizovaně, protože je obtížné testovat např. obsluhu 1000 uživatelů.

Bezpečnostní testování

Bezpečnost je velmi důležitou vlastností webových služeb. Také je obtížné jí dosáhnout kvůli jejich vlastnostem. Bezpečnostní rizika představují sebe-popisující zprávy, které řeknou o webové službě velmi mnoho informací. Dále pak zprávy mohou putovat nepředvídatelnou cestou.

U webových služeb se provádí tzv. penetrační testování, které má za cíl odhalit zranitelnosti webové služby různými druhy útoků. Další způsob pasivně odchytává komunikaci webových služeb a analyzuje obsah zpráv. Tedy k bezpečnostnímu testování lze přistupovat aktivním nebo pasivním způsobem.

3.6 Již existující nástroje pro testování webových služeb

Cílem této práce je vytvořit nástroj pro testování webových služeb. Při vytváření nejen programových nástrojů by se měl tvůrce zabývat tím, zda-li už nebyl podobný nástroj vytvořen. Takovýto průzkum je důležitý, aby se nástroj nevytvářel zbytečně znovu, kvůli inspiraci funkcionality nebo právě kvůli chybějícím funkcionalitám v již existujících řešeních.

V této části práce jsou vybrány nástroje související s funkčním a zátěžovým testováním. Jako poslední bude stručně popsán nástroj FIWS, který je výsledkem předcházející diplomové práce.

soapUI

soapUI [4] je nástroj s otevřeným kódem zaměřený na testování. Disponuje velkým množstvím funkcí včetně možnosti testování webových služeb. Umožňuje testovat SOAP i REST webové služby funkcionálními testy. Pokud je doinstalováno rozšíření loadUI, lze provádět i zátěžové testy. Rozšíření loadUI disponuje možností distribuovaného testování. Uživatelské rozhraní je intuitivní a snadné na ovládání. Projekt disponuje i zásuvným modulem pro prostředí Eclipse i NetBeans, ale tento modul není příliš stabilní a pro pohodlnější vytváření a provádění testovacích scénářů je lepší použít nativní uživatelské rozhraní.

Projekt lze provázat s automatizačním nástrojem Apache Ant. Po vytvoření a vyladění testovacích scénářů pomocí nativního uživatelského rozhraní lze tyto testovací scénáře provádět automaticky s použitím nástroje Ant. Tento způsob integrace je velmi efektivním řešením a tato práce bude tímto způsobem inspirována. Zároveň lze integrovat soapUI s JUnit, což je implementace testování modulů v jazyce Java.

JMeter

JMeter™ [2] je alternativou k soapUI, avšak nenabízí tak široký záběr funkcionalit jako soapUI. JMeter je specializován na testování webových služeb. Disponuje distribuovaným modelem pro testování webových služeb a možností integrace s automatizačními nástrojem Ant a testování modulů JUnit. Uživatelské rozhraní je obdobné jako u soapUI a je také intuitivní. Projekt JMeter nedisponuje zásuvným modulem do vývojových prostředí. Další odlišností od soapUI je možnost upravit si vzhled XML výstupů pomocí rozšířitelného jazyka pro styly (XSL, eXtensible Stylesheet Language).

The Grinder

Tento nástroj je opět velmi podobný předchozím. Jeho hlavní odlišnost spočívá v jeho ovládání. The Grinder [6] je uzpůsoben k ovládání pomocí skriptů v jazycích Jython a Closure. The Grinder je vytvořen v jazyce Java, tedy i jazyky Jython a Closure jsou používány pro ovládání The Grinder v rámci Java virtuálního stroje. Tento způsob ovládání nabízí širokou škálu možností, na druhou stranu vyžaduje znalost jednoho z těchto jazyků. The Grinder ve verzi 2 nabízí zásuvný modul pro testování s JUnit, avšak ve verzi 3 byl odstraněn. The Grinder disponuje zásuvným modulem pro vývojové prostředí Eclipse, který umožňuje spouštění a ladění testovacích skriptů.

Předchozí diplomová práce - FIWS

FIWS reprezentuje nástroj pro injekci poruch. Je specializován na vkládání poruch do komunikace webových služeb pomocí proxy monitorovací jednotky. Tato jednotka zachytává příchozí a odchozí HTTP komunikaci, analyzuje ji, případně modifikuje procházející zprávu. Nástroj je možno ovládat pomocí uživatelského rozhraní. Kompletní popis obsahuje daná diplomová práce.

Kapitola 4

Specifikace požadavků

V této části bude rozšířeno a upřesněno oficiální zadání. Specifikace požadavků je důležitou součástí vývoje softwaru, protože v této fázi jsou jednoznačně definovány požadavky na vytvářený produkt. Nejdříve budou specifikovány základní funkce nástroje. Další dvě části detailněji specifikují požadavky ze zadání. Dále pak bude následovat rozšíření v podobě distribuované architektury.

4.1 Základní koncept

Základní koncept obsahuje základní požadavky, bez kterých by nástroj byl nefunkční. Bude obsahovat požadavky na klientskou aplikaci.

Základní funkce nástroje

Nástroj bude umožňovat vytvoření testovacích dat. Testovací data se budou vytvářet automatizovaně z dokumentů WSDL, pomocí grafického uživatelské rozhraní nebo je bude možno importovat již připravené. Testy bude možno upravovat. Testy vygenerované pomocí WSDL dokumentu budou dále v textu nazvány jako kostra testu, protože budou vytvořeny testy pouze v základní podobě a často je bude třeba upravit.

Další základní funkcí je spuštění testů. Testy bude možné spouštět z příkazové řádky. Nástroj bude obsahovat grafické uživatelské rozhraní, kde mohou být obsaženy další rozšíření.

Výsledky jednotlivých testů budou uloženy a budou dostupné i po ukončení běhu nástroje.

Testovací jednotka

Nástroj bude obsahovat jádro pro spouštění testů. Tato jednotka bude umožňovat spuštění připravených testů. Jednotku bude možno spouštět jak z příkazové řádky, tak z grafického prostředí. Tato jednotka bude základem pro rozšiřující vlastnosti popsané dále ve specifikaci.

Testování SOAP webových služeb

Nástroj bude umožňovat testování SOAP webových služeb. To znamená, že jednotka bude podporovat manipulaci se SOAP zprávami a vytváření HTTP požadavků. Manipulaci se rozumí odeslání SOAP zprávy a přijetí její odpovědi, pokud to charakter testováni bude vyžadovat.

Testování REST webových služeb

Nástroj bude umožňovat testování REST webových služeb. To znamená, že jednotka bude podporovat HTTP příkazy, které jsou popsány v sekci REST webové služby. Bude také ukládat odpovědi od služby, pokud to charakter testování bude vyžadovat.

Uživatelské rozhraní pro příkazovou řádku

Nástroj bude možno ovládat pomocí příkazové řádky. Z příkazové řádky bude možné ovládat základní funkce nástroje tzn. konfigurace testů, spuštění testů, uložení výsledků testů.

Grafické uživatelské rozhraní

Grafické uživatelské rozhraní bude umožňovat přehledné vytváření testů dle požadavku na strukturu. Testy bude možno vytvářet upravovat i rušit. Testy bude možno aplikovat spuštěním jednotky pro spouštění testů. Rozhraní bude umožňovat přehledné zobrazení vybraných výsledků testů. V grafickém uživatelském rozhraní dále bude možné ovládat vzdálené procesy, které budou umožňovat předání vytvořených testů a spuštění testovací jednotky.

Struktura testovacích dat

Jednotlivé testy budou moci být vytvářeny ve větších celcích, které spolu souvisejí. Například všechny testy pro jednu službu budou uloženy ve společné složce na disku.

Automatizace generování vzorů pro testování

Pokud je služba popsána standardizovaným způsobem (WSDL, WADL apod.) bude možno pro tuto službu vygenerovat základní sadu testů na základě těchto dokumentů.

4.2 Integrace s nástroji pro testování spolehlivosti

Tento nástroj bude nabízet pouze některé možnosti testování webových služeb, proto je velmi vhodné umožnit uživateli použití tohoto nástroje společně s dalšími nástroji podobného charakteru a docílit tak komplexních možností testování.

Automatizace provádění testů

Spouštění testovací jednotky bude možné provádět pomocí automatizačních nástrojů jako např Apache Ant, aby bylo možné vytvářet skripty, které by umožňovaly integraci více nástrojů do jediného skriptu.

4.3 Distribuované testování

Tato sekce specifikace bude zaměřena na popis požadavků pro distribuované provádění testů. Toto rozšíření není součástí zadání, ale je velmi vhodné využití distribuované architektury pro zátěžové testování.

Správa procesů na vzdálených strojích

Proces testovací jednotky musí nejdříve běžet na vzdáleném stroji, pak bude možné otestovat konektivitu s touto vzdálenou jednotkou. Vzdálené jednotky bude možné testovat (konektivitu) a ukončovat z grafického uživatelského rozhraní i z rozhraní příkazové řádky.

Spuštění testů na vzdálených strojích

Z obou uživatelských prostředí bude taky možné zaslat požadované testy určité jednotce či více jednotkám a následně jim odeslat příkaz ke spuštění testování případně příkaz ke zastavení testování.

Odesílání výsledků testů hlavnímu procesu

Výsledky testů se budou zaznamenávat a bude na výběr, zda-li je posílat automaticky centrálnímu ovládání nebo na pouze na vyžádání příkazem.

4.4 Navázání na předchozí práci

Protože tato práce navazuje na diplomovou práci vztahující se k injekci chyb v komunikaci webových služeb, je třeba předchozí výsledky rozšířit a zapracovat nové možnosti. Dále pak jednotlivé součásti budou spolupracovat tak, aby pomocí jednoho nástroje bylo možné provádět zátěžové testování i testování s injekcí chyb.

Umožnění vytváření testovacích případů pro injekci chyb

Nástroj bude umožňovat vytvoření testovacích dat pro injekci chyb, které budou použity HTTP proxy jednotky, která je součástí předchozí práce.

Spuštění HTTP proxy procesu pro vkládání chyb do komunikace

Společně s možností spuštění testovací jednotky musí být vytvořena možnost spuštění HTTP proxy jednotky.

Podpora dalších formátů dat pro injekci chyb

HTTP proxy jednotka bude podporovat další formáty dat, do kterých bude možno vkládat chyby. Dalšími formáty jsou myšleny některé reprezentace dat jako například JSON, YAML, HDF.

Podpora komprimovaných odpovědí od webových služeb

HTTP proxy jednotka bude rozšířena o podporu analýzy komprimovaných odpovědí od webových služeb. Je tedy nutné implementovat podporu pro čtení dat komprimovaných typem GZIP.

Kapitola 5

Návrh implementace

Tato sekce popisuje další část procesu vývoje softwaru. Návrh je velmi důležitou součástí vývoje, protože kvalitní návrh šetří prostředky v dalších fázích vývoje. V návrhu je uveden základní koncept nástroje, který se skládá z několika součástí. Některé součásti budou převzaty z předchozí diplomové práce a upraveny tak, aby splňovaly požadavky ze zadání.

Návrh vychází z volně dostupných nástrojů soapUI a Grinder. Návrh také respektuje předchozí diplomovou práci a do nástroje bude zakomponována podpora pro injekci chyb. Vzhledem k tomu, že jde o navazující práci, bude vytvořenému nástroji ponechán jeho název FIWS (Fault Injektor for Web Services) a implementovaná rozšíření budou obsaženy ve verzi 2.0.

K implementaci jsou vhodné platformy Java a .NET. Platforma Java nabízí multiplatformní řešení a navíc je v ní vytvořena předchozí práce. Je však vhodné vzít v úvahu i druhou možnost, protože vzhledem k webovým službám nabízejí obě platformy velmi podobné možnosti implementace.

5.1 Základní koncept

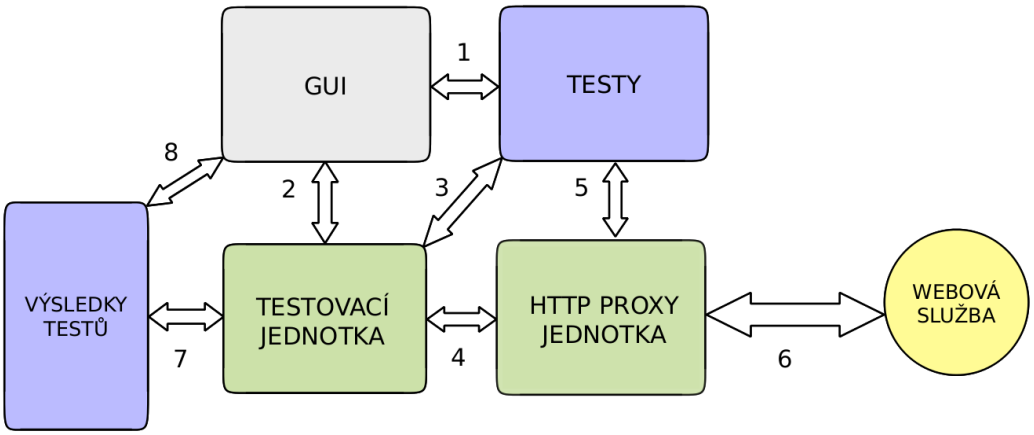
Základní koncept je znázorněn na obrázku 5.1. Základem FIWS 2.0 je testovací jednotka, která bude hrát roli univerzální klientské aplikace pro webové služby, která je schopna odesílat a přijímat HTTP požadavky webovým službám v požadovaném objemu a rychlosti s ohledem na možnosti systému, na kterém je spuštěna. Data, která bude testovací jednotka odesílat budou předem připravena a budou testovací jednotce explicitně předána. Bude tedy možné předávat data pro odesílaní podle specifikovaného umístění. Data použitá pro odesílání musí mít vždy stejný formát. To platí zejména pro data, která nebudou vytvořena v nativním uživatelském rozhraní.

Uživatelské rozhraní bude vytvořeno v grafické podobě i v podobě ovládání z příkazové řádky. Testovací jednotka se bude moci ovládat pomocí dvou rozhraní.

Injekce chyb bude realizována odděleným procesem, který bude hrát roli proxy serveru. Na obrázku znázorněna jako HTTP proxy jednotka. Bude monitorovat veškerou komunikaci mezi testovací jednotkou a webovou službou. Spuštěná proxy jednotka obsahuje analyzátor a injektor chyb. Více o injekci chyb je možno se dočíst v předchozí práci.

Tento koncept lze také použít pro vytvoření distribuované architektury, protože dovolí oddělit testovací jednotku od uživatelských rozhraní. Oddělení bude spočívat v lehce rozdílném komunikačním rozhraní. Testovací jednotka společně s proxy jednotkou dostupné přes vzdáleně dostupný proces, který bude mít Java RMI rozhraní a bude muset být spuštěn

dříve než se s ním bude vzdáleně manipulovat.



Obrázek 5.1: Základní koncept.

Základní použití FIWS 2.0 uživatelem

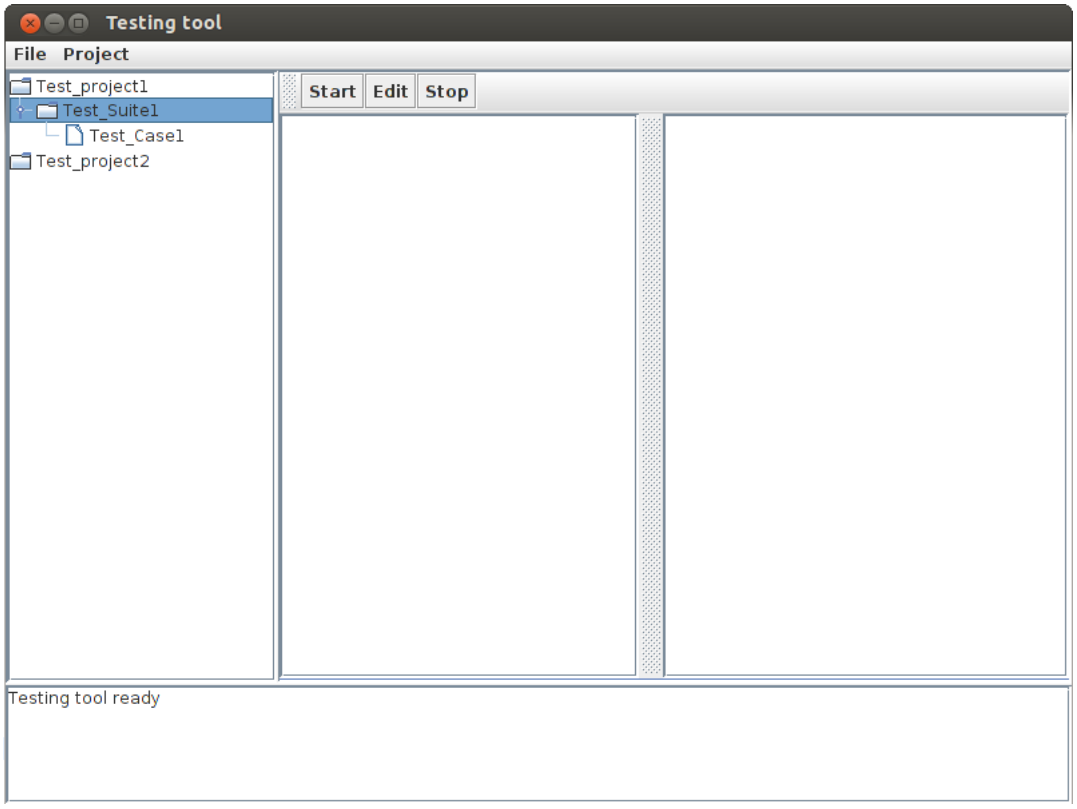
Základní použití FIWS 2.0 znázorňují očíslované šipky, jejichž význam bude nyní popsán. Nejdříve si uživatel v grafickém uživatelské rozhraní vytvoří testovací data (1). Následně spustí testovací jednotku (2), která si vyzvedne data ze specifikovaného úložiště (3). Testovací jednotka se pak spojí přes proxy HTTP jednotku (4) s webovou službou (6). Mezi tím proxy jednotka za určitých okolností vloží chybu do komunikace(5). Testovací jednotka poté přijme odpověď, která také putuje přes proxy jednotku a všechny výsledky uloží do specifikovaného úložiště (7). Uživatel pak bude moci z grafického uživatelského rozhraní vyhodnotit výsledky (8).

Grafické uživatelské rozhraní

Grafické uživatelské rozhraní bude sloužit hlavně k efektivní přípravě testovacích dat. Musí obsáhnout možnosti vytvořit HTTP požadavek i s různými datovými reprezentacemi uvnitř datové části požadavku. REST webové služby vyžadují různé typy HTTP požadavků včetně konfigurace datové části HTTP požadavku. SOAP zprávy vyžadují konfiguraci zejména datové části, tedy vytvoření XML dokumentu.

Testovací data se budou vytvářet v projektech. Než uživatel bude moci zahájit jakoukoliv práci, musí vytvořit nový projekt nebo použít již dříve vytvořený. Struktura projektu bude obsahovat sada testů a jednotlivé testy, které mohou být vytvořeny uvnitř sady testů. Struktura a nastavení projektu bude popsána v XML dokumentu.

Na obrázku 5.2 je návrh hlavního okna nástroje, kde je znázorněno rozložení ovládacích komponent. Na obrázku 5.3 je znázorněno šablona dialogového okna, která se bude používat pro zadávání různých druhů vstupů a nastavení nástroje.



Obrázek 5.2: Návrh hlavní obrazovky nástroje.

Uživatelské rozhraní z příkazové řádky

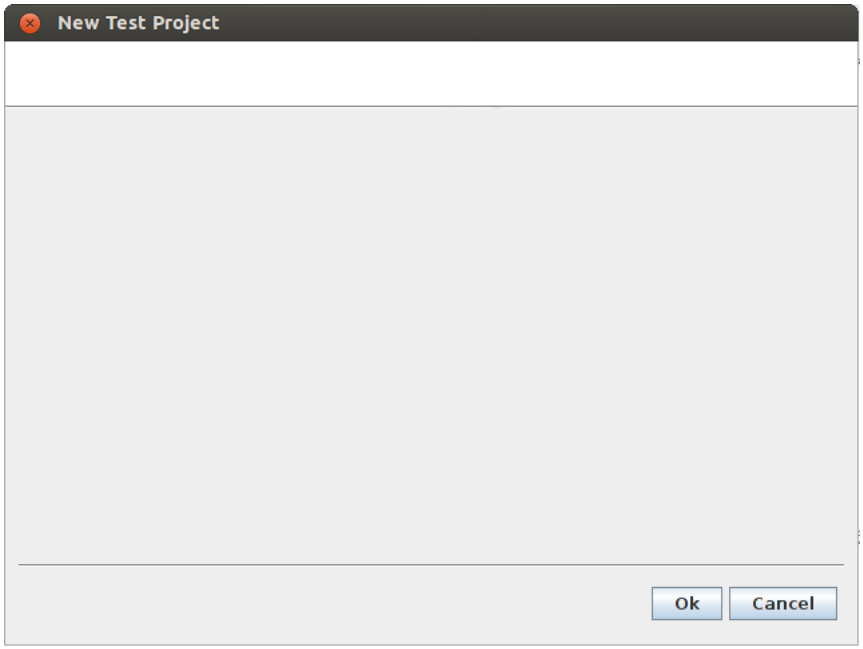
Testovací jednotka i HTTP proxy jednotka budou vybaveny rozhraním pro použití těchto jednotek z příkazové řádky. Budou přebírat potřebné parametry pro lokalizaci testovacích dat. Z příkazové řádky bude možné konfigurovat obě jednotky pomocí přepínačů nebo pomocí konfiguračního souboru, kde budou požadované parametry a jejich hodnoty uvedeny formátem klíč='hodnota'.

Testovací data a vyhodnocení testů

Testovací data se budou skládat z předem připravených HTTP požadavků nebo z dat pro injekci chyb. Data pro injekci chyb se skládají z uložených podmínek a následné injekce vybrané chyby. Každý druh testovacích dat bude umístěn v adresáři, který bude označovat jejich druh. Takto budou moci jednotky vybírat pro ně určená data. Toto je důležité při přiřazení sady testů některé z jednotek, protože nebude nutné explicitně uvádět, která data přísluší které jednotce.

Testovací jednotka

Testovací jednotka je jádrem celé práce. Bude zodpovědná za spojení se s webovou službou a odeslání a přijetí HTTP požadavku. Testovací jednotka převezme nastavení (data, počet spuštěných procesů, objem požadavků apod.) od uživatele a následně provede požadované akce. Mezi základní operace testovací jednotky patří navázání HTTP spojení s požadovanou



Obrázek 5.3: Šablona pro modální okna nástroje.

webovou službou a odeslání patřičných dat. Spojení s webovou službou si vyžádá další funkcionality. Hlavní funkcí při spojení bude definování proxy serveru, aby mohla být použita HTTP proxy jednotka. Mezi další pak bude patřit možnost autentizace a v neposlední řadě je velmi vhodná implementace bezpečnostních protokolů např. možnost použití HTTPS.

5.2 Integrace s nástroji pro testování spolehlivosti

Integrace s ostatními nástroji bude možné díky rozhraní pro příkazovou řádku. S pomocí ovládání z příkazové řádky bude možné použít FIWS 2.0 společně s automatizačními nástroji jako je například Apache Ant, který umožňuje integrovat více nástrojů pro testování do jediného skriptu.

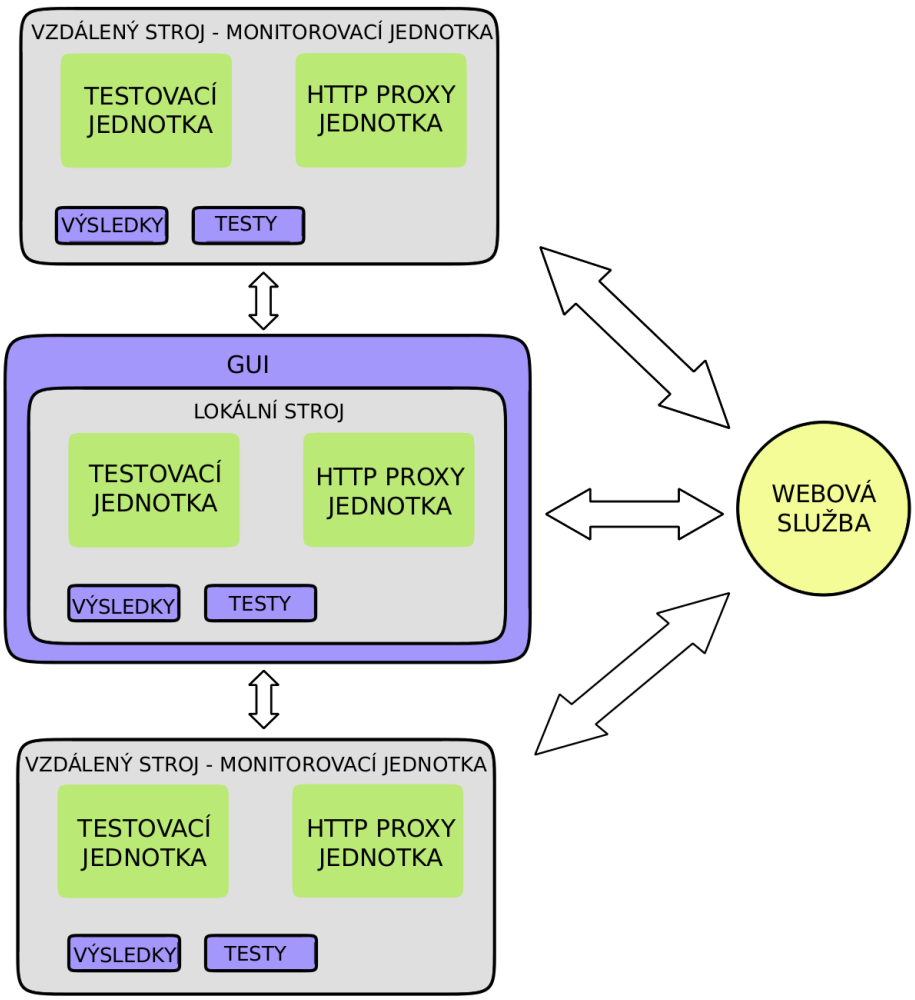
5.3 Distribuovaný model

Distribuovaný model odesílání požadavků je znázorněn na obrázku 5.4. Tento způsob odesílání umožní reálnější simulaci vysokého počtu uživatelů, protože požadavky budou odesílány z různých fyzických strojů.

Pro implementaci správy jednotlivých procesů na vzdálených strojích je možno použít technologii RMI, kterou nabízí platforma Java. V tomto případě musí být obě strany vybaveny rozhraním RMI pro vzdálenou komunikaci. Na vzdáleném stroji musí běžet monitorovací jednotka, která bude zprostředkovávat vzdálené příkazy pro testovací a proxy jednotky.

V grafickém uživatelském prostředí bude možnost spojit se s již běžícími instancemi monitorovacích jednotek a centralizovaně spravovat vzdálené stroje. V grafickém uživatelském prostředí bude pro každé vzdálené spojení vytvořeno zvláštní vlákno, protože tento přístup

zajišťuje vhodnou obsluhu jednotlivých spojení.



Obrázek 5.4: Model distribuované architektury.

Centralizovaná správa a výměna dat

Všechny vzdálené jednotky budou spravovány z jednoho centrálního bodu, kde bylo spuštěno grafické uživatelské rozhraní. Ovládání z příkazové řádky bude také možné. Při ovládání vzdálených jednotek bude možné provést test konektivity, distribuce dat a spuštění vybraných jednotek. Pomocí skriptů (Bash, Perl, apod.) bude možné distribuovat vybraná data vybraným jednotkám.

Centralizovaná správa všech jednotek je vhodná kvůli vytvářením a následné distribuci testovacích dat. V centrálním bodě se vytvoří testovací data a rozešlou se na vzdálené uzly, kde budou moci být využity. Výsledky testů budou zpět odeslány okamžitě po jejich získání a uloženy pro zobrazení a vyhodnocení uživatelem.

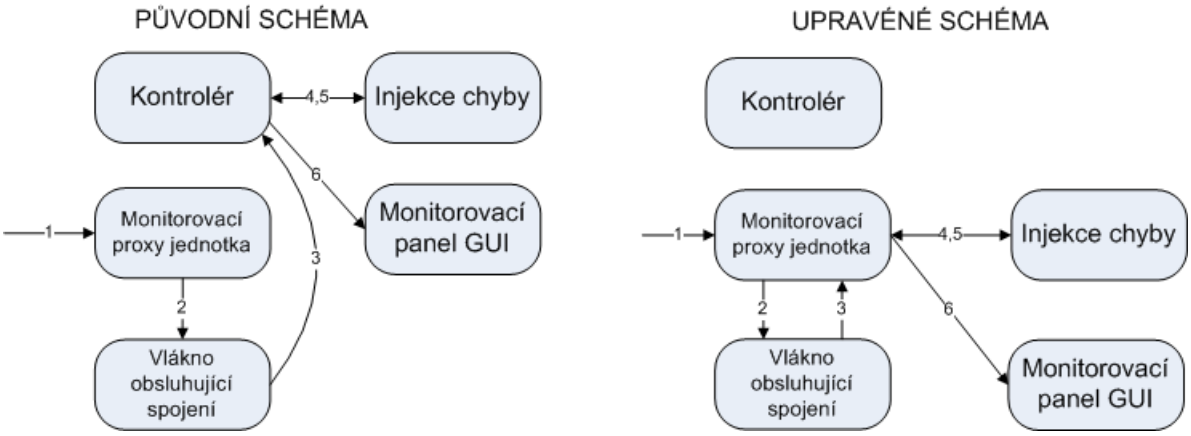
5.4 Použití injekce chyb

HTTP proxy jednotka bude vytvořena ze zdrojových kódů předchozí práce a bude přizpůsobena ke samostatnému spuštění. Funkce proxy jednotky bude spočívat v zachycení komunikace mezi testovací jednotkou a webovou službou. Tento princip lze využít k zátěžovému testováni spojeném s injekcí chyb.

Distribuovaný režim

Na základě prostudování zdrojových kódů a otestování původní implementace bylo zjištěno, že bude nutné provést změny v původním návrhu . Na obrázku 5.5 je znázorněno stávající zpracování zprávy proxy serverem a jeho plánovaná změna. Pro bližší pochopení, jak funguje stávající implementace a jak bude fungovat plánovaná implementace, jsou v obrázku umístěna čísla, jejichž význam je popsán níže.

Proxy monitorovací jednotka bežící v nekonečném cyklu přijme zprávu (1) a vytvoří nové vlákno pro jeho zpracování (2). Vlákno zpracuje zprávu a odešle jí kontroléru. Kontrolér předá zprávu injektoru(4,5) poruch, kde se provede analýza zprávy a její případné narušení. Kontrolér zastává funkci delegáta. Zobrazovací panel GUI je přiřazen ke kontroléru. Kontrolér tedy odesílá(6) zprávu do panelu GUI, kde je zobrazena. Po zobrazení je zpráva vložena na výstup síťového rozhraní a pokračuje určeným směrem. Tento stav je nevyhovující, protože při tomto stavu je komplikované vytvořit více monitorovacích jednotek na více strojích.



Obrázek 5.5: Schéma zpracování zprávy před a po úpravě proxy serveru.

Změna spočívá v přesunutí úlohy delegáta na monitorovací jednotku a kontrolér bude pouze provádět přiřazení monitorovacích panelů GUI v monitorovacích jednotkách. Pak je možné mít více nezávislých monitorovacích jednotek na různých strojích. Tyto jednotky pak budou odesílat získaná data do centrálního GUI, kde dojde k jejich zobrazení a zpracování. Stejného principu se využije při detailním návrhu architektury.

Vylepšení jádra proxy jednotky

Pro vylepšení vlastností monitorování proxy jednotky jsou nutné další zásahy do implementace. Tato sekce se zaměří na rozbor a návrh dalších funkcí proxy jednotky. Zejména se

bude jednat o interpretaci dalších způsobů kódování a komprimace zachycovaných zpráv. Dále také bude uvedeno seznámení s dalšími reprezentacemi dat, které je možné analyzovat pomocí injektoru poruch a případně narušit jejich integritu.

Položka *Content-Encoding* udává zakódování odesílaného obsahu na straně koncových bodů. Komprimace uvedená v této položce probíhá **pouze** na straně klienta nebo serveru.

Položka *Transfer-Encoding* udává zakódování obsahu zprávy během transportu sítí. Komprimace může proběhnout kdekoliv na cestě zprávy mezi koncovými body, včetně koncových bodů samotných. Toto zakódování probíhá až po komprimaci udávané v položce *Content-Encoding*. Pokud jsou v hlavičce HTTP zprávy obě položky znamená to, že zpráva byla zakódována nejdříve způsobem uvedeným v položce *Content-Encoding* a následně ještě způsobem uvedeným v položce *Transfer-Encoding*. Zpráva může mezi koncovými body být zpracována několika proxy jednotkami, kdy pokaždé může dojit ke změně zakódování. Pokaždé se však mění pouze pozdější způsob zakódovaní, tedy *Transfer-Encoding*.
Možnosti pro vylepšení chování monitorovací jednotky:

- **Komprimace zpráv**
  Obsah HTTP zprávy je možno před odesláním komprimovat. Často používanými způsoby jsou *GZip* a *Deflate*. Pokud se do hlavičky uvede způsob komprimace a daný interpret je schopný tento způsob zpracovat, je možné použít libovolný způsob komprimace. Proxy jednotka bude vylepšena o zpracování způsobů *GZip* a *Deflate*. Bude schopná zprávu dekomprimovat, analyzovat, narušit, následně komprimovat a odeslat. Pro komprimaci a dekomprimaci lze použít volně dostupné knihovny.

- **Odesílání po částech**
  Možnost odesílat zprávu po částech (*chunked encoding*) byla zavedena do HTTP protokolu od verze 1.1. To je výhodné v případech náročnějšího zpracování dat, kdy se nemusí čekat až jsou všechna data zpracování do jedné zprávy. Při odesílání po částech je odeslána HTTP hlavička a za ní jsou odeslány jednotlivé části. Jednotlivé části zprávy jsou oddělené prázdným řádkem. Každá část zprávy má na prvním řádku uvedenou svou velikost a na dalším řádku je obsah zprávy. Tento způsob přenosu však výrazně zvyšuje obtížnost injekce chyby. Proto proxy jednotka poskládá jednotlivé jednotlivé části dohromady, provede analýzu a případné narušení a odešle zprávu dále jako celek. Je také vhodné obsah zprávy před odesláním komprimovat.

- **Monitorování bez injekce**
  V předešlé verzi proxy jednotky nebylo možné zachycovat zprávy bez možnosti narušení. Byly tedy zaznamenány pouze zprávy, které vyhověly podmínce v injektoru chyb. Toto chování bude vylepšeno a bude zaznamenán veškerý provoz. Pokud proběhne injekce chyby, bude nastaven příznak narušení zprávy.

Podpora dalších datových formátů

Předešlá verze podporovala zpracování pouze formátu XML. V ostatních případech nebyla data proxy jednotkou vůbec zaznamenána. Nyní je proxy jednotka bude nejprve číst obsah parametru *Content-Type* a podle toho rozhodne, jakým způsobem data zpracuje. Pokud parametr nebude přítomen budou data odeslána na výstup bez zpracování. Další možnosti pro podporu dalších datových formátů injektoru poruch:

- **JSON**
  JSON je rozšířený datový formát používaný především webovými službami REST charakteru. Data reprezentována JSON formátem jsou v textové podobě. Lze tedy uplatnit stávající injekce poruch. Pokud však je nutné komplexnější injekci, jako například změna určité proměnné, je nutné nejdříve zpracovat data vhodným způsobem. Pro komplexnější zpracování je možno využít volně dostupných knihoven.

- **YAML**
  YAML je svou charakteristikou i použitím velmi podobný formátu JSON. Také se jedná o reprezentaci dat v textovém formátu. Bohužel implementace nebyla nalezena knihovna, která by umožňovala zpracování dle modelu YAML bez přetypování na známý Java objekt nebo kolekci. Pokud není známa implementace objektu, není možné jednoznačně určit typ objektu, na který má být přečtená data přetypována. Dostupné knihovny pro JSON formát umožňují manipulaci s JSON pomocí modelu, tedy není nutné přetypování na známý objekt nebo kolekci. Tento formát tedy nebude proxy jednotkou přímo podporován. Z výše uvedených důvodů bude s tímto formátem zacházeno jak s neformátovaným textem.

Přidání podmínek a poruch

- **Podmínka: Test existence položky v hlavičce HTTP zprávy**
  Bude přidána podmínka pro detekci položky v hlavičce HTTP zprávy. Bude možno zadat název parametru a jeho hodnotu. Pokud nebude nebude hodnota zadána bude se testovat pouze výskyt parametru.

- **Porucha: Vložení hodnoty do proměnné v JSON formátu**
  Jelikož bude přidána podpora pro datový formát JSON, bude možno zadat název proměnné, kam se vloží požadovaná hodnota. Tato porucha byla zvolena, protože neočekávané hodnoty, dokáží vyvolat neočekávané chování při nedostatečném ošetření vstupů. Obzvláště pak hodnota *null* je velmi kvalitním příkladem.

5.5 Návrh vrstev

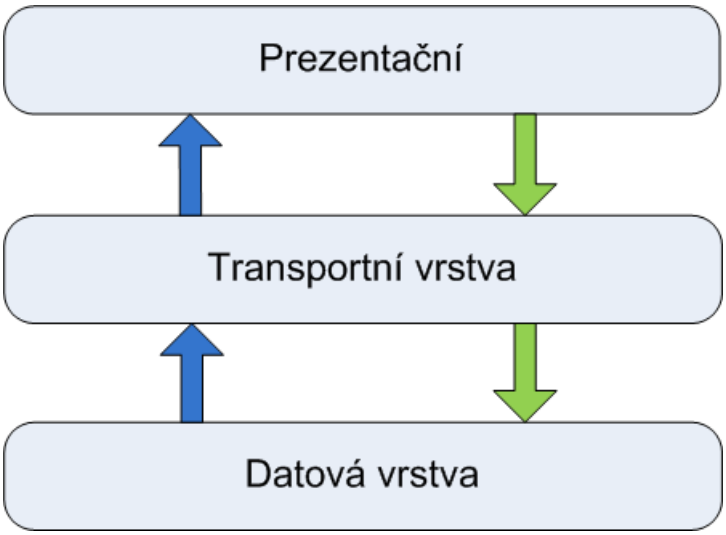
Návrh obsahuje distribuovanou architekturu a je vhodné rozvrhnout jednotlivé části do oddělených vrstev. Rozdělit aplikaci při návrhu do vrstev je výhodné, protože usnadňuje dekomponovat složitější problémy na podproblémy a snížit tak složitost následné implementace. Obrázek 6.4 ukazuje rozložení jednotlivých vrstev.

- **Prezentační vrstva**
  Prezentační vrstva obsahuje uživatelská rozhraní (grafické, příkazovou řádku). Prezentační vrstva slouží k přijmutí příkazů a dat od uživatele. Dále slouží k zobrazení požadovaných výsledků.

- **Transportní vrstva**
  Tato vrstva je zodpovědná za distribuci dat v rámci aplikace. Bude přijímat data od sousedních vrstev a plnit funkci prostředníka, při předávání dat mezi prezentační a

datovou vrstvou.

- **Datová vrstva**
  Tato vrstva zprostředkuje zápis dat na disk a jejich čtení. Manipulaci (čtení/zápis) s daty je možno realizovat několika způsoby. Díky oddělení od zbytku aplikace je možno implementaci uložení dat měnit v dalších verzích programu, aniž by došlo k významnějším zásahům do implementace vyšších vrstev.

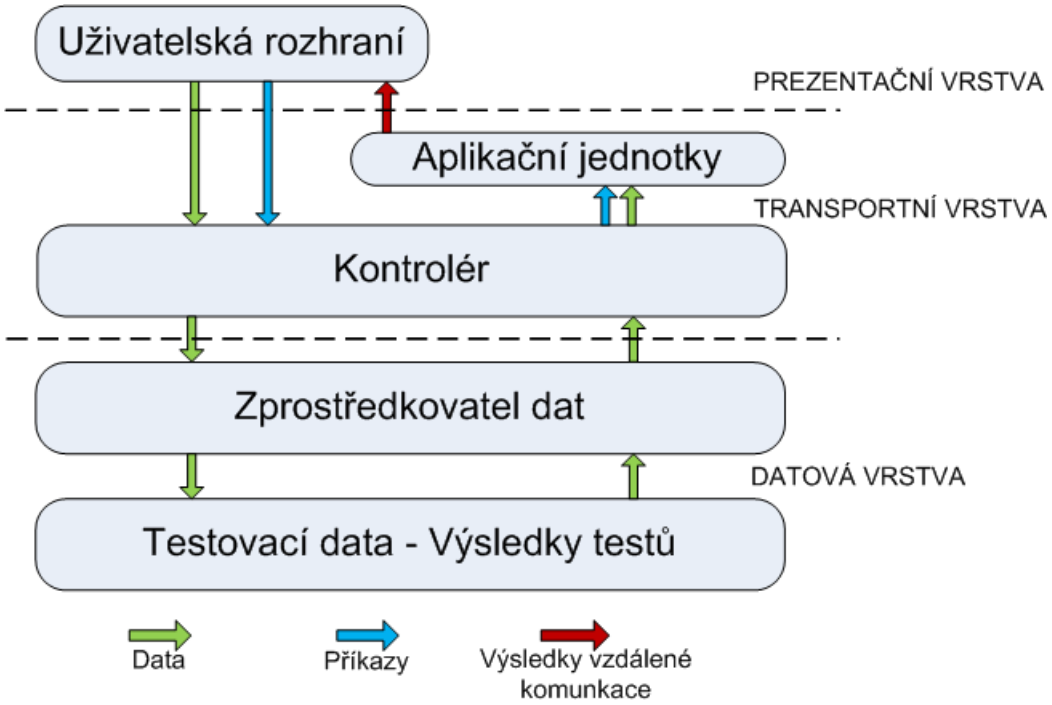


Obrázek 5.6: Návrh vrstev aplikace.

5.6 Detailní návrh architektury

V této sekci budou spojeny jednotlivé postupy z předešlých sekcí a bude představen detailní návrh architektury vytvářeného nástroje. V sekci jsou nastíněny jednotlivé součásti nástroje a jak spolu mají komunikovat. Aby tato komunikace šla snadno implementovat byl představen návrh vrstev, do kterých budou jednotlivé součásti rozmístěny, tak aby spolu komunikovaly pouze sousední vrstvy. Tento způsob návrh je výhodný kvůli rozdělení složitějších problému na podproblémy. Takováto dekompozice také usnadňuje případné testování jednotlivých součástí aplikace (Unit testing).

Jelikož tato práce navazuje na již vytvořenou implementaci, která nevyhovovala striktnímu rozdělení do vrstev, bylo nutné vytvořit návrh úprav, který je na obrázku . Po provedení úprav bude možno využít stávající implementaci a přidat k ní další prvky, které vyžaduje zadání práce.
Na obrázku 5.7 je model rozdělení jednotlivých součástí aplikace do navržených vrstev. Do prezentační vrstvy jsou jsou umístěna uživatelská rozhraní. Do transportní vrstvy je umístěn kontrolér, protože se stará o transport dat a příkazů mezi sousedními vrstvami. Jak je z obrázku patrné, uživatelská rozhraní jsou zodpovědné za vytváření dat, odesílání příkazů a zprostředkování výstupů z aplikačních jednotek. Uživatelská rozhraní komunikují pouze s kontrolérem a jsou odstíněna od odstaních operací.



Obrázek 5.7: Zobrazení implementace základního modelu.

Do transportní vrstvy jsou zahrnuty aplikační jednotky a kontrolér. Aplikační jednotky slouží ke vzdálené komunikaci s webovými službami a získávání dat. Aplikační jednotky v rámci aplikace komunikují pouze s kontrolérem a jsou tak odstíněny od uživatelských rozhraní, čtení a ukládání dat.

Kontrolér je zodpovědný za přijímání dat od prezentační vrstvy a dle druhu komunikace deleguje data do aplikačních jednotek nebo data odesílá datové vrstvě k uložení. Pokud vyžaduje čtení dat, kontrolér deleguje zprávu do datové vrstvy a přijme již data v požadovaném formátu. Hlavní úlohou kontroléru je delegace dat a příkazů mezi sousedními vrstvami.

Zprostředkovatel dat má za úkol číst data uložená na disku ve formátu XML. Veškerá data přijatá od vyšších vrstev budou Java objekty, které se budou ukládat na disk v serializované podobě. Serializaci Java objektů do formátu XML a následné uložení na disk je možno provést některou z volně dostupných knihoven.

Součástí této vrstvou jsou uložená data na disku. Tyto data budou mít hierarchickou strukturu kvůli efektivnější orientaci.

Kapitola 6

Popis implementace

Tato kapitola se věnuje popisu implementace návrhu, který byl představen v předešlé kapitole. Vytvoření jednotlivých částí programu je postupně popsáno v jednotlivých sekcích. U každé části jsou uvedeny použité technologie a postupy. Kapitola je rozdělena na sekce tak, aby bylo zřejmé, že implementace aplikace splňuje zadání.

Nejdříve je popsán výběr implementačního prostředí a jazyka. Pak je představena implementace základních součástí aplikace. V dalších sekcích pak jsou popsány implementace funkcí vytvořeného nástroje. Při popisu jednotlivých funkcí jsou pak použity vytvořené součásti a je vysvětleno jejich propojení a význam v dané funkcionalitě.

6.1 Výběr vývojového prostředí a jazyka

V dnešní době má většina jazyků podporu pro vývoj aplikací používající webové služby. Mezi nimi však vyčnívají platformy .NET a Java, které mají nejširší podporu pro vývoj aplikací pro webové služby. Vzhledem k návaznosti této práce na práci, která je implementována na platformě Java, je vhodnější použití platformy Java. Platforma Java byla také zvolena kvůli zkušenostem autora.

Při implementaci jsou použity součásti Java SE a pro vytvoření testovacích příkladů je použito součástí Java EE. Jako prostředí pro vývoj aplikace je možno použít libovolný nástroj. Pro urychlení vývoje je však vhodné použít některé integrované prostředí (IDE). Velice známá jsou prostředí NetBeans a Eclipse. Obě jsou multiplatformní a poskytují nástroje pro vývoj aplikací na platformě Java. Bylo zvoleno integrované prostředí Eclipse. Eclipse prostředí bylo zvoleno ze subjektivních důvodů, protože autor má toto prostředí v oblibě.

6.2 Základní model aplikace

V návrhu na obrázku 5.1 byl představen základní model aplikace. Dále pak na obrázku 6.4 byl popsány jak by tento koncept měl zapadat do vrstev. V této části je znázorněno jak jsou tyto dva pohledy na návrh aplikovány v implementaci. Obrázek 5.7 slouží jako vizualizace implementace. Jednotlivé části budou podrobně popsány a dále v textu budou uvedeny další diagramy vysvětlující tok dat a příkazů mezi jednotlivými součástmi.

Reprezentace dat

Pro organizaci dat bylo zavedeno hierarchické schéma, které je patrné na obrázku. Na nejvyšším úrovni je kořenový adresář, který není v grafickém uživatelském rozhraní (dále jen GUI) patrný. Kořenový adresář se nachází v pracovním adresáři aplikace a pokud není při startu aplikace přítomen, je adresář vytvořen.

Další úroveň představují testovací sady. Jedna testovací sada je určena pro jednu službu. Testovací sada obsahuje testovací případy k dané službě. Testovací sada dále obsahuje seznam testovacích případů, které budou spuštěny testovací jednotkou. Tento seznam je reprezentován XML souborem *testlist.xml*.

Jednotlivé testovací případy představují další úroveň schématu. Každý testovací případ obsahuje složky pro vstupní a výstupní data pro aplikační jednotky. Tyto složky jsou zobrazeny v GUI jako *FaultInjection* a *Http*. Testovací případ dále obsahuje XML soubor *settings.xml*, kde jsou uvedeny nezbytná nastavení pro testovací případ. Obsah souboru s nastavením a složek pro aplikační jednotky je popsán níže ve strukturovaném seznamu.

- **Data pro nastavení testu**
  Data pro nastavení testu obsahují informace, podle kterých se nastaví aplikační jednotky . V souboru *settings.xml* je uložena instance třídy *TestCaseSettingsData*, která nese informace pro spuštění pro testovací a proxy jednotku.

- **Data pro testovací jednotku**
  Data pro testovací jednotku rozdělena do složek pro vstupní (*input*) a výstupní data (*output*). Tyto složky nejsou v GUI zobrazeny. Ve složce pro vstupní data je XML soubor *input.xml*, kde jsou umístěna data pro testovací požadavek. Do složky pro výstupní data se pak ukládají výsledky testů. Data uložená v souboru *input.xml* představuje serializovanou instanci třídy *HttpMessageData*, která nese potřebné informace o HTTP požadavku.

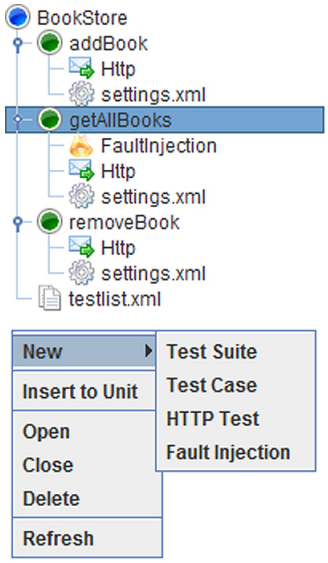
- **Data pro injekci chyb**
  Data pro injekci chyb jsou rozdělena stejným způsobem jako data pro testovací jednotku. Data uložená v souboru *input.xml* je představuje serializovanou instanci třídy *FaultInjectionData*, která nese seznam podmínek a narušení pro injektor poruch (viz.....)

Zprostředkovatel dat

Zprostředkovatel dat je reprezentován třídou *DataProvider*, která byla již v předchozí implementaci. Tato třída má na starosti serializaci a deserializaci Java objektů do respektive

z XML formátu. K tomuto druhu manipulace je použita knihovna Xstream. Veškerá data jsou uložena na disku ve formě textových dat ve formátu XML. Pokud chce nástroj data používat, data načte jako textový řetězec a pomocí knihovny Xstream dojde k deserializaci. Přesně opačným postupem se postupuje při zápisu instancí tříd na disk.

V předchozí implementaci byly metody z této třídy volány pouze staticky. Pro účely byla tato třída přepracována a metody jsou volány z instance dané třídy.



Obrázek 6.1: Náhled schématu dat zobrazeného v navigačním panelu.

Kontrolér

Centrální součást aplikace, která má za úkol delegovat příkazy z uživatelských rozhraní do aplikačních jednotek a zprostředkovat přenos dat mezi jednotlivými součástmi. Další důležitá úloha kontroléru je spuštění vláken pro aplikační jednotky.

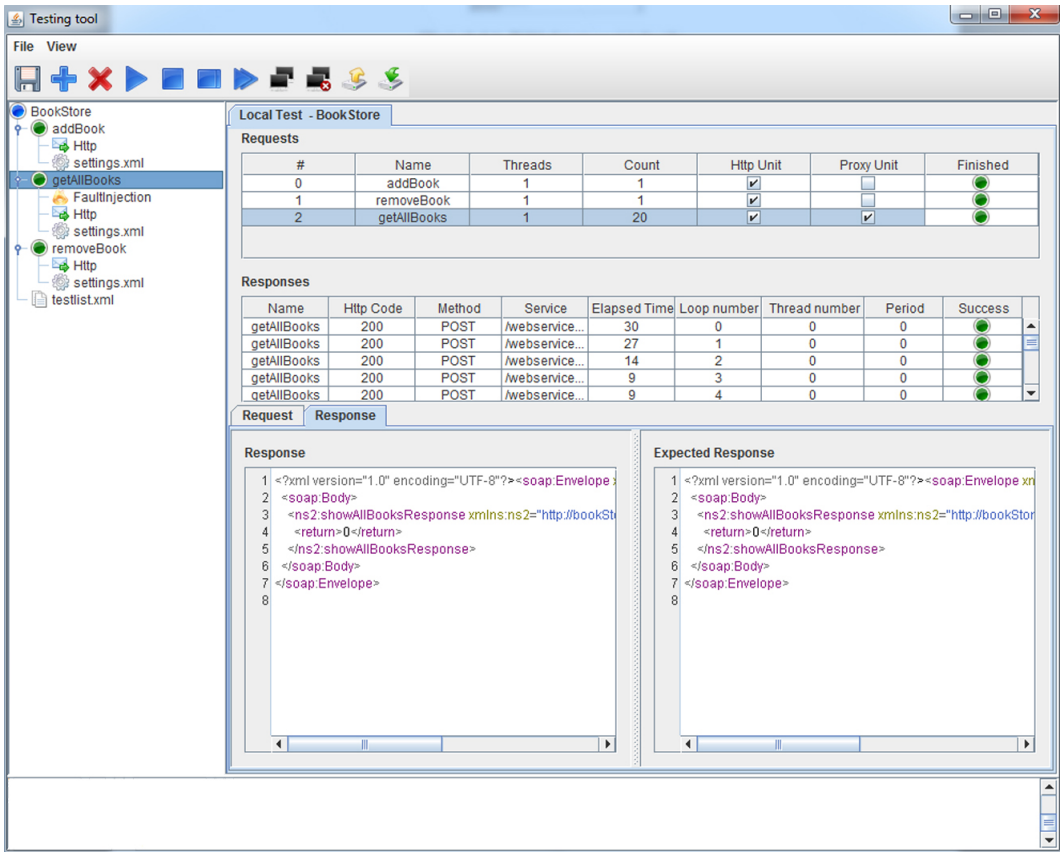
V kontroléru existují třídy ProxyUnitWorker a TestUnitWorker. Tyto třídy rozšiřují třídu *SwingWorker*. Třída *SwingWorker* umožňuje spustit vlákno na pozadí grafického uživatelského rozhraní vytvořeného pomocí sady Java Swing. Při spuštění nástroje se nejdříve vytvoří instance lokálních aplikačních jednotek. Další aplikační jednotky je možno registrovat dynamicky pomocí grafického uživatelského rozhraní. Jakmile z uživatelského rozhraní přijde příkaz ke spuštění jednotky. Instance třídy aplikační jednotky spustí nové vlákno. V nově vytvořeném vlákně pak běží testovací jednotka. Pokud jsou při testu použity obě aplikační jednotky, pak jsou vytvořena dvě oddělená vlákna.

Dále kontrolér má na starosti registraci zobrazovacích panelů do aplikačních jednotek. Kontrolér má uloženy všechny aplikační jednotky v abstraktním datovém typu *HashMap* s asociativním přístupem. Každá jednotka je uložena pod pořadovým číslem, se kterým byla vytvořena. Po vytvoření a uložení instance aplikační jednotky je vytvořen v GUI zobrazovací panel se stejným pořadovým číslem. Pomocí pořadového čísla je pak v kontroléru vyhledána patřičná aplikační jednotka a v aplikační jednotce je uložen odkaz na zobrazovací panel. Po registraci zobrazovacího panel již kontrolér ztrácí svou delegační úlohu, protože veškerá komunikace probíhá mezi aplikační jednotkou a zobrazovacím panelem.

Testovací jednotka

Testovací jednotka je jádrem této práce. Jejím hlavním úkolem je získat odpověď od webové služby. Základem jsou data pro testování. Data jsou již ve formě instance tříd, která obsahují potřebné informace. Testovací jednotce jsou předány instance tříd pro nastavení testu a HTTP požadavku. Z dat nastavení testu je použita informace o počtu vláken, který se má použít pro jeden testovací případ. Testovací jednotka podle přiděleného počtu vláken dopředu vytvoří instance požadovaných vláken pomocí třídy *Executors*. Třída *Executors* dokáže dopředu vytvořit fixní počet vláken a spustit je až jsou potřeba. Dopředu vytvořená vlákna šetří výpočetní prostředky, protože při jejich spuštění odpadá režie na jejich vytvoření. Jednotlivá vlákna implementují rozhraní *Callable*, takže před ukončením vlákna je vrácena hodnota. Při spuštění je vlákna je odkaz na toto vlákno předán do instance třídy *Future*. Instance třídy *Future* zajišťuje asynchronní zpracování výsledku vrácené návratové hodnoty vlákna.

Vlákna jsou spouštěny v počítaném cyklu a jsou přidělena do pole obsahující instance tříd *Future*. Po spuštění následuje další cyklus, který blokujícím způsobem čeká na dokončení jednotlivých vláken. Při každém přijetí výsledku od vlákna je tento výsledek testovací jednotkou odeslán do zobrazujícího panelu, kde je výsledek okamžitě zobrazen. Po skončení blokujícího cyklu jsou všechna vlákna ukončena řádným způsobem pomocí instance třídy *Executors*.



Obrázek 6.2: Náhled testovací jednotky.

Proxy jednotka

V této části je uveden stručný popis činnosti proxy monitorovací jednotky. Pro detailní popis je možno využít předchozí práci.

Základní funkce proxy jednotky je poslouchat na zadaném portu, na který přijde HTTP požadavek pro cílovou webovou službu. Jakmile přijde požadavek od klienta webové služby proxy jednotka vytvoří dvě vlákna pro obsluhu příchozího spojení a pro obsluhu odpovědi od webové služby. Vytvořená vlákna jsou instancí stejné třídy pouze pro zpracování odpovědi mají vyměněné deskriptory pro spojení. Spojení je realizováno schránkami (sockets). Vytvořená vlákna mají za úkol oddělit obsah od hlavičky a odeslat požadovaná data do injektoru poruch. Injektor poruch provede analýzu HTTP zprávy a podle zadaných podmínek provedou patřičnou deformaci dat - injekci chyby.

Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je vytvořeno v sadě nástrojů Swing. Na obrázku .. je vidět základní rozložení GUI. Na pravé straně je navigační panel, který zobrazuje obsah kořenového adresáře. Ve spodní části je možno vidět konzolu, kam se zapisují vygenerované zprávy nástroje. Zbytek plochy je hlavní panel.

V GUI je se všemi daty zacházeno jako s instancemi jejich tříd. Tedy v rámci GUI nedochází k žádným datovým operacím. Všechna data jsou odesílána do kontroléru, který je předá k zápisu. Pokud si GUI vyžádá data, kontrolér zajistí jejich přečtení a předá je danému panelu.

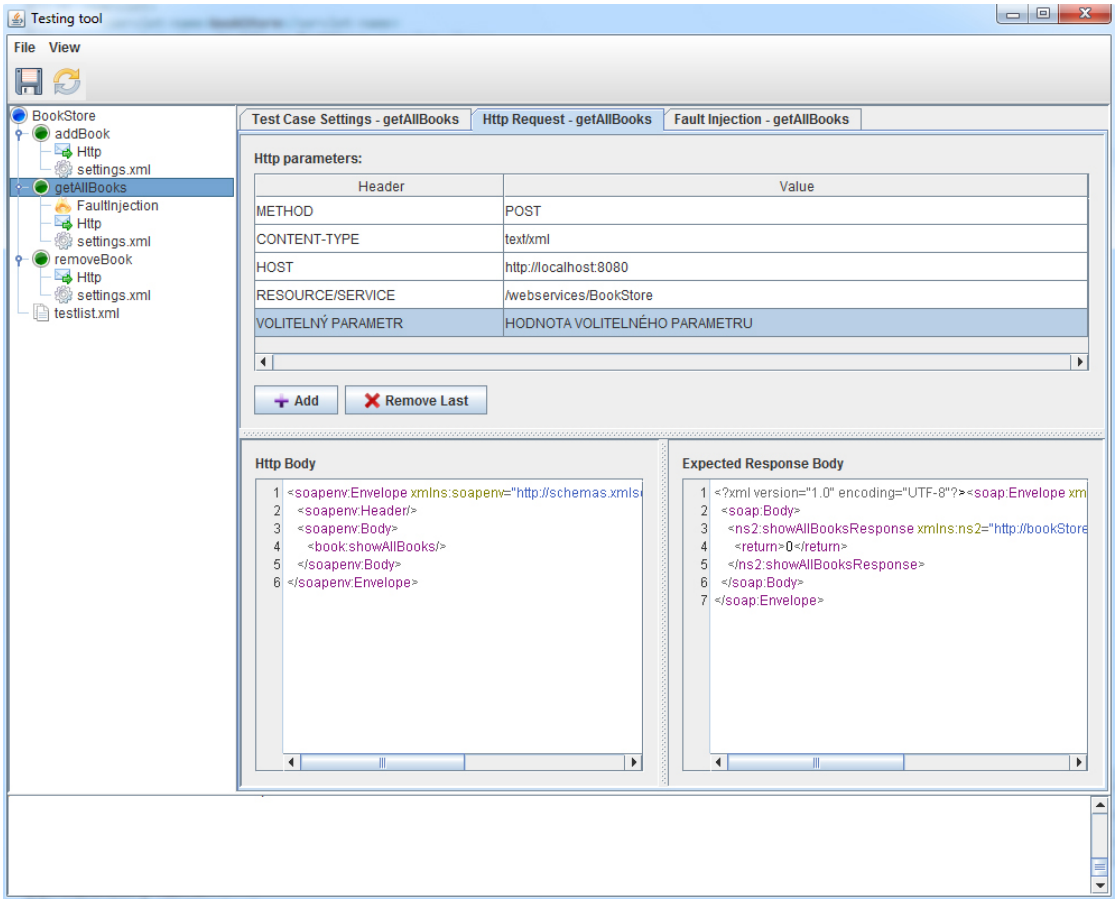
Při zobrazování výsledků testů jsou zobrazovací panel spojen s příslušnou aplikační jednotkou a přijímá výsledky testů přímo bez akce kontroléru. Delegaci komunikace z aplikační jednotky do zobrazovacího panelu zajišťují instance tříd *TestPanelListener* a *ProxyPanelListener* . Tyto instance přijímají data od aplikační jednotky a odesílají je do zobrazovacího panelu. Tento způsob delegace byl zvolen kvůli distribuované architektuře. Metody delegačních instancí jsou volány ze vzdálených aplikačních jednotek, a proto je vhodné tyto komunikační operace oddělit od metod GUI.

Základní funkce GUI jsou k dispozici v položkách menu *File* a *View*. Položka *File* obsahuje příkazy pro vytvoření kompletního schématu dat. Navíc pak obsahuje příkazy pro otevření *Open*), smazání (*Delete*), obnovu navigačního panelu (*Refresh*) a ukončení nástroje (*Exit*). Položka *View* pak nabízí příkazy pro přepínání jednotlivých hlavních panelů GUI.

Navigační panel je tvořen třídou *Navigator*, která umožňuje vyvolání kontextového menu pro intuitivnější použití. Pomocí kontextového menu je možno otevírat jednotlivé hlavní panely podle toho, na kterou položku ve struktuře bylo kliknuto. Při vytváření nových polože je nutno nejdříve vybrat položku nadřazeného prvku viz obrázek. Jedinou výjimkou je vytvoření testovací sady, protože je hierarchicky nejvýše a nemá nadřazený prvek. Pokud se uživatel pokusí vytvořit položku bez vybrání nadřazeného prvku, bude do konzole vypsána patřičná hláška. V případě testovacího případu jej nebude možné vytvořit bez označení testovací sady, do které má být vytvořen.
Grafické uživatelské prostředí má tři hlavní panely:

Panel editoru

Panel editoru je rozdělen na sekce pro editování nastavení testu, HTTP požadavku a injekce chyb. V sekci pro editaci HTTP požadavku je možné vygenerovat SOAP požadavek



Obrázek 6.3: Náhled vytvoření HTTP požadavku.

podle předloženého WSDL dokumentu. Pro generování SOAP požadavku bylo použito experimentální knihovny *soap-builder*. Dále je možno vložit očekávanou odpověď, aby bylo možné vyhodnotit test. Vyhodnocování je možné pro XML vstupy. Při otevření editoru se vždy načtou všechny části a zobrazí se příslušná sekce. Při otevření testu se jméno zobrazí v záhlaví dané sekce.

Testovací panel

Testovací panel je hlavní zobrazovací panel aplikace, odkud se ovládá spouštění testovacích požadavků a jejich vyhodnocení. Panel obsahuje lištu s příkazy pro ovládání aplikace, tabulku pro testovací případy, tabulku pro výsledky testů a panely pro zobrazení obsahu odpovědí. Pomocí kontextového menu v navigačním panelu nebo panelu nástrojů je možné přidávat testovací případy do tabulky. Obsah tabulky je možné uložit jako seznam testů do souboru *testlist.xml*. V druhé tabulce jsou zobrazeny pouze výsledky pro jeden test.

Příjem od aplikační jednotky je realizován metodou *onNewMessageEvent*, kde proběhne uložení dat do ADT [1] *HahsMap*. Pro první testovací případ jsou data zobrazována v reálném čase. Pro ostatní testovací případy jsou data uloženy v tabulce a zobrazovány na vyžádání uživatele.

[1] abstraktní datový typ

Pro distribuovaný test je potřeba připojit další aplikační jednotky a jejich zobrazovací panely k centrálnímu GUI. Tato operace se provede příkazem *Add Remote Unit*. Předpoklad pro připojení další aplikační jednotky je spuštěný RMI server na vzdálené straně. Po zadání údajů do formuláře je přidána další sekce se zobrazovacím panelem. Po načtení dat do všech jednotek je distribuovaný test spuštěn příkazem *Run All Units*. Jednotlivá data jsou předána kontroléru, který je rozešle vzdáleným jednotkám, spustí testování a komunikace již probíhá mezi aplikačními jednotkami a zobrazovacími panely v centrálním GUI.

Na panelu nástrojů je vytvořena ještě funkce pro export konfigurace distribuované ho testu. Tento export uloží údaje o spojení k vzdáleným strojům do souboru. Pak je možné pomocí importu konfigurace připojit vzdálené stroje pomocí jednoho příkazu. Hlavním důvodem exportu konfigurace je však možnost distribuovaného testu z příkazové řádky. Pro připojení vzdálených strojů při spuštění testu z příkazové řádky bude použito právě exportované konfigurace. Pro uložení těchto dat byla použita knihovna *ini4j*. Tato knihovna slouží k ukládání dat ve formátu *ini*. Tento formát byl zvolen kvůli jednoduchosti jeho případné editace uživatelem nebo jeho vytvoření bez použití GUI.

Panel proxy monitoru

V zobrazovacím panelu proxy monitorovací jednotky bylo provedeno jen několik změn. Proxy monitorovací jednotku lze spustit i samostatně bez přítomnosti testovací jednotky. Je nutno označit test v navigačním panelu a spustit jednotku. Samostatně lze spouštět i proxy jednotky na vzdálených strojích. Uplatní se stejný postup, který je popsán v předchozím odstavci pro testovací jednotku. Proxy jednotky nelze spouštět hromadně, musí se spustit každá zvlášť.

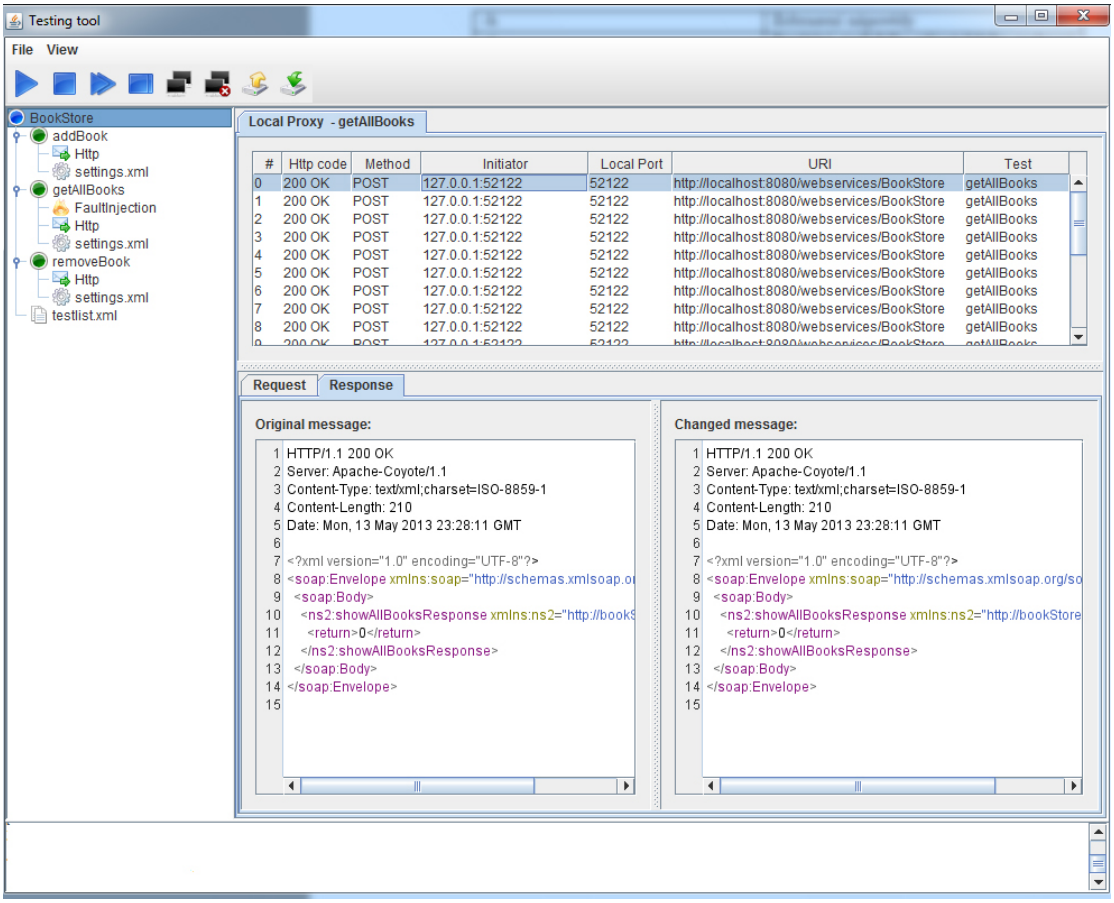
Rozhraní příkazové řádky

Rozhraní příkazové řádky je fakticky hlavním rozhraním celého nástroje. Nástroj je možné spustit pouze z příkazové řádky. Avšak je možné použít skriptovací nástroje jako je Bash v OS Linux nebo dávku .bat v případě OS Windows. Pro rozpoznání jednotlivých příkazů a jejich parametrů byla použita knihovna *Apache Commons CLI*.

Nástroj disponuje těmito příkazy:

-h	Zobrazení nápovědy
-g	Spuštění grafického uživatelského rozhraní
-r	Spuštění RMI serveru
-testlist <soubor> [-config <soubor>]	Spuštění testu, spuštění distribuovaného testu
-r	Spuštění RMI serveru
-proxy <soubor> [-config <soubor>]	Spuštění proxy jednotky, spuštění více proxy jednotek
-stopproxy [-config <soubor>] -proxy <soubor> [-config <soubor>]	Zastavení proxy jednotky, zastavení všech proxy jednotek Spuštění proxy jednotky, spuštění více proxy jednotek

Ovládání z příkazové řádky bylo vytvořeno stejným principem jako GUI. Tedy při zadání příkazu jsou data načtena a předána kontroléru. Tento přístup je možný díky architektuře



Obrázek 6.4: Náhled zobrazení proxy jednotky.

vrstev, kdy uživatelská rozhraní pouze používají možnosti kontroléru a všechny úkony se provádějí přes jednotné rozhraní. Ovládání z příkazové řádky nemá zobrazovací panel, ale má instanci třídy *TextMonitor*, která přijímá data od aplikačních jednotek a odesílá je do nižších vrstev k uložení do souboru na disku.

Pro spuštění distribuovaného testu je třeba přidat nepovinný parametr *-config* se umístěním souboru konfigurace připojení vzdálených strojů. Nejdříve se vytvoří připojení ke vzdáleným strojům, pak se odešlou testovací data a odešle se příkaz ke spuštění testu na všech jednotkách. Pokud se není parametr s umístěním konfigurace přítomen, spustí se test pouze v aplikačních jednotkách, které jsou součástí lokálního programu.

Výstupem testů při ovládání z příkazové řádky jsou soubory v adresáři *Output* každého testovacího případu. Každý výsledek testovacího případu vygeneruje soubor s meta daty a obsah Http odpovědi, která byla přijata.

6.3 Integrace s předchozím řešením

V této části je popsáno jaký postup byl zvolen pro navázání na předchozí práci. Implementace GUI předchozí práce byla těsně vázána na testovací data, která byla součástí GUI a následně se uložila na disk. Při spuštění proxy jednotky se využila data z GUI namísto uložená na disku, kdy tak docházelo k nežádoucímu svázání dat s běžící proxy jednotkou.

Tato implementace byla změněna. Nyní se veškerá vytvořená data ukládají na disk. Z disku je zprostředkovatel na příkaz kontroléru přečte a instance objektu je předána aplikační jednotce. Tento způsob byl zvolen hlavně z důvodů oddělení jednotlivých částí do vrstev a následné implementaci distribuované architektury.

Dále pak v původní implementaci měla proxy jednotka za úkol monitoring příchozích spojení a vytvoření vláken pro obsluhu spojení. Veškerou implementaci pro zobrazování dat měl v sobě integrovaný kontrolér. Tato funkcionalita byla přesunuta do proxy jednotky. Může se zdát, že tato funkcionalita by neměla být součástí proxy jednotky. Toto je však účelné, protože přesunutí registrace zobrazovacích panelů do aplikačních jednotek má cíl oddělení registrace od lokální části aplikace. Takto upravená implementace jednotky může být spuštěná na vzdáleném stroji a zaregistrovat zobrazovací panel pomocí RMI metod.

6.4 Vylepšení proxy jednotky

V této části bude popsána implementace navržených změn v proxy jednotce. Nejdříve bylo nutné upravit zpracování HTTP zprávy. Po vylepšeném přijmu HTTP zprávy je možné detekovat datový formát dané zprávy a podle toho přizpůsobit zpracování dané zprávy. Jakmile můžeme zpracovat různé datové formáty můžeme adekvátně přizpůsobit i analýzu a narušení přijaté zprávy.

Zpracování HTTP zpráv

Pro implementaci komprimačních metod *GZip* a *Deflate* jsou použity knihovny ze standardní sady jazyka Java (JDK, Java Development Kit).

Pro implementaci zpracování zprávy po částech je nutno upravit algoritmus čtení zprávy. Nejprve je třeba detekovat, že se jedná o zpracování částech. Pak se čtou jednotlivé části způsobem, který je popsán v návrhu. Po přečtení celé zprávy je se zprávou naloženo jako s běžně přečtenou zprávou.

Podpora dalších datových reprezentací

Pro zpracování obsahu ve formátu JSON je použita knihovna *GSON*. Podpora formátu v rámci vytvořeného nástroje znamená, že textové výstupy tohoto formátu jsou přehledně naformátovány a je možné použít obsah zpráv ve formátu JSON při injekci poruch.

Přidání dalších typů injekcí chyb a jejich podmínek

Pro implementaci dalších funkcí injektoru poruch bylo nejdříve potřeba vytvořit možnost zadání dat pro tyto funkce. Do modálního okna pro zadání podmínky injekce byla přidána podmínka *HeaderCondition*. Pro implementaci funkce této podmínky je implementováno rozhraní *Condition*. Nová třída je nazvána *HeaderCondition*.

Pro implementaci narušení zprávy ve formátu JSON byla také přidána položka do výběru poruch při zadávání narušení injektorem. Pro implementaci další poruchy je implementováno rozhraní *Fault*. Nová třída je nazvaná *JSONFault*. Aplikace chyby spočívá v převedení textové reprezentace na objekt, který reprezentuje model formátu JSON a bylo možné najít potřebnou proměnnou a změnit její hodnotu.

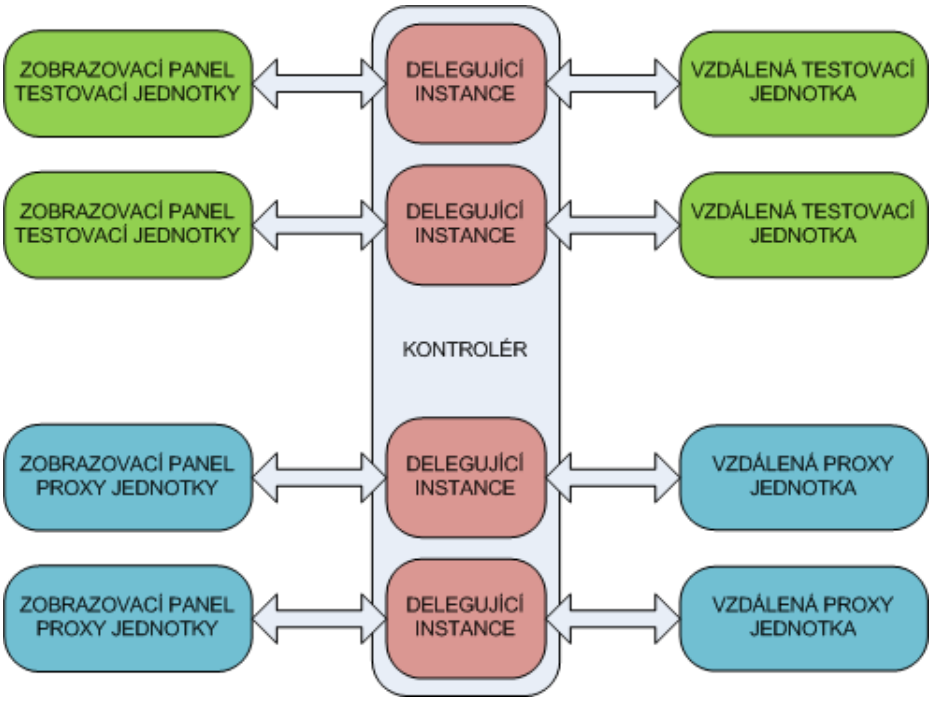
6.5 Možnost integrace s testovacími nástroji

Možnost integrování s testovacími nástroji vychází z použití rozhraní příkazové řádky. Pro integraci s dalšími testovacími nástroji je možné použít nástroj Apache Ant, který disponuje možností spouštět externí programy pomocí příkazové řádky. Při vhodné kombinaci konfigurace připojení a vytvořených seznamů testovacích případů je možné použít vytvořený nástroj s dalšími.

6.6 Implementace distribuované architektury

Tato sekce je zaměřena na detaily implementace distribuované architektury vytvořeného nástroje. Pro implementaci spojení mezi jednotlivými vzdálenými stroji byl vybrána technologie Java RMI. Nynější implementace Java RMI splňuje standard CORBA. Při dodržení rozhraní lze komunikovat se vzdálenou stranou, která je implementována jinou technologii(C++, Python apod.). Java RMI je technologie umožňující volání metod objektů, které jsou umístěny na vzdáleném stroji. Stačí tedy spustit RMI server a registrovat objekt. Požadovaný objekt je zaregistrován v registru. Registr je možné kontaktovat a vyžádat si rozhraní daného objektu.

Na obrázku **??**. je zobrazena implementace distribuované architektury vytvořeného nástroje. Vzdálené stroje jsou implementovány jako RMI server a každý server musí mít vlastní registr. Centrální GUI tedy plni roli klienta. V tomto případě jsou instance vzdálených aplikačních jednotek uchovány v ADT *HashMap*. Jednotlivé instance jsou spárovány se zobrazovacím panelem identifikačním číslem. Při interpretaci příkazu z GUI je získáno identifikační číslo z hlavičky panelu. Identifikační číslo je pak použito jako klíč k vyhledání správné aplikační jednotky. Po získání instance aplikační jednotky je příkaz odeslán.



Obrázek 6.5: Náhled zobrazení proxy jednotky.

Kapitola 7

Průběh testování webových služeb

7.1 Sledované parametry

sdafsadf

7.2 Testování webových služeb s dostupným zdrojovým kódem

asdfadf

7.3 Testování veřejně dostupných služeb

asdfadf

Kapitola 8

Závěr

dsafasdf

8.1 Možnosti dalšího vývoje

sdfsdf

Literatura

[1] *Introduction to Multimedia* [online]. Dave Marshall. Dostupné z WWW: <http://www.qiau.ac.ir/Services/elearning/dehghan/1.pdf>, 2001 [cit. 2013-05-23].

[2] *Apache JMeter* [online]. Apache software foundation. Dostupné z WWW: <http://jmeter.apache.org/>, [cit. 2013-05-23].

[3] *Serialization and De-serialization* [online]. www.codeproject.com. Dostupné z WWW: <http://www.codeproject.com/Articles/33296/Serialization-and-De-serialization>, [cit. 2013-05-23].

[4] *soapUI* [online]. SmartBear Software. Dostupné z WWW: <http://www.soapui.org/>, [cit. 2013-05-23].

[5] *The Open Source Definition* [online]. Open Source Initiative. Dostupné z WWW: <http://www.opensource.org/docs/osd>, [cit. 2013-05-23].

[6] *The Grinder* [online]. Peter Zadrozny, Philip Aston, Ted Osborne. Dostupné z WWW: <http://grinder.sourceforge.net/>, [cit. 2013-09-23].

[7] BERNERS-LEE, T.; Fielding, R.; MASINTER, L.; aj.: *Uniform Resource Identifiers (URI)* [online]. RFC 2396. Dostupné z WWW: <http://www.ietf.org/rfc/rfc2396.txt>, 1998 [cit. 2013-05-23].

[8] BERNERS-LEE, T.; LEACH, P.; MASINTER, L.; aj.: *Hypertext Transfer Protocol – HTTP/1.1* [online]. RFC 2616. Dostupné z WWW: <http://www.ietf.org/rfc/rfc2616.txt>, 1999 [cit. 2013-05-23].

[9] BRAY, T.; PAOLI, J.; MALER, E.; aj.: *Extensible Markup Language (XML) 1.1 (Second Edition)* [online]. Dostupné z WWW: <http://www.w3.org/TR/2006/REC-xml11-20060816/>, 2006 [cit. 2012-05-23].

[10] Brunner, R.; Cohen, F.; Curbera, F.: *Java Web services unleashed.* Sams, 2002, ISBN 9780672323638.

[11] ERL, T.: *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005, ISBN 0131858580, 792 s.

[12] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures.* Dizertační práce, University of California, Irvine, 2000.

[13] Graham, D.; Veenendaal, E. V.; Evans, I.; aj.: *Foundations of Software Testing: ISTQB Certification.* Intl Thomson Business Pr, 2008, ISBN 9781844803552, 9781844809899.

[14] HANSEN, M. D.: *SOA Using Java Web Services.* Prentice Hall, 2007, ISBN 0130449687, 574 s.

[15] JENDROCK, E.; EVANS, I.: *The Java EE 5 Tutorial* [online]. Dostupné z WWW: <http://docs.oracle.com/javaee/5/tutorial/doc/>, 2011 [cit. 2013-05-23].

[16] Kalin, M.: *Java Web Services: Up and Running.* O'Reilly Media, Incorporated, 2009, ISBN 9780596521127.

[17] MITRA, N.; LAFON, Y.: *SOAP Version 1.2 Part 0: Primer (Second Edition)* [online]. Dostupné z WWW: <http://www.w3.org/TR/soap/>, 2007 [cit. 2013-05-23].

[18] Myers, G. J.; Sandler, C.; Badgett, T.; aj.: *The Art of Software Testing.* Wiley, 2004, ISBN 0471469122.

[19] RAGGETT, D.; HORSE, A. L.; JACOBS, I.: *HTML 4.01 Specification* [online]. Dostupné z WWW:<http://www.w3.org/TR/html4/>, 1999 [cit. 2013-05-23].

Příloha A

Obsah CD