DATAKON 2011

Tutoriály

Editoři

Jaroslav Zendulka
Marek Rychlý

Mikulov, Hotel Eliška
Česká republika
15. - 18. října 2011
http://www.datakon.cz

DATAKON® je prestižní česká a slovenská konference s mezinárodní účastí zaměřená na teoretické a technické základy, nejlepší postupy a vývojové trendy v oblasti využití informačních technologií při budování informačních systémů včetně výsledků jejich aplikace v praxi.

DATAKON® představuje ideální platformu pro výměnu zkušeností mezi českými i zahraničními odborníky z řad dodavatelů informačních technologií, jejich zákazníků a akademického světa.

DATAKON® oslovuje zkušené odborníky i nejlepší studenty.

© Autoři článků, 2011

Vydává: Vysoké učení technické v Brně, 2011
Tisk: SDRUŽENÍ MAC, spol. s r.o., U Plynárny 85, Praha 10

ISBN 978-80-214-4330-3

DATAKON 2011

Tutorials

Edited by

Jaroslav Zendulka
Marek Rychlý

Mikulov, Hotel Eliška
Czech Republic
October 15-18, 2011
http://www.datakon.cz

DATAKON® is a high-profile traditional conference focused on theoretical and technical background, best practices and development trends in deployment of information technology for information systems development including application results of described approaches in industry practice.

DATAKON® serves as an ideal platform for experience exchange among experts of information technology products and services suppliers, their customers and the academic community both Czech, Slovak and also foreign.

DATAKON® brings together researchers, professionals, and students.

© The authors of contributions, 2011

Published by Brno University of Technology, 2011
Printed by SDRUŽENÍ MAC, spol. s r.o., U Plynárny 85, Praha 10

ISBN 978-80-214-4330-3

Předmluva

DATAKON® je prestižní česká a slovenská konference s mezinárodní účastí zaměřená na teoretické a technické základy, nejlepší postupy a vývojové trendy v oblasti využití informačních technologií při budování informačních systémů včetně výsledků jejich aplikace v praxi. Nosnými tématy ročníku 2011 jsou

- Architektury informačních systémů
- Cloud computing
- Dobývání znalostí
- Informační bezpečnost
- Integrace datových zdrojů
- Kvalita dat a zpracování neúplné informace
- Management znalostí a znalostní technologie
- Modelování procesů a služeb
- Multimediální, časová, časově prostorová a geografická data
- Nové trendy a moderní databázové technologie
- Servisně orientované architektury
- Sociální sítě na internetu
- Vyhledávání na webu
- XML technologie
- Zpracování rozsáhlých souborů dat
- Proudy dat
- Řídicí systémy v reálném čase
- Datové registry

Konference DATAKON 2011 se konala 15. až 18. října 2011 v Mikulově, v hotelu Eliška. Výběr příspěvků zajišťoval programový výbor pro rok 2011. Všechny příspěvky byly posuzovány třemi nezávislými recenzenty. Z celkového počtu 20 podaných standardních příspěvků a dvou případových studií vybral programový výbor 9 standardních příspěvků a obě případové studie. Čtyři příspěvky byly doporučeny k přijetí jako postery. Na základě výzvy pro podání pozdních posterů byl přijat ještě jeden pozdní poster s redukovaným rozsahem.

Tato kniha tutoriálů obsahuje čtyři tutoriály prezentované na konferenci. Jde o původní texty zaměřené na škálovatelná řešení pro zpracování velkého objemu dat, servisně orientovanou architekturu, geolokaci a multiagentní systémy s jasně vymezenou problematikou. Tutoriály byly recenzovány členy programového výboru.

Samostatnou knihou je kniha příspěvků, která obsahuje texty čtyř zvaných přednášek, devíti přijatých standardních příspěvků, dvou přijatých případových studií, dvou případových studií partnerů konference (IBM Česká republika spol. s r.o. a KOMIX s.r.o.), čtyř posterů a jednoho pozdního posteru.

Závěrem je třeba poděkovat všem, kteří se zasloužili o vznik tohoto ročníku konference DATAKON a této publikace. V prvé řadě chci poděkovat autorům zvaných přednášek, tutoriálů, standardních příspěvků, případových studií a posterů, za úsilí, které vynaložili při jejich přípravě. Rovněž bych chtěl poděkovat členům programového výboru za jejich nápady a práci při přípravě programu konference. Dále chci poděkovat partnerům konference za jejich podporu při přípravě konference. Velký dík patří také organizačnímu výboru konference, zejména Anně Kotěšovcové, Marku Rychlému a Petru Šalounovi, za přípravu a zajištění průběhu konference DATAKON 2011.

V Brně, září 2011
Jaroslav Zendulka
předseda programového výboru

Preface

DATAKON® is a high-profile traditional conference focused on theoretical and technical background, best practices and development trends in deployment of information technology for information systems development including application results of described approaches in industry practice. This year the following main topics have been chosen:

- Information systems architecture
- Cloud computing
- Data mining
- Information security
- Data sources integration
- Data quality and incomplete information processing
- Knowledge management and technologies
- Process and service modeling
- Multimedia, time, time space and geographic data
- New trends and modern database technologies
- Service oriented architecture
- Social networks on the Internet
- Searching on the Web
- XML technology
- Very large data sets processing
- Data streams
- Real-time systems
- Data registers

The structure of the book corresponds to the DATAKON 2011 conference program. DATAKON 2011 will be held on 15th–18th October 2011, in Mikulov, Czech Republic. The Program Committee carried out the selection of papers. All papers were reviewed in advance by three reviewers. All papers were judged only on their own merits, independent of other submissions. Finally, nine standard papers and two case studies of the total number of 20 submitted standard papers and two case studies were selected for publication. Four submissions were accepted as posters. Based on the call for late posters, one more poster with a reduced size was accepted.

This book comprises four tutorials focused on scalable solutions for large data volumes processing, service-oriented architectures, geolocation and multiagent systems.

Four invited lectures, nine accepted standard papers, two case studies, two case studies of the industrial partners of the conference (IBM Czech Republic and KOMIX companies), four posters and one late poster presented on the conference are published in a separate book.

I would like to express my acknowledgements to all people who prepared the DATAKON 2011 conference. I would like to thank the authors of invited lectures, tutorials, standard papers, case studies and posters papers submitted to DATAKON 2011 for their efforts to prepare them. I would also like to thank all the Program Committee members for their excellent work during discussions on the topics of the conference, reviewing process and the conference program. I also wish to acknowledge to the conference partners for their support. My special thanks go to the members of the organization committee, namely to Anna Kotěšovcová, Marek Rychlý and Petr Šaloun. Without their assistance and excellent work the DATAKON 2011 conference could not have been possible.

Brno, September 2011
Jaroslav Zendulka
Program Committee Chair

Organizace konference (Conference Organization)

Řídící výbor (Steering Committee)

Předseda (Chair): Jaroslav Pokorný, MFF UK Praha

Členové (Members):
Mária Bieliková, FIIT STU Bratislava
Ján Genči, FEI TU Košice
Jiří Gregor, GALEOS a.s.
Petr Hujňák, Per Partes Consulting Praha
Dušan Chlapek, FIS VŠE Praha
Karel Richta, MFF UK Praha
Jan Staudek, FI MU Brno
Petr Šaloun, FEI VŠB-TU Ostrava
Jaroslav Zendulka, FIT VUT Brno

Programový výbor (Program Committee)

Předseda (Chair): Jaroslav Zendulka, VUT Brno

Členové (Members):
Maria Bieliková, STU Bratislava
Miroslav Benešovský, NESS Europe Brno
Dušan Chlapek, VŠE Praha
Marie Duží, VŠB-TU Ostrava
Ján Genči, TU Košice
Jiří Gregor, GALEOS a.s.
Ivan Halaška, ČVUT Praha
Petr Hanáček, VUT v Brno
Tomáš Hruška, VUT Brno
Petr Hujňák, Per Partes Consulting Praha
Karel Ježek, ZČU Plzeň
Štefan Kovalík, ŽU Žilina
Jaroslav Král, UK Praha
Pavel Král, ZČU Plzeň
Petr Kučera, Komix s.r.o.
Aleš Limpouch, TopoL Software, s.r.o.
Karol Matiaško, ŽU Žilina
Peter Mikulecký, UHK Hradec Králové
Martin Molhanec, ČVUT Praha
Jaroslav Pokorný, UK Praha
Lubomír Popelínský, MU Brno
Karel Richta, UK Praha

Václav Řepa, VŠE Praha
Vojtěch Svátek, VŠE Praha
Petr Šaloun, VŠB-TU Ostrava
Petr Tůma, UK Praha
Michal Valenta, ČVUT Praha
Tomáš Vlk, TRIL s.r.o. Kladno
Peter Vojtáš, UK Praha
Jaroslav Zelený, IBM ČR Praha

Organizační výbor (Organization Committee)

Anna Kotěšovcová, organizační agentura CONFORG, s.r.o.
Marek Rychlý, VUT Brno
Petr Šaloun, VŠB-TU Ostrava
Jaroslav Zendulka, VUT Brno
Michal Žemlička, UK Praha

DATAKON 2011 organizují (DATAKON 2011 is organized by)

Matematicko-fyzikální fakulta, UK Praha
Česká společnost pro systémovou integraci
Slovenská informatická společnost
Přírodovědecká fakulta, Ostravská univerzita v Ostravě
Fakulta informačních technologií, VUT Brno

Partneři konference DATAKON 2011 (DATAKON 2011 Partners)

Profinit, s.r.o.

Vema, a.s.

IBM Česká republika, spol. s r.o.

KOMIX s.r.o.

Obsah

Dostupné škálovateľné riešenia pre spracovanie veľkého objemu dát a dátové sklady
Martin Šeleng, Michal Laclavík, Štefan Dlugolinský, Ladislav Hluchý 1
Geolokace a geolokační techniky
Jaroslav Srp 23
Servisně Orientované Architektury
Martin Nečaský, David Kusák, Karel Richta 49
Multiagentní systémy
František Zbořil 75

Rejstřík autorů **101**

Dostupné škálovateľné riešenia pre spracovanie veľkého objemu dát a dátové sklady

Martin ŠELENG[1], Michal LACLAVÍK[1], Štefan DLUGOLINSKÝ[1], Ladislav HLUCHÝ[1]

[1]*Ústav informatiky, Slovenská Akadémia Vied*
Dúbravská cesta 9, 845 07 Bratislava
{martin.seleng, michal.laclavik, stefan.dlugolinsky}@savba.sk

Abstrakt. V súčasnosti je čoraz viac druhov informácií dostupných v digitálnej forme. S nástupom Web 2.0 aplikácií, ako aj elektronizácie bankových alebo cestovných transakcií, je generované množstvo dát. Vo väčšine aplikácií potrebujeme tieto informácie a dáta ukladať, spracovávať a dopytovať sa na ne. Firmy ako Google, Facebook, Amazon, Yahoo! alebo Twitter museli hľadať nové inovatívne riešenia, ako si poradiť so škálovaním svojich systémov pri zvyšovaní počtu používateľov, objemu dát, ale aj pri spracovávaní, ukladaní a dopytovaní nad týmito dátami. Tutoriál je zameraný na predstavenie takýchto škálovateľných open-source riešení. Tieto riešenia vychádzajú z architektúr spomínaných firiem (Google – MapReduce, Amazon – Dynamo), alebo vývoj týchto riešení tieto spoločnosti podporujú (Facebook – Apache Casandra, Apache HBase, Yahoo - Hadoop). Všetky uvedené open-source riešenia používajú horizontálne škálovanie. Systém je teda možné rozširovať zapojením ďalších uzlov do klastra, pričom je možné použiť bežne dostupné počítače (tzv. commodity PCs) bez špeciálneho hardvéru. Tieto architektúry musia zvládať aj výpadky uzlov (fault-tolerance) a musia byť tvorené tak, aby vývojár aplikácie nemusel dôkladne poznať architektúru systému.

Klúčové slová: škálovateľnosť, MapReduce, NoSQL.

1 Spracovávanie veľkých objemov dát

Ukladanie a spracovávanie veľkých objemov dát bolo problémom od počiatkov vzniku Internetu. Prvé riešenia boli založené buď na distribuovanom princípe viacerých počítačov s kapacitne malými diskami alebo na obrovských úložných miestach (diskových poliach). V prípade prvého prístupu museli byť vyvinuté centrálne služby (na logickej úrovni) na správu a prístup k uloženým dátam. V prípade druhého prístupu museli byť vyvinuté služby na hardvérovej úrovni (na úrovni pevných diskov, diskových polí). Ďalším problémom, s ktorým sa museli vývojári týchto systémov vysporiadať, bola strata údajov, či nefunkčnosť niektorého z uzlov (hardvérová chyba počítača, resp. problém s konektivitou) v prípade prvého prístupu, resp. nefunkčnosťou časti diskového poľa, disku v prípade druhého prístupu. Klasickým riešením tohto problému je vytváranie replík dát uložených v týchto systémoch. V prípade prvého prístupu nastavením počtu požadovaných replík jednotlivých logických častí dát (súborový systém ako taký existuje len na logickej úrovni, distribuovaný systém sa nestará o to, ako sú dáta fyzicky uložené). V druhom prístupe sa používa technológia RAID (Redundant Array of Independent Disks), kde existuje viacero spôsobov, ako predísť strate uložených dát (viacej informácií je možné nájsť v [9]). V našom

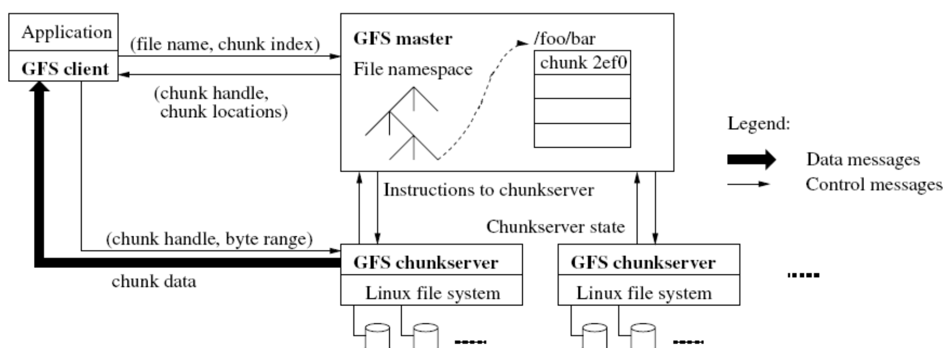
príspevku sa budeme venovať prvému spôsobu, ktorý je z hľadiska prístupu viac decentralizovaný, aj keď v niektorých implementáciách táto decentralizácie nie je úplná.

2 Systémy na ukladanie a správu veľkého množstva dát

V tejto kapitole sa budeme venovať distribuovaným súborovým systémom, systémom na ukladanie štruktúrovaných dát („distribuované databázy“) a distribuovaným dátovým skladom. Všetky tieto systémy používajú horizontálne škálovanie, na rozdiel od klasických RDBMS databáz spĺňajúcich tradičné ACID[1] (atomicity, consistency, isolation, durability) vlastnosti, kde sa používa vertikálne škálovanie.

2.1 Existujúce distribuované „súborové“ systémy na ukladanie a správu veľkého množstva dát

Jedným z prvých, kto sa musel zaoberať vo väčšom meradle decentralizáciou ukladania dát, bola spoločnosť Google, ktorá predstavila svoj distribuovaný súborový systém GFS (Google File System) v tomto článku [4]. GFS je založený na veľkom množstve bežne dostupných PC, ktoré dosť často zlyhávajú, a preto musí byť neustále monitorovaný. Tento systém podporuje malé aj veľké súbory, avšak je optimalizovaný, hlavne pre súbory rádovo v MB, resp. GB. Súbory v GFS sú popisované veľmi jednoduchým spôsobom: meno súboru je kľúč a obsah súboru je jeho hodnota (Obr. 1).

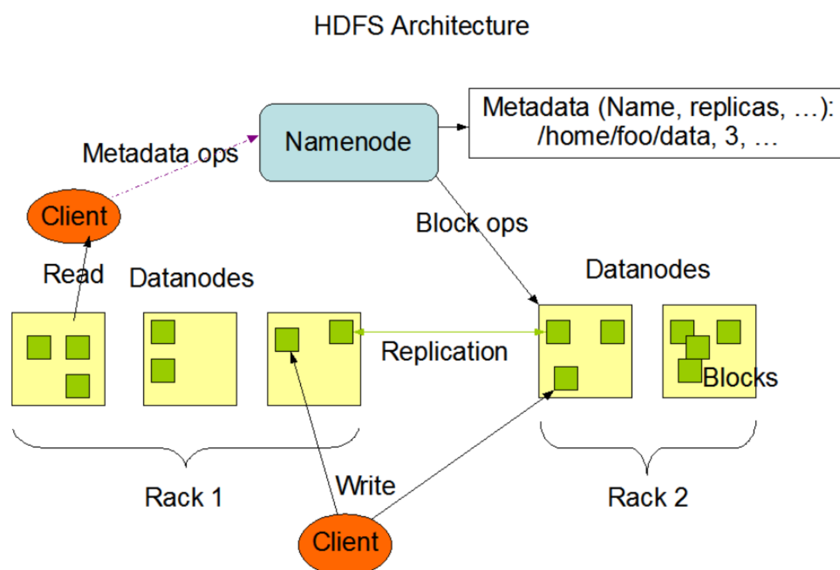


Obr. 1 Architektúra, dátový a riadiaci tok informácií v GFS

Systém ako taký neumožňuje pridanie obsahu k už existujúcemu súboru (nie je známy stav k dnešnému dňu, avšak posledná implementácia Apache Hadoop to už umožňuje). Jediný spôsob ako to dosiahnuť, je zmazať predchádzajúci súbor (neodporúčané kvôli granularite voľného priestoru na úložných zariadeniach) a prepísať ho novým súborom obsahujúcim už dané zmeny. GFS bol navrhnutý tak, aby údaje do neho nahraté už neboli nikdy vymazané (toto je možné dosiahnuť aj pridaním časových značiek k súborom), pretože podľa Google je jednoduchšie pridať nové úložné zariadenia, ako mať súbory porozhadzované po diskoch (využívajú sekvenčný zápis).

[1] http://en.wikipedia.org/wiki/ACID

Podľa vzoru GFS (konkrétne paradigmy MapReduce [2]) Doug Cutting (autor populárneho indexovacieho nástroja Apache Lucene[2] a spoluautor sťahovača Apache Nutch[3]) vytvoril projekt Apache Hadoop[4] na paralelné spracovanie dát a obidva vyššie spomínané projekty prispôsobil tak, aby podporovali ukladanie a prácu s HDFS (Hadoop Distributed File System) (Obr. 2).



Obr. 2 Architektúra, dátový a riadiaci tok informácií v HDFS

Ďalšou spoločnosťou, ktorá riešila problém ukladania veľkého množstva dát, je spoločnosť Amazon. Spoločnosť Amazon vytvorila vlastný distribuovaný systém S3 (Simple Storage Service). Tento systém je založený výlučne na REST službách (HTTP) a protokole SOAP. Dáta sú uložené v regiónoch najbližších k používateľovi a bez jeho súhlasu nie sú migrované do iných regiónov (ani kvôli zabezpečeniu replík).

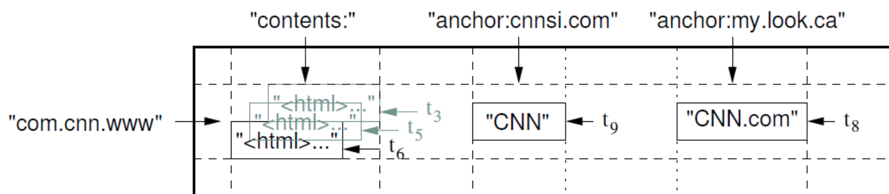
Spoločnosť Yahoo! sa rozhodla nevyvíjať vlastné riešenie na vyhľadávanie nad distribuovanou architektúrou, ale adaptovať Apache Hadoop. Svoju verziu Hadoop-u spoločnosť Yahoo! naďalej vyvíjala, ale keďže nie je klasickým prispievateľom do open-source projektov, oznámila ukončenie vývoja a naďalej bude používať verziu vyvíjanú pod spoločnosťou Apache. Na druhej strane spoločnosť Yahoo! predstavila rámec (framework) na spracovanie prúdov (streams) dát S4[5], kde sú hlavnými vývojármi jej zamestnanci. Spoločnosť Yahoo! tiež vyvinula jazyk Pig[6], ktorý slúži na dopytovanie sa nad dátami s určitou štruktúrou, ako sú napríklad záznamy webového servera a pod. Tento jazyk určitým spôsobom pripomína SQL jazyk, pričom dovoľuje používateľom implementovať vlastné funkcie UDF (User Defined Functions) pre konkrétne štruktúry dát.

---

[2] http://lucene.apache.org
[3] http://nutch.apache.org
[4] http://hadoop.apache.org
[5] http://s4.io
[6] http://hadoop.apache.org/pig/

2.2 Existujúce systémy na ukladanie a správu veľkého množstva štruktúrovaných dát

V predchádzajúcej časti sme sa venovali najmä distribuovaným architektúram na ukladanie dát (súborov) aj neštruktúrovaného typu. V tejto časti sa zameriame aj na distribuované architektúry pre ukladanie dát databázového (štruktúrovaného) typu. Opäť prvou spoločnosťou, ktorá musela riešiť problém ukladania veľkých objemov kvázi štruktúrovaných dát, ako sú napr. ukladanie obsahu stránok, prepojení medzi stránkami, textov odkazov (anchor texts), atď. bola spoločnosť Google[7]. Google publikoval článok, kde predstavil svoju ideu databázy pre veľké dáta a nazval ho BigTable [6]. Týmto projektom naštartoval vlnu databáz, ktoré sa dnes nazývajú spoločným menom: NoSQL databázy (väčšinou založené na princípe kľúč/hodnota, nie však vždy). Ďalšou podstatnou zmenou bolo, že BigTable nie je riadkovo, ale stĺpcovo orientovaná databáza. Keďže dáta sú fyzicky uložené v GFS, riadky sú použité ako kľúče a obsahy stĺpcov ako hodnoty. Na Obr. 3 je možné vidieť uložené dáta v databáze BigTable.



Obr. 3 Ukážka uloženia obsahu stránok a anchor textov dát v BigTable

Spoločnosť Amazon[8], ako jeden z najväčších internetových predajcov, musel takisto riešiť ukladanie veľkých štruktúrovaných dát, pričom takisto išiel cestou ukladania dát ako kľúč/hodnota (v podstate tiež inšpirovaný článkom o BigTable). Navrhli a implementovali distribuovanú databázu Dynamo [3], v ktorej majú uložené dáta o svojich produktoch a každý dopyt na stránke Amazon-u je vykonávaný nad touto databázou. Fyzicky sú dáta priamo uložené v súborovom systéme jednotlivých uzlov.

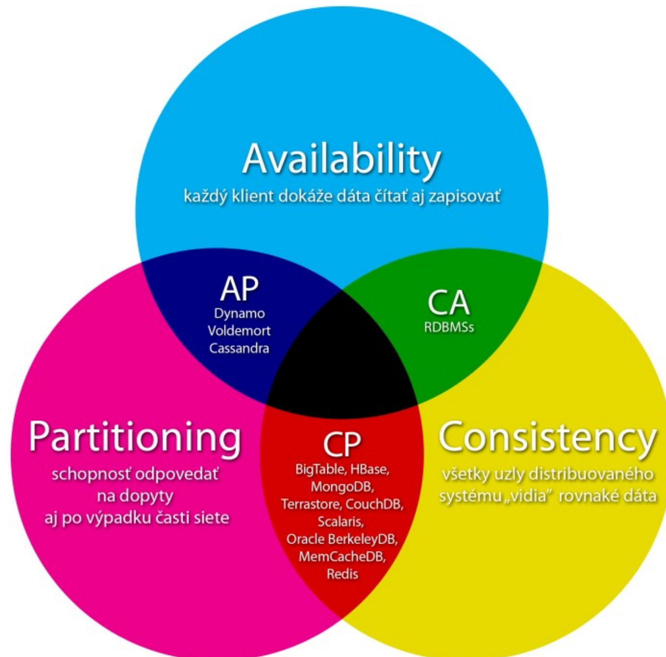
Tieto dva „hlavné vzory" NoSQL databáz sa odlišujú v prístupe k správe, ukladaniu a čítaniu dát v nich uložených. Na základe tzv. CAP (Consistency, Availability, Partition torelance) resp. Brewerovej teorémy [1], neexistuje distribuovaný systém, ktorý by spĺňal všetky 3 podmienky naraz na 100% (Obr. 4). V tejto časti si vysvetlíme čo sa skrýva pod skratkou CAP, tak ako to chápal Brewer a ako to je myslené pri tzv. NoSQL databázach:

- Consistency znamená, že v určitom čase všetky uzly distribuovaného systému „vidia" rovnaké dáta.
- Availability znamená, že každý klient po svojom dopyte dostane informáciu o tom, či operácia bola úspešná alebo nie (niekedy sa používa aj vysvetlenie, že každý klient vie vždy čítať aj zapisovať).

[7] Spoločnosť Google nebola z časového hľadiska v skutočnosti prvá (viď nižšie Berkley DB), avšak bola prvým propagátorom a masívnym priekopníkom v tejto oblasti

[8] Tak ako v prípade spoločnosti Google, nie je úplne isté, že spoločnosť Amazon bola v tomto prvá, avšak bola jednou zo začínajúcich spoločností tvoriacich tzv. „NoSQL movement" v oblasti vertikálne škálovateľných NoSQL databáz

- Partition tolerance je vlastnosť, ktorá hovorí o tom, že ak časť siete vypadne (preruší sa spojenie alebo dôjde k strate prenosov medzi uzlami na minimálne 2 dizjunktné množiny uzlov) systém bude stále schopný odpovedať na dopyty.



Obr. 4 CAP/Brewerova teoréma

Dôkaz toho, že je možné splniť maximálne 2 podmienky naraz (na 100%) je možné nájsť v [5]. V tom istom článku je možné nájsť aj návrh distribuovaného systému, ktorý bude na 100% spĺňať podmienky A a P a čiastočne je splnená aj podmienka C (systém konzistentný v čase), je uvedený aj dôkaz, že takýto systém môže existovať. V prípade BigTable je dôraz kladený na konzistenciu dát a na vrátenie odpovede aj v prípade výpadku siete/uzlov. V Dyname je väčší dôraz kladený na to, že každý klient dostane odpoveď na svoj dopyt (či už bol dopyt úspešný alebo nie) a na vrátenie odpovede aj v prípade výpadku siete/uzlov. Dynamo takisto rieši konzistenciu (nie však na 100%), pretože spoločnosť Amazon je zameraná na predaj produktov a nemôže sľúbiť určitý produkt viacerým záujemcom naraz, ak má na sklade posledný kus.

2.3 Projekty inšpirované priamo projektom BigTable

Podľa vzoru BigTable vzniklo mnoho ďalších projektov (komerčných aj open-source), ktoré si opíšeme v nasledujúcej podkapitole. V prípade projektu BigTable je dôraz kladený na:

- **C**onsistency
- a **P**artition tolerance.

2.3.1 Stĺpcovo orientované distribuované databázy

Pri stĺpcovo orientovaných databázach sú tabuľky fyzicky uložené po stĺpcoch, pričom stĺpce môžu obsahovať ľubovoľný obsah. Jednou z prvých dodnes vyvíjaných stĺpcových databáz je Apache HBase[9], ktorý má presne tie isté vlastnosti, ako boli prezentované v článku o BigTable [6]. Apache HBase je stĺpcovo orientovaná databáza 100% zaručujúca konzistenciu (C), tolerantná k prerušeniu komunikácie (P) a teda prináša so sebou problém informovania klienta či jeho požiadavka bola splnená alebo nie (A). Príčinou problému je, že v základnej konfigurácii používa jedného mastra a teda má tzv. SPOF (Single Point Of Failure). Neskôr bola pridaná podpora výberu náhradného servera v prípade pádu práve používaného servera (toto bolo dosiahnuté pomocou rámca Apache Zookeeper[10]) a teda sa nejakým spôsobom podarilo naplniť aj tretiu podmienku teorémy CAP. Dáta uložené v systéme Apache HBase sú uložené v HDFS podobne, ako je tomu v prípade BigTable od spoločnosti Google, kde sú dáta uložené v GFS (Tab. 1). Táto tabuľka je síce veľmi prázdna, ale treba si uvedomiť, že v prípade fyzického uloženia v HDFS systéme prázdne miesta nie sú uložené, pretože tabuľky v systéme HBase sú fyzicky uložené po stĺpcoch, ako to ukazuje Tab. 2.

Row Key	**Time Stamp**	**Column** *"contents:"*	**Column** *"anchor:"*		**Column** *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

Tab. 1 Ukážka tabuľky uloženej v systéme HBase – konceptuálny pohľad

Row Key	**Time Stamp**	**Column** *"contents:"*
"com.cnn.www"	t6	"<html>..."
	t5	"<html>..."
	t3	"<html>..."

Row Key	**Time Stamp**	**Column** *"anchor:"*	
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"

Row Key	**Time Stamp**	**Column** *"mime:"*
"com.cnn.www"	t6	"text/html"

Tab. 2 Ukážka tabuľky uloženej v systéme HBase – fyzické uloženie v HDFS

Ďalším, veľmi úspešným projektom inšpirovaným projektom BigTable, je projekt Hypertable[11]. Hypertable je takisto stĺpcovo orientovaná databáza. Je navrhnutá tak, aby bežala nad nejakým distribuovaným súborovým systémom, akým je napr. HDFS, ale poskytuje aj možnosť behu nad jedným strojom (standalone mód). Hypertable je integrovateľná aj s distribuovaným dátovým skladom Apache Hive[12], ktorému sa budeme venovať neskôr.

[9] http://hbase.apache.org
[10] http://zookeeper.apache.org
[11] http://www.hypertable.org
[12] http://hive.apache.org

2.3.2 Dokumentovo orientované distribuované databázy

Pri dokumentovo orientovaných databázach je do tabuliek priamo ukladaný objekt dokumentu. Použitie slova dokument je však trochu zavádzajúce. Na nasledujúcich príkladoch si môžeme ukázať, čo je myslené pod pojmom dokument v dokumentovo orientovaných databázach:

```
1. Meno="Martin", Priezvisko="Seleng", Vek=35, Adresa="Kvietkova ulica 5"
2. Meno="Ferko", Priezvisko="Mrkvicka", Vek=48, Adresa="Konvalinkova
   ulica 2", Deti=[{Meno:"Anna", Vek:12}, {Meno:"Jozef", Vek:7}]
```

Keď sa pozrieme na oba dokumenty, tak vidíme, že sú trochu podobné, ale druhý má v sebe viac informácií. Je to podobné ako pri stĺpcových databázach, kde sa stĺpce môžu ľubovoľne deliť na viac stĺpcov, avšak tieto položky sú uložené po riadkoch a nie po stĺpcoch. Dokumentovo orientované databázy nemajú pevnú štruktúru/schému a sú relatívne používateľsky príjemné. Hlavné techniky na čítanie a zapisovanie do dokumentovo orientovaných databáz sú HTTP služby používajúce najmä štandardy JSON alebo XML. Je potrebné si uvedomiť, že všetky XML databázy sú vlastne dokumentovo orientované databázy. Uloženie je podobné ako v prípade stĺpcovo orientovaných databáz (prázdne bunky sa neukladajú). Výhodou je, že sú relatívne ľahko pochopiteľné aj pre bežných používateľov, ktorí si môžu navrhovať vlastné databázy a jediné čo potrebujú je rozhranie, cez ktoré budú čítať a zapisovať. Väčšina dokumentovo orientovaných databáz poskytuje RESTful[13] API na prístup k objektom.

Medzi najznámejšie distribuované dokumentové databázy patrí MongoDB[14], ktorá je implementovaná v C++. Táto distribuovaná databáza takisto umožňuje spúšťať Map/Reduce úlohy nad svojimi dátami. Pre túto databázu existujú klienti a API v mnohých programovacích jazykoch, ako sú napr.: Java, Perl, PHP, Python, atď.

Ďalšou implementáciou je projekt Terrastore[15], ktorý poskytuje prístup cez HTTP. Momentálne existujú aj klientske aplikácie vo viacerých jazykoch ako je napr.: JAVA, PHP, Python, Scala, atď. Projekt Terrastore je integrovateľný s projektom Elasticsearch[16] určeným na vyhľadávanie a indexovanie (Elasticsearch je postavený nad projektom Apache Lucene[17]). Spolu s projektom Terrastore poskytuje Elasticsearch veľký diskový priestor, a indexovanie takmer v reálnom čase. Dokument je indexovaný okamžite, ako je pridaný do databázy.

2.3.3 Distribuované databázy typu kľúč/hodnota

Kľúč/hodnota (Key/Value) databázy, majú jednoduchú štruktúru, pretože obsahujú vždy len dve položky:

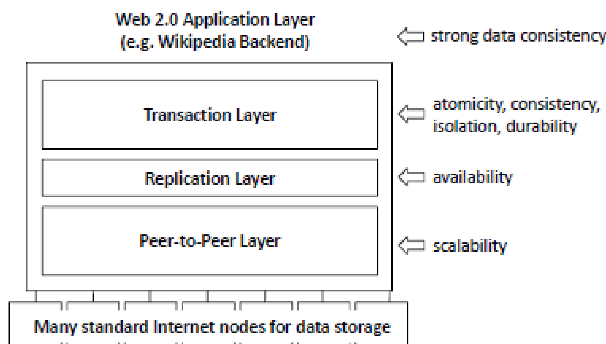
- **kľúč**, ktorý je jedinečný,
- a **hodnotu**, ktorá je priradená k tomuto kľúču a môže byť akákoľvek.

Jednou z distribuovaných databáz tohto typu je projekt Scalaris[18], ktorý bol vyvíjaný na Zuse Institute v Berlíne a spoločnosťou onScale solutions GmbH. Tento projekt vyriešil

---

[13] http://en.wikipedia.org/wiki/Representational_State_Transfer
[14] http://www.mongodb.org
[15] http://code.google.com/p/terrastore
[16] http://www.elasticsearch.org
[17] http://lucene.apache.org
[18] http://code.google.com/p/scalaris

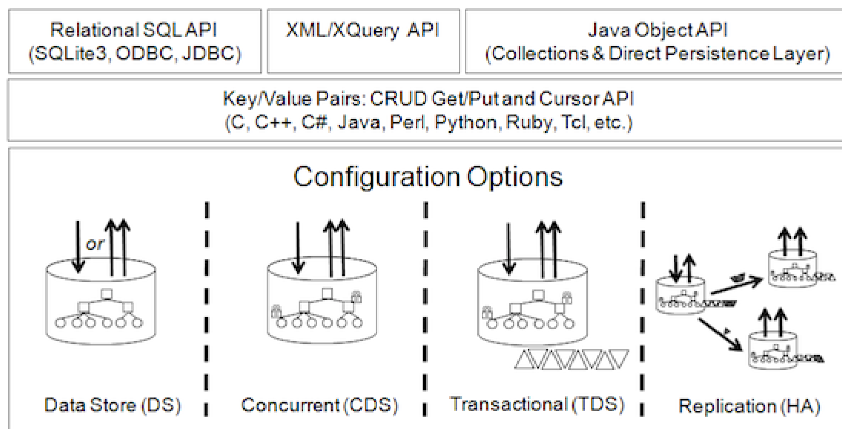
distribuovanosť a replikáciu pomocou technológie P2P (používa Chord[19] protokol). Projekt je celý napísaný v jazyku Erlang. Architektúra projektu Scalaris je založená na 3 vrstvách + 1 aplikačnej vrstve (demonštrácia bola prevedená na klone Wikipédie). Na Obr. 5 je možné vidieť vrstvy systému Scalaris s aplikačnou vrstvou[20].



Obr. 5 Architektúra systému Scalaris

Ďalšou databázou, založenou na princípe kľúč/hodnota, je Oracle Berkeley DB[21]. Táto databáza, ako jedna z mála, nebola inšpirovaná projektom BigTable. Existuje už 15 rokov, ale nie všetky vlastnosti boli implementované v tomto období. Skladá sa z troch osobitných produktov:

- Berkeley DB,
- Berkeley DB Java Edition,
- Berkeley DB XML



Obr. 6 Dizajn Oracle Berkeley DB

[19] http://en.wikipedia.org/wiki/Chord_(peer-to-peer)
[20] http://www.zib.de/de/pvs/projekte/projektdetails/article/scalaris.html
[21] http://www.oracle.com/technetwork/database/berkeleydb

Z jej názvu je možné vytušiť, že je priamo podporovaná spoločnosťou Oracle, ale vznikla na univerzite v Berkeley. Ako je vidieť na Obr. 6, Oracle Berkeley DB poskytuje 4 rôzne typy rozhraní pre ukladanie a čítanie dát:

- SQL syntaxe,
- XQuery,
- Java objektov,
- Key/Value párov

Ďalším riešením, založeným na princípe kľúč/hodnota, je MemcacheDB[22]. Tento projekt vychádza z projektu Memcached[23], ktorý je distribuovaným caching systémom pre rozličné objekty (databázové systémy, webové aplikácie, atď.). MemcacheDB, ako backend na ukladanie údajov, využíva úložisko z riešenia Berkley DB.

Posledným riešením, ktorému sa budeme venovať v sekcii kľúč/hodnota, je distribuovaná databáza Redis[24]. Podobne, ako spomenuté distribuované databázy, používa Redis na ukladanie pár kľúč/hodnota s tým, že kľúč môže obsahovať:

- texty (stringy),
- hešovacie kľúče,
- zoznamy,
- množiny,
- a utriedené množiny.

V riešení Redis sú všetky uzly navzájom pospájané a na určenie, kde ktorá replika patrí, sa používa hešovacia tabuľka s 4096 hodnotami. Jednotlivé uzly si medzi sebou stále vymieňajú informácie o svojom stave a na základe toho si upravujú stav o tom, ktorý uzol aké repliky a kľúče udržuje. Nie všetky uzly sú si rovnocenné, ako je tomu napríklad v projekte Scalaris (v rámci databáz typu kľúč/hodnota). Dá sa povedať, že Redis používa architektúru klient/server. Štandardne sú nastavené 2 repliky (klienti) + jedna na uzle servera (v podstate 3). Rozdiel medzi uzlami typu klient/server je v tom, že uzly typu klient nikdy nedostanú požiadavku na zápis (len na čítanie) páru kľúč/hodnota. Zápis je možný iba na serveri, ktorý neskôr prereplikuje/premigruje hodnotu na jednotlivých klientov. To, že sa používa 4096 rôznych hešovacích kľúčov na ukladanie kľúča ešte neznamená, že musíme mať toľko uzlov. Z toho potom vyplýva, že jednotlivé fyzické stroje budú slúžiť ako veľa uzlov (klientov aj serverov). Redis systém preto musí replikáciu vyriešiť na fyzicky rozdielnych strojoch. Takisto klientska aplikácia. Ak sa dopytuje na určitý kľúč uzla, ktorý dané dáta neobsahuje, dostane iba informáciu o tom, kde sú dáta fyzicky uložené (uzol) a následne sa musí opýtať znova.

2.4 Projekty inšpirované projektom Dynamo

V tejto časti si rozoberieme jednotlivé projekty inšpirované projektom Dynamo kladúcich dôraz na druhé 2 vlastnosti a to:

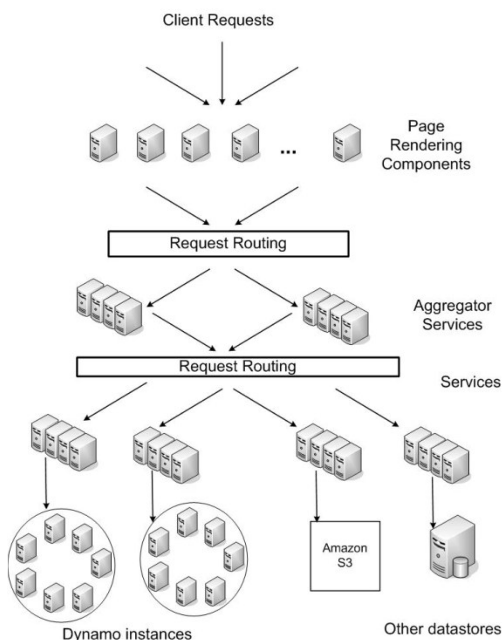
- **A**vailability
- a **P**artition tolerance.

[22] http://memcachedb.org
[23] http://memcached.org
[24] http://redis.io

V prípade projektov tohto typu sú najčastejšie vyvíjané projekty typu kľúč/hodnota, ale nájdu sa aj dokumentovo orientované projekty a dokonca aj stĺpcovo orientovaná databáza.

2.4.1 Distribuované databázy typu kľúč/hodnota

Dynamo je dátové úložisko s jednoduchým kľúč/hodnota rozhraním. Umožňuje efektívne využitie odkladacieho priestoru a je jednoducho škálovateľný, čo sa týka rastu dát a počtu dopytov nad dátami. Dynamo je určene iba pre použitie v rámci Amazon služieb (Obr. 7). Na dosiahnutie škálovateľnosti a robustnosti v produkčnej prevádzke s úlohami ako je rozdeľovanie záťaže medzi uzlami, detegovanie chýb a ich lokalizácia, obnovenie systému po výpadku, synchronizácia replík, zvládanie záťaže, konzistencia stavu, synchronizácia a plánovanie úloh, monitorovanie systému, atď., využíva Dynamo niekoľko techník, ktoré sú opísane v [3].



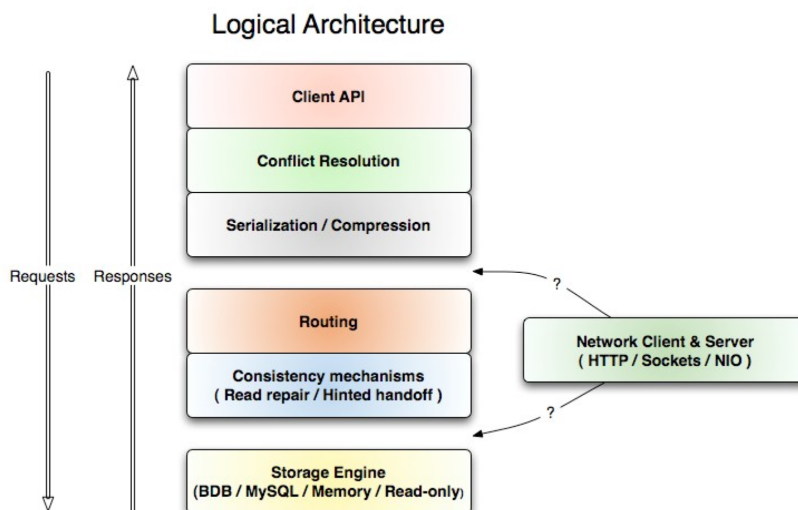
Obr. 7 Architektúra platformy Amazon [3]. Dynamický obsah webových stránok Amazonu je generovaný komponentmi Page Rendering Components, ktoré využívajú rôzne služby (Services). Služba môže na svoju činnosť využívať rôzne dátové úložiská (datastores). Niektoré služby sa správajú ako agregátory. Spájajú viacero iných služieb dokopy (Aggregator services).

Ďalšou spoločnosťou, ktorá potrebovala vyriešiť distribuovaný prístup a replikáciu údajov, bola spoločnosť LinkedIn[25] prevádzkujúca rovnomennú službu. Spoločnosť LinkedIn sa inšpirovala článkom o distribuovanej databáze Dynamo od spoločnosti Amazon a vyvinula vlastný produkt Voldemort[26]. Voldemort je podobný Dynamu, avšak je open-source a je teda do neho možné prispievať a ľubovoľne ho upravovať. Skladá sa

[25] http://www.linkedin.com

[26] http://project-voldemort.com

z rôznych logických blokov a preto je možné si „poskladať“ vlastné riešenie, ktoré najviac vyhovuje potrebám aplikácie. Logické bloky systému Voldemort je možné vidieť na Obr. 8.



Obr. 8 Logické bloky systému Voldemort

Každý z týchto blokov je zodpovedný za všetky operácie, ktoré v týchto systémoch existujú (read/get, write/put/store, delete). Napríklad vrstva Routing je pri operácii write zodpovedná za uloženie replík na N uzlov a v prípade chyby pri zápise/replikácii rieši vzniknuté problémy. Voldemort sa snaží riešiť aj konzistenciu dát, ktorá ale nie je na 100% zaručená. V prípade operácie write sa replikovanie vykonáva offline-ovo a vzniknuté inkonzistencie sa riešia až v prípade operácie read. Projekt Voldemort podporuje verzionovanie každej operácie write a to tak, že pridáva ku každej operácii write časový vektor, ktorý tvorí pár server:verzia (časová značka nemôže byť použitá, pretože v distribuovanom systéme sa uzly objavujú a miznú, replikácia trvá nejaký čas, atď.).

2.4.2 Stĺpcovo orientované distribuované databázy

Najznámejšou stĺpcovo orientovanou databázou inšpirovanou projektom Dynamo je Apache Cassandra. Apache Cassandra je distribuovaný úložný systém na manažovanie veľmi veľkého objemu štruktúrovaných dát rozmiestnených na väčšom počte uzlov, pričom je zaručená vysoká dostupnosť služby s koncepciou „no SPOF“ (no Single Point Of Failure - bez kritického bodu zlyhania). Vychádza z technológií Amazon Dynamo a Google BigTable. Z Dynama využíva spôsob manažovania klástra, replikácie dát a jeho mechanizmy na odolnosť voči chybám. Z Google BigTable zase preberá základ dátového modelu a architektúru úložiska dát. Cassandra je určená pre klástre zostavené z bežne dostupných PC, kde sa počet uzlov pohybuje rádovo v stovkách a ktoré môžu byť rozmiestnené v rôznych datacentrách. V takto riešených infraštruktúrach dochádza často k poruchám komponentov (disky uzlov, CPU, a pod.), či k výpadkom samotných uzlov [7]. Cassandra vznikla s potrebou výkonného škálovateľného a spoľahlivého riešenia dátového úložiska vo firme Facebook na zabezpečenie niektorých služieb vyžadujúcich nízku latenciu, ako je napríklad Inbox Search. Služba Inbox Search umožňuje vyše 500 mil.

používateľom vyhľadávať v správach podľa mena odosielateľa alebo kľúčových slov. Problémom bolo zachovať vysokú priepustnosť vstupných dát (miliardy záznamov) pri narastajúcom počte používateľov, ako aj vyriešiť replikáciu dát medzi geograficky vzdialenými datacentrami tak, aby služba dosahovala čo najnižšiu dobu odozvy.

Dátový model

Dátový model Cassandry tvoria tabuľky, ktorých riadky obsahujú kľúč a hodnotu. Kľúčom je ľubovoľne dlhý reťazec (typicky 16B až 36B) a hodnotu tvorí štruktúrovaný objekt. Štruktúru určujú stĺpce pričom ich počet môže byť pre jednotlivé riadky rôzny (až 2 miliardy stĺpcov na riadok, 2GB dát na stĺpec)[27]. Tabuľky sú rozdelené a distribuované medzi uzly klástra. Aj keď Cassandra umožňuje použitie viacerých tabuliek v rámci jednej schémy, prakticky sa používa vždy len jedna tabuľka na schému. Každá operácia s riadkom tabuľky je atomická na jeho repliku bez ohľadu nato, z koľkých stĺpcov riadku sa číta alebo do koľkých sa zapisuje. Stĺpce je možné zoskupovať do tzv. column families podobne ako je to v Google BigTable a column families ďalej do super column families. Zároveň je možné určiť usporiadanie stĺpcov v rámci (super) column family podľa času alebo názvu (každý stĺpec má svoju časovú značku).

Škálovateľnosť

Kľúčovou vlastnosťou Cassandry je škálovateľnosť. Tá si vyžaduje schopnosť dynamicky rozdeľovať a distribuovať dáta medzi jednotlivé uzly klástra. Cassandra využíva pri prístupe k distribuovaným dátam konzistentné hešovanie s hešovacou funkciu zohľadňujúcou usporiadanie. Pri konzistentnom hešovaní je výstupný rozsah hešovacej funkcie cyklicky ohraničený. Za najväčšou výstupnou hodnotou kľuča, ktorý je priradený uzlu, nasleduje najmenší kľúč. Každý uzol má náhodne priradený jeden kľuč z hešovacej tabuľky a obsluhuje požiadavky týkajúce sa kľúčov medzi ním a predchodcom uzla. Pridaním nového uzla do klástra sa odbremení iný uzol tým, že nový uzol dostane kľúč z hešovacej tabuľky a prevezme časť obsluhovaných kľúčov zaťaženého uzla. Taktiež sa zo zaťaženého uzla na nový uzol presunú dáta zodpovedajúce prevzatému rozsahu kľúčov.

Konzistencia dát

Vychadzajúc z CAP teorémy (CAP theorem - Consistency, Availability, tolerance to network Partitions), Cassandra predstavuje AP systém, ktorý je schopný zabezpečiť eventuálnu konzistenciu dát. To znamená, že pri replikácii je časť aktuálnych dát uložená na uzle, zatiaľ čo ich staršia verzia je na iných uzloch, ale napriek tomu je zaručené, že všetky uzly uvidia aktuálnu verziu dát. Konzistencia je v Cassandre zaručená pokiaľ platí vzťah $W + R > N$, kde W je počet zapísaných replík, R počet prečítaných replík a N replikačný faktor. Replikačný faktor N vyjadruje počet replík jedného riadku tabuľky, kde každá replika sa nachádza na inom uzle. Pomer medzi konzistenciou dát a latenciou sa v Cassandre dá nastavovať, avšak zvýšenie konzistencie bude vždy na úkor latencie.

2.4.3 Dokumentovo orientované distribuované databázy

CouchDB[28] je ďalšou populárnou dokumentovo orientovanou škálovateľnou NoSQL ba Dopytovanie sa rieši pomocou JSON objektov a existuje množstvo klientských implementácií pre rôzne jazyky.

---

[27] http://wiki.apache.org/cassandra/CassandraLimitations

[28] http://couchdb.apache.org/

3 Distribuované spracovanie dát

V predchádzajúcej časti sme opisovali časti systému MapReduce [2], predovšetkým distribuovaný súborový systém a distribuovanú databázu. V tejto časti sa zameriame na schopnosť MapReduce úloh spracovávať rozsiahle dáta [13].

Existujú tri voľne dostupné implementácie MapReduce architektúry:

- Hadoop, vyvinutý spoločnosťou Apache Foundation spolu s projektami Lucene a Nutch. Spoločnosť Yahoo! Začala používať Hadoop v roku 2008 na 10 000 jadrách [11] v produkčnom prostredí [10].
- Phoenix[29], vyvinutý na Stanfordskej Univerzite, implementovaný v C++.
- Disco[30], vyvinutý spoločnosťou Nokia, implementovaný vo funkcionálnom jazyku Erlang.

V príspevku sme už spomínali dva projekty spoločnosti Apache Software Foundation[31]: Hadoop a HBase. Hadoop je Java implementáciou paradigmy MapReduce [2] navrhnutého a vyvinutého spoločnosťou Google na spracovanie a generovanie veľkých objemov dát a zahŕňa tiež implementáciu GFS (Google Files System) [4] nazývanú HDFS (Hadoop Distributed File System). HBase je podprojekt projektu Hadoop. Frameworku Hadoop poskytuje podobnú funkcionalitu ako BigTable [6] distribuovanému súborovému systému GFS. V mnohých veciach je podobný klasickým databázovým systémom. V príspevku zároveň predstavíme architektúru uvedených systémov, distribuovaný súborový systém a demonštrujeme jednoduchú aplikáciu a jej portovanie do frameworku MapReduce.

3.1 Architektúra MapReduce

Prvým, kto takúto relatívne jednoduchú a hlavne funkčnú architektúru navrhol, bola spoločnosť Google. Prostredie, v ktorom sa v spoločnosti Google vykonávajú všetky operácie s dátami, sa dá opísať v nasledujúcich piatich bodoch (platných v čase zverejnenia článku [2]):

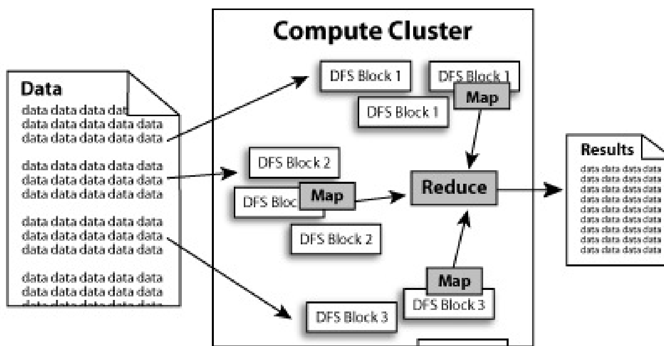
1. Jednotlivé pracovné stanice (uzly) majú štandardne dvojjadrové x86 procesory, beží na nich operačný systém Linux a majú 2-4GB pamäte (vo väčšine prípadov ide o bežné pracovné stanice, a nie servery s vysokým výkonom).
2. Používa sa bežné sieťové pripojenie s rýchlosťou 100Mb/s, resp. 1Gb/s na úrovni pracovnej stanice.
3. Klaster pozostáva zo stoviek až tisícok pracovných staníc (z toho dôvodu sú výpadky pracovných staníc bežné).
4. Používané disky sú vo väčšine prípadov lacné, s IDE rozhraním, priamo pripojené k pracovným staniciam. Používajú distribuovaný súborový systém GFS [4]. Systém štandardne používa replikáciu, aby zabezpečil dostupnosť a spoľahlivosť systému postaveného nad nespoľahlivým hardvérom.
5. Používateľ posiela do systému procesy, ktoré plánovač rozdelí na voľné pracovné stanice a spustí [2].

---

[29] http://mapreduce.stanford.edu/

[30] http://discoproject.org/

[31] http://www.apache.org/

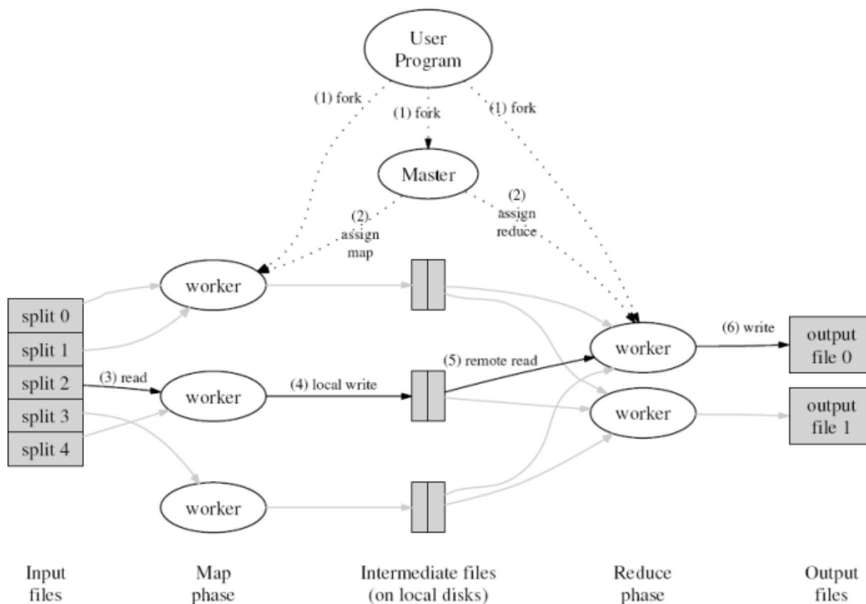


Obr. 9 MapReduce architektúra

Základná idea je teda distribuovať dáta na jednotlivé uzly v rámci distribuovaného file systému (pozri ďalšiu časť) a nad týmito dátami zabezpečiť spustenie Map metódy. Výsledky Map metódy sú vstupom pre metódu Reduce ktorá spracuje výstup Map metód do požadovaného výsledku (Obr. 9).

3.2 Framework MapReduce

Aby sme mohli využiť distribuovaný systém opísaný v časti 2.1, potrebujeme mať framework, ktorý bude zodpovedný za manažovanie procesov (rozdeľovanie, spúšťanie, atď.). V tejto časti si opíšeme framework MapReduce [2] navrhnutý spoločnosťou Google, ktorý umožňuje spúšťať paralelné procesy bez znalosti paraleleného programovania. Programovací model prostredia MapReduce je opísaný nižšie.



Obr. 10 Spustenie a vykonanie procesu v prostredí MapReduce (Google)

Na vstupe výpočtu je množina párov kľúč/hodnota. Vývojár musí implementovať dve funkcie: `Map` a `Reduce`. Funkcia `Map` dostane na vstupe pár kľúč/hodnota a vyprodukuje preň dočasný pár kľúč/hodnota. Prostredie MapReduce zoskupí dočasné hodnoty priradené tomu istému kľúču *K* a posunie ich do funkcie `Reduce`. Funkcia `Reduce` má na vstupe dočasný kľúč *K* a množinu hodnôt k nemu priradených. Tieto hodnoty spojí dokopy, pričom je predpoklad, že množina hodnôt po ukončení funkcie `Reduce` bude menšia. Dočasné hodnoty sú poskytované do funkcie `Reduce` ako iterátor. To umožňuje zvládnuť zoznam hodnôt, ktoré sú príliš veľké na načítanie do pamäte.

Na Obr. 10 je opísaný spôsob spustenia procesu v prostredí MapReduce od spoločnosti Google. Výpočtový kláster MapReduce (v implementácií Hadoop) pozostáva z jedného uzla JobTracker (v menších klástroch sa používa na tom istom uzle ako uzol NameNode) zodpovedného za manažovanie spúšťaných procesov. Uzly TaskTracker (uzly TaskTracker a DataNode musia byť totožné) sú zodpovedné za priame vykonávanie zverených úloh.

V nasledujúcich krokoch si popíšeme spustenie procesu v prostredí Hadoop:

1. Na uzle JobTracker sa spustí požadovaný proces, ktorý má naimplementované funkcie `Map` a `Reduce`.
2. JobTracker preskúma voľné uzly (musí na nich bežať TaskTracker) a podľa potreby (v závislosti od veľkosti vstupných dát) pridelí potrebné množstvo výpočtových uzlov (TaskTracker v závislosti od počtu jadier zvládne počítať 2 až 4 úlohy naraz). Súčasne je spustená aj úloha `Reduce` (v závislosti od množstva dát a uzlov sa môže spustiť aj viac úloh `Reduce`).
3. Po dokončení niektorej z `Map` úloh sa jej výsledky prekopírujú na niektorý z uzlov, kde beží úloha `Reduce`. Výsledky sa utriedia a čaká sa na ukončenie všetkých úloh `Map`.
4. Po dokončení všetkých úloh `Map` sa spustia úlohy `Reduce` a po ich ukončení dostaneme utriedený zoznam párov kľúč/hodnota.

V prostredí MapReduce (v implementácii Hadoop) sú aj ďalšie funkcie, ktoré môže vývojár naimplementovať. Sú to funkcie:

- `Combiner` – spustí funkciu `Reduce` hneď po úlohe `Map` na danom uzle (pomáha znížiť množstvo prenesených dát cez internet),
- `Partitioner` – používateľ prostredia MapReduce môže špecifikovať počet úloh `Reduce` spustených v jednej úlohe, pričom funkcia `Partitioner` je zodpovedná za rozdelenie dočasných kľúčov *K* na jednotlivé úlohy `Reduce`. Niekedy sú napríklad výsledné kľúče typu URL a vtedy je dobré mať výsledky z jednej URL v jednom súbore (spracované jednou úlohou `Reduce`),
- `Reporter` – je funkcia, ktorá oznamuje stav procesu a úloh.

3.3 Praktická ukážka úlohy v MapReduce

V tejto časti si ukážeme a detailnejšie rozoberieme jednoduchý program na zisťovanie výskytu slov v dokumente (WordCount[32]). Nižšie je uvedený listing (funkcií `Map` a `Reduce`) tohto programu v jazyku Java implementovanom pre Hadoop cluster.

Funkcia `Map`:

```
public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
  Text word = new Text();
  String line = value.toString().toLowerCase();
  StringTokenizer tokenizer = new StringTokenizer(line);
  while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
  }
}
```

Funkcia `Reduce`:

```
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
  int sum = 0;
  while (values.hasNext())
    sum += values.next().get();
  output.collect(key, new IntWritable(sum));
}
```

Funkcia `Map` dostane časť súboru (riadok) obsahujúceho text, ktorému priradí token a pre každý token (dočasný kľúč) priradí hodnotu „one“. Funkcia `Reduce` následne dostane dočasné páry kľúč/hodnota. Pre dočasný kľúč iteruje cez všetky jeho hodnoty, pričom ich počet sumuje. Keď prebehne cez všetky hodnoty, sumu zapíše ako výslednú hodnotu pre kľúč, ktorý vlastne tvorí token (slovo).

4 Dátové sklady nad MapReduce

V tejto kapitole opisujeme hlavne dátový sklad Hive [12] postavený nad architektúrou MapReduce aj s jednoduchou ukážkou práce s Hive. Okrem systému Hive existujú aj iné, ktoré poskytujú rovnakú alebo podobnú funkcionalitu ako sú napr.: Pig (podprojekt projektu Hadoop vyvíjaný hlavne spoločnosťou Yahoo![33]), Sawzall[34] od spoločnosti Google, JAQL[35] od spoločnosti IBM, Scope[36] od spoločnosti Microsoft a YAML MapReduce od firmy Greenplum[37].

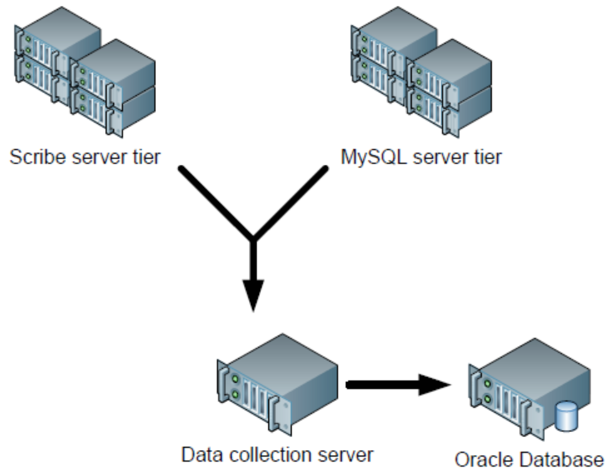
Projekt Hive bol naštartovaný spoločnosťou Facebook[38] (4 prispievatelia, všetci z Facebooku), z dôvodu spracovania veľkého množstva údajov v rámci sociálnej siete

---

[32] Originálny zdrojový kód je prebratý z http://hadoop.apache.org/core/docs/current/mapred_tutorial.html
[33] http://research.yahoo.com/node/90
[34] http://research.google.com/archive/sawzall.html
[35] http://www.almaden.ibm.com/cs/projects/jaql/
[36] http://www.cs.uwaterloo.ca/~kmsalem/courses/CS848W10/presentations/Aluc-Scope.pdf
[37] http://www.greenplum.com/technology/
[38] http://www.facebook.com/

Facebook. Pôvodný proces spracovávania dát spoločnosťou Facebook bol nasledovný[39] (Obr. 11):

- Dáta boli zbierané pomocou úloh zadaných v plánovači (boli to úlohy spúšťané v nočných časoch) a dáta boli zbierané do Oracle databázy.
- ETL[40] úlohy boli implementované v Pythone.
- V roku 2006 to bolo denne niekoľko 10 GB, v roku 2008 už 200GB nových dát, 5TB (komprimovaných) v roku 2009.



Obr. 11 Zbieranie používateľských dát zo sociálnej siete Facebook

V nasledujúcom odseku je uvedené na na čo všetko sa Hive dá použiť:

- spracovanie logov,
- dolovanie textových informácií,
- indexovanie dokumentov,
- modelovanie správania sa používateľov (túto vec má rád môj priateľ, možno ju budeš mať aj ty) a testovanie rôznych hypotéz.

4.1 Hadoop ako distribuované dátový sklad

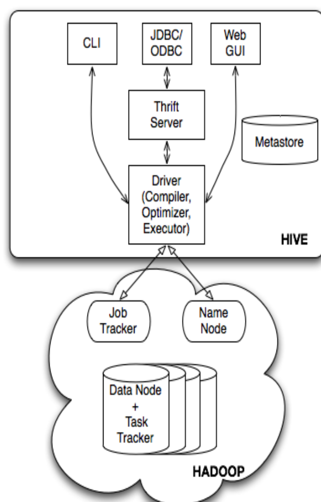
V tejto časti si predstavíme ako sa dá Hadoop použiť ako distribuovaný sklad pre dáta zo sociálnej siete Facebook. Dáta zo Scribe a MySQL servera sú nahrávané do HDFS a použijú sa MapReduce úlohy na spracovanie týchto dát. Čo v tomto návrhu chýba:

- jazyk, ktorým by sa dalo jednoducho tieto úlohy písať (bez potreby písať MapReduce programy),
- editor príkazového riadku, v ktorom by sa tieto úlohy mohli písať,
- schémy o jednotlivých tabuľkách v databázach

---

[39] http://www.cloudera.com/wp-content/uploads/2010/01/6-IntroToHive.pdf
[40] http://en.wikipedia.org/wiki/Extract,_transform,_load

Hive na všetky tieto otázky odpovedá (architektúru projektu Hive je možné vidieť na Obr.12).



Obr.12 Architektúra systému Hive

Poskytuje vlastný editor príkazového riadku (tzv. hive>), ktorý je podobný MySQL editoru, ďalej poskytuje jazyk, ktorým je možné písať dopyty (zároveň je tu podpora aj pre JDBC klientov), uloženie metadát o databázach a tabuľkách. Najväčšou výhodou projektu Hive je možnosť písať SQL dopyty, pričom Hive preloží tieto dopyty do Map a Redcue úloh (sú použité orientované acyklické grafy s veľmi jednoduchou optimalizáciou). Fyzicky je dátový sklad uložený v HDFS v jednoduchých (plain) súboroch oddelených špeciálnym znakom alebo v špeciálnych sekvenčných súboroch[41] v adresári: „/home/hive/warehouse“ a jednotlivé tabuľky sú uložené v podadresároch. Takisto je možné použiť vlastné formáty, pričom treba implementovať serializér a deserializér pre daný formát.

4.2 Pig

Pig je nadstavba Hadoop-u, ktorá zjednodušuje programovanie MapReduce aplikácií. Prináša vysoko-úrovňový programovací jazyk nad *map* a *reduce* funkciami. Pig sa skladá z dvoch hlavných komponentov:

- programovací jazyk *Pig Latin*
- kompilátor

Pig Latin[42] zjednodušuje písanie kódu aplikácie tým, že je zameraný na tok dát. Program v Pig-u je postupnosť krokov, kde každý krok je jedna vysoko-úrovňová operácia nad dátami. Operácie nad dátami pripomínajú operácie z relačných databáz, napr. filter, union, group alebo join. Podporuje dátové schémy pri spracovávaní štruktúrovaných dát a je schopný pracovať aj s neštruktúrovaným textom, či XML, v tomto má teda širšie použitie ako Hive, dá sa použiť aj na neštruktúrované dáta v zmysle kapitoly 3. Dá sa rozširovať

[41] http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/SequenceFile.html
[42] http://pig.apache.org/

vlastnými funkciami. Zjednodušuje spájanie a zreťazovanie MapReduce úloh. Programátor sa teda nemusí zaoberať implementáciou funkcií *map* a *reduce*, čo by bolo pri tvorbe komplexnejších aplikácií oveľa náročnejšie [8].

4.3 Ukážka práce so systémom Hive

V ukážke distribuovaného skladu dát použijeme diela zo stránky http://zlatyfond.sme.sk, pričom sme použili 2 diela od Pavla Országha-Hviezdoslava: Hájnikova žena[43] a Ežo Vlkolinský[44] (museli byť použité PDF verzie, pretože textové obsahujú množstvo chýb a preklepov).

V nasledujúcich krokoch si predstavíme, čo môžeme s týmito dvomi dielami, s Hadoop-om a dátovým skladom Hive urobiť. Budeme používať MapReduce klaster s implementáciou Hadoop inštalovanou na UISAV. V čase písania článku sme mali k dispozícii 7 pracovných uzlov (slaves) a 1 riadiaci server (master).

1. Predspracovanie dát ako napr.: konverzia pdf do textovej podoby, vyhodenie diakritiky a konverzia veľkých znakov na malé.
2. Nahratie dát do HDFS:

```
$hadoop fs -copyFromLocal ezo.txt /user/hadoop/ezo.txt
$hadoop fs -copyFromLocal zena.txt user/hadoop/zena.txt
```

3. Spočítanie slov (neberieme do úvahy slovenčinu, teda žiaden stemmer ani lematizátor) v jednotlivých dielach:

```
$hadoop jar hadoop-0.19.1-examples.jar grep ezo.txt ezo_freq '\w+'
$hadoop jar hadoop-0.19.1-examples.jar grep zena.txt zena_freq '\w+'
```

4. Zmažeme logy, ktoré boli vytvorené Hadoop-om:

```
$hadoop fs -rmr ezo_freq/_logs
$hadoop fs -rmr zena_freq/_logs
```

5. Prejdeme do editora príkazového riadku systému Hive a vytvoríme tabuľky zena a ezo do ktorých presunieme (`LOAD DATA INPATH`) dáta predspracované Hadoopom. Ak by sme chceli pôvodné dáta ponechať v HDFS tak použijeme príkaz: `LOAD DATA LOCAL INPATH`. Toto načítanie dát do tabuľky je veľmi rýchle (0,118 sekundy v našom prípade) pretože, sa jedná len o presunutie dát v rámci HDFS. V prípade že, sú dáta uložené v relačnej databáze, môžeme použiť ďalší z podprojektov projektu Hadoop Sqoop[45] (pôvodne vyvíjaný spoločnosťou Cloudera[46]) na prevedenie relačných dát do HDFS:

```
$hive
hive> CREATE TABLE zena (freq INT, word STRING) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
hive> CREATE TABLE ezo (freq INT, word STRING) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
hive> LOAD DATA INPATH "zena_freq" INTO TABLE zena;
hive> LOAD DATA INPATH "ezo_freq" INTO TABLE ezo;
```

[43] http://zlatyfond.sme.sk/dielo/18/Hviezdoslav_Hajnikova-zena/1
[44] http://zlatyfond.sme.sk/dielo/146/Orszagh-Hviezdoslav_Ezo-Vlkolinsky/1
[45] http://www.cloudera.com/downloads/sqoop/
[46] http://www.cloudera.com/

6. Test, že dáta boli v poriadku načítané a takisto ukážka SQL dopytu v prostredí Hive. Tento SQL dopyt je rozdelený na dve MapReduce úlohy, ktorých vykonanie trvá na klastri 44,633 sekundy:

```
hive> SELECT * FROM ezo SORT BY freq DESC LIMIT 10;
```

Všetky znaky malé		Aj veľké znaky	
Slovo	**Počet**	**Slovo**	**Počet**
sa	656	sa	656
a	598	a	440
i	427	v	362
v	375	i	352
na	322	na	303
co	271	si	242
to	254	co	232
si	245	to	214
len	201	s	182
tak	195	len	172

7. Najčastejšie sa vyskytujúce frekvencie slov v diele Hájnikova žena. Tento dopyt je takisto rozdelený na 2 úlohy a jeho vykonanie trvá 45,718 sekundy:

```
hive> SELECT freq, COUNT(1) AS f2 FROM zena GROUP BY freq SORT BY f2
DESC;
```

Všetky znaky malé		Aj veľké znaky	
Počet slov	**Počet výskytov**	**Počet slov**	**Počet výskytov**
8727	1	9153	1
1805	2	1854	2
691	3	704	3
326	4	355	4
190	5	191	5
137	6	137	6
83	7	94	7
75	8	66	8
38	11	38	11
38	9	37	9
38	10	36	10

8. Orientovaný acyklický graf vykonávania „SQL“ dopytu získame nasledovným príkazom:

```
hive> EXPLAIN SELECT freq, COUNT(1) AS f2 FROM zena GROUP BY freq SORT
BY f2 DESC;
```

9. Ďalšou vecou, ktorú Hive dokáže je ďalší „SQL“ príkaz JOIN. Najprv vytvoríme spoločnú tabuľku a následne obidve spojíme do jednej (samozrejme slová musia byť rovnaké). Toto spojenie je jedna MapReduce úloha, ktorá trvá 22,451 sekúnd:

```
hive> CREATE TABLE spojena (word STRING, ezo_f INT, zena_f INT);
hive> INSERT OVERWRITE TABLE spojena SELECT e.word, e.freq, z.freq FROM
ezo e JOIN zena z ON (e.word = z.word);
```

10. Posledná vec bude zistenie, ktoré slovo sa najčastejšie vyskytuje v obidvoch dielach spolu, pričom vypíšeme aj jednotlivé frekvencie. Tento dopyt je takisto rozdelený na 2 úlohy a jeho vykonanie trvá 44,697 sekundy:

```
hive> SELECT word, ezo_f, zena_f, (ezo_f + zena_f) AS s FROM spojena
SORT BY s DESC LIMIT 10;
```

Všetky znaky malé				Aj veľké znaky			
Slovo	Počet slov Ežo	Počet slov Žena	Počet slov spolu	Slovo	Počet slov Ežo	Počet slov Žena	Počet slov spolu
sa	656	970	1626	sa	656	970	1626
a	598	865	1463	a	440	742	1182
v	375	809	1184	v	362	786	1148
i	427	521	948	na	303	499	802
na	322	535	857	i	352	420	772
co	271	347	618	co	232	281	513
jak	118	432	550	jak	113	381	494
to	254	248	502	to	214	214	428
len	201	242	443	si	242	166	408
tak	195	238	434	z	158	249	407

5 Záver

V príspevku sme predstavili základné architektúry na škálovateľné spracovanie dát pomocou horizontálneho škálovania z bežne dostupných počítačov. Zamerali sme sa na jednotlivé časti takýchto systémov:

- Distribuovaný súborový systém,
- NoSQL databázové systémy,
- MapReduce architektúra na spracovanie rozsiahlych dát
- a distribuované dátové sklady.

Pre architektúru MapReduce a implementáciu Hadoop ako aj dátový sklad Hive sme uviedli jednoduché príklady použitia.

Poďakovanie

Táto práca bola podporená projektmi: Projekt ITMS: 26240220029, SMART II ITMS: 26240120029 a VEGA 2/0184/10.

Literatúra

1. Brewer, E.: Towards robust distributed systems. (Invited Talk), *Principles of Distributed Computing*, Portland, Oregon (2000)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified DataProcessing on Large Clusters, OSDI'04: *Sixth Symposium on Operating System Design and Implementation*, San Francisco (2004), http://labs.google.com/papers/mapreduce.html
3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available

key-value store. In: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP '07)*. ACM, New York (2007), 205-220, http://doi.acm.org/10.1145/1294261.1294281
4. Ghemawat, S., Gobioff, H., Leung, S.: The Google File System, In: *19th ACM Symposium on Operating Systems Principles*, Lake George (2003), http://labs.google.com/papers/gfs.html
5. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News 33*, 2 (June 2002), 51-59.
6. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes A., Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data, In: *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Seattle (2006), http://labs.google.com/papers/bigtable.html
7. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. In: *ACM SIGOPS Operating Systems Review*, Volume 44, Issue 2, April 2010, 35-40, http://doi.acm.org/10.1145/1773912.1773922
8. Lam, Ch.: *Hadoop in Action*. Manning Publications Co., 2010, 336 s. ISBN 9781935182191
9. Patterson D.A., Gibson, G., Katz., R.H.: A case for redundant arrays of inexpensive disks (RAID). In: *Proceedings of the 1988 ACM SIGMOD international conference on Management of data (SIGMOD '88)*, Haran Boral and Per-Ake Larson (Eds.). ACM, New York (1988), 109-116.
10. Zawodny, J.: Yahoo! Launches World's Largest Hadoop Production Application, *Yahoo! Developer Network Blog*, http://developer.yahoo.com/blogs/hadoop/posts/2008/02/yahoo-worlds-largest-production-hadoop, (2008)
11. Zawodny, J.: Open Source Distributed Computing: Yahoo's Hadoop Support, *Yahoo! Developer Network blog*, http://developer.yahoo.net/blog/archives/2007/07/yahoo-hadoop.html, (2007)
12. Šeleng, M.: Distribuované spracovanie dát nad MapReduce architektúrou (Hadoop a Hive). In *5th Workshop on Intelligent and Knowledge Oriented Technologies* : WIKT 2010 proceedings, 2010, p. 48-53. ISBN 978-80-970145-2-0.
13. Šeleng, M., Laclavík, M., Hluchý, L.: Použitie MapReduce architektúry na spracovanie veľkých informačných zdrojov. In WIKT 2008 : *3rd Workshop on Intelligent and Knowledge Oriented Technologies*, 2009, p. 27-34. ISBN 978-80-227-3027-3

Annotation:

Scalable platforms for data processing and data warehousing

With growing amount of digital data in corporate world or on the web new challenges arisen. We need to store, process and request more and more data. Companies with need to process large data such as Google, Facebook, Amazon, Yahoo! or Twitter were forced to solve this problem of their core business and they came up with innovative solutions for scalable information processing frameworks build on commodity PCs. We describe these frameworks in our tutorial focusing on Google's MapReduce, Amazon's Dynamo or open-source frameworks like Hadoop or Cassandra supported by large companies like Yahoo!, Facebook or Twitter. These frameworks can be used for data processing or replace databases and data warehouses in some application where scalability is critical.

Geolokace a geolokační techniky

Jaroslav SRP

Ústav bezpečnostních technologií a inženýrství, FD ČVUT v Praze
Konviktská 20, 110 00 Praha 1
srp@fd.cvut.cz

Abstrakt. Dovednost určit polohu elektronického zařízení, označovaná jako lokace, lokalizace nebo geolokace, je dnes jednou z klíčových technologií využívanou pro určení přesné polohy (nebo alespoň oblasti působení) dopravních prostředků (letadel, osobních a nákladních vozidel) nebo lidí (např. v případě jejich pohřešování). Využívána je také pro určení místa páchání trestné činnosti (místo, odkud hacker vede svůj útok), ale také ve službách, kterými jsou např. navigace, určení jazykové verze webové aplikace, cílená reklama, apod. Určit polohu elektronického objektu, tj. takového, který přijímá nebo dokonce sám odesílá elektronická data, je možné mnoha způsoby, které se označují jako geolokační techniky. Každá z nich má své přednosti ale i nedostatky a je použitelná jen za jistých předpokladů. Jejich souhrnný popis a základy fungování shrnuje tento tutoriál.

Klíčová slova: geolokace, lokalizace, triangulace, trilaterace, GPS.

1 Úvod

1.1 Geolokace

Každý objekt na Zemi má v konkrétním čase své umístění. Objekt může být statický nebo dynamický. Statický je takový, který svou polohu v čase nemění a jehož poloha může být trvale zanesena v různých databázích, ať už tištěných nebo elektronických, po celou dobu jeho životnosti. Všeobecně se předpokládá, že příslušná databáze byť i historických dat statických objektů, obsahuje stále platné hodnoty. Jednoduchým příkladem mohou být souřadnice domů (např. zeměpisné souřadnice, eventuelně poštovní adresy apod.) opomine-li se možnost změny popisného čísla, zanedbatelné změny souřadnice vlivem posunu litosférických desek atd.

Zcela odlišný pohled se řeší v případě dynamických objektů, tj. těch, jejichž poloha se v čase (od jejich vzniku až do zániku) mění. Při zanesení aktuální polohy uvedené kategorie objektů do zvolené databáze v daném čase nemusí být naměřené polohy platné v čase budoucím. Nicméně, i u každého dynamického objektu zůstává jeho poloha po jistou dobu platná – může jít o zlomky sekund až po dny nebo roky. Bohužel takový údaj o sobě předmětné objekty typicky neposkytují. Proto lze dobu platnosti uložených záznamů pouze odhadovat a po stanovené době ji považovat za neplatnou, i když k její změně mohlo dojít dříve nebo naopak později. Polohu dynamických objektů je proto nutné pravidelně aktualizovat.

Statické i dynamické objekty mohou být z pohledu zjišťování jejich polohy rozděleny do dvou základních kategorií. Na objekty, které oznamují svoji polohu dobrovolně, a na ty, které ji neoznamují (tzn., že ji buď neznají a nejsou schopny ji určit, nebo ji zatajují

úmyslně s cílem skrývat se). Eventuelně mohou předmětnou polohu udávat úmyslně chybně.

Určování polohy naposledy zmíněné skupiny objektů se dnes řeší v rámci mnoha výzkumných projektů v oblasti počítačové bezpečnosti i prevence před počítačovou kriminalitou. Techniky, které dovolují určit polohu libovolného uzlu v síti Internet kdekoliv na Zemi, se označují jako **geolokace**. Ovšem pod tento pojem lze zařadit také určování polohy objektů nacházejících se v jiných komunikačních sítích (např. mobilních nebo obecně radiových). Navíc techniky pro určování polohy objektů, tzv. **geolokační techniky**, nemusí být nutně aplikovány výhradně na Zemi. Jejich případnou modifikací a zavedením odpovídajícího souřadného systému by bylo možné určovat polohu objektů rovněž umístěných mimo Zemi, např. na Měsíci, nebo jinde ve vesmíru.

1.2 Význam geolokování

Jak již bylo zmíněno v předchozí sekci, využívá se geolokace pro určování poloh uzlů v síti Internet pro odhalení místa, odkud je veden kybernetický útok, odkud jsou rozesílány spamové nebo phishingové emaily, nebo obecně pro určení místa, kde se nachází uzel (tj. osobní počítač, firemní počítač, server, …), který je využíván pro trestnou činnost. Znalost přesné polohy předmětného uzlu usnadňuje orgánům činným v trestním řízení dopadení konkrétního pachatele. Tím se zároveň krátí doba příslušného nelegálního jednání a snižují se případné škody, které tímto jednáním vznikají.

Geolokace se zdaleka nevyužívá pouze jako nástroj pro eliminaci kriminality. Dnes představuje základní techniku pro široce rozšířené uživatelské aplikace a služby. Jde např. o osobní navigace, o navigace při zajišťování logistických procesů, o systémy pro monitorování pohybu dopravních prostředků (využití lze nalézt opět např. v oblasti prevence kriminality pro vypátrání odcizeného vozidla), dále o systémy pro monitorování pohybu osob (rovněž jako prostředek pro pátrání po pohřešovaných osobách, jako prostředek pro kontrolu pohybu osob v domácím vězení apod.) nebo o systémy pro monitorování pohybu zvířat. Geolokace je dokonce kritickou a navíc jedinou přesnou technikou pro orientaci lodí a letadel.

Přínosy geolokace se nově začínají využívat také v oblasti bankovních služeb pro zvýšení bezpečnosti internetového bankovnictví [1]. Jedním z možných řešení je porovnávání aktuálního umístění klienta (stát nebo konkrétní město), resp. počítače, ze kterého se připojuje ke službě internetového bankovnictví, s domácí adresou jeho banky nebo se seznamem míst, ze kterých se klient obvykle připojuje. Je-li jeho pozice nová, bankovní aplikaci neznámá, může být pro přístup klienta k jeho účtu vyžadováno dodatečné ověření identity, čímž se eliminují škody způsobené zcizením přístupového jména a hesla, ke kterému obvykle dochází v jiné zemi. Geolokace při uvedeném využití může sloužit zároveň jako prevence před phishingovými útoky, nyní v obrácené logice, kdy se porovnává umístění podvodného bankovního serveru se skutečnou polohou (lokací) banky [1].

Hlavní přínos geolokace, jak je vidět i z výše uvedených příkladů jejího využití, je zajištění resp. zvýšení bezpečnosti životního prostředí lidské společnosti a zjednodušení jejích činností resp. zvýšení jejího blaha.

Existuje mnoho různých geolokačních technik, které se liší

- zvoleným způsobem, jakým určují výslednou polohu daného objektu,
- omezením na okolní prostředí, ve kterém jsou uplatnitelné,
- přesností, s jakou určí polohu objektu od jeho skutečné polohy,
- náchylností na výkyvy vlastností okolního prostředí,

– rychlostí, za jaký časový interval určí výslednou polohu objektu.

V dalších částech tohoto článku jsou nejprve podrobně vysvětleny vybrané geolokační techniky, následně jsou uvedeny existující standardy a protokoly využívané při geolokaci a nakonec jsou prezentovány výsledky výzkumu z oblasti geolokace v síti Internet na Fakultě dopravní ČVUT v Praze.

2 Geolokační techniky

Geolokační techniky lze dělit podle různých hledisek, které vychází zejména z jejich odlišností uvedených v předchozí kapitole. Pro účely tohoto textu je zavedeno rozdělení na dvě základní třídy podle způsobu, jakým jsou získávána data pro určení polohy zkoumaného objektu:

- **aktivní techniky** zahrnují geolokační techniky, které pro určení polohy objektu potřebují do svého okolního prostředí vyslat iniciální, dotazovací nebo testovací data. Uvedeným způsobem se příslušný objekt, který předmětnou techniku implementuje, stává pro okolní prostředí viditelným, což bývá zejména v oblasti bezpečnosti záporná nebo dokonce nevyhovující vlastnost,
- **pasivní techniky** naopak představují geolokační techniky, které polohu cílového objektu určují výhradně z informací dostupných v bezprostředním okolí objektu, který uvedenou techniku implementuje (předmětná data mohou být dokonce na uvedený objekt přímo zasílána). Potřebných dat pro určení polohy cílového objektu proto nemusí být vždy dostatek, na druhou stranu o sobě objekty nedávají vědět a mohou být pro své okolí neviditelné.

Dílčím dělením resp. dělením z jiného pohledu může být použitá technologie pro zjišťování dat potřebných pro výpočet polohy objektu. Geolokační techniky lze rozdělit na:

- **rádiové**. Sem patří všechny technologie, které potřebná data získávají výlučně prostřednictvím radiových signálů šířených obvykle předem známou rychlostí, tedy ve vzduchu, ve vodě, ve vakuu apod., a obvykle přímo a nejkratší možnou cestou mezi dvěma koncovými body. Patří sem technologie satelitní, mobilní, založené na WiFi apod.,
- **síťové**, kam lze zařadit technologie využívající různá síťová řešení (nejen síť Internet). Na rozdíl od radiových technologií je specifickou vlastností síťových technologií vysoká volatilita rychlostí přenosu dat, různá délka cest a existence disjunktních cest mezi koncovými body. Vlivem uvedených faktorů nedosahují síťové technologie tak vysoké přesnosti jako technologie rádiové,
- **databázové** sice využívají síťových infrastruktur, ale informace získávají z veřejných nebo privátních databází a nerealizují žádná vlastní měření. Jejich implementace je proto nejjednodušší, ale výsledky mohou být nejméně přesné.

2.1 Triangulace a trilaterace

Triangulace a trilaterace nejsou samy o sobě funkčními geolokačními technikami. Představují ovšem klíčové algoritmy, které využívá celá řada geolokačních technik, zejména těch, které patří do třídy aktivních, a proto je vhodné, aby byly uvedeny ještě předtím, než bude vysvětlena podstata konkrétních technik.

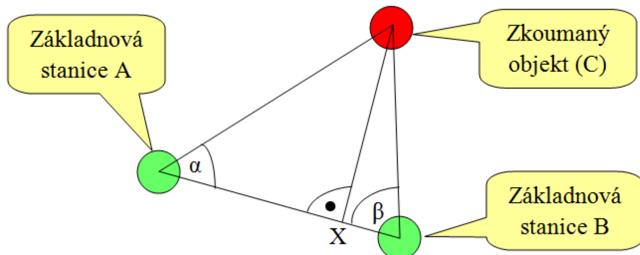
Cílem obou metod je pomocí tzv. **základnových stanic**, někdy označovaných jako sondy (pojmenování se liší v závislosti na oboru, v němž se určuje poloha zkoumaného

objektu), naměřit data potřebná k určení výsledné polohy příslušného objektu. Předpokládá se, že základnová stanice (pro jednoduchost se může jednat např. o počítač) umí vysílat signál směrem k objektu nebo opačně, že umí přijímat signál od vysílajícího objektu a zjistit všechny potřebné informace o příslušném signálu (jde zejména o změření úhlu, pod kterým je signál vysílán resp. přijímán a o určení doby přenosu signálu mezi základnovou stanicí a zkoumaným objektem, eventuelně o určení vzdálenosti základnové stanice a objektu, lze-li tento výpočet provést přímo na základnové stanici).

Triangulace

Triangulace potřebuje pro určení polohy cíle minimálně dvě základnové stanice, které se ve stejném časovém okamžiku nacházejí na dvou různých místech a v jejichž dosahu se zkoumaný objekt nachází [11]. Ve zkoumaném případě uvažujeme pouze dvojrozměrný prostor, tj. např. situaci, kdy jsou obě základnové stanice i zkoumaný objekt umístěné na povrchu Země a ve stejné nadmořské výšce.

Důležitým údajem pro určení polohy objektu je úhel, pod kterým přichází signál z objektu k základnové stanici. Je zřejmé, že pokud by předmětný objekt žádný signál nevysílal nebo byl mimo dosah základnové stanice, nebude triangulace fungovat. Obě základnové stanice a zkoumaný objekt vytváří pomyslný trojúhelník, v němž jsou známé dva úhly příchozího signálu od objektu k základnovým stanicím (úhly mohou být měřeny mezi drahou příchozího signálu od objektu a pomyslnou spojnicí obou základnových stanic, eventuelně mohou být měřeny ve směru hodinových ručiček mezi příchozím signálem a směrovou přímkou ukazující na sever [11]). Uvedená situace je znázorněna na obrázku 1, kde se jedná o úhly α a β.



Obr.1. Triangulace

Protože triangulace vychází z trigonometrie, je nutným předpokladem pro určení polohy objektu znalost vzdálenosti obou základnových stanic a znalost přesného umístění alespoň jedné z nich. Do pomyslného trojúhelníku se na spojnici obou stanic zanese kolmice ze zkoumaného objektu, její patu označme jako X. Využitím základních trigonometrických pravidel (sinova věta pro stranu *a=BC* a *b=AC* a součet vnitřních úhlů trojúhelníku (1)) a pravidel analytické geometrie (pro souřadnice bodů (2) vektor přímky AB a souřadnice bodu X (3), rovnice přímky XC pro určení vztahu c_1 a c_2 a rovnice pro určení souřadnic bodu C pomocí úhlu β (4)) lze při znalosti polohy základny A a B dopočítat polohu zkoumaného objektu C.

$$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b}; \quad \alpha + \beta + \gamma = 180° \tag{1}$$

$$A = [a_1; a_2]; B = [b_1; b_2]; C = [c_1; c_2]; X = [x_1; x_2] \tag{2}$$

$$\overrightarrow{AB} = (b_1 - a_1, b_2 - a_2); \quad X = A + \overrightarrow{AB} * \frac{|AX|}{|AB|} \tag{3}$$

$$o:(b_1 - a_1)x + (b_2 - a_2)y + k = 0; \quad \frac{\overrightarrow{BC} * \overrightarrow{BA}}{|\overrightarrow{BC}| * |\overrightarrow{BA}|} = \cos\beta \tag{4}$$

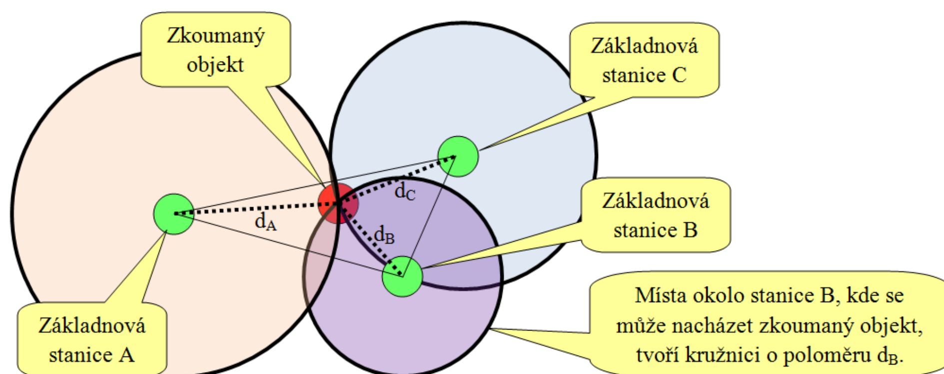
Trilaterace

Při využití techniky označované jako trilaterace je pro výpočet polohy nutná existence alespoň tří základnových stanic, které jsou ve stejný čas umístěny na různých místech ve zvoleném prostoru tak, že se zkoumaný objekt nachází v dosahu všech tří stanic [11].

Na rozdíl od triangulace není nutné znát úhly, pod jakými přichází signál od objektu k základnovým stanicím. Naopak je potřeba, aby každá stanice znala přesnou vzdálenost ke zkoumanému objektu, nebo byla schopna tuto vzdálenost určit. Jak bude v dalších částech tohoto textu uvedeno, výpočet vzdálenosti mezi základnovými stanicemi a objektem může provést zvolený centrální systém na základě znalosti dat ode všech základnových stanic (jde např. o situace, kdy základnové stanice dokážou změřit pouze čas přenosu signálu k objektu, a centrální systém na základě znalosti širších souvislostí dovede transformovat uvedený čas na skutečnou vzdálenost).

Po provedení měření vznikne kolem každé základnové stanice pomyslná kružnice o poloměru rovném naměřené hodnotě (zatím se opět předpokládá dvojrozměrný prostor), která určuje místa v jejím okolí, kde by se mohl předmětný objekt nacházet (přesné místo není stanici známo, protože při trilateraci nedochází ke zjišťování úhlu, tj. směru, odkud signál přichází).

Zmíněné pomyslné kružnice vzniknou po provedení všech měření, která jsou platná pro daný časový okamžik, okolo každé základnové stanice. V jejich společném průsečíku leží zkoumaný objekt [11]. Jeho přesná poloha se nechá opět určit pomocí vzorců analytické geometrie jen na základě znalosti přesné polohy základnových stanic a vzdáleností d_A, d_B a d_C jednotlivých stanic od objektu obdobně jako v případě triangulace, jak naznačuje obrázek 2. Při trilateraci jsou známy rozměry všech stran trojúhelníku tvořeného třemi základnovými stanicemi.



Obr.2. Trilaterace

Uvažuje-li se trojrozměrný prostor, pak okolo každé základnové stanice vzniká pomyslná koule s daným poloměrem d_A, d_B nebo d_C, v jejímž plášti se nachází zkoumaný

objekt. Jde o obecný případ uvedené metody. Výsledná poloha hledaného objektu leží v průniku plášťů všech tří koulí. Výpočet může být prováděn postupně, tj. nejprve se určí kružnice (eventuelně jediný bod) jako výsledek průniku plášťů dvou koulí, poté se vypočítá průnik uvedené kružnice s pláštěm třetí koule. Výsledkem jsou v obecném případě dva body, na rozdíl od případu dvourozměrného prostoru uvažovaného výše. V uvedeném případě je nutné buď zvolenou heuristikou vybrat správný bod ze dvou možných, nebo pro správný výběr určit průnik dvou získaných bodů s povrchem koule se středem ve čtvrté základnové stanici.

V obecném případě je potřeba určit průsečík povrchů n koulí se středy v bodech $a_j \in R^n$ a poloměry d_j pro j=1,2,…,n [3]. Řešením výše popsaného problému je nalezení vektoru $x \in R^n$ splňujícím soustavu n-nelineárních rovnic (5) nebo ekvivalentně (6) [3].

$$\left\| x - a_j \right\|_2^2 = d_j^2 \quad \text{j=1,2,...,n} \tag{5}$$

$$x^T x - 2x^T a_j + a_j^T a_j = d_j^2 \quad \text{j=1,2,...,n} \tag{6}$$

Uvedené rovnice je možné řešit gaussovou eliminací nebo ortogonální dekompozicí [3]. Při **gaussově eliminaci** se vytvoří matice $A \in R^{n \times n}$ jejíž sloupce tvoří vektory a_j pro j=1,2,…,n. Rovnice (6) se poté přepíše do tvaru (7), kde platí vztahy (8). Z rovnice (7) následně dostaneme rovnici (9), kde platí (10). Dosazením rovnice (9) do rovnice pro r v (8) dostáváme kvadratickou rovnici, z níž lze vyjádřit r (11) [3]. Hledané řešení lze nyní nalézt výpočtem pomocí rovnice (9).

$$a_j^T x = \frac{r + b_j}{2} \quad \text{j=1,2,...,n} \tag{7}$$

$$r = x^T x, \quad b_j = a_j^T a_j - d_j^2 \quad \text{pro j=1,2,...,n} \tag{8}$$

$$x = \frac{ru + v}{2} \tag{9}$$

$$u = A^{-T} e, \quad v = A^{-T} b, \quad e \in R^n, \quad e = [1,1,...,1]^T \tag{10}$$

$$r = \frac{2 - u^T v \pm \sqrt{(2 - u^T v)^2 - (u^T u)(v^T v)}}{u^T u} \tag{11}$$

Ortogonální dekompozice zde nebude blíže popsána, k dispozici je v [3].

Výše byly vysvětleny principy, které se využívají ve většině aktivních geolokacích. V následujících sekcích jsou představeny vybrané geolokační techniky. U každé je vysvětlen princip jejího fungování, zařazení do výše uvedených tříd a dále jsou zde uvedeny hlavní výhody nebo nevýhody zvoleného přístupu a možnosti jeho uplatnění.

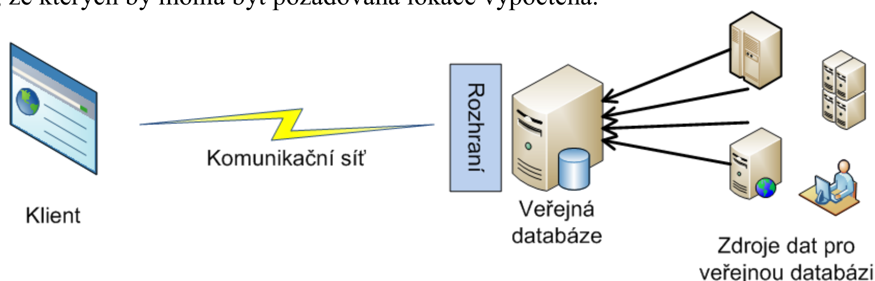
Pozn.: V případě, že se výsledný průsečík určuje z více než tří kružnic (nebo povrchů koulí), jedná se o multilateraci. V dalším textu bude pro neznámý počet základnových stanic použit pouze výraz trilaterace.

2.2 Veřejné databáze

Veřejně přístupné databáze představují speciální možnost, jak zjistit konkrétní polohu zkoumaného objektu. Zásadní odlišností od ostatních geolokačních technik je skutečnost, že geolokace využívající data uložená ve veřejných databázích neuskutečňuje žádné vlastní měření. Naopak využívá měření, která zajistil jiný subjekt a veřejně je zpřístupnil. Většina veřejných databází čerpá zdroje dat z vlastních měření založených jak na aktivních a pasivních technikách tak také z dalších veřejných databází. Potřebné informace mohou být vkládány rovněž manuálně buď na základě požadavků klienta vlastnícího příslušný objekt (např. IP uzel) nebo v případě, že není možné zajistit automatizovanou aktualizaci databáze.

Geolokace využívající veřejné geolokační databáze zašle vybranému databázovému serveru dotaz na polohu zvoleného objektu. Příslušný objekt musí být jednoznačně identifikován. Protože se tímto postupem určuje zejména poloha elektronických zařízení s přidělenou IP adresou, je předmětným identifikátorem zmíněná IP adresa sledovaného objektu. Databázový server mívá obvykle definováno rozhraní pro dotazy i jejich formát. Schéma právě popisované geolokační techniky zachycuje obrázek 3.

Zřejmou výhodou popsané techniky je její snadná implementace na straně klienta, protože není nutné provádět jakékoli výpočty ani budovat rozsáhlou síť měřících základnových stanic. Naopak uvedená technika bude selhávat, pokud bude požadovaný databázový server nedostupný, což může být způsobeno jeho selháním nebo selháním infrastruktury (sítě Internet) mezi klientem a serverem. Navíc databázové servery neobsahují potřebná data k tomu, aby mohl být na straně klienta lokalizován jakýkoliv objekt s IP adresou. Databáze buď neobsahuje lokaci zvoleného objektu, nebo neobsahuje data, ze kterých by mohla být požadovaná lokace vypočtena.



Obr.3. Komunikační schéma geolokace založené na veřejných databázích.

Geolokace založené výhradně na veřejně dostupných databázích bývají nabízeny obvykle formou webových služeb jako samostatné aplikace, které na základě poskytnutých dat vyhledají zaznamenanou polohu zkoumaného objektu (např. Google Geolocation), nebo ve formě doplňků webových prohlížečů, které určují např. příslušnost klientské stanice nebo webového serveru do konkrétní země (např. Geolocation Add-on pro Firefox).

V následujících paragrafech je podrobněji vysvětlen princip fungování vybraných veřejných databází využitelných pro geolokaci.

DNS

Uzlu, který má přidělenou IP adresu a je pro něj ve jmenné službě DNS vytvořen blok záznamů, může být přiřazen speciální DNS záznam obsahující jeho přesnou polohu. Jde o tzv. LOC záznam (*LOC Resource Record*), který obsahuje zeměpisnou šířku a délku daného uzlu a jeho nadmořskou výšku [5]. Pro zjištění uvedeného záznamu a tedy případné polohy zkoumaného uzlu stačí příslušnému DNS serveru zaslat DNS dotaz na jeho LOC záznam (formát akceptovaný rozhraním DNS serveru je určen DNS protokolem).

DNS LOC záznamy jsou nepovinné a do DNS se zanáší ručně, resp. musí být vytvořeny seznamy lokací zvolených uzlů. To jsou důvody, proč jsou LOC záznamy vyplněné jen pro některé uzly. Zároveň neexistuje žádný mechanismus, který by ověřoval správnost vložených údajů a je proto možné vkládat úmyslně nesprávnou polohu. Pokud by ale u cílového uzlu byly předmětné informace vyplněny správně, poskytovaly by neocenitelný údaj mimo jiné o jeho nadmořské výšce, kterou žádná jiná geolokační technika v prostředí IP sítí nedovede určit.

Nicméně v DNS lze najít i přesnou adresu a kontakt na správce příslušných serverů nebo domén, které mohou lokalizaci zásadně usnadnit. Opět ale i zde platí, že zmíněné údaje nemusí být aktuální nebo pravdivé, protože je zadávají přímo vlastníci daných serverů.

Maxmind GeoLiteCity, iPlane, IP2Country

Ostatní databáze se zaměřují na zjišťování konkrétních údajů o jednotlivých IP uzlech nebo celých sítí IP uzlů. Příkladem jsou např. databáze *Maxmind GeoLiteCity*, *iPlane* nebo *IP2Country* které obsahují relace mezi jednotlivými IP sítěmi a jejich příslušností do daného autonomního systému, do konkrétní země nebo města. Pomocí těchto dat lze určovat propojení mezi jednotlivými sítěmi (což jsou důležitá data pro aktivní geolokaci založenou na topologii sítě TBG, viz kapitola 2.5), umístění zkoumaného uzlu v dané zemi nebo městě apod. Z dalších databází lze zmínit např. *GeoTargetIP* nebo *GeoIP*.

Databáze WiFi

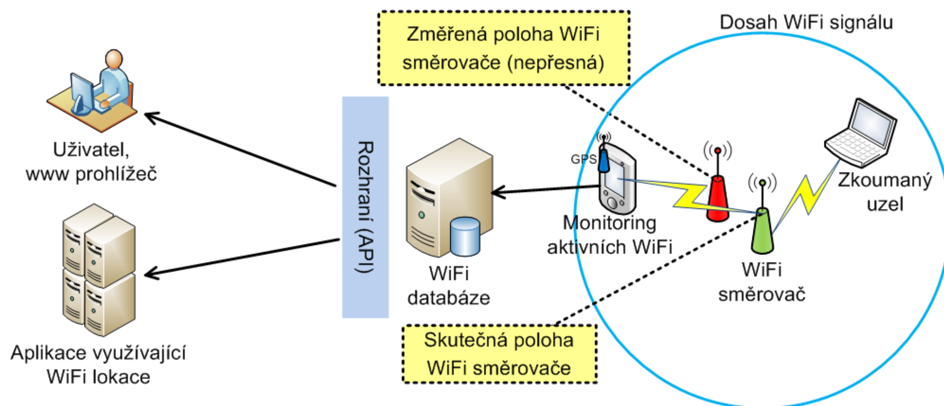
Geolokace pomocí WiFi uzlů je záležitostí několika málo posledních let. Princip spočívá v tom, že se nejprve na vybraném území zmapují aktivní WiFi směrovače, tj. takové, které v době měření vysílají do svého okolí nějaká data. Při uvedeném měření se zjišťuje zejména jejich celosvětově unikátní identifikátor, tzv. BSSID, což je obdoba MAC adresy u IP uzlů. Zároveň s tím si příslušný monitorovací systém zaznamenává i aktuální polohu, typicky pomocí geolokace GPS. Uvedeným postupem lze na daném území určit polohy WiFi směrovačů s přesností na stovky metrů (v závislosti na dosahu signálu WiFi směrovače) [4]. Pokud je současně zaznamenávána také síla jejich signálů (kterou lze chápat jako vzdálenost monitorovacího zařízení od WiFi směrovače), lze na základě více různých měření (měření stejného WiFi směrovače z různých míst) určit pomocí trilaterace jejich přesnější polohu.

Výše popsaným způsobem se připraví databáze WiFi směrovačů a jejich konkrétního umístění. Jakmile se poté jakýkoliv elektronický uzel (počítač, mobilní telefon apod.) připojí k WiFi směrovači, který je k dispozici v předmětné databázi, vystupuje navenek pod BSSID daného směrovače a jediným dotazem do WiFi databáze lze pomocí BSSID vyhledat jeho polohu. Protože se zkoumaný elektronický uzel musí nacházet v dosahu příslušného směrovače, jinak by skrze něj nemohl přistupovat ke komunikační síti (např. k internetu), známe zároveň jeho polohu s přesností na stovky metrů (opět v závislosti na dosahu signálu WiFi směrovače) [4]. Celou situaci zachycuje obrázek 4.

Výhodou geolokace pomocí databáze WiFi je její poměrně vysoká přesnost. Selhává naopak v případech, kdy příslušný WiFi směrovač není zanesen v dané WiFi databázi nebo kdy dojde ke změně jeho lokace a uvedená změna ještě není v databázi zanesena. Příprava WiFi databáze by se tedy dala zařadit mezi aktivní techniky, kdežto její využití je ryze databázová záležitost.

Příkladem WiFi databáze je např. databáze společnosti Google, která při realizaci svého projektu *Street View* kromě obrazového záznamu ulic ve městech zároveň mapuje aktivní WiFi směrovače a zaznamenává si jejich polohu pomocí GPS [8]. Pomocí speciálního

rozhraní (*Google Geolocation API*) lze na servery Google zaslat dotaz obsahující BSSID příslušného směrovače a odezvou je jeho pravděpodobná poloha. Ta je platná pro čas měření – není totiž zaručeno, že směrovač nebyl od dané doby přesunut.

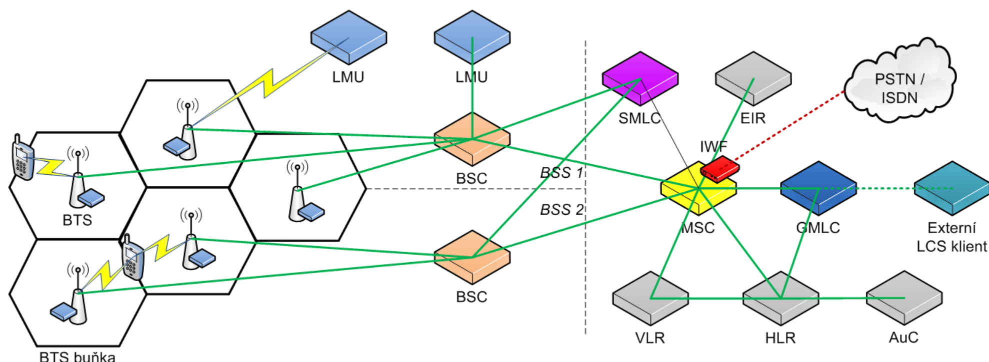


Obr.4. Geolokace pomocí WiFi směrovačů.

2.3 Mobilní (GSM) sítě

V následující sekci budou uvedeny možnosti určování polohy objektů v mobilních sítích. Protože se pokrytí Země GSM signálem neustále zvyšuje a jsou nasazovány stále nové mobilní technologie a služby (např. třetí generace mobilních telefonů, 3G), nachází v určité míře svoje uplatnění také geolokace založená na mobilních sítích [12]. Přesnost GSM geolokace a možnosti využití jejích konkrétních metod závisí na aktuálním pokrytí vybraného území základnovými stanicemi GSM sítě, tzv. BTS (*Base Transceiver Station*) resp. na charakteru jejich buněk (tzv. *BTS cell*), a na informacích, které má příslušná mobilní síť k dispozici [12]. Lokalizace pomocí GSM sítí je ve většině případů aktivní technikou, kdy probíhá měření časů přenosu signálů mezi mobilním zařízením a základnovou stanicí.

Mobilní síť může být buď standardní, nebo doplněna o lokalizační GSM rozhraní. V závislosti na jeho dostupnosti je možné volit v GSM síti příslušné metody geolokace. Zjednodušený příklad standardu GSM sítě, která uvedeným rozhraním disponuje, je uveden na obrázku 5, který byl převzat z [12].



Obr.5. Lokalizační subsystém v GSM síti.

Schéma GSM sítě s lokačním rozhraním tvoří dva základní subsystémy. Subsystém základnových stanic (BSS) se základnovými stanicemi (BTS), ke kterým se připojují mobilní zařízení, a jejich řízení (BSC, *Base Station Controller*). Druhý je subsystém sítě, jehož základ tvoří mobilní ústředny příslušného operátora (MSC, *Mobile services Switching Centre*) s registry o všech účastnících mobilního operátora (HLR) a o účastnících pohybujících se aktuálně v okolí dané ústředny (VLR), dále má k dispozici autentizační centrum (AuC) a registr mobilních zařízení (EIR) [12].

Dále popisované rozhraní využívají např. geolokační aplikace v rámci nebo mimo příslušné mobilní sítě (Externí LCS klient) pro určení polohy cílových mobilních zařízení. Pro její zjištění zašle LCS klient příslušný požadavek bráně mobilního geolokačního centra (GMLC). Potřebné lokační informace jsou následně zjištěny z registrů HLR a VLR ve spolupráci s MSC.

Pro určení konkrétní polohy mobilního zařízení jsou základnové stanice GSM sítě s lokačním rozhraním vybaveny lokačními jednotkami (LMU), které zajišťují potřebná měření pro konkrétní geolokační techniky. Předmětným měřením je především určení doby přenosu signálu z mobilního zařízení na základnovou stanici, čímž lze určit jejich vzájemnou vzdálenost. Naměřená data z jednotlivých LMU jsou zpracovávána v jednotce SMLC, která vyhodnocuje polohu cílového zařízení [2] a [8]. Obdobné schéma s dílčími změnami platí rovněž pro GPRS a UMTS sítě.

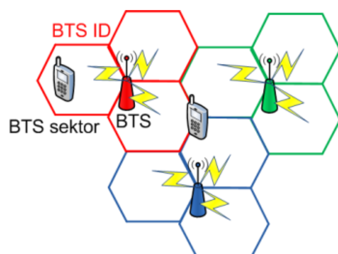
Dále jsou rozepsány vybrané geolokační techniky pro mobilní sítě převzaté z [11] a [12].

Cell ID

Při určování polohy mobilního zařízení pouze na základě informace, ke které základnové stanici mobilní sítě je mobilní zřízení připojeno (tzv. *Cell Identification*, nebo zkráceně *CI*), je nutné znát polohu základnové stanice a její sektor, v němž se zařízení nachází. Jak je vidět na obrázku 6, základnová stanice může díky svým směrovým anténám rozdělit prostor, který pokrývá, na více částí, tzv. sektorů. Potom lze určit konkrétní sektor, v němž se zařízení nachází, namísto celého prostoru pokrytého základnovou stanicí.

Daná technika funguje v sítích GSM, GPRS i UMTS. V základní podobě je značně nepřesná. Odchylka výjimečně přesahuje i 35 km v závislosti na velikosti buňky resp. konkrétního sektoru základnové stanice.

Přesnost se zvyšuje kombinací s další technikou, kterou je měření doby přenosu signálu mezi mobilním zařízením a základnovými stanicemi (*Round Trip Delay*).



Obr.6. Sektory v mobilní síti.

Round Trip Delay (RTD)

Podstatou techniky RTD je změření doby přenosu datových zpráv z mobilního zařízení na základnovou stanici a zpět. Pro úspěšné určení lokace zařízení je dále potřeba, aby v okolí mobilního zařízení byly dosažitelné minimálně tři základnové stanice, a je nutná znalost

jejich přesné polohy. Za pomoci trilaterace, kde zkoumaným objektem je zaměřované mobilní zařízení, se určí jeho výsledná poloha.

V GSM síti je vzorkování prováděno po 1.85 μs což odpovídá vzdálenosti 550 m, která představuje minimální možnou chybu při výpočtu polohy uvedenou technikou.

TOA, TDOA

Technika TOA (*Time of Arrival*) je založená na určení času, kdy byl signál z mobilního zařízení doručen na základnovou stanici resp. na její lokační jednotku. V synchronních sítích je poté možné pomocí trilaterace určit polohu cílového zařízení, protože je znám čas odeslání signálu z mobilního zařízení (ten je do datové zprávy vložen jako časové razítko).

Ovšem v GSM a 3G sítích, které jsou asynchronní (mobilní síť nemá rozhraní pro sdílení času), nemají základnové stanice k dispozici přesný čas, kdy mobilní zařízení odeslalo příslušný signál, musí být synchronizovány lokační jednotky. Synchronizace proběhne výměnou lokálních časů mezi lokačními jednotkami a určení jejich rozdílu. Podle [12] lze pro výpočet polohy mobilního zařízení použít vzorec (12) pro $i=1,2,3$, kde $d_i()$ je vzdálenost i-té lokační jednotky (LMU) a mobilního zařízení (MZ), $T^i_{doruč.}$ je čas doručení signálu na LMU, $T_{odeslání}$ je čas odeslání signálu z MZ, c je rychlost světla, (x_i, y_i) jsou souřadnice LMU, (x_{MZ}, y_{MZ}) jsou neznámé souřadnice MZ a ΔT_{MZ} je neznámý rozdíl lokálních hodin mezi MZ a LMU.

$$d_i(MZ, LMU_i) = (T^i_{doruč.} - T_{odeslání}) \times c = \sqrt{(x_i - x_{MZ})^2 + (y_i - y_{MZ})^2} + c \times \Delta T_{MZ} \quad (12)$$

Výsledkem jsou tři rovnice (jde o rovnice hyperbol), jejichž vzájemným rozdílem (d_1-d_2, d_2-d_3, d_3-d_2) se eliminuje neznámá ΔT_{MZ}. Z nové soustavy se již snadno určí souřadnice (x_{MZ}, y_{MZ}) mobilního zařízení (zařízení leží v průniku hyperbol). Geolokace s uvedenými rozdíly rovnic se označuje jako TDOA (*Time Difference of Arrival*) a je vhodná v těch mobilních sítích, kde není možná časová synchronizace jejích prvků.

O-TDOA, U-TDOA

Geolokační technika O-TDOA (*Observed Time Difference of Arrival*) je založena na TDOA. Polohu mobilního zařízení určuje lokační server z časů doručení UMTS rámců z alespoň tří základnových stanic (BTS) na mobilní zařízení. Zařízení leží opět v průniku hyperbol definovaných metodou O-TDOA. Uvedená technika proto funguje výhradně v UMTS sítích.

Naopak U-TDOA technika (*Uplink Time Difference of Arrival*) určuje polohu mobilního zařízení podle časů doručení signálu z daného zařízení na základnové stanice (BTS) resp. její lokační jednotky. Není proto potřeba znát čas jeho odeslání. Lokační server mobilní sítě poté pomocí trilaterace určí výslednou polohu mobilního zařízení.

E-OTD

Poslední vybranou technikou geolokace v mobilních sítích je E-OTD (*Enhanced Observed Time Difference*). Mobilní zařízení odešle signál všem dostupným základnovým stanicím (BTS) a ta nejbližší odešle signál zpět danému zařízení. Externí server následně zanalyzuje dobu mezi odesláním a příjmem příslušných signálů a určí polohu zařízení.

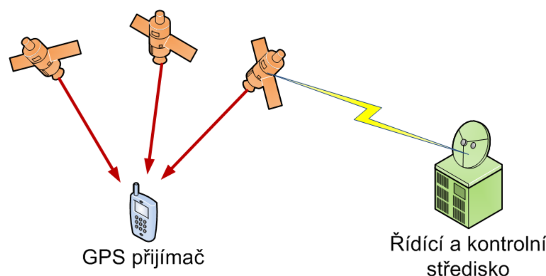
Uvedená metoda je mnohem přesnější než předchozí zde uvedené, průměrná chyba dosahuje 50 až 125 metrů. Na druhou stranu vyžaduje přenos většího množství dat.

2.4 Satelitní systémy

Satelitní systémy tvoří samostatnou skupinu bezdrátových technologií pro určování polohy zkoumaných objektů. Zatímco předchozí technologie i geolokace v Internetu, která bude

následovat, se používají jak pro určování polohy cílového objektu třetí stranou tak i pro určení polohy objektu jím samým, dostupné satelitní technologie využívá výhradně daný objekt pro určení své vlastní polohy. Satelitní systémy kvůli své koncepci neposkytují informace o lokacích vybraných zařízení a ani nemají příslušná data k dispozici.

Obecné schéma satelitního lokačního systému je uvedeno na obrázku 7. Jednotlivé satelity vysílají v daných intervalech signál (zprávy), který obsahuje čas jeho odeslání a souřadnice příslušného satelitu v definovaném formátu. Přijímač po obdržení zpráv z alespoň tří satelitů určí jejich souřadnice, vzdálenost (podle časové značky ve zprávě) a pomocí trojrozměrné trilaterace určí průniky plášťů pomyslných koulí okolo každého satelitu. Průsečíkem je aktuální poloha přijímače. K řízení a správě satelitů je určeno řídící a kontrolní středisko, které zabezpečuje správnou funkčnost satelitů. Např. synchronizuje čas mezi jednotlivými satelity.



Obr.7. Komponenty satelitního lokačního systému.

GPS

Geolokaci technologií GPS (*Global Positioning System*) zajišťuje 24 satelitů umístěných na 6 různých oběžných drahách tak, aby z každého místa Země v každý okamžik byly viditelné alespoň 4 satelity. Oběžné dráhy satelitů jsou navrženy s periodou 11h 58 min, což při započtení rotace Země znamená, že každý satelit se nad stejným bodem Země objeví každých 23h 56 min [2].

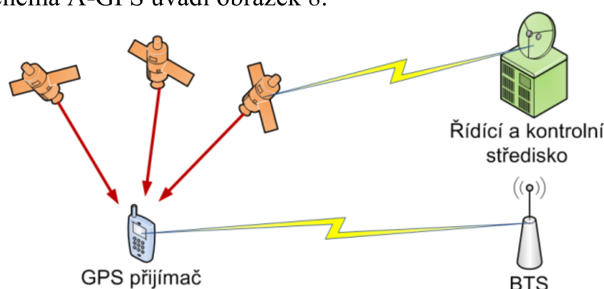
Vzhledem k rozmístění oběžných drah satelitů systému GPS bude navigace nepřesná na Zemských pólech (zejména její vertikální pozice). Kvůli nutné synchronizaci lokálních časovačů, která musí zajistit, že všechny satelity budou mít nastaven shodný čas s odchylkou max. 3 ns [11], jsou satelity vybaveny atomovými hodinami [2].

Satelity odesílají každou 1 ms unikátní C/A kód generovaný frekvencí 1.023 MHz podle stanoveného algoritmu. GPS přijímač si při zachycení příslušného C/A kódu vytváří jeho vlastní elektronickou repliku a poté ji porovnává s příchozím signálem. Z posunu obou signálů určí přijímač rozdíl v časování jeho lokálních hodin a hodin na satelitu. Z vlastní navigační zprávy, která je odesílána každých 30 sekund, zjistí parametry satelitu (např. jeho souřadnice), korekci jeho hodin, informace o ionosféře (které jsou důležité např. pro určení rychlosti přenosu signálu) apod. Na základě uvedených informací je GPS přijímač schopen spočítat čas přenosu zprávy ze satelitu a určit tím vzájemnou vzdálenost (rychlost přenosu zprávy se blíží rychlosti světla ve vakuu). Při zjištění příslušných parametrů z alespoň čtyř satelitů může přijímač pomocí trojrozměrné trilaterace vypočítat svoji polohu. [2]

V [2] jsou uvedeny konkrétní výpočty polohy přijímače včetně výpočtu odchylky jak pro běžné využití tak pro leteckou dopravu, která vyžaduje přesnější výpočty.

A-GPS

Technika asistovaného GPS (*Assisted GPS*) je kombinací lokalizace pomocí GPS a mobilních sítí. Schéma A-GPS uvádí obrázek 8.



Obr.8. Komponenty A-GPS.

Principem je zrychlit určení polohy mobilního zařízení skrze GPS tak, že mobilní síť dodá mobilnímu zařízení jí známé informace např. o dostupných satelitech, synchronizovaném GPS čase na satelitech, souřadnicích satelitů nebo korekci časů pro GPS přijímač nebo výchozí polohu (je-li v mobilní síti dostupá lokační jednotka). GPS přijímač proto již nemusí uvedené informace o stavu GPS, které získal prostřednictvím mobilní sítě, vypočítávat, čímž se výrazně zrychlí geolokace pomocí GPS. [11]

A-GPS je použitelné ve všech synchronních i asynchronních mobilních sítích. S odchylkou kolem 10ti metrů převyšuje techniky jako je Cell ID, E-OTD nebo O-TDOA.

Galileo

Připravovaný projekt Galileo Evropské vesmírné agentury bude obdobou systému GPS. Hlavní rozdíly jsou v počtu satelitů, kterých bude 30. Perioda oběhu po příslušné oběžné dráze je stanovena na 14 hodin a v každém okamžiku budou z každého místa Země viditelné minimálně 4 satelity. Rozmístěním satelitů budou rovněž pokryty polární oblasti. Dalším vylepšením oproti GPS je průměrná chyba lokace okolo 30 cm. [6]

2.5 Geolokace v Internetu

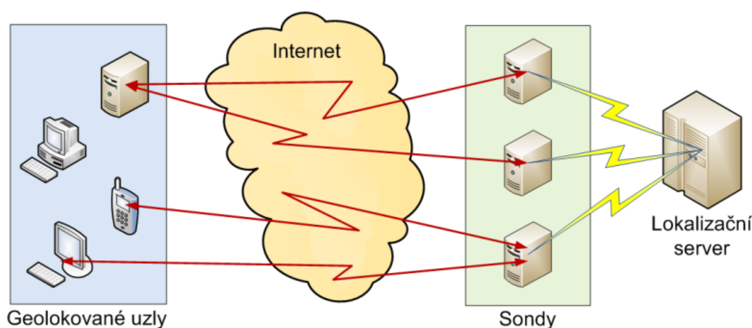
Geolokace v Internetu je specifickým případem určování polohy zkoumaných objektů. Používá se pro určení polohy elektronického uzlu, který je připojen k síti Internet a to buď přímo nebo prostřednictvím jiných uzlů nebo sítí.

Geolokaci uzlů připojených k Internetu lze rozdělit do dvou velkých skupin. Na pasivní geolokaci, což jsou techniky, kdy se nerealizuje žádné vlastní měření a pro výpočet polohy se využívají pouze data dostupná ve veřejných databázích, viz sekce 2.2. Do pasivní geolokace lze zařadit také odposlouchávání provozu v síti Internet od zkoumaného uzlu. Na jeho základě lze odhadovat jeho přibližné umístění. Lze např. určit, v blízkosti kterého směrovače se uzel nachází, což může určit polohu daného uzlu s přesností na příslušný kraj nebo zemi. Umístění příslušného směrovače již bývá obvykle snadno zjistitelné např. z veřejných databází. [14]

Druhou významnou částí technik jsou aktivní geolokační techniky v internetu, které bude blíže popisovat tato sekce. V současné době se při aktivním režimu geolokace v Internetu využívají obdobné principy jako pro geolokaci mobilních zařízení pomocí mobilních sítí nebo satelitních systémů. Obecný princip spočívá v určení doby přenosu datových zpráv v Internetu (dále jen *paketů*) mezi základnovou stanicí (které se v Internetu označují jako *sondy*) a zkoumaným objektem, kterým může být jakékoliv

elektronické zařízení, které je v Internetu jednoznačně identifikovatelné, tzn., že má přidělenou IP adresu (např. počítač, tablet, „chytrý“ mobilní telefon apod.). Základnová stanice měří dobu přenosu datové zprávy ke zkoumanému objektu a zpět od něj, která se označuje jako tzv. RTT (*Round Trip Time*) – objekt po obdržení na příslušnou zprávu *ping* reaguje odezvou, kterou zasílá odesílateli, v tomto případě tedy základnové stanici [13]. Obecné schéma systému pro geolokaci v Internetu uvádí obrázek 9.

Lokalizační server plní řídící úlohu a určuje, popřípadě identifikuje na základě dalších informací, který uzel v Internetu bude geolokován. Na základě příslušné identifikace sestaví odpovídající úlohy pro sondy, které jim definovaným způsobem předá. Podoba úloh závisí na zvolené geolokační technice, které jsou uvedeny dále. Sonda při provádění odpovídající úlohy odešle zadaným způsobem paket služby PING nebo TRACEROUTE geolokovanému uzlu, který reaguje okamžitou odezvou zpět příslušné sondě. Sonda po příjmu odezvy spočítá čas, který byl pro přenos paketu potřeba, a opět definovaným způsobem předává všechna naměřená data zpět lokalizačnímu serveru, který zpracuje data od všech sond a na jejich základě vypočítá pravděpodobnou polohu zkoumaného uzlu [13].



Obr.9. Systém pro geolokaci v Internetu

Základním předpokladem pro uvedené měření je skutečnost, že datové pakety jsou při svém přenosu skrze síť Internet předávány mezi jednotlivými směrovači na cestě mezi sondami a geolokovanými uzly se stabilním zpožděním až na přípustnou odchylku. Na základě uvedených informací a při znalosti rychlosti přenosu paketů v konkrétní části sítě Internet je možné převést dobu přenosu paketů na skutečnou vzdálenost mezi dvěma uzly (např. mezi sondou a geolokovaným uzlem). Určit rychlost přenosu paketů v příslušné části sítě lze např. vzájemným měřením doby jejich přenosu mezi dvojicemi sond, jejichž vzdálenost musí být známa, resp. ji lze vypočítat na základě znalosti jejich přesného umístění. Z uvedeného tvrzení vyplývá, že je velmi důležité vhodně rozmístit sondy tak, aby v definovaném území dostatečně pokrývaly všechny části sítě Internet.

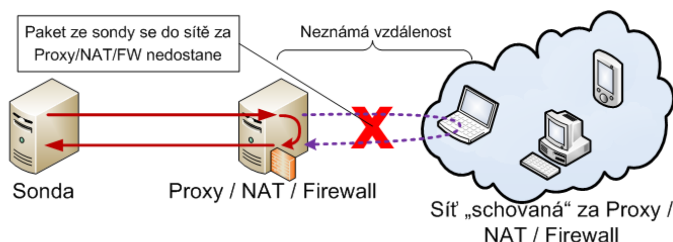
Lokalizační server musí znát přesnou polohu sond a jejich vzdálenost od zkoumaného uzlu, kterou určí např. výše uvedeným způsobem. Pomocí trilaterace ve dvourozměrném prostoru nakonec vypočítá výslednou polohu geolokovaného uzlu.

Omezení geolokace

Internet jako komunikační nástroj pro určení vzdálenosti geolokovaných uzlů od sond představuje na rozdíl od bezdrátových technologií určitá úskalí. Níže jsou vybrané nejdůležitější z nich [13] a [14].

Nadprůměrné zpoždění při přenosu dat na některých podsítích může mít nemalý vliv na přesnost geolokace. Konkrétní výsledky jsou prezentovány v kapitole 4.

Skrývání sítí je další velmi častou překážkou při geolokaci uzlů v Internetu. Proxy nebo NAT servery představují pro geolokaci koncový bod příslušné cesty mezi sondou a zkoumaným uzlem, který se ve skutečnosti nachází v síti, kterou příslušný server odděluje od zbytku Internetu. Sondy jsou v uvedeném případě schopny určit polohu uvedeného serveru. Není již ale možné určit síťovou topologii, která se za nimi skrývá, a nemohou proto ani určit vzdálenost příslušného uzlu od daného serveru. Situaci zachycuje obrázek 10. Stejná situace nastává i v případě, že předmětným serverem je firewall, který nepropustí pakety ze sondy do sítě umístěné za ním.



Obr.10. Skrytá síť za Proxy, NAT nebo Firewallem.

Specifickým případem skryté sítě je VPN, která vytváří pro sondy neviditelné komunikační spoje. Sondy z celé VPN sítě vidí pouze její přístupový bod. Zasílané pakety mohou být zasílány skrze onen přístupový bod těmito neviditelnými tunely až ke zkoumanému uzlu. Protože poloha přístupového bodu a zkoumaného uzlu se obvykle liší, určí sonda resp. lokalizační server polohu chybně.

Přidělování dočasných adres může znamenat, že stejná IP adresa může být v různých časových okamžicích přidělena různým uzlům s různou polohou. Geolokace předmětných uzlů proto musí být vázána na konkrétní časový okamžik, jinak bude chybná.

Vysoké budovy nebo jiné vedení spojů pro Internet ve vertikálním směru snižuje přesnost geolokace, protože se předpokládá pouze dvojrozměrné vedení komunikačních spojů.

V následujících sekcích jsou uvedeny základní geolokační techniky pro určování poloh uzlů v Internetu [13] a [14].

Nejkratší Ping

Určení polohy IP objektu podle nejkratší doby přenosu zprávy *ping* je nejjednodušší metodou geolokace v Internetu. Všechny sondy změří RTT ke zkoumanému cíli, na jejichž základě se určí sonda, která je k danému uzlu nejblíže. Výslednou polohou je poloha příslušné sondy. Nevýhodou uvedené techniky je její vysoká nepřesnost.

GeoPing

GeoPing je zdokonalením předchozí metody, kdy se navíc uvažuje předpoklad vycházející z experimentálních pozorování, že IP objekty, které vykazují shodné zpoždění od dané sondy, se nacházejí ve stejné geografické oblasti, tj. že jsou blízko sebe.

Kromě sond se využívají i tzv. referenční body (obdoba sond), jejichž poloha je známá, ale samy nerealizují žádná měření. Jako výsledná poloha zkoumaného objektu se označí poloha některého z referenčních bodů (nebo poloha sondy), který vykazuje nejvíce shodnou dobu odezvy jako příslušný objekt. Přesnost techniky je ovlivněna hustotou referenčních bodů a je proto opět velmi nepřesná.

Geolokace podle topologie

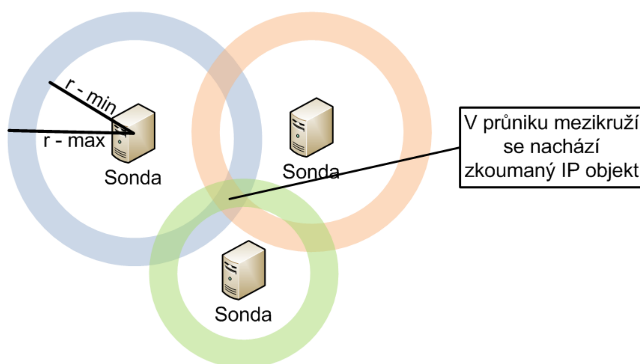
Jednou z technik využívající pro výpočet polohy IP uzlu naměřených RTT časů od všech sond je CBG (*Constraint Based Geolocation*). Časy RTT jsou lokalizačním serverem převedeny na skutečnou vzdálenost mezi sondami a cílovým uzlem, čímž se okolo sond vytvoří pomyslné kružnice o poloměru odpovídajícím danému RTT vymezující body, kde se může uzel nacházet. Pomocí trilaterace se určí jejich průsečík, který je označen jako poloha daného uzlu.

Úskalím, která z uvedené techniky plynou, jsou různé možnosti průniku zmíněných kružnic, resp. různé výsledky trilaterace. Kružnice se buď neprotnou vůbec (příslušný stav je indikován jako neúspěšná geolokace), nebo přesně v jednom bodě, nebo vymezí jistý prostor.

Rozšířením CBG je technika TBG (*Topology Based Geolocation*), která navíc určuje polohu uzlů (směrovačů) ležících na cestě mezi sondami a zkoumaným uzlem. Předmětné směrovače jsou při dalších výpočtech využívány jako referenční body, tj. uzly s již známou polohou.

Octant

Opakovaným měřením RTT mezi danou sondou a konkrétním IP objektem lze pozorovat jistou míru volatility doby přenosu paketů *ping*. Určením minimální a maximální doby RTT z opakovaných měření lze okolo každé sondy určit dvě kružnice, které určují minimální a maximální vzdálenost od sondy, kde se může geolokovaný uzel nacházet. Oproti běžné trilateraci, která se použije pro výpočet i v tomto případě, se místo kružnic okolo základnových stanic (sond) uvažují mezikruží, jak znázorňuje obrázek 11 [13]. Výsledkem výpočtu není jediný bod, ale oblast, v níž se nachází zkoumaný objekt.



Obr.11. Princip techniky Octant.

Pro převod RTT na skutečnou vzdálenost vychází Octant z měření RTT mezi sondami, jejichž vzdálenost dokáže lokalizační server určit z jejich polohy. Z daného měření je pak možné určit rychlost přenosu paktů a použít je pro převod RTT mezi sondou a objektem na skutečnou vzdálenost.

3 Protokoly a standardy geolokace

Existence standardů a protokolů pro geolokaci objektů se značně liší podle oboru resp. prostředí, v němž se lokalizace uskutečňuje. Mnohem více standardů nalezneme v mobilních sítích nebo obecně v bezdrátových sítích, které využívají výhradně

standardizovaná řešení, naopak v prostředí Internetu se jedná spíše o proprietální řešení, která jsou více či méně využívána uživateli Internetu.

U databázově orientovaných geolokačních technik se lze setkat s experimentálními protokoly. Příkladem je ***RFC 1876***, který definuje způsob ukládání lokačních informací o daném IP uzlu v síti internet, konkrétně na DNS serverech, a dotazování se na ně [5]. Obdobou je lokační protokol ***Geolocation API Network Protocol*** od společnosti Google, který definuje podobu dotazu a význam atributů v odpovědi na dotaz o poloze uzlu s danou MAC adresou. Na rozdíl od RFC 1876 neříká nic o způsobu ukládání lokalizačních údajů, protože tato část systému je ve správě zmíněné společnosti.

Mnohem obsáhlejší jsou standardy používané v bezdrátových resp. mobilních sítích, které kromě přesné podoby zasílaných datových zpráv definují také architekturu celého systému. Mezi nejpoužívanější patří standard ***GSM 3.71*** popisující začlenění lokačních služeb do GSM sítě, obdobný standard pro GPRS sítě ***GPRS 43.059*** a pro UMTS sítě ***UMTS 23.171***, ***23.271*** a ***25.305*** [12]. Uvedené standardy mimo jiné definují, že v příslušných sítích budou lokalizační data uchovávána na lokalizačních serverech (viz sekce 2.3). Z aplikačního hlediska je doplňuje protokol MLP (*Mobile Location Protocol*), který definuje rozhraní tvořené MLP servery mezi lokalizační klientskou aplikací a lokalizačním serverem libovolné sítě, např. v síti mobilní (GSM, UMTS, …). Dále definuje způsob dotazování se MLP serverů na polohu příslušného uzlu (mobilního telefonu, tabletu apod.) [9].

Zvláštní skupinou technik z pohledu standardizací jsou satelitní systémy. Jde obvykle o unikátní řešení, které mají definovanou vlastní architekturu, popřípadě i vlastní komunikační protokoly, způsob výpočtu polohy apod. Jediným široce používaným systémem je GPS, jehož architektura, protokoly a standardy pro GPS přijímače jsou uvedeny v [2]. Příklad konkrétního protokolu pro GPS přijímač uvádí [7].

V prostředí Internetu je situace ještě volnější. Standardy a protokoly, které by byly závazné pro všechny implementace spadající do stejné kategorie, zde téměř neexistují. Každé řešení určující jistým způsobem polohu zvoleného objektu si obvykle definuje vlastní datové úložiště, vlastní komunikační protokoly i originální sestavení komponent systému ve funkční celek. Přesto existují jisté snahy o vytváření široce využívaných protokolů a standardů. Příkladem je rozhraní ***W3C Geolocation API*** pro webové aplikace. To je realizováno metodami v jazyce JavaScript, které jsou přidány do libovolné webové aplikace běžící na lokalizačním serveru. Podmínkou je, aby klient využíval služeb uvedené webové aplikace pomocí kompatibilního webového prohlížeče, který dovede určit aktuální polohu zmíněné stanice (např. pomocí připojeného GPS zařízení, podle MAC adresy WiFi routeru apod.), a předmětné informace o poloze zašle jako odpověď na dotaz zpět na příslušný server [10].

4 Geolokace SuHa

V předchozích kapitolách bylo shrnuto využití různých geolokačních technik v závislosti na použitých komunikačních kanálech a vlastnostech okolního prostředí, nejběžněji používané metody pro výpočet výsledné polohy objektu a používané standardy při návrhu a vývoji příslušných systémů.

V této kapitole je představeno konkrétní řešení geolokace v prostředí Internetu, které bylo realizováno v letech 2007 až 2010 zpočátku na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze, později vývoj pokračoval na Fakultě dopravní Českého vysokého učení technického v Praze. Zmíněný projekt nese pracovní označení *SuHa*.

4.1 Komponenty systému SuHa

Geolokační systém SuHa se svým základním konceptem nijak neliší od obecného schématu uvedeným na obrázku 9.

Jádro systému tvoří lokalizační server, kde se shromažďují získaná geolokační měření z jednotlivých sond. Sondy zvolenou metodou, což je nejčastěji služba *traceroute*, určí dobu odezvy zkoumaného objektu. Server zároveň funguje jako manažer celého systému SuHa a řídí začleňování nových sond do systému a deaktivaci nefunkčních nebo nedostupných sond. Dále zajišťuje registrace klientských aplikací, které umožňují např. zadávat úlohy pro geolokování vybraného IP uzlu, volit konkrétní parametry geolokace apod. V neposlední řade, řídí přístupy uživatelů k celému systému i klientským aplikacím a spravuje jejich oprávnění.

4.2 Algoritmus geolokace SuHa

Algoritmus implementovaný v geolokaci SuHa původně vycházel z algoritmu Octant popsaným v sekci 2.5. Pro usnadnění implementace se uvažovaly pouze maximální doby odezev (tj. doby přenosu paketů mezi sondou a geolokovaným objektem), uvažovala se tedy pouze pomyslná kružnice okolo každé sondy s maximálním poloměrem. Uvedené řešení poskytovalo výsledky s poměrně velkým rozptylem (tj. určená oblast, kde by se mohl zkoumaný objekt nalézat, byť s menší pravděpodobností, byla poměrně velká, od jednotek až po desítky kilometrů) od odhadovaného místa polohy daného objektu.

V pozdější fázi byl proto algoritmus upraven tak, že se ze sondou naměřených hodnot eliminoval jistý počet minimálních a maximálních hodnot, o nichž se předpokládalo, že se jedná o výchylky od běžných časů odezev, a ze zbylých hodnot byl spočítán aritmetický průměr. Výsledkem poté již nebyla maximální odezva, ale „očekávaná průměrná“ odezva v běžném případě. Řešení v této podobě je samozřejmě náchylné na případné odchylky zpoždění v Internetu danými např. denní zátěží apod. Řešení uvedeného aspektu je uvedeno níže v podrobném popisu algoritmu geolokace SuHa, který je graficky znázorněn na obrázku 12.

Zadání úlohy geolokace sondám

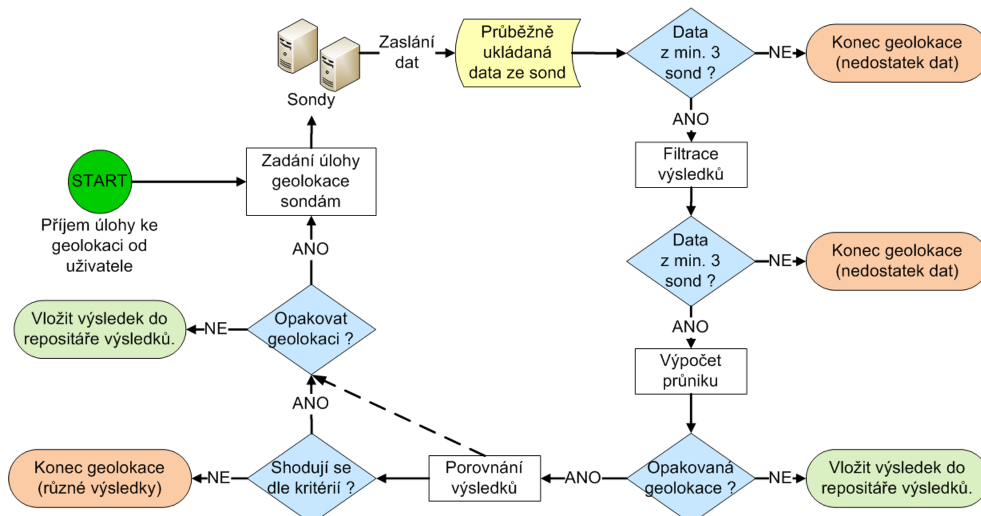
Proces zahájení geolokace zvoleného IP objektu počíná odesláním příslušného úkolu uživatelem lokalizačnímu serveru skrze klientskou aplikaci, jak bylo popsáno výše.

Lokalizační server umožňuje provést geolokaci jednou nebo opakovaně a eliminovat tím ojedinělé odchylky při měření odezev mezi sondami a zkoumanými objekty. Kromě zadání úlohy uživatelem existuje proto druhá možnost, jak zaúkolovat sondy, a tou je automatické přidělení úlohy ke geolokaci předmětného cíle samotným lokalizačním serverem. Protože je opakovaná geolokace pomalejší (čas na její provedení odpovídá součtu provedených dílčích geolokací), byť zaručuje vyšší pravděpodobnost správnosti výsledku, musí si uživatel tuto možnost zvolit v klientské aplikaci. V opačném případě bude geolokace provedena pouze jedenkrát. Princip opakované geolokace je rozveden podrobněji dále.

Příjem dat ze sond

Poté, co sondy obdrží úlohu od lokalizačního serveru, provedou pomocí služby *traceroute* měření odezvy mezi ní a zadaným IP cílem. Z naměřených hodnot odstraní zadaný počet minimálních a maximálních hodnot a ze zbylých dat vypočítají aritmetický průměr. Příslušnou hodnotu poté sondy zasílají průběžně, jakmile mají k dispozici potřebná naměřená data, zpět lokalizačnímu serveru. Server od odeslání úlohy všem sondám čeká na

doručení všech výsledků, ale maximálně do stanoveného časového limitu. Neobdrží-li do požadovaného limitu alespoň 3 hodnoty tak, aby mohla být provedena trilaterace, prohlásí server geolokaci za neúspěšnou. V opačném případě provede filtraci získaných výsledků.



Obr.12. Algoritmus geolokace SuHa.

Filtrace výsledků

Lokalizační server zadává kromě úkolů na geolokaci zkoumaného cíle (dále jen doba odezvy cíle) také úlohy, při nichž každá sonda měří dobu odezvy od všech ostatních sond v systému (dále jen doba odezev sond). Příslušné výsledky zpracuje stejným způsobem jako při geolokaci IP cíle a vypočtenou hodnotu zasílá zpět lokalizačnímu serveru.

Při filtrování časů odezev cíle uvažuje lokalizační server také časy odezev mezi sondami, které slouží k určení důvěryhodných a nedůvěryhodných sond.

Z počtu získaných výsledků (doby odezev cíle zaslaných příslušným počtem sond; ne všechny sondy stihnou odpovědět do časového limitu) se určí jistá limitní hranice, která bude rozhodující, zda bude sonda označena jako důvěryhodná nebo nikoliv. Označme uvedenou hranici jako *Lim*.

Poté se pro každou sondu X, která dodala výsledek s dobou odezvy cíle lokalizačnímu serveru, porovná obousměrně doba odezvy se všemi ostatními sondami, jednu z nich označme jako Y. Dobu odezvy ze sondy X na sondu Y označme jako $\boldsymbol{a}$. V opačném směru, tj. ze sondy Y na X, ji označme jako $\boldsymbol{b}$. Pokud není splněna podmínka (13), která byla určena experimentálně geolokací SuHa, je oběma sondám X a Y přiděleno po jednom trestném bodu. Splněním podmínky (13) je zajištěno, že obě zmíněné sondy vykazují jistou míru spolehlivosti při měření časů odezev, až na odchylku vyplívající z (13). Její nesplnění naopak naznačuje skutečnost, že jedna ze sond (eventuelně obě) je umístěna za firewallem, je realizována nad systémem způsobující zpoždění při přenosu paketů apod. Protože z uvedené dvojice měření nelze určit chybnou sondu, získávají obě dvě po jednom trestném bodu.

Jako důvěryhodné jsou označeny sondy, které získaly počet bodů menší nebo roven hodnotě *Lim*. Ostatní sondy jsou nedůvěryhodné (nadlimitní počet ostatních sond označilo předmětné sondy jako chybné) a jejich výsledky s dobami odezev cíle jsou z dalšího výpočtu geolokace odstraněny.

$$a - b \geq 5 \text{ a zároveň } 10 \times b < 9 \times a \tag{13}$$

Výpočet průniku

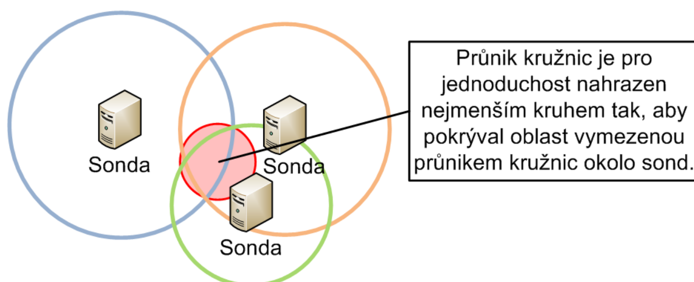
Zbývá převést doby odezev mezi sondami a cílem na skutečnou vzdálenost. Podle zmíněných odezev se určí předem daný počet sond, které jsou nejblíže zkoumanému cíli. Předmětný počet byl opět určen experimentálně. Uvedeným výběrem sond se vyberou ty, které leží v části sítě Internet s podobným zpožděním. Pro skupinu vybraných sond se na základě doby odezev mezi nimi a skutečnými vzdálenostmi mezi nimi, které jsou známé, určí rychlost šíření paketů v předmětné části Internetu. Získaná rychlost se využije pro převod doby odezvy mezi sondami a cílem na skutečnou vzdálenost.

Protože průnikem všech pomyslných kružnic vzniklých okolo sond a s poloměrem rovným určené vzdálenosti cíle od sondy (viz popis trilaterace v sekci 2.1) nemusí být určen jediný bod, ale plocha (algoritmus vychází z Octantu), určuje se výsledný průnik postupně. V iniciální fázi se vypočítá průnik dvou kružnic (pro sondy, jímž je cíl nejblíže, protože výsledné časy odezev se na lokalizačním serveru nejprve uspořádají od nejmenšího k největšímu). Nalezený průnik se nahradí kružnicí, jejíž průměr je roven vzdálenosti dvou průsečíků obou kružnic (neprotínají-li se v jediném bodě), nově vytvořená kružnice uvedené průniky protíná a její střed leží uprostřed jejich spojnice. Takto získaná kružnice nahrazuje určený průnik. Následuje iterace, při níž se určí průnik kružnice získané v předchozím kroku s kružnicí okolo další sondy v uspořádaném pořadí. Příklad finálního výsledku zachycuje obrázek 13.

Porovnání výsledků

V případě opakované geolokace zadá lokalizační server sondám ihned novou úlohu ke geolokaci téhož cíle, pokud je zatím k dispozici pouze jediný výsledek. Jsou-li výsledky minimálně dva, tj. geolokace téhož IP uzlu proběhla minimálně dvakrát, pak se porovnávají odhadované (geolokované) pozice zkoumaného objektu i poloměry kruhů od těchto pozic (viz obrázek 13).

Je-li rozdíl výsledků v limitech stanovených při zadání úlohy uživatelem nebo defaultní, pak se pokračuje až do dosažení stanoveného počtu opakování. Jako výsledek je označen ten, který dosáhl nejmenšího průměru zmíněného kruhu.

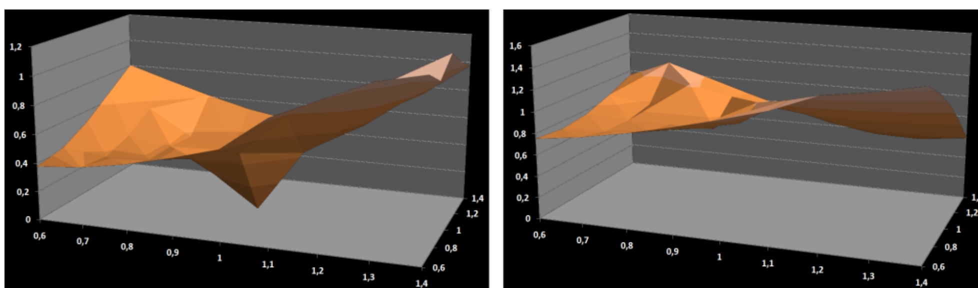


Obr.13. Výsledek geolokace SuHa.

4.3 Chyby v měření

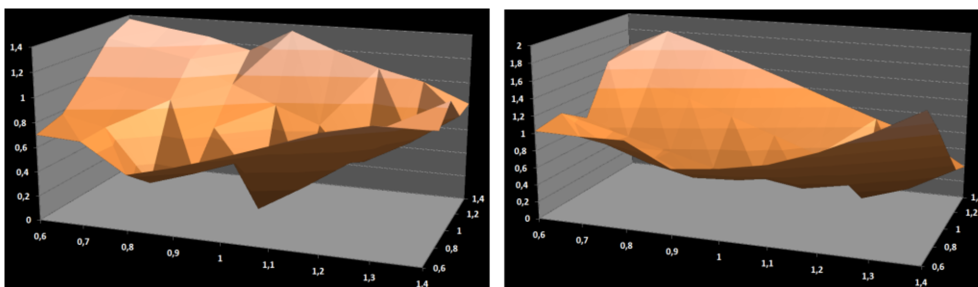
Klíčovým faktorem pro přesné určování polohy IP uzlů algoritmem SuHa jsou přesné výsledky měření získané ze sond. Filtrací výsledků z nedůvěryhodných sond se podařilo eliminovat naměřené hodnoty, u kterých je pravděpodobnější, než u výsledků z důvěryhodných sond, že představují zkreslenou nebo nepravdivou informaci o době

zpoždění paketu mezi sondou a cílem. Nicméně např. vlivem zpoždění na Internetových linkách může nastat situace, že jedna, více nebo všechny sondy dodají lokalizačnímu serveru ne zcela korektní data. Na obrázku 14 je zobrazen výsledek jednoho z testů algoritmu SuHa, testující uvedenou skutečnost, kdy byly použity 3 sondy rozmístěné do trojúhelníku a geolokovaný objekt se nacházel uvnitř daného trojúhelníku. Osy *x* zobrazují měnící se chybu 1. sondy, která postupně dává výsledek roven 60% až 140% správné hodnoty, osa *z* zachycuje totéž pro 2. sondu. 3. sonda zde zanesena není, ale v levém grafu vrací 100% správné hodnoty a v pravém 140% (tj. delší dobu zpoždění). Osa *y* zachycuje odchylku výsledku geolokace zkoumaného objektu v % (pro ilustraci 1% odpovídá zhruba 9 km). Z grafů je vidět, že dávají-li všechny sondy stejně špatné výsledky (v pravém grafu, kdy všechny sondy vrátily 140% správné hodnoty, je vidět vpravo dole „cíp“ s hodnotou chyby 0,6%), je výsledek geolokace lepší, než v případě že chybuje jen jedna ze sond. V levém grafu je vidět, že pokud sondy vrátily správné hodnoty („cíp“ v grafu uprostřed), je chyba rovna 0% (v ostatních měřeních pro bezchybné sondy byla chyba vždy menší než 0,1%) [13].



Obr.14. Závislost chyb výsledků ze sond na chybě výsledné polohy objektu – horizontální poloha.

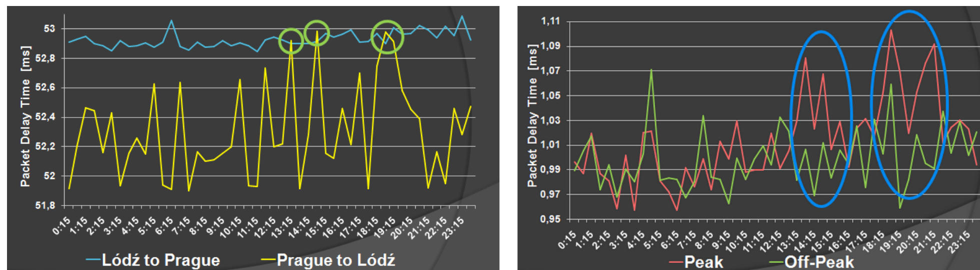
V průběhu testování algoritmu se ukázalo, že na správnosti výsledku resp. velikosti chyby má vliv vzájemné rozmístění sond a zkoumaného IP uzlu. Na obrázku 14 byly sondy rozmístěny do vrcholů rovnoramenného trojúhelníku se základnou v horizontální poloze. Na obrázku 15 je uvedeno stejné měření, ale sondy byly tentokrát umístěny do rovnoramenného trojúhelníku se základnou ve vertikální poloze. [13]



Obr.15.Závislost chyb výsledků ze sond na chybě výsledné polohy objektu – vertikální poloha.

Protože experimentální výsledky geolokace SuHa dosahovaly při měřeních na území České republiky (s rozmístěním sond na území Evropy) chyby až 80 km, byla provedena analýza časů zpoždění na Internetových linkách mezi sondami v průběhu dne. Na obrázku

16 je na levém grafu příklad uvedeného měření mezi sondami v Praze a v Polsku. Je vidět, že doba zpoždění je závislá na směru, ze kterého bylo měření provedeno. V pravém grafu je uvedeno zpoždění měřené pouze ze sondy v Praze v průběhu pracovního dne a o víkendu. Z měření vyplývá, že v pracovní den vykazuje zpoždění větších výkyvů než ve sváteční den. [13]



Obr.16. Zpoždění na Internetových linkách mezi sondami.

Z uvedených měření je patrné, že lokalizační server musí zpracovávat výsledky ze sond, které byly určeny na základě měření ve shodný časový okamžik. Aplikace výsledků z různých časových období by zvýšila chybu výsledné polohy zkoumaného IP objektu.

4.4 Metody pro zlepšení geolokace

Pro snížení chyby geolokace SuHa, která podle výše popsaného algoritmu dosahovala až 80 km na území České republiky, byla aplikována dodatečná vylepšení algoritmu. Z nich nerozsáhlejší byla dvě a jsou uvedena v následujících paragrafech.

Vektorový prostor

Myšlenka zavedení vektorového prostoru, který bude popsán dále, vycházela z předpokladu, že když sondy měří doby odezev cíle, pak mohou být předmětná měření zatížena stejnou mírou chyby jako vzájemné měření odezev mezi sondami, které se nacházejí ve stejné oblasti jako cíl. Důsledkem uvedeného předpokladu by pak byla skutečnost, že výsledná geolokace cíle bude zatížena stejnou mírou nepřesnosti jako vzájemná geolokace jednotlivých sond v příslušné oblasti.

Algoritmus SuHa byl proto na základě uvedených předpokladů upraven následovně. Po obdržení výsledků od sond, jako odezvy na úkol geolokovat zadaný IP cíl, a po provedení filtrace výsledků tak, jak byla popsána v sekci 4.2, se ze seřazených výsledků vyberou 3 s nejnižší hodnotou. Těmto hodnotám odpovídají 3 sondy, které se nacházejí nejblíže zkoumanému cíli.Každá z těchto 3 sond se následně geolokuje algoritmem SuHa s tím, že ona sama se geolokace jako sonda neúčastní a chová se jako běžný zkoumaný IP uzel. Protože jsou známy přesné polohy sond **<*X, Y*>**, porovnají se jejich přesné lokace s polohou **<*X‘, Y‘*>**, kterou určil algoritmus SuHa, a pro každou ze tří uvedených sond se určí opravný vektor V_O**=<*X-X‘, Y-Y‘*>**.

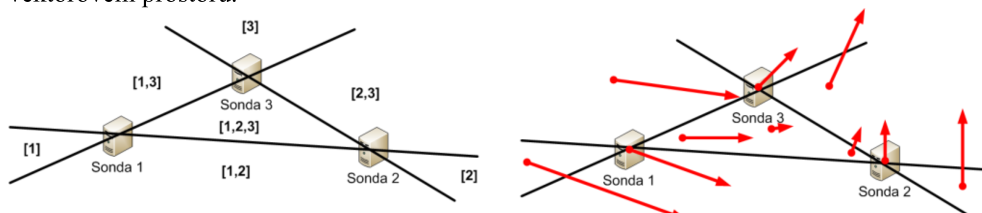
V první variantě byly vektory V_O všech 3 sond pouze vektorově sečteny a zkráceny na $^1/_3$, protože se uvažují 3 vektory. Vznikl vektor, kterým byl opraven výsledek geolokace zkoumaného IP uzlu. Opravené výsledky dosahovaly zlepšení v rozmezí <-161%, +81%>, přičemž průměr činil -5% (tj. zhoršení z 80 km na 84 km vlivem ojedinělých extrémních zhoršení výsledků) a medián představoval zlepšení o 16% (tj. z 80 km na 67 km).

Cílem druhé varianty bylo ještě více zlepšit přesnost zajištěnou v 1. variantě. Byl proto zaveden dvourozměrný vektorový prostor jako nadstavba nad dvourozměrným

geografickým prostorem, v němž byly rozmístěny sondy a IP cíle. Uvedený vektorový prostor obsahoval v každém svém bodě opravný vektor, který převáděl souřadnice vypočítané algoritmem SuHa na souřadnice skutečného geografického prostoru Země. Řešení spočívalo v tom, že zmíněné 3 nejbližší sondy vytvořily trojúhelník, jehož vrcholy odpovídaly opravným vektorům pro sondy, v těžišti trojúhelníku byl vektor nulový, tzn., že jeho souřadnice odpovídala souřadnicím skutečným. Vektory $\boldsymbol{V_N}$ v ostatních bodech uvnitř uvažovaného trojúhelníku byly určeny vztahem (14), kde V_{Oi} je opravný vektor pro i-tou sondu, d_i je vzdálenost geolokované polohy cíle a geolokované polohy i-té sondy a d_{Ti} je vzdálenost geolokované polohy i-té sondy od polohy těžiště zmíněného trojúhelníku.

$$V_N = \sum_{i=1}^{3} V_{Oi} \times \frac{d_{Ti}}{d_i} \tag{14}$$

Vně trojúhelníku byl vztah (14) uvažován pouze pro dvě nebo jen jednu sondu v závislosti na poloze cíle vůči sondám. Obrázek 17 zobrazuje ve své levé části 3 sondy tvořící trojúhelník. Uvnitř se vektor V_N určuje pomocí (14) ze všech tří sond, ale vně pouze z těch nejbližších (naznačeno čísly sond v hranatých závorkách) podle příslušnosti do daného sektoru. V pravé části obrázku je příklad vektorů V_O na sondách a vektorů V_N ve vektorovém prostoru.



Obr.17. Vektorový prostor geolokace SuHa.

Testování však ukázalo, že pokud se cíl nachází vně trojúhelníku, což nastávalo často vzhledem k rozmístění a hustotě sond při experimentálním testování, dochází ke zhoršení opravených výsledků. Zlepšení konkrétně dosahovalo intervalu <-300%, +70%>, kdy průměr činil -60% a medián -16% (tj. zhoršení z 80 km na 128 km resp. 93 km). Uvedená oprava se proto ukázala jako nepřijatelná. Vylepšit ji měl výběr N-úhelníku, který je popsán v dalším paragrafu.

N-úhelník

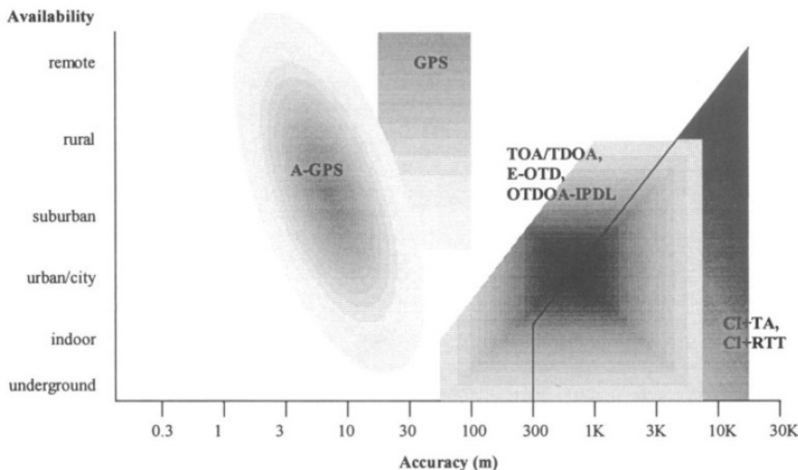
Pro zajištění lepších opravených výsledků pomocí zavedeného vektorového prostoru, byla na základě provedených testů stanovena hypotéza, že pokud budou vybrány takové sondy (na základě jejich polohy určené algoritmem SuHa), aby okolo určené polohy cíle, vytvořily N-úhelník (cíl by ležel uvnitř N-úhelníku), pro N rovno 3 nebo 4, může dojít ke zpřesnění geolokace. Testování však ukázalo, že při daném rozmístění sond, obsahuje předmětný N-úhelník ve svých vrcholech také sondy, které jsou poměrně vzdálené od polohy cíle. Uvedený efekt způsobil, že implementací daného řešení nedošlo ani ke zlepšení ani ke zhoršení výsledků získaných zavedením vektorového prostoru popsaného ve druhé variantě předchozího paragrafu.

5 Závěr

Konkrétní geolokační techniku lze zvolit podle dostupných zdrojů informací, podle vlastností okolního prostředí, ale také podle času, který je k dispozici na výpočet. Při

požadované kvalitě a množství dat lze zvolit techniku více přesnou, v opačném případě je nutné spokojit se s méně přesným určením polohy.

Obrázek 18 shrnuje přesnost resp. míru odchylky pro bezdrátové geolokační techniky (lokace v mobilních sítích a satelitních systémech) [12]. Tabulka 1 uvádí pouze míru spolehlivosti technik v prostředí Internetu [14]. Konkrétní hodnoty závisí buď na správnosti vyplněných dat (u databází), nebo na hustotě a rovnoměrnosti rozmístění sond, nebo na stabilitě zpoždění paketů v síti Internet. Obecně se chyby v prostředí Internetu pohybují od jednotek až desítek kilometrů po stovky kilometrů.



Obr.18. Přesnost radiových (bezdrátových) geolokačních technik.

Tab.1. Přesnost geolokačních technik v Internetu.

Technika	Přesnost
Nejkratší ping	nízká
DNS LOC záznam	nízká až střední
Geo ping	střední až vysoká
CBG	vysoká
TBG	vysoká
Octant	vysoká
SuHa	vysoká[1]

Literatura

1. Alexander, M.: Keeping Online Banking Save: Why Banks Need Geolocation and Other New Techniques Right Now. In: *BankersOnline.com*. 25.4.2005. (Anglicky)
2. Blewitt, G.: Basics of the GPS Technique: Observation Equations. University of Newcastle, Newcastle upon Tyne (1997). (Anglicky)
3. Coope, I. D.: Reliable computation of the points of intersection of n spheres in R^n. *Anziam Journal* (2000), vol. 42, 461-477. (Anglicky)

[1] Přesnost experimentálních výsledků algoritmu SuHa dosahuje v nejhorším případě výsledků algoritmu CBG.

4. Čížek, J.: Hackujeme Google: vím, kde je tvůj Wi-Fi router [online]. 16.12.2010 [citováno 5.8.2011]. *Živě.cz*. Dostupné z www: <http://www.zive.cz/clanky/ hackujeme-google-vim-kde-je-tvuj-wi-fi-router/sc-3-a-155025/default.aspx>.
5. Davis, C., et al: Location Information in the DNS. *RFC 1876*. January 1996. (Anglicky)
6. European Space Agency [online]. 2011 [citováno 9.8.2011]. ESA Galileo navigation. Dostupné z www: < http://www.esa.int/esaNA/galileo.html>. (Anglicky)
7. EverMore Technology Inc.: EverMore GPS Receiver, User Protocol Manual. Issue D (2003). (Anglicky)
8. Fleischer, P.: Data collected by Google cars [online]. 27.4.2010 [citováno 5.8.2011]. *European Public Policy Blog*. Dostupné z www: <http://googlepolicyeurope. blogspot.com/2010/04/data-collected-by-google-cars.html>. (Anglicky)
9. LIF (Location Interoperability Forum): Mobile Location Protocol, version 2.0.0. LIF TS 101 v2.0.0 (2001) (Anglicky)
10. Popescu, A.: Geolocation API Specification [online]. 2010 [Citováno 19.8.2011]. W3C. Dostupné z www: <http://dev.w3.org/geo/api/spec-source.html>. (Anglicky)
11. Roxin, A., Gaber, J., Wack, M., Nait-Sidi-Moh, A.: Survey of Wireless Geolocation Techniques. In: *Globecom Workshops*. IEEE, Washington (2007). ISBN 978-1-4244-2024-7. (Anglicky)
12. Sahi, P.: Geolocation on Cellular Networks. *Geographic Location in the Internet*, B. Sarikaya (Eds.), Kluwer Academic Publishers, New York (2002), 13-50. ISBN 0-306-47573-1. (Anglicky)
13. Srp, J.: Technological Aspects of Geolocation of IP Targets based on Packet Delay Measurement. In: *Proc. of Cyter 2011*. ČVUT, Praha (2011). ISBN 978-80-01-04846-7. (Anglicky)
14. Thorvaldsen, O. E.: Geographical Location of Internet Hosts using a Multi-Agent System. Master of Science thesis. Norwegian University of Science and Technology, Trondheim (2006). (Anglicky)

Poděkování

Autor děkuje za podporu výzkumu Českému vysokému učení technickému v Praze, Fakultě dopravní, Ústavu bezpečnostních technologií a inženýrství, Ministerstvu vnitra ČR a EU v rámci projektu FOCUS.

Annotation:

Geolocation and Geolocation techniques

Ability to determine the position of electronic equipments, known as location, localization or geolocation is in these days one of the key technologies to determine the exact location (or at least the area) of transport (aircraft, cars and trucks) or people (e.g. if they are missing). It is also used for determining the place of the criminal activities (e.g. the place where the hacker is doing the attack) but also in services such as navigation, determination of language version for web applications, targeted advertising, etc. It is possible in many ways to determine the location of an electronic object that receipts or even sends electronic data. These ways are called geolocation technologies. Each has its advantages but also weaknesses and it is applicable only under certain assumptions. This tutorial summarizes their summary description and the basis of their operations.

Servisně Orientované Architektury

Martin NEČASKÝ, David KUSÁK, Karel RICHTA

Katedra softwarového inženýrství, MFF UK Praha
Malostranské nám. 25, 118 00 Praha
{jmeno.prijmeni}@mff.cuni.cz

Abstrakt. Pojem "servisně orientovaná architektura", neboli SOA, je v dnešní době skloňován ve všech pádech odbornou i laickou veřejností. Jedná se o architektonický přístup k budování firemní IT infrastruktury, založený na konceptu služeb. Snaží se o vzájemné "sladění" potřeb organizace s její IT infrastrukturou. Hlavním přínosem aplikace SOA uvnitř společnosti by mělo být zvýšení její flexibility a efektivity jejího chodu. Právě konceptem SOA a aplikací tohoto konceptu v praxi se v tutoriálu zabýváme. V jeho první části se posluchači seznámí se základními aspekty SOA. V další části je zavedena metodika pro tvorbu SOA řešení, tzv. metodika byznysem řízeného vývoje. Hlavní částí tutoriálu je potom rozbor fází životního cyklu SOA řešení, tj. modelování, vývoj, nasazení v prostředí organizace, monitorování a nakonec jeho adaptací dle měnících se požadavků.

Klíčová slova: servisně-orientovaná architektura, byznysem řízený vývoj, služba, byznys proces

1 Úvod

Fungování dnešních organizací je silně závislé na kvalitě jejich IT infrastruktury. Organizace požadují, aby byla přizpůsobena procesům, v rámci kterých provádějí svoji primární činnost. Často jsou procesy vykonávány napříč celou organizací a jsou do nich zapojováni i externí partneři (zákazníci, dodavatelé, …). Organizace požadují, aby bylo možné IT infrastrukturu flexibilně přizpůsobovat měnícím se požadavkům a aby bylo možné monitorovat a optimalizovat nejenom IT infrastrukturu, ale i organizaci samotnou.

V tomto tutoriálu se seznámíme se servisně-orientovanou architekturou (SOA). SOA je někdy zjednodušována na softwarovou architekturu postavenou na volně vázaných softwarových komponentách zvaných služby. Ukážeme však, že se jedná o podstatně složitější koncept, jehož hlavním cílem je sladění činnosti IT s potřebami organizace. Z pohledu SOA odlišujeme dvě základní části organizace. První část budeme nazývat byznys. Reprezentuje potřeby organizace a je zodpovědná za provádění primárních činností organizace (např. primární činností nemocnice je léčení pacientů). Druhou část budeme nazývat IT. Ta poskytuje byznysu IT podporu. Jedná se o IT oddělení organizace nebo o jejího dodavatele IT řešení. Základní charakteristikou SOA je právě kladení důrazu na potřeby byznysu a chápání IT jako podpory k uspokojení těchto potřeb.

Tutoriál je strukturován následovně. V kapitole 2 se nejprve seznámíme se základními pojmy v terminologii SOA včetně samotného pojmu SOA. Také se zaměříme na principy, které charakterizují SOA řešení. V kapitole 3 se potom seznámíme s metodikou byznysem řízeného vývoje, která je doporučována právě pro realizaci SOA řešení. V kapitole 4 probereme životní cyklus SOA řešení a ukážeme jak jej realizovat v praxi. V závěru pak budeme krátce diskutovat témata, která se do tutoriálu nevešla.

J. Zendulka, M. Rychlý (eds.), DATAKON 2011, Mikulov, 15.–18.10.2011, pp. 49–74.

2 Co je SOA?

Servisně-orientovaná architektura (SOA) je architektonický styl budování softwarových systémů v organizacích založený na službách. ***Službou*** rozumíme komponentu, která nabízí určitou jasně definovanou funkcionalitu a je nezávislá na svém okolí. Nemusí být nutně automatizovaná pomocí software. Příkladem je služba, která ověřuje správnost adresy zadané klientem při objednávání výrobků nebo služba, která realizuje platbu kreditní nebo debetní kartou. První může být řešena částečně softwarově, ale částečně je také třeba manuální ověření adresy pracovníkem organizace. Druhá je typicky řešena plně automatizovaně.

Každá služba má dvě části – kontrakt a implementaci. ***Implementací*** rozumíme realizaci služby. Není přístupná jejímu okolí. V případě, že lze službu automatizovat, je implementací softwarový program. V případě, že nemůže být plně automatizována, je služba implementována pomocí pracovníků organizace.

Kontrakt je potom část služby, která je přístupná jejímu okolí – tj. klientům uvnitř či vně organizace. Popisuje, jakou funkcionalitu služba nabízí a jakým způsobem. Konkrétně popisuje, kde a kdy je služba dostupná, za jakých podmínek (technických, ekonomických, právních, atd.), jak dlouho potrvá vyřízení požadavku, atd. Pokud je služba dostupná prostřednictvím softwarového rozhraní, popisuje kontrakt i toto rozhraní. Popisuje technické detaily toho, jak může službu využít softwarový klient (např. komunikační formát, ve kterém musí klient formulovat požadavek a ve kterém mu je vrácena odpověď).

Komunikace v rámci organizace je založena na výměně dokumentů obsahujících informace. SOA tuto skutečnost plně reflektuje a ***dokument*** je tak dalším stěžejním termínem v terminologii SOA. Kontrakt každé služby sestává z popisu toho, jaké typy vstupních dokumentů služba může konzumovat a jaké typy výstupních dokumentů může produkovat. Např. služba pro příjem objednávky očekává jako vstup objednávku a produkuje jako výstup potvrzení nebo zamítnutí objednávky.

Velkým přínosem SOA je možnost přirozeně kombinovat existující služby do větších celků, které realizují procesy v rámci organizace. Protože se SOA zaměřuje primárně na byznys, jsou procesy v terminologii SOA nazývány byznys procesy. ***Byznys proces*** (angl. ***business process)*** je sada vzájemně svázaných aktivit a úloh, které vytvářejí specifický produkt nebo službu pro cílovou skupinu zákazníků (spotřebitelů). V učebnicích bývá někdy označován jako obchodní proces - prostým překladem z angličtiny. Není to zcela korektní, neboť původní význam nesouvisí pouze s obchodem a obchodováním, ale je obecnější - vyjadřuje povinnost, úkol, starost. Obvykle se byznys procesy dělí do tří skupin:

- ***Správní procesy*** (angl. ***management processes***) - to jsou procesy, které řídí celkový chod systému (např. řízení firmy, strategický management).
- ***Operační procesy*** (angl. ***operational processes***) - to jsou ty základní procesy (angl. ***core processes***), které realizují vlastní byznys a vytvářejí požadované hodnoty (např. výroba, prodej).
- ***Podpůrné procesy*** (angl. ***supporting processes***) - to jsou procesy, které podporují operační procesy (např. účetnictví, technická podpora).

Byznys proces bývá často dekomponován na ***podprocesy*** (angl. ***subprocesses***), které mají vlastní atributy, ale nakonec slouží pro dosažení části základního cíle hlavního procesu. Tyto podprocesy lze rozkládat tak dlouho, až dospějeme k aktivitám, které již není vhodné dekomponovat - představují ***úlohy*** (angl. ***tasks***), které jsou v daném kontextu atomické a zřejmé. Návrh byznys procesu by měl respektovat požadavky na kvalitu - zvýšení účelnosti - efektivnosti (přínos pro zákazníka) a zvýšení efektivity (přínos pro organizaci).

Byznys proces je zahájen na základě požadavku uskutečnit nějaký cíl a končí, když je tento cíl uspokojen, příp. konstatováním, že jej uspokojit nelze. Abychom se tomuto riziku vyhnuli, je vhodné byznys procesy analyzovat, plánovat a řídit. ***Procesně orientovaná organizace*** je řízena přímo na základě svých byznys procesů. Systematicky analyzuje probíhající procesy a snaží se odstranit tu funkční strukturu, která nepřináší nějaké hodnoty. Např. procesně orientované školící zařízení zajišťující výuku se řídí zejména potřebami byznys procesu učení, neboť to je jádro toho, proč toto zařízení existuje.

2.1 Principy SOA

Orientace na služby neznamená jen dekompozici logiky systému na služby a jejich využití k realizaci byznys procesů. Nedílnou součástí SOA je také sada principů, bez jejichž dodržení můžeme jen těžko splnit požadavky byznysu a udržet vybudovanou IT architekturu v souladu s jejími byznys procesy a především ji adaptovat vzhledem k měnícím se požadavkům byznysu. Uveďme osm základních principů poprvé zavedených v [3]. Abychom mohli nazývat IT řešení servisně orientované, musí být postaveno na službách, ale také musejí být dodrženy tyto principy.

- ***Standardizace kontraktů služeb*** je princip, který vyžaduje, aby kontrakty služeb byly popisovány předem dohodnutým (tj. standardizovaným) způsobem aplikovaným napříč celou organizací. Např. se jedná o dohodnutí jmenných konvencí používaných při popisu rozhraní služby, způsobu popisu obsahu a struktury dokumentů vyměňovaných se službou, sdílení definic struktury dokumentů, způsob popisu politik pro řízení přístupu ke službě či vydávání nových verzí, atd.
- ***Volné vázání služeb*** znamená minimalizovat počet závislostí mezi službami. Okolí služby by mělo být závislé pouze na kontraktu, který je službou zveřejněn. Neměly by být vytvářeny žádné závislosti na nezveřejněné vlastnosti služby, které plynou např. z její implementace či technologií použitých k implementaci.
- ***Abstrakce kontraktů služeb*** je princip, který klade důraz na abstrahování kontraktu služby od nepotřebných detailů. Např. jakákoliv informace o implementaci služby či technologii použité k implementaci by neměla být součástí kontraktu. Abstrakce je důležitým předpokladem pro princip volného vázání služeb.
- ***Opakovaná použitelnost služeb*** je základním principem SOA. Klade důraz na možnost využívat službu opakovaně v různých kontextech organizace, kde je potřeba využít její funkcionalitu. Jinými slovy princip říká, že musí být možné využívat službu pro konstrukci různých byznys procesů, jejichž součástí je funkcionalita nabízená službou.
- ***Autonomie služeb*** je princip, který klade důraz nejenom na možnost kdykoliv upravovat či zcela měnit implementaci služby nezávisle na okolí, ale také na to, aby bylo možné v případě nutnosti upravovat přímo kontrakt služby.
- ***Bezestavovost služeb*** klade důraz na udržování minimální stavové informace na straně služby. Stav je totiž typicky charakteristikou klienta a jeho udržování na straně služby by mohlo mít dopad na její dostupnost.
- ***Dohledatelnost služby*** je princip, který vyžaduje, aby služba byla v rámci organizace dohledatelná pro každého, kdo potřebuje využít její funkcionalitu. Může to např. znamenat udržování registru služeb.
- ***Možnost komponovat služby*** je vedle opakované využitelnosti služeb dalším základním principem SOA. Klade důraz na to, aby bylo možné novou funkcionalitu primárně řešit kompozicí existujících služeb a implementovat nové služby zcela od začátku (na zelené louce) až když kompozice není možná.

3 Byznysem řízený vývoj

V dnešní, stále těsnější ekonomice jsou na IT ze strany byznysu kladeny se stále větší intenzitou nové požadavky. Aby IT přežilo v prostředí kontrolovaném byznysem, musí svoji činnost propojit (angl. *align*) s požadavky byznysu. Byznys procesy se neustále mění a společnost se musí pružně přizpůsobovat novým strategiím. Zde se objevuje problém s procesem vývoje podnikového software (angl. *Enterprise Software Development Process*). Ten postrádá potřebnou agilitu a nestíhá tak tempo byznysu, který se snaží držet své postavení v silném konkurenčním boji na trhu. IT se proto musí vymanit z vytváření software orientovaného na IT a posunout se směrem k vytváření řešení orientovaných na byznys.

Byznysem řízený vývoj (angl. ***Business-driven development***, zkr. ***BDD***) [11] je metodika vývoje IT řešení, která přímo uspokojuje požadavky byznysu a jeho potřeby. Je doporučováno, aby IT řešení založené na SOA bylo realizováno právě dle metodiky byznysem řízeného vývoje. V této části tutoriálu si metodiku vysvětlíme podrobněji.

3.1 Potřeba byznysem řízeného vývoje

Velká část z IT rozpočtu společnosti je spotřebována na údržbu či úpravu stávajících aplikací. Ty nebyly vytvářeny s požadavkem na flexibilitu a proto, zatímco byznys se předhání s konkurencí v boji o propracovanější procesy, IT není schopno dostatečně rychle reagovat na požadované změny. Tradiční aplikace a architektury nejsou schopny držet krok s inovacemi byznysu primárně proto, že implementované procesy nejsou adaptabilní na měnící se požadavky byznysu. Požadavky byznysu se nezřídka kdy transformují do nezávislých IT projektů, které potom pouze omezeně spolupracují. Možnost opakovatelné použitelnosti artefaktů již jednou vytvořených v rámci různých projektů je tak mizivá.

Tradičním přístupem k vývoji software je také nepříznivě ovlivněna náročnost úprav takto vytvořených systémů - ta je díky neflexibilním architekturám tak vysoká, že zásahy do takového systému jsou pro byznys pouze obtížně obhajitelné. Pro vytváření aplikací dostatečně flexibilních a schopných reagovat na neznámé požadavky musí být zvolen systematičtější přístup k vývoji aplikací.

Pro vývoj flexibilnějších aplikací musí být ustanoven mechanizmus, kde IT snahy přímo vycházejí z požadavků byznysu a strategií skrz tzv. ***aplikační rámec (angl. framework)***, který je standardizován, dobře pochopen, a může být vykonáván opakovaně a úspěšně. Podnik může dosáhnout flexibility byznysu modelováním byznys procesů, které kolektivně definují cestu, kterou byznys jde.

Základem je namodelovat byznys procesy z jednotlivých aktivit. Měřením byznys procesů nebo klíčových ***případů užití*** (angl. ***use cases***) pomocí ROI (angl. *Return On Investment*, tj. *výnosnost investice*), KPI (angl.. *Key Performance Indicator*, tj. klíčový ukazatel výkonu) nebo jiné metriky může podnik použít modely byznys procesů jako základní mechanizmus pro sdělování byznys požadavků do IT světa.

Prvním krokem pro BDD je vytvoření modelu byznys procesů. Struktura IT řešení se také musí změnit, aby modely byznys procesů byly vstupem pro návrh a vývoj v životním cyklu softwarového řešení. IT infrastruktura musí být také připravena navrhovat a vyvíjet procesní aktivity jako softwarové komponenty.

Použití BDD poskytuje podnikový model existujících a nových byznys procesů IT oddělení. Analýza nového procesu může odhalit, že již existují softwarové komponenty, které by naplnily byznys požadavky procesu, tudíž tyto komponenty mohou být využity pro realizaci byznys procesu. Nebo se zjistí, že žádné vhodné komponenty neexistují a pak bude potřeba takové softwarové komponenty vytvořit a přidat je do IT portfolia podniku.

Podobně, pokud je potřeba existující byznys proces změnit, model byznys procesu se musí přepracovat, aby odpovídal změně. Model je poté doručen do IT k podrobnější technické revizi.

BDD pomáhá zvyšovat agilitu byznysu a také pomáhá stanovit priority pro IT aktivity a propojit je s příkazy ze strany byznysu. Nepřímo také pomáhá zjednodušovat proces proplácení prostředků pro IT.

3.2 Principy a praktiky byznysem řízeného vývoje

Při vývoji každého IT řešení je nutné si stanovit metodiku vývoje. BDD je obecná metodika a je tedy nutné, aby si ji každá organizace upřesnila tak, aby vyhovovala jejím potřebám. Je však dobré se držet několika obecných principů a praktik. Ty jsou důležité pro hladkou spolupráci mezi byznysem a IT. V této části tutoriálu si z nich představíme ty základní [8].

Adaptace metodiky

Tento princip říká, že je důležité adaptovat metodiku BDD vždy dle charakteristik konkrétního projektu v organizaci. Pro malý projekt realizovaný jedním lokálním týmem bude lepší aplikovat méně striktní metodiku, zatímco pro velký projekt, který je vyvíjen více týmy ve více lokalitách bude nutné aplikovat striktnější metodiku. Je též důležité adaptovat míru striktnosti metodiky dle jednotlivých fází života projektu. Na začátku, kdy je potřeba více kreativního myšlení, se můžeme řídit benevolentnějšími pravidly. Později v projektu je naopak zapotřebí striktnějších pravidel (například kontrola změn v kódu, zaznamenávání testů, atd.).

Organizace by také měla dbát na kontinuální zlepšování metodiky. Po skončení každé iterace a projektu by se měly sesbírat zpětné vazby a podle nich patřičně metodiku vylepšovat. Plán každého projektu a přidružené odhady by měly být v rovnováze s nejistotou projektu. Na začátku projektu, kdy je nejistota projektu velká, by se mělo projektové řízení soustředit primárně na aktivity související s odstraněním nejistoty, než na věštění termínů dokončení té které fáze. Čím méně bude v projektu zbývat nejistoty, tím bude projektový plán přesnější. A čím dříve tomu tak bude, tím lépe.

Vyvážení vzájemně neslučitelných požadavků zainteresovaných osob

Tento princip formuluje důležitost vyvážení často protichůdných požadavků ze strany byznysu a ze strany dalších *zainteresovaných osob (angl. stakeholders)*. Většina zainteresovaných osob by si přála aplikaci, která by dělala přesně to, co si přejí oni sami. Na druhou stranu chtějí redukovat čas na vývoj. Příkladem může být realizace řešení pomocí „krabicového“ software. To může být připraveno k produkčnímu nasazení výrazně rychleji a levněji než vyvíjení aplikace “na míru všem zainteresovaným“. Nevýhodou ovšem je nevyhovění části požadavků zainteresovaných stran.

Neméně důležité je pochopit požadavky a umět je uspořádat dle důležitosti. Správným postupem je zachycení byznys procesů ve společnosti a svázání těchto procesů s projekty a požadavky. To umožní efektivně odhadnout důležitost projektů a požadavků. Důležitost se pochopitelně vyvíjí a mění na základě toho, jak se vyvíjí naše porozumění požadavkům zainteresovaných stran.

Dalším vhodným doporučením je centralizace vývojových aktivit okolo požadavků zainteresovaných osob. Použitím ***vývoje řízeného případy užití*** (angl. ***use case driven development***) a návrhu soustředěného na uživatele, jsme schopni akceptovat, že potřeby zainteresovaných osob se mohou vyvíjet v průběhu trvání projektu a my lépe pochopíme, které požadavky jsou skutečně důležité pro zainteresované strany. Vývojový proces musí

vyhovět těmto změnám. Důležité je také vědět, jaké provozní zdroje společnosti *(angl. assets)* jsou dostupné, a poté vyvážit znovupoužití provozních zdrojů společnosti s požadavky zainteresovaných stran. Příklady provozních zdrojů společnosti mohou být staré aplikace, služby, znovu-použitelné komponenty a vzory. Znovu-použití provozních zdrojů společnosti může v mnoha případech vést k redukci nákladů projektu. Znovu-použití ověřených provozních zdrojů společnosti znamená vyšší kvalitu nové aplikace.

Spolupráce napříč týmy

Mnoho vytvářených SOA projektů vyžaduje vzájemnou spolupráci velkého počtu lidí, kteří nezřídka pracují v různých lokalitách. Pro distribuovanou povahu SOA řešení se proto musí při jeho vytváření dbát na správnou spolupráci mezi všemi zúčastněnými týmy. Prvním krokem k zajištění efektivní spolupráce je motivace jednotlivců v týmu, aby pracovali nejlépe, jak dovedou. Dalším krokem je podpoření komunikace napříč funkcemi. Je obecně známým faktem, že vznikají komunikační bariéry mezi analytiky, vývojáři, testery a dalšími rolemi. Tyto bariéry lze efektivně omezit správnou motivací a správným nastavením zodpovědností. Nastavením vzájemně se překrývajících zodpovědností pro dané role se dá nenásilnou formou vynutit častá smysluplná komunikace mezi jednotlivými rolemi. Příkladem může být analytik, který konzultuje s vývojářem své kroky, spolupracuje s testery na tvorbě testovacích scénářů, atd. V týmu se také musí udržovat povědomí o tom, jak práce daného člověka zapadá do vytvářeného celku. Tím se členům týmu dostává cenná motivace.

S růstem vzájemně spolupracujících týmů musí také vznikat efektivní prostředí pro spolupráci. Takové prostředí by mělo co nejvíce využívat automatizovaných postupů a prostředků pro kolaboraci mezi týmy a pro usnadnění často vykonávané práce členy týmu. Tím je myšleno například automatizované sbírání, uchovávání a reportování stavů komponent členům týmů, automatické sestavování aplikací, atd. Cílem je umožnit členům týmů soustředit se maximálně na svou práci a co nejméně je obtěžovat organizačními a technologickými záležitostmi, které si nevyžadují jejich zapojení.

Iterativní předvádění hodnoty

Tento princip je založen na několika pravidlech. Prvním je doručování zvyšující se hodnoty pro umožnění časné a kontinuální zpětné vazby. Toho je docíleno rozdělením projektu do množiny iterací. V každé iteraci se zjistí požadavky, provede se návrh, implementace a testování aplikace. Výsledkem každé iterace je produkt, který je vždy o krok blíže finálnímu řešení. Díky tomu je možné demonstrovat aplikaci koncovým uživatelům a zainteresovaným osobám, nebo je nechat s aplikací přímo pracovat a získat tak od nich rychlou zpětnou vazbu.

Dalším pravidlem je využití předvádění a zpětné vazby pro přizpůsobení svých plánů. Namísto spolehnutí se na hodnocení specifikací jako např. specifikace požadavků, návrhový model, či plánu je možné ohodnotit "jak dobře funguje aplikace ve stávající m stavu?" Důležité jsou výsledky testování. Tohle poskytuje dobré pochopení toho, jaký je stav projektu, jak rychle dokáže tým pokročit a zda je potřeba udělat korekci rozhodnutí, aby mohl být projekt zdárně dokončen. Informace z každé dokončené iterace se použijí pro doplnění úhrnného plánu projektu a pro vybudování detailního plánu pro další iteraci.

Třetím pravidlem je akceptace vzniku změn a jejich správa. Dnešní aplikace jsou příliš komplexní na to, aby perfektně vyhovovaly požadavkům, návrhu, vývoji, testování hned napoprvé. Namísto toho, nejvíce efektivní metodiky vývoje aplikací přijímají nevyhnutelnost změn. V průběhu časných a kontinuálních zpětných vazeb se dozvíme, jak zlepšovat aplikaci a iterativní přístup poskytuje možnost implementovat tyto změny

inkrementálně. Všechny tyto změny musí být spravovány vhodným procesem a nástroji. Čtvrtým pravidlem je vytlačit klíčová rizika brzy v životním cyklu projektu. Hlavní technická, byznysová a programová rizika musí být uchopena co nejdříve. Čím déle se jejich rozřešení odsouvá, tím větší riziko projekt nese. Toho je dosaženo kontinuálním vyhodnocováním toho, jakým rizikům čelíme, seřazením podle závažnosti a uchopením zbývajících nejvíce závažných rizik v následující iteraci. V úspěšných projektech se zabývají časné iterace sběr vizí a požadavků od zainteresovaných osob a také architektonickým návrhem a implementací nosných částí a jejich otestováním pro zmírnění technologických rizik.

Pozvednutí úrovně abstrakce

Jeden z hlavních problémů, kterým při vývoji software čelíme, je složitost (komplexita). Složitost má přímý dopad na produktivitu. Práce na vyšší úrovni komplexity redukuje složitost a usnadňuje komunikaci.

Jedním z efektivních způsobů, jak snížit složitost je znovupoužití existujících provozních zdrojů společnosti jako jsou znovupoužitelné komponenty, systémy minulé generace (angl. *legacy systems*), existující byznys procesy, vzory, atd. SOA je pro znovupoužitelnost existujících provozních zdrojů společnosti velmi přínosná a to díky principu "volného vázání" aplikací.

Jiným přístupem pro redukování složitosti a zkvalitnění komunikace je použití nástrojů vysoké úrovně, koncepčních rámců (angl. *framework*) a jazyků. Standardní jazyky pro modelování (jako UML) a programovací jazyky pro rychlý vývoj aplikační logiky poskytují schopnost vyjádřit vysoce-úrovňové konstrukty jako byznys procesy a komponenty služeb k usnadnění spolupráce při skrytí nepotřebných technických detailů. Další možností pro řízení složitosti je zaměření se na architekturu, bez ohledu na to, zda se snažíme definovat byznys, systém či aplikaci. Ve vývoji software se snažíme mít architekturu navrženou, implementovanou a otestovanou v raných fázích projektu. Dobrý návrh architektury v rané fázi může vytvořit pružnou kostru struktury pro náš systém a tím snáze řídit složitost, když v dalším průběhu projektu přidáváme více lidí, komponent, schopností a kódu.

Kontinuální zaměření se na kvalitu

Zlepšování kvality není pouze plnění požadavků či produkování produktů s očekávanou kvalitou. Namísto toho kvalita také obsahuje identifikaci měřítek a kritérií pro demonstraci pokroku stejně jako implementaci procesu ujišťujícího, že produkt vytvořený týmem dosahuje požadovaného stupně kvality, který může být zopakovatelný a řiditelný.

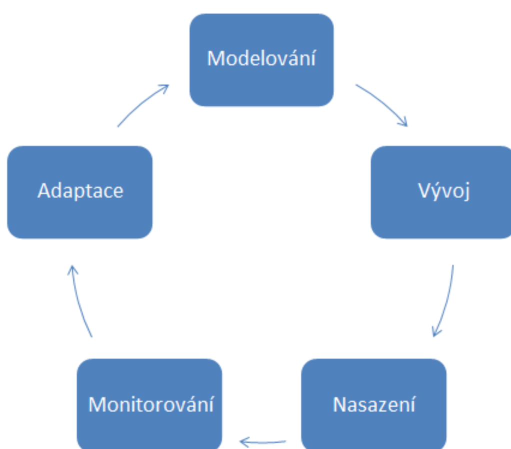
Zajištění vysoké kvality vyžaduje vice než účast testovacího týmu. Vyžaduje jistou kvalitu od celého týmu. To zahrnuje všechny členy týmu a všechny fáze životního cyklu vývoje řešení. Analytici jsou zodpovědní za ujištění se, že požadavky jsou testovatelné a že jsou specifikovány jasné kroky, které mají být provedeny v testech. Vývojáři musí mít testování na paměti při návrhu aplikací a musí být zodpovědní za testování svého kódu. Management se musí ujišťovat, že jsou použity správné plány pro testování a že jsou použity správné zdroje pro budování testovacích nástrojů a že jsou testy prováděny. Testeři jsou experti kvality. Směrují zbytek týmu k porozumění otázkám kvality software a jsou zodpovědní za různé testy. Když se vyskytne problém kvality, každý člen týmu by měl být ochoten přispět svou částí pro uchopení problému.

Jedna z hlavních výhod iterativního vývoje je, že umožňuje testování brzy a kontinuálně. Pokud dodržujeme to, co jsme psali v předchozích krocích a většina důležité funkcionality je implementována v raných fázích projektu, dá se předpokládat, že v čase

odevzdání projektu bude již důležitá funkcionalita delší dobu naimplementována a testována. Není překvapením, že většina projektů adoptujících iterativní vývoj tvrdí, že zvýšení kvality je primárně hmatatelné díky vylepšeným procesům. Jak inkrementálně budujeme naší aplikaci, měli bychom také inkrementálně tvořit automatizované testy. Už při návrhu by se mělo myslet na to, jak se bude testovat. Dobré návrhářské rozhodnutí může výrazně vylepšit možnost automaticky testovat.

4 Životní cyklus SOA řešení

V této kapitole představíme životní cyklus SOA řešení, který vyplývá z metodiky byznysem řízeného vývoje. Fáze životního cyklu přehledně ukazuje Obrázek 1.



Obrázek 1: Životní cyklus SOA řešení vyplývající z metodiky byznysem řízeného vývoje

Životní cyklus sestává z pěti fází. První fází je ***modelování***, v níž je úkolem namodelovat požadované byznys chování na úrovni byznys procesů a konceptuálního popisu dat. V následující ***vývojové*** fázi je cílem na základě artefaktů z předchozí fáze navrhnout a zrealizovat služby implementující byznys procesy a poté tyto služby vhodně propojit. Následuje fáze ***nasazení***, ve které jsou funkční služby nasazeny do běhového prostředí podniku. Další fází je ***monitorování*** řešení a sběr klíčových hodnot důležitých pro vyhodnocování kvality řešení v následující fázi ***adaptace***.

Projekty vytvářející SOA řešení řídící se popisovaným BDD tedy sestávají z více iterací, kdy na konci každé iterace se ve fázi adaptace vyhodnotí kvalita a správnost dosavadního celku a v další iteraci jsou provedeny úpravy řešení. Počet iterací i přesný podíl té které fáze životního cyklu v dané iteraci se může lišit v každém projektu. Dobrou praktikou může být naplánovat v první iteraci objemnější fázi modelování z důvodu vytvoření prvotního kompletního modelu řešení, zatímco ve fázi vývoje naplánovat pouze realizaci klíčových procesů. V dalších iteracích je možno provést přesný opak - fáze modelování již zdaleka nebude tak rozsáhlá jako v první iteraci, dominantní fází co do rozsahu se stane fáze vývoje.

4.1 Modelování SOA

Cílem fáze modelování je identifikovat a popsat procesy, které probíhají v rámci organizace. Začínáme od obecného (ale přesného) popisu společnosti (její přidané hodnoty a dílčích cílů, které k dosažení hodnot vedou) a postupně jej upřesňujeme až na úroveň návrhu jednotlivých služeb. Mezi oběma úrovněmi je několik stupňů. Ve fázi modelování však popisujeme organizaci nezávisle na konkrétních službách. Ty identifikujeme a navrhujeme až ve fázi vývoje, kterou popisujeme v další kapitole.

Architektura organizace

Přirozeným základem popisu SOA řešení je ***architektura organizace*** **(*business architecture*)** [15]. Namísto technických a implementačních detailů se zaměřuje na popis procesů, na základě kterých organizace funguje. I když je architektura organizace vzdálena cílovému technickému řešení, je vhodné ji popsat přesně a strukturovaně.

Popis architektury je vhodné vizualizovat pomocí diagramů, které použijeme pro diskuzi s netechnickými pracovníky organizace. Architektura není popsána pomocí jediného diagramu. Je doporučováno použít různé typy diagramů na různých úrovních abstrakce. V praxi se používají nejčastěji:

- value chain diagram,
- business context diagram,
- business capability diagram,
- diagram případů užití,
- diagramy byznys procesů.

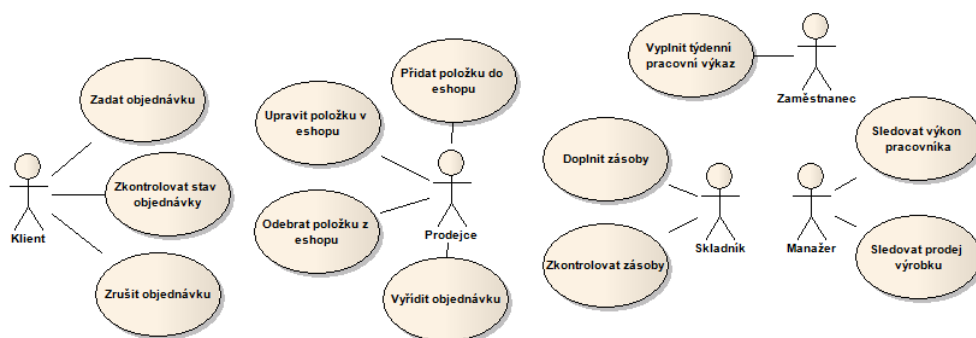
První tři typy diagramů slouží k popisu organizace jako celku. První popisuje organizaci z pohledu jejích klíčových na sebe navazujících agend (řetězec), v rámci kterých organizace vytváří přidanou hodnotu pro svoje klienty. Druhý popisuje celkový tok dokumentů v organizaci. Třetí shrnuje funkční požadavky organizace důležité z pohledu jejích klíčových aktivit. Tj. popisuje, co organizace potřebuje k tomu, aby mohla vykonávat svoji činnost.

My se ale v tutoriálu detailněji těmito typy diagramů nebudeme zabývat. Posuneme se na nižší úroveň abstrakce, kde se věnujeme identifikaci a popisu jednotlivých byznys procesů. Připomeňme si, že byznys proces je definován jako sekvence kroků prováděných jedním či více aktéry za účelem dosažení konkrétních výstupů, které slouží k naplnění cílů jednotlivých klíčových aktivit v rámci organizace.

Prvním krokem vedoucím k identifikaci a popisu byznys procesů může být ***diagram případů užití*** (***use case diagram***) [16]. Ten má dva typy komponent – aktéry a případy užití. ***Aktér (actor)*** reprezentuje klíčovou roli v organizaci. ***Případ užití (use case)*** identifikuje konkrétní ucelenou činnost, kterou daný aktér provádí s cílem dosáhnout určitého cíle.

Příklad diagramu případů užití ukazuje Obrázek 2. Byl vytvořen pomocí nástroje Enterprise Architect[1]. Aktéři jsou zobrazeni jako postavy. Diagram popisuje aktéry *Klient*, *Prodejce*, *Skladník*, *Manažer* a *Zaměstnanec*. Případy užití jsou zobrazeny jako elipsy. V našem případě jsme jako případy užití popsali např. činnosti *Zadat objednávku*, *Vyřídit objednávku* či *Doplnit zásoby*. Propojení mezi aktéry a případy užití potom znázorňují, jaké činnosti provádí jaký aktér. Např. aktér *Klient* provádí činnosti *Zadat objednávku*, *Zkontrolovat stav objednávky* a *Zrušit objednávku*.

[1] http://www.sparxsystems.com/



Obrázek 2: Ukázka diagramu případů užití

Činnosti jsou pomocí případů užití pouze identifikovány a popisovány. Je ale dále potřeba činnosti popsat detailněji. Popis je v terminologii případů užití nazýván *scénář*. Scénář je sekvence kroků, které aktér při činnosti vykonává. Často nám pro popis případu užití nestačí pouze jeden scénář. V určitých specifických situacích je totiž činnost prováděna dle jednoho či více *alternativních scénářů*. Případ užití je pak popsán jako soubor scénářů, kde jeden je *hlavní* a ostatní *alternativní*.

Scénáře případů užití nám přirozeně definují byznys procesy. Budeme je proto jako byznys procesy modelovat. Nejjednodušším řešením je modelovat hlavní a alternativní scénáře jednoho případu užití jako samostatný byznys proces. Není to ale striktní pravidlo. Někdy může být výhodné scénáře případu užití modelovat jako podproces jiného procesu. To je v situaci, kdy případ užití je vykonáván také jako součást jiného případu užití. Je také možné sdílet jednotlivé kroky mezi byznys procesy atd.

Formálně je byznys proces modelován pomocí modelu byznys procesů a model je vizualizován pomocí vhodné notace, například jako diagram aktivit UML [16] nebo jako ***BPMN diagram*** [1] (angl. ***Business Process Model and Notation***). V praxi se používají obě notace. UML diagramy aktivit mají širší použití. BPMN diagramy naproti tomu nabízejí silnější vyjadřovací prostředky, které jsou důležité právě při návrhu SOA řešení. Zde se podrobněji zaměříme na BPMN diagramy.

BPMN diagram popisuje byznys proces jako sekvenci kroků - aktivit, které musí být v rámci procesu vykonány. Nejjednodušší aktivity se nazývají ***úlohy (tasks)*** a jsou z hlediska BPMN atomické - dále nedělitelné. Úlohy jsou zobrazeny jako obdélníky (často se zakulacenými rohy). Složitější aktivity pak mohou být složeny z jiných aktivit, které představují ***podprocesy (subprocesses)***. Podprocesy jsou označeny ikonkou se znaménkem plus, aby se zdůraznilo, že to není atomická úloha. Speciálním případem podprocesů jsou ***transakce*** - podprocesy, jejichž všechny aktivity musí být vykonány, nebo zrušeny jako celek.

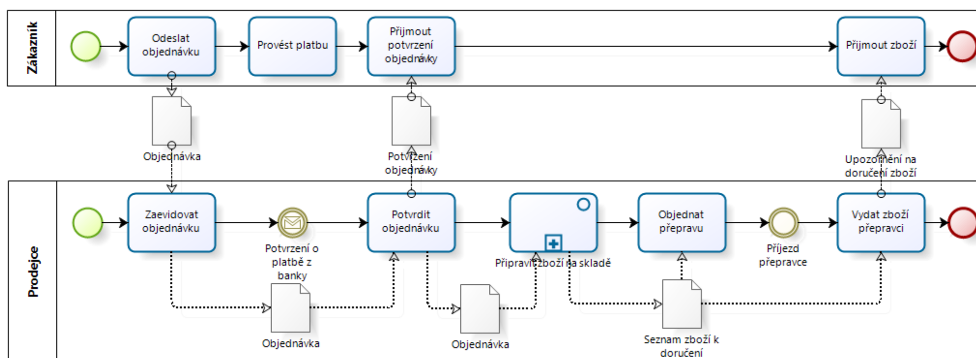
Procesy a aktivity lze uzavřít do tzv. ***oblastí zodpovědnosti (swimlanes - plavecké dráhy)***, které lze označit zodpovědným aktérem - ten, kdo bude za tuto část procesu zodpovědný.

Procesní kroky a aktivity na sebe navazují - to se vyjadřuje pomocí plných šipek představujících ***tok řízení (sequence flow)***. Během provádění mohou procesy posílat zprávy - ty se vyznačují pomocí čárkovaných šipek ***toků zpráv (message flows)***. Přechody řízení mohou být kombinovány pomocí tzv. ***bran (gateways)***. Brány se vyznačují pomocí kosočtverců (diamonds) obsahujících symbol příslušné operace (fork/join, inclusive decision merge).

K popisu procesů patří události - nastartování procesu, ukončení procesu, příp. událost, která nastane během provádění procesu. Událost je znázorněna jako kruh, uvnitř kterého je symbol, který charakterizuje událost (časová událost, čekání na zprávu, podmínka, atd.)

Příklad diagramu modelujícího scénáře případu užití *Vyřídit objednávku* jako byznys proces v notaci BPMN ukazuje Obrázek 3. Byl vytvořen pomocí nástroje BizAgi Modeler[2]. Skládá se ze dvou oblastí odpovědnosti. První reprezentuje zákazníka, druhý prodejce. Model popisuje byznys proces z pohledu obou aktérů. Zákazník začíná byznys proces prvním krokem *Odeslat objednávku*. Následují pak další zákazníkovi kroky, dokud není proveden krok *Přijmout zboží*, kterým zákazník proces ukončuje. Na straně prodejce je proces spuštěn krokem *Zaevidovat objednávku* od zákazníka. Prodejce potom čeká na *potvrzení o platbě z banky*. To je modelováno jako událost. Po přijetí potvrzení obchodník *potvrzuje objednávku zákazníkovi*, zajišťuje přípravu zboží na skladě a *objednává přepravu*. Po příjezdu přepravce předává zboží přepravci a ukončuje proces. Všimněme si, že prodejcův krok *Připravit zboží na skladě* je modelován jako podproces. Jeho detail je modelován v jiném diagramu, který neuvádíme.

BPMN diagram znázorňuje jak tok řízení (plné šipky), tak tok dokumentů (přerušované šipky). Tok řízení modeluje časovou souslednost kroků v rámci oblasti zodpovědnosti. Tok dokumentů modeluje, jaké dokumenty jsou předávány mezi různými kroky stejných nebo různých oblastí zodpovědnosti. Pro jednotlivé toky dokumentů modelujeme i dokumenty, které jsou předávány. Např. mezi prvními kroky *Odeslat objednávku* a *Zaevidovat objednávku* existuje tok, v rámci kterého je předáván dokument objednávka.



Obrázek 3: Ukázka diagramu byznys procesu *Vyřídit objednávku* v notaci BPMN

Informační model organizace

V modelu byznys procesů identifikujeme kromě procesních kroků také toky dokumentů mezi nimi. Krok dokumenty konzumuje a produkuje. V BPMN diagramu jsou dokumenty modelovány jako tzv. ***artefakty (artifacts)*** a jsou znázorňovány jako obdélníky se zahnutým pravým horním rohem. To, zda je dokument vstupní nebo výstupní je modelováno směrem šipky, která dokument s krokem spojuje. V našem příkladu jsou některé dokumenty modelovány. Např. vstupním i výstupním dokumentem aktivity *Zaevidovat objednávku* je *Objednávka*. Vstupním dokumentem aktivity Potvrdit objednávku je opět *Objednávka*. Výstupním dokumentem aktivity je *Potvrzení objednávky*.

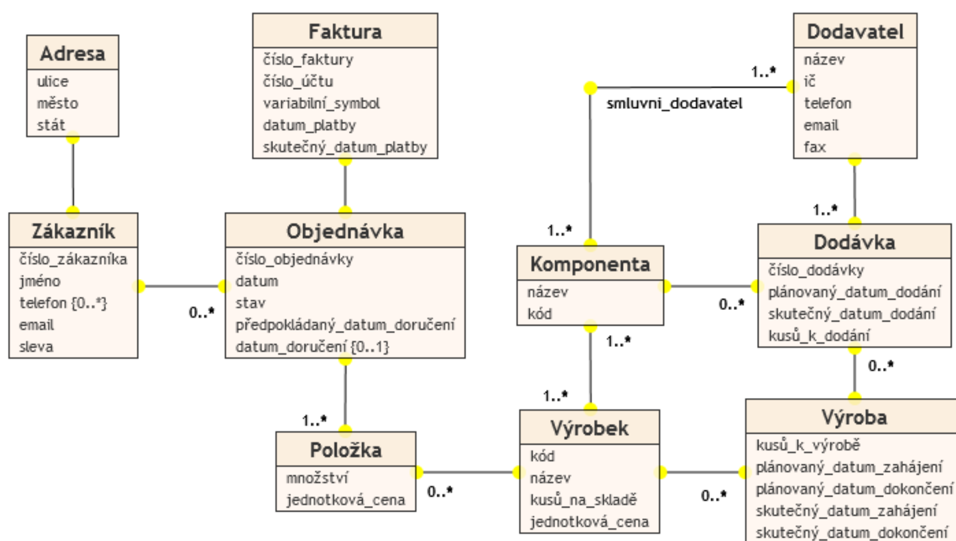
Stejně jako modelujeme byznys procesy, je doporučováno modelovat také dokumenty [15]. To jaké typy dokumentů mají být uvažovány, vyplývá z modelu byznys procesů. Nyní

[2] http://www.bizagi.com/

ale potřebujeme typy dokumentů specifikovat detailněji. Jinými slovy potřebujeme popsat, jaké informace jsou v dokumentech vyměňovány. Model informací je nazýván ***informační model organizace*** (někdy také nazývaný ***sémantický model organizace***). Dále jej budeme nazývat jen informační model. Informační model má dvě části. První částí je konceptuální model datové domény. Modeluje doménu jako celek a zaměřuje se na popis její sémantiky. Druhou jsou konceptuální modely pro jednotlivé typy dokumentů. Ty nerozšiřují sémantiku domény. Pouze říkají, jaká část domény je reprezentována v jakém typu dokumentů.

Nejprve je potřeba jednotným a formálním způsobem popsat strukturu a sémantiku dat, se kterými organizace pracuje v rámci svých byznys procesů. Přitom je nutné, aby byl popis nezávislý na konkrétním procesu nebo typech dokumentů. K tomu slouží ***konceptuální model datové domény*** (zkráceně ***model domény***). Typicky se jedná o UML diagram tříd [16] na vysoké úrovni abstrakce (nezajímají nás konkrétní datové typy, klíčové atributy či metody tříd). Třídy popisují entity důležité z pohledu organizace (např. *objednávka*, *výrobek*, …). Používáme také atributy a asociace pro popis jejich charakteristik (např. *cena* a *kód* výrobku) a vztahů mezi nimi (např. *objednávka objednává výrobek*), atd.

Příklad modelu domény naší organizace ukazuje Obrázek 4. Byl vytvořen pomocí nástroje eXolutio[3]. Je vyjádřen a vizualizován jako UML diagram tříd. UML diagramy tříd jsou k tomuto účelu běžně používány. Model není z důvodu nedostatku místa kompletní. Pokrývá agendu zákazníků, objednávek a faktur, skladů, výroby a dodavatelů. Celý model by pokrýval i všechny zbylé agendy.



Obrázek 4: Ukázka konceptuálního modelu datové domény znázorněné pomocí UML diagramu tříd

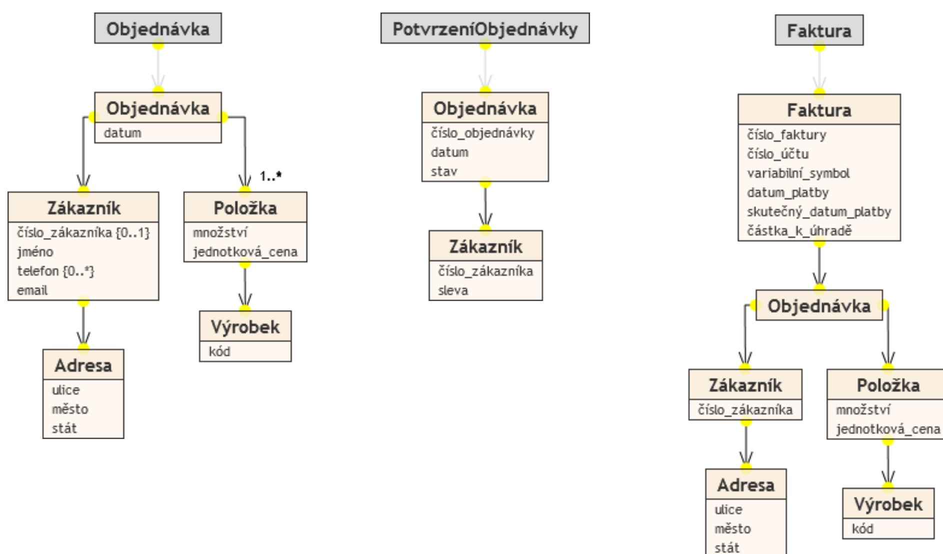
Na další úrovni je potom potřeba modelovat jednotlivé typy dokumentů, které jsou zasílány v rámci organizace. Z pohledu organizace dokument obsahuje informace, které jsou popsány částí konceptuálního modelu datové domény. Je běžné, že dokument má hierarchickou strukturu. Je nositelem nějaké hlavní informace (např. *objednávka*) a v ní jsou vnořeny v rámci dokumentu další doplňující informace (např. *zákazník*, který objednávku vytvořil a *seznam objednaných výrobků* v objednávce). Místo slovního popisu toho, jaké informace daný typ dokumentů obsahuje, jej modelujeme formálně jako

[3] http://eXolutio.com

podmnožinu modelu domény. Tj. pro každý typ dokumentu vyznačíme tu část modelu domény, která modeluje informace přenášené v dokumentech, a vyznačíme požadovanou hierarchickou strukturu. Výsledný popis typu dokumentu nazýváme ***konceptuální model dokumentu*** (zkráceně ***model dokumentu***) [12].

Příklad konceptuálních modelů dokumentů zobrazuje Obrázek 5. Příklad byl vytvořen opět pomocí nástroje eXolutio. Obrázek 5(a) ukazuje model *objednávek*, které zasílají zákazníci při objednávání výrobků. Obsahuje část objednávky (datum), informaci o objednávajícím zákazníkovi (kompletní, vč. adresy) a seznam položek (kompletní, vč. kódu objednaného výrobku). Jsou zde drobné rozdíly v násobnostech. Číslo zákazníka má násobnost 0..1 (na rozdíl od modelu domény). To je proto, že objednávku může odesílat nový zákazník, který ještě není evidován v systému. Obrázek 5(b) ukazuje model *potvrzení objednávek*, které jsou zákazníkovi vráceny po zpracování zaslané objednávky. Obsahuje detailnější informaci o objednávce (číslo přiřazené objednávce v systému a její stav). Na druhou stranu obsahuje méně detailní informaci o zákazníkovi (jen jeho číslo a slevu) a neobsahuje seznam položek. Poslední Obrázek 5(c) ukazuje model *faktur*. Obsahuje navíc fakturu a jeho další struktura je podobná (ne však stejná) s modelem *objednávek*.

Při realizaci potom potřebujeme navrhnout pro reprezentaci dokumentů konkrétní formáty. Ty ale nejsou nyní důležité. Jen poznamenejme, že formát je navržen právě na základě modelu dokumentu. Jeho jednotlivé komponenty jsou mapovány do gramatiky popisující komunikační formát (např. mapujeme konceptuální model dokumentu na XML schéma v případě, že používáme pro komunikaci formát XML).



Obrázek 5: Ukázky modelů pro (a) objednávky, (b) potvrzení objednávek a (c) faktury

Interoperabilita mezi službami

Kroky byznys procesů nebo jejich části budeme ve výsledném SOA řešení realizovat jako služby. Je nutné, aby služby byly vzájemně interoperabilní. Může se totiž stát, že danou službu budeme chtít využít pro realizaci různých kroků. Kroky ale budou mít různé vstupní a výstupní dokumenty. Potřebujeme, aby služba byla schopna konzumovat, resp. produkovat tyto různé typy dokumentů. Jinými slovy potřebujeme, aby služba byla

interoperabilní. Interoperabilita má dvě dimenze: strukturální a sémantickou. ***Strukturální interoperabilita*** znamená, že služba rozumí všem potřebným technickým formátům, ve kterých jsou příchozí a odchozí dokumenty reprezentovány. ***Sémantická interoperabilita*** pak znamená, že služba má alespoň stejnou interpretaci domény jakou mají typy jejích vstupních a výstupních dokumentů, i když konkrétním technickým formátům rozumět nemusí. Potom sice není schopna přímo konzumovat či produkovat dokumenty v požadovaných formátech, ale je možné ji doplnit o tzv. mediátory, které zajistí konverzi do formátů, kterým již rozumí. Díky sémantické interoperabilitě je zajištěno, že mediátory existují.

Sémantické interoperability dosáhneme právě tím, že všechny možné typy dokumentů, které se mohou v našem SOA řešení vyskytnout, jsou modelovány na základě společného modelu domény. Ten totiž modeluje společnou sémantiku. Je vhodné použít nástroj, který umožní nejenom dokumenty modelovat ale také jejich modely navázat na společný model domény. Vazby potom usnadní orientaci v sémantice dokumentů a také usnadní správu změn v modelu domény a v modelech dokumentů. To diskutujeme v kapitole 4.5.

Interoperability je někdy dosahováno pomocí aplikace standardů pro reprezentaci dokumentů. Ty jsou často založeny na jazyce XML. Standardy jsou specifické pro daná odvětví lidské činnosti. Existují standardy dvou druhů. ***Konceptuální standardy*** definují informační model organizace, která působí v daném odvětví. ***Technické standardy*** definují přímo komunikační formáty pro výměnu určitých typů dokumentů. První je možné převzít a přizpůsobit v rámci vlastní organizace. Pomůže se zajištěním sémantické interoperability nejenom v rámci vlastní organizace, ale i mezi organizacemi navzájem. Druhý zajišťuje strukturální i sémantickou interoperabilitu. Avšak jen do chvíle, než je potřeba standard rozšířit či jinak upravit pro vlastní potřeby (což je nutné prakticky vždy). V tu chvíli vzniká riziko narušení sémantické a nemožnosti obnovení strukturální interoperability.

V kontextu České republiky standardy v některých odvětvích existují. Příkladem může být Datový standard Ministerstva zdravotnictví ČR[4].

4.2 Vývoj SOA

V této kapitole popisujeme činnosti, které jsou prováděny ve fázi vývoje SOA řešení. Vstupem je popis byznys procesů ve formě modelu byznys procesů doplněný o konceptuální model domény a modely dokumentů, jež si jednotlivé kroky byznys procesů předávají. Oba tyto modely byly vytvořeny v přechozí fázi modelování. Na jejich základě jsou ve fázi vývoje vykonávány tyto aktivity:

- identifikace kandidátských služeb,
- návrh rozhraní služeb a určení způsobu jejich realizace,
- definování SLA (z angl. *Service Level Agreement*, tj. dohoda o úrovni poskytování služeb),
- realizace služeb,
- propojení služeb.

Na základě modelu byznys procesů se ve spolupráci s byznysem procesy kategorizují podle vhodných kritérií. Těmi může například být potřeba rychlé dodávky, rizikovost procesu, atd. Poté je ve spolupráci s byznysem vytvořen přibližný plán vývoje procesů. Ten musí obsahovat informaci, které procesy budou realizovány jako první. Tudíž v závislosti na iteraci projektu nemusíme provádět realizaci všech byznys procesů. Identifikace

[4] http://ciselniky.dasta.mzcr.cz/

kandidátských služeb by nicméně měla být provedena pro všechny byznys procesy, může sloužit jako dobrá zpětná vazba pro fázi modelování.

Identifikace kandidátských služeb

Z modelu byznys procesů je potřeba nejdříve identifikovat vhodné kandidáty na služby, tzv. *kandidátské služby*. Byznys proces je složen ze sekvence aktivit, které jsou buď atomické (úlohy) nebo složené (podprocesy). Jako kandidátské služby označíme všechny byznys procesy a všechny jejich úlohy a podprocesy, které realizujeme.

Dalším krokem je odstranění duplicitních kandidátských služeb. Těmi jsou kandidátské služby, které mají stejné vstupní a výstupní dokumenty a které jsou určené k vykonávání stejné logiky. Dalším důvodem odebrání kandidátských služeb z kandidátského portfolia je přítomnost dvou či více podobných služeb, z nichž jedna může nahradit zbylé. V portfoliu kandidátských služeb tedy zůstane jedna z těchto služeb, či vznikne nová kandidátská služba. V obou případech je vhodné konzultovat kroky s autorem modelu byznys procesů.

Při další analýze mohou také vznikat ještě další kandidátské služby. Např. v případě kdy je shledán nějaký sled kroků byznys procesu jako znovupoužitelný, přidáme kandidátskou službu odpovídající tomuto sledu. Kandidátské služby představující dané kroky v portfoliu kandidátských služeb zůstanou.

Model byznys procesů může obsahovat úlohy a podprocesy, které již v organizaci jsou realizované a vystavené jako služby. Při realizaci kandidátských služeb se tato skutečnost zohlední. Pro tuto chvíli je důležité si uvědomit, že ne všechny kandidátské služby budou nutně nově implementovány.

Identifikaci kandidátských služeb předvedeme na příkladu. Vstupem bude náš příkladový model procesu *vyřízení objednávky*. Z atomických operací byznys procesu můžeme identifikovat tyto kandidátské služby: *Zaevidovat objednávku*, *Potvrdit objednávku*, *Objednat přepravu*, *Vydat zboží přepravci*. Jako další kandidáty na služby identifikujeme z byznys procesů a podprocesů - *Vyřízení objednávky*, *Příprava zboží na skladě*.

Určení způsobu realizace kandidátských služeb

Po identifikaci kandidátských služeb přichází na řadu realizace služeb. Pro každou službu z portfolia kandidátských služeb bude určeno, jak se má realizovat.

V zásadě existuje několik způsobů, jak realizovat službu. První možností je *použití existující služby*. Pokud již existuje v portfoliu organizace služba realizující požadovanou funkcionalitu (bez ohledu na to, zda představuje úlohu byznys procesu, či byznys proces samotný), měli bychom ji využít. Je to nejsnazší varianta realizace služby a také základní princip SOA. Tato varianta nebývá často použitelná při prvních SOA projektech, protože organizace ještě vůbec žádné portfolio použitelných služeb nemá. Použití existující služby je možné pouze tehdy, pokud má kandidátská služba rozhraní kompatibilní (sémanticky i strukturálně) s rozhraním existující služby. Pokud nemá, musí se zvolit jiná varianta.

Druhou možností je *použití externí služby*. V určitých případech totiž může být výhodné využít službu vystavenou jiným subjektem. Tímto způsobem mohou být realizovány kandidátské služby odvozené z úloh a podprocesů modelu byznys procesů mající kompatibilní rozhraní s touto externí službou. Použitím služby externího dodavatele odpadnou relativně vysoké náklady na vlastní implementaci, avšak z dlouhodobého hlediska se může jevit “pronájem” dodávané služby nákladnější než vybudování vlastní funkcionality podobného rázu. Nese to s sebou i další rizika. Jedním z nich je bezpečnost dat. Vně organizace totiž nemáme zcela pod kontrolou zasílaná data. Ty mohou být v případě konzumace služby od nedůvěryhodného subjektu v ohrožení. Dalším rizikem je

správné nastavení SLA. SLA definuje dostupnost konzumované služby, její odezvu a další nefunkční parametry. Tyto nefunkční aspekty služby nemůžeme jako subjekt, jenž službu nevlastní, přímo ovlivnit, proto bychom neměli využívat tuto možnost pro realizaci klíčových služeb.

Třetí možností je *využití existující aplikace*. To může být dobrá volba, pokud taková funkcionalita existuje a pokud služba představuje úlohu byznys procesu (tato metoda není aplikovatelná pro realizaci služeb popsaných celým byznys procesem). Existující byznys funkcionalitou jsou myšleny existující aplikace či systémy, které plní požadovanou funkcionalitu, ale neexistuje k nim vhodné rozhraní. Vytvořit rozhraní k existující byznys funkcionalitě může být snazší na realizaci než vytváření celé nové služby. Záleží ovšem na použitých technologiích, schopnostech realizačního týmu a na preferencích organizace (např. nemusí být žádoucí nákladně budovat rozhraní pro byznys funkcionalitu systému minulé generace, který plánujeme nahradit).

Nejnákladnější možností, která ale zároveň nejlépe vyhoví požadavkům, je *vybudování nové služby* ("na zelené louce"). Je nevyhnutelná při realizaci služeb, které představují celé byznys procesy. Je také nevyhnutelná při realizaci služeb představujících úlohy byznys procesu, které není možno realizovat žádnou z předchozích možností. Pokud může funkcionalitu kandidátské služby nahradit kompozice nových či jiných kandidátských služeb, měla by tato možnost být využita. Zvláštním případem jsou kandidátské služby představující uživatelské vstupy procesu (angl. *human tasks*). Jejich realizace spočívá ve vystavení obrazovkového rozhraní pro uživatele. To je ovšem mimo rozsah tutoriálu a dále se jí nevěnujeme.

Na příkladu ukážeme, jak by mohl být určen způsob realizace služeb identifikovaných z modelu byznys procesů. Kandidátskou službu *Zaevidovat objednávku* bychom realizovali obalením části existující byznys funkcionality systému pro správu objednávek. Systém pro správu objednávek v dané organizaci nenabízí žádné strojově použitelné rozhraní, toto rozhraní tudíž musí být vytvořeno. Kandidátská služba *Potvrdit objednávku* by mohla být realizována obalením části byznys funkcionality stávajícího systému pro správu objednávek, podobně jako u kandidátské služby *Zaevidovat objednávku.* Kandidátskou službu *Objednat přepravu* bychom mohli realizovat použitím externí služby. Tato služba může být nabízena přepravní společností. Kandidátskou službu *Vydat zboží přepravci* bychom realizovali existující službou, kterou nabízí systém spravující sklad. Realizací kandidátské služby *Příprava zboží na skladě* by mohla být existující služba systému pro správu skladu, která představuje byznys proces pro přípravu zboží na skladě. Tato služba již v organizaci existuje a lze ji použít. Kandidátská služba *Vyřízení objednávky* bude realizována vlastní implementací pomocí orchestrace existujících služeb.

Výběr technologií pro realizaci služeb

Je také potřeba zvolit technologii pro realizaci služeb. Službu je možné realizovat jako webovou službu (angl. web service), REST službu (angl. REST service), atd. Specifikum webových služeb je, že webová služba umožňuje sdružovat několik různých funkcionalit vzájemně souvisejících. Každá funkcionalita je definována jednou operací, skupina těchto operací lokalizovaná na stejném umístění je potom nazývána webová služba. Může to být trochu matoucí, dále v tutoriálu nebudeme pojem "operace" používat. Každou SOA službu budeme chápat jako samostatnou funkcionalitu a čtenář znalý webových služeb si může námi popisované SOA služby představovat jako webové služby mající každá jednu operaci.

Pro služby je také potřeba rozhodnout strojový formát, který bude použit pro výměnu dokumentů mezi službami. Např. můžeme rozhodnout, že dokumenty budou vyměňovány jako XML dokumenty. Konkrétní XML formáty potom odvodíme přímočaře

z konceptuálních modelů jednotlivých typů dokumentů. Každá služba vychází z nějakého procesu, úlohy či sledu úloh. V modelu mají určeny typy vstupních a výstupních dokumentů, které jsou modelovány právě pomocí konceptuálních modelů dokumentů,

Realizace služeb

Po zvolení způsobu realizace služeb a výběru technologií můžeme služby realizovat. Probereme pouze realizaci služby jako takové, propojení s realizovaným celkem je popsáno v následující části Propojení služeb. Popsali jsme čtyři možnosti realizace služeb. Dvě z nich jsou založeny na využití existující služby (interní nebo externí) a proces realizace je tedy přímočarý.

Pokud se rozhodneme využít stávající aplikaci či systém, který sám o sobě žádnou službou není, musíme jej obalit tak, aby se z něj služba stala. V tomto případě závisí pracnost realizace služby na technologii použité pro implementaci obalovaného systému, na jejím návrhu a v neposlední řadě též na velikosti rozhraní realizované služby. Pro komunikaci s takovým systémem musí být zrealizován *konektor* umožňující vyvedení funkcionality systému tak, aby byla služba schopna technologicky přijatelným způsobem přistupovat ke zdrojům systému. Konektor může být buď dodán autorem systému, pokud se jedná o dodávaný systém, nebo musí být realizován vlastními prostředky. Takovýto konektor bývá navržen znovupoužitelně, takže by se celý či z větší části měl dát znovupoužívat pro realizace více služeb komunikujících s daným systémem. Při vlastní implementaci takového konektoru je zapotřebí zapojit do projektu tým implementující či spravující daný systém, ve většině případů se totiž musí konektor budovat jako komponenta na straně systému. Samotná implementace služby a její náročnost závisí hlavně na návrhu konektoru samotného. V mnoha případech může jít o přímočaré transformování položek zprávy na zdroje systému a zpět

Druhou možností je *implementace nové služby*. Ta se v první řadě odvíjí od typu byznys funkcionality, kterou má služba reprezentovat. Pokud má služba představovat atomickou úlohu byznys procesu, může být realizována jako aplikace vystavující rozhraní pro komunikaci přes zvolenou technologii. Implementace musí být provedena v programovacích jazycích a platformách umožňujících komunikaci přes zvolenou technologii. Vhodnými platformami jsou např. JEE (Java Enterprise Edition) či Microsoft .NET. Náročnost vývoje je plně závislá na složitosti implementované byznys logiky. Přesný proces vývoje, použitá platforma pro vývoj a detaily vývoje záleží na zvyklostech organizace a jejích procesech. Popis je však již mimo rozsah tohoto tutoriálu. Více je možno se dočíst v [7][6][2].

Pokud má služba představovat byznys proces či podproces, je vhodné pro realizaci použít specializované nástroje obsahující v sobě ***orchestrační nstroj*** (angl. ***orchestration engine***). Často se služby implementují jako webové služby a používá se orchestrace založená na jazyce ***BPEL*** (angl. ***Business Process Execution Language***). Výhodou realizace pomocí takto specializovaných nástrojů je nízká náročnost orchestrace služeb. Náročnost bývá podstatně nižší, než vývoj orchestrační logiky přímo v nějakém programovacím jazyce, jako tomu bylo v případě implementace služby zastupující atomickou úlohu. Při požadavku čekání na vnější vstupy od uživatele či na vnější události v rámci běhu procesu by měl být proces implementován pomocí více transakcí s uchováním stavu, často označován jako dlouhotrvající proces (angl. *long-running process*). V opačném případě se jedná o jedno-transakční zpracování procesu obvykle nazývané jako krátkodobý proces (angl. *short-running process*). Více informací je možno získat v [10][5].

Na vývoj služeb lze také nahlížet jako na vývoj kompozitních aplikací majících definované rozhraní umožňující volat danou aplikaci jako službu. Vývoj kompozitních

aplikací může být prováděn buď za pomocí orchestračních logiky popsané výše či pomocí specifické technologie pro vytváření kompozic služeb. Touto technologií je ***SCA*** (***Service Component Architecture***)[5]. Vývoji kompozitních aplikací se v tutoriálu blíže nezabýváme a čtenář může bližší informace získat v [10].

Propojení služeb

Nakonec se zaměřme na realizaci propojení dvou různých služeb. Propojit služby potřebujeme vždy, když realizují různé úlohy či části procesu, které si vyměňují dokumenty. Nejjednodušší řešení je odkázat se z jedné služby na druhou přímo z kódu programu. To ale předpokládá neměnné rozhraní odkazované služby a neměnnou metodu přístupu (místo, kde je služba dostupná atd.). Změna místa či zpětně nekompatibilní změna jejího rozhraní by znamenala změnu kódu a nové nasazení řešení. Takovýto způsob je nepružný a neefektivní. Ukažme si dva prostředky, které vedou k efektivnějšímu propojování služeb.

Prvním prostředkem je tzv. *registr služeb*. Slouží pro správu služeb v organizaci. Umožňuje konzumentům služeb napojovat se na služby nepřímo. Konzumenti se mohou registru služeb dotazovat na vhodnou službu tak, že specifikují rozhraní požadované služby a popíší nefunkční požadavky, které na ni mají. Registr vyhledá službu či služby, které odpovídají dotazu a vrátí jejich seznam. Pro každou službu vrátí místo, kde je dostupný její kontrakt včetně lokace služby. Konzument potom pracuje s touto informací dynamicky. Při změně místa služby tedy stačí změnit údaj v registru služeb a konzumenty není potřeba žádným způsobem měnit.

Druhým prostředkem je tzv. ***ESB*** (angl. ***Enterprise Service Bus***, tj. ***podniková sběrnice služeb***). ESB, jak již z názvu napovídá, představuje centrální sběrnici používanou pro propojení služeb uvnitř společnosti. Nabízí několik základních funkcionalit, z nichž ty nejzajímavější zmiňujeme.

Na ESB je možné vystavit tzv. ***zástupnou službu*** (angl. ***proxy service***) k jakékoliv službě v organizaci. Jedná se o tzv. ***virtualizaci služeb*** (angl. ***service virtualization***). Konzumenti namísto původní služby používají službu zástupnou, která zprostředkovává volání původní služby. Díky tomu zamezuje tzv. špagetovému propojování služeb - stavu, kdy jsou služby propojené napřímo, což je dlouhodobě neudržitelná strategie. Další výhodou je také možnost konzumenty odclonit od změn na původní službě. Aby byla konfigurace zástupných služeb co nejsnazší, ESB pro tento účel často využívá funkce registru služeb. Zástupná služba může být na ESB také vystavena pro použití přes jiné přenosové protokoly, než podporuje původní služba. Vnitřní logika ESB potom zprostředkovává tzv. ***přemostění protokolu***, tj. stav, kdy se volání zástupné služby na ESB přeloží do protokolu vhodného pro původní službu.

ESB také usnadňuje práci s daty. ***Transformace dat*** (angl. ***data transformation***) je funkcionalita, která definuje, jak se má procházející zpráva transformovat, než je doručena k cílové službě. To je neocenitelné v situaci, kdy se poskytovaná služba v čase vyvíjí a mění se jí rozhraní. ESB tak může konzumenty služby odclonit od určitých změn na původní službě. ***Obohacení dat*** (angl. ***data enrichment***) je speciálním typem transformace dat. Jedná se o obohacení procházející zprávy obohatit o další data - například o data z databáze, či o data jiné služby.

Směrování na základě obsahu (angl. ***content-based routing***) je další funkcionalitou ESB, která umožňuje na základě obsahu příchozí zprávy volit službu, která bude provolána.

[5] http://www.osoa.org/display/Main/Service+Component+Architecture+Home

Díky tomu může ESB plnit funkci ***brány*** (angl. ***gateway***) umožňující přijímat na jedné zástupné službě zprávy od více různých služeb a směrovat je podle dynamických pravidel.

Mezi další funkcionality ESB patří například možnosti ***monitorování průtoku zpráv***, ***správa transakcí***, ***znovu-doručování zpráv*** v případě nedostupnosti služby, atd.

Propojení externích služeb

Pro komunikaci s externími službami je zapotřebí dodržet obecné zásady bezpečné komunikace, např. použití ***SSL*** (z angl. ***Secure Socket Layer***) za použití ověřených certifikátů pro přenos zpráv.

4.3 Nasazení SOA

Nejvíce přímočarou fází procesu vývoje SOA řešení je nasazení do běhového prostředí podniku. Tady se realizované služby přidají mezi ostatní služby a nastává jejich zkušební provoz a monitorování.

Běhovými prostředími mohou např. být různé JEE aplikační servery, běhová prostředí pro BPEL, a další. U každé služby může být běhové prostředí odlišné, závislé na implementaci dané služby.

4.4 Monitorování SOA

Po uvedení SOA řešení do provozu je velmi důležité monitorovat jeho efektivitu. Existují dvě úrovně pohledu na efektivitu SOA řešení - organizační a technická. Efektivita z prvního pohledu znamená, že je dosahováno cílů a výstupů identifikovaných v modelu organizace. Efektivita z druhého pohledu znamená, že řešení funguje optimálně z hlediska SLA - tj. jsou dodržovány definované časy odezvy, nedochází k výpadkům, atd. Více k měření SLA lze nalézt např. v [14].

Proces měření efektivity z organizačního pohledu se v literatuře nazývá ***monitorování pracovních aktivit*** (zkr. *BAM* z angl. ***Business Activity Monitoring***) [17]. BAM je založen na sběru dat k tzv. *klíčovým výkonnostním indikátorům* (zkr. *KPI* z angl. *Key Performance Indicators*) a jejich vyhodnocování. Pro monitorování je potřeba nejprve definovat KPI organizace. Ty vycházejí z modelu organizace. V jednodušším případě se jedná např. o různé ukazatele prodeje, počty objednávek, počty nových zákazníků, statistiky návratu stávajících zákazníků, využívání zákaznických služeb atd. Také se může jednat o měření prosté frekvence a doby vykonávání byznys procesů implementovaných v SOA řešení. Ve složitějším případě se jedná o časování výskytů různých složitějších událostí, které je potřeba detekovat a reportovat. Mohou být např. používány k monitorování chování zákazníků.

Po identifikaci KPI je potřeba implementovat procesy pro jejich monitorování. V jednodušším případě se jedná o ukládání údajů o spouštění různých služeb, jejichž výstupy odpovídají definovaným KPI. Ve složitějším případě to znamená implementovat mechanismus *detekce komplexních událostí* (zkr. *CEP* z angl *Complex Event Processing* nebo zkr. *BEP* z angl. *Business Event Processing*).

Důležité je, že implementace měření KPI v SOA řešení je usnadněna právě tím, že SOA řešení je orientováno na organizaci jako takovou, tj. na její byznys procesy a informační model. Implementace měření KPI je tedy v podstatě přímočará a mohou ji pomocí vhodných nástrojů provádět i netechničtí pracovníci organizace, kteří jsou zodpovědní za vyhodnocování efektivity organizace.

4.5 Adaptace SOA

Je přirozené, že se mění organizace i její okolí. Je to způsobeno měnícími se požadavky zákazníků a partnerů organizace, měnící se legislativou, konkurenčními organizacemi, novými technologiemi, atd. Změny je potřeba reflektovat v jednotlivých částech SOA řešení. SOA řešení neobsahuje pouze služby, ale i další artefakty na různých úrovních abstrakce. Změnou mohou být zasaženy libovolné z nich. Při jejich úpravách je nutné mezi nimi udržet konzistenci. To znamená, že při změně jednoho artefaktu je nutné změnit odpovídajícím způsobem i všechny ostatní související artefakty. Změna může nastat na jedné z následujících úrovní:

- v modelu organizace (tj. v její architektuře nebo v informačním modelu),
- v rozhraní služeb,
- v implementaci služeb.

Než rozebereme jednotlivé úrovně detailněji, podívejme se ještě na vztah principů SOA ke správě změn. S principy jsme se seznámili na začátku tutoriálu. Některé principy stojí proti sobě. Na jedné straně máme princip znovupoužitelnosti služeb. Ten způsobí, že změna v rozhraní služby či jejím SLA má dopad na všechny kontexty, ve kterých je služba použita, a může vynutit jejich změny. Na straně druhé máme principy jako je volné vázání služeb mezi sebou, autonomie služeb či abstrakce služeb. Tyto principy naopak minimalizují dopad změn ve službě na její okolí.

Změny v modelu organizace

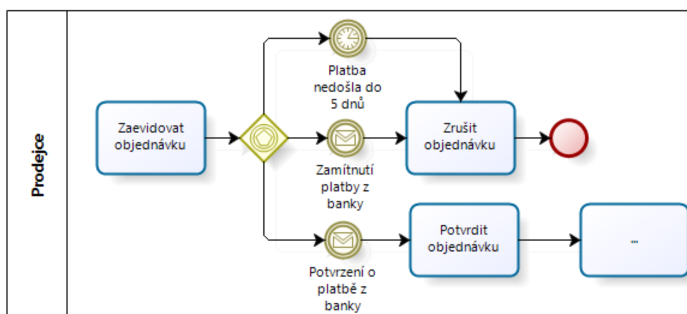
Každý nový požadavek organizace by měl být nejprve analyzován z pohledu architektury organizace. Umožní nám to sledovat dopad změny na různé artefakty SOA řešení a nakonec změnu správně realizovat v systému. Může dojít ke změnám na různých úrovních architektury - v klíčových aktivitách, případech užití či v jednotlivých procesech. Je důležité zajistit konzistenci mezi jednotlivými úrovněmi abstrakce - změna v jedné úrovni má pravděpodobně dopad na jiné.

Nejméně často dochází ke ***změnám v klíčových aktivitách*** organizace. Ty modelují, v rámci jakých agend organizace funguje. Zásah do klíčových aktivit je tedy nutný v případě restrukturalizace organizace či změnou v jejím zaměření. Navazuje na ní sada změn v celých skupinách případů užití (buď vytváření nových případů užití pro novou agendu, nebo odstraňování případů užití odstraňované agendy).

Častěji dochází ke ***změnám v případech užití***. Požadavky zákazníků či partnerů organizace i požadavky vycházející zevnitř organizace mohou vést k vytvoření nových případů užití. V případě, že některá potřeba vymizí, znamená to odstranění existujícího případu užití. K tomu ale dochází jen velmi zřídka. Změny v případech užití je nutné reflektovat v procesech. Nový případ užití může vést k vytvoření nového procesu. Před tím bychom se ale měli snažit zjistit, zda již není případ užití popsán existujícím procesem nebo jeho částí. Teprve pokud ne, založíme nový proces. Při vytváření se snažíme využít existující kroky jiných procesů a vazeb mezi nimi. Pokud odstraňujeme případ užití, nemusí to nutně znamenat odstranění příslušného procesu či všech jeho kroků. Ten totiž může popisovat i jiné případy užití.

Obrázek 6: Změny v případech užití

Příklady změn v případech užití ukazuje Obrázek 6. Nejprve jsme přidali nový případ užití *Vyskladnit zboží*.



Obrázek 7: Změna v modelu byznys procesu pro vyřízení objednávky

Jeho scénář však nemodelujeme jako zcela nový proces, neboť je již modelován jako součást podprocesu *Připravit zboží na skladě* v procesu modelujícím scénář případu užití *Vyřídit objednávku*. Proto jej pouze vydělíme do samostatného BPMN diagramu a vložíme jej zpět jako podproces. Dále jsme přejmenovali případ užití *Zkontrolovat zásoby* na *Zkontrolovat množství zásob*. Nezměnili jsme tím ale proces, který scénář tohoto případu užití modeluje. Jen je nyní název výstižnější. Potom jsme přidali nový případ užití *Zkontrolovat trvanlivost zásob*. Jeho scénář není modelován pomocí žádného existujícího procesu ani jeho části. Proto musíme vytvořit nový proces.

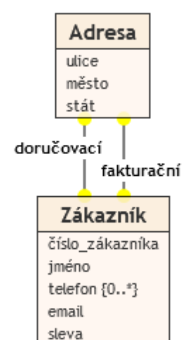
Nejčastěji dochází ke ***změnám v jednotlivých procesech*** – např. při jejich optimalizaci. Především jsou doplňovány o nové kroky či nová propojení mezi existujícími kroky. Existující propojení mohou být přepojována. Při vytváření nových kroků se snažíme nejprve využít existující kroky. Můžeme také potřebovat kroky či propojení odstraňovat (např. při optimalizaci zjistíme, že celá větev procesu je zbytečná). Musíme dbát na to, zda odstraňovaný krok není použit jinde.

Změnu v procesu ukazuje Obrázek 7. Doplnili jsme kromě potvrzení o platbě z banky ještě dvě další události: *zamítnutí platby* a situaci, kdy *platba nedošla do stanoveného počtu dní*. Dále jsme doplnili krok *Zrušit objednávku*, který je proveden v případě, že nastane jedna z těchto dvou situací. Po jeho vykonání proces končí.

Změny v informačním modelu organizace

Změny v organizaci i mimo ni mohou mít samozřejmě také dopad na informační model. Je proto nutné analyzovat každý nový požadavek také z tohoto úhlu pohledu. Výsledkem mohou být změny na obou úrovních informačního modelu - v konceptuálním modelu datové domény i v konceptuálních modelech jednotlivých typů dokumentů.

Ke ***změnám v konceptuálním modelu datové domény*** dochází, pokud potřebujeme pracovat s novým typem informací, příp. pokud se mění naše chápání již existujícího typu informací. První případ vede k rozšiřování modelu o nové části. Druhý případ pak může znamenat odstranění některých částí modelu. Spíše ale dochází k přehodnocení existujících částí modelu a jejich nahrazení detailnějším či naopak méně detailním ekvivalentem (tj. s určitým typem informací potřebujeme pracovat na větší či naopak menší úrovni detailu).

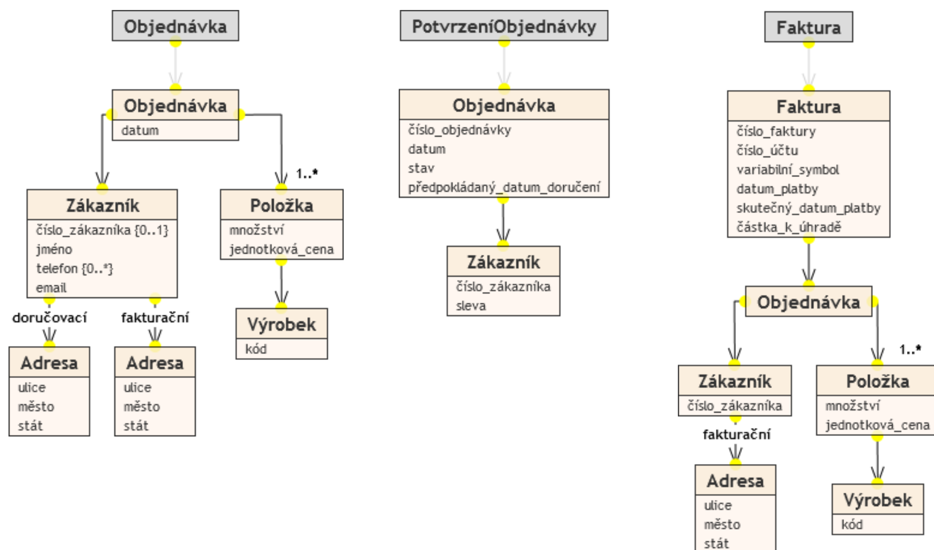


Obrázek 8: Změna v konceptuálním modelu domény

Změnu v modelu domény naší organizace ukazuje Obrázek 8. Původní asociace mezi třídami *Zákazník* a *Adresa* modelovala *adresu zákazníka*. Přišel požadavek odlišit *doručovací* a *fakturační adresu*. Tuto změnu tedy nejprve vyjádříme v modelu domény. Potřebujeme rozdělit původní asociaci na dvě nové, jak je ukázáno na obrázku.

Při změnách v modelu domény může být narušena konzistence s konceptuálními modely jednotlivých typů dokumentů. Je proto nutné identifikovat, jaké modely dokumentů

jsou změnou zasaženy a jakým způsobem. Změnu je poté nutné správně *propagovat*. V našem případě je narušena konzistence modelu *objednávek* a modelu *faktur*. Model *potvrzení objednávek* není narušen, neboť *adresy zákazníků* nereprezentuje. V případě *objednávky* potřebujeme rozdělit stávající asociaci na dvě nové, jak ukazuje Obrázek 9(a). V případě *faktury* potřebujeme pouze nahradit původní asociaci novou, která modeluje *fakturační adresu*, jak ukazuje Obrázek 9(c). Příklad demonstruje, že pro správnou propagaci změn vždy potřebujeme experta, který propagaci provede. Automatizace sice možná je, ale jen v některých speciálních případech. Nástroje nám mohou pomoci s lokalizací nekonzistencí a nabídkou možných řešení. Poznamenejme, že současné nástroje propagaci změn nepodporují dostatečně, jak je analyzováno v [4] [13]. Experimentálně je propagace změn podporována v nástroji eXolutio[6].



Obrázek 9: Ukázky změn v (a) modelu objednávek, (b) modelu potvrzení objednávek a (c) modelu faktur

Někdy je také potřeba ***měnit*** přímo ***konceptuální model konkrétního typu dokumentu***. To je v případě, kdy např. potřebujeme rozšířit vstup nebo výstup nějakého kroku procesu o nový typ informace nebo naopak nějaký typ informace vyjmout. V případě nového typu informace jej již máme v ideálním případě popsaný v modelu domény a pouze jej do modelu dokumentu přeneseme. V horším případě ho musíme nejprve vytvořit v modelu domény. Obecně se snažíme nejprve měnit model dokumentu tak, abychom nemuseli rozšiřovat model domény (často by totiž rozšíření vedlo k duplicitám v modelu domény). Až když to není možné (nový typ informace prostě v modelu domény není namodelovaný), rozšíříme model domény. Jednoduchý příklad ukazuje Obrázek 9(b). Potřebujeme rozšířit model potvrzení objednávky o informaci o předpokládaném datu doručení. Pohledem do konceptuálního modelu domény zjistíme, že datum již modelujeme. Pouze tedy odpovídajícím způsobem rozšíříme model dokumentu, aniž bychom museli modifikovat model domény.

[6] http://www.eXolutio.com

Změny v rozhraní služeb

Změny na úrovni modelu organizace musejí být reflektovány ve službách. Konkrétně mají na služby dopad změny v procesech (vytvoření procesu, kroku či spojení a smazání procesu, kroku či spojení) a změny v modelu dokumentů.

V případě změn v procesech jsou nejčastější změny uvnitř procesů, tj. jsou vytvářeny či odstraňovány jejich kroků a jsou měněny jejich propojení. Občas dochází k vytváření nových procesů či odstraňování existujících. Řešení takových změn vychází z postupů, které jsme již popsali v popisu fáze vývoje. Mohou tedy vést k vytváření nových služeb. Před vytvářením nové služby bychom se ale měli vždy zamyslet, zda nelze potřebnou funkcionalitu pokrýt existující službou.

Vytvoření nového spojení dvou kroků může vést k situaci, kdy potřebujeme propojit dvě služby, které do této chvíle propojeny nebyly. Cílová služba propojení proto musí porozumět dokumentům, které produkuje zdrojová služba. V ideálním případě používají obě služby pro reprezentaci dokumentů stejný formát, tj. že služby jsou navzájem strukturálně interoperabilní. Může se ale stát, že služby nejsou strukturálně interoperabilní. Potom máme dvě možnosti. Buď rozšíříme rozhraní cílové služby tak, že služba bude přijímat nový typ dokumentů (příp. obdobně rozšíříme rozhraní zdrojové služby). To může znamenat i zásah do implementace služby.

Druhou možností je předřadit cílové službě mediátor, která provádí konverzi formátu zdrojové služby na formát cílové služby. Pokud je zajištěna sémantická interoperabilita, mediátor řeší strukturální rozdíly ve formátech dokumentů. Pokud není zajištěna, nemusí být vždy možné mediátor vytvořit.

Uvažujme situaci, kdy v dokumentech reprezentujeme adresu jako jeden řetězec. Z nějakého důvodu nyní potřebujeme v adrese odlišit ulici, číslo popisné, město a stát. Taková změna znamená zásah do modelu domény organizace, kde potřebujeme model adresy změnit na model ulice, čísla popisného, atd. Jedná se tedy o změnu sémantiky a původní služba není sémanticky interoperabilní s novým typem dokumentů. To však ještě neznamená, že mediátor nemusí existovat. Mediátor může být založen na různých heuristikách, jejichž výstupy jsou validovány oproti externí službě ověřující existenci adresy. V případě, že se nepodaří existenci adresy ověřit nebo výsledná adresa není jednoznačná, může být součástí mediátoru i lidský faktor – pracovník zodpovědný za validaci adres manuální kontrolou a telefonickým kontaktováním zákazníka, který adresu zadal.

Změny v modelech dokumentů ovlivňují typy dokumentů, které musejí být služby schopny produkovat nebo konzumovat. Dopad na službu je podobný jako v předchozím případě. V případě strukturálních změn je možné službě představit mediátor, který službu od změn odstíní. Změny v sémantice mohou znamenat zásah do implementace služby. To, zda dochází ke změnám v sémantice, poznáme z toho, zda byl upraven model domény. Pokud ano, mohla být změněna sémantika.

Změny v implementaci služeb

Změnou v implementaci služby rozumíme jak úpravu stávající implementace služby, tak i její nahrazení novou službou se stejným rozhraním. Změny v implementaci by teoreticky, dle principu volného vázání služeb, neměly mít vliv na okolí služby. Jenže to není tak úplně pravda. Bude mít pravděpodobně dopad např. na dostupnost a výkon služby, tj. ovlivní to, zda služba nadále splňuje SLA. Může tedy dojít ke změně SLA, čemuž se okolí musí přizpůsobit.

Verzování služeb

V předchozím textu jsme popsali, jak změny v požadavcích vedou přes model organizace a informační model až ke změnám v jednotlivých službách. Také jsme ukázali, že propagace změn až do stávajících služeb ovlivní okolí služeb. Do okolí spadají jak ostatní služby a klienti služby uvnitř organizace tak i klienti služby vně organizace. Vliv na okolí ale většinou není žádoucí. Znamená totiž zásah do rozhraní ostatních služeb a klientů v okolí a nezřídka i do jejich implementace. Možná jsme schopni tento zásah provést uvnitř organizace. Nejsme ho ale schopni provést vně organizace. Z těchto důvodů je doporučováno zasaženou službu ponechat a vydat novou službu, která implementuje požadované změny. Po nějakou dobu tak původní a nová verze služby existují vedle sebe. To zajistí, že stávající okolí služby bude schopno nadále využívat funkcionalitu prostřednictvím původní služby a postupně se přizpůsobovat na její novou verzi.

V této části probereme přístupy k verzování běžné v praxi [9]. Je vhodné odlišit následující typy změn:

- opravy chyb v implementaci
- zpětně kompatibilní změny (tj. takové, které umožní klientům dále se službou komunikovat beze změn)
- zpětně nekompatibilní změny

V případě oprav chyb je doporučováno nevytvářet novou verzi. V případě zpětně kompatibilních změn je to doporučováno a ve třetím případě je vytvoření nové verze zpravidla povinností. Také jsou často odlišovány tzv. *minoritní nové verze* (*minor versions*) a *majoritní nové verze* (*major versions*). Zpětně kompatibilní změny jsou vydávány jako minoritní verze. Zpětně nekompatibilní pak jako majoritní verze. Minoritní a majoritní je doporučováno odlišit jiným stylem číslování. Je doporučováno používat čísla verzí ve tvaru *X.Y*, kde *X* značí číslo majoritní verze a *Y* pak číslo minoritní verze.

Mezi typy ale samozřejmě není ostrá hranice. Oprava chyby může teoreticky vést až ke zpětně nekompatibilní změně. Zejména však není ostře vymezena zpětná kompatibilita. Např. pokud je rozraní služby zpětně kompatibilní se svojí původní verzí syntakticky, neznamená to ještě nutně, že jsou verze kompatibilní ve svých SLA. Je také nutné pro účely rozhodování o zpětné kompatibilitě uvažovat opravdu všechny součásti rozhraní služby, včetně např. integritních omezení, které jsou na příchozí či odchozí dokumenty kladeny.

Pokud tedy nakonec dojde k potřebě vytvoření nové verze (služby nebo operace), je nutné novou verzi vytvořit a nasadit. Vytvoření nové verze začíná modelováním změn, jak jsme popsali výše. Z modelu vyplynou potřebné změny ve službách či operacích, které implementujeme a nasadíme. Existují dvě základní možnosti:

1. Nasadit novou verzi jako novou službu se svým vlastním přístupovým bodem. Výhodou je přímočarost a tedy větší efektivita řešení bez prostředníka. Nevýhodou je, že klient musí vědět, jak k nové verzi přistoupit.
2. Nasadit novou verzi jako novou službu s přístupovým bodem sdíleným s ostatními verzemi. Společný přístupový bod je pak služba - prostředník, která umí dle obsahu zprávy (např. čísla verze) rozpoznat požadovanou verzi a na ní požadavek směrovat. Výhodou je, že klient stále přistupuje ke stejnému přístupovému bodu nezávisle na verzi. Nevýhodou je složitější a méně efektivní architektura. Službu prostředníka můžeme implementovat jako součást registru služeb. Jako prostředník také může sloužit ESB.

V určité životní fázi služby či její operace nakonec dojde k potřebě odstranit starou verzi, příp. odstranit službu či operaci jako takovou. I tento proces je však potřeba dobře řídit. Je

doporučováno [9] vždy před samotným označit ji jako ***zastaralou*** (***deprecated***) a poté po nějakou dobu monitorovat její používání. Pokud ji již nikdo nepoužívá, můžeme teprve přistoupit k odstranění. Pokud je používána, kontaktujeme klienta služby.

5 Závěr

V tutoriálu jsme představili koncept servisně-orientované architektury (SOA). Ukázali jsme, že SOA není jenom softwarová architektura, jejíž podstatou je realizace software jako kompozice služeb, ale že se jedná o sadu různých doporučení a principů, které je nutné dodržet, aby mohlo být IT řešení nazýváno servisně orientované. Ukázali jsme, že principy usnadňují především sladění potřeb byznysu s IT.

Umožnili jsme však čtenáři pouze nahlédnout do světa SOA. I když jsme se snažili v některých částech (především v popisu životního cyklu SOA řešení) proniknout i do hlubších detailů, nedostali jsme se k mnoha tématům, kterým bychom se jistě měli v úplném popisu SOA detailně věnovat. Tak např. jsme nepopsali role pracovníků, kteří jsou do životního cyklu SOA řešení zapojení, jako např. byznys analytik, IT analytik, IT architekt, atd. Také jsme se nedostatečně věnovali řízení SOA (SOA Governance), zajištění bezpečnosti v rámci SOA řešení, či systémové integraci, která zahrnuje řadu zajímavých problémů. Nevěnovali jsme se také technickému řešení SOA. Sem spadá řada technologií, jako např. sada standardů konsorcia W3C pro realizaci služeb jako tzv. webových služeb, technologie BPEL pro orchestraci webových služeb, službám realizovaným pomocí technologie REST a mnoho dalších. Každé z těchto témat by jistě vystačilo na samostatnou kapitolu či dokonce knihu.

Věříme však, že jsme poskytli základní přehled toho, co SOA představuje, jaké principy v sobě zahrnuje a jak zhruba probíhá realizace SOA řešení v organizaci. Zájemce se nyní může věnovat hlubšímu studiu SOA problematiky, k čemuž může využít řadu zahraničních knižních titulů, které jsou na trhu dostupné.

Literatura

1. *Business Process Model and Notation (BPMN) v2.0.* OMG, 2011. [http://www.omg.org/spec/BPMN/2.0/].
2. Erl, T. et al.: *SOA with .NET and Windows Azure*. Prentice Hall, Boston, 2010.
3. Erl, T.: *SOA: Principles of Service Design*. Prentice Hall, Boston, 2008.
4. Hartung, M., Terwilliger, J., Rahm, E.: Recent Advances in Schema and Ontology Evolution, In: *Schema Matching and Mapping, Data-Centric Systems and Applications,* Springer Berlin, Heidelberg, 2011, 149–190.
5. Havey, M.: *SOA Cookbook.* Packt Publishing, 2008.
6. Hewitt, E.: *Java SOA Cookbook.* O'Reilly Media Inc., Sebastopol, 2009.
7. Kalin, M.: *Java Web Services: Up and Running.* O'Reilly Media Inc., Sebastopol, 2009.
8. Kroll, P., Royce, W.: *Key principles for business-driven development.* [http://www.ibm.com/developerworks/rational/library/oct05/kroll/index.html] (staženo 23.8.2011).
9. Lubinski, B.: *Versioning in SOA.* Microsoft Corporation, 2007. [http://msdn.microsoft.com/en-us/library/bb491124.aspx] (staženo 24.8.2011).

10. Margolis, B., Sharpe, J.: *SOA for the Business Developer-Concepts, BPEL, and SCA.* MC Press, 2007.
11. Mitra, T.: *Business-driven development.* [http://www.ibm.com/developerworks/webservices/library/ws-bdd/] (staženo 23.8.2011).
12. Nečaský, M., Mlýnková, I.: When Conceptual Model Meets Grammar: A Formal Approach to Semistructured Data Modeling. In: *Lecture Notes in Computer Science, Vol. 2010, Num. 6488,* Springer (2010), 279-293.
13. Nečaský, M., Mlýnková, I., Klímek, J.: Model-Driven Approach to XML Schema Evolution. To appear in: *Lecture Notes in Computer Science,* Vol. 2011.
14. *Reference Architecture for an SLA Management Framework*. SLA@SOI EU FP7 project.
15. Rosen, M., Liblinsky, B., Smith, K.T., Balcer, M.J.: *Applied SOA: Service-Oriented Architecture and Design Strategies.* Wiley Publishing, Inc., Indianapolis, 2008.
16. *Unified Modeling Language (UML) v2.4*. OMG, 2011. [http://www.omg.org/spec/UML/2.4/].
17. *Úvod do zpracování událostí.* Firma Galeos. [http://www.galeos.cz/uploads/Soubory/Studie/Uvod_do_zpracovani_udalosti.pdf] (staženo 24.8.2011).

Multiagentní systémy

František ZBOŘIL ml.

Ústav inteligentních systémů, FIT VUT v Brně
Božetěchova 2, 621 00 Brno
zborilf@fit.vutbr.cz

Abstrakt. Inteligentní prvky hrají nezanedbatelnou roli při zpracování procesů a analýze dat v rozsáhlých distribuovaných systémech. Umělí agenti, kteří jsou v těchto typech aplikací často využívání, jsou autonomní, tj. mají plnou kontrolu nad svým konáním, pokud se této kontroly nevzdají přijetím závazku vůči ostatním prvkům systému (například vůči jiným agentům, uživatelům stanic v systémech atd.). Tutoriál představí principy fungováni agentů v distribuovaných systémech, principy jejich rozhodování, přijímání a plnění závazků k vykonání služeb a také principy přijímání, uchovávání a zpracovávání znalostí. Důraz bude kladen na výstavbu agentních systémů v souladu se specifikacemi standardů v těchto oblastech podle FIPA, které vedle specifikace agentních platforem zahrnují i jazyky pro komunikaci mezi agenty založené na řečových aktech (jazyk ACL), komunikační protokoly, specifikace životního cyklu agentů či jejich bezpečnost. Součástí tutoriálu bude také přehled metodik návrhu multiagentních systémů a programovacích jazyků vystavěných na multiagentních paradigmatech.

Klíčová slova: FIPA architektury, Agent Communication Language, JADE, Metodiky návrhu multiagentních systémů

1 Úvod

Účelem tohoto tutoriálu bude seznámení odborné veřejnosti s technikami implementace aplikací využívajících agentních technologií. Vedle struktury multiagentního systému, abstraktní architektury agentní platformy a životního cyklu agenta v systému se zaměříme i na způsoby komunikace mezi agenty jak uvnitř platforem, tak i mezi platformami, na jazyk, který ke komunikaci používají a na jednotlivé komunikační protokoly. Poslední část tutoriálu bude věnován metodikám návrhu multiagentních systémů a ukážeme zde, co je vše třeba specifikovat a modelovat, pokud chceme vytvořit kompletní návrh takového systému.

Agentní technologie jsou předmětem seriozního výzkumu již přes dvacet let, přičemž dnes je zájem soustředěn především na zapojení agentních metod do reálných systémů, a základní výzkum spíše rozvíjí metody a systémy z předchozích období. Implementační rámec těchto systémů je vymezen od počátku minulého desetiletí sadou specifikací organizace FIPA (Foundation for Intelligent Physical Agents). Všechny v současnosti standardizované specifikace dostaly svoji konečnou podobu do roku 2004 a jejích záběr sahá od specifikací architektur, komunikačních jazyků, protokolů až po bezpečnostní otázky spojené s multiagentními systémy.

J. Zendulka, M. Rychlý (eds.), DATAKON 2011, Mikulov, 15.–18.10.2011, pp. 75–99.

2 Implementace umělých agentů

Programátor, který zvolil agentní technologie pro implementaci svého systému, má obecně dvě možnosti. První možností je použít klasických programovacích nástrojů a implementovat komponentu vykazující agentní chování v nich. Druhou možností je pak sáhnout po současných agentních a multiagentních vývojových prostředích. Jelikož tento tutoriál bude zaměřen více na způsob integrace multiagentních podsystémů do jiných aplikací, nebo propojování heterogenních multiagentních částí navzájem, zmíníme způsoby realizací agentů jako takových jen stručně.

Obvykle se za vymezující agentní rysy považují následující:

- **Samostatnost** / **autonomie**: Agent jedná samostatně bez vnějšího řízení, pokud se části své samostatnosti nevzdá (například přijetím závazku respektovat nějaké systémové normy).
- **Reaktivita**: Agent je schopen pohotově a flexibilně reagovat na změny prostředí, ve kterém se nachází.
- **Iniciativa** / **proaktivita**: V případě racionálních agentů, kteří své konání vztahují k nějakým zvoleným cílům, je proaktivitou myšleno ovlivňování prostředí svými efektory tak, aby jeho nový stav přiblížil dosažení těchto cílů.
- **Sociální schopnosti**: V případě multiagentních systémů se hovoří o sociálních schopnostech na úrovni komunikace mezi agenty, sjednávání závazků spolupráce a také o schopnostech formovat koalice a fungovat v nich i na základě již zmíněných sociálních norem.

Výzkum v oblasti tvorby umělého agenta s racionálním chováním byl nejintenzivnější v devadesátých letech minulého a začátkem tohoto století. Dnes umělé agenty chápeme buďto jako čistě reaktivní, bez reprezentace okolí modelem a odvozování chování výpočty nad tímto modelem, nebo jako racionální, které mají nějaký model a jejich chování je dáno tímto modelem a specifikovanými cíly.

Za nejvýznamnější reprezentanty racionálních agentů se dnes pokládají hlavně systémy založené na záměrech agentů a praktickém rozhodování [12] a systémy Agentně-orientovaného programování navržené Shohamem [24].

Systémy řízené záměrem mají svého význačného zástupce v BDI systémech, které vznikaly od teoretických systémů založených na modálních logikách směrem k formálnímu návrhu agenta AgentSpeak(L) [21]. Mezi následné realizace se řadí systému JAM! [16], 3APL/2APL [13] a dnes asi nejpoužívanější systém tohoto typu JASON [11]. Tyto systémy pracují na základě reaktivního plánování s procedurálními znalostmi, kdy struktura záměru se modifikuje v každém z implementačních cyklů podle aktuálních stavů agentových představ a přání. Právě mentální stavy agenta ve formě představ (Beliefs), přání (Desires) a záměrů (Intentions) daly těmto systémům jejich společný název – BDI systémy.

Agentně orientované programování bylo dlouho bez kvalitnější realizace, což napravil systém Agent Factory [1], který jako jeden ze svých tří implicitně podporovaných jazyků řadí i jazyk Agent Factory Programming Language 2 / AFPL2 postavený právě na agentně orientovaném programování. Dalšími dvěma jazyky je jeden pro implementaci reaktivních agentů a jazyk systému JASON. Agentně orientované programování je postaveno na dobrovolném přijímání závazků na vykonání nějaké služby v nějakém čase. Opět zde nalezneme mentální stavy, mezi které patří představy, vědomí závazků, nebo vědomí o svých schopnostech. Agent je schopen přijímat závazky o vykonání určité akce v požadovaném čase a za případných dalších podmínek. Tyto závazky přijímá, pokud je

požádán a ví, že má schopnosti úkol vykonat a nebrání mu v tom například dříve přijaté závazky.

3 Multiagentní systémy podle FIPA specifikací

FIPA je nezisková organizace zaměřená na vytváření standardů pro heterogenní multiagentní systémy. Tato organizace vytvořila do roku 2004 sadu standardizačních dokumentů, které definují principy výstavby heterogenních multiagentních systémů. Heterogenitou se myslí různé způsoby implementace agentů, které tvoří jeden systém, pokud sdílejí komunikační jazyk, protokoly či přístupy k službám. Za agenta je zde považován prvek systému, který v něm provádí nějakou činnost a během ní je schopen komunikovat s ostatními agenty a sjednávat služby, které je schopen a ochoten vykonat, nebo které požaduje vykonat. FIPA specifikace se tedy nezabývají konkrétní implementací agentů, ale vytváření prostředí, ve kterém tito agenti mohou pracovat.

Vedle agentů samotných jsou důležitou součástí takového systému i agentní platformy. Ta zajišťuje agentovu existenci v systému, řídí jeho životní cyklus a zajišťuje komunikaci mezi jednotlivými agenty. Komunikaci mezi jednotlivými částmi systému (včetně komunikace mezi agenty a platformami) probíhá v nějakém agentním jazyce. Následující kapitoly představí jednotlivé tyto části systému a základy jejich fungování a interakcí.

3.1 Agent v systému FIPA

Agent je obvykle chápán, jak jsme uvedli v kapitole 2, jako autonomní, reaktivní a flexibilní jednotka, schopná vlastní aktivity za účelem dosahování aktuálních cílů. V rámci tohoto tutoriálu bude agentem jednotka, která autonomně (samostatně) jedná v systému, je schopna komunikovat v jazyce srozumitelném všem ostatním agentům a jeho aktivity jsou zaměřeny na přijímání závazků na vykonání určité služby. FIPA se zabývá pouze vnějšími projevy agentů. Neřeší vnitřní rozhodovací mechanismy agentů či reprezentace prostředí. Na druhou stranu neřeší ani problémy organizačního charakteru, jako je například společné plánování nebo formování koalic. Agent je v systému identifikován svým jménem a adresou a pro své fungování včetně komunikace v systému využívá služeb platforem.

3.2 Abstraktní FIPA agentní platforma

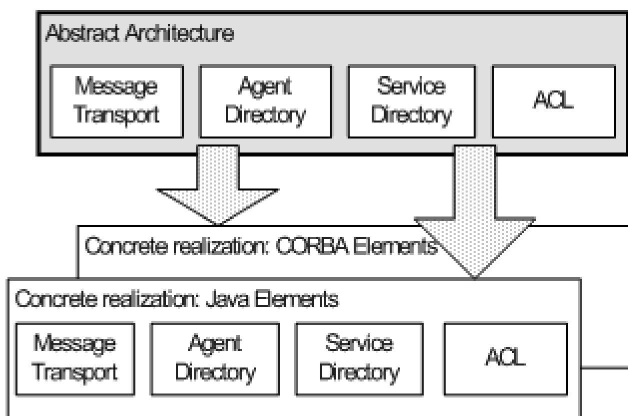
Pod pojmem agentní platforma se rozumí softwarové dílo běžící na nějaké hardwarové architektuře nad nějakým operačním systémem, které má schopnost agenta přijímat, řídit jeho činnost a také nabízí sadu služeb včetně služeb umožňujících registrace agentů, jejich služeb, komunikaci a migraci agentů v systému. V součinnosti s nějakou agentní platformou agent tráví celou dobu své existence. První z FIPA specifikací se věnuje právě agentním platformám a popisuje, jak by taková architektura měla vypadat, jaké služby zaručovat a jaké funkčnosti/operace by měly jednotlivé služby nabízet. Na obrázku 1 je znázorněna struktura FIPA abstraktní platformy z hlediska její funkčnosti ve formě služeb a naznačeny možné realizace jejich abstraktních částí.

3.3 Služby v systému

Službou se rozumí proces, který může za daných podmínek proběhnout a podporovat činnost agenta, nebo nějaké jiné služby. Na venek má služba množinu chování vůči okolí a služby mohou být poskytovány agenty, nebo i implementovány v neagentních prvcích,

například v platformách samotných. Ke službám se přistupuje přes vstupní body služeb (service-directory-entry), které jsou uloženy v adresářích služeb (service-directory-service).

Na následujícím obrázku jsou ukázány jednotlivé vyžadované funkčnosti, které v sobě zahrnují služby, jenž by měla realizace FIPA platformy zaručovat.



Obrázek 1 Složení abstraktní FIPA agentní platformy její implementace [2]

Jednotlivé abstraktní moduly zobrazené na obrázku si představíme blíže:

Adresář služeb / Service Directory: Zajišťuje službu Service Directory Service (SDS). Tato služba slouží pro vyhledávání služeb, které platforma poskytuje. Může se jednat jak o služby platformy, tak o služby na platformě zaregistrované od agentů nebo i jinak. Každá SDS obsahuje vstupní body služeb, včetně jednoho odkazujícího na sebe jako na službu.

Adresář agentů / Agent Directory: Zajišťuje službu Agent Directory Service (ADS). ADS služba udržuje informace o registrovaných agentech, resp. o jejich vstupních bodech. Každý agent je na nějaké takové službě registrován a je jej možné vyhledávat ostatními agenty podle jeho jména.

Agentní komunikační jazyk (ACL): Komunikační jazyk a jeho implementace v platformě je vyžadována pro to, aby mohlo docházet k iteracím mezi agenty. Jazyky pro komunikaci mezi agenty budou blíže představeny v kapitole **XX**.

Message Transport: Zaručuje přenos zpráv mezi jednotlivými částmi systému pomocí služeb přenosů zpráv (Message Transport Services). Přenos zpráv musí být zaručen jak mezi agenty, kteří se nacházejí uvnitř jedné platformy, tak i mezi agenty na různých platformách. Každý agent je spojen s jedním modulem zaručujícím přenosové služby. Mimo služeb samotného přenosu by měl zaručovat i službu pro kódování do patřičné reprezentace zpráv podle daných protokolů nazývanou Encoding Service (ES).

3.4 Identifikace agentů a služeb v systému

Adresářové služby uchovávají vstupní body agentů a služeb, které pak využívají pro své operace. Tyto operace zmíníme později, nyní ale ukážeme, jak takové vstupní body pro agenty a pro služby, respektive jejich struktura a jednotlivé parametry vypadají.

Vstupní bod služeb

Vstupní bod služby (Service-directory-entry), který je registrován v adresářích na platformě, obsahuje jméno služby, typ služby, lokátor služeb a případné další atributy, jako třeba cena za provedení služby, ontologie spojená s danou službou apod. Typ služby vymezuje skupinu služeb a tím mimo jiné také vymezuje jmenný prostor služeb. Může se jednat například o přenosové služby, adresářové služby, služby vztahující se k bezpečnosti systémů atd. Jméno služby musí být unikátní v systémových adresářích a identifikuje službu. Dále je rozlišen typ služby a lokátor, který obsahuje jeden nebo víc popisů lokátorů (Service-location-description). Tyto popisy se opět skládají z několika složek a to z adresy služby, typu signatury a signatury. Adresa je z adresového prostoru podle určeného typu služby. Například pro ADS je to TCP/IP adresa a port. Signatura a její typ udávají, jakým způsobem služby zpřístupnit. Typ signatury udává, o jaký typ přístupu se jedná a signatura je přístup k službám u tohoto typu, například název metody.

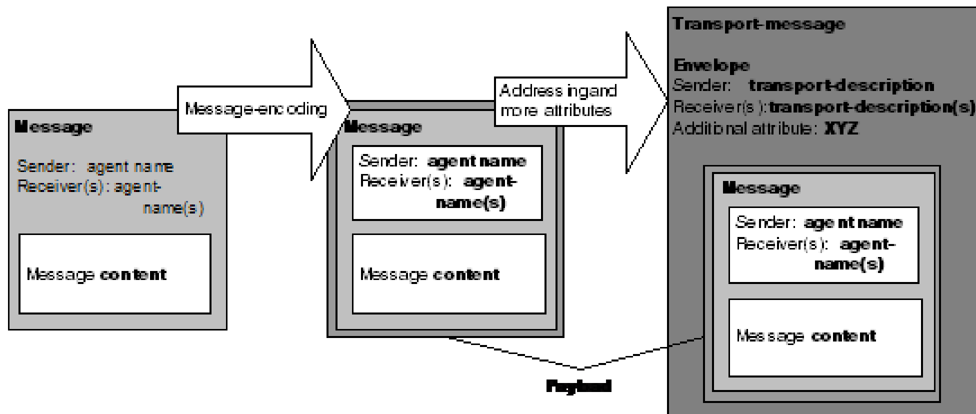
Vstupní bod agenta

Obdobně jako je tomu u služeb, i u agenta je vstupní bod struktura, která se skládá z jména agenta a lokátoru agenta. Dále lokátor je složen z jednoho nebo více popisů přenosu (transport-description), které určují, kde a jak kontaktovat agenta. Přenos je popsán typem transportu (transport-type) a specifickou adresou (transport-specific-address) pro daný typ přenosu. Dále může být součástí popisů přenosu i přenosově specifické vlastnosti, například pro typ přenosu http je adresou URL adresa, kde se agent daného jména nachází. Úkolem služby MTS je pak doručení zpráv agentu na tyto adresy.

3.5 Přenos zpráv v systému

Komunikace mezi platformami nebo i mezi agenty v rámci jedné platformy probíhá předáváním řečových aktů kódovaných v jazycích KQML [14][15] nebo FIPA-ACL, které budou blíže popsány v kapitole 4. Než se ovšem k těmto jazykům a způsobu komunikaci nimi dostaneme, budeme se alespoň stručně zabývat komunikaci předáváním zpráv na úrovni jejich přenosu.

Na této úrovni dochází k přenosu transportních zpráv a to je úkolem právě platformních služeb MTS. Transportní zpráva (transport-message) musí být příslušně zakódovaná vzhledem k použité sadě protokolů a její struktura je uvedena na obrázku 2. Jako celek je tato zpráva tvořena obálkou a samotným obsahem zprávy. Obálka obsahuje identifikátor příjemce, odesilatele, datum odeslání, reprezentace obsahu, která může být ve formách řetězce, XML formátu, nebo bitově komprimována a další atributy. Na obrázku 2 je také schematicky zachycen proces překladu původní zprávy agenta do formátu obsahu zprávy a její zapouzdření do obálky. MTS ze vstupních bodů příjemce určí typ přenosu a adresu specifickou k tomuto typu přenosu, což následně využije pro sestavení obálky. Typ přenosu také určí způsob kódování původní zprávy na obsah transportní zprávy. To vše pak umožní doručení zpráv správnému adresátovi v systému.



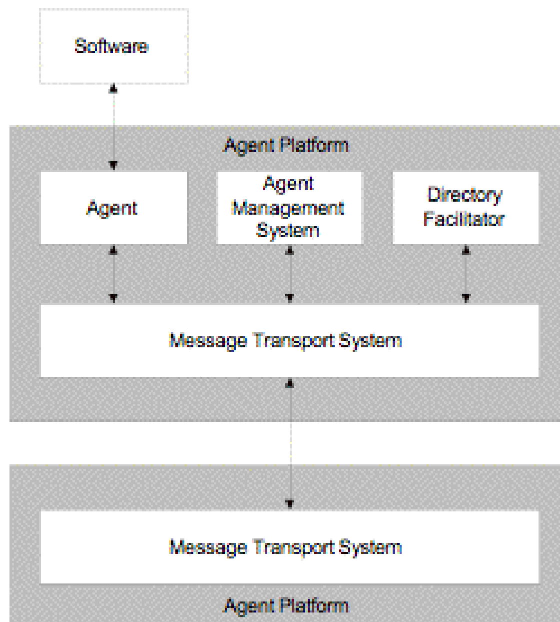
Obrázek 2: Struktura transportní zprávy a process jejího vytvoření [2]

3.6 Životní cyklus agenta na platformě.

Abstraktní FIPA architektura je v dalších specifikacích dále upřesněna detailnějším popisem fungování agenta na platformě a jeho životního cyklu na ní. Připomeňme, že agenti jsou spojeni s platformou, která v minimální verzi řídí jejich životní cyklus v této platformě od příchodu agenta na platformu nebo jeho vytvoření po odchod agenta nebo jeho ukončení, nabízí služby adresářového typu a zajišťuje komunikaci v agentním systému. Pro demonstraci fungování agenta na platformě jsou na obrázku 3 uvedeny části jako systém pro řízení agenta na platformě (Agent Management Service, AMS), modul adresářových služeb (Directory Facilitator, DF) a systém pro přenos zpráv (Message Transport System, MTS). Dále také bývá uváděn komunikační kanál agenta (Agent Communication Channel, ACC) používaný pro interakci s ostatními částmi v systému přes služby MTS.

Upřesnění také dochází v identifikaci agentů v systému. Jmenný prostor respektuje současný stav adresování, například http a URL adres a umožňuje jejich využití pro adresaci agentů v systémech respektující výše uvedené uspořádání s agentními platformami. Při komunikaci se v těchto systémech používá agentní identifikátor (AID). Tedy agent je v systému identifikovatelný svým agentním identifikátor (AID). AID jsou ve formě dvojic parametr/hodnota a obsahují

- *jméno:* jednoznačné pojmenování agenta, které je obvykle sestaveno ze jména agenta jako takového a jména platformy, na které byl agent vytvořen (Home Agent Platform, HAP), které jsou odděleny zavináčem. Tento parametr je povinný a nemění se po celou dobu existence agenta v systému.
- *adresa*: transportní adresy (například URL), kam má být zpráva doručena. Adresa, pokud je uvedena, je ve formátu, který odpovídá některému z protokolů podporovaných MTS platformy. Může být použito více transportních adres pro různé transportní protokoly.
- *vyhledávací body (resolvers)* : služby, které mohou poskytnout transportní adresu pro daného agenta. Na těchto adresách lze protokolem ‘request’ požadovat vykonání vyhledávací služby.



Obrázek 3: Modlulární model agentní platformy [3]

Agent tedy nemusí mít udanou transportní adresu přímo, ale může pouze odkazovat na služby, obvykle AMS platforem, kde by mělo být možné agenta dohledat a získat jeho transportní adresu.

Služby řízení agenta (Agent Management System, AMS)

Každá agentní platforma musí obsahovat právě jeden AMS. Agent se registruje na platformě u AMS platformy a jeho AID je na tomto AMS uchováváno. Zaregistrovaný agent obdrží přístup ke komunikačnímu kanálu platformy ACC. Platforma může zahrnovat několik fyzických stanic a tak pro určení přesné pozice agenta je potřeba vedle jeho jména znát i jeho lokátory včetně transportních adres. Pokud je možné agenty přemisťovat v rámci systému, pak tato mobilita je opět vykonávána přes AMS. Vyhrazeným AID pro nějakou platformu je

```
(agent-identifier
    :name aid@hap_name
    :addresses (semence hap_transport_address))
```

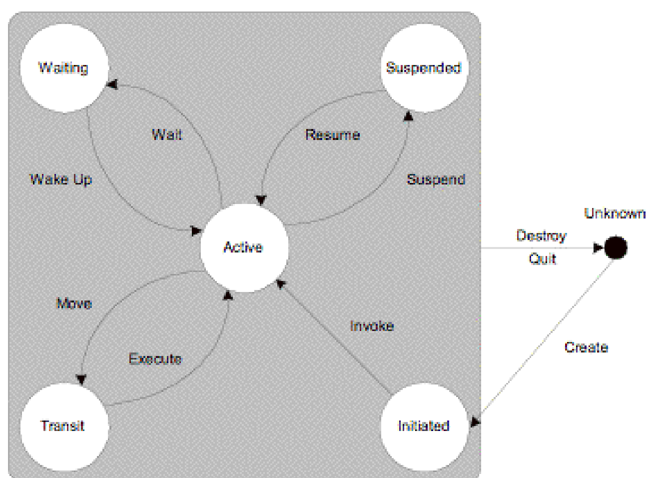
kde hap_name zastupuje ‘hap’ jméno platformy a jméno agenta je standardně aid. Adresa pak je adresou platformy podle použitého transportního systému. Zprostředkovatel adresářové služby pak může vykonávat operace pro registraci, vyhledávání, modifikaci a odregistrování služeb. Mezi funkce AMS patří vytváření, ukončování a přemisťování agentů. Konkrétně se jedná o operace ‚register’, ‚deregister‘, ‚modify‘ a ‚search‘. Jednotlivé argumenty, ontologie a protokoly pro tyto operace lze nalézt v [9]. AMS má dále oprávnění požadovat změnu agentních stavů například z aktivní na odložené.

Zprostředkovatel adresářových služeb (Directory Facilitator DF)

Adresářové služby slouží jednak pro agenty, kteří si míní zaregistrovat do SDS služby, které nabízí stejně tak jako pro agenty, kteří služby vyhledávají. Pokud je DF na platformě přítomen, pak jeho AID je následující:

```
(agent-identifier
    :name df@hap_name
    :addresses (sequence hap_transport_address))
```

Toto AID je obdobné jako AID AMS. Jméno agenta je ale vyhrazeno pro řetězec ‘df’. Agent tedy po příchodu na platformu a uvedení do aktivního stavu může využívat adresářové služby tohoto modulu. Stejně jako AMS musí tento modul poskytovat operace registrace, odregistrování, modifikace a vyhledávání záznamů a oba moduly také musí být schopny komunikovat registračním protokolem ‚subscribe’ [4]



Obrázek 4: Životní cyklus agenta na platformě [3]

Stavy agenta na platformě

Z obrázku 4 jsou vidět možné stavy agenta během jeho existence na platformě. Jak již víme, pokud chce agent fungovat na platformě, musí se registrovat na AMS a dokud jej AMS neuvede do aktivního stavu, je ve stavu inicializovaném. Agent je aktivní, pokud jeho činnost není pozastavena, pokud není ve stavu čekání například na zprávu nebo na dokončení nějaké služby, nebo pokud není přenášek na jinou platformu. Z pohledu fungování platforem je pro jednotlivé stavy odlišným způsobem prováděno doručování zpráv. Pro stavy Waiting / Suspended / Iniciated jsou zprávy, které přichází agentovi uchovávány ve vyrovnávací pamětí, zatímco zprávy pro agenta ve stavu Active jsou agentu doručovány přímo. Pokud je agent ve stavu Transit, můžou být stejně jako v neaktivních stavech zprávy umísťovány do vyrovnávací paměti, nebo přeposílány na koncové místo agentova přenosu, což bude opět nějaká platforma se službami přenosu zpráv.

4 Komunikace v MAS

Ačkoli komunikace včetně komunikačních jazyků, protokolů či ontologií je také součástí FIPA specifikací, vyhradíme tomuto tématu samostatnou kapitolu. Jednak některé části, jako třeba jazyk KQML či teorie řečových aktů se vymykají rozsahu FIPA specifikací, jednak také výklad o komunikaci na úrovni řečových aktů provedeme důkladněji a rozsah bude odpovídat samostatné kapitole.

4.1 Teorie řečových aktů

Předávání zpráv v rámci distribuovaných systémech je z pohledu multiagentních paradigmat chápáno jako provedení tzv. řečového aktu. Totiž komunikace a jednotlivé úkony během komunikace jsou považovány za stejný akt agenta vůči okolí, jako by byl agentem proveden externí akt některým ze svých efektorů. Každá zpráva předávaná v MAS mezi agenty je chápána jako jeden z typů řečových aktů (tzv. performativ) a typ upřesňuje význam zprávy. Jako příklad uveďme obsah zprávy daný predikátem `dvere(koupelna, zavrene)`. Taková zpráva může mít různý dopad po obdržení agentem pro různé typy řečových aktů. Pokud bychom rozlišili tři typy aktů – dotaz, informování a prosbu, tak společně s těmito typy by agent mohl buďto vznášet dotaz, zdali jsou dveře od koupelny zavřené (a očekávat v rámci interakčního protokolu dotazováni odpověď ano/ne/nevím), nebo by se mohlo jednat o informaci, že jsou dveře od koupelny jsou zavřené a nebo by šlo o prosbu, aby agent-příjemce dveře do koupelny zavřel. Součásti zprávy by kromě obvyklé identifikace příjemce a odesilatele by měl tedy být obsah, typ řečového aktu a dále také určení jazyka, ve kterém je zapsán obsah zprávy, protokol, v rámci kterého se komunikace odehrává atd.

4.2 Jazyk Knowledge Query Manipulating Language (KQML)

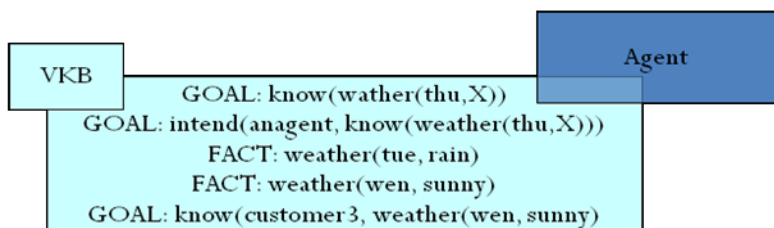
KQML je jedním z prvních jazyků, který je obvykle uváděn jako základní pro další komunikační jazyky v MAS. Díky němu je možné formulovat komunikační akty a předávat je v rámci systému. KQML byl představen na modelu, který tvoří skupina agentů propojených komunikačními kanály a každý agent má virtuální bázi znalostí (Virtual Knowledge Base, VKB). VKB obsahuje znalosti a cíle agenta ve formě predikátů a tyto znalosti a cíle tvoří mentální stav agenta. Na základě vykonání řečových aktů se pak stav této báze může měnit. Příklad na obrázku XX ukazuje agenta, který má cíle znát počasí ve čtvrtek, mít záměr informovat jiného agenta o stavu čtvrtečního počasí apod.

Struktura KQML zprávy

Ze syntaktického pohledu by KQML zpráva je podobná seznamu v jazyce LISP. Jedná se o strukturu začínajícím identifikací typu řečového aktu následovanou dvojicemi parametr – hodnota oddělených dvojtečnou. Syntaxi KQML ilustrujme na následujícím příkladě:

```
(inform
        :sender (agent-identifier:name i)
        :reciever (agent-identifier:name j)
        :content "weather(today, raining)"
        :language PROLOG
)
```

Uvedená zpráva je odeslána agentem s identifikátorem '*name i*' agentovi s identifikátoren '*name j*' a jedná se o informaci, že v aktuální den prší zapsanou jako predikát v jazyce PROLOG. Vedle uvedených čtyř parametrů zprávy počítá KQML specifikace s dalšími možnostmi parametrů, jako jsou ontologie, nebo identifikace zprávy či protokolu, ke které se tento řečový akt vztahuje.



Obrázek 5: Model mentálního stavu agenta pro KQML

Typy řečových aktů pro KQML

KQML má vymezenou množinu typů řečových aktů, které mohou být během komunikace mezi agenty použity. Jednotlivé typy řečových aktů by se daly rozdělit do několika kategorií:

- Informační typy zpráv: *tell*, *deny*, *untell*. Tyto zprávy mohou měnit stav virtuální báze agenta, který zprávy přejímá, pokud tomu odpovídá mentální stav příjemce. Je zde zajímavé, že kromě podání informace a možným popření informace je k dispozici i typ řečového aktu, který má mít za efekt 'zapomenutí' informace příjemcem.
- Práce s virtuální bází znalostí: *insert*, *delete*, … Přímým přístupem do VSK ostatních agentů dochází k jejím modifikacím.
- Odpovědi jako *error*, *sorry* následující dotazy na znalost, nebo požadavky na provedení nějakých služeb.
- Dotazy *ask-if*, *ask-about*, *evaluate* mají zapříčinit získání znalosti o pravdivosti nějaké formule nebo o stavu nějakého objektu.
- Proud zpráv je vytvářený akty *stream-about, stream-all, eos.* První dva akty zahajují proud zpráv o nějakém subjektu, třetí z nich proud zpráv ukončuje.
- Typy zpráv pro řízení sítě jako *forward, broadcast, pipe …*

Další skupinou řečových aktů jsou takové, které se používají, pokud agent využívá ke komunikaci nějakého prostředníka, jenž zprostředkovává služby nebo předává požadavky dalším agentům v systému. Tento prostředník je v KQML nazýván *broker*.

Brooker jako zprostředkovatel komunikace

Předcházející typy řečových aktů by byly použity, pokud by přímo komunikoval jeden agent s druhým. Pokud je během komunikace použit prostředník komunikace, pak má v KQML modelu za úkol řídit komunikaci, předávat a směrovat zprávy, případně zprostředkovávat vykonání nějakého požadavku. Dále má také za úkol uchovávat databázi adres agentů v systému a spravuje registr služeb poskytovaných agenty. Tedy do jisté míry

je zde podoba s FIPA platformami a adresářovými či AMS službami, které poskytují. Jelikož je ale KQML staršího data než FIPA, lze inspiraci mezi oběma systémy hledat spíše od KQML k abstraktním FIPP platformám nežli naopak.

Komunikace s prostředníkem pak probíhá následovně: Agent ve zprávě zasílané prostředníkovi požaduje zpracování služby nebo nalezení vhodného agenta (příp. agentů), který je schopen služby zpracovat. K tomu má možnost použít řečových aktů některého z následujících typů:

- *brooker-one* (*brooker-all*) – zprostředkovatel předá žádost vhodnému agentovi a posléze sděluje výsledek zpracování úlohy žadateli.
- *recommend-one* (*recommend-all*) – v tomto případě zprostředkovatel předá žádost s odkazem na žadatele. Agent, který žádost zpracoval, předá její výsledek přímo žadateli (bez dalšího zapojení zprostředkovatele).
- *recruit-one* (*recruit-all*) – jedná se pouze o žádost na nalezení vhodného agenta. Zpracování služby je na žadateli – zprostředkovatel plní pouze funkci „zlatých stránek".

Rozdíl mezi verzemi *one* a *all* je v tom, že v případě řečových aktů s uvedeným '–one' na konci bude výsledkem pouze jeden agent resp. jedna možnost výsledku na provedení služby (například při vyhledávání), kdežto v případě typů aktu s uvedeným ‚–all' na konci se bude jednat o všechny nalezené výsledky na žádost.

Slabé stránky KQML

Ačkoli je KQML doposud využíván například v robotice, lze v něm nalézt několik slabých míst, které limitují jeho vhodnost pro použití v multiagentních systémech. Jedná se zejména o následující problémy:

- Některé z typů řečových aktů mají spíše charakter **operací s virtuální bází** znalostí, nebo komunikačními kanály. Přímý přístup do struktur jiného agenta je přitom porušením jedné ze základních vlastností agenta a tou je jeho nezávislost – autonomita. Agent by měl měnit své mentální postoje výhradně vlastními procesy.
- Postrádá **formální specifikaci** jednotlivých typů zpráv. Jednotlivé řečové akty nemají definovanou sémantiku a jsou představeny pouze intuitivně.
- Některé typy řečových aktů se ukázaly být nedůležité. Jiné naopak se ukázaly jako potřebné a přibyly v následníku tohoto jazyka, tj. v jazyce FIPA-ACL.

4.3 Jazyk Agent Communiaction Language podle FIPA (FIPA-ACL)

Jazyk FIPA ACL [5] je součástí FIPA specifikací a jsou v něm odstraněny výše uvedené nedostatky. Svojí syntaxí a i řadou typů řečových aktů vychází z KQML. Stejně jako u KQML je řečovým aktem opět struktura uvádějící typ řečového aktu, partnery v komunikaci, obsah řečového aktu a další atributy.

Struktura řečového aktu FIPA-ACL

Syntax FIPA-ACL zprávy se je téměř shodná se syntaxi uvedené u KQML jazyka. Forma dvojic parametr – hodnota hierarchicky uspořádané a vymezené obyčejnými závorkami je zavedena pro všechny zápisy včetně ontologických struktur, nebo i jazyků navržených pro zápis obsahu řečového aktu, jako je FIPA Semantic Language. http://www.fipa.org/specs/fipa00030/SC00030H.html Více o tomto jazyku v [6]. Dalšími možnými jazyky pro zápis obsahu zpráv jsou například KIF, RDF, nebo PROLOG či jiné deklarativní jazyky.

V následující tabulce jsou uvedeny všechny povolené parametry, které mohou být součástí řečového aktu.

Parametr	Význam
Performative	Typ zprávy
Sender	Identifikace odesilatele
Receiver	Identifikace příjemce
Reply-to	Identifikace agenta, který má obdržet odpověď
Content	Obsah zprávy.
Language	Jazyk obsahu zprávy
Encoding	Specifikace kódování obsahu
Ontology	Ontologie
Protocol	Interakční protokol
Conversation-id	Identifikátor konverzace
Reply-with	Identifikátor, kterým má být označena odpověď
In-reply-to	Identifikátor odpovědi
Reply-by	Časové vymezení (do kdy agent čeká odpověď)

Řečové akty v FIPA-ACL

V dokumentech FIPA je uvedeno celkem 22 možných typů řečových aktů. V následujících dvou tabulkách jsou tyto typy uvedeny s tím, že nejprve uvedeme takové typy řečových aktů, které souvisí s předáváním informací mezi agenty, žádostmi o informace apod.

Typ k. aktu	Význam
Confirm	Agent potvrzuje, že informace, o které si druhý nebyl jist pravdivostí, pravdivá je.
Disconfirm	Agent nepotvrzuje pravdivost informace.
Inform	Agent informuje druhého o pravdivosti nějaké informace.
Inform If	Akt, kterým agent informuje druhého, zdali nějaká informace pravdivá je, nebo není
Inform Ref	Agent informuje druhého o stavu objektu, na který byl dotazován

Query If	Agent se táže druhého, zdali je obsah zprávy pravdivý.
Query Ref	Agent žádá druhého o informování o stav objektu, specifikovaného přiloženou referencí
Subscribe	Agent žádá druhého, aby jej informoval o stavu objektu specifikovaného přiloženou referencí a potom i pokaždé, když se stav tohoto objektu změní.

Druhou skupinou budou takové typy řečových aktů, které souvisí se sjednáváním kontraktů a závazků, požadováním vykonání služeb a také s příslušnými odpověďmi a informacemi o výsledcích vykonaných služeb.

Typ k. aktu	**Význam**
Accept Proposal	Přijmutí předchozího požadavku na vykonání akce.
Agree	Souhlas s provedením akce v budoucnu.
Cancel	Agent informuje druhého, že již netrvá na provádění domluvené akce druhým.
Call for Proposal	Požadavek na podávání návrhů provedení nějaké akce
Failure	Agent informuje druhého, že se pokusil vykonat smluvenou akci, ale neúspěšně.
Not Understood	Agent sděluje, že nerozuměl předchozí zprávě.
Propagate	Požadvek, aby zpráva byla poskytnuta prostřednictvím příjemce dalším agentům.
Propose	Agent dává návrh, že provede nějakou akci společně s případnými podmínkami, za kterých akci provede.
Proxy	Agent žádá druhého, aby určil skupiny agentů na základě přiložených podmínek a zaslal jim vloženou zprávu.
Refuse	Agent odmítá vykonat akci a přikládá vysvětlení důvodů odmítnutí.
Reject Proposal	Agent odmítá návrh druhého během vyjednávání.
Request	Agent žádá druhého o vykonání nějaké akce.
Request When	Agent žádá druhého o vykonání nějaké akce, pokud nastane uvedená podmínka.
Request Whenever	Obdobné jako předchozí s tím, že agent žádá druhého o vykonání akce pokaždé, když uvedená podmínka nastane.

Na závěr této podkapitoly je vhodné uvést to, že za agenta schopného komunikovat v jazyce ACL je považován ten, který nezávisle na protokolech a mentálních stavech je schopen odpovědět na přijatou zprávu zprávou řečového aktu ‘nerozumím’ (not_understood).

Sémantika FIPA-ACL řečových aktů

Specifikace uvádí pro každý z řečových aktů jeho sémantiku a to pomocí formulí modální logiky. Tato logika zavádí operatory pro představy (belief) a nejisté představy (uncertainity) vzhledem k nějaké formuli. Do jisté míry souvisí tato logika s BDI logikou navrženou Raoem a Georgefem [23]. Celkem jako modální operátory jsou použity následující operátory:

- $\mathbf{B}_i f$ agent *i* věří v platnost formule *f*.
- $\mathbf{U}_i f$ agent *i* si není si jistý, zdali platí f, ale věří, že spíše platí *f*, nežli $\neg f$ (má představu o platnosti).
- $\mathbf{C}_i f$ agent i si přeje, aby platila formule *f*.
- $\mathbf{Bif}_i f \equiv \mathbf{B}_i f \vee \mathbf{B}_i \neg f$ agent *i* věří v platnost formule *f*, nebo platnost její negace.
- $\mathbf{Uif}_i f \equiv \mathbf{U}_i f \vee \mathbf{U}_i \neg f$ agent *i* má představu o platnosti formule *f*.

Oproti klasické BDI logice je v těchto standardech použit operátor pro ‘mění nejisté představy’ o platnosti či neplatnosti nějaké formule. V následujícím příkladu specifikace sémantiky řečového aktu budou tyto operátory použity a jejich význam blíže objasněn.

Každý z typů řečových aktů má uvedenu podmínku použití řečového aktu a jeho racionální efekt na příjemce zprávy po obdržení aktu. Předpoklad pro provedení (Feasible Precondition, FP) se značí jako $\mathbf{FP}(a_i)$ a reprezentuje podmínku provedení aktu a_i. Racionální efekt (Rational Effect, RE), značený $\mathbf{RE}(a_i)$, je racionální efekt aktu a_i. Pro nedeterministický výběr aktů $a=a_1|a_2|\ldots|a_n$ jsou příslušné formule ve tvarech $\mathbf{FP}(a)=a_1 \vee a_2 \ldots \vee a_n$ a $\mathbf{RE}(a)= a_1 \vee a_2 \ldots \vee a_n$.

Při budování multiagentního systému s agenty komunikujícícmi jazykem ACL mají být tyto podmínky patřičně implementovány. Pro ukázku uvedeme, jak je specifikována sémantiky typu aktu INFORM a vysvětlíme tuto specifikaci.

. Bude se tedy jednat o informování nějakého agenta *i* jiným agentem *j* o platnosti nějaké formule *f*. Pro agenta i a řečový akt `inform` s argumenty *j*, ϕ jsou uvedeny následující zápisy:

```
<i, inform (j, ϕ )>
```

Je přepokladem provedení formule

$$FP: B_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$$

Která má ten význam, že agent *i* věří, že platí formule *f* a má představu, že agent *j* nemá ani víru v platnost a ani představu o platnosti formule ϕ.

Racionální efekt provedení tohoto řečového aktu je pak definován jako

$$RE: B_j\phi$$

tedy že agent *j* bude věřit v platnost formule ϕ.

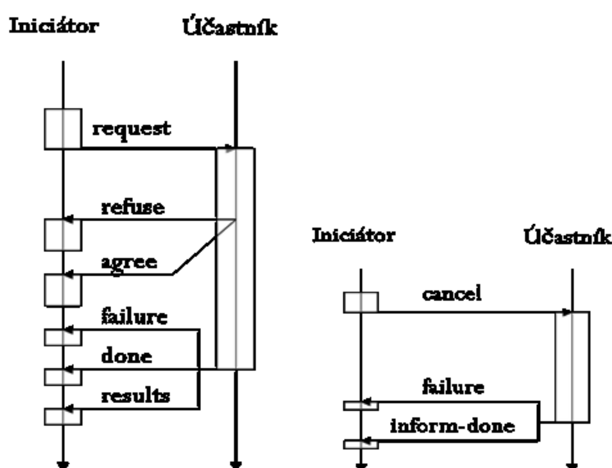
Obdobným způsobem jsou definovány všechny typy řečových aktů. Nyní ale přejdeme k definicím některých komunikačních protokolů v FIPA specifikacích.

4.4 Protokoly pro vyjednávání a sjednávání kontraktů

Specifikace FIPA představují hned několik protokolů pro komunikaci mezi agenty. Jedná se o protokoly pro žádost o poskytnutí informací, služeb, protokoly pro registrování služeb, Holandské a Anglické dražby a také o protokol kontraktní sítě pro sjednávání kontraktů mezi agenty. Některé z nich představí následující kapitoly.

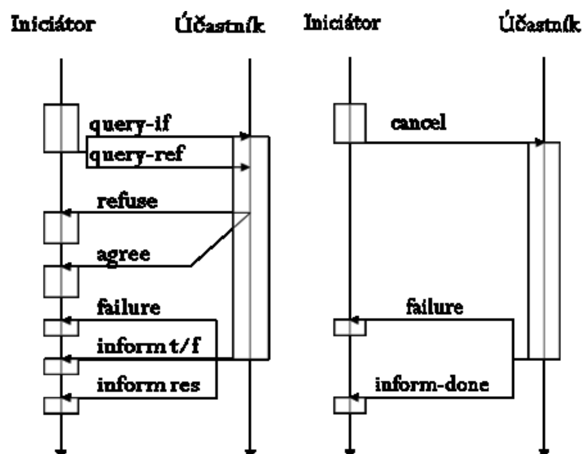
Protokoly požadavků služeb a dotazování

V obou případech, tj. při dotazování a požadování, se jedná o velmi podobné protokoly. Vždy v nich vystupují dva agenti, přičemž jeden z nich je iniciátor, který vznáší požadavek na službu nebo informaci a druhým je účastník, který službu či sdělení dotazu odmítne vykonat, nebo vykoná. Jednotlivé protokoly jsou znázorněny pomocí sekvenčních diagramů na následujících obrázcích.



Obrázek 6: Protokol REQUEST a, podání žádosti b, zrušení žádosti [7]

Iniciátor, tedy agent, který požaduje nějakou službu, provede komunikační akt typu *request* vůči agentům, od kterých očekává, že by mohli požadavek vykonat (například byli vyhledání pro daný popis služby přes adresářové služby platforem). Odpovědí může být souhlas s provedením nebo odmítnutí provedení dané služby. Pokud byl poskytnut souhlas s vykonáním služby, pak iniciátor obdrží po čase ještě informaci o výsledku provedení služby. Iniciátor může také zrušit svůj požadavek (obr. 6b) na provedení služby a obdrží odpověď, zdali tím služba skončila, nebo zdali nemohlo být zrušení úspěšně provedeno.



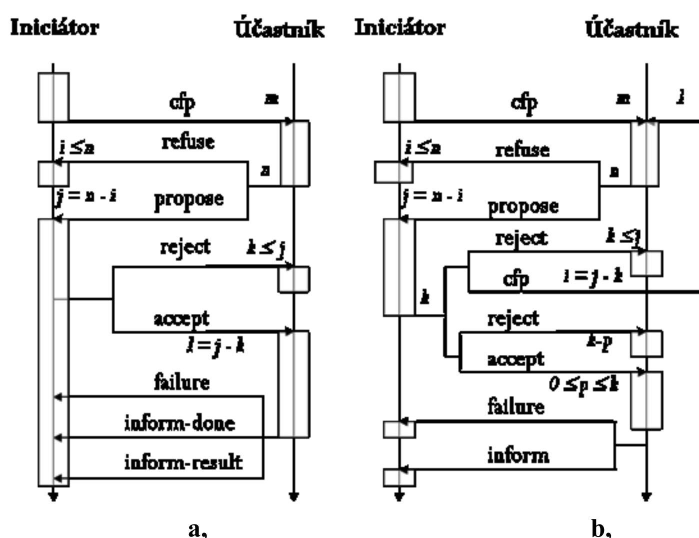
Obrázek 7: Protokol QUERY a, vznesení dotazu b, zrušení žádosti [8]

Protokol dotazování se velmi podobá předchozímu protokolu. Rozdílem je pouze to, že iniciace je provedena jinými řečovými akty a že pokud účastník přijme požadavek iniciátora, a pokud toto zpracování neselže, pak po zpracování dotazu zašle účastník zpět iniciátorovi komunikační akt typu *inform* s obsahem buďto pravda/nepravda, nebo s nějakým popisem, který je obsahem tohoto komunikačního aktu, podle toho, jaký inicializační akt byl použit.

Protokol kontraktní sítě

Protokoly kontraktní sítě (Contract Net Protocol, CNP) ve svých dvou verzích jsou základním protokolem pro sjednávání kontraktů mezi agenty. Jejich postavení mezi protokoly dokresluje skutečnost, že je rovněž přítomen ve FIPA specifikacích. Jedná se o protokol pro sjednávání kontraktů mezi agentem, který požaduje službu a agentem, který službu nabízí. Oproti jednoduchým REQEST protokolům zde dochází k vyhodnocování případných nabídek a lze nabídky upřesňovat.

Na obrázku 8 jsou sekvenčními diagramy představeny oba dva protokoly kontraktní sítě. První z nich je bez opakování, druhý s opakováním vznášení požadavků. V obou dvou případech dochází k inicializaci komunikace jedním z agentů (iniciátorem), který posílá zprávu s řečovým aktem Call for Proposals (CFP). Jedná se o akt v tomto případě k *m* agentům, kteří jsou vyzíváni, aby podávali návrhy k uzavření kontraktu na vykonání požadované služby. Specifikace této služby by měla být součástí obsahu CFP řečového aktu. V obou případech odpoví agenti buďto tím, že odmítnou reagovat na CFP, a nebo nabídnou služby a podmínky, za kterých jsou ochotni tyto služby vykonat. V případě CNP bez opakování je část nabídek akceptována a je očekávána informace o výsledku provedení služeb. V cyklické verzi protokolu kontraktní sítě může iniciátor pokračovat opětovným vznesením (upřesněných) požadavků a celý proces opakovat.



Obrázek 8: Protokoly kontraktní sítě a, bez opakování [9] b, s opakováním [10]

Ontologie

Poslední podkapitolka stručně představí pojem ontologie, jelikož ta je často součástí komunikace mezi agenty. Ontologie je chápána jako slovník pro nějakou doménu zájmu, o které se vede rozhovor mezi agenty. Ontologie kromě jednotlivých entit, ze kterých se skládá předmět konverzace, může obsahovat i vlastnosti jednotlivých entit, nebo jejich vzájemné vazby. Která ontologie je během komunikace používána může být, jak již bylo uvedeno v kapitole 4.3, součástí ACL zprávy. Ve specifikacích FIPA se ontologie definuj9 například pro protokoly týkající se adresářových služeb. Obvyklou součástí ontologických specifikací jsou vedle slovního popisu jednotlivých prvků v i povinnost jejich existence, typ a rezervované hodnoty.

5 JADE: Nástroj pro implementaci FIPA systému

JADE je vývojové prostředí pro multiagentní systémy respektující standardy FIPA. Jedná se v současnosti o nejrozšířenější takové prostředí a je naprosto dominantním v této oblasti. Implementace agentů se primárně předpokládá v jazyce Java, ale jedná se o flexibilní systém, který umožní propojení s jinými vývojovými systémy (například s již zmiňovaným systémem JASON pro implementaci BDI agentů). Systém JADE je vyvíjen italskou společností Tilab a od roku 2000 se jedná o Open source software.

Součástí JADE jsou prostředí pro běh agentů, knihovna tříd pro implementaci agentů a multiagentního systému a také grafické prostředí pro správu a ladění vytvářeného systému. JADE umožňuje propojení více platforem a fungování takového propojeného systému jako celku, včetně meziplatformního posílání zpráv. V následujících kapitolách představíme principy fungování multiagentního systému v tomto prostředí a také zmíníme základní třídy pro implementaci reaktivních agentů.

JADE agentní platforma

Stejně, jak bylo uvedeno v kapitole o FIPA specifikacích, agenti mohou být registrováni na platformě přes AMS a využívat adresářových služeb DF. Pokud má agent vzniknout na platformě, nebo na ní má být přenesen, musí být zaregistrován na AMS dané platformy. AMS pak spravuje fungování tohoto agenta během celé doby jeho existence na platformě. Jako celek se platforma se může skládat z více kontejnerů platforem, tj. fyzických stanic, na kterých běží jednotlivé části platforem. Pak musí docházet k jejich propojení a vytvoření jedné virtuální platformy. V JADE je toto implementováno tím způsobem, že každý platformní kontejner je i RMI kontejner a ty vzájemně komunikují protokoly IIOP. Součástí JADE je také grafické prostředí pro snadnější správu celého systému.

Cyklus interpretace agenta

Agent v systému JADE je interpretován, jak je obvyklé, v cyklech. V každém cyklu, pokud agent nemá být ukončen, proveden jeden krok z definovaného chování agenta z nějaké množiny chování, které bylo agentu přiřazeno. Pokud tímto krokem bylo ukončeno chování agenta, je toto chování odstraněno a v dalším kroku je proveden jeden krok následujícího z definovaných chování. Programátorovým úkolem je vytvořit třídy pro chování, které bude specifické pro vyvíjeného agenta a povede k jeho racionálnímu chování vzhledem k jeho cílům, například k flexibilnímu a pohotovému reagování na požadavky a jejich následné vykonávání, pro komunikaci předepsanými protokoly adp.. Interpretační cyklus agenta je zobrazen na obrázku 9.

Implementace agentů s různými typy chování

Chování je pro agenta vytvořeno instancí některé z příslušných tříd. Ze stávajících tříd v distribucích JADE lze definovat jednoduché chování, které proběhne během jednoho cyklu a slouží pro přiřazení jednorázové akce do agentova portfolia chování. Pokud by chování mělo trvat po vice implementačních cyklů, pak lze odvozovat z tříd pro cyklická chování. Pak v každém z cyklů dojde k vyvolání příslušné metody chování, dokud není toto chování zablokováno. Případ jednoduchého chování může být takový, kdy vyvolání akce z tohoto chování proběhne jednou po uplynutí určitého počtu interpretačních cyklů, nebo proběhne po obdržení nějaké zprávy. Speciálním typem cyklického chování je takové, které proběhne vždy po uplynutí určeného počtu cyklů. Také je možné definovat složitější chování, jako je sekvence několika chování, nebo jejich paralelní kompozice. Také pro některé protokoly, jako například CNP, je možné definovat chování pro jednotlivé stavy protokolu a tak vytvořit chování agenta schopného komunikovat tímto protokolem.

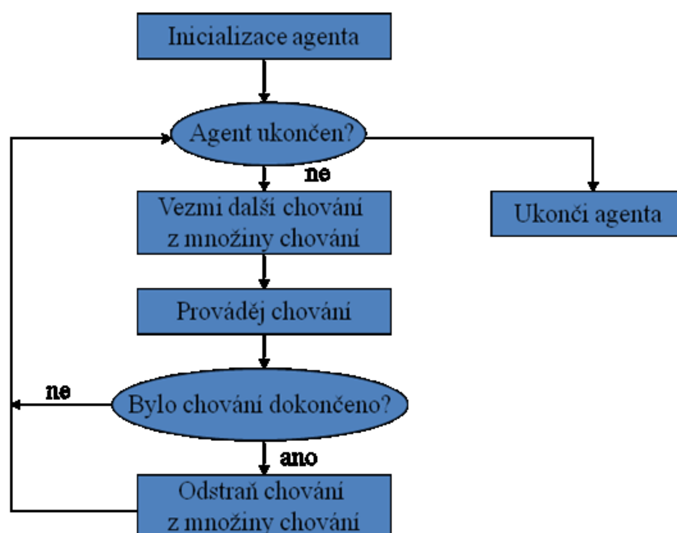
Agent „Hello Word”

Příklady implementací agentů začneme jak je obvyklé a to implementací agenta který pozdraví. Pro jeho implementaci stačí vytvořit třídu odvozenou z třídy Agent a definovat v ní spouštěcí metodu *setup()*.

```
import jade.core.Agent;

public class MyAgent extends Agent {
protected void setup() {
        System.out.println("Hello!
              Agent "+getAID().getName()+" is ready.");
    }
}
```

V této metodě vytiskneme pozdrav a ten doplníme informací o jménu agenta. Jméno je získáno z AID agenta, které zpřístupní metoda getID(). Konkrétní jméno je pak dále dosaženo metodou getName() nad AID. Pokud by byl zájem například o HAP agenta, pak by byla nad AID použita metoda getHAP().



Obrázek 9: Interpretační cyklus JADE agenta

Příklady jednoduchého a cyklického chování

Jednoduché chování agenta odvozené ze třídy OneShotBehaviour proběhne neprodleně a pouze jednou. Pro uvedený příklad bude vytvořena ACL zpráva typu Inform a přiřazen nějaký příjemce (v tomto případě neuveden). Správa je následně odeslána zavoláním přislušné metody nad objektem agenta.

```
class MyOneShotBehaviour extends OneShotBehaviour{
    public void action() {
     ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
     msg.addReceiver(...);
     myAgent.send(msg);
    }
 }
```

Druhý příklad bude cyklické chování, kdy v každém cyklu bude provedena nějaká činost, která by byla implementována v bloku „else …". To ovšem jen do té doby, dokud bude agent dostávat nějaké zprávy, to znamená, pokud metoda agenta 'receive' nevrátí konstantu 'null'.

```
class MyCyclicBehaviour extends CyclicBehaviour{
   public void action() {
    ACLMessage request = myAgent.receive();
    if (request == null){
```

```
        block();
      }
      else{
        ...
      }
    }
  }
```

Registrace služeb agenty

Registrace služeb agentů v DF probíhá podle následujícího kódu. Uvažovaný agent pro předpověď počasí v době svého spuštění vytvoří patřičný popis pro DF třídy DFAgentDescription, ve kterém nastaví jméno poskytovatele služby na své AID, dále nastaví jméno služby a požadovanou ontologii, která se pro sjednávání služby má respektovat. Nakonec vytvořený vstupní bod služby zaregistruje, k čemuž slouží statická metoda JADE třídy DFService.

```
public class WeatherForecastAgent extends Agent {
  protected void setup() {
    String serviceName = "WeatherForecast";
    try {
      DFAgentDescription dfd = new DFAgentDescription();
      dfd.setName(getAID());
      ServiceDescription sd = new ServiceDescription();
      sd.setName(serviceName);
      sd.setType("weather-forecast");
      sd.addOntologies("weather-forecast-ontology");
      sd.addOntologies(FIPANames.ContentLanguage.FIPA_SL);
      sd.addProperties(new Property("country", "Italy"));
      dfd.addServices(sd);
      DFService.register(this, dfd);
    }
    catch (FIPAException fe) {
      fe.printStackTrace();
    }
  }
```

Pokud agent končí svoji existenci na platformě, odregistruje i danou službu z DF. K tomu slouží metoda ‚deregister' pro statický objekt DFService.

```
  protected void takeDown(){
    try{
      DFService.deregister(this);
    }
    catch(FIPAException fe){
      fe.printStackTrace();
    }
  }
}
```

Propojení jiných nástrojů se systémem JADE

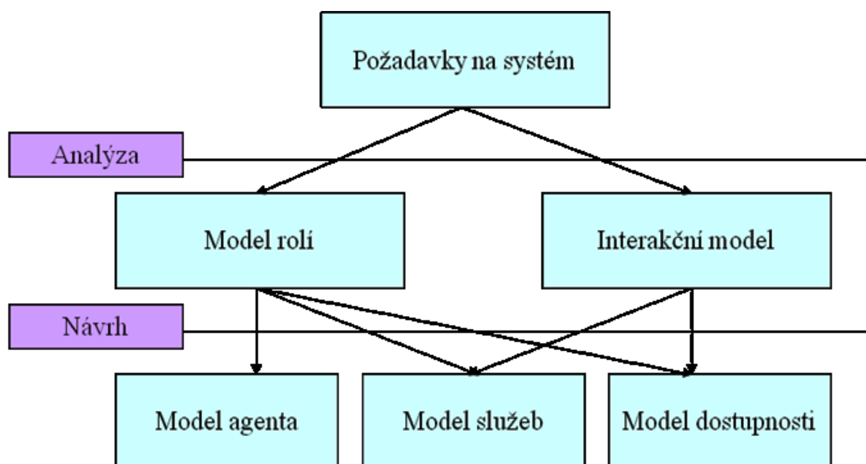
Vedle uvedených možností vývoje multiagentních systémů umožňuje JADE propojení systému s jinými (multi)agentními systémy. V současnosti je tato možnost součástí distribucí systému JASON i jiných. Lze také udělat vlastní propojení, jak bylo publikováno například v [26]. Zde lze najít způsob spojení modelovacích a simulačních nástrojů TMass a PNTalk se systémem JADE. Většina práce spočívá ve vytvoření komunikační infrastruktury přes http protokol a vytvoření MTS na straně nových systémů, skrz které se jejich agenti dokáží zaregistrovat na AMS JADE platformy a skrz tento AMS fungovat v JADE systému jako rovnocenní partneři.

6 Návrh multiagentních systému, systémy GAIA a Prometheus

Metodiky návrhu multiagentních systémů jsou přirozeně jednou z oblastí výzkumu. Mezi první z navržených metodik patřila KGR [18] pro návrh multiagentních systému s BDI agenty, které do značné míry kopírovaly princip implementace BDI agentů. Na úroveň multiagentních systémů a iterací mezi agenty, nebo skupinou agentů, se zaměřila pokročilejší metodiky GAIA [25]. Zde se již hovořilo o rolích agentů a jejich vymezení. Mezi v současnosti používané a podporované metodiky patří Prometheus [20], který se dočkal i realizace pro různá prostředí včetně prostředí Eclipse. Následující podkapitoly nejprve stručně představí některé postupy v metodice GAIA, aby se dále věnovaly návrhu multiagentních systémů v systému Prometheus.

6.1 Metodika GAIA

Hierarchická struktura modelů, které jsou vytvářeny při použití metodiky GAIA, jsou ukázány na obrázku 10. Ve fázi analýzy se jedná o identifikaci rolí agentů v systému a jejich vzájemných interakcí. Ve fázi návrhu systému se k jednotlivým rolím identifikují služby, navrhnou vnitřní rozhodovací mechanismy agenta a infrastruktura pro komunikaci agentů v systému.



Obrázek 10: Hierarchická struktura modelů v metodice GAIA

V tomto textu popišeme pojem role a jak je modelována v GAIA. Pojem role zde není původní pro oblast multiagentních systémů, ale je zde blíže specifikována pomocí souboru agentových schopností a povinností, které roli vymezují. Konkrétněji, role jsou určeny skrz následující požadavky na agenty.

povinnosti (*responsibilities*): Jsou rozlišovány povinnosti životné a chráněné. Životné povinnosti jsou takové, které by měl agent dosáhnout, pokud platí nějaké uvedené podmínky. Chráněné povinnosti jsou takové, které by měl agent udržovat po celou dobu, co drží danou roli.

oprávnění (*permissions*): uvádějí, k užívání jakých prostředků je agent zastávající danou roli oprávněn. Také lze uvést limity pro používání těchto prostředků. V textu představující tuto metodiku se uvažuje s prostředky ve formě informací a s možnými oprávněními *číst*, *generovat* a *modifikovat* tyto informace.

aktivity (*activities*): aktivity jsou akce, které má být agent schopen provádět, , pokud má danou roli v systému zastávat, a které nevyžadují komunikaci s ostatními agenty v systému.

protokoly (*protocols*): výčet protokolů, které mají agenti v dané roli být schopni sledovat během komunikace. Protokoly jsou specifikovány takto:

- *Účel protokolu*: krátký slovní popis protokolu
- *Iniciátor rozhovoru*: role, která zahajuje interakci
- *Účastníci rozhovoru*: role, které se interakce účastní
- *Vstupy*: informace, které má iniciátor v tom okamžiku, když zahajuje interakci
- *Výstupy*: informace sdělené účastníky nebo sdělené účastníkům během interakce
- *Zpracovávání*: procesy, které iniciátor provádí během interakce (zpracování informací, volba argumentů apod.)

Příkladem může být následující definice role agenta:

Role Schema: ***brooker***
Description ***„zprostředkovává služby"***
Protocol
and Activities: ***receiveRequests,searchServices,***
lookupCandidates,makeContract,informContractor
Permissions: ***reads:*** ***agents_abilities***
generates: ***contractors***
Responsibilities
Liveness:
fulfullRequest=(receiveRequests.findContract.
makeContract.informContractor)$^{\omega}$
findContractor=(searchServices.lookupCandidates)
Safety: ***true***

Agent, který má zastávat roli zprostředkovatele (brookera), musí znát protokoly pro přijímání požadavků, vyhledávání služeb, uzavírání kontraktů a informování o kontraktech. Jeho aktivitou je pak vyhledávání kandidátů ve svých adresářích. Bude mít oprávnění číst agentní schopnosti a generovat uzavřené kontrakty.

6.2 Metodika Prometheus

Ačkoli GAIA je kvalitně vybudovaná metodika, žádný nástroj, který by podporoval vývoj multiagentních systémů podle této metodiky, není v současnosti obecně rozšířen. Jak jsme

ale uvedli v úvodu této kapitoly, pro metodiku Prometheus, ktera umožňuje navrhovat multiagentní systémy s BDI agenty, takový nástroj existuje. Následující odstavce stručně představí jednotlivé fáze návrhu touto metodikou.

Úrovně návrhu systémů

Z koncepčního pohledu lze rozlišovat tři vertikální úrovně návrhu – úroveň specifikace systému, úroveň návrhu architektury a úroveň detailního návrhu jednotlivých komponent. Na úrovni specifikace systému se deklarují cíle, kterých mají agenti v systému dosahovat a dále scénáře fungování jednotlivých rolí, což dohromady dává nějaký počáteční popis funkcionalit systému. Návrh architektury pak zahrnuje návrh protokolů, deskriptorů agentů a diagram přehledu systému. Úroveň detailního návrhu je posledním krokem v návrhu systému a provádí se zde definice procesů, schopností agentů, jejich plánů a určení možných událostí, které mohou v systému nastat.

Specifikace systému

U specifikací cílů lze definovat stromovou strukturu, která cíle rozkládá na podcíle s AND/OR vazbami. Splnění nadřazeného cíle je pak dosaženo, pokud je splněn alespoň jeden, nebo naopak všechny z podcílů. Pro jednotlivé cíle a podcíle jsou pak naznačeny možné scénáře, které mohou vést k jejich naplnění. Ty jsou ve formě obecných posloupností událostí, vjemů, akcí, či podcílů. Funkcionality jsou pak části chování systémů, které jsou vytvořeny buďto podle scénářů, nebo naopak jsou identifikovány jako možné v systému a na jejich základě jsou nalezeny nové scénáře a případně i nové cíle, které jsou agenti schopni dosahovat.

Návrh architektury

Funkcionality seskupeny například podle toho, jaká data používají, nebo v jaké části systému se vyskytují, určují roli agenta. Také funkcionality, které mají vzájemnou spojitost a funkcionality, které spolu interagují, umožní role identifikovat. Na úrovni návrhu systému také je potřeba specifikovat, které role a za jakým účelem spolu komunikují a poté specifikovat protokoly, kterými nalezené role budou komunikovat. To vše poté umožní vytvořit diagram tříd, který znázorní role agentů, přidělené prostředky, protokoly, funkcionality a vzájemnou dostupnost rolí v systému.

Úroveň detailního návrhu

Posledním krokem v návrhu metodikou Prometheus je detailní specifikace vjemů, akcí, zpráv a případně dalších možných události vzhledem k jednotlivým rolím. Tyto události musí být schopen agent zastávající tyto role adekvátně zpracovávat v souladu s nalezenými funkcionalitami. Tento návrh je také zásadním podkladem pro implementace konkrétních BDI agentů. Na jeho základě má programátor za úkol vytvořit pro zjištěné události a funkcionality správné a relevantní plány, které bude agent schopen za běhu přidávat do své struktury záměrů a tak řídit své chování vzhledem k zadaným cílům. Pro správný návrh systému tak vzniknou agenti, jejichž chování v systému bude odpovídat počátečním předpokladům.

7 ZÁVĚR

Uvedená exkurze do světa multiagentních systémů měla nastínit současné možnosti a směry v implementacích systémů s agenty. V multiagentních systémech dochází k interakci

mezi agenty většinou zasíláním řečových aktů a k navazování spolupráce dochází při komunikacích protokoly pro poskytování a vyžadování poskytnutí nějakých služeb. Multiagentní systémy v současnosti mají pokud možno respektovat zveřejněné specifikace organizace FIPA minimálně ve způsobu implementací agentních platforem, komunikačních jazyků a protokolů. Jedním z nástrojů, který umožňuje tvorbu takových nástrojů je nástroj JADE. V textu byly ukázány základy způsobu implementací agentů v JADE a také byly zmíněny obrysy metodik návrhů multiagentních systémů jako takových.

8 LITERATURA

1. AFAPL2. *Agent Factory.* [Online] 2009. http://www.agentfactory.com/index.php/AFAPL2.
2. FIPA Abstract Architecture Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00001/SC00001L.html.
3. FIPA Agent Management Specification. *FIPA.* [Online] 2004. http://www.fipa.org/specs/fipa00023/SC00023K.html.
4. FIPA Subscribe Interaction Protocol Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00035/SC00035H.html.
5. FIPA Communicative Act Library Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00037/SC00037J.html.
6. FIPA SL Content Language Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00008/SC00008I.html.
7. FIPA Request Interaction Protocol Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00026/SC00026H.html.
8. FIPA Query Interaction Protocol Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00027/SC00027H.html.
9. FIPA Contract Net Interaction Protocol Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00029/SC00029H.html.
10. FIPA Iterated Contract Net Interaction Protocol Specification. *FIPA.* [Online] 2002. http://www.fipa.org/specs/fipa00030/SC00030H.html.
11. **Bordini, Rafael, Hübner, Jomi, Fred a Wooldridge, Michael.** *Programming multi-agent systems in AgentSpeak using Jason.* Wiley-Interscience, 2007.
12. **Bratman, Michael.** *Inentions, Plans and Practical Reason.* Harvard University Press, 1987.
13. *3APL: A Programming Language for Cognitive Agents.* **Dastani, Mehdi, Dignum, Frank a Meyer, John-Jules.** European Research Consortium for Informatics and Mathematics, 2003. ERCIM News.
14. **Finin, Tim, Weber, Jay a Wiederhold, Gio.** *DRAFT Specification of the KQML Agent-Communication Language.* 1993.
15. *KQML as an agent communication language.* **Finn, Tim, Labrou, Yannis a Mayfield, James.** MIT Press, 1996. Software Agents. stránky 291-316.
16. *JAM: a BDI-theoretic mobile agent architecture.* **Huber, Marcus.** New York 1999. Proceedings of the third annual conference on Autonomous Agents.
17. *The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System.* **d'Iverno, Mark, Kinny, Luck, Michael a Georgeff, Michael.** Kluwer Academic Publisher, 2004. Autonomous Agents and Multi-Agent Systems. stránky 5-53.
18. **Kinny, David, Georgeff, Michael a Rao, Anand.** Modelling and Design of Multi-Agent Systems. *ATAL 1996.* 1996.

19. **Myers, Karen.** User Guide for the Procedural Reasoning System. *Technical Report.* Menlo Park, CA : SRI International, Artificial Intelligence Center, 1997.
20. **Padgham, Lin a Winkoff, Michael.** *Developing Intelligent Agent Systems* John Wiley & Sons, 2004.
21. *AgentSpeak(L): BDI Agents speak out in a logical computable language.* **Rao, Anand.** Amsterdam : Springer-Verlag, 1996. Agents Breaking Away, Lecture Notes in Artificial Intelligence.
22. *Modeling Rational Agents within a BDI-Architecture.* **Rao, Anand.** San Mateo 1991. Proceedings of the Second International Conference on principles of Knowledge Representation and Reasoning.
23. **Rao, Anand a Georgeff, Michael.** *Formal Models and Decision Procedures for Multi-Agent Systems.* Melbourne : Australian Artificial Intelligence Institute (61), , 1995.
24. **Shoham, Yoav.** Agent Oriented Programming. *Artificial Intelligence 60.* 1993, stránky 51-92.
25. **Zambonelli, Franco, Jennings, Nicholas a Wooldridge, Michael.** Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering Methodology, 12(3).* 2003.
26. *Connecting Jade with PN agent.* **Žák, Jakub, a další.** 2010, Proceedings of Seventh EUROSIM Congress on Modelling and Simulation.

Rejstřík autorů

Dlugolinský, Štefan 1

Hluchý, Ladislav 1

Kusák, David 49

Laclavík, Michal 1

Nečaský, Martin 49

Richta, Karel 49

Srp, Jaroslav 23
Šeleng, Martin 1

Zbořil, František 75