České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Bohuslav Koukal**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Integrace systémů České filharmonie**

Pokyny pro vypracování:

Agenda provozu České filharmonie a správy Rudolfina je nyní realizována pomocí několika oddělených systémů, takže aktualizace dat v jednom systému je nutné ručně zanést do systémů ostatních. Tento způsob práce je neefektivní a je náchylný k chybám. V rámci bakalářské práce navrhněte integrační řešení pro systémy České filharmonie, které nutnost manuálního předávání dat omezí na nejnutnější minimum.

Očekávané výstupy práce:

1. Analýza stávajících systémů ČF, především definice předávaných dat, analýza rozhraní systémů a požadavky na integraci systémů.
2. Návrh softwarového řešení, které umožní sdílet aktualizace dat v systémech ČF, včetně definice nutných API pro jednotlivé systémy.
3. Implementace integračního řešení (nebo jeho prototypu) podle bodu 2 a jeho otestování.

Seznam odborné literatury:

1. J. Blanchette: The Little Manual of API Design, Trolltech, 2008
2. J.Arlow, I. Neustadt: UML a unifikovaný proces vývoje aplikací, Computer Press, 2003

Vedoucí: Ing. Ondřej Macek, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

[redacted]

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

[redacted]

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 26. 3. 2015

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra počítačů

Bakalářská práce

Integrace systémů České filharmonie

Bohuslav Koukal

Vedoucí práce: Ing. Ondřej Macek, Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský

Studijní obor: Softwarové inženýrství

22. května 2015

Poděkování

Děkuji vedoucímu této práce Ondřeji Mackovi za jeho profesionální a přátelský přístup během vedení této práce. Děkuji pracovníkům České filharmonie, že mi umožnili nahlédnout do tajů fungování IT v prostředí, kde technologie nejsou cílem, ale prostředkem. Lucii Prchalové pak děkuji za formální korektury této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. 5. 2015 ……..

Abstract

Czech Philharmonic business departments are using different IT systems for their work. Currently it is not possible to share data among the systems, so required data are transferred manually which is both inefficient and a cause of data inconsistencies.

In this work I have analyzed current state, described required data flow among the systems and designed a solution to automate the data sharing.

I have implemented the solution by developing an independent intermediate component listening to changes in current systems and distributing these changes. The component can be also easily configured so that it works properly after data flow changes or another system is added to the communication flow or after some system is removed or replaced.

Abstrakt

Oddělení České filharmonie pro své fungování používají různé IT systémy. Tyto systémy často pracují se stejnými daty, která však mezi sebou nedokážou sdílet. Potřebná data jsou proto přenášena manuálně, což je neefektivní a k chybám náchylný způsob.

V rámci této práce jsem podrobně zanalyzoval současný stav, popsal požadovaný tok dat mezi systémy a navrhnul řešení, které přenosy dat mezi systémy zautomatizovalo.

Řešení jsem realizoval pomocí implementace samostatné nezávislé komponenty, která poskytuje rozhraní pro naslouchání změnám v současných systémech a pro distribuci těchto změn. Tato komponenta je zároveň jednoduše konfigurovatelná tak, aby svou funkci plnila i v případě změny předávaných dat nebo výměny některého systému či jeho přidání nebo odebrání.

Obsah

1 Úvod ... 1
2 Analýza současného stavu ... 2
2.1 Organizační oddělení České filharmonie ... 2
2.2 IT systémy používané Českou filharmonií ... 3
2.3 Potřebné datové sady v jednotlivých systémech ... 5
2.4 Funkční požadavky ... 7
2.5 Omezení daná současným stavem ... 7
3 Návrh řešení ... 9
3.1 Architektura řešení ... 9
3.2 Návrh technologií pro vývoj centrální komponenty ... 12
3.3 Možnosti komunikace mezi komponentami ... 12
3.4 Komunikační rozhraní ... 14
3.5. Zajištění správné interpretace zpráv ... 15
3.6 Konfigurace toku zpráv ... 19
3.7 Zabezpečení komunikace ... 19
4 Implementace ... 21
4.1 Použité technologie a knihovny ... 21
4.2 Struktura aplikace ... 21
4.3 Detaily implementace ... 22
4.4 Konfigurace aplikace ... 26
5 Testování ... 29
5.1 Unit testování ... 29
5.2 Akceptační testování ... 30
6 Nasazení ... 32
7 Zhodnocení ... 33
7.1. Průběh a časová náročnost projektu ... 33
7.2 Možnosti dalšího vývoje ... 33
8 Závěr ... 35

Literatura ... 1
A Definice API ... 5
A.1 Introduction ... 5
A.2 Reference ... 5
A.3 Responses ... 10
B Obsah přiloženého CD ... 11

Seznam obrázků

Obrázek 1 Současný stav komunikace 10
Obrázek 2 Komunikace po implementaci řešení jako samostatné komponenty 11
Obrázek 3 Návrh struktury tabulky pro mapování identifikátorů zdrojů mezi systémy 16
Obrázek 4 Zasílání zpráv v případě namapovaných číselníků 17
Obrázek 5 Zasílání zpráv v případě nenamapovaných číselníků 18
Obrázek 6 Spring Container [25] 22
Obrázek 7 Pokrytí kódu unit testy 30
Obrázek 8 Časová náročnost jednotlivých fází projektu 33

Seznam tabulek

Tabulka 1 Ukázka reálných dat v systému Orchestr 3
Tabulka 2 Propagace událostí nad daty souvisejícími s akcemi České filharmonie 5
Tabulka 3 Propagace událostí nad daty souvisejícími s akcemi externích pořadatelů 6
Tabulka 4 Propagace událostí nad položkami akcí 7
Tabulka 5 Mapování HTTP požadavků na operace CRUD [11] 12
Tabulka 6 Seznam číselníků obsažených v jednotlivých zdrojích 19
Tabulka 7 Akceptační test pro vznik nové externí akce v systému Rudolf 31

Seznam výpisů kódu

Výpis kódu 1 Konfigurace pro automatické scanování komponent 23
Výpis kódu 2 Vytvoření bean ve Spring kontejneru 24
Výpis kódu 3 Dependency Injection pomocí anotace @Autowired 24
Výpis kódu 4 Ukázka konfigurace toku zpráv 27
Výpis kódu 5 Ukázka adresování komponent 27

Seznam použitých zkratek

Čf	Česká filharmonie
IMC	Intermediate component, centrální komponenta, poskytující rozhraní pro vzájemné propojení systémů České filharmonie
JAR	Java Archive
JRE	Java Runtime Environment
API	Application Interface
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
RPC	Remote procedure call
ATOM	Atom Syndication Format
RSS	Rich Site Summary
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTTP	Hypertext Transfer Protocol
SMTP	Simple Mail Transfer Protocol
MVC	Model-view-controller
IoC	Inversion of Control
CRUD	Create, Read, Update, Delete; jedná se o názvy možných operací nad daty (C – vznik záznamu, R – čtení záznamu, U – editace záznamu, D – smazání záznamu)

1 Úvod

Oddělení České filharmonie ke svému fungování a podpoře interních procesů používají různé IT systémy. Tyto systémy byly vyvíjeny v průběhu přibližně patnácti let. Jsou psány v různých programovacích jazycích, používají různé platformy a technologie. K některým systémům Česká filharmonie navíc nevlastní zdrojové kódy, proto je velmi složitá jakákoli jejich změna nebo úprava.

Jedním z hlavních problémů současného stavu je, že systémy sice používají stejná data (např. informace o koncertech), ale protože spolu vzájemně nedokážou komunikovat, data je zapotřebí mezi systémy přenášet manuálně. Tento způsob sdílení dat je nejen personálně náročný, ale je náchylný k chybám a nekonzistencím. V neposlední řadě je pak problémem absence pružné reakce systémů na akutní změny[1].

Tento stav se Česká filharmonie pokoušela vyřešit v roce 2012 vypsáním veřejné zakázky pro tvorbu komplexního informačního systému, který by podporoval všechny interní procesy Čf. Tento systém však nakonec nebyl realizován ze dvou důvodů. Prvním důvodem byla cena samotného systému. Tím druhým pak nevýhoda robustnosti, tzn. vysoká cena za následnou implementaci případných změnových požadavků. Bylo tedy rozhodnuto, že současné systémy budou ponechány oddělené a případně budou měněny nebo aktualizovány samostatně. Cílem této práce je vyřešit výše zmíněné problémy s předáváním a aktualizací dat.

Obsahem je podrobná analýza současného stavu, tj. zodpovězení následujících otázek: Které systémy jsou nyní v Čf používány? Která oddělení používají které systémy? Jaká data se v systémech nacházejí a jaké operace jsou nad nimi prováděny? Který systém je logickým původcem kterých operací? Na základě této analýzy jsem navrhnul ideální tok zpráv mezi systémy[2]. Z něj pak vychází konkrétní návrh architektury aplikace, návrh použitých technologií a implementace řešení.

[1] Jedná se např. o onemocnění protagonisty koncertu, kdy změna je do jednoho ze systémů zanesena, ale v dalších systémech se před začátkem akce již objevit nestihne.

[2] Návrh toku zpráv je odpovědí na otázku, do kterých systémů se má propagovat změna určitých dat v určitém systému a jakou obecnou formu má tato propagace mít.

2 Analýza současného stavu

Pro úspěšné zformulování a pochopení požadavků Čf na integraci IT systémů jsem nejprve musel provést analýzu současného stavu. Cílem této analýzy bylo definovat potřebná data pro jednotlivé IT systémy. Z definice těchto dat následně vychází návrh komunikačního toku mezi systémy. Analýza byla provedena ve třech krocích. V prvním kroku jsem zjišťoval, jak Česká filharmonie funguje, jakou má organizační strukturu a jaké jsou činnosti jednotlivých oddělení. Dále jsem musel zjistit, jaké IT systémy tato oddělení pro svou činnost využívají a s jakými daty tyto systémy pracují. Třetím krokem pak bylo definovat data, která jsou v systémech společná, události, které nad těmito daty v jednotlivých systémech typicky vznikají a požadovaný dopad těchto událostí na data v ostatních systémech (tj. které systémy se o konkrétní události nad konkrétními daty mají dozvědět, za jakých podmínek apod.). Pro analýzu byly použity tři hlavní informační zdroje – konzultace s pracovníky Čf, nahlížení do uživatelských rozhraní a databází současných systémů a Zadávací dokumentace k veřejné zakázce na komplexní IS pro Českou filharmonii z roku 2012 [1] (dále jen Zadávací dokumentace), která však nebyla realizována.

2.1 Organizační oddělení České filharmonie

Analýza se zabývá pouze odděleními, která přímo souvisejí s tématem práce. Nepokouším se zde zmapovat kompletní organizační strukturu Čf.

2.1.1 Koncertní oddělení

Typickou agendou koncertního oddělení je správa akcí České filharmonie – koncertů, koncertních cyklů či edukačních programů. [1]

2.1.2 Oddělení pronájmů

Česká filharmonie sídlí v budově Rudolfina a tuto budovu využívá nejen k pořádání vlastních akcí, ale pronajímá ji i jiným subjektům. O vytížení kapacity budovy a pronájmy v době, kdy se zde nekonají akce Čf, se stará právě toto oddělení.

2.1.3 Marketingové oddělení

Kromě klasické marketingové agendy má toto oddělení na starost především správu webových stránek. Ty ve skutečnosti spravuje externí firma, ale z hlediska Čf tato agenda spadá právě pod Marketingové oddělení.

2.1.4 Účetní oddělení

Toto oddělení má na starosti klasickou účetní a finanční agendu – mzdy, příchozí a odchozí transakce a vše, co s touto agendou souvisí. [1]

2.2 IT systémy používané Českou filharmonií

Oddělení Čf, zmíněná v kapitole 2.1, pro svou agendu používají různé IT systémy. V této kapitole se zabývám systémy, kterých se týká potřeba sdílení dat v reálném čase, a opomíjím systémy, které nepotřebují reagovat na aktuální změny. Příkladem takového systému je především účetní systém, do kterého jsou na vyžádání importovány údaje z ostatních systémů za uplynulé časové období nebo systém pro fakturaci, který funguje analogicky. [1]

2.2.1 Orchestr

Tento systém má za úkol podporovat procesy, týkající se akcí České filharmonie. Typicky se jedná o zveřejnění koncertů a koncertních cyklů v určitém časovém úseku. U každého koncertu jsou uvedeny podrobnosti – název, program, sólisté, dirigent, termíny[3] a místo konání. V současné chvíli není používán žádný komplexní systém. Zveřejnění a změny těchto akcí pro interní potřebu se řeší tabulkou v textovém dokumentu, která je následně zainteresovaným osobám distribuovaná e-mailem. Ukázkový obsah reálných dat v tomto dokumentu je uveden v Tabulce 1. [2]

Tabulka 1 Ukázka reálných dat v systému Orchestr

07/09/2014 neděle 10.30 Dvořákova síň	Filharmonici na pokračování Wolfgang Amadeus MOZART/arr. Bill HOLCOMBE: Figarova svatba (předehra), KV 492 Antonín DVOŘÁK/arr. David WALTER: Smyčcový kvartet F dur "Americký",op. 96 4. věta (úprava pro dechy) Georges BIZET/arr. Bill HOLCOMBE: Habanera z opery Carmen Malcolm ARNOLD: Three shanties, 1. věta Allegro con brio Ludwig van BEETHOVEN/arr. Franz WESTER: Allegro pro hrací hodiny Eugene BOZZA: Téma s variacemi op. 42, část 6 Darius MILHAUD: Krb krále René , část 6 Václav TROJAN: Variace na lidové písně op. 8, 4. věta Presto Ferenc FARKAS: Saltarello z cyklu Early Hungarian Dances Afflatus quintet	EA1
27/09/2014 sobota 19:30 Dvořákova síň	Slavnostní koncert k Roku české hudby 2014 Bedřich SMETANA: Výběr z Českých tanců (Medvěd, Slepička, Dupák) Wolfgang Amadeus MOZART: Kvintet Es dur KV 452 Leoš JANÁČEK: V mlhách Antonín DVOŘÁK: Kvintet A dur op. 81 Jitka Čechová, Ivo Kahánek, Igor Ardašev, Martin Kasík – klavír	M

[3] Koncert může mít více termínů.

	Jana Brožková – hoboj Kateřina Javůrková – lesní roh Irvin Venyš – klarinet Václav Vonášek – fagot Bennewitzovo kvarteto	

2.2.2 Rudolf

Systém Rudolf je využíván ke správě prostor Rudolfina. V systému jsou k jednotlivým akcím (ať už pronájmům nebo akcím Čf) přiřazovány položky. Položkou koncertu je např. využití šatny či ladírny před koncertem. Jednou z hlavních funkcí systému je předejít termínovým kolizím a mít přehled o využití prostor (nástrojů apod.) v Rudolfinu.

2.2.3 Teplo

Tento systém slouží k ovládání vzduchotechniky a topení v budově Rudolfina. Data dokáže automaticky čerpat ze systému Rudolf. Proces čerpání dat je však neefektivní[4] a nejsou získávány všechny potřebné informace v nejvhodnější formě.

2.2.4 Ticketing

Systém Ticketing je používán k prodeji vstupenek na všechny akce České filharmonie a na některé akce externích pořadatelů. Prodej vstupenek probíhá jak online přes webový portál, tak na pokladnách Rudolfina, které jsou na tento systém připojeny.[5]

2.2.5 Web

Webové stránky zobrazují informace o akcích České filharmonie i o ostatních akcích v Rudolfinu. Zároveň fungují jako portál s možností prokliku do ticketingového systému pro online zakoupení vstupenek.

2.2.6 E-mailový rozesílač

Česká filharmonie nemá žádný centrální automatizovaný e-mailový rozesílač. Rozesílání důležitých zpráv o akutních událostech funguje následujícím způsobem: Pokud nastane akutní změna (např. na zítřejší koncert je třeba posílit pořadatelskou službu oproti původnímu plánu), pak původce (ohlašovatel) této změny pošle ručně e-mailové zprávy na seznam předdefinovaných adres lidí, kterých by se tato změna mohla týkat. Takový postup však znamená časté plnění e-mailových schránek příjemců zprávami, které se jich netýkají.

[4] Systém Teplo si vždy znovu stahuje celý aktuální obsah systému Rudolf a tento stažený obsah pak porovnává s minulým stavem pro nalezení nových a změněných položek.

[5] V průběhu bakalářské práce byl původní ticketingový systém vyměněn za systém externího poskytovatele, takže od února 2015 jsou vstupenky objednávány a distribuovány přes webové stránky tohoto poskytovatele a na jeho výdejních místech.

2.3 Potřebné datové sady v jednotlivých systémech

Na základě Zadávací dokumentace a nahlížení do uživatelských rozhraní a do databází systémů uvedených v kapitole 2.2 jsem identifikoval konkrétní data, která jsou potřebná pro správnou funkci jednotlivých systémů, a formát těchto dat. Jedná se o tři sady dat, které jsou rozdělené podle toho, který systém je logickým původcem vzniku nebo změny těchto dat.

2.3.1 Akce České filharmonie

Akce Čf jsou zakládány a upravovány v systému Orchestr. Při vzniku nebo úpravě akce je potřeba propagovat informaci o této události podle Tabulky 2.

Tabulka 2 Propagace událostí nad daty souvisejícími s akcemi České filharmonie

Označení odpovídají událostem CRUD, tedy např. CU – Akce v tomto systému vzniká a je editována; R if CU – Systém musí obdržet zprávu v případě vzniku nebo editace akce; R if U or vznik kolize – Systém musí obdržet zprávu v případě editace akce nebo zjištěné kolize v jiných systémech.

	Orchestr	**Rudolf**[6]	**Ticketing**	**Web**[7]	**Rozesílač**
Název akce	CU	R if CU	R if C	R if C	R if U
Termíny	CU	R if C	R if C	R if C	R if U or vznik kolize
Místo konání	CU	R if C	R if C	R if C	R if U or vznik kolize
Kategorie	C	R if C	R if C	R if C	
Popis	CU	R if CU	R if CU	R if CU	
Účinkující	CU	R if CU	R if CU	R if CU	R if U
Cyklus	C	R if C	R if C	R if C	

[6] Akce Čf má v systému Rudolf formu položky se speciálním příznakem.

[7] Údaje pro web (fotografie, videa, marketingový popis akce apod.) zadává externímu správci webu marketingové oddělení. Jiné oddělení tyto údaje nepotřebuje. Je pouze potřeba, aby aplikace informovala správce webu, pokud se změní některý detail akce, zadávaný v systému Orchestr.

2.3.2 Akce externích pořadatelů

Akce externích pořadatelů jsou vkládány v systému Rudolf a ve stejném systému jsou i upravovány a mazány. Vznik a změny těchto dat by měly být propagovány podle Tabulky 3. Pracovníci různých oddělení, kterých se tyto akce týkají, mají do systému Rudolf přístup, proto není potřeba žádné události propagovat do systému Orchestr.

Tabulka 3 Propagace událostí nad daty souvisejícími s akcemi externích pořadatelů

Označení odpovídají událostem CRUD, tedy např.: CU – Akce v tomto systému vzniká a je editována; R if CU – Systém musí obdržet zprávu v případě vzniku nebo editace akce;R if UD or vznik kolize – Systém musí obdržet zprávu v případě editace akce nebo zjištěné kolize v jiných systémech.

	Rudolf	**Ticketing**	**Web**	**Rozesílač**
Název akce (pouze hlavní akce)	CUD	R if C	R if C	R if UD
Termíny	CUD	R if C	R if C	R if UD or vznik kolize
Místo konání	CUD	R if C	R if C	R if UD or vznik kolize
Kategorie	C	R if C	R if C	
Pořadatel	C	R if C (tzv. kategorie)	R if C	
Popis/účinkující etc.	CUD	R if CU	R if CU	R if UD
Cyklus		R if C		

2.3.3 Položky akcí (Čf i externích)

Položky akcí jsou vytvářeny, měněny a mazány v systému Rudolf, jak je uvedeno v Tabulce 4. Položkou akce je např. využití ladírny před koncertem, pořadatelská služba nebo i koncert samotný. Některé položky jsou přiřazené osobám Čf. O vzniku, změně a smazání položky by měl být informován e-mailový rozesílač. Rozesílač potom sám na základě vlastního nastavení rozhodne, zda a případně kam bude zprávu o této události distribuovat.

Tabulka 4 Propagace událostí nad položkami akcí

Označení odpovídají událostem CRUD, tedy např. CUD – Akce v tomto systému vzniká, je editována a mazána;R if CUD – Systém musí obdržet zprávu v případě vzniku, editace nebo smazání akce.

	Rudolf	**Rozesílač**
Název/předmět položky	CUD	R if CUD
Datum od-do	CUD	R if CUD
Čas od-do	CUD	R if CUD
Místnost	CUD	R if CUD
Zainteresované osoby	CUD	R if CUD

2.4 Funkční požadavky

Požadavky REQ 1, REQ 2 a REQ 3 definují, co bude výsledný systém umožňovat – popisují jeho jednotlivé plánované funkce. Jejich splnění bude ověřeno nasazením do reálného testovacího prostředí nebo použitím testovacích scénářů, jejichž cílem bude reálné prostředí co nejvěrněji simulovat.

REQ 1: Systém bude schopný zjistit změnu dat v některé z datových sad, definovaných v kapitole 2.3.
REQ 2: Systém bude schopný změnu dat distribuovat dalším systémům.
REQ 3: Systém bude obsahovat možnost konfigurace toku zpráv podle kombinace datových sad a událostí nad nimi.

2.5 Omezení daná současným stavem

Současný stav přináší omezení, která budu při vývoji řešení muset vzít v úvahu.

Některé systémy jsou stále vyvíjeny a měněny a je pravděpodobné, že v budoucnu bude zapotřebí předávat i jiná data, než která vyplynula z mé analýzy současného stavu. Z této skutečnosti plyne požadavek REQ 4. Dále je pravděpodobné, že v budoucnu přibudou nové systémy, které bude chtít Čf zapojit do komunikace, z čehož vyplývá požadavek REQ 5. Systémy jsou psané v různých programovacích jazycích a běží na

různých platformách, proto musí být zajištěno, aby všechny systémy dokázaly zprávy číst a interpretovat (požadavek REQ 6).

Chování některých současných systémů lze modifikovat pouze obtížně. U některých systémů není možné zasahovat přímo do zdrojových kódů a změny se musejí dělat na úrovni databázové logiky. Jiné systémy spravuje externí dodavatel a jejich úpravy jsou zpoplatněny, proto je nutné jejich změny jednoznačně a konečně definovat a omezit na nejnutnější minimum. Z těchto omezení vycházejí požadavky REQ 7 a REQ 8, jejichž splnění bude ověřeno konzultacemi se zadavatelem práce.

REQ 4: Systém bude nezávislý na obsahu předávaných zpráv. Obsah předávaných zpráv bude možné změnit, aniž by musely být provedeny zásahy do systému.

REQ 5: Systém bude možné jednoduše přizpůsobit v případě změny systémů, účastnících se komunikace (přidání, odebrání, výměna systému).

REQ 6: Systém bude zprávy předávat v platformově nezávislém a strojově čitelném formátu.

REQ 7: Systém bude navržen tak, aby byly minimalizovány potřeby změn v současných systémech Čf.

REQ 8: K systému bude dodána podrobná a jednoznačná dokumentace komunikačního rozhraní, která bude definovat, jak se mají navenek projevit změny, které bude zapotřebí implementovat na straně současných systémů.

3 Návrh řešení

V rámci návrhu řešení bylo zapotřebí navrhnout především architekturu systému, která bude co nejlépe umožňovat splnění požadavků, definovaných v kapitolách 2.4 a 2.5. Po zvolení vhodné varianty řešení v této kapitole dále navrhuji způsob komunikace mezi systémy včetně definice komunikačního rozhraní, které budou systémy využívat, a technologie, které použiju pro implementaci. Posledním tématem v této kapitole je pak otázka zabezpečení celého systému.

3.1 Architektura řešení

Řešení může být realizováno dvěma způsoby. Prvním způsobem je pouhé provedení změn v současných systémech tak, aby mezi sebou mohly přímo komunikovat. Druhým možným řešením je pak vytvoření a sestavení komponentového modelu, kdy budou systémy komunikovat skrze nově vytvořenou centrální komponentu.

Výhodou řešení bez centrální komponenty by měla být jeho jednoduchost – nebude nutné vytvářet nový systém, ale provedení změn v současných systémech umožní předávání dat, definovaných v kapitole 2.3. Nevýhodou tohoto přístupu je však nemožnost jakékoli následné konfigurace a složitost realizace dalších změn, což odporuje požadavkům REQ 3, REQ 4, REQ 5 a REQ 7.

Proto jsem se pro integraci systémů České filharmonie rozhodl použít komponentový model a vytvořit samostatnou centrální komponentu, která stávající systémy propojí. Výhodou tohoto řešení je především možnost jednoduché konfigurace toku zpráv (REQ 3), nezávislost na systémech, které se komunikace účastní (REQ 5) a možnost přesné definice formátu zpráv, které tato centrální komponenta přijímá a rozesílá (REQ 8).

V kapitole 3.1.1 se v souvislosti s tímto návrhem zabývám problematikou vytvoření komponentového modelu obecně, v kapitole 3.1.2 pak problematiku návrhu komponentového modelu aplikuji na tuto konkrétní práci.

3.1.1 Návrh obecného komponentového modelu

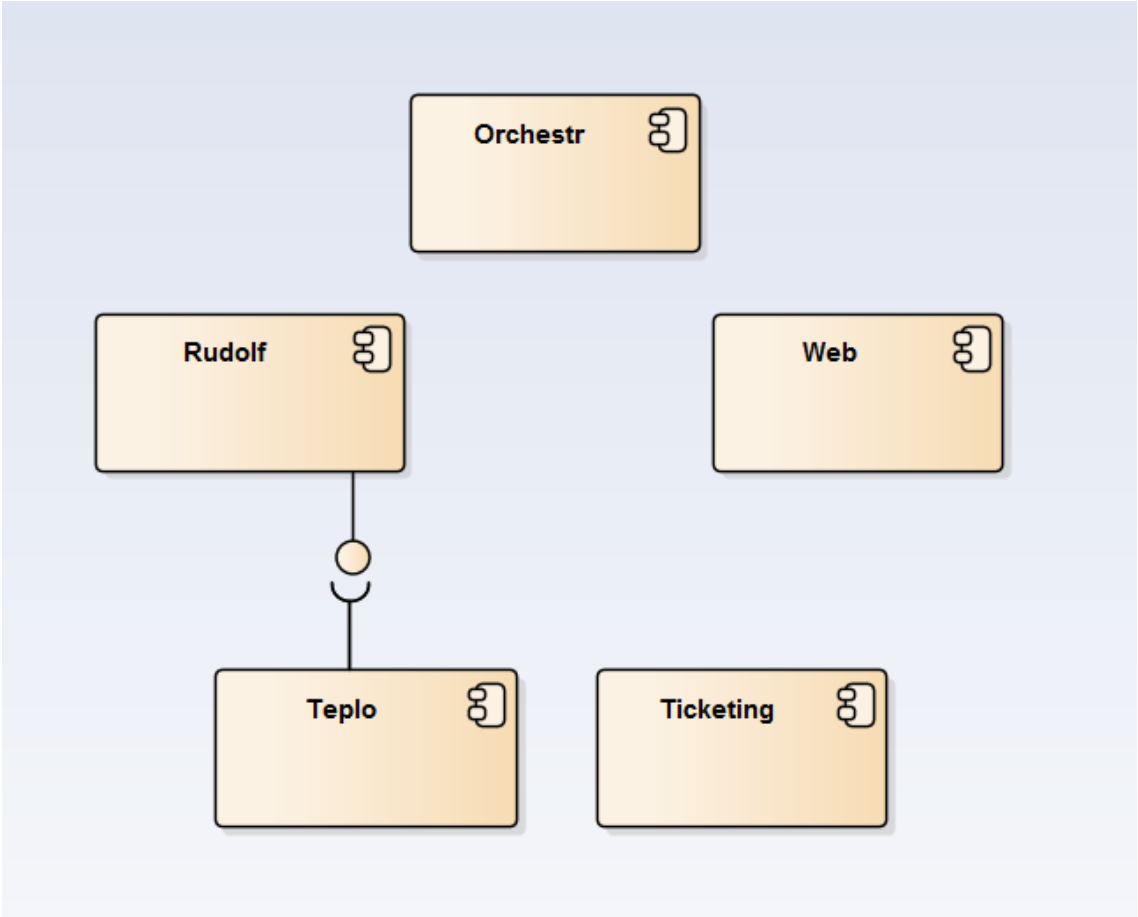
Komponenta je program (modul, package, namespace – dle terminologie konkrétního programovacího jazyka), který vytváří nějakou funkcionalitu. Její implementace je navenek skrytá (funguje jako tzv. „černá skříňka“), viditelné je pouze rozhraní, přes něž lze komponentu používat. Rozhraní popisuje, jaké služby komponenta poskytuje a co pro poskytování těchto služeb požaduje. [4]

Při vytváření kompletně nové aplikace se tato rozdělí na podcelky (komponenty), u nichž je definováno, jaké služby budou poskytovat a požadovat. Komponenty se dále dělí stejným způsobem, dokud konečné celky nejsou „dostatečně jednoduché k implementaci“. [4] Rozhraní komponent je popisováno jako množina metod s parametry. Při integraci existujících komponent je zapotřebí definovat rozhraní, přes které budou komponenty propojeny a jednotlivé komponenty upravit tak, aby toto rozhraní dokázaly používat.

3.1.2 Plán vývoje komponenty pro integraci služeb Čf

Jak bylo zmíněno v úvodu kapitoly 3, pro integraci systémů České filharmonie jsem se rozhodl použít komponentový model a vytvořit samostatnou centrální komponentu, která stávající systémy propojí.

Obrázek 1 znázorňuje současný stav komunikace – existují různé systémy České filharmonie, které nemají žádné rozhraní, přes které by spolu dokázaly komunikovat.



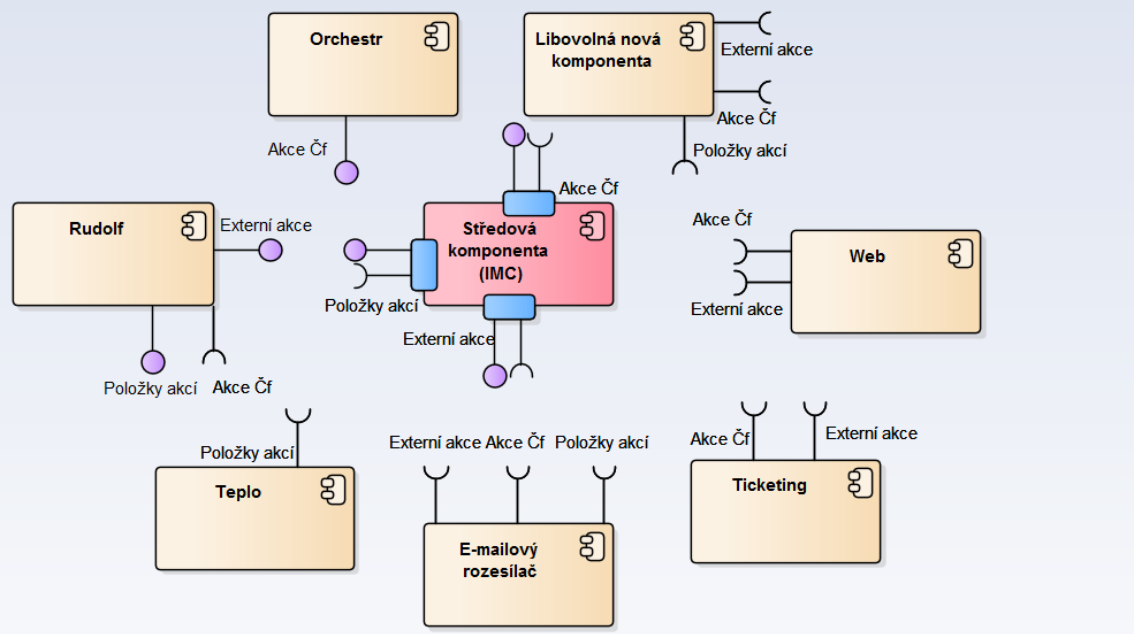
Obrázek 1 Současný stav komunikace

Systémy v současné chvíli nemají žádné rozhraní, přes které by spolu mohly komunikovat. Jediná relevantní komunikace se děje přímo mezi systémem Rudolf a Teplo.

Pro umožnění komunikace mezi systémy navrhuji přidat centrální komponentu (dále označovaná i jako IMC[8]). Ta bude jako komunikační rozhraní poskytovat tři hlavní dvojice metod, přičemž každá dvojice bude mít na starost komunikaci ohledně jedné datové sady. Dvojice metod pak budou následující:

1. Přijmi zprávu o události nad objektem datové sady ve zdrojovém systému.
2. Odešli zprávu o této události do cílových systémů.

Zdrojové systémy (pro datovou sadu Akce České filharmonie se jedná o systém Orchestr, pro datové sady Položky akcí a Externí akce o systém Rudolf) budou implementovat rozhraní pro odeslání zprávy o události. Cílové systémy, které se chtějí o těchto událostech podle konfigurace dozvědět, musí implementovat rozhraní pro přijetí zprávy o události. Požadovaná rozhraní jednotlivých systémů znázorňuje Obrázek 2.



Obrázek 2 Komunikace po implementaci řešení jako samostatné komponenty

Přidání centrální komponenty umožní vzájemnou komunikaci systémů skrze rozhraní IMC. Ta vystaví tři rozhraní pro přijetí informace o změně některé ze tří datových sad ve zdrojových systémech a tři rozhraní pro distribuci této změny do cílových (naslouchajících) systémů. Zdrojovým systémem pro datovou sadu Akce Čf je Orchestr, pro sadu Externí akce a Položka akce je to pak systém Rudolf.

[8] InterMediate Component

3.2 Návrh technologií pro vývoj centrální komponenty

Pro vývoj komponenty jsem se rozhodl využít technologií jazyka Java, se kterými mám nejrozsáhlejší zkušenost. Java se pro tento účel hodí především proto, že se jedná o jazyk objektově orientovaný, distribuovaný, nezávislý na použitém operačním systému i architektuře [5] a pro účel této aplikace dostatečně robustní. Bude se jednat o webovou aplikaci, kompilovatelnou Mavenem [6] s využitím frameworku Spring Boot [31]. Z důvodu jednoduchosti a snadné přenositelnosti se bude jednat o standalone aplikaci [30], která v sobě bude obsahovat vlastní aplikační server [7]. Pro persistenci dat bude využívat MS SQL [35] databázový stroj České filharmonie. Více podrobností o použitých technologiích obsahuje kapitola 4 Implementace.

3.3 Možnosti komunikace mezi komponentami

Systémy Čf budou s komponentou komunikovat pomocí webových služeb. Webová služba je softwarový systém, umožňující interakci více strojů (systémů) v síti. [10] V současnosti existuje několik rozšířených webových služeb. Patří mezi ně REST (Representational State Transfer) [11], RPC (Remote Procedure Call) [12] a SOAP (Simple Object Access Protocol) [13].

3.3.1 REST

Základem REST rozhraní jsou zdroje (data), pro něž REST definuje jednotný přístup v podobě CRUD operací, spouštěných HTTP požadavky. Běžně používané mapování HTTP požadavků na operace CRUD znázorňuje tabulka 5. REST dokáže pomocí těchto čtyř základních metod pracovat s objekty i s jejich kolekcemi.

Tabulka 5 Mapování HTTP požadavků na operace CRUD [11]

HTTP požadavek	CRUD operace
GET	Read
POST	Create
PUT	Update
DELETE	Delete

3.3.2 RPC

Tato technologie dovoluje programu vykonat proceduru/funkci/metodu (dle terminologie konkrétního programovacího jazyka), která je uložena na jiném stroji nebo v jiném programu. Volající stroj nejprve zabalí všechna potřebná data do formy vhodné pro přenos (tzv. marshalling) a tento balíček odešle volanému stroji. Volaný stroj data rozbalí (tzv. unmarshalling), proceduru zavolá a provede, výsledek zabalí a odešle volajícímu, který výsledek rozbalí a předá vlastní proceduře, která vše odstartovala. [12]

3.3.3 SOAP

SOAP je protokol, který přenáší zprávy ve formátu XML, nejčastěji pomocí protokolu HTTP. Pro výměnu zpráv však lze použít např. SMTP. [13] SOAP zpráva je XML dokument, který obsahuje element Envelope, signalizující, že se jedná o SOAP zprávu. Uvnitř tohoto elementu se nachází hlavička a tělo zprávy, případně element obsahující chybová hlášení. [14]

3.3.4 Srovnání REST – RPC

Rozdíl mezi těmito dvěma technologiemi je v rovině sémantiky (definice operací) a mezi tím, co je distribuováno. V REST jsou definovány pouze CRUD operace, nad daným zdrojem, zatímco RPC je sémantika definována volanými metodami. V REST je pak distribuován stav zdroje, zatímco v RPC je distribuováno chování. [11]

Výhody REST oproti RPC jsou především následující: [11]

- Jednoduché a změnám odolné rozhraní, snadná rozšiřitelnost.
- Malé nároky na klienta z hlediska porozumění sémantice operací.
- Transparentnost - zdroj lze na "cestě" velice snadno cacheovat, transformovat atd.
- REST může pro komunikaci používat i jiných formátů než XML.

3.3.5 Srovnání REST – SOAP

Výhody REST oproti SOAP jsou především následující: [15]

- REST je pro většinu případů použití jednodušší (strmější učicí křivka).
- Výkonnější (SOAP používá pouze XML, REST může použít jednodušší formáty).
- Rychlejší (nepoužívá rozsáhlé zpracování).
- Designem je podobný jiným webovým technologiím.

Výhody SOAP oproti REST jsou naopak tyto:

- Podpora distribuovaného prostředí (REST předpokládá přímou komunikaci mezi dvěma body).
- Standardizovaný.
- Vestavěné řízení chyb.

Pro úplnost je ještě potřeba zmínit, že srovnání SOAP – REST není úplně relevantní. Pro REST neexistuje žádný „oficiální“ standard, protože se jedná v podstatě o styl architektury komunikace, zatímco SOAP je protokol. [15] [11]

3.3.6 Použití webových služeb pro integraci systémů Čf

Pro komunikaci mezi systémy Čf a komponentou jsem se rozhodl využít architektury REST, komunikace přes HTTP a jako formát zpráv použít JSON. Důvody jsou následující:

1. REST architekturu komunikace lze jednoduše navrhnout tak, aby co nejlépe splňovala požadavky na dobře navržené API [16], jako je jednoduchá zapamatovatelnost, návrh odolný proti vzniku chyb, lehká rozšiřitelnost nebo

fakt, že kód psaný pro komunikaci s tímto API má všechny předpoklady k dobré čitelnosti. [17]
2. REST architektura je relativně mladým konceptem, který se za krátkou dobu stihl masově rozšířit a dá se předpokládat jeho obliba i v budoucnosti, což znamená, že případné budoucí úpravy a rozšíření stávající aplikace budou jednodušší.
3. K REST existuje množství návodů a tipů a řešení chyb a problémů je dobře dokumentováno.
4. Pro práci s REST rozhraním existují v Javě a frameworku Spring Boot kvalitní knihovny. [18]

3.4 Komunikační rozhraní

Pro kompletní popis rozhraní jsem použil službu Apiary[9]. [19] Největší výhodou této služby pro mě byla návodná syntaxe popisu API, která zajišťuje, že definice API je srozumitelná a obsahuje kompletní informace, potřebné pro funkční komunikaci.

Podrobně definované API, které IMC poskytuje je v příloze A. API, které musí implementovat současné systémy Čf je pak součástí přiloženého CD.

V návrhu API jsou definované tři hlavní zdroje[10], které odpovídají datovým sadám, definovaným v kapitole 2.3. Pokud nad některým z těchto zdrojů vznikne ve zdrojovém systému událost, tento systém o události informuje IMC, která tuto zprávu propaguje do cílových (naslouchajících) systémů (viz Obrázek 2).

Zdroje a události, které nad nimi IMC odposlouchává, jsou následující[11]:

3.4.1 Czech Philharmonic action [/CPAction]

- Create (POST, PUT)
- Update (PUT)

Zdroj `CPAction` odpovídá datové sadě Akce České filharmonie, definované v kapitole 2.3. IMC zajímá vznik a úprava tohoto zdroje v systému Orchestr. Smazat akci je teoreticky také možné, ale děje se to pouze v době předběžného návrhu celoročního programu, kdy distribuce jakýchkoli úprav jiným systémům nemá smysl.

Vznik nové akce ve zdrojovém systému může být IMC oznámen zasláním požadavku `POST` i zasláním požadavku `PUT`. Důvodem je snaha o zjednodušení úprav současných systémů[12] podle požadavku REQ 7. Stejným způsobem jsou navrženy i následující dva zdroje.

[9] Jedná se o zajímavý český technologický startup, který „…chce být Githubem pro webová APIs. Pomáhá programátorům navrhnout nové API, popsat jeho dokumentaci, API i dokumentaci automaticky testovat a sbírat zpětnou vazbu či chybová hlášení od uživatelů.“ [20] Apiary založil Jakub Nešetřil.

[10] Zdroj je oficiální termín návrhu REST API. Je to klíčová abstrakce informace, tedy se jedná o jakoukoli informaci, která může být pojmenována, např. dokument, obrázek, dočasná služba (dnešní počasí), osoba apod. [21] V této práci navíc používám termín „zdrojový systém“, který označuje systém, v němž vznikla událost nad REST zdrojem.

[11] Ve zdrojovém kódu, a proto i v návrhu API, který s ním již přímo souvisí, používám oproti analýze anglická pojmenování.

[12] V editačním formuláři v současném systému lze např. pouze přidat tlačítko „Publish to other systems“ a systém nebude muset rozlišovat, zda se jedná o úpravu nebo o vznik nové entity. Toto rozlišení udělá IMC na základě toho, zda zná identifikátor této entity. V případě, že identifikátor nezná, provede metodu POST. Pro opačné použití (oznamovat úpravu existující entity pomocí POST) IMC připravena není.

3.4.2 Action Item [/Item]

- Create (POST, PUT)
- Update (PUT)
- Delete (DELETE)

Zdroj `Item` odpovídá datové sadě Položky akcí. Jak je uvedeno v kapitole 2.3, položka akce vzniká, je upravována a může být mazána v systému Rudolf.

3.4.3 External action [/ExternalAction]

- Create (POST, PUT)
- Update (PUT)
- Delete (DELETE)

Zdroj `ExternalAction` odpovídá sadě Externí akce. IMC zajímá vznik, úprava a smazání tohoto zdroje v systému Rudolf. Podobně jako u zdroje `CPAction` není smazání standardní událostí, proto je distribuováno pouze do systému Rozesílač a v dalších systémech se bude muset řešit manuálně.

3.5. Zajištění správné interpretace zpráv

IMC nepotřebuje znát obsah předávaných zpráv s výjimkou odkazů na objekty, které existují ve zdrojovém i v cílovém systému a v každém z těchto systémů jsou reprezentovány odlišně[13]. Těmito objekty jsou jak tři hlavní zdroje, tak i číselníky, které tyto zdroje používají.

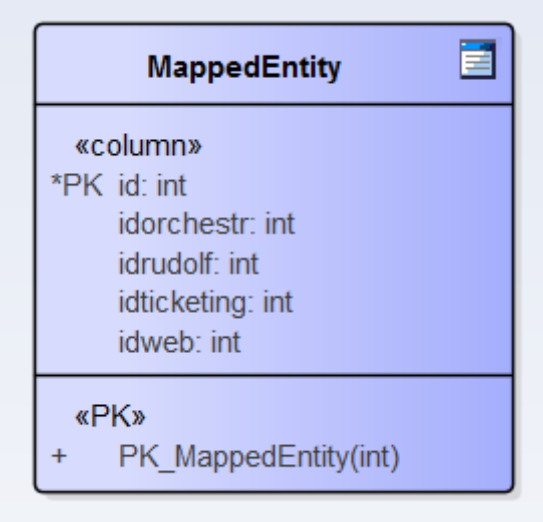
Informace o těchto objektech zdrojový systém předává IMC nikoli textově, ale pouze v podobě identifikátoru, pod kterým jsou tyto objekty ve zdrojovém systému uloženy. IMC pak provede „přemapování“, tzn. do každého cílového systému posílá zprávu, která obsahuje identifikátor zdroje v cílovém systému. Proces mapování identifikátorů probíhá dvěma různými způsoby - podle toho, zda se jedná o jeden ze tří hlavních zdrojů nebo o číselník.

3.5.1 Mapování identifikátorů hlavních zdrojů

Pokud do IMC přijde zpráva o vzniku nové entity ve zdrojovém systému (např. ze systému Orchestr přijde `POST CPAction`), tato zpráva v těle obsahuje identifikátor nově vzniklé entity. IMC si tento identifikátor uloží a odesílá zprávy o vzniku této entity do cílových systémů. Ty potom v těle odpovědi vrací identifikátor, pod kterým mají novou entitu uloženou. V případě dalších zpráv, které se týkají této entity (`PUT`, `DELETE`), IMC najde odpovídající identifikátor pro cílový systém a zašle každému cílovému systému zprávu s příslušným identifikátorem. Návrh struktury databázové tabulky, která bude

[13] V systému Rudolf je Organizer uložen v podobě oficiálního právního názvu pro smluvní účely, v systému Ticketing pak je uveden pouze veřejně známý název pro účely tisku vstupenek. Párování nelze provést ani na základě nějakého reálného identifikátoru (IČO), protože ani ten některé systémy neobsahují.

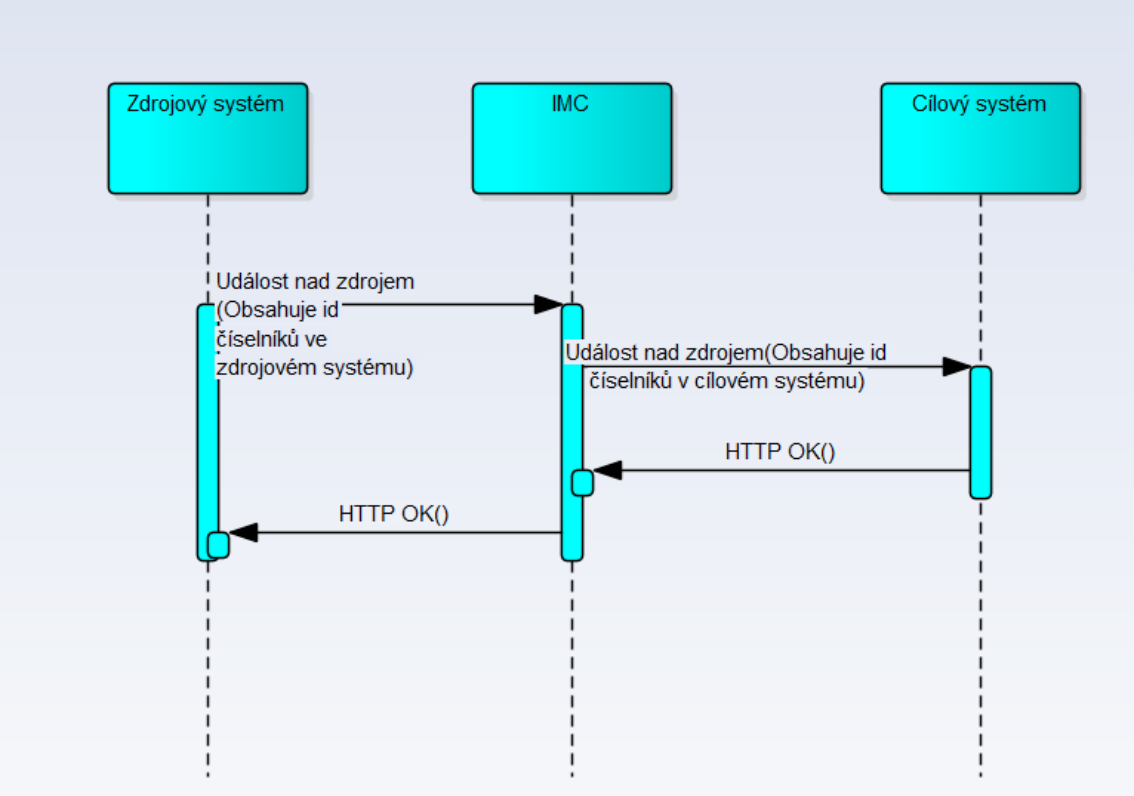
sloužit k udržování záznamů o hodnotách identifikátorů v jednotlivých systémech, znázorňuje Obrázek 3.



Obrázek 3 Návrh struktury tabulky pro mapování identifikátorů zdrojů mezi systémy

3.5.2 Mapování identifikátorů číselníků

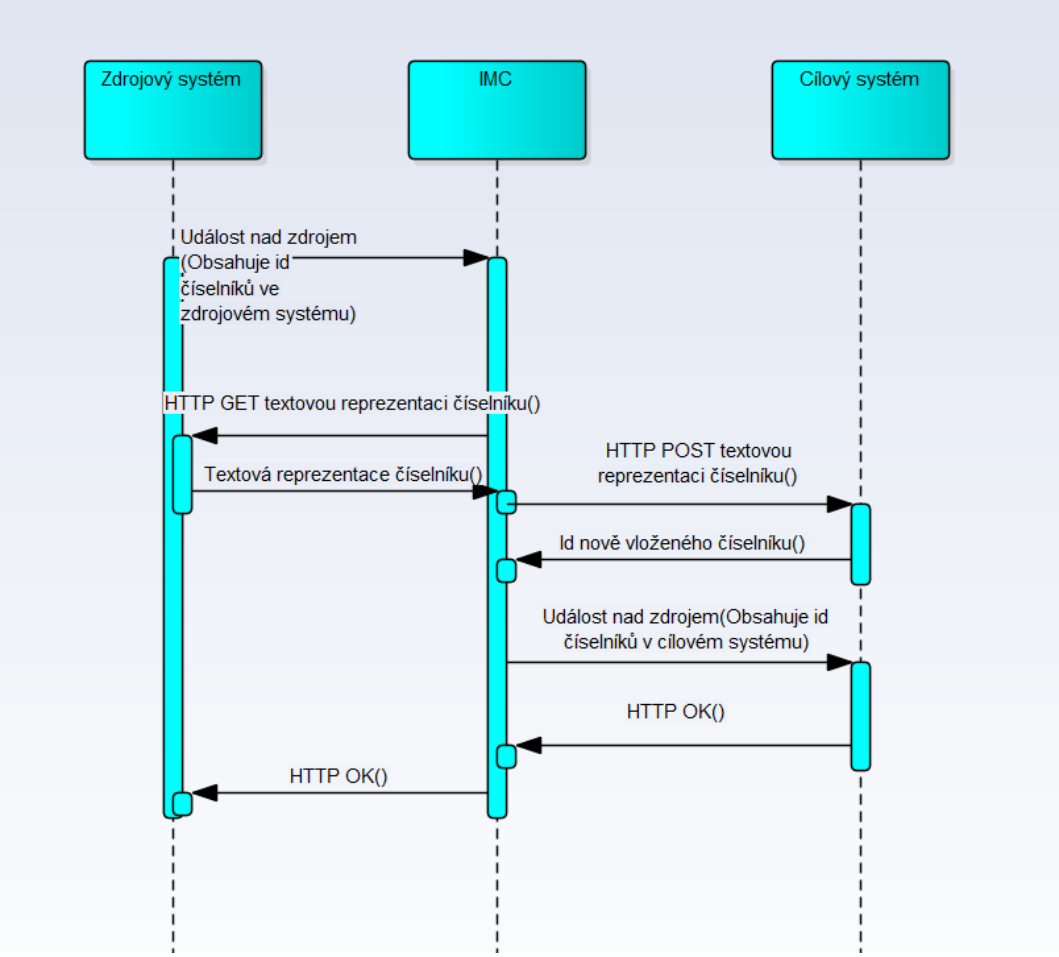
Pokud je v příchozí zprávě obsažen identifikátor některého z mapovaných číselníků, IMC provede přemapování identifikátoru stejným způsobem, jako v případě identifikátoru hlavních zdrojů. Komunikaci znázorňuje Obrázek 4.



Obrázek 4 Zasílání zpráv v případě namapovaných číselníků

Pokud však IMC nezná hodnotu identifikátoru číselníku ve zdrojovém systému, pak komunikace probíhá jiným způsobem. IMC si nejprve vyžádá textovou reprezentaci číselníku ve zdrojovém systému, tuto reprezentaci odešle do cílového systému, který vrátí identifikátor, pod kterým vytvořil tento nový záznam a teprve potom IMC posílá původní zprávu s již přemapovaným číselníkem. Tento případ je znázorněn na Obrázku 5.

Problémem tohoto postupu je, že číselníky v cílových systémech musí být upraveny ručně, protože často má stejný záznam v různých systémech odlišnou reprezentaci. Nově vložený číselník v cílovém systému tak nejspíš nebude moci být plnohodnotně používán, dokud nebude ručně upraven. Řešení, které jsem navrhl, jsem však oproti jiným řešením[14] vyhodnotil jako nejjednodušší a nejodolnější vůči neočekávaným stavům.



Obrázek 5 Zasílání zpráv v případě nenamapovaných číselníků

Z Obrázku 5 plyne, že zdrojový systém musí pro každý číselník obsažený ve zdroji implementovat metodu `GET /Název číselníku/{id}`, která vrací textovou reprezentaci číselníku. Cílové systémy pak musí implementovat metodu `POST /Název číselníku`, která vrací id nově vloženého záznamu. Seznam číselníků a jejich příslušnost ke zdrojům je uveden v tabulce 6.

[14] Dalším uvažovaným řešením byla např. odpověď cílového systému s id až ve chvíli ručního schválení záznamu, což by mohlo zjednodušit implementaci v cílovém systému. Největším problémem takového řešení je situace, kdy do cílového systému mají být doručovány další zprávy, obsahující tento číselník, do doby, než bude číselník vložen a než IMC bude znát jeho id v cílovém systému.

Tabulka 6 Seznam číselníků obsažených v jednotlivých zdrojích

Číselník	Obsažen ve zdroji	Význam
Place	CPAction, ExternalAction	Místo konání akce
Category	CPAction	Kategorie akce (např. edukativní akce)
Cycle	CPAction	Cyklus, tj. série akcí, na něž je nabízena abonentní vstupenka
ItemSubject	Item	Předmět položky pronájmu (např. pronájem Sukovy síně, ladění křídla…)
Organizer	ExternalAction	Organizátor externí akce, smluvní subjekt pronájmu

S mapováním číselníků však souvisí problematika, kterou je před spuštěním systému zapotřebí vzít v úvahu. Číselníky už jsou v jednotlivých systémech naplněny daty, proto je nejprve nutné namapovat manuálně. Pokud by nebyly namapovány, pak by se do cílových systémů vkládaly nové duplicitní záznamy a cílové systémy by musely tuto situaci řešit. U číselníku `Category`, `Cycle` a `ItemSubject` by úvodní manuální namapování nemělo představovat problém, protože tyto číselníky obsahují desítky nebo stovky záznamů. Problém však nastává u číselníku `Place`, který obsahuje vyšší stovky záznamů a především u číselníku `Organizer` (v reálu se jedná o smluvní subjekty Čf), který obsahuje cca 10 000 záznamů [1]. Tyto záznamy lze těžko párovat automaticky, protože v jednotlivých systémech mají výrazně odlišnou reprezentaci[13]. Analýza a řešení tohoto problému je nad rámec této práce. Důsledkem je očekávání, že zatímco Čf bude schopna spustit reálný provoz komponenty pro zdroj `CPAction` a `Item` v nedaleké době, provoz pro zdroj `ExternalAction` pravděpodobně v blízké době spuštěn nebude.

Podrobně definované API pro jednotlivé komponenty je v příloze A – Definice API.

3.6 Konfigurace toku zpráv

Požadavek REQ 3 hovoří o možnosti konfigurace toku zpráv podle kombinace zdroje a události nad ním. Konfiguraci navrhuji provádět pomocí XML souboru, jehož výhodou je dobrá čitelnost a pochopitelnost. Konfigurační soubor bude pro každý zdroj a událost obsahovat názvy systémů, do kterých má být zpráva o této události distribuována.

3.7 Zabezpečení komunikace

Komunikace mezi komponentami může být napadena dvěma různými způsoby. Útočník buď bude odposlouchávat komunikaci, nebo se ji může pokusit podvrhnout zasíláním zpráv IMC. V komunikaci nejsou přenášeny zneužitelné údaje (uživatelská jména a hesla, údaje ke kreditním kartám apod.), ale jsou přenášeny údaje citlivé (cena pronájmu, jména smluvních subjektů). Odposlouchávání komunikace tak může být

problémem. Stejně tak může být problémem podvržení zpráv, které by mohlo vést ke zmatkům v cílových systémech, kde by se začala objevovat falešná data.

Pro zabezpečení komunikace jsem navrhnul použít technologii OAuth2 [32] v kombinaci s asymetricky šifrovaným protokolem HTTPS [42]. Proces autorizace pomocí protokolu OAuth2 je následující: [32]

1. Systém, který chce zaslat zprávu, musí požádat o přístupový kód (access token) požadavkem se správným jménem a heslem.
2. Pokud se jméno a heslo shoduje s nastavením IMC, systém dostane odpověď, obsahující access token, kód pro obnovení přístupového kódu (refresh token) a čas, kdy přístupový kód vyprší.
3. Pro další požadavky musí systém do hlavičky umístit získaný přístupový kód, díky kterému je do vypršení timeoutu autorizován.
4. Po vypršení timeoutu si systém může požádat o nový access token pomocí refresh tokenu.

Po konzultacích s odpovědnými pracovníky Čf jsem zjistil, že není jasné, zda současné (a potenciální budoucí) systémy budou schopny implementovat tyto dva druhy zabezpečení, proto jsem se rozhodl použít pouze základní zabezpečení pomocí Basic access authentication [22]. Tento způsob bude vyžadovat pouze umístění atributu `Authorization: Basic` s příslušným jménem a heslem do hlavičky zasílané zprávy.

4 Implementace

4.1 Použité technologie a knihovny

Jak je zmíněno a odůvodněno v kapitole 3, pro vývoj komponenty jsem se rozhodl využít jazyka Java a frameworku Spring Boot, kompilovatelného Mavenem. Základním konfiguračním souborem Mavenu je soubor `pom.xml` (Project Object Model), umístěný v kořenovém adresáři. Tento soubor popisuje projekt jako objekt XML strukturou, která definuje části projektu a závislosti na externích knihovnách. [6]

V projektu jsem použil pro umožnění komunikace přes protokol HTTP knihovnu `org.apache.httpcomponents.httpclient`. Pro sestavení celé aplikace stylem Inversion of Control (IoC) [24] je použita knihovna `spring-boot-starter-web` [25] frameworku Spring.

Persistentní vrstva [33] může využívat buď databázi PostgreSQL [34], kterou jsem využíval při vývoji aplikace (knihovna `org.postgresql`), nebo databázi MS SQL [35], která je využívána v produkčním prostředí (knihovna `com.microsoft.sqlserver.sqljdbc4`). Pro komunikaci s persistentní vrstvou je využívána knihovna `spring-boot-starter-jdbc` [25].

Pro práci s obsahem zpráv, které jsou předávány ve formátu JSON používám externí knihovny `org.codehaus.jackson.jackson-mapper-asl` [36] a `org.json.json` [37].

Dalšími knihovnami jsou pak `org.hamcrest` [38], `junit` [39], `org.mockito` [40] a `org.springframework.spring-test` [25] pro unit testy a pro logování událostí knihovna `log4j` [41].

4.2 Struktura aplikace

Vnější struktura aplikace je dána výchozím nastavením Mavenu [6], které je pro účely mé aplikace lehce modifikované. Kořenový adresář obsahuje soubor `pom.xml`, v adresáři `src/main/java` se nacházejí kompilovatelné java soubory, v adresáři `src/main/resources` konfigurační soubor `application.properties` a adresář `src/test/java` obsahuje třídy testů.

Pro vnitřní strukturu aplikace vycházím z architektonického vzoru vícevrstvé architektury. [23] Protože však IMC neobsahuje prezentační vrstvu, nejedná se o klasický příklad třívrstvé architektury (prezentační vrstva – business logika – datová vrstva).

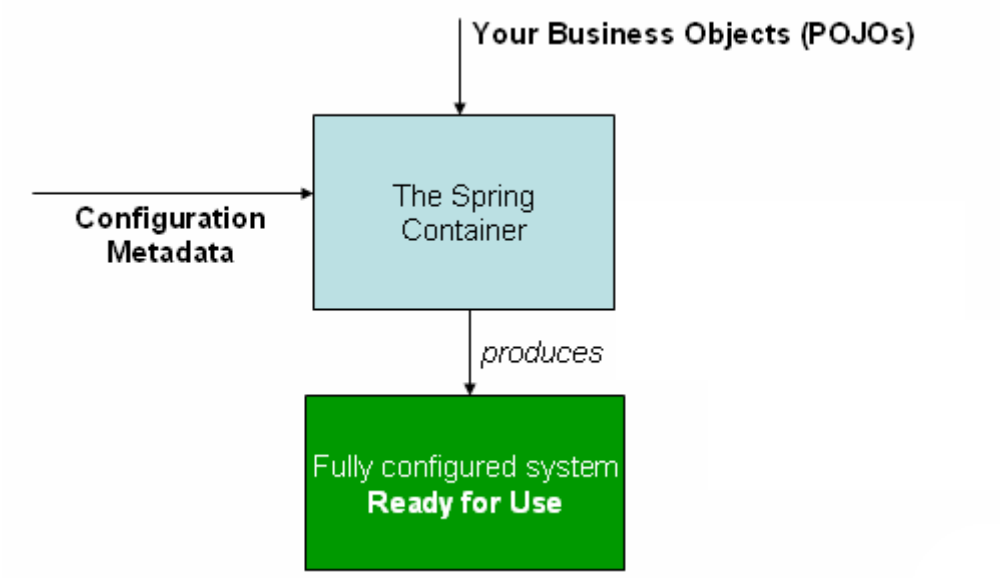
„Vrchní“ vrstvou aplikace je `Controller`, který řídí příjem zpráv o událostech a vracení odpovědí na tyto zprávy. Veškerou logiku zpracování zpráv předává servisní vrstvě. Servisní vrstva slouží pro přístup k datové vrstvě, validaci zpráv, posílání zpráv a pro generování příslušných odpovědí na zprávy. Datová vrstva obsahuje čtyři klasické metody pro práci s databázovými entitami odpovídající operacím CRUD – create, get, update a delete. Vrstva `Model` obsahuje definici objektů, používaných v ostatních částech aplikace pro dodržení objektového přístupu. Klíčovými objekty jsou zpráva k zaslání, komponenta (tj. systém Čf), mapovaný zdroj a mapovaný číselník. Adresář `Resources` obsahuje textové konstanty, používané v ostatních částech aplikace. Jedná se o chybová hlášení, texty pro logování událostí a o seznam a názvy komponent a čí-

selníků, které jsou mapovány. Adresář `Utilities` pak obsahuje třídy, které vykonávají nějaký jednoduchý a jasně ohraničený úkol, který se v ostatních částech aplikace opakuje. Jedná se o parsování konfiguračních xml souborů, práci s JSON objekty, zasílání zpráv a o různé metody přemapovávání id zdrojů a číselníků mezi systémy.

4.3 Detaily implementace

4.3.1 Sestavení aplikace pomocí Inversion of Control

Jednou ze základních funkcionalit Springu je použití návrhového vzoru Inversion of Control (Obrácení řízení), pro který používá techniku Dependency Injection (Vkládání závislostí). Výhodou této techniky je, že jedna třída programu (obecně komponenta programu) může používat jinou, aniž by na ni měla v době sestavování programu referenci. [24] Na obrázku 6 je znázorněno, jak je tato technika provedena ve Springu. Spring má vlastní kontejner, ve kterém existují instance java tříd, které jsou pomocí konfiguračních metadat označeny jako součást kontejneru (v terminologii Springu tzv. beany). Pokud některá třída potřebuje použít instanci takové třídy, Spring jí „podstrčí“ (tzv. nainjectuje) beanu z kontejneru.



Obrázek 6 Spring Container [25]

Konfigurační metadata mohou být Springu předána ve formě XML konfiguračních souborů, java anotacemi nebo Spring anotacemi. Ve své práci jsem zvolil poslední jmenovaný způsob. Ukázka použití konfigurace pomocí Spring anotací je ve Výpisu kódu 1. Výpis kódu 2 demonstruje vytvoření dalších bean ve Spring kontejneru. Výpis kódu 3 pak znázorňuje způsob, jakým jsou beany z kontejneru injektovány do tříd, které je používají.

Výpis kódu 1 Konfigurace pro automatické scanování komponent

Při použití anotace @ComponentScan Spring automaticky projde všechny třídy, specifikované atributem basePackages a do kontejneru umístí třídy s anotací @Component, @Service, @Controller a @Repository. Dále do kontejneru umístí instance tříd, označené anotací @Bean ve třídách s anotací @Configuration. [24] Anotace @EnableAutoConfiguration je součástí frameworku Spring Boot. Díky této anotaci Spring konfiguruje aplikaci na základě závislostí v pom.xml. Na základě závislosti spring-boot-starter-web Spring konfiguruje aplikaci jako standalone MVC webovou aplikaci. [25]

```
@ComponentScan
@EnableAutoConfiguration
public class Application
{
  public static void main(String[] args) throws Exception {
    SpringApplication.run(Application.class, args);
  }
}

@Controller
public class IMCController {

}

@Service
public class IMCService {

}

@Repository
public interface IDao {

}
```

Výpis kódu 2 Vytvoření bean ve Spring kontejneru

Beany jsou ve Spring kontejneru vytvořeny jako instance metody s anotací @Bean v třídě s anotací @Configuration nebo jako instance třídy s anotací @Component, @Controller, @Service nebo @Repository vytvořená bezparametrickým konstruktorem.

```
@Configuration
public class AppConfig {

  @Bean
  public MappedEntityIdResolver Resolver() {
    return new MappedEntityIdResolver();
  }

  @Bean
  public RestTemplate RestTemplate() {
    return new RestTemplate();
  }

  @Bean
  public MessageSender sender() {
    return new MessageSender();
  }

}

@Controller
public class IMCController {

}
```

Výpis kódu 3 Dependency Injection pomocí anotace @Autowired

```
@Controller
public class IMCController {

  @Autowired
  private RestTemplate rt;

  @Autowired
  private IMCService service;
}

@Service
public class IMCService {

  @Autowired
  IDao dao;
}

@Repository
public interface IDao {

}
```

4.3.2 Komunikační rozhraní

Příchozí požadavky jsou přiřazeny jednotlivým metodám `Controlleru` pomocí anotace `@RequestMapping`, která je součástí knihovny `org.springframework.web.bind.annotation`. Ta pro každou metodu určuje adresu a typ požadavku, který je touto metodou obsluhován. Příkladem může být anotace `@RequestMapping(value = "IMC/CPAction", method = RequestMethod.PUT)`, která znamená, že anotovaná metoda obsluhuje HTTP PUT požadavek na adrese `/IMC/CPAction`.

Pro odesílání zpráv aplikace používá třídu `RestTemplate` knihovny `org.springframework.web.client` a její metodu `exchange()`, která odesílá objekt typu `HttpEntity` knihovny `org.springframework.http` s tělem požadavku.

4.3.3 Mapování id mezi systémy

Mapování id zdrojů mezi systémy probíhá vždy při zavolání `POST` požadavku (vzniku nového záznamu ve zdrojovém systému). Nejprve je vytvořen objekt typu `mappedResource`, z příchozího těla zprávy je přečteno jeho id ve zdrojovém systému a toto id je mu nastaveno. Dále jsou posílány zprávy o této události do cílových systémů a každý cílový systém vrací id nově vložené entity. Tato id aplikace postupně přiřazuje do `mappedResource` objektu, který po odeslání všech zpráv uloží do tabulky s analogickou strukturou, která je znázorněna na obrázku 3. Při dalších příchozích zprávách ohledně této entity program pro každou zprávu zasílanou do cílového systému z databáze získá id v cílovém systému a do těla zprávy toto id vloží pomocí třídy `JSONObject` a její metody `put()`. Analogickým způsobem probíhá mapování id číselníků.

4.3.4 Konflikty a neočekávané stavy

Konflikty a neočekávané stavy mohou nastat na dvou místech – uvnitř IMC nebo uvnitř cílových systémů. Konflikt uvnitř IMC nastává v případě, že zdrojový systém oznamuje vznik zdroje se stejným id, které již v IMC je mapováno. Druhou možností je pokus o smazání položky, jejíž id v IMC naopak namapováno není. V takovém případě IMC vrací status CONFLICT a v těle zprávy oznamuje příčinu.

Pokud některý z cílových systémů vrátí status HTTP CONFLICT nebo v něm vznikne jakákoli jiná chyba (popř. na specifikované adrese systém vůbec neběží), všechny následující zprávy jsou odeslány a do zdrojového systému je vrácena zpráva, obsahující v hlavičce status CONFLICT a v těle zprávu o příčině tohoto stavu tak, aby mohla být chyba napravena manuálně. Pokud byla chyba vrácena ve zprávě, ve které IMC očekávala id nově vzniklé entity, pak je id této entity v cílovém systému nastaveno na hodnotu 0.

4.3.5 Zabezpečení komunikace

Z důvodů uvedených v kapitole 3.7, především nedokončené analýzy bezpečnostních možností a omezení současných systémů[15] Čf byla nakonec komunikace zabezpečena pomocí Basic access authentication. Tato autentizace požaduje, aby zdrojový systém poslal IMC v hlavičce požadavku jméno a heslo, díky němuž IMC ověří jeho identi-

[15] V současné chvíli není jasné, zda je ve všech zainteresovaných systémech dosažitelná implementace bezpečnosti pomocí navrhovaného způsobu OAuth2 a protokolu HTTPS, současně analýza zabezpečení vnitřní sítě Čf je nad rámec této práce.

tu. Toto jméno a heslo je nastaveno v souboru `application.properties`. IMC pak zasílá cílovým systémům zprávy, které v hlavičce také obsahují přihlašovací údaje. Tyto údaje jsou konfigurovány v souboru `Components.xml`. Důležité je, že veškerá komunikace probíhá nešifrovaně přes protokol HTTP, proto případný útočník může hesla odposlechnout. V případných navazujících pracích je zapotřebí udělat analýzu možností zabezpečení na straně systémů Čf a komunikaci zabezpečit pomocí HTTPS protokolu či jiného bezpečně šifrovaného spojení.

4.4 Konfigurace aplikace

4.4.1 Konfigurace toku zpráv

Aplikace obsahuje dva konfigurační XML soubory. Pomocí prvního lze konfigurovat tok zpráv (`Messages.xml`), druhý slouží ke konfiguraci komponent, které mají zájem účastnit se komunikace (`Components.xml`) tak, aby byl splněn požadavek REQ 3. Oba dokumenty jsou umístěny v kořenovém adresáři aplikace. U spustitelného JAR souboru pak ve stejném adresáři, jako je tento JAR soubor. Pro parsování je použita knihovna `org.w3c.dom` a její třídy `Document, Element, Node` a `NodeList`.

Tok zpráv je konfigurován v souboru `Messages.xml`. Tento soubor obsahuje kořenový element `<component>`. Kořenový element obsahuje kolekci elementů `<resource>` s povinným atributem `name`, který udává název REST zdroje (výběr z možností `Item`, `CPAction`, `ExternalAction`). Každý element `resource` obsahuje kolekci elementů `<action>` s atributem `name`, udávajícím název události (`POST`, `PUT` nebo `DELETE`). Tato událost pak má neomezené množství cílových systémů, uvedených jako element `<target>`. Každý `target` na události `POST` ještě může obsahovat elementy `<needsIdOf>` specifikující, že zpráva o vzniku entity ve zdrojovém systému bude do cílového systému distribuována s informacemi o id v dalších cílových systémech. V tomto případě pak záleží na pořadí uvedení cílových systémů.

Výpis kódu 4 Ukázka konfigurace toku zpráv

```
<?xml version="1.0" encoding="UTF-8"?>
<component>
  <resource name="Item">
    <action name="POST">
      <target>
        <name>teplo</name>
        <needsIdOf>rudolf</needsIdOf>
      </target>
      <target>
        <name>mailer</name>
      </target>
    </action>
    <action name="PUT">
      <source>
        <name>rudolf</name>
      </source>
      <target>
        <name>teplo</name>
      </target>
      <target>
        <name>mailer</name>
      </target>
    </action>
  </resource>
</component>
```

Adresy dalších komponent, společně s hesly, které zabezpečují komunikaci, jsou konfigurovány v souboru `Components.xml`.

Výpis kódu 5 Ukázka adresování komponent

```
<?xml version="1.0" encoding="UTF-8"?>

<components>
  <component address="http://localhost:8082"
  authorization="orchestrusername:orchestrpwd">orchestr</component>
  <component address="http://localhost:8083"
  authorization="rudolfusername:rudolfpwd">rudolf</component>
  <component address="http://localhost:8084"
  authorization="ticketingusername:ticketingpwd">ticketing</component>
</components>
```

4.4.2 Přidání nové komponenty nebo mapovaného číselníku

Mapované číselníky pro každý zdroj je zapotřebí konfigurovat přímo ve zdrojovém kódu, konkrétně v souboru `resouces.mapping.EnumMapping.java`. Jsou zde uvedeny tři informace – jak se číselník jmenuje (toto jméno koresponduje s tabulkou v databázi a s adresou číselníku v ostatních systémech pro `GET` a `POST` požadavky), jak se jmenuje id číselníku (to je používané v těle HTTP požadavků) a které zdroje používají které číselníky.

Názvy a adresy komponent a názvy jejich id jsou uvedeny v souboru `resources.StringConstants.java`. Pokud je komponenta mapovaná, je zapotřebí ještě toto mapování uvést v souborech `MappedEntity.java`, `MappedEntityIdResolver.java` a v databázových tabulkách. Tyto změny jsou podrobně popsány v Manuálu k rozšíření na CD.

4.4.3 Soubor application.properties

Aplikace je konfigurována v souboru `application.properties`. Zde je konfigurováno logování událostí (název logovacího soubor), připojení k databázi (typ databáze a connection string) a port, na kterém běží integrovaný aplikační server Tomcat.

5 Testování

5.1 Unit testování

Unit (jednotkové) testování je automatické testování pro ověření korektní implementace dílčích částí nebo jednotek zdrojového kódu. [26] Různé definice unit testování definují nejrůznějšími způsoby, ale shodují se na základních třech aspektech: [27]

- Unit testy by se měly zaměřovat na malé části softwaru.
- Unit testy jsou vytvářeny samotnými programátory.
- Provedení unit testů by mělo být výrazně rychlejší, než provedení jiných druhů testů.

Předmětem diskuze je, co vlastně má být testovanou jednotkou. Objektově orientovaný přístup za tuto jednotku často považuje třídu, procedurální přístup pak např. jednotlivou funkci. Žádný přístup však není apriori špatný, vždy záleží na kontextu aplikace a na domluvě týmu. [27]

Důležitý aspekt unit testů je i to, zda jsou vytvářeny jako testy „pospolité“ nebo „izolované“ (sociable tests/solitary tests [28]). Pospolité testy volají reálné metody jiných vrstev, zatímco izolované metody používají mock objekty, které simulují správné chování částí kódu, jejichž správná funkce nás v tomto konkrétním testu nezajímá. Druhý přístup osobně považuji za čistší (jedna chyba v kódu rozbije jeden test, ne víc), ale obzvlášť v malých projektech se může jednat o zbytečnou režii, protože se v nich zdroj chyby dá vysledovat jednoduše.

Při testování aplikace jsem se řídil zásadou „good enough“[16] – pro logiku aplikace jsem vytvářel izolované unit testy s kompletními mock objekty, zatímco např. pro třídy, které fungují jen jako upravená implementace javovských tříd (`ResultSetExtractor`) jsem testy nevytvářel. Pokrytí kódu testy je znázorněno na Obrázku 7. Celkové pokrytí přesahuje 90 %, přičemž tři nejdůležitější třídy, obsahující logiku aplikace jsou pokryty ze 100 % (Controller), 98 % (Datová vrstva) a 94 % (Service).

[16] Produkt je přijatelný ve chvíli, kdy je dostatečně dobrý. Jakékoli zbytné vylepšování nad hranice „dostatečně dobrého“ je neekonomické. Co je „dostatečně dobré“ záleží na zamýšleném použití. [29]

Total Coverage: 91.30 %

Filename	Coverage	Total	Not Executed
philharmonic.dao.mapper.EntityResultSetExtractor	0.00 %	12	12
philharmonic.resources.LoggingConstants	0.00 %	1	1
philharmonic.AppConfig	0.00 %	7	7
philharmonic.Application	0.00 %	3	3
philharmonic.dao.mapper.EntityRowMapper	33.33 %	3	2
philharmonic.resources.ErrorMessages	81.25 %	16	3
philharmonic.utilities.AddressesParser	85.19 %	27	4
philharmonic.resources.StringConstants	87.50 %	8	1
philharmonic.utilities.MessagesParser	89.74 %	39	4
philharmonic.utilities.JsonUtil	93.02 %	43	3
philharmonic.service.IMCService	94.04 %	319	19
philharmonic.dao.IDaoImpl	97.62 %	42	1
philharmonic.model.ErrorHolder	100.00 %	9	0
philharmonic.model.Message	100.00 %	16	0
philharmonic.model.MappedResource	100.00 %	1	0
philharmonic.model.Enum	100.00 %	6	0
philharmonic.model.Component	100.00 %	6	0
philharmonic.model.MappedEntity	100.00 %	11	0
philharmonic.utilities.MessageSender	100.00 %	29	0
philharmonic.utilities.MappedEntityIdResolver	100.00 %	34	0
philharmonic.resources.mapping.EnumMapping	100.00 %	24	0
philharmonic.controller.IMCController	100.00 %	34	0
Total	91.30 %	690	60

Obrázek 7 Pokrytí kódu unit testy

Testů je celkem 78. U Controlleru (10 testů) je testováno, že metody naslouchají správným akcím na správných adresách a volají servisní metody se správnými parametry. U datové vrstvy (4 testy) je testováno, že její metody (`create`, `get`, `delete` a `update`) sestavují korektně dotazy na databázi. Servisní vrstva (43 testů) testuje především tři hlavní metody, volané Controllerem – `process` `POST`, `PUT` a `DELETE` `request`. U každé z těchto metod je testováno, zda při svém spuštění pro různé kombinace vstupů vrací požadovaný status, případně správné chybové zprávy, zda odesílá správné zprávy cílovým komponentám a zda korektně komunikuje s datovou vrstvou a třídou, která zajišťuje přidávání a změny id zdrojů a číselníků v těle zpráv.

5.2 Akceptační testování

V časovém rozsahu, v němž byla aplikace vyvíjena, nebylo možné provést úpravy současných systémů Čf v takovém rozsahu, aby bylo řešení testováno v reálném provozu. Pro účely testování jsem proto vytvořil programy (dále nazývané jako mocky systémů), které navenek simulují požadované chování systémů Čf, definované API definicí. Dále jsem vytvořil akceptační testy, které pro kombinaci daného stavu aplikace, dané konfigurace a daného vstupu popisují požadovaný výstup. Akceptační testy byly schváleny Českou filharmonií a úspěšně provedeny.

Pro každý ze tří zdrojů akceptační testy testují čtyři základní scénáře.

1. **Bezproblémový průchod** - Cílové komponenty vrací HTTP status OK, všechny používané číselníky jsou namapovány.
2. **Průchod s nekonzistentním mapováním číselníků mezi systémy** - Cílové komponenty vrací na všechny zprávy HTTP status OK. Zdrojový systém posílá zprávy s číselníky, které IMC nezná (nejsou namapovány).
3. **Chybná volání metod** - Zdrojový systém volá metody IMC v rozporu s definovaným API, tedy volá metodu s id zdroje 0, volá metodu a id zdroje vůbec neuvádí, volá metodu a uvádí nečíselné id nebo volá metodu a tělo požadavku není validní JSON.

4. **Chyby a konflikty v cílových komponentách** – cílové komponenty neběží, vrací CONFLICT nebo vrací různé nedefinované chybové odpovědi.

Ukázka akceptačního testu pro vznik nové externí akce v systému Rudolf je v tabulce 7. Kompletní akceptační testy jsou na přiloženém CD.

Tabulka 7 Akceptační test pro vznik nové externí akce v systému Rudolf

Test 1.1.	
Název	Vznik nové akce v systému Rudolf (pomocí volání POST)
Očekávaný výsledek	IMC pošle všechny zprávy na definované adresy, položku namapuje, vrací OK
Input	POST `{"id": "1", "name": "Symfonický orchestr Českého rozhlasu", "dates": [{"date": "2015-04-16","time": "19:30:00"}, {"date": "2015-04-17","time": "19:30:00"}], "placeId": "201", "categoryId": "201", "description": "Musorgskij, Glier, Šostakovič.", "organizerId": "201", "cycleId": "201"}`
Výsledek mailer mock loggeru	1. Neobsahuje žádnou novou zprávu.
Výsledek ticketing mock loggeru	1. Obsahuje zprávu o zavolání POST 2. Tělo požadavku obsahuje `placeId: 301, categoryId: 301, cycleId: 301, organizerId: 301`
Výsledek web mock loggeru	1. Obsahuje zprávu o zavolání POST 2. Tělo požadavku obsahuje `placeId: 401, categoryId: 401, cycleId: 401, organizerId: 401` 3. Tělo požadavku obsahuje `ticketingId:{id vrácené systémem Ticketing}`
Výsledek IMC loggeru	1. Obsahuje zprávu o zavolání POST 2. Obsahuje zprávu o zavolání ticketing/ExternalAction POST + odpověď. 3. Obsahuje zprávu o zavolání web/ExternalAction POST + odpověď. 4. Obsahuje zprávu o odpovědi na původní požadavek s hlavičkou CREATED.
Výsledek databáze	1. V tabulce ExternalAction je záznam s hodnotami `rudolfId: 1, ticketingId:{id vrácené od systému ticketing}, webId:{id vrácené od systému web}`

6 Nasazení

Aplikace je distribuována jako spustitelný JAR[17] soubor. Pro nasazení je zapotřebí provést následující kroky. Kroky jsou zde popsány jen zběžně, podrobný popis je na přiloženém CD v Manuálu nasazení.

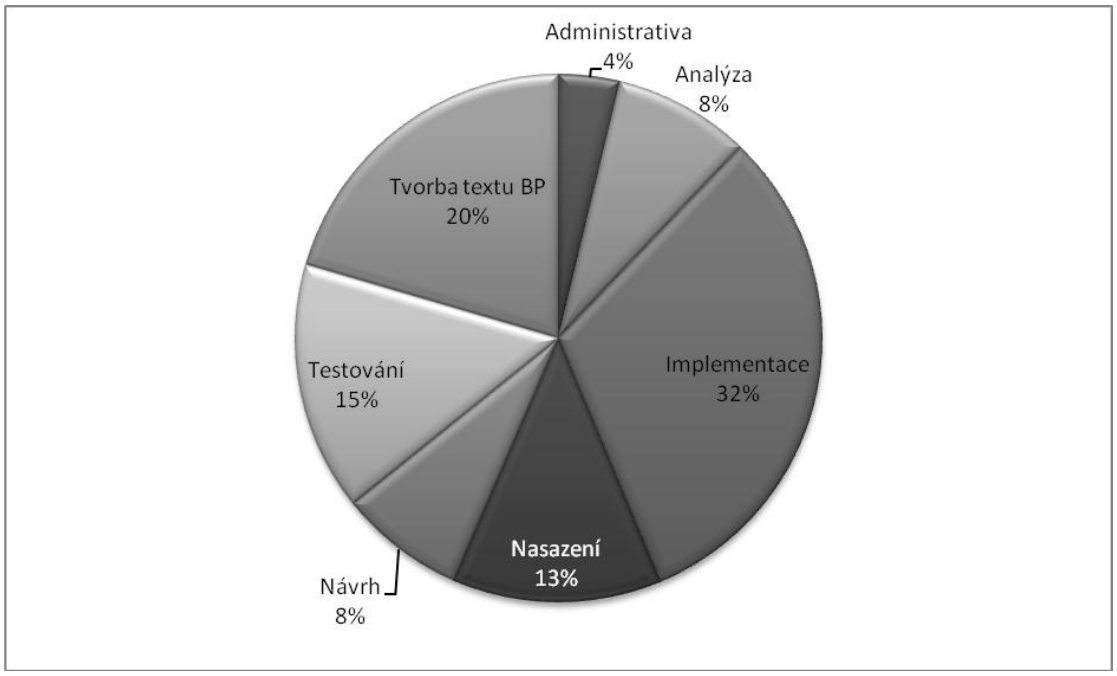
1. Na cílovém stroji nainstalovat JRE[18] verze 8.
2. Založit databázi, založit v ní tabulky pomocí SQL skriptů (příloha B), manuálně namapovat potřebné číselníky.
3. Do stejného adresáře, v němž se nachází spustitelný JAR umístit soubory `Messages.xml`, `Components.xml` a `application.properties`.
4. V `Messages.xml` nastavit tok zpráv pro komponenty, které se zapojí do komunikace, v `Components.xml` nastavit adresy těchto komponent a v `application.properties` provést nastavení připojení k databázi, logování a serveru.
5. Spustit aplikaci.

[17] JAR je souborový formát, používaný k distribuci programů napsaných v jazyce Java. Jsou v něm obsaženy především soubory zdrojového kódu s příponou .class a další zdroje, potřebné pro fungování programu (knihovny, obrázky apod.). [9]
[18] JRE je software, potřebný pro spuštění webové Java aplikace. [8]

7 Zhodnocení

7.1. Průběh a časová náročnost projektu

Průběh projektu měl z metodik vývoje nejblíže k iterativnímu vývoji – po určitých časových úsecích jsem dodával a nasazoval otestované funkční celky. Po každém nasazení proběhlo zhodnocení a plánování další práce, které často přineslo změny do původní analýzy a návrhu. Přestože některé aktivity projektu lze zařadit do více kategorií, na Obrázku 8 je znázorněna přibližná poměrná část, kterou jsem strávil prací na jednotlivých fázích projektu. Samotnou prací na projektu jsem strávil přibližně 260 hodin, psaním textu práce pak přibližně 70 hodin.



Obrázek 8 Časová náročnost jednotlivých fází projektu

7.2 Možnosti dalšího vývoje

Vývoj aplikace přinesl několik nevyřešených otázek a tím i potenciál k dalšímu rozšíření. První otázkou je způsob mapování rozsáhlých číselníků mezi systémy, konkrétně především smluvních subjektů (externích pořadatelů akcí v Rudolfinu) mezi systémem Rudolf a dalšími systémy. Tento problém je v mé práci analyzován pouze okrajově.

Druhou problematikou je zabezpečení komunikace mezi systémy. Ta je zabezpečena údaji sloužícími k autentizaci, ale tyto údaje jsou předávány v nešifrovaném HTTP

spojení. Druhou možností rozšíření této práce je tak implementování přenosu dat některým z bezpečně šifrovaných protokolů, např. HTTPS.

Třetí možností pokračování této práce je zvýšení robustnosti aplikace. Jedná se například o možnost kompletnější konfigurace (např. přidání nové komponenty bez zásahu do zdrojového kódu), o automatické řešení konfliktních a neočekávaných stavů (např. opakovaným odesíláním neúspěšných zpráv) nebo o definici kritických vazeb mezi systémy, kdy by se při některých neočekávaných stavech měla veškerá komunikace ukončit.

8 Závěr

Za nejdůležitější výstupy své práce považuji kompletní analýzu současného stavu, která definuje tři předávané sady objektů a události, které nad nimi lze vykonat v jednotlivých systémech – akce České filharmonie, akce externích pořadatelů a položky akcí. Druhým důležitým výstupem je návrh aplikace. Aplikace podle tohoto návrhu splňuje požadavky na rozšiřitelnost o nové komponenty a mapované číselníky a je kromě těchto mapovaných entit nezávislá na konkrétním obsahu komunikace.

Samotná aplikace je jednoduše nasaditelná, protože obsahuje vlastní integrovaný server, platformově nezávislá, stejně jako formát zpráv, které posílá mezi systémy.

Splnění požadavků je dokázáno pomocí úspěšného projití akceptačních testů, které slouží jako podrobná specifikace chování aplikace v různých stavech pro různé vstupy. Rozhraní komunikace je kompletně specifikováno pro každou jednotlivou komponentu, která se komunikace účastní a lze jej aplikovat na nové komponenty, které mohou být v budoucnu přidány.

Po provedení příslušných změn v současných systémech České filharmonie tak nyní nic nebrání uvedení aplikace do reálného provozu.

Literatura

[1] MAREČEK, David, MgA., Ph.D. ČESKÁ FILHARMONIE. *ZADÁVACÍ DOKUMENTACE pro zadání veřejné zakázky malého rozsahu v obdobě zákona 137/2006 sb.: Zavedení provozního IS a vytvoření internetové prezentace České filharmonie*. Praha, 2011.

[2] ČESKÁ FILHARMONIE. *KALENDÁRIUM 2014/2015*. Praha, 2014.

[3] Representational State Transfer. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Representational_State_Transfer

[4] CHROMÍK, Eduard, Pavel HEROUT, Pavel BRADA, Eduard CHROMÍK a ŠNAJBERK. UvodDoKomponent. In: *Wiki KIVu* [online]. 2015 [cit. 2015-04-26]. Dostupné z: http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage

[5] Java (programovací jazyk). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Java_%28programovac%C3%AD_jazyk%29

[6] Apache Maven. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://en.wikipedia.org/wiki/Apache_Maven

[7] BARRY, Douglas K. Application Server Definition. In: *Web Services, Service-Oriented Architectures, and Cloud Computing*[online]. [cit. 2015-04-26]. Dostupné z: http://www.service-architecture.com/articles/application-servers/application_server_definition.html

[8] JANSSEN, Cory. What is the Java Runtime Environment (JRE)?. In: *Techopedia - Where IT and Business Meet* [online]. [cit. 2015-04-26]. Dostupné z: http://www.techopedia.com/definition/5442/java-runtime-environment-jre

[9] JAR (file format). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://en.wikipedia.org/wiki/JAR_(file_format)

[10] Webová služba. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Webov%C3%A1_slu%C5%BEba

[11] Representational State Transfer. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Representational_State_Transfer

[12] Remote procedure call. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Remote_procedure_call

[13] SOAP. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/SOAP

[14] SOAP Syntax. *W3schools - Web Services Tutorial* [online]. [cit. 2015-04-26]. Dostupné z: http://www.w3schools.com/webservices/ws_soap_syntax.asp

[15] WERNECK, Pedro. web services - SOAP vs REST (differences). In: *Stack Overflow* [online]. 2013 [cit. 2015-04-26]. Dostupné z: http://stackoverflow.com/questions/19884295/soap-vs-rest-differences

[16] BLOCH, Joshua. How To Design A Good API and Why it Matters. [přednáška]. Google Tech Talks, 24. ledna 2007. In: *Youtube* [online]. [vid. 26. 4. 2015]. Záznam dostupný z https://www.youtube.com/watch?v=aAb7hSCtvGw

[17] BLANCHETTE, Jasmin Christian. The Little Manual of API Design. *Chair for Logic and Verification* [online]. Trolltech, 2008 [cit. 2015-04-26]. Dostupné z: http://www21.in.tum.de/~blanchet/api-design.pdf

[18] Java API for RESTful Web Services. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services

[19] APIARY LTD. *Apiary* [online]. [cit. 2015-04-26]. Dostupné z: https://apiary.io/

[20] BEDNÁŘ, Vojta. Průvodce českými startupy: Apiary. In: *Tyinternety.cz - Startupy, sociální sítě, ty internety!* [online]. 2012 [cit. 2015-04-26]. Dostupné z: http://www.tyinternety.cz/startupy/pruvodce-ceskymi-startupy-apiary/

[21] REST API Design - Resource Modeling. In: SUBRAMANIAM, Prakash. *Agile Development and Experience Design | ThoughtWorks* [online]. 2014 [cit. 2015-04-26]. Dostupné z: http://www.thoughtworks.com/insights/blog/rest-api-design-resource-modeling

[22] Basic access authentication. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Basic_access_authentication

[23] Multitier architecture. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://en.wikipedia.org/wiki/Multitier_architecture

[24] Vkládání závislostí. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Vkl%C3%A1d%C3%A1n%C3%AD_z%C3%A1vislost%C3%AD

[25] WEBB, Phillip, Dave SYER, Josh LONG, Stéphane NICOLL, Rob WINCH, Andy WILKINSON, Marcel OVERDIJK, Christian DUPUIS a Sébastien DELEUZE. Spring Boot Reference Guide. *Spring* [online]. 2013, 2015 [cit. 2015-04-26]. Dostupné z: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/

[26] Unit testing. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Unit_testing

[27] FOWLER, Martin. UnitTest. In: *Martin Fowler* [online]. 2014 [cit. 2015-04-26]. Dostupné z: http://martinfowler.com/bliki/UnitTest.html

[28] FIELDS, Jay. *Working Effectively with Unit Tests*. CreateSpace Independent Publishing Platform, 2014. ISBN 978-1503242708.

[29] MALÝ, Martin. Jak budeme psát webové aplikace za tři roky?. In: *Zdroják - o tvorbě webových stránek a aplikací* [online]. 2010 [cit. 2015-04-26]. Dostupné z: http://www.zdrojak.cz/clanky/jak-budeme-psat-webove-aplikace-za-tri-roky/

[30] Portable application. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-26]. Dostupné z: http://en.wikipedia.org/wiki/Portable_application

[31] Spring Boot. THE SPRING TEAM. *Spring* [online]. 2015 [cit. 2015-04-26]. Dostupné z: http://projects.spring.io/spring-boot/

[32] CLARKSON, Roy. 2014. Spring REST Service OAuth: A simple OAuth protected REST service built with Spring Boot and Spring Security OAuth. *Royclarkson (Roy Clarkson) on GitHub* [online]. [cit. 2015-05-05]. Dostupné z: https://github.com/royclarkson/spring-rest-service-oauth

[33] Vrstva persistence dat. 2006. ADÁMEK, Petr. *FI WIKI* [online]. [cit. 2015-05-13]. Dostupné z: https://kore.fi.muni.cz/wiki/index.php/Vrstva_persistence_dat

[34] PostgreSQL. 2004. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-13]. Dostupné z: http://cs.wikipedia.org/wiki/PostgreSQL

[35] Microsoft SQL Server. 2001-. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-13]. Dostupné z: http://cs.wikipedia.org/wiki/Microsoft_SQL_Server

[36] Maven Repository: org.codehaus.jackson » jackson-mapper-asl. 2009. *Maven Repository* [online]. [cit. 2015-05-13]. Dostupné z: http://mvnrepository.com/artifact/org.codehaus.jackson/jackson-mapper-asl

[37] Maven Repository: org.json » json: JSON in Java. 2007. *Maven Repository* [online]. [cit. 2015-05-13]. Dostupné z: http://mvnrepository.com/artifact/org.json/json

[38] Hamcrest: Hamcrest - library of matchers for building test expressions. 2009. *Google Code* [online]. [cit. 2015-05-13]. Dostupné z: https://code.google.com/p/hamcrest/

[39] *JUnit* [online]. 2014. [cit. 2015-05-13]. Dostupné z: http://junit.org/

[40] FABER, Szczepan. *Mockito.org: Tasty mocking framework for unit tests in Java* [online]. [cit. 2015-05-13]. Dostupné z: http://mockito.org/

[41] Log4j 2 Guide. 2013. *Apache Log4j 2* [online]. [cit. 2015-05-13]. Dostupné z: http://logging.apache.org/log4j/2.x/

[42] HTTPS: Hypertext Transfer Protocol Secure. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/HTTPS

A Definice API

A.1 Introduction

This documentation defines API of IMC (intermediate component listening to events in Czech Philharmonic IT systems).

A.2 Reference

A.2.1 Resource: Czech Philharmonic actions [/CPAction]

Action, whose organizer is Czech Philharmonic. Source system is Orchestr. The CPAction resource has following attributes:

- `id` - Id is id in source system (Orchestr), required attribute
- `placeId` - Place Id in source system (Orchestr)
- `categoryId` - Category Id in source system (Orchestr)
- `cycleId` - Cycle ID in source system (Orchestr)
- `name`
- collection of `dates` and `times`
- `description`
- collection of `players`

Body of CPAction requests [POST], [PUT]

```
{
  "id": "1",
  "name": "Tučňáci OPĚT v Rudolfinu",
  "dates": [
    {"date": "2015-04-14","time": "11:00:00"},
    {"date": "2015-04-15","time": "11:00:00"}
  ],
  "placeId": "12",
  "categoryId": "3",
  "description": "Poodhalte zákulisí klasické hudby a život
  velkého orchestru. Proč se na koncertě nemluví? Je dirigent na
  pódiu opravdu potřeba? Proč vypadají hudebníci jako tučňáci?
  Dokázala by vaše paní učitelka nebo maminka dirigovat Českou
  filharmonii? A jak dlouhá je cesta od skladatele k
  posluchači?",
  "players": [
    {"name": "Pavel LIŠKA","role": "Průvodce pořadem"},
    {"name": "Vojtěch JOUZA","role": "Dirigent"}
    {"name": "Alice NELLIS","role": "Režie"}
  ],
  "cycleId": "2"
}
```

Create CPAction [POST] - response

```
+ Response 201 (application/json)
  CREATED

+ Response 400 (application/json)
  + Body
    {
      "message": "Wrong syntax - your request does not have id."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Conflict - Entity with id 1 already exists in
      your system."
    }
```

Update CPaction [PUT] - response

```
+ Response 200 (application/json)
  OK

+ Response 400 (application/json)
  + Body
    {
      "message": "Wrong syntax - your request does not have id."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Conflict - Target system Teplo returned
      CONFLICT."
    }
```

A.2.2 Resource: Actions of external organizer [/ExternalAction]

Action, whose organizer is an external subject. The ExternalAction resource has following attributes:

- `id` - Id in source system (Rudolf), required attribute
- `name`
- `placeId` - Place Id in source system (Rudolf)
- `categoryId` - Category Id in source system (Rudolf)
- `cycleId` - Cycle Id in source system (Rudolf)
- `organizerId` -Organizer Id in source system (Rudolf)
- `description`
- collection of `dates` and `times`
- collection of `players`

Body of ExternalAction request [POST], [PUT]

```
{
  "id": "123",
  "name": "Symfonický orchestr Českého rozhlasu",
  "dates": [
    {"date": "2015-04-16","time": "19:30:00"},
    {"date": "2015-04-17","time": "19:30:00"}
  ],
  "placeId": "11",
  "categoryId": "14",
  "description": "Musorgskij, Glier, Šostakovič.",
  "organizerId": "8",
  "players": [
    {"name": "Kateřina Javůrková","role": "lesní roh"},
    {"name": "Petr Altrichter","role": "dirigent"}
  ],
  "cycleId": "3"
}
```

Body of ExternalAction request [DELETE]

```
{
  "name": "Symfonický orchestr Českého rozhlasu",
  "dates": [
    {"date": "2015-04-16","time": "19:30:00"},
    {"date": "2015-04-17","time": "19:30:00"}
  ],
  "placeId": "11",
  "categoryId": "14",
  "description": "Musorgskij, Glier, Šostakovič.",
  "organizerId": "8",
  "players": [
    {"name": "Kateřina Javůrková","role": "lesní roh"},
    {"name": "Petr Altrichter","role": "dirigent"}
  ],
  "cycleId": "3"
}
```

Create External action [POST] - response

```
+ Response 201 (application/json)
  CREATED

+ Response 400
  + Body
    {
      "message": "Body of your request is not valid JSON."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Entity with id 123 already exists in your
      system."
    }
```

Update External action [PUT] - response

```
+ Response 200 (application/json)
  OK

+ Response 400
  + Body
    {
      "message": "Your request cannot contain id 0."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Target system Teplo thrown an internal error."
    }
```

Delete External action [DELETE/{id}] - response

```
+ Response 200 (application/json)
  OK

+ Response 400
  + Body
    {
      "message": "Id in your request is non-numeric value."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Target system Teplo thrown an internal error."
    }
```

A.2.3 Resource: Action items [/Item]

Item of action can belong either to CP or to an external action.
The item has following attributes:

- `id` - Id in source system (Rudolf), required attribute
- `itemSubjectId` - Id of item subject
- `dateFrom`
- `timeFrom`
- `dateTo`
- `timeTo`
- `room` - Only text representation, not id

Body of Item request [POST], [PUT]

```
{
  "id": "132",
  "itemSubjectId": "11",
  "dateTimeFrom": "2014-12-26 00:00:00",
  "dateTimeTo": "2014-12-26 23:59:59",
  "room": "Sloupový sál"
}
```

Body of Item request [DELETE]

```
{
  "itemSubjectId": "11",
  "dateTimeFrom": "2014-12-26 00:00:00",
  "dateTimeTo": "2014-12-26 23:59:59",
  "room": "Sloupový sál"
}
```

Create Item of action [POST] - response

```
+ Response 201 (application/json)
  CREATED

+ Response 400
  + Body
    {
      "message": "Body of your request is not valid JSON."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Entity with id 132 already exists in your
      system."
    }
```

Update Item of action [PUT] - response

```
+ Response 200 (application/json)
  OK

+ Response 400
  + Body
    {
      "message": "Your request cannot contain id 0."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Target system Teplo thrown an internal error."
    }
```

Delete Item of action [DELETE/{id}] - response

```
+ Response 200 (application/json)
  OK

+ Response 400
  + Body
    {
      "message": "Id in your request is non-numeric value."
    }

+ Response 401
  Unauthorized

+ Response 409
  + Body
    {
      "message": "Target system Teplo thrown an internal error."
    }
```

A.3 Responses

There are following error responses returned by IMC:

- 400 - This response is returned if request is not sent correctly - there is missing id attribute, id is 0 or non-numeric value or request body is not valid JSON. In this case IMC does not send any messages to target systems.
- 401 - Each request must be secured by Basic access authentication - so it must contain header attribute "Authentication: Basic" followed by "Name:Password" in Base64 encoding.
- 409 - This response means that there occurred a conflict in one of two possible sources. First possible source of conflict is that there are some wrong ids - there is already such id (when sending POST message) or there is no such id (when sending DELETE). If there is no such id when sending PUT, then POST is invoked. Conflict returned on POST means: ID already exists in source system or target component returned an error.

B Obsah přiloženého CD

Adresář	**Obsah**
Aplikace	Hotová spustitelná aplikace IMC.jar Konfigurační soubor application.properties Konfigurační soubor Messages.xml Konfigurační soubor Components.xml
Dokumentace	Manuál k nasazení aplikace ve formátu .docx a .pdf Manuál k rozšíření aplikace ve formátu .docx a .pdf API systémů ve formátu .docx a .pdf
Testovací prostředí	Návod ke spuštění testovacího prostředí.pdf Scénáře akceptačních testů.pdf Soubory, potřebné ke spuštění testovacího prostředí (simulace systémů Čf)
Text bakalářské práce	Text bakalářské práce ve formátu .docx a .pdf
Zdrojové kódy	Zdrojové kódy aplikace