VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÉ SLUŽBY PRO INTEGRACI BMPS DLE STANDARDU BPMN 2.0
WEB SERVICES TO INTEGRATE BMPS IN BPMN 2.0 SPECIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID MIHOLA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. TOMÁŠ HRUŠKA, CSc.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Mihola David**
Program: Informační technologie
Název: **Webové služby pro integraci BPMS dle standardu BPMN 2.0**
Web Services to Integrate BPMS in BPMN 2.0 Specification
Kategorie: Paralelní a distribuované výpočty

Zadání:

1. Seznamte se s vybranými částmi specifikace BPMN 2.0 a vyberte vhodné BPMN bloky k implementaci.
2. Prozkoumejte vhodné metody integrace mezi BPMS a okolními systémy a analyzujte možnosti synchronizace procesu.
3. Navrhněte dle analýzy podle bodu 2. zvolené metody integrace, popř. synchronizace nebo výměny zpráv paralelních procesů např. definicí příslušných služeb pro zvolené BPMN bloky, nejlépe s uvedením back-end logiky a obsluhy v UI.
4. Dle provedeného návrhu proveďte vývoj a integraci potřebných MVC komponent s využitím platformy .NET Core a souvisejících nástrojů.
5. Vyhodnoťte dosažené výsledky a zhodnoťte případné možnosti dalšího rozvoje.

Literatura:

- Visual Studio: https://docs.microsoft.com/cs-cz/visualstudio/windows/?view=vs-2019&preserve-view=true
- C#: https://mva.microsoft.com/en-us/training-courses/c-fundamentals-for-absolute-beginners-16169
- MVC v .NET Core: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-5.0&tabs=visual-studio
- webové technologie: https://www.w3schools.com/
- BPMN: https://is.muni.cz/el/1433/jaro2014/PV165/um/46771256/pr_06_bpmn.pdf
- modelování: https://camunda.com/bpmn/

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hruška Tomáš, prof. Ing., CSc.**
Konzultant: Brabec Milan, Ing., Děkanát FIT VUT
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 4. února 2022

Abstrakt

Cílem této práce je vybrat a implementovat vhodné BPMN bloky zajišťující integraci s okolními webovými systémy, výměnu zpráv mezi paralelně běžícími BPMN procesy a jejich synchronizaci. Pro realizaci cílů byl implementován informační systém dle návrhového vzoru MVC. Systém především umožňuje nahrávat BPMN modely, konfigurovat je a vytvářet z nich BPMN procesy, které lze v systému řídit a řešit. Synchronizaci BPMN procesů jednotlivých organizací, které využívají tento systém, a výměnu zpráv mezi nimi specifikují uživatelé v BPMN modelech bloky Message Event a Signal Event. Systém následně provádí takto namodelovanou synchronizaci a výměnu zpráv. Integrace systému s okolními systémy je zajištěna dotazováním se na jejich aplikační rozhraní, která systém umožňuje konfigurovat v GUI. Nakonfigurovaná API lze začlenit do BPMN modelů blokem Service Task a automatizovaně je volat se získanými daty v rámci interpretace BPMN procesů. Při implementaci systému byl také kladen velký důraz na správné členění kódu a zabezpečení.

Abstract

The goal of this thesis is to chose and implement appropriate BPMN blocks, which enable integration with surrounding web based systems, exchange of messages between parallel BPMN processes and their synchronization. The goal was reached with an implementation of an information system, which is based on the MVC design pattern. The system primarily enables uploading of BPMN models, their configuration and creation of BPMN processes described by those models, which can be then managed and solved within the system. Synchronization of BPMN processes owned by different organizations, which are using instances of this system, and message exchange between them is specified by users in BPMN models with blocks Message Event and Signal Event. The system then executes modeled synchronization and message exchange. Integration of this system with other systems is ensured via Application Programming Interface calls, which can be configured in the GUI. Configured APIs can be integrated into BPMN models by using the Service Task block. That enables automation of API calls with data obtained during the interpretation of BPMN processes. Great emphasis was also placed on security and correct code structuring during the implementation.

Klíčová slova

Business Process Model and Notation, BPMN, informační systém, MVC, webové služby, API, procesní modelování, automatizace, zabezpečení

Keywords

Business Process Model and Notation, BPMN, information system, MVC, web service, API, process modeling, automation, security

Citace

MIHOLA, David. *Webové služby pro integraci BMPS dle standardu BPMN 2.0.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Tomáš Hruška, CSc.

Webové služby pro integraci BMPS dle standardu BPMN 2.0

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Tomáše Hrušky, CSc. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Mihola
27. dubna 2022

Poděkování

Děkuji panu prof. Ing. Tomáši Hruškovi, CSc., za odborné vedení práce a panu Ing. Milanu Brabcovi za užitečné rady a připomínky, které měly kladný dopad na výsledek práce.

Obsah

Kapitola 1

Úvod

Informační technologie jsou nedílnou součástí našich životů, nejen osobních, ale především těch pracovních. Počty profesí, u kterých je hlavní náplní ovládání počítače, neustále rostou. Na pozicích, kde se práce s počítačem zpočátku nevyžaduje, se pracovníci dříve či později také potkávají s úkoly administrativního typu, které použití počítače vyžadují. Zejména v dnešní době platí toto dvojnásobně. Díky moderním technologiím lidé mohou častěji zůstávat doma. Plánování postupů, řízení zaměstnanců a jejich vzájemná komunikace je omezena na emaily, telefonní a video hovory. Je důležité zajistit, aby interpretace takto předávaných informací byla jednotná a nedocházelo ke zbytečným nedorozuměním.

Komunikaci mají v náplni své práce především manažerské role. Tito pracovníci potřebují komunikovat nejen se svými podřízenými, ale také musí dohlížet nad plánováním nových procesů ve firmě a sledovat ty, které již probíhají. Často také musí sdílet informace o probíhajících procesech ve společnosti s jinými manažery, kteří jsou ze spolupracujících organizací, nebo prezentují firemní procesy investorům. Protože externisté zpravidla neznají způsob komunikace a plánování procesů v dané společnosti, je běžné, že je tato oblast standardizována.

Jeden ze standardů zabývající se převážně modelováním podnikových procesů je *Business Process Model and Notation 2.0*[1] (dále jen BPMN). Aplikace tohoto standardu je i předmětem této práce. Jelikož se jedná o poměrně rozsáhlý standard, zabývá se tato práce především těmi částmi, které specifikují komunikaci. Zejména pak automatizací komunikace mezi různými procesy. Může se jednat o komunikaci a synchronizaci procesů v rámci jedné organizace nebo o komunikaci a synchronizaci v rámci jednoho procesu, kterého se účastní více organizací, případně obojího. Zabývá se také dotazováním se různých internetových služeb a získáváním patřičných odpovědí, což umožňuje automatizaci administrativních úkonů procesu, které by jinak musel provádět člověk.

Tato práce je rozdělena do sedmi kapitol. Druhá kapitola popisuje procesní modelování, jeho techniky a tři konkrétní technologie, pomocí kterých lze procesy modelovat. Další kapitola hovoří o standardu BPMN, uvádí jeho cíl, rozsah použití, historii a především základní stavební prvky BPMN diagramů. Čtvrtá kapitola popisuje technologie pro implementaci aplikačních rozhraní, které umožňují integraci a vzájemnou komunikaci webových systémů. Tato kapitola také uvádí, jak lze zabezpečit komunikaci po internetu. V páté kapitole je uveden návrh informačního systému implementující konfiguraci a vykonávání procesů modelovaných BPMN diagramy. Dále se tato kapitola zabývá návrhem komunikace mezi instancemi tohoto informačního systému a komunikací s ostatními systémy pomocí jejich

[1] doslovně česky *Model a notace obchodního procesu*

aplikačních rozhraní. Šestá kapitola odůvodňuje výběr použitých programovacích nástrojů, architektury a návrhového vzoru. Poté tato kapitola popisuje způsob implementace důležitých oblastí navrženého informačního systému, zejména výměnu zpráv a dotazování se pomocí webových služeb. V závěrečné kapitole jsou shrnuty prozkoumané oblasti z teoretické části práce a dosažené výsledky z části praktické. Tato kapitola obsahuje i návrh na další rozvoj dosažených výsledků a vylepšení těch stávajících.

Kapitola 2

Modelování podnikových procesů

Tato kapitola se zabývá modelováním podnikových procesů. Popisuje, co je procesní modelování, jaké se využívají techniky a jak lze jednotlivé techniky rozdělit. Dále obsahuje popis několika konkretní nástrojů pro modelování.

2.1 Definice procesního modelování

Procesní modelování je v organizacích široce používaná metoda přispívající ke zvýšení povědomí a znalosti podnikových procesů. Pomáhá také k identifikaci a rozebrání organizační složitosti podnikového procesu na dílčí části a umožňuje jeho popis. Jedná se o přístup, který se obvykle snaží daný problém vystihnout graficky. Grafický popis znázorňuje, jak podniky provádějí své operace, z pravidla aspoň pomocí prvků vyznačující činnosti, události, stavy a tok řízení mezi nimi. Kromě zmíněných prvků mohou také procesní modely obsahovat informace týkající se příslušných dat, organizační a informační zdroje a případně jiné artefakty, jako jsou další zúčastněné strany či výkonnostní metriky. [22]

2.2 Techniky modelování podnikových procesů

Stávající techniky modelování lze řadit do tří kategorií. První kategorie zahrnuje intuitivní techniky grafického modelování, jako je *Program Evaluation and Review Technique* (dále jen PERT). Většinou se zabývají plánováním procesů, jejichž průběh ještě nezačal. Umožňují vést diskuze o podnikových požadavcích a optimalizaci procesů. Důležitým aspektem těchto technik bývá doba provedení jednotlivých úkonů procesu a samotného procesu jako celku.

Do druhé kategorie spadají techniky, které jsou založené na matematických či jiných rigorózních paradigmatech. Nejznámějším zástupcem této kategorie jsou *Petriho sítě*. Tyto techniky se obvykle používají pro analýzu procesů nebo jejich provádění. Za použití vhodného softwaru lze pomocí nich také provádět experimenty s procesními scénáři a spouštět simulace za účelem získání optimálního řešení [22].

Třetí kategorie tvoří jakýsi kompromis mezi první a druhou. Obsahuje techniky, které jsou dostatečně intuitivní, ale mají také jistý logicky základ, jasně definované stavební bloky, jsou obvykle standardizované a jejich grafické znázornění lze zapsat s použitím některého ze serializačních formátů. Tyto aspekty umožňují jejich přenositelnost a vývoj softwaru, který na základě modelu podnikového procesu umožňuje řídit jeho průběh. Mezi techniky z této kategorie řadíme *Event-driven Process Chain* (dále jen EPC) a již zmíněný standard BPMN.

Výběr vhodné techniky není jednoduchý, je srovnatelný například s tím, když softwarový inženýr vybírá programovací jazyk, pomocí kterého budou programátoři implementovat daný projekt. Proto následující sekce obsahují detailnější popis zmíněných technik.

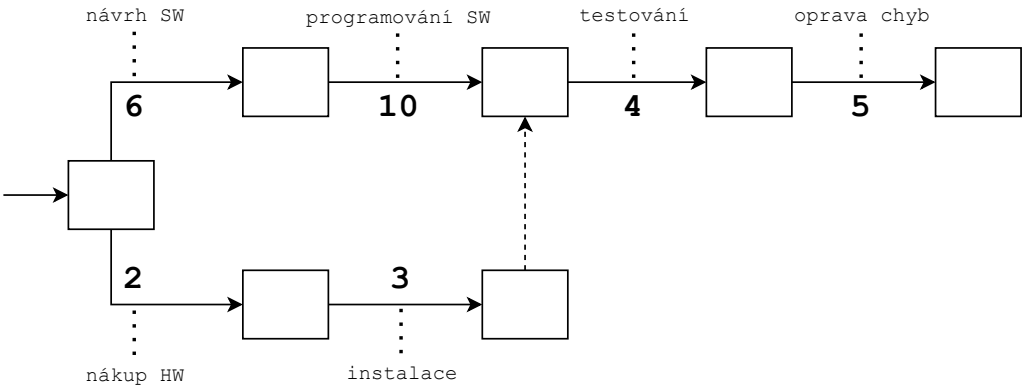
2.3 Program Evaluation and Review Technique

PERT diagram byl vytvořen Úřadem pro speciální projekty Námořnictva Spojených států amerických za účelem plánování projektu vývoje ponorkové balistické střely Polaris. Tvorbou PERT diagramu dochází k rozdělení projektu na jednotlivé úkoly, které lze poté samostatně analyzovat a odhadnout potřebnou dobu pro jejich realizaci. Na základě těchto informací může následně projektový manažer stanovit dobu pro dokončení projektu a případně vyčíslit jeho rozpočet. [15]

PERT diagram používá pro grafickou reprezentaci kruhy nebo obdélníky představující uzly. Každý uzel reprezentuje dosaženou událost nebo milník. Jednotlivé uzly jsou spojeny pomocí šipek tvořených plnou čarou, které značí pořadí takto spojených událostí nebo milníků. To znamená, že pokud šipka směřuje z uzlu č. 1 do uzlu č. 2, musí být událost reprezentovaná prvním uzlem dokončena před tím, než započne práce na dokončení události či milníku reprezentovaného druhým uzlem. Samotné šipky pak reprezentují plnění jednotlivých úkolů a je jim přiřazena časová složitost. [15]

Uzly, které jsou ve stejné fázi vykonávání ale v různých úkolových řadách, se označují jako paralelní. Takto vzniklé události jsou na sobě nezávislé a probíhají současně. K vytvoření paralelních úkolových řad dochází tehdy, když je z uzlu vedeno více šipek. Uzly mezi paralelními větvemi diagramu mohou být spojeny přerušovanou šipkou. Tento druh šipky nevyznačuje plnění úkolu a nemá tedy ani přiřazenou časovou složitost. Slouží pro synchronizaci mezi paralelními větvemi a zajišťuje, že událost, ze které vede šipka, je dokončena před událostí, do které tato šipka míří. [7]

Konkrétní příklad PERT diagramu popisující vývoj vestavěného systému může vypadat například následovně.



Obrázek 2.1: PERT diagram popisující vývoj vestavěného systému

2.4 Petriho sítě

Petriho sítě představil v roce 1962 Dr. Carl Adam Petri ve své disertační práci *Kommunikation mit Automaten*. Jedná se o mocný modelovací formalismus využívaný v informatice, systémovém inženýrství a mnoha dalších oborech. Petriho sítě kombinují matematicky definovanou teorii s grafickým znázorněním dynamického chování systémů. Teoretický aspekt Petriho sítí umožňuje systémy přesně modelovat a analyzovat jejich chování. Grafická reprezentace naopak umožňuje modelovaný systém vizualizovat. Právě tyto dva aspekty jsou důvodem úspěchu Petriho sítí. [28]

Petriho síť je formálně definována jako pětice $N = (P, T, I, O, M_0)$, kde:

(1) $P = \{p_1, p_2, ..., p_m\}$ je konečná množina míst,

(2) $T = \{t_1, t_2, ..., t_n\}$ je konečná množina přechodů, $P \cup T \neq \emptyset$ a $P \cap T = \emptyset$,

(3) $I : T \times P \to \boldsymbol{\mathcal{N}}$ je *vstupní funkce*, která definuje orientované křivky z míst do přechodů, kde $\boldsymbol{\mathcal{N}}$ je množina nezáporných celých čísel,

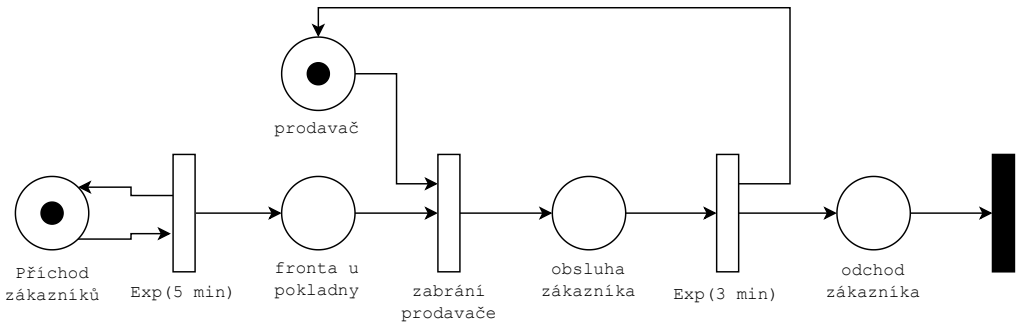
(4) $O : T \times P \to \boldsymbol{\mathcal{N}}$ je *výstupní funkce*, která definuje orientované křivky z přechodů do míst,

(5) $M_0 : P \to \boldsymbol{\mathcal{N}}$ je *počáteční značení*.

Značení znamená přiřazení žetonů do míst Petriho sítě. Orientovaná křivka směřující z místa p_j do přechodu t_i definuje p_j jako *vstupní místo* přechodu t_i. Orientovaná křivka směřující z přechodu t_i do místa p_j pak definuje p_j jako *výstupní místo* přechodu t_i. *Vstupní místo* je místo, ze kterého je žeton při provádění přechodu odebrán, a naopak *výstupní místo* je místo, do kterého je žeton po provedení přechodu přesunut. [28]

Graficky jsou místa v Petriho sítích značena elipsami, obvykle je ale znázorňujeme kružnicemi. Přechody jsou značeny obdélníky, orientované křivky šipkami. Žetony mohou být značeny tečkami nebo číslem, které vyjadřuje počet teček – žetonů. Značení žetonů se obvykle vpisuje do značky místa. Jednotlivé značky Petriho sítí lze dále textově anotovat za účelem určení jejich významu. [13]

Konkrétní příklad systému, modelovaného pomocí Petriho sítě, je obchod s jedním prodavačem, do kterého přichází zákazníci s časovými intervaly danými exponenciálním rozložením pravděpodobnosti. Prodavači zabere obsluha každého zákazníka časový interval daný také exponenciálním rozložením pravděpodobnosti. To odpovídá systému typu M/M/1 dle Kendalovy klasifikace [25]. Graficky je tento systém modelován následovně.



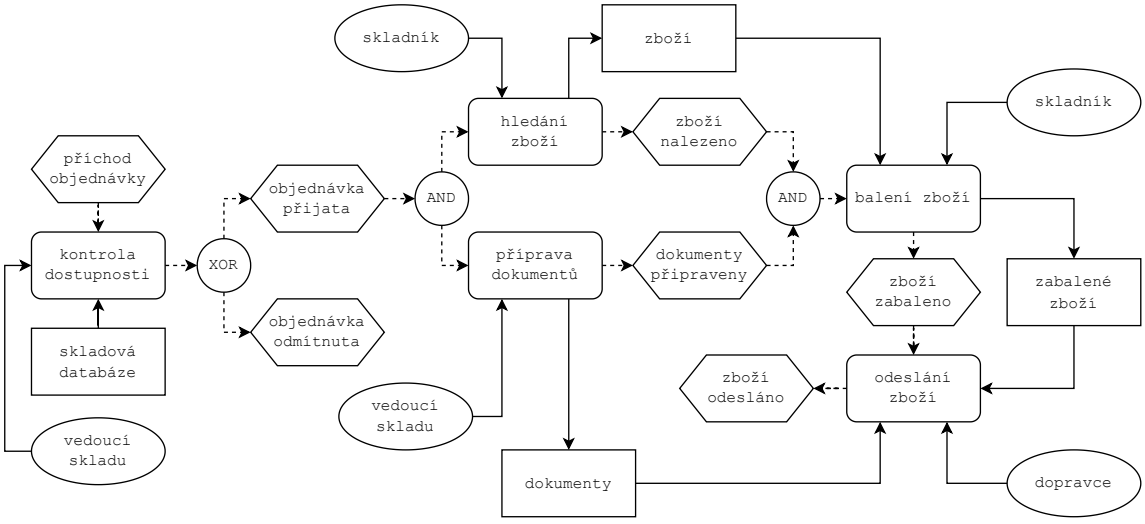
Obrázek 2.2: Petriho síť obchodu s jedním prodavačem

2.5 Event-driven Process Chain

Metoda modelování EPC byla vyvinuta firmou SAP v období mezi roky 1990 a 1992 [18]. Jedná se o metodu vizualizace po sobě následujících událostí a funkcí, pomocí které se zobrazuje logické načasování podnikových procesů. Označení *event-driven* tedy odpovídá popisu dynamické části podnikového procesu.

EPC diagramy jsou tvořeny událostmi, funkcemi, informačními objekty, organizačními jednotkami, logickými operátory a vztahy mezi prvky, z nichž jsou nejdůležitější řídící toky a toky informací. Události popisují, za jakých okolností funkce a procesy fungují nebo do jakého stavu vyústí. Graficky jsou značeny šestiúhelníky. Funkce popisují transformace vstupů z počátečního stavu do výsledného stavu. Graficky jsou značeny obdélníky se zaoblenými rohy. Informační objekty zobrazují objekty v reálném světě, které mohou být vstupy nebo výstupy funkcí. Graficky jsou značeny obdélníky se špičatými rohy. Organizační jednotky určují, které osoby nebo organizace jsou ve struktuře podniku odpovědné za určitou funkci. Graficky jsou značeny ovály. Operátory jsou typu AND, OR a XOR. Jejich význam je ale odlišný od běžného použití. Operace AND odpovídá současné aktivaci všech toků řízení. Operace OR odpovídá aktivaci jednoho nebo více řídících toků a operace XOR odpovídá rozhodnutí, který řídící tok zvolit. Graficky jsou logické operátory značeny kruhem s nápisem operátoru. Operátor AND může být také značen symbolem pro konjunkci, operátor OR symbolem pro disjunkci. Řídicí toky označují logické nebo chronologické závislosti mezi událostmi a funkcemi. Graficky jsou značeny přerušovanou šipkou. Toky informací znázorňují spojení mezi funkcemi a jejich vstupy a výstupy. Graficky jsou značeny plnou šipkou. [18][29]

Konkrétní příklad EPC diagramu, který popisuje obsluhu objednávky zboží, může vypadat následovně.



Obrázek 2.3: EPC diagram popisující obsluhu objednávky zboží

Kapitola 3

Popis vybraných úseků standardu BPMN

Tato kapitola se zabývá cílem vzniku a rozsahem použití standardu BPMN. Poté zmiňuje typy BPMN elementů, které standard pro modelování definuje, a popisuje ty nejpoužívanější. Nakonec kapitola obsahuje popis druhů diagramů, pomocí kterých lze vytvářet BPMN modely.

Kapitola vychází z volně dostupného anglicky psaného standardu *Business Process Model and Notation 2.0* [9]. Standard a zejména jednotlivé elementy, které definuje pro procesní modelování, jsou zde shrnuty a popsány v míře detailu dostatečné pro pochopení jeho základních principů a porozumění kapitolám 5 a 6. V dalších textech této kapitoly již není na standard explicitně odkazováno, nicméně čtenář vyžadující větší míru detailu je povzbuzen si ji ve standardu dohledat například na základě zde použitých názvů.

3.1 Cíl a rozsah použití BPMN

Standard BPMN byl vyvinut skupinou *Object Management Group*[1]. Primárním cílem standardu bylo poskytnout vhodnou notaci, která by byla srozumitelná všem zaměstnancům podniku, kteří s BPMN modelem budou pracovat nebo jej budou vytvářet. Mezi tyto zaměstnance patří analytici, kteří vytvářejí počáteční návrhy procesů, techničtí pracovníci, kteří jsou odpovědní za technologie, kterými se budou tyto procesy realizovat. Dále také řešitelé jednotlivých úkolů procesů a samozřejmě nesmíme opomenout manažerské pracovníky. Může se jednat o vedoucí týmu či vrcholný management, kteří dle vzniklých modelů řídí samotnou realizaci daného podnikového procesu.

Dalším neméně důležitým cílem bylo, aby vzniklé BPMN modely bylo možné předávat mezi různými systémy a vizualizovat je jednotnou grafickou notací vhodnou pro podnikové procesy. Za tímto účelem bylo vytvořeno schéma v XML, které popisuje, jak mají být jednotlivé prvky standardu serializovány, a také způsob jejich vizualizace.

Jelikož modelování podnikových procesů je velmi rozsáhlá oblast, standard BPMN umožňuje pohled na výsledné modely třemi diagramy. Označují se jako Process Diagram[2], Collaboration Diagram[3] a Choreography Diagram[4]. Při návrhu zmíněných diagramů do-

[1] OMG, stránky organizace jsou dostupné z `https://www.omg.org`.
[2] doslovně česky *diagram procesu*
[3] doslovně česky *diagram kolaborace*
[4] doslovně česky *diagram choreografie*

šlo skupinou OMG k přezkoumání již existujících notací a byly z nich vybrány ty nejlepší myšlenky, které jsou nyní zavedeny do standardu BPMN. Mezi přezkoumané notace patří například *UML Activity Diagram*, *Activity-Decision Flow Diagram* a *Event-Process Chains*.

Dle míry abstrakce, respektive hloubky detailu, lze BPMN modely dělit na modely deskriptivní, analytické a spustitelné. Mezi deskriptivními a analytickými modely je úzká hranice. Z deskriptivního modelu se stává analytický v okamžiku, kdy je modelovaný podnikový proces popsán dostatečně konkrétně a členění modelu na jednotlivé bloky je co možná nejvíce atomické. Spustitelné modely vznikají z analytických obvykle až v systému, ve kterém budou interpretovány (spouštěny), doplněním konkrétních atributů k jejich jednotlivým elementům i celému procesu.

3.2 BPMN elementy

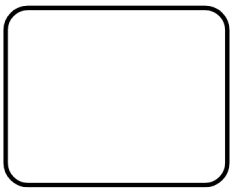
BPMN elementy jsou rozděleny do pěti kategorií, které se nazývají Flow Objects[5], Connecting Objects[6], Swimlanes[7], Data a Artifacts[8]. Rozdělení BPMN elementů do těchto kategorií umožňuje vytvořit jednoduchý a srozumitelný mechanismus pro vytváření modelů podnikových procesů libovolné složitosti a komplexity. Dalším důvodem je schopnost čtenáře dostatečně porozumět výslednému modelu, i když nezná přesnou sémantiku konkrétních elementů, protože je dokáže zařadit právě do zmíněných kategorií. Kategorie jsou dále ještě členěny do podkategorií, které jsou popsány v následujících sekcích.

3.2.1 Flow Objects

Tokové objekty jsou nejdůležitější BPMN elementy, které definují chování podnikového procesu. Obvykle jsou odkazovány jako BPMN bloky nebo jen bloky. Tokové objekty jsou dále členěny do tří podkategorií na Activities[9], Events[10] a Gateways[11].

Activities

Aktivity reprezentují podprocesy a úkoly. Podprocesy jsou neatomické a mohou opět obsahovat veškeré BPMN elementy. Slouží tedy pro vnořování BPMN modelů. Naopak úkoly jsou atomické a reprezentují druh práce, kterou je nutné vykonat. Práci popsanou úkolem často vykonává člověk. Podle toho, jakou práci je potřeba vykonat, lze úkoly dále konkretizovat a graficky odlišit dle následujícího výčtu.



Obrázek 3.1: Grafická reprezentace elementu Activity

[5] doslovně česky *tokové objekty*
[6] doslovně česky *spojovací objekty*
[7] doslovně česky *plavecké dráhy*
[8] doslovně česky *artefakty*
[9] doslovně česky *aktivity*
[10] doslovně česky *události*
[11] doslovně česky *brány*

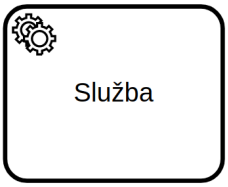
- **User Task**[12] představuje práci, kterou musí vykonat uživatel systému, jde například o vyplnění formuláře.

Obrázek 3.2: Grafická reprezentace bloku User Task

- **Manual Task**[13] představuje manuální činnost, kterou musí vykonat pracovníci, například naložení kamionu.

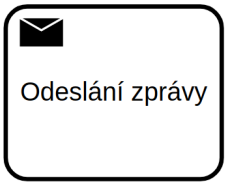
Obrázek 3.3: Grafická reprezentace bloku Manual Task

- **Service Task**[14] představuje volání webové nebo jiné služby, které může být plně automatizováno, nebo může vyžadovat zásah uživatele systému. Jedná se například o vyplnění vstupních údajů pro její volání.



Obrázek 3.4: Grafická reprezentace bloku Service Task

- **Send Task**[15] představuje odeslání zprávy jiné organizaci v modelu, například formou emailu.



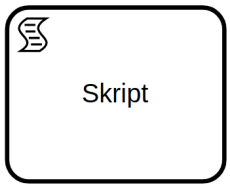
Obrázek 3.5: Grafická reprezentace bloku Send Task

[12] doslovně česky *uživatelský úkol*
[13] doslovně česky *manuální úkol*
[14] volně česky *služba*
[15] volně česky *odeslání zprávy*

- **Recieve Task**[16] představuje příjem zprávy od jiné organizace v modelu, například vyzvednutí dopisu na poště.

Obrázek 3.6: Grafická reprezentace bloku Recieve Task

- **Script Task**[17] představuje vykonání kódu napsaného v podporovaném jazyce. Opět může být úkol plně automatizován, nebo může vyžadovat zásah uživatele systému, například pro analyzování proběhlého výpočtu.



Obrázek 3.7: Grafická reprezentace bloku Script Task

- **Business Rule Task**[18] představuje propojení výpočetního systému BPMN modelů s výpočetním systémem podnikové logiky organizace a možnost tak interpretovat data dle požadavků podnikové logiky.

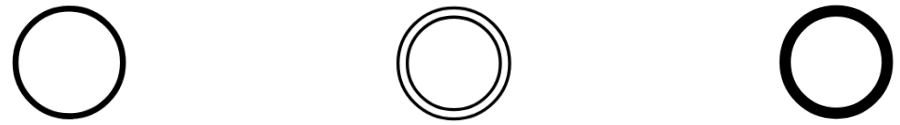
Obrázek 3.8: Grafická reprezentace bloku Business Rule Task

Events

Události spouštějí, modifikují nebo ukončují procesy. Obvykle jsou vykonány automaticky výpočetním systémem BPMN modelů bez zásahu uživatele. Z pohledu postupu procesu a umístění události v něm se události dělí na Start Event[19], Intermediate Event[20] a End Event[21].

[16] volně česky *příjem zprávy*
[17] volně česky *skript*
[18] volně česky *podnikové pravidlo*
[19] doslovně česky *startující událost*
[20] volně česky *průběžnou událost*
[21] doslovně česky *ukončující událost*

Dále jsou události na základě jejich významu konkretizovány a graficky odlišeny. Lze je navíc dle směru komunikace rozdělit na události typu Throwing[22] a události typu Catching[23]. Obecně platí, že události typu Catching obsluhují stav vzniklý událostí typu Throwing, pokud existuje komplementární dvojice. Následuje výčet běžně nejpoužívanějších událostí.



Obrázek 3.9: Grafická reprezentace bloku Event

- **Message Event**[24] je definován jako událost typu Throwing i Catching. Blok Throwing Message Event odesílá data bloku Catching Message Event nebo jinému BPMN elementu přijímající zprávy, který je na něj napojený pomocí Message Flow, viz sekce 3.2.2. V případě, že zpráva nenese žádná data, má pouze synchronizační význam.

Obrázek 3.10: Grafická reprezentace bloku Message Event

- **Signal Event**[25] je podobný bloku Message Event a je také definován jako událost typu Throwing a Catching. Od zprávy se liší tím, že nemá specifikovaného příjemce. Blok Throwing Signal Event vysílá data nebo pouze signalizuje svou aktivitu. Na odeslaná data respektive signalizaci může poté reagovat jakýkoliv blok Catching Signal Event.

Obrázek 3.11: Grafická reprezentace bloku Signal Event

- **Timer Event**[26] je definován pouze jako událost typu Catching, která reaguje na splnění časového předpokladu. Může jít například o spuštění modelovaného procesu ve stanoveném čase.

[22] volně česky *vytvářející*
[23] volně česky *reagující*
[24] volně česky *zpráva*
[25] volně česky *signál*
[26] volně česky *časovač*

Obrázek 3.12: Grafická reprezentace bloku Timer Event

- **Conditional Event**[27] je definován pouze jako událost typu Catching, která reaguje na splnění podmínky, například při dosažení určité teploty.

Obrázek 3.13: Grafická reprezentace bloku Conditional Event

Gateways

Brány umožňují a definují, jakým způsobem lze větvit procesy a jednotlivé větve zase slučovat. Vyhodnocení bran probíhá automaticky výpočetním systémem procesů. Brány jsou na základě svého významu konkretizovány a graficky odlišeny, stejně jako aktivity či události. Následuje výčet běžně nejpoužívanějších bran.

Obrázek 3.14: Grafická reprezentace bloku Gateway

- **Exclusive Gateway**[28] obsahuje podmínku a interpretace BPMN modelu pokračuje právě jednou z výstupních větví brány, která tuto podmínku splňuje.

Obrázek 3.15: Grafická reprezentace bloku Exclusive Gateway

- **Inclusive Gateway**[29] obsahuje podmínku a interpretace BPMN modelu pokračuje všemi výstupními větvemi brány, které tuto podmínku splňují.

[27] doslovně česky *podmíněná událost*
[28] doslovně česky *exkluzivní brána*
[29] doslovně česky *inkluzivní brána*

Obrázek 3.16: Grafická reprezentace bloku Inclusive Gateway

- **Parallel Gateway**[30] neobsahuje žádnou podmínku a interpretace BPMN modelu pokračuje všemi výstupními větvemi brány, umožňuje tedy současné provádění více úkolů či jiných BPMN elementů.

Obrázek 3.17: Grafická reprezentace bloku Parallel Gateway

3.2.2 Connecting Objects

Spojovací objekty umožňují definovat propojení tokových objektů a definovat příjemce zasílaných zpráv mezi těmito objekty, tvoří tak kostru BPMN diagramů. Dále je pomocí nich k tokovým objektům umožněno připojit data a anotace v podobě artefaktů. Dělí se na Sequence Flow[31], Message Flow[32], Association[33] a Data Association[34].

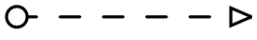
Sequence Flow

Sekvenční tok propojuje tokové objekty a jako jediný spojovací objekt určuje pořadí, ve kterém budou tokové objekty vykonány.

Obrázek 3.18: Grafická reprezentace elementu Sequence Flow

Message Flow

Tok zpráv spojuje tokové objekty odesílající zprávy s tokovými objekty přijímajícími zprávy a určuje tedy odesílatele a adresáty zpráv.



Obrázek 3.19: Grafická reprezentace elementu Message Flow

[30] doslovně česky *paralelní brána*
[31] doslovně česky *sekvenční tok*
[32] doslovně česky *tok zpráv*
[33] doslovně česky *asociace*
[34] doslovně česky *asociace dat*

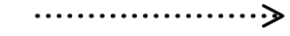
Association

Asociace zobrazuje vztahy mezi tokovými objekty a jejich artefakty popisujícími jejich význam.

Obrázek 3.20: Grafická reprezentace elementu Association

Data Association

Asociace dat umožňuje určit vstupní a výstupní data tokových objektů.



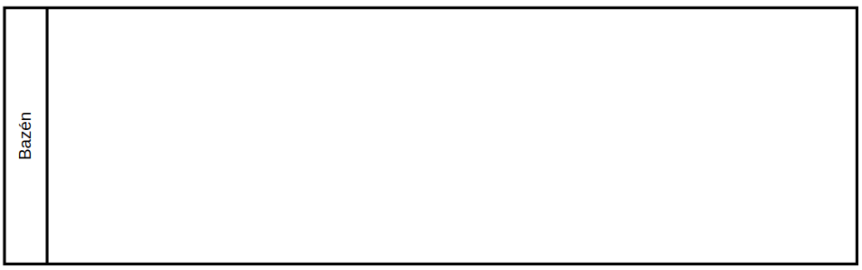
Obrázek 3.21: Grafická reprezentace elementu Data Association

3.2.3 Swimlanes

Plavecké dráhy jsou BPMN elementy, které reprezentují účastníky podnikového procesu. Dělí se na Pool[35] a Lane[36].

Pool

Bazény definují účastníky obvykle jako organizace, které se podílí na provádění daného BPMN modelu. Pro zaznačení komunikace mezi organizacemi se používají zprávy, které lze zasílat pouze mezi bazény.



Obrázek 3.22: Grafická reprezentace elementu Pool

Lane

Dráhy dále rozšiřují element Pool. Slouží ke specifikaci různých oddělení organizace nebo určují řešitelské role úkolů v dané dráze.

[35] doslovně česky *bazén*
[36] doslovně česky *dráha*

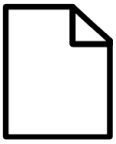
Obrázek 3.23: Grafická reprezentace elementu Lane

3.2.4 Data

Datové objekty umožňují zobrazit data potřebná pro vykonání podnikového procesu nebo data, která v rámci procesu vznikají. Dělí se na Data Objects[37], Data Inputs[38], Data Outputs[39] a Data Store[40].

Data Objects

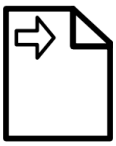
Datové objekty představují data, která prochází procesem. Jedná se například o dokumenty nebo dopisy.



Obrázek 3.24: Grafická reprezentace elementu Data Objects

Data Inputs

Vstupní data jsou data vstupující do celého podnikového procesu. Lze si je představit jako vstupní parametr procesu.

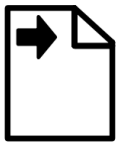


Obrázek 3.25: Grafická reprezentace elementu Data Inputs

Data Outputs

Vstupní data jsou data vzniklá během podnikového procesu, jsou jeho výsledkem. Lze si je představit jako produkt procesu.

[37] doslovně česky *datové objekty*
[38] doslovně česky *vstupní data*
[39] doslovně česky *Výstupní data*
[40] doslovně česky *datové úložiště*



Obrázek 3.26: Grafická reprezentace elementu Data Outputs

Data Store

Datové úložiště poskytuje mechanismus pro ukládání, získávání a aktualizaci dat, která mají přetrvávat i mimo rozsah podnikového procesu.

Obrázek 3.27: Grafická reprezentace elementu Data Store

3.2.5 Artifacts

Artefakty umožňují poskytnout čtenářům dodatečné informace o podnikovém procesu, na samotný proces ale nemají žádný vliv. Obsahují Group[41] a Text Annotation[42].

Group

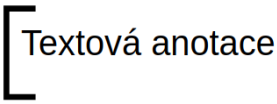
Skupina umožňuje ohraničit BPMN elementy a zdůraznit tak jejich souvislost.

Obrázek 3.28: Grafická reprezentace elementu Group

Text Annotation

Textová anotace umožňuje tvůrci podnikového procesu pomocí textového popisu přiblížit čtenářům vzniklého BPMN modelu význam procesu nebo jeho částí.

[41] doslovně česky *skupina*
[42] doslovně česky *textová anotace*



Obrázek 3.29: Grafická reprezentace elementu Text Annotation

3.3 Typy BPMN diagramů

Process Diagram

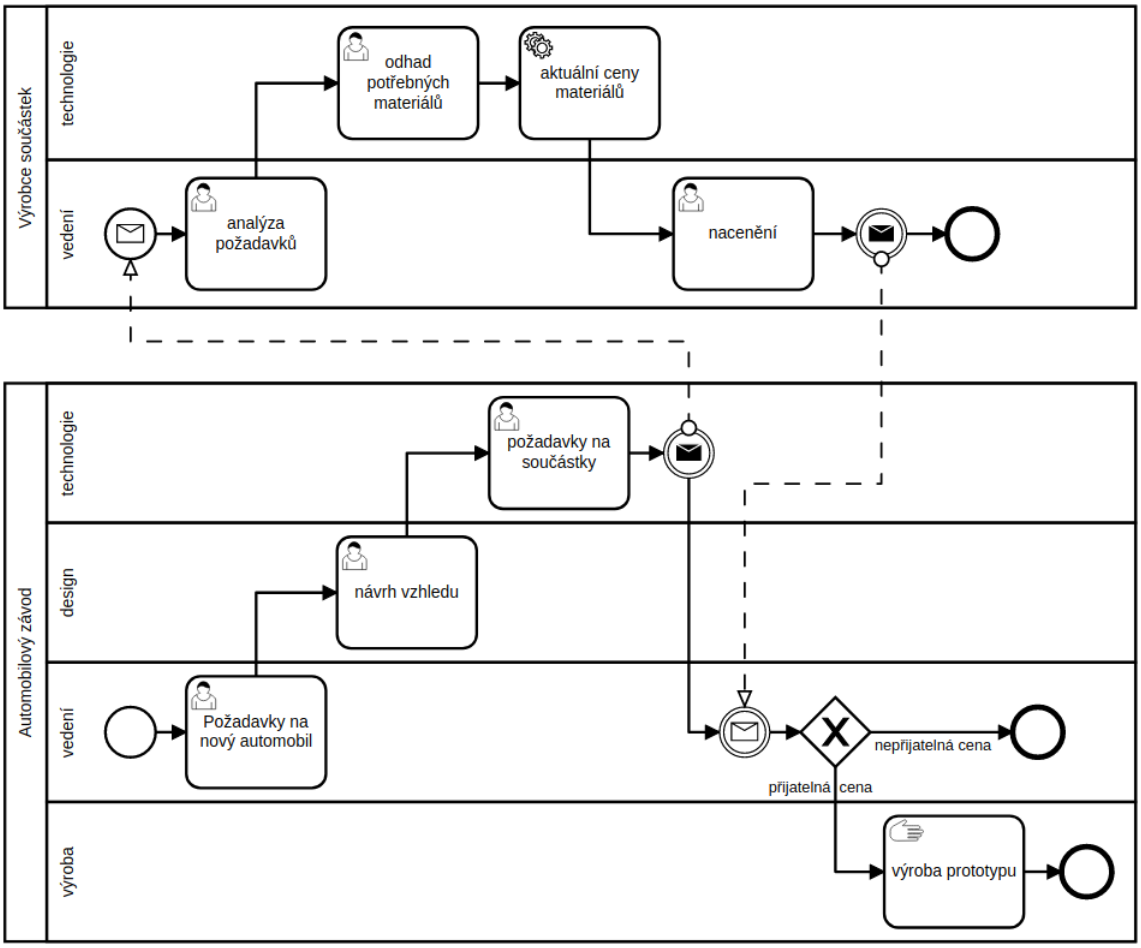
Procesní diagram je nejčastěji používaný typ BPMN diagramu. Modelují se pomocí něj podnikové procesy. Používá se k popisu pořadí událostí, úkolů a podmínek, za jakých mají být události a úkoly prováděny pro úspěšné splnění modelovaného podnikového procesu. Proces začíná vždy blokem Start Event a končí blokem End event nebo chybou.

Choreography Diagram

Diagram choreografie znázorňuje způsob, jakým účastníci koordinují vzájemné interakce. Důraz není kladen na provádění úkolů, ale spíše na výměnu informací mezi účastníky. Stejně jako proces choreografie začíná blokem Start Event a končí blokem End Event.

Collaboration Diagram

Diagram kolaborace je používán pro zobrazení různých účastníků. Jedná se obvykle o organizace a modeluje jejich vzájemné interakce. Každý účastník je reprezentován jedním elementem Pool, ve kterém je namodelován jeho podnikový proces. Procesy účastníků spolu interagují pomocí zasílání zpráv. Příklad BPMN modelu modelovaného diagramem kolaborace, který popisuje možný postup tvorby prototypu nového automobilu, je zobrazen na obrázku 3.30.



Obrázek 3.30: Příklad BPMN modelu modelovaného diagramem kolaborace

Kapitola 4

Metody integrace webových systémů a zabezpečení komunikace přes internet

V této kapitole je popsáno, jaké jsou možné způsoby integrace a vzájemné komunikace webových systémů. Způsob komunikace systému je obvykle založen na zveřejnění aplikačního rozhraní, které může být dotazováno okolními systémy. Aplikační rozhraní mohou být implementována za použití různých technologií. Tato kapitola zmiňuje ty, které jsou dnes nejpoužívanější. Na konci kapitoly jsou navíc identifikována možná nebezpečí při komunikaci na veřejné síti a probrány způsoby, jak jim lze předejít.

V prví části je popsán starší protokol SOAP[1], jehož míra využití postupně klesá a u nových aplikací se používá spíše ojediněle. Poté je popsána architektura rozhraní REST[2], která je dnes nejpoužívanější. Dále se kapitola zabývá dotazovacím jazykem GraphQL[3], který nabývá na popularitě a je možné, že v budoucnosti oba zmíněné principy předčí. [2]

4.1 SOAP – Simple Object Access Protocol

Jedná se o aplikační protokol, který je nástupcem staršího způsobu komunikace mezi systémy pomocí XML-RPC[4]. Na rozdíl od REST není závislý na HTTP[5] protokolu a lze jej například zapouzdřit i do protokolu SMTP[6]. [34]

Protokol je založen na serializačním formátu XML[7] a určuje strukturu posílaných zpráv, která je definována schématem napsaném v XSD[8] formátu. Každá zpráva je zabalena do obálky (*envelope*) a je tvořena hlavičkou (*header*), která však není povinná a ve zprávě může chybět, a tělem (*body*) zprávy, které obsahuje dotaz, respektive odpověď na dotaz. Dotaz, který žádá informace o dané knize, může vypadat například následovně. [6]

[1] Simple Object Access Protocol
[2] Representational State Transfer
[3] Graph Query Language
[4] XML Remote Procedure Call
[5] Hypertext Transfer Protocol
[6] Simple Mail Transfer Protocol
[7] *Extensible Markup Language*
[8] XML Schema Definition, schéma SOAP je dostupné z `http://schemas.xmlsoap.org/soap/envelope/`

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body xmlns:m="http://library.cz/quotation">
      <m:GetBook>
         <m:Id>1</m:Id>
      </m:GetBook>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Kde element `m:GetBook` označuje jméno volané procedury a element `m:Id` obsahuje hodnotu parametru *Id* procedury *GetBook*. Odpověď na tento dotaz obsahující informace o žádané knize je následující.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body xmlns:m="http://library.cz/quotation">
      <m:GetBookResponse>
         <m:Id>1</m:Id>
         <m:Title>Some Book</m:Title>
         <m:Pages>123</m:Pages>
         <m:Author>Name Surname</m:Author>
      </m:GetBookResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Výhodou této komunikace mezi systémy je, že se jedná o standardizovaný protokol, což jasně určuje strukturu zprávy. Jak lze vidět na příkladech, nevýhodou je objemnost přenášených dat, která je dána použitím serializačního formátu XML. S tím také souvisí náročné zpracování a validace příchozích dat. [19]

4.2 REST – Representational State Transfer

REST není protokol, jedná se pouze o návrh architektury, který by měly splňovat systémy, které chtějí implementovat tak zvané RESTful aplikační rozhraní. Na rozdíl od SOAP, který zapouzdřuje identifikaci zdroje a volání vzdálené procedury, se REST zabývá pouze datovými strukturami a předáváním jejich stavu. REST je tudíž závislý na protokolu, na kterém je implementován, což je protokol HTTP. Zdroje jsou u RESTful aplikačních rozhraní identifikovány na základě URL[9], která má obecně následující tvar
`<protokol>://<host>/<cesta aplikace>/<typ zdroje>/<id zdroje>` [4], kde

- *protokol* je `http` nebo `https`,
- *host* je doménové jméno serveru a případně port, na kterém služba běží,
- *cesta aplikace* slouží jako jmenný prostor sdružující typy zdrojů, tato část může být vynechána,

[9] Uniform Resource Locator doslovně česky *jednotný lokátor zdroje*

- *typ zdroje* většinou udává jméno datové struktury zdroje, se kterou služba na dané URL pracuje,
- *id zdroje* udává identifikaci konkrétního objektu, který je dotazován, nebo (pokud je tato část vynechána) je dotaz vztažen na celou kolekci objektů daného zdroje.

Takovéto URL vedoucí na zdroje se označují jako koncové body aplikačního rozhraní. Konkrétně může URL vypadat například takto: `http://library.cz/Books/1`.

Nad každým zdrojem jsou definované operace, které jsou obecně známé pod akronymem CRUD [4]. Implementace jednotlivých CRUD operací je navázána na protokol HTTP, kde

- **C** zastává operaci *Create* – vytvoření objektu/objektů daného typu, operace se váže na HTTP dotaz typu `POST`,
- **R** zastává operaci *Read* – čtení objektu/objektů, operace se váže na HTTP dotaz typu `GET`,
- **U** zastává operaci *Update* – aktualizaci objektu/objektů, operace se váže na HTTP dotaz typu `PUT`,
- **D** zastává operaci *Delete* – odstranění objektu/objektů, operace se váže na HTTP dotaz typu `DELETE`.

Pokud klient zašle dotaz na některý koncový bod implementující danou CRUD operaci, je mu vrácena odpověď s patřičným HTTP stavovým kódem a případně i s daty. Stavový kód indikuje úspěch nebo neúspěch dané operace. Příklad odpovědi získané po odeslání `GET` dotazu, čili operace *Read*, na zmíněnou URL `http://library.cz/Books/1` vypadá následovně. Data jsou v tomto příkladě serializována do formátu JSON[10] a obsahují informace o žádané knize v dotazu. Odpověď obsahuje i základní HTTP hlavičku.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 85

{
   "id": 1,
   "title": "Some Book",
   "pages": 123,
   "author": "Name Surname"
}
```

Samotný REST nedefinuje formát, kterým jsou předávána data. Nejčastěji jsou data serializována do formátu JSON, méně často do XML, lze se také setkat s aplikačními rozhraními, která v odpovědi vrací přímo HTML obsah. Jiné formáty dat jsou méně obvyklé. Daný systém může implementovat aplikační rozhraní podporující více formátů, pak má klient možnost specifikovat formát přenášených dat v hlavičce HTTP požadavku v poli `Accept` [4].

Mezi výhody návrhu architektury REST patří zejména jednoduchost implementace a široká podpora různými knihovnami a frameworky[11]. Další výhodou je integrace s protokolem

[10] *JavaScript* Object Notation
[11] rámcová řešení

HTTP a z toho plynoucí jednodušší zpracování dotazů a také flexibilita při výběru formátu přenášených dat [19]. Hlavní nevýhodou REST je, že se nejedná o standard. To způsobuje odlišnosti u jednotlivých implementací, ať už jde o použitý formát dat, nebo i způsob sestavování URL koncových bodů.

4.3 GraphQL – Graph Query Language

GraphQL také není protokol, jedná se o silně typovaný jazyk definující syntax, pomocí něhož lze vytvořit dotaz žádající přesně konkrétní data. To znamená, že aplikační rozhraní využívající technologii GraphQL přesně nedefinuje strukturu dat odesílaných v jednotlivých dotazech a odpovědích na ně. Struktura dat zaslaných v odpovědi je dána na základě dotazu, který sestavuje a zasílá klient [14]. Server definuje pouze schéma popisující všechny možné dotazy, které může klient sestavit. Klient navíc může dotaz před jeho odesláním na základě schématu validovat [2]. Každé schéma musí mít objekt typu `Query` a může, ale nemusí mít objekt typu `Mutation`, tyto objekty definují vstupní body pro všechny GraphQL dotazy [27]. Nejčastěji je použití GraphQL vázáno s protokolem HTTP a jeho metodami `GET` a `POST`. GraphQL lze ale také zapouzdřit například do protokolu MQTT[12] [8], není tedy závislý na zapouzdřujícím protokolu [33]. Schéma může vypadat například následovně.

```
type Book {
   id: ID
   title: String!
   pages: Int
   author: Author
}

type Author {
   name: String
   books: [Book]
}

type Query {
   book(id: ID): Book
   books: [Book]
}
```

Objekty typu `Book` a `Author` jsou definované uživatelem a obsahují buď jeden ze skalárních typů (`Int`, `Float`, `String`, `Boolean`, `ID`), uživatelem definovaný výčtový typ nebo jiný uživatelem definovaný objekt [27]. Hranaté závorky definují pole a vykřičník specifikuje, že daná hodnota nesmí být prázdná (`null`). Příklad validního dotazu na základě tohoto schématu je pak následující.

[12] MQ Telemetry Transport

```
query GetBook {
   book(id: 1) {
      title
      author {
         name
      }
   }
}
```

Odpověď na takto specifikovaný dotaz vypadá následovně. Data odpovědi jsou serializována do fomátu JSON, což je také jediný formát, který GraphQL používá [2].

```
{
   "data": {
      "book": {
         "title": "Some Book",
         "author": {
            "name": "Name Surname"
         }
      }
   }
}
```

Hlavní výhodou aplikačního rozhraní implementovaného pomocí GraphQL je, že server odesílá v odpovědi pouze ta data, o která si klient v dotazu žádá, jak je vidět v odpovědi s porovnáním k SOAP a REST. Další výhodou je jeho nezávislost na použitém protokolu aplikační vrstvy. Naopak nevýhodami jsou jeho zatím malá rozšířenost, komplexní dotazy mohou představovat příliš velkou zátěž na server a problémy mohou nastat i u implementace ukládání výsledků dotazů do HTTP vyrovnávacích pamětí[13]. [26]

4.4 Zabezpečení komunikace přes internet

Zabezpečení komunikace přes internet, nebo obecněji bezpečnost na internetu, je velmi rozsáhlé téma. Tato sekce se zabývá zejména tím, co musí dva systémy, které spolu chtějí přes internet komunikovat, zajistit, aby jiný systém připojený k internetu nemohl této skutečnosti zneužít. Za zneužití bude považováno falešné vydávání se za jeden ze systémů, odposlech jejich komunikace, zásah do komunikace pozměněním zasílaných zpráv a opakované zaslání zachycené zprávy, která již byla předtím doručena. Jak lze zajistit, aby ke zmíněným zneužitím nedošlo, je popsáno v následujících bodech.

4.4.1 Autentizace

Autentizace[14] zajišťuje, že uživatel je tím, za koho se vydává. Autentizace obvykle probíhá na základě poskytnutí nějaké tajné informace. Tuto informaci může znát nebo k ní mít přístup pouze uživatel, který má být autentizován. Jde například o uživatelské jméno a heslo, biometrické údaje, bezpečnostní žeton atd. Autentizaci lze také zajistit digitálním podpisem. [31]

[13] HTTP servery, které si dočasně ukládají výsledky dotazů z primárních serverů daných HTTP stránek.
[14] anglicky *authentication*

4.4.2 Důvěrnost

Důvěrnost[15] zajišťuje, že obsah zpráv nemůže číst neoprávněná osoba. To lze zajistit buď odepřením přístupu ke zprávě, což není ale na internetu obecně možné, nebo šifrováním (utajením) zprávy, což sice útočníkovi umožní číst zasílaná data, ale neumožní mu získat informaci, kterou obsahují. Data pro něj bez dešifrování budou bezcenná.

Nauka o šifrování se nazývá kryptografie, v informatice hovoříme především o symetrické a asymetrické kryptografii. Symetrická kryptografie používá jeden tajný klíč, kterým odesílatel šifruje zprávu a příjemce ji tím samým klíčem dešifruje. Nejpoužívanější algoritmy pro symetrické šifrování jsou v současné době například AES[16], *ChaCha20* či IDEA[17] [24]. Známější algoritmy jako DES[18], 3DES[19] a *Blowfish* nejsou dnes již považované za bezpečné, byla u nich objevena bezpečnostní rizika nebo způsob, jak lze šifru překonat hrubou silou v dostatečně krátkém čase [24]. Síla šifry však závisí především na kvalitě klíče a ta, vzhledem k tomu, že většina symetrických klíčů je generována jako pseudonáhodná posloupnost bitů, je dána jeho délkou. Za dostatečně bezpečné jsou dnes považovány klíče o délce aspoň 128 bitů [20], u kterých není u běžných počítačů reálné jejich prolomení hrubou silou v přijatelně krátkém časovém úseku. Pro zajištění bezpečnosti i do budoucna je ale vhodné používat klíče o délce 256 bitů [20]. Doposud bylo dokázáno, že kvantové počítače mohou v nejhorším případě při útoku hrubou silou snížit počet pokusů o dešifrování z 2^n u běžných počítačů na $2^{n/2}$, kde n je délka klíče v bitech [5]. To znamená, že klíč o délce 256 bitů bude i při útoku na kvantovém počítači dostatečně bezpečný, jeho bezpečnost bude odpovídat zhruba bezpečnosti 128 bitového klíče u běžných počítačů.

U asymetrické kryptografie se používá dvojice klíčů, soukromý a veřejný. Pro tyto klíče platí, že jsou matematicky svázány [23]. Odesílatel šifruje zprávu veřejným klíčem, který může znát kdokoliv, buď je veřejně dostupný nebo jej odesílatel od příjemce získá na počátku komunikace, která není ale šifrovaná. Pro dešifrovaní zprávy je potřeba soukromý klíč, ten zná jen a pouze příjemce. Takto je zajištěno šifrování v jednom směru komunikace, pro opačný směr se postupuje obdobně. Obvykle si tedy na počátku komunikace respondenti vymění veřejné klíče a až poté probíhá komunikace šifrovaně. Pro asymetrickou kryptografii se používají například algoritmy RSA[20] a *ElGamal*, které jsou založené na složitosti výpočtu diskrétního algoritmu [1], či algoritmus ECC[21], který pracuje na bázi eliptických křivek. Stejně jako u symetrické kryptografie závisí bezpečnost šifry především na délce klíče. Soukromý klíč lze ale na základě jeho matematické spojitosti s veřejným klíčem rekonstruovat rychleji než při rekonstrukci hrubou silou. Proto je nutné používat delší klíče než u symetrické kryptografie. Konkrétně pro algoritmus RSA je bezpečnost klíče o délce 1024 bitů ekvivalentní s bezpečností symetrického klíče o délce 80 bitů, 2048 bitový RSA klíč odpovídá symetrickému klíči o délce 112 bitů a pro zajištění bezpečnosti odpovídající symetrickému klíči o délce 256 bitů je třeba použít RSA klíč o délce 15360 bitů, což už je prakticky nepoužitelná délka [3]. Algoritmus ECC je v tomto směru lepší a pro zajištění bezpečnosti na úrovni 256 bitového symetrického klíče stačí pouze 521 bitový ECC klíč [21]. Tato čísla jsou bezvýznamná na kvantových počítačích, na kterých lze díky Shorově algoritmu provést na jinak běžných počítačích časově složité výpočty, které zabezpečují účinnost asymetrických

[15] anglicky *confidentiality*
[16] Advanced Encryption Standard
[17] International Data Encryption Algorithm
[18] Data Encryption Standard
[19] Triple Data Encryption Standard
[20] Rivest Shamir Adleman
[21] Elliptical Curve Cryptography

klíčů, velmi rychle. Konkrétně se jedná o rychlost rozkladu čísla na prvočísla. O tu se opírá diskrétní algoritmus i algoritmy založené na eliptických křivkách. Dnes nejrychlejší běžný algoritmus GNFS[22] pro rozklad čísla na prvočísla pracuje s exponenciální časovou složitostí, Shorův algoritmus na kvantovém počítači dokáže tu samou úlohu řešit s polynomiální časovou složitostí [11]. Bude tedy nutné vynalézt jiné metody, aby s rozvojem kvantových počítačů byla asymetrická kryptografie bezpečná.

Oba přístupy mají výhody i nevýhody. U symetrické kryptografie je zásadním problémem distribuce klíčů, tedy jak zajistit, aby tajný klíč znali pouze a jedině příjemce a odesílatel, kteří se obvykle neznají a nikdy se fyzicky nepotkali. U asymetrické kryptografie je zase problematická výpočetní náročnost šifrování, jinak řečeno, asymetrické šifry jsou pomalé. Proto se často používá kombinace symetrického a asymetrického šifrování, například u protokolu SSL[23] a jeho nástupce TLS[24] nebo protokolu SSH[25] [12]. Kombinace spočívá v tom, že výměna symetrického klíče je šifrována asymetrickou šifrou za využití veřejného klíče jednoho z respondentů. Následně je zbytek komunikace šifrován již takto vyměněným symetrickým klíčem. Nejznámějším algoritmem pro výměnu klíčů je algoritmus Diffie-Hellman.

4.4.3 Integrita dat

Integritou dat[26] ve smyslu zabezpečení komunikace je myšleno zajištění toho, že data po cestě od odesílatele k příjemci nikdo nezmění. Při komunikaci na internetu je toto řešeno vygenerováním charakteristiky zprávy[27] o fixní délce, připojením této charakteristiky ke zprávě a jejím odesláním. Příjemce nejdříve oddělí charakteristiku zprávy a získá tak původní zprávu, stejným algoritmem jako odesílatel vygeneruje její charakteristiku a porovná ji s obdrženou charakteristikou. Pokud jsou stejné, nikdo po cestě zprávu nezměnil. [17]

Aby výše uvedený princip fungoval, je nutné pro generování charakteristiky použít kryptografickou hašovací funkci. Tato funkce zajišťuje, že je velmi náročné vygenerovat zprávu, která by měla požadovanou charakteristiku, stejně tak náročné je najít dvě rozdílné zprávy se stejnou charakteristikou a na základě charakteristiky není možné zprávu zrekonstruovat. Dále taková funkce musí při generování charakteristiky zprávy zohlednit všechny její bity a i při změně jediného bitu musí být nově vygenerovaná charakteristika od té původní natolik odlišná, aby se toho nedalo zneužít. Nejpoužívanější bezpečné kryptografické funkce jsou například MD6[28], SHA-3[29] či BLAKE2. [30]

I po splnění všech zmíněných předpokladů není stále zajištěno, že zprávu po cestě nikdo nezmění, a to z jednoho prostého důvodu, útočník může změnit zprávu a současně i její charakteristiku. Aby k tomu nedošlo, je nutné použít šifrování, není však nutné šifrovat celou zprávu, stačí zašifrovat charakteristiku.

[22] General number field sieve
[23] Secure Sockets Layer
[24] Transport Layer Security
[25] Secure Shell
[26] anglicky *integrity*
[27] anglicky *message digest*
[28] Message-Digest Algorithm 6
[29] Secure Hash Algorithm 3

4.4.4 Neodmítnutelnost

Neodmítnutelnost[30] zajišťuje, že uživatel nemůže popřít provedení dané akce a poskytuje mu ujištění, že jeho zpráva s požadavkem na provedení této akce byla doručena. K zajištění neodmítnutelnosti se obvykle používá digitální podpis. Nejznámějším algoritmem, který implementuje digitální podpis je DSA[31].

Digitální podpis je založen na asymetrické kryptografii, ale na rozdíl od zajištění důvěrnosti se zde pro šifrování používá privátní klíč. Privátním klíčem se nešifruje zpráva, pouze její charakteristika, která se získá stejným způsobem jako u zajištění integrity dat a má i stejnou funkci, a to zajistit, že zprávu nikdo nezměnil. Pro ověření odesílatele si příjemce obstará od odesílatele veřejný klíč, dešifruje zašifrovanou charakteristiku, vygeneruje charakteristiku příchozí zprávy a porovná je. Pokud jsou stejné, nedošlo po cestě ke změně zprávy, ale především odesílatel je opravdu ten, který zprávu podepsal, protože jen on vlastní privátní klíč, kterým byla zašifrována charakteristika. [24]

Samozřejmě to není tak jednoduché, vygenerovat pár asymetrických klíčů si může kdokoliv a poté se podvodně vydávat za někoho, kým není. Aby k tomu nedocházelo, musí mít odesílatel k veřejnému klíči certifikát, který ověřuje jeho identitu. Certifikáty vydávají certifikační autority, které při tomto procesu ověří žadatelovu identitu a podepíší jeho veřejný klíč svým soukromým. Opět by se dalo namítat, že potenciální útočník si může vytvořit i vlastní certifikační autoritu a certifikovat si falešné digitální podpisy. Tomu se dá předejít jen tak, že příjemce si ověří nejen identitu vlastníka veřejného klíče, ale i pravost certifikační autority. Ta se zajišťuje tak zvaným řetězcem důvěry, který funguje na principu, že nadřazené certifikační autority podepisují klíče svým podřazeným certifikačním autoritám. Tento řetězec končí u kořenové certifikační autority, která již nemůže být dále ověřena, ale vzhledem k tomu, že je pouze jedna, nedá se o její pravosti pochybovat. [17]

4.4.5 Ochrana proti přehrání

Přehrání[32] je druh útoku, u kterého útočník zachytí jinak validní zabezpečenou komunikaci a snaží se ji použít opakovaně nebo ji pouze pozdržet. Jako ochrana proti tomuto typu útoku se nejčastěji používají časová razítka. Podle časového razítka se určí stáří zprávy a zprávy starší než stanovená hodnota jsou ignorovány. [32]

K tomu, aby časová razítka fungovala, musí být zajištěno, že příjemce i odesílatel mají synchronizovaný čas a že maximální stáří zprávy zhruba odpovídá době jejího doručení. U synchronizace času mohou být komplikací rozdílná časové pásma, ve kterých se respondenti nacházejí. Proto se obvykle pro časová razítka používá koordinovaný světový čas[33] [10]. Další problém synchronizace času může být v tom, že každý respondent používá jiný zdroj hodin, který je nepřesný a k tomu nepřesný s opačnou fází, nebo že útočník zaútočí přímo při synchronizaci času a podvrhne nesprávný čas. U stanovení maximálního stáří zprávy je problematická proměnlivá doba jejího doručení, která je způsobena především změnami v zatížení internetové sítě. Z toho je patrné, že pokud bude útočník dostatečně rychlý, může se mu povést i přes použití časového razítka zprávu replikovat. Proto může být časové razítko ještě doplněno například o sekvenční číslo zprávy nebo o nějakou jednorázovou informaci [32]. Jak sekvenční číslo tak jednorázová informace vyžadují uchování stavu, což může být problematické, protože komunikace na internetu bývá často nestavová.

[30] anglicky *nonrepudiation*
[31] Digital Signature Algorithm
[32] anglicky *replay attack*
[33] anglicky *Coordinated Universal Time* (UTC)

Kapitola 5

Návrh implementace příslušných služeb pro zvolené BPMN bloky

Aby bylo možné demonstrovat implementaci služeb jednotlivých BPMN bloků a ostatních BPMN elementů, bude implementován celý informační systém soustředěný na tuto tématiku, tedy *Business Process Management System* (dále jen BPMS). Tato kapitola hovoří o návrhu tohoto informačního systému, který bude umožňovat řešení podnikových procesů modelovaných pomocí standardu BPMN, a to i při spolupráci více instancí tohoto systému provozovaného různými organizacemi. Bude se jednat o distribuovaný informační systém (dále jen systém).

V první části kapitola obsahuje výběr BPMN bloků a jiných elementů, které budou implementovány v rámci této práce. Dále je v kapitole popsán návrh systému z pohledu uživatele. Následuje popis způsobu výměny zpráv a synchronizace paralelních procesů. Poté je vytvořen návrh rozložení prvků grafického uživatelského rozhraní a nakonec je vyhotoven datový model, se kterým systém pracuje.

5.1 Výběr BPMN bloků a jiných elementů pro implementaci

Mezi hlavní cíle této práce patří integrace BPMS s okolními systémy voláním webových služeb, výměna zpráv mezi paralelně běžícími BPMN procesy a jejich synchronizace. Pro dosažení prvního cíle je k dispozici jediný BPMN blok, a to Service Task. Výměna zpráv bude zajištěna čtveřicí událostí[1] Throwing Message Event, Catching Message Event, Throwing Signal Event a Catching Signal Event. Úkoly Send Task a Recieve Task nejsou pro účel automatizované výměny zpráv vhodné, jelikož by při jejich vykonávání mělo dojít k zásahu uživatele. Synchronizace je prováděna při změnách aktivit jednotlivých bloků běžícího BPMN modelu specifikovaného diagramem kolaborace a především zasíláním zpráv, které případně ani nemusí nést žádná data. Synchronizaci aktivních bloků bude systém zajišťovat automaticky bez specifikace této skutečnosti v BPMN modelu.

Aby bylo možné zajistit aspoň základní funkčnost celého BPMN modelu a aby bylo možné zmíněné bloky použít, je nutné implementovat i další BPMN bloky a elementy. Pro sémanticky korektní zasílání zpráv pomocí bloků Message Event je nutné, aby model obsahoval více elementů Pool, každý pro jinou organizaci, mezi kterými budou tyto zprávy přenášeny. Dále je pro plnou automatizaci volání webových služeb a výměnu zpráv nutné v průběhu vykonávání běžící instance BPMN modelu (dále jen workflow) získávat data,

[1] V aktuální verzi implementace je lze specifikovat pouze jako Intermediate Event.

která budou pro tyto účely použita. To bude zajištěno implementací bloku User Task. Aby bylo možné rozlišit řešitelské role jednotlivých bloků User Task, bude navíc umožněno dělit elementy Pool na jednotlivé dráhy – elementy Lane. Nezbytné jsou samozřejmě také bloky Start Event a End Event pro vyjádření začátku a konce podnikového procesu. K propojení zmíněných bloků bude možné použít element Sequence Flow a pro vyznačení odesílatele a příjemce zpráv element Message Flow.

Přestože je výběr implementovaných BPMN bloků a jiných elementů poměrně úzký, návrh datového modelu a implementace budou provedeny tak, aby bylo možné v budoucnu tento výběr rozšířit. Vybrané elementy budou mít funkcionalitu popsanou výčtem níže. Funkcionalita je zvolena tak, aby bylo splněno zadání této práce a aby odpovídala standardu BPMN, i když v některých případech ne v jeho plném rozsahu.

- **Pool** identifikuje organizaci, která používá instanci tohoto systému. Definuje tak spolupráci více organizací na jednom BPMN modelu, mezi nimiž dochází k zasílání zpráv a synchronizaci pomocí webových služeb. Přiřazení systémů organizací k elementům Pool je popsáno v sekci 5.2.4.
- **Lane** umožňuje dále členit modelovaný BPMN proces v rámci jedné organizace. Každá dráha identifikuje roli BPMN bloků, které se ve dráze nacházejí. Návaznost namodelovaných rolí na řešitelské role v systému je popsána v sekci 5.2.4.
- **Start Event** určuje bod začátku modelovaného podnikového procesu.
- **End Event** určuje bod konce modelovaného podnikového procesu.
- **Throwing Message Event** umožňuje zaslat získaná data při interpretaci BPMN modelu specifikovanému bloku Catching Message Event v rámci tohoto modelu. Zaslaná zpráva ale také nemusí nést žádná data, pak slouží pouze k synchronizaci. Data, která lze odeslat, jsou zmíněna v sekci 5.2.4. Odesílání dat je uvedeno v sekci 5.3.
- **Catching Message Event** přijímá data zaslaná blokem Throwing Message Event, který je s tímto blokem spojen pomocí elementu Message Flow.
- **Throwing Signal Event** odesílá získaná data v rámci interpretace všem blokům Catching Signal Event, které jsou o tato data přihlášeny. Tyto bloky nemusí být ve stejném BPMN modelu. Opět nemusí odeslaná zpráva obsahovat žádná data, poté slouží pouze k synchronizaci. Výběr odesílaných dat je popsán v sekci 5.2.4. Odesílání dat je uvedeno v sekci 5.3.
- **Catching Signal Event** přijímá data od plně kvalifikovaného odesílatele – bloku Throwing Signal Event. Odesílatel není specifikován v BPMN modelu pomocí elementu Message Flow, ale až po jeho nahrání do systému. Postup výběru odesílatele je popsán v sekci 5.2.4.
- **User Task** umožňuje prostřednictvím formulářů vyplňovat data, která budou dále využita pro dotazování se webových služeb a zasílání zpráv. Způsob, jakým budou formuláře tvořeny, je popsán v sekci 5.2.4, implementace formulářů je obsažena v sekci 6.4.
- **Service Task** reprezentuje webovou službu, jejíž volání bude plně automatizováno při interpretaci BPMN modelu. Volání webové služby znamená dotázání se aplikačního rozhraní jiného systému nebo aplikace. Definice webových služeb, které půjde

prostřednictvím bloku volat, je popsána v sekci 5.2.7, výběr webové služby pro konkrétní blok Service Task je objasněn v sekci 5.2.4 a implementace dotazu na aplikační rozhraní je obsažena v sekci 6.3.

- **Sequence Flow** udává pořadí interpretace jednotlivých BPMN bloků modelovaného procesu. Každý blok musí mít právě jeden vstupní a právě jeden výstupní element Sequence Flow. Toto neplatí pro blok Start Even, který má pouze jeden výstupní element, a také pro blok End Event, který naopak má pouze jeden vstupní element.
- **Message Flow** identifikuje dvojici příjemce a odesílatel reprezentovanou bloky Throwing Message Event a Catching Message Event. Každý tento blok musí mít právě jeden element Message Flow, u Throwing Message Event jako výstupní a u Catching Message Event jako vstupní. Spojené bloky musí být namodelovány v jiných elementech Pool.

5.2 Návrh MVC komponent systému z pohledu uživatelských akcí a rolí

Informační systém bude členěn na několik oblastí, které implementují určitou logicky oddělenou část. Tyto oblasti budou dále nazývány moduly. Konkrétně se bude jednat o modul pro správu uživatelů, modul pro správu systémů, modul pro správu agend, modul pro správu BPMN modelů, modul pro správu workflow, modul pro řešení úkolů a modul pro správu webových služeb. Každý modul bude implementovat potřebné řadiče a k nim příslušné modely a pohledy návrhového vzoru *model-pohled-řadič*[2] (dále jen MVC) [16].

Uživatelé se budou do systému přihlašovat pomocí uživatelských účtů. Systém definuje pět rolí pro autorizaci, které opravňují uživatele provádět operace v systému, viz popis jednotlivých modulů. Role jsou nazvány Správce workflow[3], Správce agend[4], Správce modelů[5], Správce webových služeb[6] a Administrátor[7], který má automaticky práva všech předchozích rolí. Každému uživatelskému účtu lze přiřadit více rolí, což způsobí nárůst (součet) jeho pravomocí, ale mohou existovat i účty bez role. Všem účtům bude po úspěšné autentizaci umožněno řešit úkoly a přistupovat k potřebným modulům pro jejich řešení nehledě na přiřazené role.

5.2.1 Modul pro správu uživatelů

Pro používání tohoto modulu bude oprávněn zejména uživatel v roli Administrátor, uživatelé bez této role budou oprávněni tento modul využít pouze ke změně svého hesla. Hlavní funkcí modulu pro správu uživatelů bude tvorba a editace uživatelských účtů.

Tvorba uživatelského účtu obnáší zadání titulu, jména a příjmení uživatele, jeho emailu, telefonního čísla a unikátního uživatelského jména, které slouží pro přihlašování do systému. Editace uživatele zahrnuje již pouze změnu titulu, jména, příjmení, emailu a telefonního čísla. Vytváření uživatele a jeho případné editace také zahrnuje přiřazení či odebrání rolí. Heslo k účtu si vytvoří sám uživatel při prvním přihlášení. Bude požadováno, aby heslo mělo minimálně deset znaků a obsahovalo aspoň jedno malé písmeno, jedno velké písmeno, číslici

[2] anglicky *Model-View-Controller*
[3] Mělo by se jednat o vedoucího týmu řešícího nějaké workflow.
[4] Role reprezentuje pracovníky vyššího management.
[5] Role reprezentuje analytiky, kteří mají na starost návrh podnikových procesů.
[6] Roli budou zaštiťovat zaměstnanci s adekvátním vzděláním, nejčastěji z oddělení IT.
[7] Obecně jakýkoliv uživatel s dostatečnými oprávněními v organizaci.

a aspoň jeden nealfanumerický znak. Tyto požadavky prakticky znemožní pokus o ovládnutí účtu hrubou silou.

5.2.2 Modul pro správu systémů

Použití modulu pro správu systémů bude umožněno výhradně uživateli s rolí Administrátor. Hlavní funkcí tohoto modulu bude připojení jiné instance systému tak, aby organizace, které systémy provozují, mohly společně sdílet BPMN modely a spolupracovat na provádění z nich vzniklých workflow.

Připojení systému pro spolupráci bude obnášet vyplnění URL, na kterou budou mířeny dotazy, názvu systému, výběr úrovně zabezpečení komunikace, vyplnění jeho popisu a napsání zprávy pro příjemce této žádosti o spojení. Následně bude zaslána žádost o propojení systémů na specifikovanou URL adresu. Žádost o propojení bude obsahovat URL žádajícího systému, datum žádosti, jméno, příjmení, email a telefonní číslo odesílatele žádosti a napsanou zprávu. Na základě těchto informací uživatel rozhodne o navázání spojení. V kladném případě vyplní název a popis systému. Propojení pak potvrdí svým heslem. Následně spolu mohou systémy již komunikovat a podílet se tak na společném řešení workflow. Editace systému umožňuje změnu jeho názvu a popisu, kontaktní URL a úrovně zabezpečení komunikace. Způsoby autentizace systémů a zabezpečení jejich komunikace jsou popsány v sekci 6.5.2.

5.2.3 Modul pro správu agend

Modul pro správu agend budou oprávněni využívat k úpravám uživatelé v roli Správce agendy a Administrátor. Ostatní uživatelé budou moci modul pouze prohlížet. Modul bude umožňovat vytvářet a editovat agendy. Agenda zastřešuje uživatele, spolupracující systémy, BPMN modely a z nich vzniklé workflow, na jejichž řešení se podílejí uživatelé dle řešitelských rolí v agendě a s ní spolupracující systémy. Správci agendy bude umožněno role vytvářet, ale většinou jejich vytvoření bude provedeno automaticky nahráním BPMN modelu, viz sekce 5.2.4. Význam řešitelských rolí je popsán v sekci 5.2.6. Nakonec bude modul umožňovat přiřazení systémů, se kterými budou uživatelé agendy spolupracovat.

Tvorba agendy bude umožněna pouze uživateli v roli Administrátor. Pro vytvoření agendy je nutné zadat její název, nepovinný popis a jejího správce, kterým bude jeden z uživatelů v roli Správce agendy. Administrátorovi bude také umožněno správce agendy později změnit. Úprava agendy obnáší změnu jejího názvu, vyplnění popisu a přiřazení řešitelských rolí v rámci agendy. Úpravy může provádět Administrátor nebo její správce.

5.2.4 Modul pro správu BPMN modelů

Modul pro správu BPMN modelů bude umožňovat operace od nahrání BPMN modelu přes jeho konfiguraci a sdílení po spuštění workflow. Nahrání modelu a spuštění workflow bude provádět správce agendy, pod kterou má být model nehrán, tedy uživatel s rolí Správce agendy. Ostatní úpravy budou provádět uživatelé v roli Správce modelu. Administrátor může provádět všechny níže popsané operace. Uživatelé, kteří nemají jednu ze jmenovaných rolí, mohou modul pouze prohlížet.

Nahrání modelu znamená vytvoření souboru typu BPMN[8] a k němu vytvoření souboru obsahujícího patřičnou grafickou reprezentaci v SVG[9], nahrání těchto souborů do systému a zadání názvu modelu. Systém provede syntaktickou a sémantickou analýzu nahraného BPMN modelu. V případě úspěchu převede nahrané soubory do odpovídajících záznamů v tabulkách databáze tak, aby bylo možné model spustit a vytvořit tak běžící instanci BPMN modelu – workflow, kterou systém dokáže interpretovat. Dále budou na řešitelské role v agendě, pod kterou je model nahrán, navázány jednotlivé elementy Lane z BPMN modelu dle jejich názvů. Pokud odpovídající role v agendě neexistuje, bude přidána. Význam řešitelských rolí je popsán v sekci 5.2.6.

V případě, že je nahraný model popsán diagramem procesu, pokračuje Správce modelů jeho konfigurací. To znamená definováním atributů jednotlivých elementů modelu tak, aby z nahraného analytického modelu vznikl model spustitelný. Pokud je nahraný model popsán diagramem kolaborace, musí nejdříve Správce modelů přiřadit elementům Pool systémy organizací, které na provádění modelu mají spolupracovat. Tyto systémy musí mít nejdříve umožněny přístup k agendě, do které byl model nahrán. Jakmile jsou všechny elementy Pool obsazeny, může Správce modelů sdílet model se specifikovanými systémy. Sdílením modelu dojde za použití webových služeb k přenosu nutných částí modelu do databází spolupracujících systémů. Správci modelů těchto systémů poté mohou pokračovat s konfigurací částí modelu, která jim připadá.

Konfigurace modelu obnáší případnou změnu rolí elementům Lane a definici bloků modelu. Tyto definice již mezi sebou spolupracující systémy nesdílí, až na konfigurace bloků Message Event a Signal Event, které jsou nutné pro výměnu zpráv. Toto sdílení bude systém provádět automaticky při jejich editaci. Definice bloku obnáší vyplnění jeho popisu, který je poté využit jako zadání úkolu vzniklého z daného bloku a případně dalších údajů, které se liší na základě jeho typu. Bloky, u kterých se definují další údaje, jsou tyto údaje popsány následujícím výčtem.

- **User Task** obsahuje údaje o jeho vstupních a výstupních atributech. Vstupní atributy mohou být atributy bloků User Task nebo Service Task, které definovanému bloku v BPMN modelu předchází. Při definici vstupních atributů bude uživatel pouze určovat, jestli se mají zobrazit, a řešitel úkolu je nebude moci upravovat. Takto lze zajistit, že řešitel úkolu může při jeho řešení reagovat na výsledky již vyřešených úkolů.

  Výstupní atributy představují buď atributy, které je nutné vyplnit pro volání webových služeb nadcházejících bloků Service Task, nebo atributy, které má řešitel úkolu vyplnit pro jeho úspěšné splnění. V případě atributů pro blok Service Task definuje Správce modelů, jestli mají být data zadána v rámci řešení definovaného úkolu. U atributů, které má uživatel v rámci řešení úkolu vyplnit, je specifikován název atributu, jeho popis, povinnost a typ. Atribut může být typu řetězec, číslo, text, booleovská hodnota, datum, výběr a soubor. V případě výběru a souboru je dále definována jejich specifikace, což u výběru představuje jeho hodnoty a u souboru typy souborů. Nakonec definice bloku User Task obnáší určení jeho obtížnosti, to znamená určení počtu dnů, které má uživatel na jeho vyřešení. Implementace výše popsané funkcionality je popsána v sekci 5.3.

- **Service Task** nese informaci o webové službě, která bude v rámci interpretace bloku volána. Volanou webovou službu vybere Správce modelů z předdefinovaných webových

[8]Soubory s koncovkou .bpmn typu XML, systém byl testován vůči souborům vygenerovaným pomocí nástroje bpmn.io dostupného z `https://bpmn.io`.

[9]Scalable Vector Graphics

služeb, viz sekce 5.2.7. Dále zde bude možné provést mapování výstupů předcházejících bloků Service Task na vstupy tohoto. Tímto způsobem lze zajistit zřetězené volání služeb bez zásahu uživatele. Implementace volání webových služeb je objasněna v sekci 6.3.

- **Throwing Signal Event a Throwing Message Event** obsahují údaje o odesílaných datech. U těchto typů bloků se označují výstupní atributy z předcházejících bloků User Task, jejichž data mají být odeslána. Implementace této funkcionality je popsána v sekci 6.4.
- **Catching Signal Event** nese údaje o odesílateli dat a jejich struktuře. Odesílatele (blok Throwing Signal Event) uživatel postupně vybere podle systému, agendy, BPMN modelu a elementu Pool, ve kterém se blok nachází. To znamená, že blok Throwing Signal Event nemusí být ve stejném modelu a ani ve stejném systému. Takto lze zajistit výměnu zpráv a synchronizaci mezi více workflow v rámci jednoho systému, ale i mezi workflow více systémů. Struktura příchozích dat se poté automaticky synchronizuje na základě výběru odesílaných atributů bloku Throwing Signal Event, stejně tak dochází k synchronizaci struktury odesílaných dat u bloků Message Event. Zajištění synchronizace je popsáno v sekci 5.3.

Jakmile Správce modelů dokončí definice bloků ve své části modelu, může model spustit. Spuštění obnáší vyplnění názvu workflow, nepovinného popisu, jeho správce, což je uživatel s rolí Správce workflow, který bude dohlížet na provádění části workflow náležící jeho organizaci. Správci agendy je umožněno kdykoliv tyto atributy změnit, správce workflow může měnit pouze název a popis. Pokud na workflow spolupracuje více systémů, musí spuštění odsouhlasit všechny organizace, což znamená zadat svého správce workflow pro správu jejich části, vyplnit název a případně popis. Jakmile si všechny zúčastněné organizace vyžádají spuštění workflow, dojde na základě modelu k vytvoření běžící instance workflow a spustí se jeho interpretace, což obvykle znamená vytvoření prvního úkolu a určení jeho řešitele.

5.2.5 Modul pro správu workflow

Pro provádění úprav v tomto modulu bude oprávněn Správce workflow. Aby mohl Správce workflow editovat konkrétní workflow, musí být také určen jako jeho správce. Administrátor má opět veškerá práva a ostatní uživatelé mohou modul pouze prohlížet.

Úpravy workflow zde neznamenají změnu jeho struktury nebo definovaných vstupů a výstupů jednotlivých bloků. Tyto změny se musí provádět na nahraném BPMN modelu před spuštěním workflow. Úpravy mají vliv pouze na zajištění správného průběhu workflow. Správci workflow bude umožněno měnit řešitele úkolu, datum jeho očekávaného vyřešení, prioritu a datum očekávaného vyřešení celého workflow. Dále mu bude umožněno měnit stav celého workflow, to znamená aktivní workflow pozastavit nebo úplně zrušit a pozastavené workflow obnovit nebo zrušit. Nakonec bude správci workflow umožněno obnovit řešení již vyřešeného úkolu, aby bylo možné opravit případná pochybení. V tomto případě bude přesunuta aktivita zpět na obnovený blok a po jeho vyřešení bude nutné opětovně vyřešit i všechny následující bloky, aby byla provedená oprava reflektována na průběh workflow.

5.2.6 Modul pro řešení úkolů

Modul pro řešení úkolů je přístupný uživatelům se všemi operacemi modulu, nehledě na jejich roli. Zobrazuje uživateli přehled jeho úkolů a umožňuje jejich řešení. Úkoly jsou dvo-

jího typu, běžný úkol a webová služba. Druhý typ úkolu je vytvořen pouze v případě, že selže automatizované volání webové služby, a nebude tak možné získat všechna očekávaná data.

Vyřešení úkolu obnáší vyplnění jeho povinných atributů, které byly specifikovány při definici bloků nahraného BPMN modelu. V případě webové služby bude uživateli umožněno měnit vstupní data pro volání služby a službu opětovně volat. Pokud volání v tomto případě uspěje, budou vyplněna výstupní data a uživatel bude moci úkol vyřešit. Pokud ani to neuspěje, bude mu umožněno výstupní data vyplnit ručně. V průběhu řešení může kdykoliv uživatel uložit provedené změny a řešení opustit. Z úkolů bude uživateli umožněno přejít na workflow úkolu a do jeho agendy. Dále mu bude zobrazeno datum, do kterého má úkol vyřešit a jeho priorita. Podle těchto parametrů budou také úkoly řazeny, nejdříve dle priority a poté dle data dokončení. Nakonec bude modul umožňovat uživateli prohlížet úkoly, které již vyřešil.

Úkoly jsou uživatelům přiřazovány na základě rolí, které plní v agendách. Ty uživatelům přiděluje její správce. Pokud má agenda pod určitou rolí více uživatelů, bude úkol přiřazen tomu, který má právě nejméně aktivních úkolů. Vytváření úkolů probíhá automaticky. Jakmile je vyřešen aktuální úkol, systém workflow interpretuje, automaticky odesílá zprávy a volá webové služby, dokud není na řadě opět blok typu User Task. Poté systém aktivuje tento úkol a pokud na workflow spolupracuje více systémů, synchronizuje s nimi jeho aktuální stav.

5.2.7 Modul pro správu webových služeb

Tento modul bude umožňovat definici volání webových služeb jiných systémů, teď se nejedná o spolupracující systémy, ale o jakýkoliv systém poskytující aplikační rozhraní s patřičnými vlastnostmi. K tomuto modulu bude mít přístup uživatel v roli Správce webových služeb nebo Administrátor. Jeho úkolem bude definovat vstupy a výstupy webové služby a další informace potřebné pro její volání. Takto definované služby pak budou dostupné Správcům modelů při definici bloků typu Service Task.

Systém bude umožňovat volat ostatní systémy, které implementují HTTP koncové body. Primární typ aplikačního rozhraní bude REST, který je dle informací získaných v kapitole 4 nejpoužívanější. Konfigurace volání webové služby bude nicméně poměrně flexibilní, aby bylo možné volat co nejvíce různých aplikačních rozhraní, i když nebudou přímo typu REST.

Při tvorbě nové webové služby[10] bude Správce webových služeb vyzván k vyplnění jejího jména, URL, na které se webová služba nachází, typu serializace vstupních parametrů služby, HTTP metodu dotazu, druhu autentizace a nepovinného popisu.

Serializace dat bude umožněna do čtyř formátů. Ty jsou JSON[11], XML[12], u kterého bude dále možné specifikovat, jestli mají být atributy objektů serializovány jako XML atributy, nebo jako XML značky. Dále mezi tyto formáty patří serializace dat do části URL označované jako dotaz[13]. Nakonec bude možné data přímo vepsat do URL, což bude umožněno ohraničením názvu vstupního parametru dvěma složenými závorkami. Takto specifikovaná místa URL budou při volání webové služby nahrazena vyplněnými daty těchto parametrů.

[10] Webová služba, kterou bude možné volat v rámci provádění workflow.
[11] MIME typ `appliaction/json`
[12] MIME typ `application/xml`
[13] anglicky *query*, MIME typ `application/x-www-form-urlencoded`

Jako HTTP metodu bude možné zvolit `GET`, `POST`, `PUT`, `PATCH` a `DELETE`. Pokud při výběru HTTP metody `GET` Správce webových služeb zvolí serializaci do JSON nebo XML, budou vstupní parametry, pro případy netradičních aplikačních rozhraní, serializovány dle požadavku a připojeny k URL.

Protože většina aplikačních rozhraní vyžaduje určitý způsob autentizace, zejména kvůli bezpečnosti a pro omezení počtu dotazů na rozhraní za jednotku času, lze pro dotazování webové služby konfigurovat autentizační HTTP mechanismus typu `Basic` nebo `Bearer` využívající HTTP hlavičku `Authorization`. Pro proprietární způsoby autentizace a jiné případy bude dále umožněno konfigurovat vlastní HTTP hlavičky.

Konfigurace parametrů, se kterými bude webová služba volána, bude probíhat podobně jako konfigurace výstupních atributů u bloku typu User Task. Rozdílné budou datové typy, které v tomto případě jsou řetězec, číslo, booleovská hodnota, objekt a pole. Objekt bude možné konfigurovat tak, aby opět obsahoval parametry všech datových typů a u pole se bude volit jeden ze jmenovaných typů[14]. Dalším rozdílem je to, že parametrům webových služeb bude možné konfigurovat druhý název – alias, pod kterým budou data serializována do dotazu. Tímto způsobem je možné zajistit, že uživatel bude například vyplňovat pole nazvané *hodnota* a v HTTP požadavku bude vyplněná hodnota serializována a odeslána pod jménem *value*. Posledním rozdílem je, že Správci webových služeb bude umožněno definovat statická data dotazu, což jsou hodnoty, které budou při každém dotazu stejné. Jedná se například o části dotazu, které konfigurují následnou odpověď.

Konfigurace výstupních parametrů služby bude probíhat poloautomaticky. Jakmile bude mít Správce webových služeb hotovou konfiguraci vstupních parametrů, může provést testovací volání webové služby. Při provádění testovacího volání bude vyzván k vyplnění nakonfigurovaných vstupních parametrů testovacími daty, která jinak budou zadávat řešitelé úkolů workflow při jejich řešení. Výsledek dotazu bude Správci webových služeb zobrazen a bude mu umožněno z něj vygenerovat výstupní parametry. U takto vygenerovaných parametrů bude následně možné změnit jejich název, uvést popis a určit, jestli jsou pro úspěšné volání služby povinné. Výstupní parametry bude možné i úplně odstranit, v tomto případě budou příchozí data vzniklá voláním webové služby, která by byla mapována na tyto parametry, zahazována.

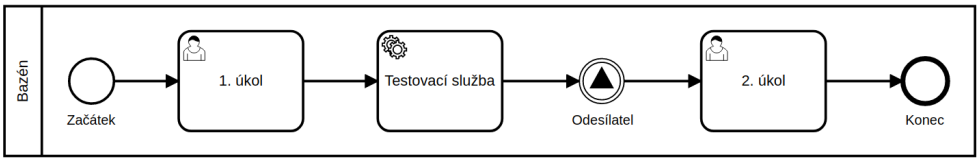
5.3 Způsob výměny zpráv a synchronizace paralelních procesů

V tomto případě je synchronizace paralelních procesů zajištěna právě výměnou zpráv. Zasláním zprávy dochází k předání získaných informací v rámci provádění workflow na základě konfigurace jeho BPMN modelu. To současně způsobí synchronizaci paralelních procesů, mezi kterými došlo k výměně této zprávy. Dále lze za synchronizaci považovat přeposílání metadat o workflow při změnách jeho stavu nebo metadat jeho modelu při změnách jeho konfigurace.

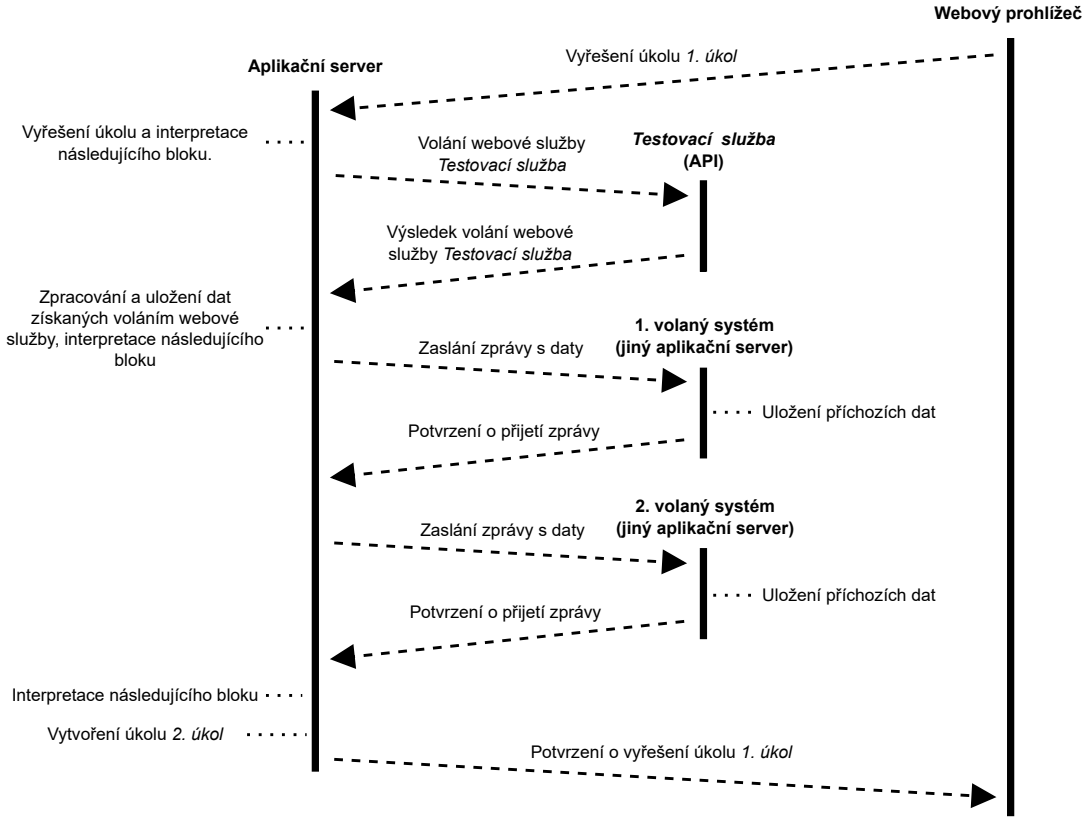
K výměně zpráv, respektive synchronizaci, dochází nejčastěji při spolupráci více organizací (systémů). Méně častá je pak komunikace v rámci jednoho systému. Jelikož požadavky na odeslání zpráv jsou generovány s malou frekvencí, lze očekávat, že komunikace bude probíhat nejméně s rozestupem několika sekund, realisticky bude tento rozestup v řádů hodin či dnů. Pro sjednocení je vhodné implementovat komunikaci pouze po síti i přesto, že při požadavcích na zasílání zpráv v rámci systému bude systém komunikovat sám se sebou.

[14] Aktuální implementace podporuje pouze základní datové typy (řetězec, číslo a booleovská hodnota).

Samotné systémy jsou ve svém jádře HTTP servery, viz sekce 6.1. Proto je nejvhodnější pro komunikaci použít HTTP protokol a do systémů přidat implementaci HTTP koncových bodů přijímající zprávy a samozřejmě také HTTP klienta, který bude zprávy generovat. To znamená, že systém, který odesílá zprávu nebo provádí synchronizaci, se chová v tomto okamžiku jako HTTP klient. Komunikace vždy probíhá po akci uživatele, ať už při konfiguraci nahraného BPMN modelu nebo při vyřešení úkolu a následné interpretaci workflow. Dochází tedy k tvorbě vnořených HTTP dotazů. Stejně tak dochází k tvorbě vnořených HTTP dotazů při volání webových služeb v rámci provádění workflow. Pro workflow vygenerované z velmi jednoduchého BPMN modelu na obrázku 5.1 bude komunikace po vyřešení bloku typu User Task (*1. úkol*) probíhat dle diagramu HTTP komunikace na obrázku 5.2. O zprávu bloku typu Throwing Signal Event (*Odesílatel*) mají v tomto případě pro demonstraci zájem dva bloky typu Catching Signal Event z jiných workflow.



Obrázek 5.1: Jednoduchý BPMN model



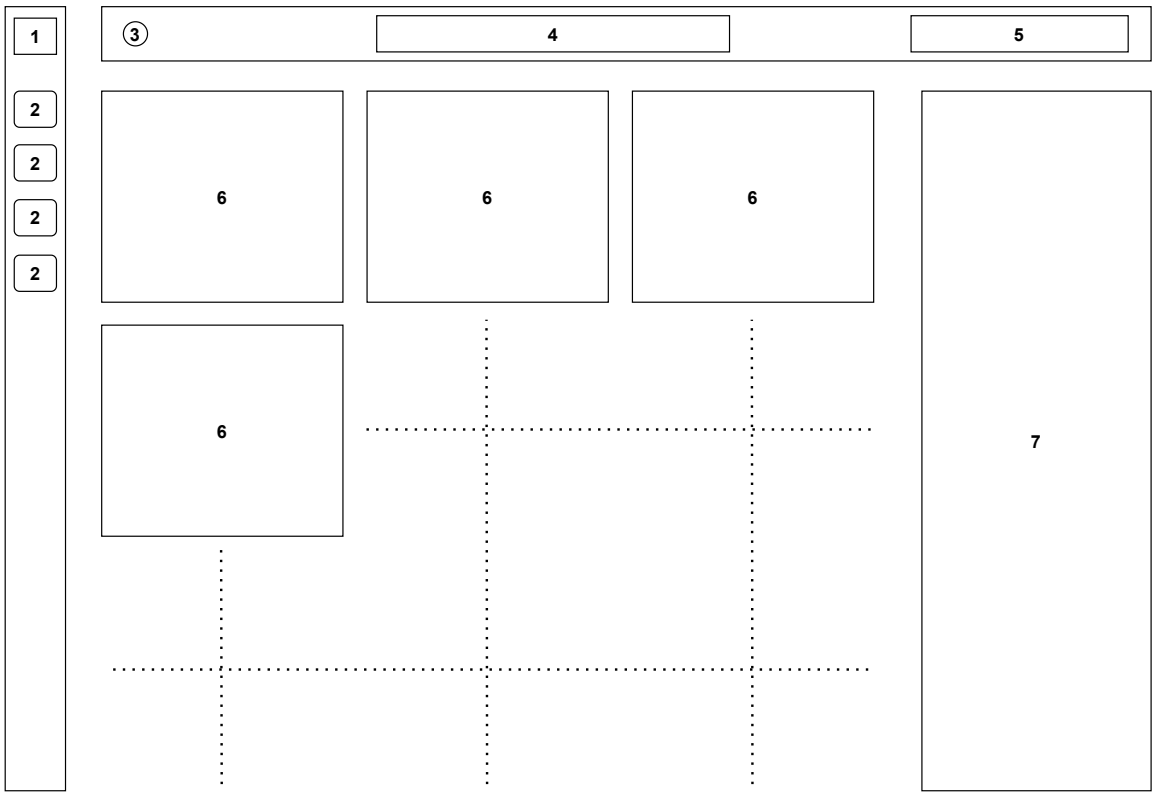
Obrázek 5.2: Diagram HTTP komunikace po vyřešení bloku *1. úkol*

Odesílaná data v dotazech jsou serializována do formátu JSON[15]. Jelikož odesílatel dotazu a jeho příjemce jsou systémy téže implementace, není nutné dále specifikovat schéma odesílaných dat, protože serializovaná data budou deserializována do objektu stejného datového typu, z jakého byla serializována.

5.4 Návrh rozložení prvků grafického uživatelského rozhraní

Aby bylo grafické uživatelské rozhraní pro uživatele snadno použitelné, je především nutné konzistentní umístění obsahu a ovládacích prvků. To bude zajištěno pro každý modul implementací zobrazení přehledu a detailní zobrazení dle koster na obrázcích níže. Tvorba a úpravy objektů budou primárně řešeny pomocí modálního okna[16], případně přímo transformací části grafického rozhraní v místě úpravy, pokud je to takto vhodnější. Definice bloků nahraných BPMN modelů bude probíhat také pomocí modálního okna, které bude zobrazováno po kliknutí na daný blok modelu.

5.4.1 Zobrazení přehledu



Obrázek 5.3: Kostra zobrazení přehledu

V následujícím výčtu jsou popsány očíslované prvky grafického uživatelského rozhraní zobrazení přehledu z obrázku 5.3.

1. Tlačítko pro zobrazení detailního menu s popisem ikon použitých pro navigaci.

[15] MIME typ `application/json`
[16] rozšiřující vyskakovací okno

2. Ikony, které slouží pro navigaci mezi moduly aplikace. V případě detailního menu jsou doplněné o název modulu.

3. Tlačítko pro zobrazení upozornění aktuálně přihlášeného uživatele.

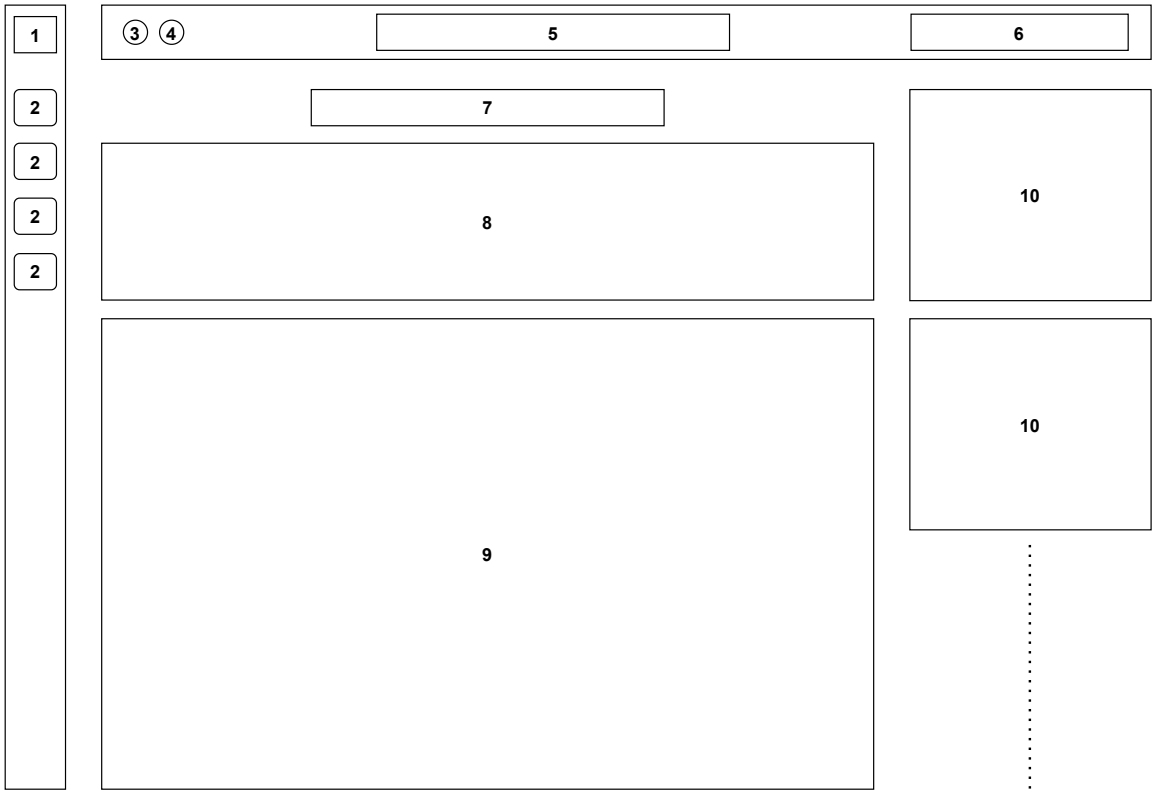
4. Název stránky.

5. Tlačítka pro tvorbu nového objektu a provádění jiných operací.

6. Dlaždice se základními informacemi o objektech modulu. Kliknutím na dlaždici dochází k přechodu na detailní zobrazení.

7. Výběr atributů, podle kterých lze filtrovat zobrazované objekty modulu.

5.4.2 Detailní zobrazení



Obrázek 5.4: Kostra detailního zobrazení

V následujícím výčtu jsou popsány očíslované prvky grafického uživatelského rozhraní detailního zobrazení z obrázku 5.4.

1. Tlačítko pro zobrazení detailního menu s popisem ikon použitých pro navigaci.

2. Ikony, které slouží pro navigaci mezi moduly aplikace. V případě detailního menu jsou doplněné o název modulu.

3. Tlačítko umožňující návrat zpět k zobrazení přehledu.

4. Tlačítko pro zobrazení upozornění aktuálně přihlášeného uživatele.
5. Název stránky.
6. Tlačítka pro zahájení editace objektu a provádění jiných operací.
7. Název objektu.
8. Informace o objektu.
9. Jiný obsah specifický pro objekt daného modulu.
10. Dlaždice se základními informacemi o objektech modulu shodné se zobrazením přehledu. Kliknutím je možné přepínat mezi jejich detailními zobrazeními.

5.5 Datový model

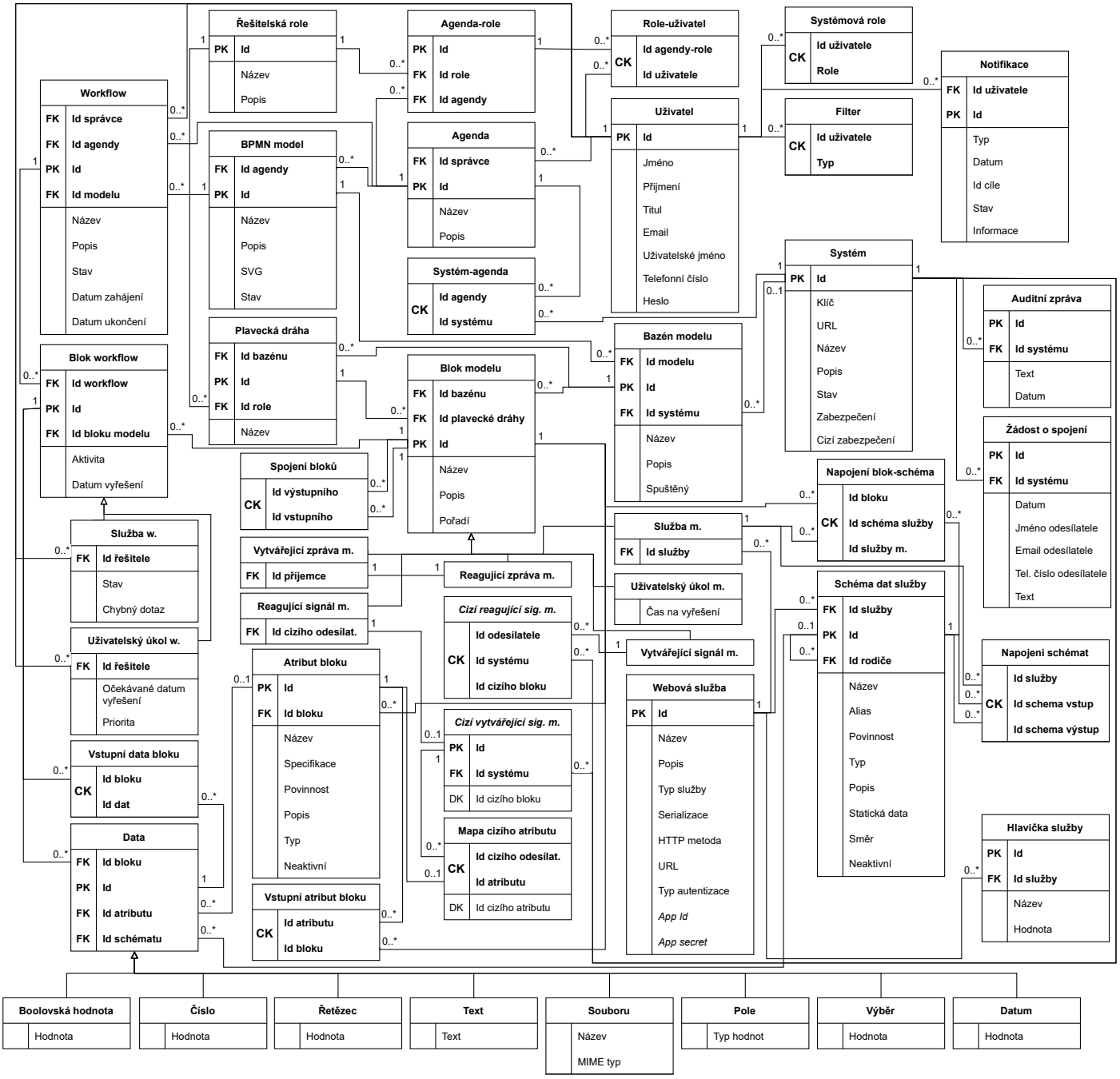
Pro ukládání dat vzniklých v rámci manipulace se systémem bude použita relační databáze. Správný návrh tabulek databáze a relací mezi nimi je pro úspěšnou implementaci zásadní. Nedostatky v návrhu lze jen těžko obejít ve vyšších vrstvách implementace systému. Změny datového modelu v průběhu implementace mohou být časově velmi náročné a mohou vyžadovat významné zásahy do již naimplementovaných úseků. Proto byl před začátkem vývoje vytvořen model relačního schématu databáze[17] zobrazený na obrázku 5.5. Navíc v případě tohoto systému je nutné při návrhu zohlednit skutečnost, že bude existovat více instancí tohoto systému, které budou spolu spolupracovat a sdílet data. Z toho důvodu bude ve všech tabulkách jako primární klíč použit *univerzální unikátní identifikátor*[18] (dále jen UUID) i v případech, kde by bylo možné použít jiný kandidátní klíč. Funkce univerzálního unikátního identifikátoru je obvykle taková, že jednoznačně identifikuje záznamy tabulek napříč různými databázemi. To bude také jeho funkce u záznamů tabulek, které mezi sebou systémy nesdílí. U záznamů, které mezi sebou systémy sdílí, bude naopak využita skutečnost, že pravděpodobnost vygenerování shodných UUID je prakticky nulová, k zajištění, že při ukládání sdílených dat nebude porušeno integritní omezení v podobě unikátního primárního klíče. Až na ojedinělé případy nebude díky tomu nutné vytvářet cizí klíče do jiných databází, protože stejné záznamy budou mít napříč databázemi stejný primární klíč.

Význam zkratek v modelech tabulek relačního schématu databáze je následující.

- **PK**[19] – primární klíč,
- **FK**[20] – cizí klíč,
- **CK**[21] – složený klíč z cizích klíčů do jiných tabulek, v dané tabulce je unikátní a plní roli primárního klíče,
- **DK**[22] – klíč do cizí databáze.

[17] Model a na základě něj vytvořené schéma databáze jsou ve 3. normální formě.
[18] anglicky *Universally Unique Identifier* (UUID) nebo *Globally Unique Identifier* (GUID)
[19] z anglického *primary key*
[20] z anglického *foreign key*
[21] z anglického *composite key*
[22] Z anglického *distant key*, tato zkratka a anglický termín se běžně nepoužívají.

U překrývajících se vazeb mezi tabulkami je nutné brát v potaz, že v modelu relačního schématu databáze nikdy není specifikovaná vazba M ku N[23], a podle toho je interpretovat. Vazby, které jsou napojeny na názvy odvozených tabulek, je nutné interpretovat tak, jako by tyto vazby byly napojeny na primární klíče těchto tabulek.



Obrázek 5.5: Model relačního schématu databáze

[23]vazba s `0..*` na začátku i na konci

Kapitola 6

Implementace MVC komponent informačního systému

Kapitola o implementaci obsahuje popis výběru použitých programovacích nástrojů, zvolenou architekturu a návrhový vzor. Dále popisuje implementaci systému ve vztahu k použité architektuře a návrhovému vzoru. Poté uvádí, jak je implementována automatizace volání webových služeb, výměna zpráv a synchronizace paralelních procesů ve vztahu k datovému modelu. Nakonec jsou zde uvedeny mechanismy zabezpečení proti případným útokům.

6.1 Výběr programovacích nástrojů

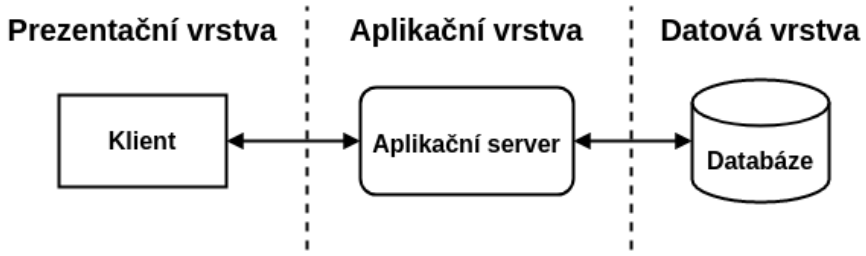
Jelikož dnes je již naprostá většina podnikových aplikací webových, je i tato aplikace navržena pro používaní ve webovém prohlížeči. Implementovat webový server není předmětem této práce, proto byly použity nástroje, které tuto implementaci již obsahují. Konkrétně byla zvolena otevřená platforma .NET. Tato platforma umožňuje použití mnoha jazyků a i jejich kombinaci. Pro tuto práci byl zvolen jazyk C#. Výhodou této platformy oproti použití skriptovacích jazyků, jako jsou například PHP[1] či *Ruby*, je statická typová kontrola a překlad do bajtkódu, což zajišťuje rychlejší běh aplikace.

6.2 Výběr architektury a návrhového vzoru

Jak už bylo zmíněno, k aplikaci se přistupuje prostřednictvím webového prohlížeče, který komunikuje s aplikačním serverem. Aplikace nezobrazuje statické stránky, ale prezentuje vytvářená data, u kterých je vyžadováno jejich perzistentní uložení. Z toho plyne nutnost použití databáze. Aplikaci tedy tvoří webový klient, aplikační server a databázový server, to znamená, že se jedná o třívrstvou architekturu.

Tento typ architektury rozděluje aplikaci do tří fyzicky oddělených vrstev[2], což umožňuje běh jednotlivých částí aplikace na různých zařízeních. Tyto vrstvy obvykle označujeme jako prezentační vrstvu[3], aplikační vrstvu[4] a datovou vrstvu[5]. Přenos dat mezi jednotlivými fyzickými vrstvami je zobrazen na obrázku 6.1.

[1] Hypertext Preprocessor
[2] anglicky *tier*
[3] anglicky *presentation tier*
[4] anglicky *application tier*
[5] anglicky *data tier*



Obrázek 6.1: Přenos dat u třívrstvé architektury

Jako použitý návrhový vzor byl, vzhledem k vestavěné podpoře v platformě .NET a jeho častému používání u aplikací s grafickým uživatelským rozhraním, vybrán vzor MVC.

6.2.1 Prezentační vrstva (fyzická)

Prezentační vrstva slouží prostřednictvím webového prohlížeče k interakci uživatele se systémem. Klient vhodně zobrazuje informace uživateli strukturované pomocí HTML[6]. HTML struktura je graficky stylována pomocí CSS[7] a ovládání jednotlivých prvků HTML stránek či validace vstupů je zajištěna pomocí skriptovacího jazyka *JavaScript*.

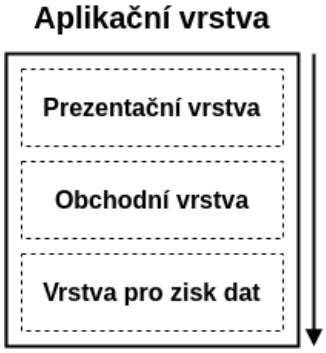
Pro zajištění toho, aby si systém pamatoval uživatelem zvolené výběry filtrů a zobrazoval podle nich objekty v přehledových stránkách aplikace, je použit mechanismus HTTP protokolu *Cookies*. Při změně výběru některého z filtrů je volán server, který provede filtrování, zašle webovému prohlížeči požadovaná data pro zobrazení a také požadavek na adekvátní změnu zapsaných *Cookies*. Současně server tuto změnu výběru filtrů zapíše do databáze, aby při pozdějším přihlášení nebo při přihlášení v jiném prohlížeči mohl server opět zažádat prohlížeč o zapamatování odpovídajících *Cookies*. Poté již v každém následujícím dotazu klienta server používá pro filtrování informace z *Cookies*, aby nemusel zbytečně provádět dotazy na databázi.

Další zajímavostí prezentační vrstvy je použití mechanismu *WebSockets* pro zajištění oboustranné komunikace mezi serverem a klientem, v tomto případě webovým prohlížečem. Tento mechanismus umožňuje serveru zasílat data klientovi i bez jeho předchozího dotazu. V systému jsou *WebSockets* implementovány pomocí knihovny *SignalR* a použity pro vytváření upozornění uživatelů na akce, které se v systému dějí, například, že má uživatel k řešení nový úkol.

6.2.2 Aplikační vrstva

Aplikační vrstvu tvoří webový server, který zpracovává HTTP požadavky. Implementace zpracování příchozích požadavků je dále členěna do tří logických vrstev[8], a to prezentační vrstvy[9], obchodní vrstvy[10] a vrstvy pro zisk dat[11]. Vzájemné postavení jednotlivých vrstev je zobrazeno na obrázku 6.2.

[6] Hypertext Markup Language
[7] Cascading Style Sheets
[8] anglicky *layer*
[9] anglicky *presentation layer* (PL); Na rozdíl od anglických názvů je český název stejný jako pro prezentační vrstvu ve smyslu architektury, nejedná se ale o totéž.
[10] anglicky *business layer* (BL)
[11] anglicky *data access layer* (DAL)



Obrázek 6.2: Vzájemné postavení logických vrstev

Třídy a soubory, ve kterých je obsažena implementace jednotlivých vrstev, jsou nazvány pohledy[12] pro prezentační vrstvu, fasády[13] pro obchodní vrstvu a repositáře[14] pro vrstvu pro zisk dat. Volání pohledů a funkcí fasád zajišťují řadiče[15], které jsou implementovány ve stejnojmenných třídách a souborech. Řadiče přijmou HTTP požadavek, deserializují obdržená data a předávají je pro daný HTTP koncový bod odpovědné funkci fasády. Fasáda data zpracuje a vrací obecně jiná data pro zobrazení. Ta jsou předána pohledu, který vygeneruje odpovídající HTML stránku, a řadič ji následně odesílá jako HTTP odpověď na původní dotaz.

Prezentační vrstva (logická)

Prezentační vrstva představuje pohled z návrhového vzoru MVC. Stará se o generování HTML obsahu, který je poté zaslán v odpovědi na dotaz klienta. Generování HTML stránek je zajištěno použitím jazyka *Razor*, který běží na straně serveru a umožňuje tvorbu dynamických HTML stránek. Jedná se o kombinaci HTML a v tomto případě jazyka C#.

Obchodní vrstva

Obchodní vrstva zasahuje v návrhovém vzoru MVC do řadiče a modelu. Tato vrstva obsahuje hlavní logiku systému. Interpretuje příchozí data, provádí potřebné výpočty a volá funkce vrstvy pro zisk dat. Data získaná z databáze nebo jen upravená výpočty případně transformuje do vhodného formátu pro zobrazení a sestavuje z nich objekt, který je prostřednictvím řadiče předán do pohledu – prezentační vrstvy.

Vrstva pro zisk dat

Vrstva pro zisk dat v návrhovém vzoru MVC představuje model. Obsahuje především různé předem připravené dotazy na databázový server, do kterých se při jejich volání doplní například už jen identifikátor objektu, nebo tyto dotazy sestavuje dle požadavků, například při filtrování.

[12] anglicky *views*
[13] anglicky *facades*
[14] anglicky *repositories*
[15] anglicky *controllers*

6.2.3 Datová vrstva

Datová vrstva je dnes z naprosté většiny tvořena databází typu SQL[16], mohou to být ale i jiné druhy databází například grafové nebo pouze souborový systém. Tento systém používá databázi *SQL Server Express* od společnosti *Microsoft*, která má na platformě .NET velmi dobrou podporu. Pomocí knihovny *Entity Framework Core* lze implementovat tak zvaný *Code First*[17] přístup. Tento přístup umožňuje definovat schéma databáze pomocí tříd v jazyce C# za využití objektově relačního mapování. Následně lze pomocí funkcí dostupných v této knihovně provádět přímo dotazy na databázi bez nutnosti použití SQL.

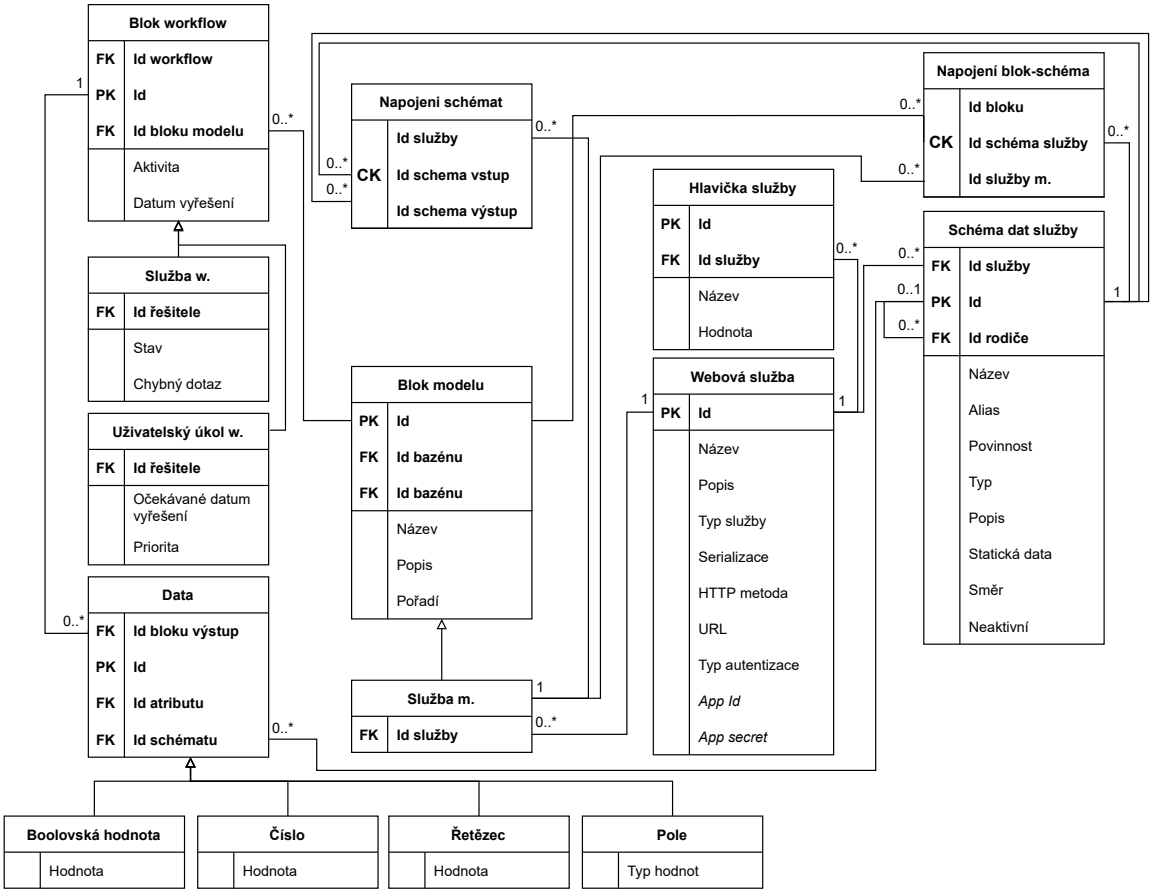
6.3 Automatizace volání webových služeb

Tato sekce objasňuje, jakým způsobem byla v informačním systému implementována definice webových služeb, jejich navázání na nahraný BPMN model při jeho konfiguraci a volání webových služeb v rámci interpretace workflow. Způsob implementace je popsán ve vztahu k výňatku pro tento účel dotčených tabulek modelu relačního schématu databáze na obrázku 6.3.

Definice webové služby zasahuje do tabulek Webová služba, Hlavička služby a Schéma dat služby. Na základě záznamu v tabulce Webová služba a na záznam navázaných řádků z tabulky Hlavička služby je vytvořena při každém volání webové služby hlavička protokolu HTTP. Pouze sloupce Název, Popis a Typ služby z tabulky Webová služba nejsou použity, ty slouží pro orientaci uživatelů. Sloupce App Id a App secret jsou použity pro autentizaci vůči volané webové službě. Význam a použití ostatních sloupců tabulek v hlavičce HTTP protokolu by měl být zřejmý z jejich názvů.

Zasílaná data v těle HTTP dotazu jsou serializována na základě záznamů z tabulky Schéma dat služby, tato tabulka ale slouží také k deserializaci příchozích dat z odpovědi na dotaz. Její záznamy slouží jako předloha – metadata pro dotazy a odpovědi. Rozlišení, jestli bude schéma použito pro serializaci, nebo deserializaci, určuje sloupec Směr. Sloupec Název určuje, pod jakým jménem budou uživatelé vyplňovat vstupní parametr webové služby vázané na dané schéma, respektive jak se bude jmenovat výstupní parametr, který jim po volání bude zobrazen. Sloupec Popis slouží pouze pro uživatele a obsahuje další rozšiřující informace o parametru. Sloupec Alias slouží k pojmenování parametru do serializovaného dotazu, respektive k namapování parametru při deserializovnání odpovědi. Pokud je nevyplněný, používá se pro tento účel sloupec Název. Samotná serializace závisí na sloupcích Id rodiče a Typ. Unární vazba zprostředkovaná sloupcem Id rodiče umožňuje schémata uspořádat do stromové struktury. Pojem stromová struktura je zde nutné brát s rezervou, protože z kardinality unární vazby je patrné, že tato stromová struktura může mít více uzlů na 1. úrovni, kde by se jinak nacházel kořen. To se sice neslučuje s definicí stromu jakožto datové struktury, ale pro účely serializace je to výhodné. Vnitřním uzlem této stromové struktury může být pouze datový typ objekt nebo pole, což specifikuje sloupec Typ. Zbylé možné datové typy, a to řetězec, číslo a boolovská hodnota, mohou být pouze na listové úrovni. Toto uspořádání umožňuje serializovat data strukturovaně, v tomto případě do formátů JSON a XML. U formátu XML je v případě více uzlů na 1. úrovni přidána kořenová značka. Při serializaci dat do formátu daného MIME typem `application/x-www-form-urlencoded`, který nepodporuje zanoření, jsou serializovány pouze uzly 1. úrovně datových typů řetězec, číslo a boolovská hodnota. Obdobně probíhá deserializace, zde dochází k mapování struktu-

[16] Structured Query Language
[17] doslovně česky *prvně kód*



Obrázek 6.3: Výňatek tabulek modelu relačního schématu databáze souvisejících s automatizací volání webových služeb

rovaných příchozích dat na jednotlivé úrovně stromové struktury datových schémat. Sloupec Povinnost udává, zda u vstupních parametrů musí být data pro odeslání dotazu vyplněna, a u výstupních parametrů určuje, zda musí být data voláním webové služby získána. Pokud jedno nebo druhé není při automatickém volání webové služby v rámci interpretace workflow splněno, je nutný zásah uživatele a dojde k vytvoření úkolu. Sloupec Statická data se používá pouze u vstupních parametrů, pokud je vyplněn, tak se při každém volání webové služby použije tato hodnota a uživatelům ji není v rámci interpretace workflow umožněno měnit. A nakonec sloupec Neaktivní, který určuje, jestli má být schéma použito pro serializaci, deserializaci a zobrazení uživatelům. K deaktivaci schématu dochází v okamžik, kdy se uživatel pokusí smazat schématem popsaný parametr služby, ale smazání není možné provést, protože na něm jsou již závislá data z předchozích volání této služby.

Při konfiguraci nahraného BPMN modelu zajišťuje navázání webové služby na blok modelu Service Task vazba mezi tabulkou Služba m. a tabulkou Webová služba. Pokud není navázání provedeno, je při interpretaci workflow blok přeskočen. Zadávání vstupních a zobrazování výstupních parametrů webových služeb se konfiguruje přidáním záznamu do tabulky Napojení blok-schéma. Vstupní parametry lze zadat pouze u bloků User Task, které předcházejí bloku Service Task, a naopak výstupní parametry lze zobrazit jen v blocích User Task následujících za blokem Service Task. Zřetězení služeb, jinak řečeno použití výstupních

parametrů jedné služby jako vstupní parametry druhé, je zajištěno přidáním záznamu do tabulky Napojeni schémat.

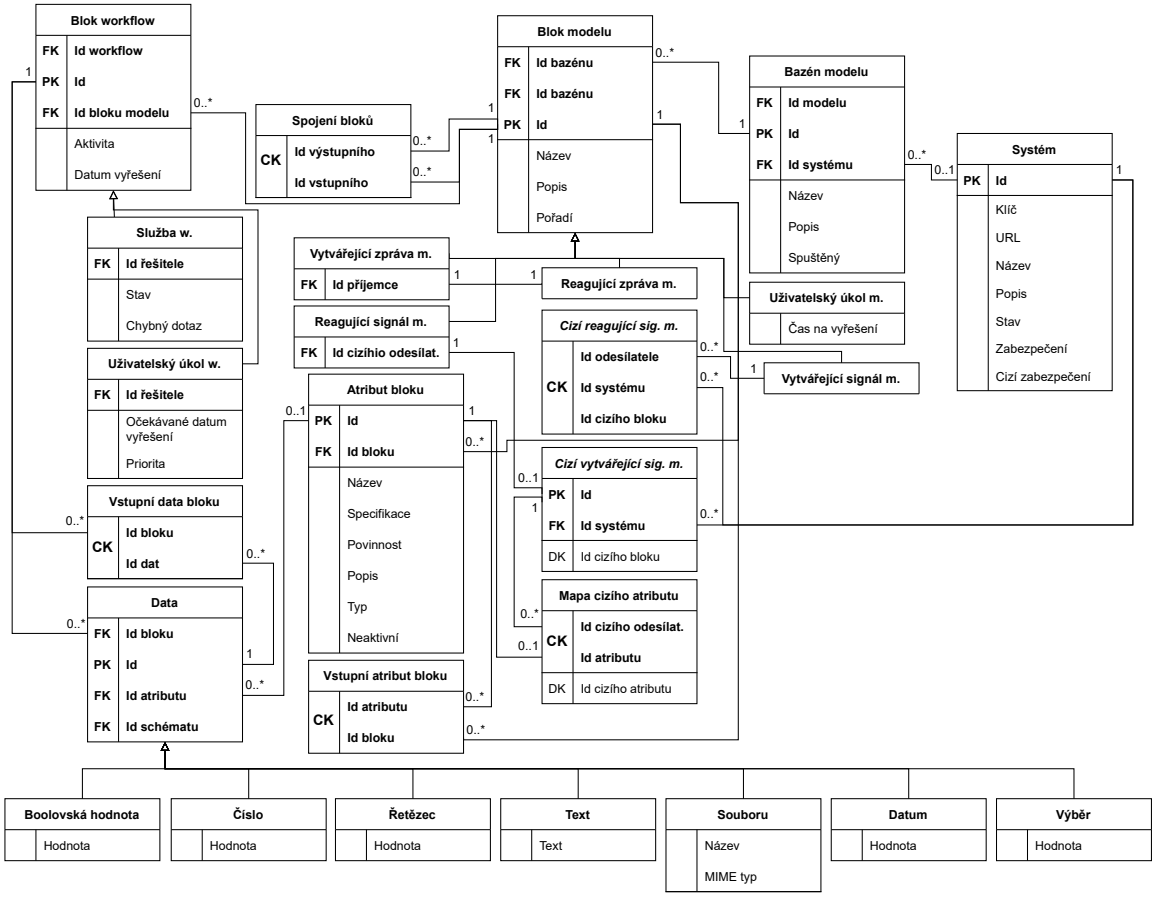
Samotná data, která budou odeslána při dotazu na webovou službu, respektive získána z odpovědi na toto volání, jsou ukládána do tabulky Data a tabulek z ní odvozených. Záznamy v těchto tabulkách jsou vytvořeny při spuštění workflow na základě záznamů v tabulce Schéma dat služby. Sloupec Typ této tabulky určuje, do které z odvozených tabulek bude záznam zapsán, a tedy jaký datový typ bude mít sloupec Hodnota. U pole je nutné znát datový typ jeho prvků, který je uložen ve Sloupci Typ hodnot. U objektu není nutné ukládat žádnou další informaci, proto pro něj nemusí být vytvořena odvozená tabulka. Hodnoty takto vytvořených záznamů jsou naplňovány při řešení úkolů uživateli a poté použity při serializaci nebo jsou naplněny po volání webové služby při deserializaci odpovědi. Při spuštění workflow dochází také ke generování záznamů do tabulky Blok workflow a tabulek z ní odvozených.

6.4 Výměna zpráv a synchronizace paralelních procesů

Tato sekce popisuje, jakým způsobem bylo v informačním systému implementováno zasílání zpráv, synchronizace paralelních procesů a další k tomu potřebné oblasti. Způsob implementace je popsán ve vztahu k výňatku pro tento účel dotčených tabulek modelu relačního schématu databáze na obrázku 6.4.

Aby bylo možné odeslat nějaká data, je nejdříve nutné je získat. Data zadávají uživatelé při řešení úkolů. Jaká data má uživatel v rámci řešení úkolu zadat, je definováno při konfiguraci nahraného BPMN modelu, viz sekce 5.2.4. Konkrétně uživatel při konfiguraci vytváří záznamy v tabulce Atribut bloku, které fungují na podobném principu jako záznamy tabulky Schéma dat služby z minulé sekce 6.3. Sloupce Název, Povinnost, Popis a Neaktivní mají tudíž stejný význam jako v tabulce Schéma dat služby. Sloupec Typ v tomto případě umožňuje vybrat mezi typy vstupů, které budou uživateli generovány pro vyplnění při řešení úkolu. Vyplněná data budou ukládána jako záznamy v tabulkách odvozených z tabulky Data. Pro tyto záznamy budou pro jejich vyplnění ve webovém prohlížeči generovány následující HTML elementy.

- `select` s možnostmi výběru *nevybráno*, *ano* a *ne* pro záznam v tabulce Boolovská hodnota.
- `input` typu `number` pro záznam v tabulce Číslo.
- `input` typu `text` pro záznam v tabulce Řetězec.
- `textarea` pro záznam v tabulce Text.
- `input` typu `file` pro záznam v tabulce Soubor. Samotná data souboru jsou ukládána na pevný disk serveru, na kterém systém běží. Pro pojmenování ukládaného souboru je použit primární klíč daného záznamu, aby nedošlo ke kolizím uživateli stejně pojmenovaných souborů.
- `input` typu `date` pro záznam v tabulce Datum.
- `select` s uživatelem specifikovanými možnostmi výběru pro záznam v tabulce Výběr.



Obrázek 6.4: Výňatek tabulek modelu relačního schématu databáze souvisejících se zasíláním zpráv a synchronizací

Záznamy ve zmíněných tabulkách jsou vytvářeny při spuštění workflow, kde záznamy v tabulce Atribut bloku slouží jako předloha – metadata. Současně dochází také k vytváření záznamů v tabulce Blok workflow a z ní odvozených tabulkách.

Výměnu zpráv zajišťují BPMN bloky Message Event a Signal Event. V případě bloků Message Event je výměna zpráv, respektive odesílání zpráv, jednodušší. Vazba mezi těmito bloky se zjistí přímo z nahraného BPMN modelu a odpovídající záznamy se zapíší do tabulek Vytvářející zpráva m. a Reagující zpráva m. Adresa doručení zprávy se získá na základě vazby záznamu tabulky Vytvářející zpráva m. se záznamem tabulky Bazén modelu a záznamu tabulky Bazén modelu se záznamem tabulky Systém. V okamžiku, kdy dochází k odesílání zprávy, je již zaručeno, že záznam v tabulce Bazén modelu bude odkazovat na platný záznam v tabulce Systém. Informace o dvojicích BPMN bloků Throwing Signal Event a Catching Signal Event jsou ukládány do tabulek Vytvářející signál m. a Reagující signál m. V případě těchto bloků je určení příjemce respektive odesílatele složitější, protože blok Throwing Signal Event může být nakonfigurován tak, aby odesílal zprávu bloku Catching Signal Event, který nemusí být ve stejné databázi. Tuto skutečnost zohledňují tabulky Cizí vytvářející sig. m. a Cizí reagující sig. m., které se pro sjednocení implementace používají i v případě, že se oba bloky nacházejí ve stejné databázi. Adresa příjemce, respektive odesílatele, se u těchto tabulek zjistí na základě vazby s tabulkou Systém. Samotné odeslání dat je popsáno v sekci 5.3.

Pro výběr atributů, jejichž data budou při interpretaci odesílána, slouží vazební tabulka Vstupní atribut bloku. Záznamy se do této tabulky zapisují při konfiguraci bloků Throwing Message Event a Throwing Signal Event, kde uživatel vybírá odesílané atributy bloků User Task, které konfigurovanému bloku předcházejí. Záznamy lze do této tabulky obdobně zapsat i při konfiguraci bloku User Task. V tomto případě budou data odkazovaných atributů zobrazena uživatelům při řešení úkolu specifikovaného tímto blokem. Při každé změně konfigurace, která zasahuje do odesílaných atributů bloků Throwing Message Event a Throwing Signal Event, dochází ke sdílení těchto změn se systémem, v jehož databázi je uložen odpovídající blok Catching Message Event či Catching Signal Event. Uživatelé tohoto systému tak mají aktuální informaci o tom, co jim bude zasláno. V případě bloku Throwing Message Event může být díky kardinalitě vazby odpovídající tabulky s tabulkou pro blok Catching Message Event sdílen přímo odesílaný atribut včetně identifikátoru. V databázi systému s blokem Catching Message Event je sdílený atribut na tento blok pouze navázán. U dvojice bloků Throwing Signal Event a Catching Signal Event to tímto způsobem udělat nelze, protože blok Throwing Signal Event může mít více příjemců. Proto je nutné si uchovávat identifikátory atributů z jiných databází, aby bylo možné sdílet jejich změny a samozřejmě také zasílaná data. Tyto identifikátory jsou ukládány v tabulce Mapa cizího atributu.

6.5 Bezpečnost

Tato sekce popisuje, jaké jsou použity bezpečnostní mechanismy pro autentizaci a autorizaci uživatelů, ukládání citlivých dat a komunikaci mezi systémy.

6.5.1 Autentizace a autorizace uživatelů

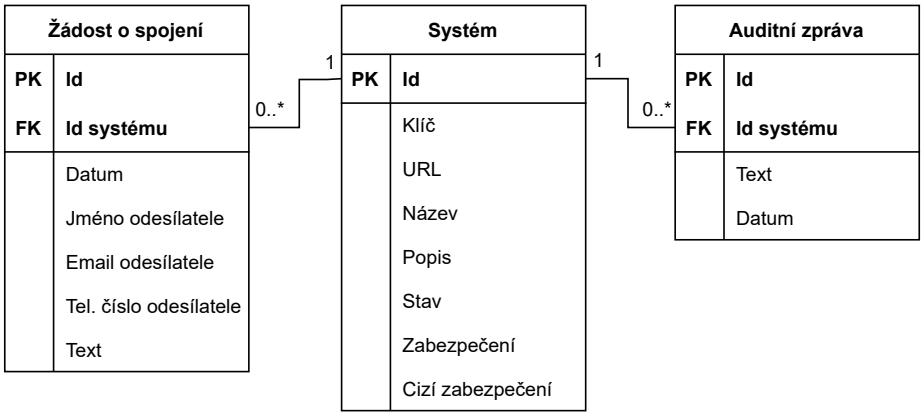
Pro autentizaci a autorizaci byl použit v .NET pro tento účel vestavěný mechanismus *Cookie Authentication*. Tento mechanismus umožňuje správu rolí a řízení přístupu buď k celému modulu systému, nebo pouze k některým jeho akcím formou anotací nad třídou řadiče pro daný modul nebo anotací nad jednotlivými jeho metodami. Uživatel se přihlašuje prostřednictvím uživatelského jména a hesla. Po 30 minutách neaktivity je uživatel automaticky odhlášen a musí se znovu přihlásit. Správa uživatelských účtů je popsána v sekci 5.2.1.

6.5.2 Zabezpečení komunikace mezi systémy

Zabezpečení komunikace mezi systémy zahrnuje autentizaci systémů i zabezpečení samotných přenášených dat. U zabezpečení přenášených dat si uživatelé mohou vybrat, jestli je chtějí zabezpečit a jaká má být úroveň tohoto zabezpečení. Zabezpečení dat dává smysl především, když systém používá pouze protokol HTTP a nikoliv jeho šifrovanou verzi HTTPS. Některé organizace ale mohou chtít například z důvodu monitorování své lokální sítě používat pouze protokol HTTP a uživatelům povolovat přístup pouze v rámci této sítě nebo pomocí VPN[18] a stále komunikovat s ostatními organizacemi pomocí nějakého prostupu v infrastruktuře sítě. Pro tento případ je nutné přenášená data zabezpečit. Způsob implementace autentizace a zabezpečení dat je popsán ve vztahu k výňatku pro tento účel dotčených tabulek modelu relačního schématu databáze na obrázku 6.5.

Pro autentizaci komunikace mezi systémy je použita HTTP autentizace typu `Bearer`. Autentizační žeton obsažený v HTTP hlavičce `Authorization` generuje systém, který zasílá dotaz. Získá údaje o dotazovaném systému z databáze, z hodnot sloupce Zabezpečení, který

[18] virtual private network



Obrázek 6.5: Výňatek tabulek modelu relačního schématu databáze souvisejících se zabezpečením komunikace mezi systémy

udává úroveň zabezpečení nakonfigurovanou uživateli systému generující dotaz, a sloupce Cizí zabezpečení, který udává úroveň zabezpečení nakonfigurovanou uživateli dotazovaného systému, vybere tu, která vyžaduje vyšší zabezpečení. Pokud uživatelé jednoho ze systémů změní úroveň zabezpečení, je tato skutečnost automaticky sdílena. A na základě získané úrovně zabezpečení je postup pro její dosažení následující.

- **Autentizace**: Dotazující se systém generuje pro autentizaci objekt, který obsahuje tři proměnné, a to identifikátor nově vytvářené auditní zprávy, charakteristiku zprávy a klíč, kterým je šifrováno tělo HTTP zprávy. Kromě identifikátoru nově vytvářené auditní zprávy jsou zbylé proměnné v tomto případě nevyplněny a identifikátor slouží pouze ke zvýšení entropie výsledného autentizačního žetonu, jinak také není použit.
- **Autentizace a auditní zprávy**: Při použití autentizace a auditních zpráv generuje systém objekt pro autentizaci stejným způsobem jako při pouhé autentizaci. Ke změně dochází až na straně příjemce, kde dochází k uplatnění identifikátoru nově vytvářené auditní zprávy.
- **Autentizace, auditní zprávy a integrita dat**: Při použití autentizace, auditních zpráv a integrity dat je navíc od předchozích úrovní zabezpečení vyplněna i proměnná obsahující charakteristiku zprávy. Tato proměnná je naplněna výsledkem kryptografické hašovací funkce BLAKE2 vypočteného z těla HTTP dotazu.
- **Autentizace, auditní zprávy, integrita dat a důvěrnost**: V případě nejvyšší úrovně zabezpečení je navíc od předchozí úrovně vyplněna i proměnná obsahující klíč pro dešifrování těla zprávy. Tento symetrický klíč o délce 256 bitů systém generuje nově pro každý dotaz, zašifruje jím tělo HTTP dotazu a převede takto získaná binární data na data textová pomocí kódování *Base64*. Pro šifrování je použit algoritmus AES.

Poté systém vytvořený objekt v jednom z předchozích bodů serializuje a zašifruje jej klíčem uloženým ve sloupci Klíč. Opět je pro šifrování použita symetrická šifra a algoritmus AES. K zašifrovaným datům připojí identifikátor dotazovaného systému, který je uložen ve sloupci Id. Nakonec tato binární data převede na data textová, stejně jako tělo zprávy v případě šifrování, pomocí kódování *Base64* a dotaz odešle.

Dotazovaný systém obdrží dotaz a převede autentizační žeton z HTTP hlavičky zpět na binární data. Data rozdělí na identifikátor systému a zašifrovaný serializovaný objekt pro autentizaci. Z databáze získá na základě příchozího identifikátoru informace o systému, který dotaz odeslal. Stejně jako systém odesílající dotaz určí úroveň požadovaného zabezpečení, získaným klíčem dešifruje serializovaný objekt pro autentizaci a deserializuje jej. Na základě získané úrovně zabezpečení poté systém pokračuje následovně.

- **Autentizace**: V případě, že postup od obdržení dotazu proběhl úspěšně, pokračuje systém se zpracováním dotazu.
- **Autentizace a auditní zprávy**: Při autentizaci a auditních zprávách navíc systém vytvoří na základě typu dotazu auditní záznam s identifikátorem z autentizačního objektu a pokusí se jej uložit do databáze. Pokud uložení selže, znamená to, že by byla narušena integrita dat. Narušení integrity dat by způsobil duplicitní primární klíč, což samozřejmě není možné, a autentizace selže. Tento způsob tvorby auditních zpráv částečně chrání proti útoku přehráním.
- **Autentizace, auditní zprávy a integrita dat**: Při autentizaci, auditních zprávách a integritě dat systém navíc vygeneruje charakteristiku zprávy z těla HTTP dotazu, stejně jako to provedl dotazující se systém, a porovná ji s charakteristikou z autentizačního objektu. Pokud se neshodují, autentizace selže.
- **Autentizace, auditní zprávy, integrita dat a důvěrnost**: V případě nejvyšší úrovně zabezpečení sytém dekóduje tělo HTTP dotazu z *Base64* textové reprezentace zpět na binární data a klíčem z autentizačního objektu data dešifruje. Poté postupuje stejně jako u předchozích úrovní autentizace a pokud dešifrování nebo jiná z úrovní autentizace selže, selže i celková autentizace.

Po zpracování dotazu na něj dotazovaný systém odesílá odpověď, která je zabezpečena stejným způsobem. Výše popsaný způsob zabezpečení na nejvyšší úrovni zajišťuje autentizaci, důvěrnost, integritu dat, neodmítnutelnost a částečně také ochranu proti útoku přehráním.

Jedna věc nebyla ještě objasněna, a to jakým způsobem dochází k distribuci klíčů mezi systémy. K tomu je použit záznam v tabulce Systém reprezentující danou instanci systému. Tento záznam je shodně zapsán do databáze při každém nasazení nové instance systému[19] na cílový server. Při napojení dvou systémů je vygenerován nový klíč, který je předán šifrovaně symetrickým klíčem uloženým v rámci tohoto záznamu. Nově vytvořený a předaný klíč následně slouží pro jejich vzájemnou autentizaci při další komunikaci. Takovýto způsob distribuce klíčů má oproti použití asymetrické kryptografie výhodu v tom, že zažádat o spojení systémů může jen ten, kdo zná tento symetrický klíč. To znamená, že žádat mohou pouze ty organizace, které využívají korektně nainstalovanou instanci tohoto systému. Přesto je ale navíc implementován mechanismus, který zajišťuje, že nedojde k napojení dvou systémů bez souhlasu uživatele. A to tak, že při napojení dvou systémů je u napojovaného vytvořen deaktivovaný záznam v tabulce Systém a záznam v tabulce Žádost o spojení. Deaktivovaný systém zatím nemůže provádět žádné dotazy, aktivita je určená sloupcem Stav. Uživateli v roli Administrátor je následně umožněno takto vytvořený systém aktivovat. K tomu účelu jsou mu zobrazeny informace posledního záznamu z tabulky Žádost o spojení. Jedná se o datum žádosti, jméno a příjmení žadatele o spojení, jeho telefonní

[19] Jedná se o tak zvaná *seed* data, která kromě zmíněného systému také obsahují uživatelský účet s rolí Administrátor, aby bylo možné vytvořit ostatní účty.

číslo a email a text s odůvodněním žádosti. Uživatel může na základě těchto informací kontaktovat žadatele a ověřit, že je žádost legitimní. Poté, co tak provede, může přijmout, nebo odmítnout spojení. Své rozhodnutí musí potvrdit zadáním svého hesla.

6.5.3 Zabezpečení uložených citlivých dat

Za citlivá data jsou považována hesla uživatelů, tajné klíče používané pro autentizaci při komunikaci mezi systémy a údaje používané pro autentizaci nakonfigurovaných webových služeb.

Hesla jsou do databáze ukládána v nečitelné podobě neumožňující zpětné získání jejich původních hodnot. K tomu, aby i dvě stejná hesla byla v databázi uložena jinak, je navíc použita tak zvaná sůl. Jedná se o pseudonáhodnou posloupnost bitů, která je nově vygenerovaná pro každé heslo. Pro uložení do databáze je z původního hesla a soli získána pomocí funkce pro odvození klíče PBKDF2[20], která pracuje na principu kryptografické hašovací funkce, posloupnost bitů – krytpografický heš. A až tento krytpografický heš společně se solí jsou uloženy do databáze. Při přihlašování uživatele je z databáze nejdříve získán uložený krytpografický heš a sůl. Poté je na základě zadaného hesla a soli vypočítán kryptografický heš pro zadané heslo a je porovnán s uloženým krytpografickým hešem. V případě, že jsou si rovny, autentizace proběhla úspěšně a uživateli je povolen přístup dle jeho rolí v systému, jinak je vyzván ke zopakování přihlášení.

V případě tajných klíčů pro autentizaci systémů a údajů pro autentizaci webových služeb nelze ukládat tato data stejně jako hesla, protože je nutné s nimi v systému pracovat. Tato data jsou do databáze ukládána šifrovaně symetrickou šifrou, a to opět pomocí algoritmu AES. Klíče použité pro šifrování mají délku 256 bitů a jsou do kódu dočasně zavedeny před jeho kompilací a pak opět smazány, aby nebyly uloženy v systémech kontroly verzí kódu, jako je například *GitHub*. Poté už je je nutné zajistit, aby ke zkompilovaným souborům měly přístup pouze oprávněné osoby a klíče byly bezpečně uloženy, například na hardwarovou peněženku.

[20]Password-Based Key Derivation Function 2

Kapitola 7

Testování

Tato kapitola se zabývá provedenými testy, které zajistily, že systém pracuje správně dle návrhu v kapitole 5 a že se v systému při provádění operací nevyskytují chyby. Dále tato kapitola popisuje, jakým způsobem bylo ověřeno, že je systém správně navržen a implementované funkce umožňují uživatelům plnit cíle vytyčené v zadání práce.

7.1 Funkční testování

Funkční testování systému bylo zaměřeno zejména na jeho ovládání z grafického uživatelského rozhraní. Jelikož je automatizace testování grafického uživatelského rozhraní poměrně náročná, byly tyto testy prováděny převážně ručně. Pro částečnou automatizaci byl použit koncept *seed data*. Jedná se o nahrání počátečních dat do databáze. Tímto způsobem bylo zajištěno, že pro testování určité funkcionality není nutné v prostředí provádět kroky, které by testování umožnily. Další výhodou je, že po provedení testu lze resetovat stav databáze a provést jej znovu, ať už při nalezení chyby, nebo pro otestování jiného případu užití. Nakonec nahrání testovacích dat do databáze umožňuje simulovat některou funkcionalitu, která má data v databázi vytvářet, ale ještě není implementovaná. Plně automatizované testování pomocí *Unit* testů bylo prováděno pouze u algoritmicky náročných částí implementace. Jedná se zejména o analýzu nahrávaných BPMN modelů, o funkce řešící automatizaci dotazů na webové služby a o funkce implementující zabezpečení. Testování automatizace volání webových služeb bylo provedeno na několika bezplatně dostupných webových službách. Jednalo se například o ARES[1] a *Weather API*[2].

Při testování byl brán ohled na členění systému do jednotlivých modulů a testování probíhalo v několika etapách. Nejdříve jsem testoval jednotlivé funkce modulu při jejich implementaci. Po dokončení implementace všech funkcí modulu, jsem testoval modul jako celek. Jakmile jsem vyřešil problémy vzniklé integrací funkcí modulu a nenacházel jsem další chyby, požádal jsem o otestování někoho dalšího, kdo neznal způsob implementace. Obvykle se jednalo o kolegu z práce nebo rodinného příslušníka. Tento postup se osvědčil, většinou nedocházelo k nalezení nějaké chyby, ale spíše byly vznášeny připomínky na logiku jednotlivých operací nebo ovládání uživatelského rozhraní. Tyto postřehy jsem následně zakomponoval do výsledné implementace modulu. Po dokončení implementace všech modulů jsem odstranil testovací data z databáze a testoval jsem jejich vzájemnou integraci. Tyto

[1] *Administrativní registr ekonomických subjektů* je informační systém, který umožňuje vyhledávání nad ekonomickými subjekty registrovanými v České republice.

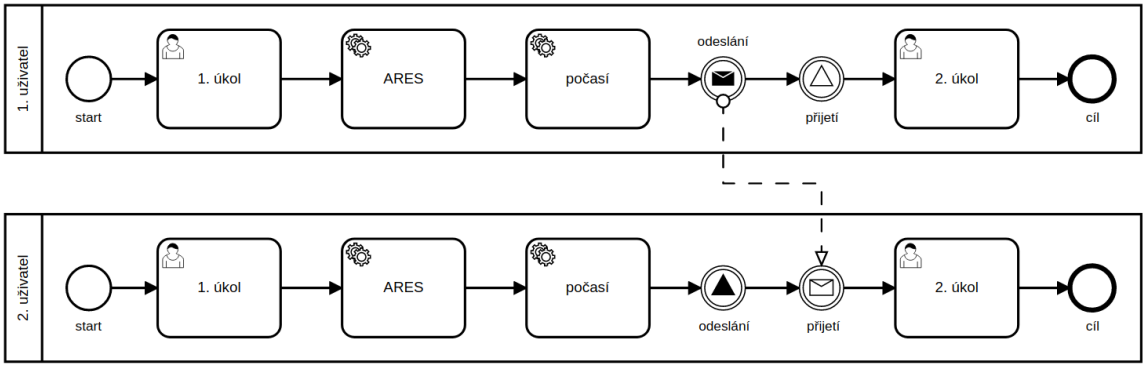
[2] Jedná se o aplikační rozhraní, které umožňuje získávat informace o počasí kdekoliv na zeměkouli.

testy již zahrnovaly celý proces využití systému od vytvoření uživatelských účtů, agendy a nakonfigurování webové služby, přes nahrání BPMN modelu do systému a jeho konfiguraci, po spuštění workflow z modelu a řešení jeho jednotlivých úkolů. Nejdříve byl tento proces několikrát testován v rámci jednoho systému. Po odstranění nalezených chyb byl popsaný proces opakovaně testován i mezi více systémy.

7.2 Uživatelské testování

Jakmile byly všechny funkce systému otestovány, byl systém prezentován dvěma potenciálním uživatelům. Jednalo se o uživatele, kteří aktuálně používají systém SBPM od společnosti *Sperling, s.r.o.* Tento systém se také zabývá tématikou konfigurace a interpretace BPMN modelů, neumožňuje ale dotazování se webových služeb, ani zasílání zpráv mezi paralelně běžícími procesy a jejich synchronizaci. Uživatelské testování se tak soustředilo zejména na tyto funkce.

Každému uživateli byla poskytnuta jiná instance systému, aby si mohl vyzkoušet zasílání zpráv a synchronizaci i s jinou organizací. Nejdříve ale byli uživatelé vyzváni k tomu, aby si vytvořili jednoduchý BPMN model pouze s bloky User Task a Service Task. Tento model měli nahrát do systému, nakonfigurovat jeho jednotlivé bloky, spustit jej a pokusit se vyřešit všechny úkoly vzniklého workflow. Konfigurace bloku Service Task také znamenala, že si předtím uživatelé museli v systému vytvořit webovou službu, kterou měl blok volat. Následně byl jeden z uživatelů vyzván, aby do svého systému nahrál BPMN model, viz obrázek 7.1. Tento model měl daný uživatel sdílet druhému uživateli. Propojení systémů bylo nastaveno již předem. Nejedná se o funkci, kterou by uživatelé běžně prováděli. Po sdílení modelu jej měli oba uživatelé nakonfigurovat tak, aby blok *1. úkol* obsahoval nějaké atributy, které poté odešlou v bloku *odeslání*. Dále měli nastavit, aby bylo v bloku *1. úkol* vyplněno IČO pro volání webové služby ARES a aby se webová služba *počasí* volala se získanou informací o městě, ve kterém sídlí společnost, o níž byly získány informace voláním první webové služby. Obě tyto služby byly předem v systému připraveny. Následně měl uživatel s blokem Catching Signal Event nakonfigurovat správného odesílatele a oba uživatelé si měli v blocích *2. úkol* zobrazit získané údaje z volání webových služeb a také příchozí data od druhého uživatele.



Obrázek 7.1: BPMN model použitý pro uživatelské testování

S grafickým uživatelským rozhraním byli uživatelé spokojení. Samozřejmě se vyskytly i drobné připomínky. Ty ale většinou byly v tom smyslu, že jsou uživatelé zvyklí na používání systému SBPM, který se po stránce ovládání a vzhledu výrazně liší. Právě moderní

vzhled byl spíše předmětem chvály. Automatizace volání webových služeb uživatelům připadala velmi dobře zpracovaná a použitelná v praxi. Zasílání zpráv a synchronizace se jim líbila zejména v rámci jednoho systému. U zasílání zpráv a synchronizace mezi více systémy měli uživatelé pochybnosti o tom, jestli je reálné, aby daný systém používaly všechny organizace, se kterými chtějí spolupracovat. To ale není problém systému, nicméně jsme s uživateli vymysleli řešení, které by tento problém mohlo eliminovat. Systém by do budoucna mohl umožňovat sdílet BPMN modely, provádět zasílání zpráv a synchronizaci i mezi jednotlivými agendami v rámci jednoho systému. Stejně jak to aktuálně probíhá mezi spolupracujícími systémy. Nová agenda by pak mohla být vyčleněna právě pro tuto organizaci.

Kapitola 8

Závěr

Hlavním cílem práce je seznámení se se standardem BPMN, návrh způsobu realizace integrace BPMS s ostatními systémy a synchronizace paralelních BPMN procesů nebo výměna zpráv mezi nimi. Práce se také zabývá implementací provedeného návrhu.

V teoretické části práce byly nejdříve prozkoumány jiné metody modelovaní podnikových procesů, které mohou být v některých případech alternativou ke standardu BPMN. Dále byl v dostatečném rozsahu popsán standard BPMN, zejména se zaměřením na BPMN bloky, a byl proveden výběr vhodných bloků a jiných BPMN elementů pro implementaci. Na konci teoretické části byl čtenář seznámen s možnostmi integrace informačních systémů pomocí aplikačního rozhraní a se zabezpečením komunikace přes internet.

Praktická část začíná zjištěním, že pro naplnění cílů práce bude vhodné implementovat distribuovaný informační systém. Dále je popsán návrh tohoto systému z pohledu uživatelských akcí a rolí, je navrhnut způsob výměny zpráv a způsob synchronizace paralelních BPMN procesů. Následně se práce zabývá návrhem základních zobrazení grafického uživatelského rozhraní a vyhotovením návrhu datového modelu systému. Poté je čtenář seznámen s výběrem programovacích nástrojů, architektury a návrhového vzoru a je mu popsána implementace systému ve vztahu ke zvolené architektuře a návrhovému vzoru. Následuje popis způsobu integrace systému s okolními systémy formou automatizovaného volání webových služeb v rámci interpretace BPMN workflow a objasnění implementace výměny zpráv a synchronizace paralelních BPMN procesů. V závěru práce jsou popsány bezpečnostní mechanismy, které systém implementuje, aby bylo možné jeho nasazení v praxi. Nakonec je uveden způsob testování funkcí systému a ověření použitelnosti implementované funkcionality uživatelským testováním.

Přestože výsledná implementace systému vhodně demonstruje vytyčené cíle práce a v některých oblastech je i předčí[1], je zde několik oblastí, ve kterých by mohlo dojít k vylepšením a dalšímu rozvoji.

Vylepšení by mohlo být provedeno u interpretace BPMN workflow, kdy po vyřešení úkolu uživatelem může následovat několik vnořených HTTP dotazů, jak je zobrazeno na obrázku 5.2, a vyřešení úkolu tak může trvat poměrně dlouho. Jako řešení se nabízí provést v rámci HTTP dotazu iniciovaném uživatelem pouze nezbytné operace pro vyřešení úkolu, zaslat odpověď na tento dotaz a zbylou část interpretace přesunout do nového vlákna či procesu. Další vylepšení se týká grafického uživatelského rozhraní. Aktuálně systém podporuje především zobrazení na horizontálních monitorech, a i když vzhledem k povaze sytému není

[1] zejména v oblasti správy uživatelských účtů a bezpečnosti či důrazu na moderní a intuitivní grafické uživatelské rozhraní

očekáváno použití na mobilních zařízeních, do budoucna by bylo vhodné tuto možnost přidat. Řešením je navrhnout umístění ovládacích prvků na menších obrazovkách a definovat různé CSS styly pro různě velké zobrazovací plochy.

Další rozvoj systému by byl potřeba v implementaci více BPMN bloků a jiných elementů. Pro reálné použití je potřeba implementovat zejména BPMN brány a přidat implementaci zbylých BPMN úkolů a událostí. Tato oblast bude spojená zejména s rozvojem datového modelu systému, se sémantickou analýzou nahraných BPMN modelů a s interpretací běžících workflow. S implementací BPMN bran poté souvisí i druhá oblast možného rozvoje. Jde o to, že při použití bran při návrhu BPMN modelu se z tohoto modelu efektivně stává graf. Ukládání grafových struktur do relačních databází je sice možné, ale není optimální. Proto by bylo vhodné prozkoumat možnosti uložení těchto dat do grafových databází a zjistit, jaké jsou možnosti jejich integrace s databázemi relačního typu.

Literatura

[1] Abari, O., Shola, J. a Philip, S. COMPARATIVE ANALYSIS OF DISCRETE LOGARITHM AND RSA ALGORITHM IN DATA CRYPTOGRAPHY. *International Journal of Computer Science and Information Security*. Únor 2015, sv. 13, s. 24–31.

[2] AltexSoft. *Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC* [online]. Květen 2020 [cit. 2022-01-14]. Dostupné z: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/.

[3] Barker, E. *Recommendation for Key Management: Part 1 – General*. Květen 2020 [cit. 2022-03-25]. Dostupné z: https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[4] Battle, R. a Benson, E. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Journal of Web Semantics* [online]. 2008, sv. 6, č. 1, s. 61–69, [cit. 2022-01-14]. DOI: https://doi.org/10.1016/j.websem.2007.11.002. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1570826807000510.

[5] Bennett, C. H., Bernstein, E., Brassard, G. a Vazirani, U. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing*. 1997, sv. 26, č. 5, s. 1510–1523, [cit. 2022-03-26]. DOI: 10.1137/S0097539796300933. Dostupné z: https://doi.org/10.1137/S0097539796300933.

[6] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N. et al. *Simple Object Access Protocol (SOAP) 1.1* [online]. The World Wide Web Consortium, květen 2000 [cit. 2022-01-16]. Dostupné z: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[7] Chai, W. a Brush, K. *PERT chart* [online]. Únor 2021 [cit. 2021-11-29]. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/PERT-chart.

[8] Desai, S. *Live IoT Data Subscription with Apollo GraphQL and MQTT* [online]. Srpen 2019 [cit. 2022-01-19]. Dostupné z: https://medium.com/@shantanoodesai/live-iot-data-subscription-with-apollo-graphql-and-mqtt-60b7c5a86cde.

[9] Object Management Group. *Business Process Model and Notation (BPMN)*. Standard. Leden 2011 [cit. 2021-12-15]. Dostupné z: https://www.omg.org/spec/BPMN/2.0.

[10] Grégr, M. *Network and Services logging*. 2021 [cit. 2022-03-28]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=/Fcourse/FISA-IT/Flectures/Fisa-logovani.pdf.

[11] Hayward, M. *Quantum Computing and Shor's Algorithm.* Únor 2005 [cit. 2022-03-27]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1509&rep=rep1&type=pdf.

[12] IBM. *An overview of the SSL or TLS handshake* [online]. Březen 2022 [cit. 2022-03-26]. Dostupné z: https://www.ibm.com/docs/en/ibm-mq/7.5?topic=ssl-overview-tls-handshake.

[13] International Organization for Standardization. *High-level Petri Nets - Concepts, Definitions and Graphical Notation.* Standard. Květen 2002 [cit. 2022-01-22]. Dostupné z: http://www.petrinets.info/docs/pnstd-4.7.4.pdf.

[14] Khachatryan, G. *What is GraphQL?* [online]. Červen 2018 [cit. 2022-01-18]. Dostupné z: https://medium.com/devgorilla/what-is-graphql-f0902a959e4.

[15] Kopp, C. M. *Program Evaluation Review Technique (PERT) Chart* [online]. Říjen 2020 [cit. 2021-11-28]. Dostupné z: https://www.investopedia.com/terms/p/pert-chart.asp.

[16] Krasner, G. E. a Pope, S. T. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.* 1988 [cit. 2021-12-18]. Dostupné z: https://web.archive.org/web/20100921030808/http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf.

[17] Matoušek, P. *Síťové služby a jejich architektura.* Publishing house of Brno University of Technology VUTIUM, 2014. 396 s. ISBN 978-80-214-3766-1.

[18] Meinhardt, S. a Popp, K. Configuring Business Application Systems. In: *Handbook on Architectures of Information Systems.* Leden 2006, s. 705–721. DOI: 10.1007/3-540-26661-5_30. ISBN 978-3-540-26661-7. Dostupné z: https://www.researchgate.net/publication/226169852_Configuring_Business_Application_Systems.

[19] Monus, A. *SOAP vs REST vs JSON - a 2021 comparison* [online]. Březen 2021 [cit. 2022-01-15]. Dostupné z: https://raygun.com/blog/soap-vs-rest-vs-json/.

[20] National Security Agency. *NSA Suite B Cryptography* [online]. Leden 2009 [cit. 2022-03-24]. Dostupné z: https://web.archive.org/web/20090207005135/http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml.

[21] Olenski, J. *What is ECC and why would I want to use it?* [online]. Květen 2015 [cit. 2022-03-25]. Dostupné z: https://www.globalsign.com/en/blog/elliptic-curve-cryptography.

[22] Recker, J., Rosemann, M., Indulska, M. a Green, P. Business Process Modeling - A Comparative Analysis. *Journal of the Association for Information Systems* [online]. aisel.aisnet.org. Duben 2009, sv. 10, č. 4, s. 333–363, [cit. 2021-12-06]. DOI: 10.17705/1jais.00193. Dostupné z: https://aisel.aisnet.org/jais/vol10/iss4/1/.

[23] Savvy Security. *What is Asymmetric Encryption? Understand with Simple Examples* [online]. Leden 2021 [cit. 2022-03-26]. Dostupné z: https://cheapsslsecurity.com/blog/what-is-asymmetric-encryption-understand-with-simple-examples/.

[24] Svetlin Nakov, P. *Practical Cryptography for Developers*. 2018. ISBN 978-619-00-0870-5. Dostupné z: https://cryptobook.nakov.com.

[25] Teknomo, K. *Queuing Theory Tutorial - Classification of Queuing Model using Kendal Notation* [online]. 2014 [cit. 2022-01-22]. Dostupné z: https://people.revoledu.com/kardi/tutorial/Queuing/Kendall-Notation.html.

[26] Tennakoon, J. *GraphQL — Common Disadvantages Over REST and Solutions to Overcome them* [online]. Červen 2021 [cit. 2022-01-18]. Dostupné z: https://levelup.gitconnected.com/graphql-common-disadvantages-over-rest-and-solutions-to-overcome-them-70cbaca42a44.

[27] The GraphQL Foundation. *Schemas and Types* [online]. 2022 [cit. 2022-01-15]. Dostupné z: https://graphql.org/learn/schema/.

[28] Wang, J. Petri Nets. In: *Formal Methods in Computer Science*. červen 2019, s. 201–243 [cit. 2022-01-22]. DOI: 10.1201/9780429184185-8. ISBN 9780429184185. Dostupné z: https://www.researchgate.net/publication/334339931_Petri_Nets.

[29] Wikipedia contributors. *Event-driven process chain — Wikipedia, The Free Encyclopedia*. 2021 [cit. 2022-01-23]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Event-driven_process_chain&oldid=1055874605.

[30] Wikipedia contributors. *Cryptographic hash function — Wikipedia, The Free Encyclopedia*. 2022 [cit. 2022-03-26]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Cryptographic_hash_function&oldid=1077451766.

[31] Wikipedia contributors. *Information security — Wikipedia, The Free Encyclopedia*. 2022 [cit. 2022-03-21]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Information_security&oldid=1080183606.

[32] Wikipedia contributors. *Replay attack — Wikipedia, The Free Encyclopedia*. 2022 [cit. 2022-03-27]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Replay_attack&oldid=1071941133.

[33] Wikipedie. *GraphQL — Wikipedie: Otevřená encyklopedie*. 2020 [cit. 2022-01-17]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=GraphQL&oldid=19282537.

[34] Wikipedie. *SOAP — Wikipedie: Otevřená encyklopedie*. 2021 [cit. 2022-01-16]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=SOAP&oldid=20344803.

Příloha A

Obsah přiloženého paměťového média

V následujícím výčtu jsou uvedeny a popsány adresáře a soubory projektu na nejvyšší úrovni zanoření.

- `BPMS/`: Adresář obsahuje jádro systému – HTTP server, třídy definující HTTP koncové body systému – řadiče, soubory generující HTML obsah – pohledy, soubory s CSS styly, soubory s JavaScript kódem a konfigurační soubor systému.
- `BPMS_BL/`: Adresář obsahuje třídy, které implementují převážnou část obchodní logiky systému – fasády, a k nim další pomocné třídy.
- `BPMS_Common/`: Adresář obsahuje výčtové typy a jiné statické třídy, které je nutné odkazovat ze všech částí systému.
- `BPMS_DAL/`: Adresář obsahuje třídy, které pomocí ORM definují schéma databáze, a třídy implementující komunikaci s databází – repozitáře.
- `BPMS_DTOs/`: Adresář obsahuje třídy definující objekty používané k mapování získaných dat z databáze pro jejich následné zobrazení v pohledech.
- `Models/`: Adresář obsahuje BPMN modely, které lze nahrát do systému, a vyzkoušet si tak jeho funkce.
- `BPMS.sln`: Soubor pro otevření projektu ve Visual Studio 2022.

Příloha B

Návod pro lokální spuštění

Pro lokální spuštění je nutné mít nainstalované produkty .NET SDK 6.0[1], SQL Server[2] databázi a nástroj LibMan[3]. Následuje návod pro lokální spuštění na operačním systému Windows a operačních systémech linuxového typu.

B.1 Spuštění na Windows

1. Pokud nechcete používat Visual Studio 2022, postupujte dle návodu pro Linux.
2. Otevřete projekt ve Visual Studio 2022 pomocí souboru `BPMS.sln`.
3. Zkontrolujte, že Vaše instalace SQL server podporuje LocalDB.
4. Pokud ne, otevřete soubor `BPMS/appsettings.json` a v záznamu `"DB"` nahraďte připojovací řetězec k databázi záznamem identifikující prázdnou databázi, kterou chcete použít. Jinak tento krok můžete přeskočit, databáze se vytvoří sama.
5. V souboru `BPMS/appsettings.json` nahraďte obsah záznamu `"FileStore"` adresářem s adekvátními právy na zápis, do kterého mají být ukládány soubory nahrané do systému.
6. Stáhněte klientské knihovny vybráním možnosti *Restore Client-Side Libraries* po kliknutí pravým tlačítkem myši na soubor `BPMS/libman.json`.
7. V souboru `BPMS/Properties/launchSettings.json` případně změňte URL nebo pouze port, na kterém aplikace poběží.
8. Spusťte ladění ve Visual Studio 2022.
9. Na přihlašovací stránce zvolte *První přihlášení* a vytvořte si heslo pro účet s přihlašovacím jménem *admin*. Pomocí tohoto účtu lze následně vytvořit další uživatelské účty.
10. V modulu *Systémy* editujte URL systému *Tento systém* tak, aby odpovídala URL, na které je systém dostupný.

[1] odkaz pro stažení: `https://dotnet.microsoft.com/en-us/download`

[2] odkaz pro stažení: `https://www.microsoft.com/en-us/sql-server/sql-server-downloads`

[3] návod pro instalaci: `https://docs.microsoft.com/en-us/aspnet/core/client-side/libman/libman-cli?view=aspnetcore-6.0`

B.2 Spuštění na Linux

1. Otevřete soubor `BPMS/appsettings.json` a v záznamu `"DB"` nahraďte připojovací řetězec k databázi záznamem identifikující prázdnou databázi, kterou chcete použít.
2. V souboru `BPMS/appsettings.json` nahraďte obsah záznamu `"FileStore"` adresářem s adekvátními právy na zápis, do kterého mají být ukládány soubory nahrané do systému.
3. Navigujte do adresáře `BPMS` a spusťte zde příkaz `libman restore`, který stáhne klientské knihovny.
4. V souboru `BPMS/Properties/launchSettings.json` případně změňte URL nebo port, na kterém aplikace poběží.
5. V adresáři `BPMS` spusťte systém použitím příkazu `dotnet run`.
6. Navigujte na použitou URL ve webovém prohlížeči.
7. Na přihlašovací stránce zvolte *První přihlášení* a vytvořte si heslo pro účet s přihlašovacím jménem *admin*. Pomocí tohoto účtu lze následně vytvořit další uživatelské účty.
8. V modulu *Systémy* editujte URL systému *Tento systém* tak, aby odpovídala URL, na které je systém dostupný.