Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Analýza a návrh informačního systému pro vedení agend malých obcí

Bc. Miloš Mojžiš

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Miloš Mojžiš

Systémové inženýrství a informatika
Informatika

Název práce

Analýza a návrh informačního systému pro vedení agend malých obcí

Název anglicky

Analysis and design of an information system for managing agendas of small municipalities

---

Cíle práce

Cílem diplomové práce je analyzovat a navrhnout informační systém prostřednictvím modelování v jazyce UML. Modelovaný informační systém bude sloužit k podpoře, zjednodušení a automatizaci vybraných agend malých obcí, které často provádějí řadu agend prostřednictvím tabulkového kalkulátoru či papírové formy. Malé obce často nejsou personálně dostatečně vybaveny k tomu, aby všechny agendy efektivně zvládly. Při návrhu bude autor spolupracovat s vedením své domovské obce Nelahozeves.

Dílčí cíle diplomové práce jsou:

- teoreticky popsat postupy, standardy a nástroje analýzy a návrhu informačního systému, zejména nejlepší praktiky softwarového vývoje, základní principy modelování informačního systému, modelování pomocí jazyka UML, stručně analyzovat dostupné nástroje,

- popsat věcnou problematiku, která bude podkladem pro analýzu a návrh informačního systému, tzn. analyzovat nejdůležitější agendy malých obcí a jejich potenciál automatizace,

- specifikovat uživatelské požadavky (částečně ve spolupráci s představiteli vybrané obce),

- provést analýzu a návrh vybraných oblastí IS pomocí jazyka UML.

Metodika

Metodika diplomové práce je založena na studiu odborných informačních zdrojů z oblasti analýzy a návrhu informačních systémů a z oblasti věcné problematiky, kterou má informační systém řešit, tj. vybraných agend malých obcí. Autor zároveň bude čerpat ze své odborné praxe v obou oblastech, neboť působí jako IT analytik a zároveň externě spolupracuje s obcí Nelahozeves v několika oblastech její samosprávné činnosti. Získané či již existující znalosti autora budou využity pro analýzu a návrh příslušného informačního systému. Autor pro tvorbu praktické části práce hodlá spolupracovat přímo s představiteli obce. Pro návrh modelu informačního systému budou použity metody softwarového inženýrství a jazyk UML (Unified

Modeling Language). Na základě shrnutí teoretických poznatků a výsledků vlastní části práce budou formulovány závěry diplomové práce, zejména možnost využití navrženého modelu v praxi a jeho předpokládané přínosy.

Doporučený rozsah práce

60-80

Klíčová slova

Analýza a návrh, UML, informační systém, obec, agenda obce, automatizace

---

Doporučené zdroje informací

ARLOW, J. – NEUSTADT, I. *UML 2 a unifikovaný proces vývoje aplikací : objektově orientovaná analýza a návrh prakticky.* Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

BOOCH, Grady, James RUMBAUGH a Ivar JACOBSON. The unified modeling language user guide. 2nd ed. Upper Saddle River: Addison-Wesley, 2005. ISBN 9780321267979.

BRUCKNER, Tomáš. Tvorba informačních systémů: principy, metodiky, architektury. Praha: Grada, 2012. Management v informační společnosti. ISBN isbn978-80-247-4153-6.

BUCHALCEVOVÁ, Alena a Iva STANOVSKÁ. Příklady modelů analýzy a návrhu aplikace v UML. Praha: Oeconomica, 2013. ISBN 978-80-245-1922-7.

KOČÍ, Roman. Obecní samospráva v České republice. Praha: Leges, 2012. ISBN 978-80-87576-28-1

PAGE-JONES, Meilir. Základy objektově orientovaného návrhu v UML. Praha: Grada, 2001. Moderní programování. ISBN 80-247-0210-x.

VRANA, Ivan. Projektování informačních systémů s UML. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2008. ISBN 978-80-213-1817-5.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

doc. Ing. Jan Tyrychtr, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

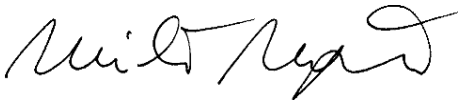
Děkan

V Praze dne 08. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Analýza a návrh informačního systému pro vedení agend malých obcí“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 29. 3. 2023



Poděkování

Rád bych touto cestou poděkoval vedoucímu diplomové práce Ing. Janu Tyrychtrovi, Ph.D., za vedení diplomové práce a za kritické připomínky, které mi umožnily zvýšit kvalitu práce.

Analýza a návrh informačního systému pro vedení agend malých obcí

Abstrakt

Tato diplomová práce se zabývá analýzou a návrhem informačního systému pro podporu agend malých obcí v jazyce Unified Modelling Language (UML). Modelovaný informační systém, bude-li následně implementován, bude sloužit k podpoře, zjednodušení a automatizaci vybraných agend malých obcí, které často provádějí řadu agend prostřednictvím tabulkového kalkulátoru či dokonce papírové formy.

V teoretické části práce jsou popsány postupy, standardy a nástroje analýzy a návrhu informačního systému, zejména nejlepší praktiky softwarového vývoje a základní principy modelování informačního systému pomocí jazyka UML. V rámci teoretické části jsou rovněž vybrány agendy, které by měly být součástí analyzovaného systému.

V praktické části je provedena samotná analýza a návrh informačního systému pro podporu agend malých obcí v podobě návrhu UML diagramů – diagramů případů užití, diagramů tříd, stavových diagramů, diagramů aktivit a sekvenčních diagramů. Jednotlivé diagramy jsou opatřeny doprovodnými textovými popisy. Při návrhu řešení autor úzce spolupracuje s vedením své domovské obce Nelahozeves.

Klíčová slova: Analýza a návrh, UML, informační systém, obec, agenda obce, automatizace

Analysis and design of an information system for managing agendas of small municipalities

Abstract

This thesis deals with the analysis and design of an information system for supporting the agendas of small municipalities in the Unified Modelling Language (UML). The modelled information system, if subsequently implemented, will serve to support, simplify and automate selected agendas of small municipalities, which often implement a number of agendas through a spreadsheet or even in paper form.

The theoretical part of the thesis describes the procedures, standards and tools of information system analysis and design, especially the best practices of software development and the basic principles of information system modelling using the UML language. Agendas that should be part of the analysed system are also selected as part of the theoretical section.

In the practical part, the actual analysis and design of the information system for supporting the agendas of small municipalities is carried out in the form of the design of UML diagrams – use case diagrams, class diagrams, state diagrams, activity diagrams and sequence diagrams. Individual diagrams are provided with accompanying text descriptions. When designing the solution, the author works closely with the management of his home village of Nelahozeves.

Keywords: Analysis and design, UML, information system, municipality, agenda of municipality, automation

Obsah

1 Úvod

Podpora informačních technologií je v dnešní době nezbytností a samozřejmostí prakticky ve všech oblastech lidského života. Výjimkou nejsou ani samosprávy obcí včetně těch nejmenších. Malých obcí je přitom na území České republiky velké množství, podle studie OECD je Česká republika v tomto ohledu unikátem[1]. Z celkem 6 258 obcí (dle statistiky Českého statistického úřadu k 1. 1. 2021[2]) má méně než 500 obyvatel 54 % obcí. Nejmenších obcí do 200 obyvatel je 1 400, tedy 22 %. Legislativa České republiky klade přitom na obce celou řadu požadavků, a byť jsou povinnosti a role obcí odstupňovány podle velikosti, celou řadu agend musí vykonávat všechny obce bez rozdílu. U malých obcí naráží výkon samosprávných agend na řadu problémů, zejména finančních a personálních. Malé obce si nemohou dovolit zaměstnávat větší množství referentů pro různé agendy, ostatně často sám starosta vykonává v nejmenších obcích svou funkci jako neuvolněný, tj. při zaměstnání. Svěřené agendy jsou nezřídka vykonávány s pomocí aplikace Microsoft Excel či dokonce na papíře.

Právě u malých obcí je tedy kladen velký důraz na efektivitu výkonu samosprávné činnosti, kde hrají informační technologie nezastupitelnou roli. Tomu bylo přizpůsobeno téma této diplomové práce, která se zabývá analýzou a návrhem informačního systému pro podporu agend malých obcí.

Analýza a návrh informačního systému jsou realizovány pomocí grafického jazyka Unified Modelling Language (UML), který je v současné době považován za standard pro objektově orientovanou analýzu a návrh softwarových aplikací.

Při analýze a návrhu je kladen důraz na celkovou jednoduchost, vnitřní soudržnost a logiku aplikace, čehož je dosaženo i s pomocí průběžných konzultací s představiteli vybrané malé obce.

---

[1] https://www.irozhlas.cz/zpravy-domov/v-poctu-malych-obci-je-cesko-evropskym-unikatem-ctvrtina-vesnic-ma-mene-nez-200_1709031400_jgr

[2] https://www.czso.cz/csu/czso/maly-lexikon-obci-ceske-republiky-2021

2 Cíl práce a metodika

2.1 Cíl práce

Cílem diplomové práce je analyzovat a navrhnout informační systém prostřednictvím modelování v jazyce UML. Modelovaný informační systém bude sloužit k podpoře, zjednodušení a automatizaci vybraných agend malých obcí, které často provádějí řadu agend prostřednictvím tabulkového kalkulátoru či papírové formy. Při návrhu bude autor spolupracovat s vedením své domovské obce Nelahozeves.

Dílčí cíle diplomové práce jsou:

- teoreticky popsat postupy, standardy a nástroje analýzy a návrhu informačního systému, zejména nejlepší praktiky softwarového vývoje, základní principy modelování informačního systému, modelování pomocí jazyka UML, stručně analyzovat dostupné nástroje,
- popsat věcnou problematiku, která bude podkladem pro analýzu a návrh informačního systému, tzn. analyzovat nejdůležitější agendy malých obcí a jejich potenciál automatizace,
- specifikovat uživatelské požadavky (částečně ve spolupráci s představiteli vybrané obce),
- provést analýzu a návrh vybraných oblastí IS pomocí jazyka UML.

2.2 Metodika

Metodika diplomové práce je založena na studiu odborných informačních zdrojů z oblasti analýzy a návrhu informačních systémů a z oblasti věcné problematiky, kterou má informační systém řešit, tj. vybraných agend malých obcí. Autor zároveň čerpal ze své odborné praxe v obou oblastech, neboť působí jako IT analytik a zároveň externě spolupracuje s obcí Nelahozeves v několika oblastech její samosprávné činnosti. Získané či již existující znalosti autora jsou v této práci využity pro analýzu a návrh příslušného informačního systému.

První, teoretická část práce, popisuje jednotlivé fáze analýzy a návrhu informačního systému se zaměřením na objektově orientovaný přístup a jazyk Unified Modelling Language (UML). Zmíněna je i návaznost UML na metodiku vývoje UP, jakož i CASE nástroje pro modelování v jazyce UML. V teoretické části jsou popsány i všechny typy diagramů použité v této práci.

Druhým, byť menším pilířem teoretické části práce je problematika agend malých obcí. S oporou platné legislativy a navazujících metodických materiálů jsou shrnuty hlavní agendy, kterými se malé obce ve své činnosti zabývají, a na základě stanovených kritérií jsou následně vybrány agendy, kterým se práce věnuje v praktické části.

Praktická část se pak zabývá vlastní analýzou a návrhem aplikace pro podporu vybraných agend malých obcí. Praktická část se opírá o teoretická východiska z první části. Pro návrh modelu informačního systému jsou použity metody softwarového inženýrství a jazyk UML.

Na základě shrnutí teoretických poznatků a výsledků vlastní části práce jsou formulovány závěry diplomové práce, zejména možnost využití navrženého modelu v praxi a jeho předpokládané přínosy.

Metodická poznámka: Na základě konzultací s vedoucím práce autor zvažoval způsob odkazování na podkladovou literaturu a citování použitých zdrojů. Ve finální verzi práce jsou nakonec používány dva způsoby odkazování:

- *Standardní citační norma ČSN ISO 690 – pro odkazy na literaturu, kterou práce využívá opakovaně a systematičtěji a která je souhrnně uvedena v kapitole 7*
- *a poznámky pod čarou pro jednorázové a méně významné odkazy*[3].

[3] Důvodem pro využití poznámek pod čarou je, že méně významných a spíše jednorázových odkazů autor v práci využívá poměrně frekventovaně a nepovažoval za vhodné tyto zdroje zařazovat mezi hlavní použitou literaturu, protože by pak mezi nimi snadno zanikly zdroje hlavní a významné.

3 Teoretická východiska

Téma diplomové práce stojí na dvou základních stavebních kamenech:

- analýza a návrh informačního systému,
- problematika agend malých obcí.

První oblast, analýza a návrh informačního systému, spadá pod obor informačních technologií. Problematika agend malých obcí je jednou z hlavních oblastí veřejné správy.

V této teoretické části rozebereme každou z uvedených oblastí samostatně, s mírně větším důrazem na první zmiňovanou, s ohledem na to, že práce je předmětem obhajoby na Katedře informačního inženýrství Provozně ekonomické fakulty ČZU. V navazující praktické části, v níž provádíme samotnou analýzu a návrh hypotetického informačního systému pro podporu agend malých obcí, obě domény spojujeme dohromady.

Spojením obou oblastí získáváme disciplínu označovanou jako business analýza[4], jejíž náplň odpovídá této práci.

3.1 Analýza a návrh informačního systému

Vzhledem k zaměření diplomové práce nelze teoretickou část zahájit jinak než definováním pojmů „analýza“ a „návrh“ informačního systému se zaměřením na objektově

[4] Autor, i pod vlivem části podkladové literatury (Bruckner, 2012) zvažoval, zda v této práci pracovat s termínem „business“ nebo „byznys“. Nakonec se rozhodl užívat původní anglický termín „business“, neboť subjektivně nepociťuje míru počeštění tohoto výrazu jako dostatečnou, a rovněž jakožto profesí business analytik (nikoliv byznys analytik) osobně vnímá určitý významový posun mezi termíny „business“ a „byznys“. Zatímco „business“ označuje obecněji odvětví, obor, branže, tak termín „byznys“ je v odborné i populární literatuře používán spíše ve smyslu „obchod“ či „obchodní podnikání“. Významový rozdíl mezi oběma termíny hraje roli i v souvislosti s touto prací: Předmětem navrhovaného informačního systému jsou veřejnosprávní agendy malých obcí. Tyto činnosti nelze považovat za „byznys“, neboť nejde o podnikatelskou činnost a jejich cílem není dosažení zisku. Věcnou náplň těchto činností ale lze podřadit pod termín „business“ (ve smyslu odvětví).

orientované programování, na jehož principech staví i jazyk Unified Modelling Language (UML), který je použit pro modelování aplikace v praktické části práce.

3.1.1 Úvod do analýzy a návrhu informačního systému

Smyslem analýzy informačního systému je vytvoření analytického modelu, v němž budou zachyceny podstatné požadavky a charakteristické rysy budoucího informačního systému. Analytický model popisuje, co má systém dělat, ale nezabývá se (příliš) otázkou, jak vnitřně je systém uspořádán a jak jsou jednotlivé funkce implementovány – to je již věcí navazující fáze, návrhu informačního systému. Je třeba v této souvislosti podotknout, že hranice mezi analýzou a návrhem bývá místy velmi neostrá a nezřídka je určena metodickým postupem, na jehož základ pracuje příslušný vývojový tým.

„Základním krokem analýzy je správné rozdělení požadavků mezi jednotlivé subsystémy a komponenty. Chybné rozdělení funkcí má špatný vliv na celý systém. Od softwaru se například může požadovat něco, co by měl dělat hardware (anebo naopak), systém může být pomalý a některé komponenty se nemusí dát nahradit lepšími.“ (Wiegers, 2008, s. 273)

Vytvoření modelu analýzy je považováno za strategickou aktivitu vývoje informačního systému. Při tvorbě analytického modelu se dbá hlavně na to, aby byl model co nejjednodušší a aby byla dodržena některá další pravidla:

- analytický model má být vytvořen v jazyce, kterému rozumí zadavatel,
- každý diagram modelu má objasňovat část chování systému,
- popis případů užití nemá být zbytečně detailní (detailní specifikace patří do návrhu),
- analytický model má obsahovat především třídy problémové domény,
- analytický model by měl být užitečný pro maximální okruh osob participujících na projektu (uživatelé, návrháři, vývojáři, management).

3.1.2 Historický vývoj disciplíny analýzy a návrhu

Projekt vývoje informačních systémů se skládá z celé řady fází, přičemž s ohledem na zaměření a cíle této diplomové práce se budeme soustředit zejména na fázi předcházející vlastní realizaci, tedy v obecném slova smyslu projektové přípravě. V jejím rámci se definuje architektura budoucího informačního systému a detailně specifikuje jeho návrh. Elaborační fáze vývoje informačního systému se někdy přirovnává k projektové přípravě stavby. Každý považuje samozřejmé, že například výstavbě domu předchází vypracování detailní projektové dokumentace, jejíž podoba se řídí poměrně striktními standardy a normami. Je pozoruhodné, že naproti tomu při vývoji informačních technologií tato zdánlivá samozřejmost ne vždy platí, takže jsme občas svědky situace, kdy je komplikovaný informační systém podniku budován bez jakékoliv architektonické představy (Bruckner, 2012, s. 235).

Stejně jako u „fyzické“ stavby je kvalitní projekční příprava klíčovým předpokladem úspěšnosti projektu. V opačném případě hrozí živelný vývoj (který však nelze zaměňovat s agilními metodikami, jak bývá občas zvykem), který jen málokdy končí funkčním informačním systémem.

Jedním z důvodů, proč zejména v počátcích vývoje softwaru a obecně oboru informačních technologií nebyla kladena patřičná pozornost analýze a návrhu, je skutečnost, že pro tuto oblast neexistovaly žádné metodiky, což je u nově vznikajícího odvětví logické a pochopitelné.

Ačkoliv vznik této disciplíny, kterou lze šířeji označit jako softwarové inženýrství, sahá do 50. let minulého století (Ramakrishnan, 2012), k významnějšímu rozvoji došlo teprve v 70. a 80. letech v reakci na tzv. softwarovou krizi ze 60. let. Jejími charakteristickými znaky bylo neúnosné prodlužování a prodražování projektů, nízká kvalita programů, nesnadnost či nemožnost údržby a inovace, špatná produktivita práce programátorů, neefektivita vývoje, nejistota výsledku a řada dalších (Softwarové inženýrství, 2001). Až tyto značně neuspokojivé faktory softwarového vývoje vyvolaly reálnou potřebu zavedení softwarových nástrojů, formálních metod a metodik pro analýzu a návrh informačních systémů.

Přesto i v dalších desetiletích nebylo výjimkou, že, díky relativně malé složitosti softwarových produktů (přinejmenším ve srovnání se současností) tvořil celý kód jeden či malá skupina programátorů, aniž by programování předcházela řádná analytická fáze.

Její potřebu přitom definoval už v 60. letech průkopník softwarového inženýrství Larry Constantine, když již tehdy požadoval, aby vývoji počítačového programu předcházel jeho návrh (Constantine, 1968, s. 409).

Další vývoj disciplíny softwarového inženýrství přinesl postupně základní členění elaborační fáze tvorby softwarového produktu na dvě základní části:

- analýzu, která slouží k pochopení potřeb zákazníka (těch, které mají být řešeny připravovaným softwarovým produktem) a jejich transformaci do systematické podoby
- a návrh, který představuje samotné zadání informačního systému.

I Bruckner a kol. striktně obě disciplíny odlišuje, když zdůrazňuje, že „analýza systému je jeho poznáním, nikoliv návrhem“ (Bruckner, 2012, s. 27).

Poznání, že při vývoji informačního systému je třeba oddělit fáze analýzy, návrhu a následně vlastní konstrukce, však bylo teprve předstupněm pro vznik objektově orientované analýzy a návrhu. Tato disciplína vznikala postupně od 60. let a do současné podoby dospěla v 90. letech.

Zmínit můžeme například ideu třídy, kterou přinesli O.-J. Dahl a K Nygaard v roce 1966, kdy se pojem „třída“, tak jak jej chápeme dnes, objevil v návrhu jazyku Simula-67 (Page-Jones, 2001, s. 62). Jazyk neměl příliš následovníků, ale jeho koncepty se staly významnou inspirací pro pozdější jazyky. Jedním z nich byl Smalltalk, uvedený počátkem 80. let, následovaný dalšími objektově-orientovanými jazyky, mezi nimiž byl dodnes používaný jazyk C++. Ruku v ruce s tím se vyvíjely i metody objektově-orientované analýzy a návrhu, bylo ale složité nalézt modelovací jazyk, kterým by bylo možné popsat cílový informační systém dostatečně podrobně a v celé šíři. V 80. a 90. letech si získaly větší

rozšíření metody Object Modelling Technique (OMT)[5], Object-Oriented Software Engineering (OOSE)[6] či Boochova metoda[7]. Každá z nich představovala kompletní metodiku pro analýzu a návrh a každá z nich měla své výhody a nevýhody. Zatímco například Boochova metoda byla silnější pro fáze návrhu a konstrukce systému, OOSE poskytovala „excelentní“ podporu pro případy užití (jakožto způsobu vyjádření uživatelských požadavků), analýzu a high-level návrh. Do třetice OMT bylo nejužitečnější pro analýzu informačních systémů náročných na data (Booch, 2005, s. xvi).

Paralelní existence několika různých modelovacích jazyků vyvolala potřebu jejich určitého sjednocení, kdy autoři či spoluautoři jednotlivých metod cítili, že bude vhodnější vyvinout jeden společný jazyk a v jeho rozvoji pokračovat společně. V tomto okamžiku se poprvé objevuje termín Unified Modelling Language (UML).

3.1.3 Počátky jazyka Unified Modelling Language (UML)

Vývoj UML byl zahájen v roce 1994, s cílem sjednocení Boochovy metody a metody OMT (viz předcházející kapitola). U vzniku UML stojí trojice IT metodiků – Grady Booch, Ivar Jacobson a Jim Rumbaugh, kteří jsou ve vztahu k UML označováni též přezdívkou The Three Amigos. (Page-Jones, 2001, s. 64). Vydáním UML byl do značné míry odstraněn symbolický, terminologický a sémantický chaos, který v oblasti dosud panoval.

První publikovaná verze z roku 1994 nesla číslo 0.8 a brzy si získala velký zájem, kdy řada softwarových organizací deklarovala, že je připravena UML využívat jako strategický nástroj pro svůj business. Klíčovým se stalo též zastřešení vývoje softwarovou firmou Rational Corporation (dnes součást IBM), neboť firma ve svém softwarovém nástroji Rational Rose poskytla pro UML bezkonkurenční podporu a rovněž paralelně vznikající metodika Rational Unified Process (RUP) byla navržena tak, aby byla s UML maximálně kompatibilní. Ačkoliv zpočátku panovaly obavy o uzavřenost metodiky (Arlow, 2007, s. 29

[5] Například https://www.geeksforgeeks.org/software-engineering-object-modeling-technique-omt/

[6] Například https://archive.org/details/objectorientedso00jaco/page/43/mode/2up

[7] Například https://www.conceptdraw.com/How-To-Guide/booch-ood-diagram

- 30), s ohledem na její komerční podhoubí, UML zůstal i v dalších letech a desetiletích otevřeným standardem. Již v roce 1996 byl sdružením OMG (Object Management Group[8]) prohlášen za průmyslový standard v oblasti objektově orientované analýzy a návrhu.

Specifikace se v dalších letech postupně upřesňovala. V roce 2000 byl jazyk upgradován na verzi 2.0, která přinesla kromě jiného sémantiku akcí, umožňující podrobnější specifikaci prvků souvisejících s chováním modelů (Arlow, 2007, s. 30). Tato verze je po několika menších aktualizacích (dosud poslední verze je 2.5.1 z roku 2017[9]) platná dodnes.

3.1.4 Objektově orientovaná analýza a návrh

Objektově orientovaný přístup k vývoji softwaru, jehož podmnožinou je objektově orientovaná analýza a návrh, je dnes široce akceptovaným standardem pro vývoj softwaru. V objektově orientovaném programování (OOP) je program strukturován jako kolekce tříd, kde každá třída popisuje typ objektu.

Objekt představuje entitu – věc, osobu nebo místo – přirozeně se vyskytující v programu. Objekty jsou v OOP hlavním stavebním kamenem znovupoužitelnosti zdrojového kódu. OOP rovněž poskytuje bezpečnost a ochranu dat zavedením viditelnosti objektů a jejich částí (viz dále). Spolu s objekty má OOP také další vlastnosti, jako jsou například metody a zprávy, abstrakce a zapouzdření, stejně jako dědičnost a polymorfismus.

Při vývoji objektově orientovaných softwarových systémů se všechno stává třídou. Třída je seskupení objektů (například osob nebo věcí) majících společné vlastnosti. Instance třídy se nazývá objekt.

Při vývoji tříd jsou rozhodující jejich atributy a operace. Atributy definují strukturu třídy, operace definují její chování.

[8] OMG je sdružení dodavatelů IT, založené za účelem definování přenosného a interoperabilního objektového modelu s metodami a daty, které fungují na všech typech vývojových prostředí a na všech typech platforem.

[9] https://www.omg.org/spec/UML/2.5.1/About-UML/

Další klíčovou vlastnosti OOP je zapouzdření, zajišťující bezpečnost a ochranu dat. Zapouzdření umožňuje skrytí vnitřní implementace třídy před vnějším světem. S okolím třída komunikuje pouze pomocí veřejného rozhraní.

Dalším důležitým pojmem OOP je dědičnost. Stejně jako lidé dědí majetek po svých předcích, třídy dědí vlastnosti a chování od své rodičovské třídy.

Ve shrnutí důležitých vlastností OOP nemůžeme pominout polymorfismus. Jedná se o vlastnost umožňující používat jednotné rozhraní při práci s různými typy objektů, přičemž pro různé objekty může být implementace rozhraní různá. Třídy sdílející stejné rozhraní tak mají stejné sady metod (což je klíčové pro vnějšího uživatele tříd, který může ke všem třídám implementující totéž rozhraní přistupovat jednotně), ale tělo příslušné metody může být u každé třídy různé.

3.1.5 Modelování informačního systému

V této kapitole shrneme druhy modelů a diagramů, které v této práci používáme a jejichž vlastnosti následně popíšeme v úrovni podrobnosti odpovídající praktické části práce.

Původní autoři UML člení ve své stěžejní publikaci (Booch, 2005) modely na strukturní, behaviorální (v českém jazyce často označované jako modely chování) a architekturní, tohoto členění se budeme držet i v této práci, byť se lze setkat i s jiným členěním (Vrana, 2008, s. 8).

3.1.5.1 Strukturní model

Strukturní model slouží k vizualizaci a specifikaci statických aspektů systému. Strukturní diagramy UML zobrazují prvky systému, které jsou nezávislé na čase a které vyjadřují koncepty systému a jejich vzájemný vztah. Strukturální model nikdy nepopisuje dynamické chování systému. Prvky v těchto diagramech jsou často pojmenovány podstatnými jmény v přirozeném jazyce a jsou navzájem spojeny strukturními nebo sémantickými vztahy. Například strukturní model systému po správu odpadového

hospodářství obce může obsahovat prvky, jako je odpadová nádoba, tarif, nemovitost či vyúčtování, a dále konektory propojující tyto prvky. Zkušení modeláři na těchto diagramech také ukáží vztahy k prvkům chování.

Diagramy zahrnuté do **strukturního modelu** jsou[10]:

- *diagram tříd* (class diagram): znázorňuje třídy systému, jejich atributy, operace a vazby mezi třídami;
- *diagram komponent* (component diagram): znázorňuje logické a fyzické komponenty, z nichž se systém skládá, jejich organizaci a vzájemné vazby;
- *diagram složené struktury* (composite structure diagram): popisuje vnitřní strukturu nějaké třídy a vazby na její okolí;
- *diagram objektů* (object diagram): kopíruje strukturu diagramu tříd, přičemž ale zachycuje informace o konkrétních instancích objektů;
- *diagram nasazení* (deployment diagram): popisuje fyzickou topologii informačního systému, zahrnující klíčové hardwarové prvky (obvykle servery) a logické artefakty na nich běžící.

Nejpoužívanějším strukturním diagramem je bezesporu diagram tříd. Ostatní diagramy se při analýze a návrhu informačních systémů používají výrazně řidčeji, jak dosvědčuje i autorova dlouholetá odborná praxe.

3.1.5.2 Model chování

Model chování popisuje dynamiku informačního systému. Diagramy modelu chování zobrazují prvky systému, které se mění v čase a které zprostředkovávají dynamické koncepty systému a jejich vzájemný vztah. Prvky v těchto diagramech se označují často slovesy v přirozeném jazyce a vztahy, které je spojují, obvykle vyjadřují plynutí času.

[10] Definice vycházejí zejména z Boocha, ale i v další podpůrné literatuře jsou diagramy definovány obdobně.

Například diagram chování systému odpadového hospodářství může obsahovat prvky, jako je *Zapsat tarif k nádobě, Vystavit vyúčtování* či *Nahrát sestavu s výsypy.*

Mezi diagramy modelu chování řadíme:

- *diagram případů užití* (use case diagram): dekomponuje systém z pohledu uživatele na jednotlivé tzv. případy užití, reprezentující ucelenou součást systému, která z pohledu koncového uživatele (aktéra) plní určitý cíl,
- *diagram aktivit* (activity diagram): znázorňuje tok procesů určité ucelené části systému,
- *stavový diagram* (state machine diagram): znázorňuje, jak se určitý prvek může pohybovat mezi stavy.
- *sekvenční diagram* (sequence diagram): strukturovaná reprezentace chování určité části systému jako série sekvenčních kroků v průběhu času.

3.1.5.3 Metodika tvorby diagramů

Jak je popsáno výše, UML je modelovací jazyk. Nejedná se o metodiku a samotná specifikace UML tedy nijak neurčuje, jakými postupy by měly jednotlivé diagramy vzniknout a v jakém pořadí. K tomu je zapotřebí použití odpovídající metodiky.

Metodika tvorby softwarového vybavení definuje při vývoji softwaru otázky „kdo, co, kdy a jak“ (Arlow, 2007, s. 53). Obecně jde o popis procesu, jakým jsou uživatelské požadavky transformovány do podoby softwarového produktu.

Bruckner (2012, s. 112 - 119) popisuje v souvislosti s tvorbou informačních systémů dvě základní metodiky[11]:

- Rational Unified Process (RUP) z dílny již zmiňované firmy Rational Corporation a její zobecněná otevřená verze Unified Process (UP). Metodika byla

[11] Ve skutečnosti zmiňuje čtyři. Vedle RUP a agilního programování jsou v publikaci uvedeny ještě metodiky Microsoft Solutions Framework (MSF) a Multidimensional Management and Development of Information Systems (MMDIS). MSF však ve skutečnosti není v pravém slova smyslu metodika, jde spíše o soubor principů, konceptů a doporučení. Zejména však jde o produkt, který Microsoft již mnoho let

vyvíjena do určité míry přímo s ohledem na UML. Její klíčovou myšlenkou je iterativní vývoj a metodika klade vysoký důraz na elaborační fázi, v níž vzniká analýza a návrh systému.

- Agilní metodiky, které se soustřeďují na kontinuální dodávky nových verzí software v krátkých, obvykle dvou- až třítýdenních intervalech a které vycházejí z poznání, že konvenční metodiky vývoje software často u větších projektů selhávají.

UML je obvykle dávno do souvislosti spíše s konvenční metodikou UP/RUP, mimo jiné též z důvodu personální propojenosti (Ivar Jacobson), která přispěla k vysoké kompatibilitě a návaznostem jednoho na druhé (například RUP respektuje notaci UML, pracuje s termíny jako případ užití apod.). Naopak agilní metodiky se ze své podstaty méně soustředí na projekční fázi vývoje software, a v jejich „genetické výbavě“ tak není pro rozsáhlou analytickou fázi velký prostor. To samozřejmě neznamená, že by při agilním programování nebylo možné vytvářet různé modely či diagramy popisující specifické chování systému, v letité odborné praxi autora práce se tak ale děje spíše ve vybraných konkrétních případech: Například vznikne třídový model vybrané oblasti zájmu, u nějž je to z nějakého důvodu vhodné a potřebné, ale třídový model už nevznikne pro systém jako celek. Tento přístup koresponduje i s klíčovou prioritou agilního programování vtělenou do tzv. Manifestu agilního programování: „Fungující software (má prioritu) před vyčerpávající dokumentací“[12] Základním nositelem informací o podobě informačního systému je v agilním vývoji obvykle přímo zdrojový kód (Kadlec, 2004, s. 134). Agilní metodiky mají velmi „neortodoxní“ vztah k formalitám a dokumentům. Například extrémní programování jako jedna z „krystalických“ agilních metodik nedefinuje žádné konkrétní dokumenty, které by ve fázi analýzy a návrhu měly vzniknout, s výjimkou uživatelských příběhů (user stories)

nepodporuje. Druhá zmiňovaná metodika MMDIS je navržena a vyvíjena Fakultou informatiky a statistiky Vysoké školy ekonomické v Praze (VŠE). Při veškerém respektu k práci autorského týmu jde o metodiku, která nemá mimo území České republiky relevantní využití, a proto se jí autor v této práci dále nezabývá.

[12] https://agilemanifesto.org/iso/cs/manifesto.html

a projektové nástěnky. Obdobný vztah k dokumentaci je definován i v dnes hojně rozšířené agilní metodice Scrum (Kadlec, 2004, s. 138 a 152).

Zaměření této práce, jejímž cílem je provést analýzu a návrh informačního systému s pomocí UML, ale nikoliv tento informační systém vyvinout, tak lépe odpovídá metodika RUP, která aktivity „analýza“ a „návrh“ soustřeďuje víceméně celé do fáze „rozpracování“, před zahájením samotné konstrukce systému (Arlow, 2007, s. 63).[13] To samozřejmě neznamená, že v průběhu konstrukce není možné již zpětně zasáhnout do vytvořených modelů, vychází se ale z předpokladu, že základní stavební analyticko-návrhové „kameny“ byly postaveny již v elaborační fázi.

Pokud byl tedy učiněn závěr, že analýza a návrh informačního systému pomocí UML více odpovídá metodice RUP než agilním metodikám[14], je třeba zodpovědět navazující otázku, jaká vodítka RUP pro tvorbu UML diagramů vlastně dává.

Ačkoliv, jak již bylo uvedeno výše v této kapitole, představují RUP a UML jisté spojené nádoby se synergickým efektem, neboť oba standardy vznikaly do značné míry současně a s předpokladem společného využití, lze v RUP nalézt jen málo návodů pro způsob, jakým jednotlivé diagramy vytvořit. Přesto lze formulovat alespoň základní premisy:

1. Klíčovou úvodní fázi je sběr požadavků od zákazníka, projekt má být řízen skrze případy užití (Booch, 2005, s. 34).
2. Analytické modely předcházejí návrhovým (to vyplývá ze samotného principu UML, neboť návrhové diagramy jsou založeny na analytických).

Jak uvádí Booch, „řízení pomocí případů užití znamená, že případy užití jsou použity jako primární artefakty pro dosažení požadovaného chování systému, pro kontrolu a validaci

[13] Toto konstatování činí autor navzdory tomu, že je jinak velkým příznivcem agilního programování.

[14] V praxi by si analytik takovou otázku stěží položil, protože metodika vývoje se zpravidla nepodřizuje modelovacímu jazyku, ale spíše naopak.

architektury systému, pro testování a pro komunikaci se zúčastněnými stranami (stakeholders) projektu“.

Naproti tomu Vrana (2008, s. 8 aj.) klade na první místo diagram tříd, což zdůvodňuje tvrzením, že „systém nejlépe pochopíme, když nejdříve vyšetříme jeho statickou strukturu“ (s. 35). Naopak diagram případů užití uvádí Vrana v sekvenci diagramů až téměř na konci. Současně ale zdůrazňuje (s. 78), že „diagramy nelze tvořit v pevném pořadí, nejedná se o mechanický proces. Každý projekt vyžaduje individuální přístup“.

I autor této práce ve své odborné praxi často začíná modelováním diagramu tříd, byť za rozšířenější přístup lze, v souladu s metodikou RUP (která, jak je třeba znovu zdůraznit, není součástí UML) považovat „řízení pomocí případů užití“ (viz výše). Oběma přístupům by ale měl v každém případě předcházet sběr požadavků od zákazníka (koncového uživatele). Zda po sběru a systematickému popisu požadavků bude analytik pokračovat tvorbou diagramu případů užití či diagramu tříd, jíž pak není příliš podstatné, ostatně analýza aplikace tak jako tak není sekvenční činnost a tvorba jednotlivých diagramů se v průběhu času navzájem prolíná a zpětně ovlivňuje, přičemž tvorba jednotlivých diagramů je iterativní.

3.1.6 Nástroje pro modelování v UML

Jedním z klíčových rozhodnutí, které musí analytik před zahájením analýzy a návrhu systému učinit, je volba vhodného nástroje. Nástroje pro analýzu a návrh informačního systému se souhrnně označují jako CASE nástroje, kde zkratka CASE znamená „Computer Aided Software Engineering“, čili počítačem podporované softwarové inženýrství. UML je navržen přímo za tím účelem, aby mohl být v CASE nástrojích implementován. Smyslem CASE nástrojů je snížení času a nákladů na vývoj softwaru a zvýšení kvality vyvíjených systémů. Rozsáhlé softwarové systémy se bez podpory CASE nástrojů neobejdou (Arlow, 2007, s. 28).

CASE nástroj může být využit při všech fázích analýzy a návrhu, počínaje sběrem požadavku a konče detailním návrhem, kdy některé nástroje umožňují z modelu přímo generovat skeletony zdrojového kódu.

Většina klasifikací nástrojů CASE začíná zvážením, zda je nástroj tzv. Upper CASE, Lower CASE nebo Integrated CASE (Krishnamurthy, 1995). Upper CASE nástroj (front-end CASE) poskytuje podporu pro rané fáze životního cyklu vývoje systémů, jako je analýza požadavků a návrh. Lower CASE (back-end CASE) poskytuje podporu pro pozdější fáze životního cyklu, jako je generování kódu a testování. Integrované nástroje CASE podporují ranou i pozdější fázi. Další klasifikace obvykle uvádějí, které další funkce jsou nástrojem podporovány, jako jsou diagramy datových toků, datové modely atd.

CASE nástrojů s podporou UML je na trhu velké množství a volba správného nástroje závisí na celé řadě faktorů:

1. Na první místo si autor dovolí položit faktor korporátního standardu. Ve středních a větších IT společnostech je zpravidla nemyslitelné, aby si každý projektový tým volil sám vhodný CASE nástroj, neboť ten je určen již na úrovni firmy jako celku.
2. Odhlédneme-li od předchozího bodu, lze za další klíčový faktor označit rozsah, v jakém má být CASE nástroj použit. Autor se ve své praxi setkal s projekty, u nichž jediným vzniklým diagramem byl například diagram případů užití. Na jiných projektech se naopak používá široká škála diagramů a UML model slouží též jako dokumentace systému, byť zde už i současné nástroje narážejí na určité limity: Jedním z typických problémů je slabá podpora tvorby specifikací, kdy CASE nástroje obvykle umožňují tvorbu pouze prostého textu (Arlow, 2007, s. 535).
3. Důležitým faktorem je podpora otevřených standardů – zejména tedy samotného UML, ale rovněž formátu XMI – XML Metadata Interchange. Jde o formát standardizovaný sdružením OMG, který slouží k výměně metadat s pomocí jazyka XML. Ačkoliv jde o obecný standard použitelný pro různé metajazyky, jeho základním smyslem je podpora pro ukládání informací reprezentovaných v jazyce UML.[15] Význam podpory XMI spočívá

[15] https://www.omg.org/spec/XMI/

v možnosti vyměnit jeden nástroj za jiný, například pokud přestane pro projekt vyhovovat, skončí podpora dodavatele apod.

4. V neposlední řadě hraje roli též cena produktu.

Mezi nejpoužívanější CASE nástroje patří Enterprise Architect, MagicDraw, LucidChart či Microsoft Visio[16]. Diagramy v této práci jsou vytvořeny v aplikaci Enterprise Architect, kterou autor preferuje zejména pro její kvalitní a propracovanou podporu UML 2.5 a jejíž licencí disponuje pro potřeby své hlavní pracovní činnosti a má oprávnění jí využít i pro potřeby této práce. (Modeláři bez této možnosti využijí pravděpodobně některý z bezplatných UML nástrojů, jako je StarUML, Gliffy apod., jejichž možnosti jsou však místy značně limitující.) S pomocí aplikace Enterprise Architect byly zhotoveny rovněž všechny ilustrační diagramy použité v následujících kapitolách (z větší části jsou anglicky, neboť autor využil v nástroji funkci pro „předgenerování“ diagramů podle zadaných parametrů).

Autor v počáteční fázi tvorby závěrečné práce zvažoval též použití aplikace Microsoft Visio, a to zejména pro její kvalitní, propracované a intuitivní uživatelské rozhraní, jakož i silnou integraci s dalšími produkty rodiny Microsoft Office. Jako klíčový nedostatek aplikace Microsoft Visio se ale (přinejmenším ve vztahu k UML modelování) ukázala nemožnost znovupoužití téhož objektu ve více diagramech – pokud například uživatel vytvoří v jednom diagramu třídu, nemůže ji v jiném diagramu přímo použít, lze ji pouze zkopírovat, což výrazně komplikuje následnou údržbu modelu a zvyšuje riziko nekonzistencí. Tento nedostatek lze, dovedeno do důsledků, obejít pouze tak, že celá logika aplikace bude namodelována v jediném diagramu, což je v rozporu se zásadou srozumitelnosti diagramu[17]. Za určitou nevýhodu Visia lze označit i skutečnost, že je jednostranně zaměřeno na vizuální stránku diagramů a nenajde tedy pravděpodobně uplatnění v projektech, kde jsou požadovány techničtější funkce, jako je automatické generování kódu. Modely vytvořené v rámci této práce v aplikaci Enterprise Architect sice

[16] Například https://www.ionos.com/digitalguide/websites/web-development/the-best-uml-tools/

[17] Například https://bellekens.com/2012/02/21/uml-best-practice-5-rules-for-better-uml-diagrams/

nejsou přímo určeny pro automatické generování kódu, po určitých úpravách by však tímto způsobem mohly být použity.

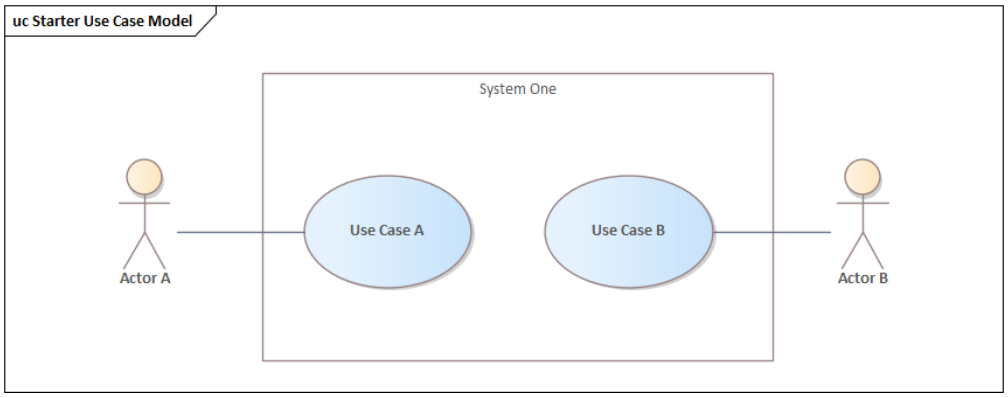
3.1.7 Specifikace požadavků

Jazyk UML neposkytuje žádné doporučení, které by se týkalo způsobu zachycování požadavků zadavatele systému (obvykle zákazníka). S požadavky se vypořádává výhradně prostřednictvím mechanismu případů užití (kterým se věnuje následující kapitola 3.1.8), případně dalšími druhy modelů (Booch, 2005, s. 450). Přesto se však mnoho analytiků a návrhářů domnívá, že případy užití samy nestačí a že jejich tvorbě by měl předcházet sběr a specifikace požadavků (Arlow, 2007, s. 79 - 80). Proto i v této práci, v její praktické části, započne autor analytickou činnost formulací požadavků. Vzhledem k charakteru a zaměření práce se autor omezuje na požadavky funkční (co má výsledný systém dělat) a nezabývá se požadavky nefunkčními (omezujícími či upřesňujícími způsob, jakým má být cílový systém implementován, ať už jde o požadavky na intuitivnost ovládání, výkon, spolehlivost apod.). V souladu s řadou odborných doporučení (například Buchalcevová, 2013, s. 78 a násl.) jsou požadavky uváděny ve formě tabulky s následujícími atributy:

- identifikátor požadavku,
- název požadavku,
- popis požadavku,
- priorita požadavku, u níž je dána přednost mezinárodně ustálené konvenci M – Must have, S – Should have, C – Could have a W – Want to have (Arlow, 2007, s. 82).

3.1.8 Diagram případů užití

Diagram případů užití popisuje chování systému ve vztahu k vnějším uživatelům – aktérům. Případ užití je v tomto kontextu ucelenou funkčností, kterou systém poskytuje při komunikaci s aktérem (Vrana, 2008, s. 58). Aktérem je pak vnější uživatel systému. Aktér komunikuje se systémem, ale není jeho součástí. Aktérem je typicky osoba používající systém, ale může jím být rovněž například jiný systém.



Obrázek 1: Jednoduchý diagram případů užití se dvěma aktéry a dvěma případy užití

Modelování případů užití je poměrně populární technikou objektově orientované analýzy, často jde o první model, které v rámci analýzy a návrhu informačního systému vzniká, neboť „případy užití jsou cenným nástrojem, který nám pomůže lépe porozumět funkčním požadavkům na systém. První průchod případy užití by měl být proveden záhy po začátku tvorby“ (Fowler, 2009, s. 108). Případy užití považuje za klíčové artefakty analýzy i metodika UP, která jejich nalezení a strukturování řadí do úvodní fáze sběru požadavků (Arlow, 2007, s. 77). Ostatně UP je přímo označován za „use-case driven“ metodiku.

Velkou výhodou případů užití (jsou-li namodelovány správně) je jejich relativně dobrá srozumitelnost pro budoucí uživatele. Z uvedeného důvodu se často používají k odsouhlasení budoucích funkčností systému.

Každý případ užití tedy popisuje jeden ze způsobů použití informačního systému, popisuje jednu jeho požadovanou funkčnost. Zastřešujícím prvkem je samotný diagram případů užití, který znázorňuje jednotlivé případy užití, aktéry a jejich vazby. Na diagram pak navazuje sada scénářů (specifikací) případů užití, které formou sekvence kroků popisují interakci mezi aktérem (uživatelem) a systémem.

Zároveň jde však o disciplínu, ve které se často chybuje. Mezi typické chyby patří nesprávné používání stereotypů extend a include[18] nebo rozvláčné a stále se opakující pasáže ve specifikacích případů užití, které v důsledku zatemňují podstatu funkčnosti.

(V této souvislosti je potřebné vysvětlit termín „stereotyp“ ve vztahu k UML. Stereotyp je prvkem modelu, který identifikuje účel ostatních prvků modelu. UML poskytuje standardní sadu stereotypů pro určité účely, jako je třeba právě typ vazby mezi případy užití. Další stereotypy může vytvářet modelář pro specifické potřeby.)

Častým jevem, který se objevil zejména s nástupem webových aplikací, je zaměňování či chybné ztotožnění případu užití s jednou webovou stránkou, k němuž dochází v analytické fázi označované jako *vyhledávání případů užití.* Je tomu tak zejména proto, že často skutečně platí, že jeden případ užití je realizován právě jednou stránkou (budeme-li se držet tématu práce, tak to může být například „Vypiš kartu občana“), ale v řadě případů může být jeden případ užití realizován více stránkami (například případ užití „Proveď platbu za odpad“ může být realizován v podobě několika navazujících stránek sdružených do průvodce) anebo naopak na jedné stránce se může vyskytovat více případů užití („Vypiš základní informace o občanovi“ a na základě stisknutí tlačítka na stránce „Vypiš podrobné informace o občanovi“, přičemž podrobné informace se ze serveru načtou pomocí AJAX, aniž by se volala jiná stránka).

Na další častou chybu upozorňuje Kraval (2010, s. 102): „Zde musím zdůraznit jednu opravdu velmi důležitou skutečnost, se kterou mají problémy zejména programátoři, kteří se podílejí na analytických pracích. Případ užití není totéž, co spuštěná funkce. Základním prvkem pro identifikaci případu užití je požadavek jako nutkání použít informační systém a z něj se odvíjí odhalení a definice daného případu užití. Případ užití odhalujeme na základě sekvence děje: Nikdo nic nepotřebuje, venku se něco odehraje, nastane událost a požadavek na to, aby se použil systém. Tento požadavek neboli nutkání použít systém je obrazem

[18] Například https://www.modernanalyst.com/Community/CommunityBlog/tabid/182/ID/3436/Top-10-mistakes-in-Use-Case-Modelling.aspx

nalezeného případu užití, který vznikl právě proto, aby tento požadavek užitku (požadavku na užitek) splnil.“

Kraval dále v textu popisuje způsoby, jakými by se případy užití měly identifikovat. Popisuje základní dva přístupy:

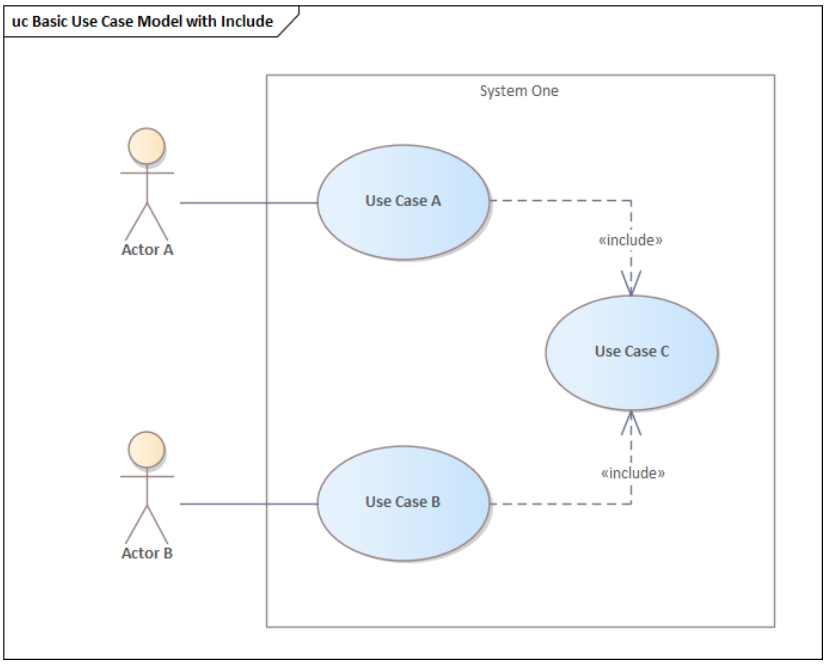
- Aktérová škola: „Najděme nejprve prvky z okolí systému (tzv. aktéry neboli prvky typu Actor), které budou potřebovat a používat systém, případně najděme ty prvky z okolí, které budou komunikovat se systémem, resp. ty prvky z okolí, kterým bude systém předávat informace. Až nějaký takový prvek (neboli prvek typu Actor) z okolí najdeme, zeptejme se, proč a nač aktér tento systém potřebuje, resp. jak jej použije, resp. jak komunikuje. Tímto najdeme všechny případy užití.“ Autor pak tuto metodu označuje za nepříliš účinnou a zatíženou řadou chyb, neboť dle jeho názoru vede k soustředění na aktéry (například jejich správné pojmenování) a nikoliv na jejich činnosti. Aktérovou školu následně doporučuje spíše jako doplňkový postup. Aktérovou školu naopak popisuje (avšak aniž by byla takto přímo pojmenována), víceméně jako jediný doporučený postup i Arlow (2007, s. 91) a obecně, i z hlediska autorovy osobní zkušenosti, se jedná o metodicky nejčastěji doporučovaný postup.
- Procesní škola: „Najděme všechny procesy podniku v okolí, které vedou k použití systému, následně tak najdeme případy užití vyvolané těmito procesy podniku.“ Tuto metodu Kraval preferuje (a autor této práce rovněž), neboť se analytická činnost soustředí na funkcionalitu, a nikoliv na aktéra, který pak z modelu do určité míry vyplyne.

Ve vztahu k tématu této práce tak lze například rozhovorem se zadavatelem identifikovat, že jedním z požadavků na systém je zápis úhrady za svoz odpadů. identifikovat například případ užití „Zapsat úhradu za svoz odpadů“. Jedná se zjevně o mnohem důležitější poznatek, než že tuto činnost má provádět referent na podatelně obecního úřadu.

Dalším častým problémem při návrhu diagramu případů užití je správné použití stereotypů *extend* a *include*. Tyto stereotypy vyjadřují typ vazby mezi případy užití.

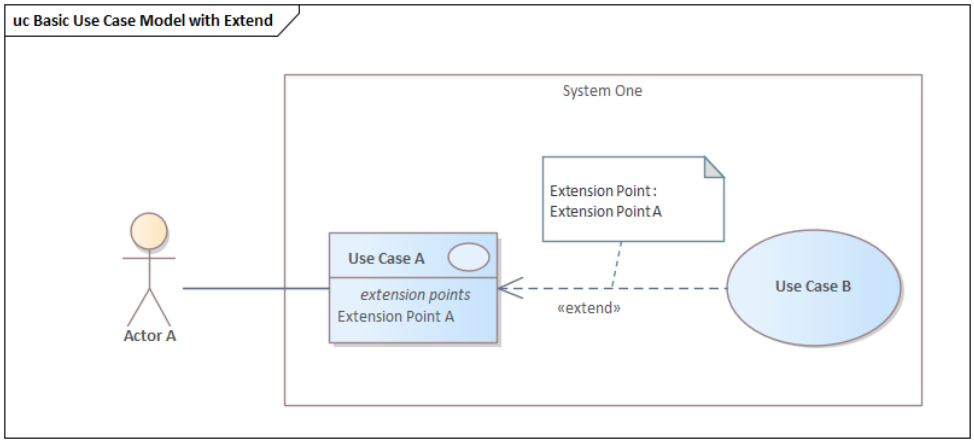
Jednodušší je porozumění vazbě *include*, která podporuje znovupoužití společných funkcionalit jejich vyčleněním do samostatných případů užití. Rizikem jejího použití je zejména přílišnou fragmentací diagramu a tím jeho znepřehledněním. Například Arlow prezentuje správné použití vazby *include* na případech užití „Změnit údaje o zaměstnanci“, „Prohlížet údaje o zaměstnanci“ a „Vymazat údaje o zaměstnanci“ vyčleněním případu užití „Najít údaje o zaměstnanci“ a jeho následným propojením vazbu include se třemi zmiňovanými (Arlow, 2007, s. 122). Formálně jde jistě o správný příklad[19], zároveň však může svádět k tomu, že modelář popsaným způsobem začne vyčleňovat zcela triviální části funkcionality, a tím výrazně zhorší čitelnost diagramu. Běžnému čtenáři totiž musí být i bez vazby include zcela zjevné, že například při změně údajů o zaměstnanci je nezbytné nejprve zaměstnance vyhledat v databázi. Takovýchto triviálních operací, které se neustále opakují v mnoha případech užití, je často velké množství. Je samozřejmě potřebné vyhledání zaměstnance v rámci tvorby zadání informačního systému někde popsat, ale to ještě nutně neznamená, že taková funkcionalita musí být reprezentována samostatným případem užití. A i v případě, že se analytik pro tento přístup rozhodne, je vhodné (s pomocí vhodného CASE nástroje) tyto „includované“ případy užití užívat spíše ve vztahu k návrhářům a programátorům (neboť ti z něj dobře odvodí, kde se nachází znovupoužitelnost), než vůči zákazníkovi, který se v diagramech plných vazeb *include* orientuje obvykle velmi špatně.

[19] Záleží ale pochopitelně na dalších faktorech a celkovém uspořádání a filozofii vznikajícího informačního systému. Vůbec nemusí platit, že případ užití „Změnit údaje o zaměstnanci“ vždy includuje případ užití „Najít údaje o zaměstnanci“, neboť typicky se změna údajů určitých kmenových údajů provádí sekvencí kroků: 1) najít záznam, 2) zobrazit záznam, 3) změnit záznam, přičemž ve třetím kroku se již záznam znovu nevyhledává, protože změna se vztahuje k již zobrazenému (a tedy dříve vyhledanému) záznamu. Nehledě na to, že i samotné zobrazení záznamu nemusí vyžadovat jeho předchozí vyhledání, pokud například má uživatel uložen daný záznam v „oblíbených položkách“ apod. Při modelování případů užití je tak potřebné sledovat hlavní cíl – identifikovat a namodelovat případy užití, tedy entity, které slouží k dosažení nějakého cíle pro koncového uživatele, a méně se zabývat implementačními detaily, které patří až do navazujících fází analýzy a návrhu.



Obrázek 2: Ukázka diagramu případů užití s využitím vazby include

Ještě složitější bývá správné porozumění vazbě extend. Ta totiž, jak upozorňuje například Kraval (2010, s. 118), neslouží k podmíněnému připojení další funkcionality, ale k volitelnému rozšíření, přičemž klíčovým předpokladem je, že původní bázový případ užití musí mít schopnost „být použit“ i bez rozšíření. Z tohoto důvodu je propojovací šipka směřována od rozšiřujícího případu užití směrem k rozšiřovanému.



Obrázek 3: Ukázka diagramu případů užití s využitím vazby extend

Na nalezení případů užití pak navazuje tvorba jejich specifikace, tedy scénářů. Zde je třeba říci, že jazyk UML neobsahuje pro specifikace případů užití žádnou formální strukturu. Autoři UML vycházeli z předpokladu, že veškerá specifikace bude vytvořena v podobě modelů, přičemž detaily případů užití budou popisovány formu diagramů aktivit či sekvenčních diagramů. To je samozřejmě možné, dle autorovy zkušenosti však zákazníci často potřebují k porozumění problematice i psaný text. Shodně s Kravalem (Kraval, 2010, s. 112) tak autor preferuje tvorbu sice stručných, ale přesto textových specifikací, které jsou o diagramy aktivit či sekvenční diagramy doplněny pouze tam, kde to v daném kontextu dává smysl (2008, s. 61). Například k výše uvedenému identifikovanému případu užití „Zapsat úhradu za svoz odpadů“ pravděpodobně nebude zapotřebí navrhovat samostatný diagram aktivit. Jinak tomu může být například u případu užití „Zapiš nového obyvatele“, kdy v návaznosti na zápis nového obyvatele do systému může být potřebné vykonat celou řadu navazujících aktivit, typicky propsání záznamu do různých informačních systémů veřejné správy. Tyto kroky již může být vhodné namodelovat pomocí diagramu aktivit (viz kapitola 3.1.10) a/nebo sekvenčního diagramu (viz kapitola 3.1.11).

3.1.9 Diagram tříd

Druhou klíčovou analytickou disciplínou, přímo navazující na tvorbu modelu případů užití, či s ní spíše úzce provázanou a prolínající se, je návrh diagramu tříd. Prakticky všechny odborné publikace, které komplexněji popisují problematiku UML a ze kterých autor čerpá v této práci (Booch, Page-Jones, Vrana, Kraval), popisují tvorbu diagramu tříd na prvním místě, výjimku zde tvoří pouze Arlow. (Což nevylučuje, že řada publikací i samotná metodika UP předpokládá tvorbu diagramu případů užití jako první činnost, viz kapitola 3.1.8.) I autor práce často diagramem tříd začíná, avšak spíše pouze v případech, kdy jsou již rozsah budoucího informačního systému a jeho hranice do značné míry vymezeny buď předchozím studiem příslušné business domény nebo například zadávací dokumentací. Obecně by návrh případů užití, alespoň v první verzi (bez nároku na naprostou úplnost), měl diagramům tříd předcházet, neboť v opačném případě by se mohlo snadno stát, že v diagramu tříd budou modelovány třídy, které ve skutečnosti vůbec nejsou pro informační systém potřebné. Současně diagram tříd představuje úroveň abstrakce, které

někteří zákazníci již nejsou schopni příliš porozumět, a je tedy potřebné specifikaci zadání opřít spíše o případy užití.

Třídy jsou nejdůležitějším stavebním kamenem každého objektově-orientovaného systému (Booch, 2005, s. 47). Třída je společným popisem sady objektů, které sdílejí tytéž atributy, operace, relace a sémantiku. Diagram tříd znázorňuje třídy a jejich vazby a je klíčovou spojnicí mezi požadavky na informační systém a jeho vlastní implementací, neboť třídy namodelované v diagramech tříd mají často přímý obraz v systému v podobě skutečných tříd implementovaných v programovacím jazyce či v databázi.

Poslední věta předcházejícího odstavce může být zároveň můstkem k upozornění na častý negativní jev při navrhování podoby tříd a jejich vazeb. Podobně jako u diagramů případů užití, analytici bývají nezřídka zatíženi budoucí podobou informačního systému. Tím nemá být řečeno říci, že by analytik neměl při návrhu diagramů o budoucí podobě informačního systému přemýšlet, je ale třeba si zároveň hlídat, aby tento „vysněný“ obraz nezatěžoval ty analytické fáze, které mají od konkrétní implementace či podoby ještě do určité míry abstrahovat. Stejně jako u návrhu případů užití je víceméně škodlivé, pokud analytik příliš do detailu přemýšlí o podobě obrazovek, kterými budou případy užití obsluhovány, u diagramu tříd bývá často nevhodnou zátěží přílišné zaměření na relační databázi. Z tohoto důvodu je někdy doporučováno vytvářet samostatně analytický diagram tříd a návrhový diagram tříd (Arlow, 2007, s. 341).

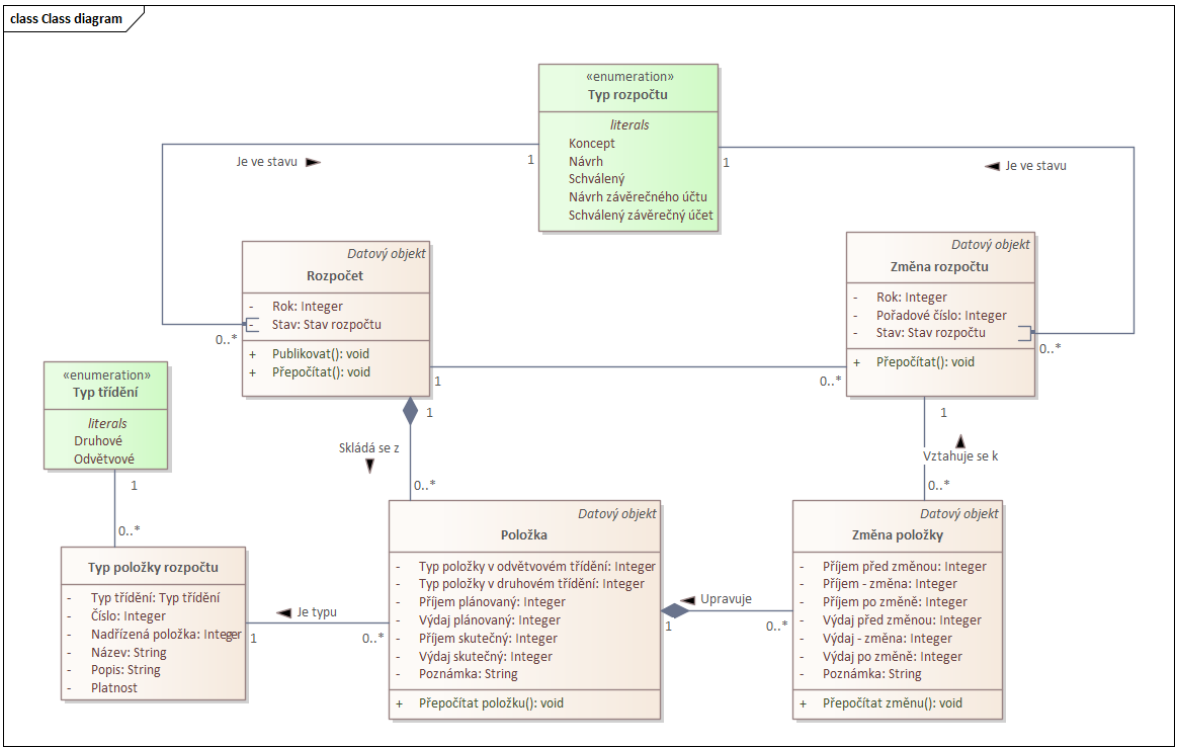
Diagram tříd se totiž v mnoha ohledech až příliš podobá databázovému modelu, což snadno svádí k jejich zaměňování a k aplikování databázových omezení na diagram tříd.

Typickým případem je použití dědičnosti. Ačkoliv konstrukce zobecnění je považována za jeden z klíčových aspektů navrhování diagramu tříd (Page-Jones, 2001, s. 100), v diagramech tříd často schází, neboť analytik (často spíše analytik se znalostí programování) samozřejmě ví, že v relačních databázích nelze dědičnost přímo namodelovat, a proto ji nepoužije ani v návrhu diagramu. Sekundárním důvodem pak může být snaha přiblížit diagram, stejně jako u případů užití, zákazníkovi, který, není-li v daném oboru vzdělán, princip dědičnosti málokdy plně chápe.

Ve skutečnosti je třeba od databázové roviny při návrhu abstrahovat, s tou si musí poradit návrhář databáze; pro transformaci objektových návrhů do podoby relační databáze je k dispozici celá řada postupů i frameworků.

To však neznamená, že se od samého počátku návrhu máme soustředit na dědičnost. Naopak, návrh diagramu tříd je třeba začít od jednodušších abstrakcí, typicky od tříd samotných a jejich základních asociací. Kraval k tomu píše: „Při vyhledávání vztahu mezi třídami v Class Diagramu zásadně začínáme vyhledáváním vztahu *Asociace*. Tento vztah je totiž jednodušší, je velmi dobře viditelný, a navíc vztah *Generalizace* nalezneme až poté, když zřetelně vidíme vztahy *Asociace*. Důležité pravidlo zní: Nehledejme nejprve vztahy *Generalizace*, ale *Asociace*." (Kraval, 2010, s. 46) Teprve následně z běžných asociací lze dále odvozovat kompozice (identifikace vazby typu celek/část) či právě generalizace.

Současně se vazbám definují multiplicity. Zde nalezneme další podstatný rozdíl vůči relační databázi: Zatímco relační databáze neumožňuje přímo namodelovat vazbu M : N a musíme si zde vypomoci pomocnou spojovací tabulkou, v diagramu tříd můžeme tuto vazbu bez problémů použít, byť v praxi je její využitelnost spíše omezená, neboť při modelování často narazíme na potřebu přidat ke spojení M : N další doplňující informace (jako je například platnost vazby), takže spojovací třídu stejně musíme namodelovat, abychom tyto dodatečné informace měli kde zachytit. Samozřejmě je též možné přijmout koncept podporovaný Arlowem, který rozlišuje analytické a návrhové třídy, kdy analytické třídy vznikají ve fázi analýzy a návrhové ve fázi návrhu, byť Arlow zdůrazňuje, že návrhové třídy často vznikají zpřesňováním analytických tříd, kdy například „je třeba koncepční analytickou třídu rozbít na několik podrobných návrhových tříd", což může být i případ vazby M : N. Rozlišování analytických a návrhových tříd vychází z filozofie Arlowovy publikace, která se nezaměřuje čistě na výklad jazyka UML, ale popisuje jej ve vazbě na metodiku UP. V této souvislosti lze doplnit, že UML samotný analytické a návrhové třídy nerozlišuje, byť i Booch jakožto jeden z three amigos popisuje koncept „pokročilých tříd" (advanced classes), které však chápe čistě ve smyslu doplňování dalších informací (zejména klasifikátorů) do již existujících tříd (Booch, 2005, s. 118).



Obrázek 4: Ukázka diagramu tříd

Obrázek 4 znázorňuje ukázkový diagram tříd. Třídy jsou v UML reprezentovány v podobě obdélníků. V záhlaví je uveden název třídy, následuje seznam atributů a následně pod oddělovací čarou seznam operací dané třídy.

Třídy jsou mezi sebou propojeny asociací (plná čára), případně jiným typem vazby (na obrázku je užita kompozice, znázorněná plným kosočtvercem). U vazeb tříd je zároveň žádoucí uvádět na obou stranách její násobnost a vztah slovně popsat, včetně indikace (ve formě plné šipky), v jakém směru je třeba vztah číst (ne vždy je to z důvodů rozvržení diagramu zleva doprava).

Tvorba diagramu tříd je dlouhodobý iterativní proces a finální úroveň podrobnosti se může velmi lišit v závislosti na tom, k čemu je dále diagram použit. Klíčovým faktorem je, zda je UML návrh použit programátorem pouze jako zadání, případně vodítko pro tvorbu zdrojového kódu, či zda jsou z návrhu přímo generovány skeletony kódu; pak musí být diagram tříd opravdu velmi podrobný a rovněž prostý všech chyb. Vzhledem k tomu, že výstupy této práce nebudou (přinejmenším ne přímo) použity ke generování zdrojových

kódů, ani jako zadání pro programátora, autor se v diagramech tříd drží „rozumné“ úrovně podrobnosti (dané jeho vlastní zkušeností), která představuje určitou „zlatou“ rovnováhu mezi srozumitelností modelu pro zákazníka a jeho použitelností pro programátora (to znamená, že pro každou business doménu je vytvořen diagram tříd pouze v jedné úrovni podrobnosti, čili není rozlišován analytický a návrhový diagram). V diagramech pak je například použita dědičnost či polymorfismus, ale například u atributů není uváděna viditelnost. U tříd jsou uváděny jen některé významnější operace.

3.1.10 Diagram aktivit

Diagramy aktivit se používají k modelování chování business procesů. Diagramy aktivit ukazují pracovní postup od počátečního bodu do koncového bodu s potřebnými podrobnostmi.

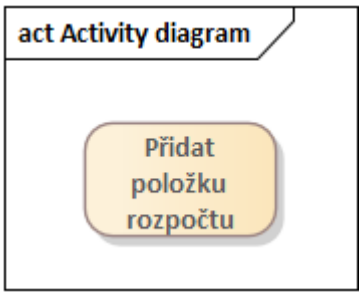
V diagramech aktivit se znázorňují primární aktivity a vztahy mezi aktivitami v daném procesu, tak jak jsou vykonávány v určité sekvenci. Na rozdíl od sekvenčního diagramu (viz kapitola 3.1.11) ale diagram aktivit umožňuje i větvení a paralelní aktivity.

Diagram aktivit je též určitou obdobou stavového diagramu (viz kapitola 3.1.12), na rozdíl od stavového diagramu se ale soustředí na aktivity a přechody mezi nimi. Vzhledem k tomu, že jsou určeny zejména ke komunikaci se zadavateli, měla by být při návrhu zachována přiměřená složitost těchto diagramů.

Jak uvádí Arlow (s. 286), diagramy aktivit lze připojit k libovolnému objektu modelovanému v rámci analýzy či návrhu a tím modelovat jeho chování. Diagramy aktivit se obvykle připojují k následujícím typům prvků:

- případy užití,
- třídy,
- rozhraní,
- komponenty,
- spolupráce,
- operace.

Ačkoliv název „Diagram aktivit“ může vyvolávat dojem, že v jednom diagramu modelujeme primárně vazby a návaznosti mezi aktivitami, není tomu tak. Správnost návrhu diagramu aktivit totiž vyžaduje důsledné odlišování termínů „aktivita“ a „akce“. Aktivitu lze považovat za „zastřešující“ objekt, pro níž navrhujeme diagram. Samotný diagram se pak skládá z uzlů, kterými jsou nejčastěji akce. Akce je dále nedělitelný krok dané aktivity.[20]

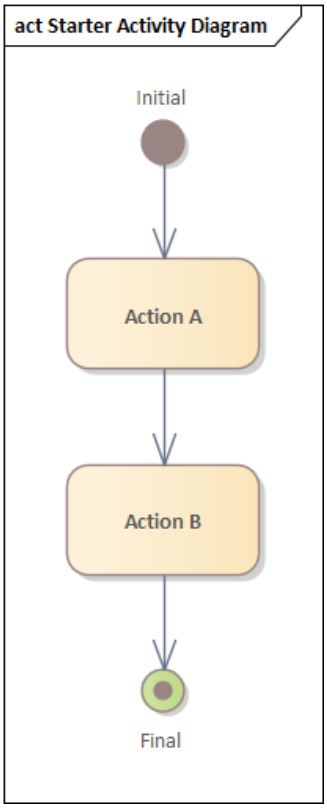


Obrázek 5: Symbol akce

Je-li zapotřebí z určitých důvodů vyjádřit návaznosti aktivit mezi sebou, je tak možné učinit jinými mechanismy, které se ale v této práci nevyužívají a nebudeme se jim tedy věnovat.

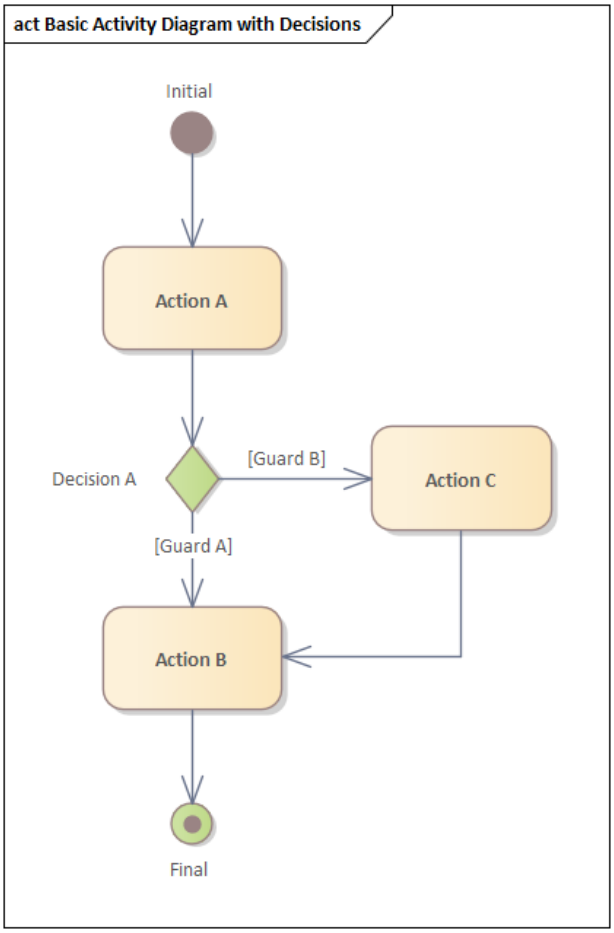
Akce jsou spojeny pomocí řídicích toků. Řídicí tok se zapisuje v podobě čáry se šipkou. V diagramech aktivit se dále znázorňují počáteční a koncové uzly.

[20] Podrobnější vysvětlení lze nalézt například na http://blok.kurzy-uml.cz/diagram-aktivit-aktivita-nebo-akce/.



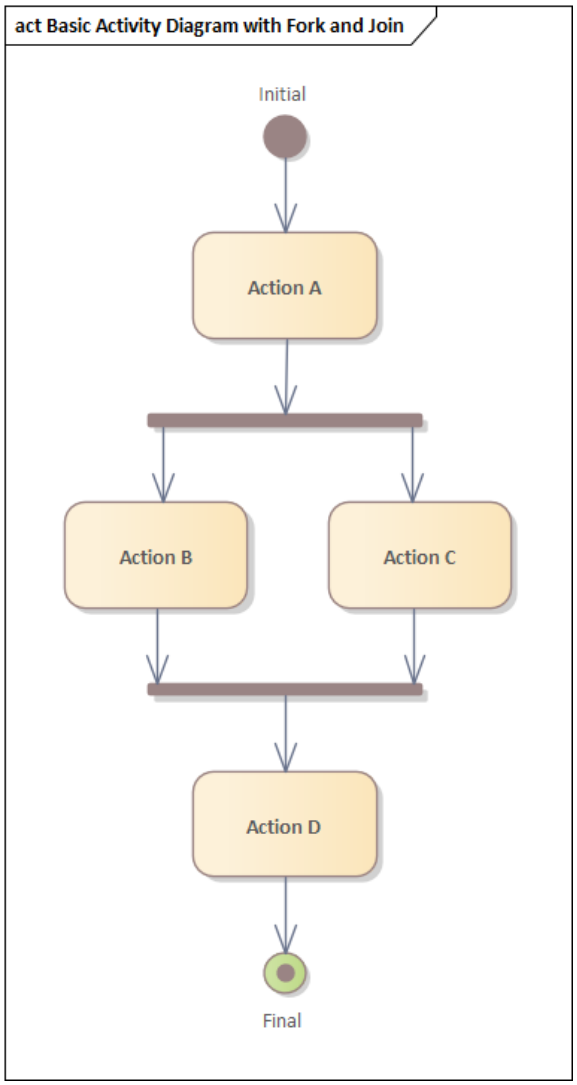
Obrázek 6: Jednoduchý diagram aktivit s počátečním a koncovým uzlem, dvěma akcemi a řídicími toky mezi nimi.

Pro větvení se používají rozhodovací a slučovací uzly. Jejich zápis je totožný: tvar kosočtverce. Řídicí toky vycházející z rozhodovacího uzlu mají stanoveny podmínky, které stanoví tok řízení, pokud je splněna podmínka.



Obrázek 7: Diagram aktivit s podmínkou

Dalším způsobem rozdělení diagramu do více větví je rozvětvení a spojení. Obě používají stejný zápis – buď vodorovný nebo svislý pruh (orientace závisí na tom, zda je řídicí tok znázorněn horizontálně nebo vertikálně).



Obrázek 8: Rozvětvení a spojení

Rozdíl mezi rozhodováním a rozvětvením spočívá v tom, že u rozhodování se řídicí tok vydá vždy jen jednou z popsaných cest (na základě vyhodnocení podmínky), zatímco větvení znamená paralelní tok oběma (všemi) větvemi.

Posledním prvkem diagramů aktivit používaným v této práci je tzv. oddíl aktivity, někdy nazývaný též plavecká dráha (Fowler, 2009, s. 119). Oddíly aktivit seskupují k sobě aktivity, které spolu navzájem souvisejí. Přesnou sémantiku rozdělení UML nespecifikuje a je plně na modeláři.

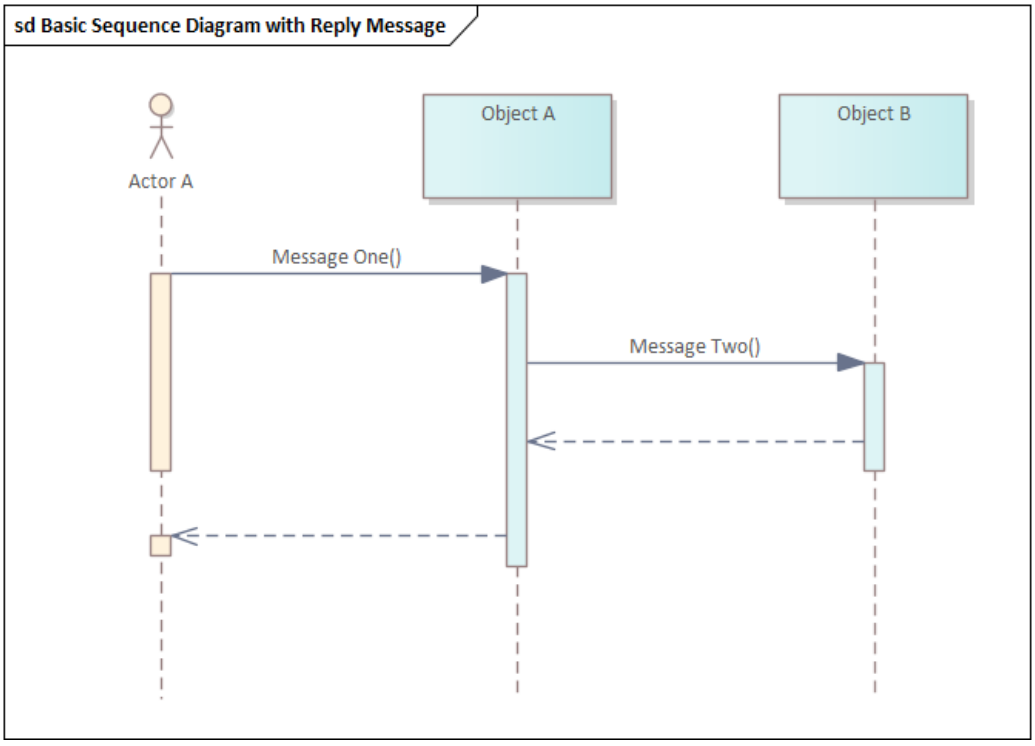
3.1.11 Sekvenční diagram

Sekvenční diagram popisuje interakce mezi třídami z hlediska výměny zpráv v průběhu času (Booch, 2005).

Sekvenční diagramy znázorňují objekty, které se účastní případu užití, a zprávy, které mezi nimi v průběhu času přecházejí. Sekvenční diagram je dynamický model, který ukazuje sekvenci zpráv, které jsou předávány mezi objekty v definované interakci. Jsou velmi užitečné pro pochopení specifikací v reálném čase a složitosti případů užití.

Sekvenční diagram může znázorňovat všechny možné scénáře případu užití, ale obvykle se vytvářejí sekvenční diagramy pro každý scénář zvlášť.

Základní prvky sekvenčního diagramu znázorňuje Obrázek 9.



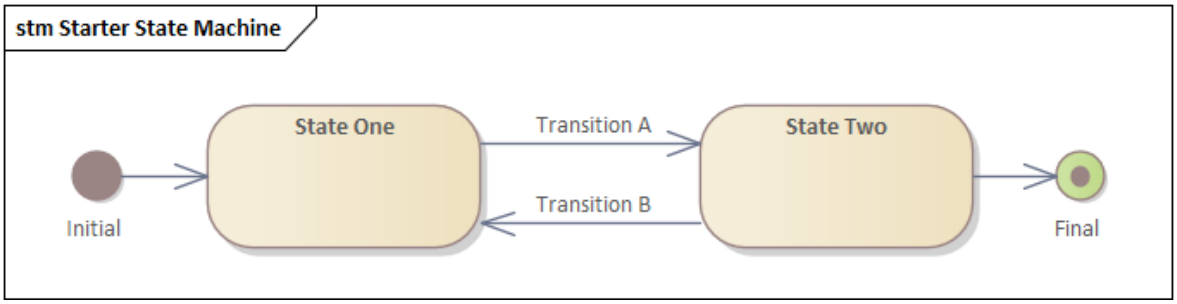
Obrázek 9: Jednoduchý sekvenční diagram

Svislé čáry se označují jako čáry života a reprezentují jednotlivé účastníky v sekvenčním diagramu. Účastník je identifikován v záhlaví čáry. Účastníkem může být aktér, objekt nebo i další prvky, jejichž použití není pro tuto práci důležité.

Mezi jednotlivými účastníky jsou předávány zprávy, reprezentované šipkami. Ty mohou být synchronní (šipka s vyplněným hrotem, viz obrázek) nebo asynchronní (obyčejná šipka). Návratové šipky se znázorňují přerušovanou čarou, nemusí se však používat vždy (Fowler, 2009, s. 69).

3.1.12 Stavový diagram

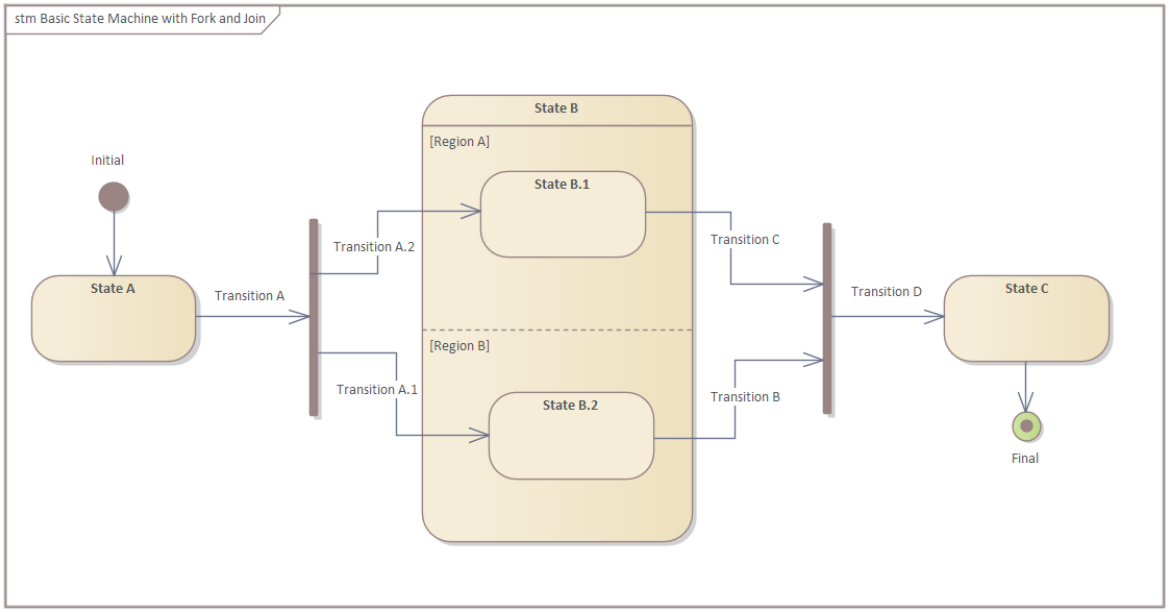
Stavové diagramy popisují dynamické chování určitého objektu, obvykle jednoho, patří tak mezi diagramy sloužící k popisu dynamiky systému. Stavový diagram specifikuje posloupnost událostí, kterými daný objekt prochází v průběhu svého životního cyklu v reakci na události. Počáteční a koncový stav jsou vyjádřeny pseudostavy. Šipky mezi stavy reprezentují přechody – změnu z jednoho stavu na jiný.



Obrázek 10: Stavový diagram zobrazující dva stavy a dva přechody a dále počáteční a koncový pseudostav.

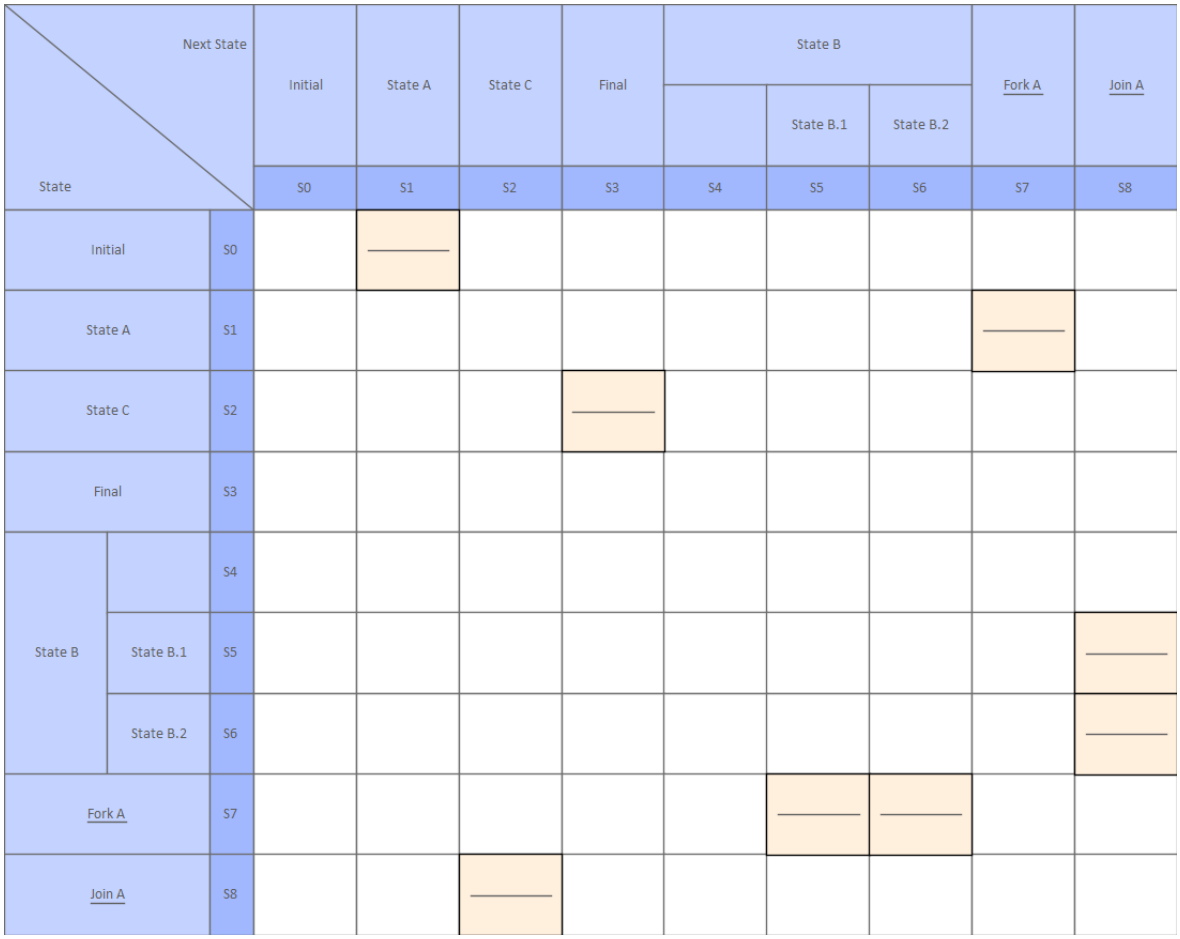
Přechod může (ale nemusí) být definován i komplexněji. Kompletní struktura přechodu se skládá ze tří částí: signatura-spouštěče [podmínka]/aktivita. Signatura je obvykle jedna událost, která spouští potenciální změnu stavu. Podmínka, je-li uvedena, vyjadřuje, že k přechodu může dojít pouze v případě splnění dané podmínky. A konečně aktivita je chování, které se vykoná během přechodu (Fowler, 2009, s. 110).

Stavový diagram, podobně jako diagram aktivit, může obsahovat větvení.



Obrázek 11: Stavový diagram s větvením

Některé UML nástroje, jako například právě v této práci používaný Enterprise Architect, umožňují i zobrazení stavového diagramu v maticové formě, kde prvky matice znázorňují povolené přechody mezi stavy (viz Obrázek 12).



Obrázek 12: Stavový diagram v maticovém zobrazení

3.1.13 Diagram balíčků

Diagram balíčků je jednoduchý typ diagramu, který vizuálně popisuje strukturu systému. Lze jej použít k popisu vztahu mezi balíčky a prvky, které obsahují.

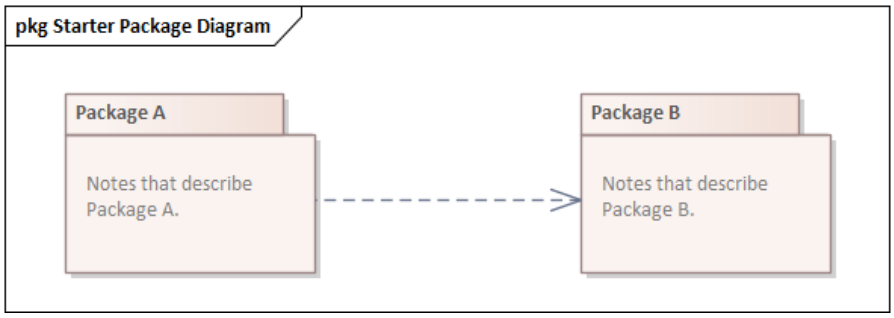
Hlavním prvkem, který je znázorněn v diagramu balíčků, je samotný balíček, který seskupuje určitou část systému – třídy, případy užití a další prvky. Balíčky mohou obsahovat jiné balíčky a tím tvořit hierarchii.

Mezi balíčky existuje řada důležitých vztahů, včetně závislostí, které znázorňují, že určitý balíček je závislý na jiném balíčku.

Diagramy balíčků často slouží k základnímu pochopení členění (dekompozice) určitého systému, podobně jako je například obsah publikace. Ve vztahu ke zdrojovému

kódu v objektově orientovaných jazycích jsou pak balíčky ekvivalentní tzv. jmenným prostorům (namespaces), které jsou přirozeným způsobem seskupování tříd s navzájem související funkcionalitou či logikou (Fowler, 2009).

Obrázek 13 znázorňuje jednoduchý diagram balíčků. Obdélníky představují jednotlivé balíčky, šipka reprezentuje závislost, v tomto případě závislost balíčku *Package A* na balíčku *Package B*.



Obrázek 13: Diagram balíčků ve své nejjednodušší podobě

Vedle toho diagramů balíčků nabízí i další prvky a notace (například pro podporu hierarchie balíčků), které tato práce nevyužívá a nejsou tedy dále popisovány.

3.2 Teoretická analýza business domény

Pro práci business analytika navrhujícího budoucí informační systém jsou potřebné dva základní typy znalostí:

- technologické znalosti zahrnující teoretické základy informatiky, teorii algoritmů, znalosti architektur a principů programování a pochopitelně důkladnou znalost disciplíny analýzy a návrhu informačního systému;
- business znalosti, tedy znalosti příslušné business domény či oboru, kterého se navrhovaný informační systém týká.

Aby tedy bylo možné provést řádnou analýzu a návrh informačního systému pro agendy malých obcí, je třeba k obecné problematice analýzy a návrhu informačního systému, jíž se věnuje předcházející kapitola, připojit druhý klíčový pilíř. Tímto druhým

stavebním kamenem je samotná business problematika, kterou má cílový systém řešit. V této druhé podkapitole teoretické části se tedy zaměříme na problematiku agend malých obcí.

3.2.1 Členění obcí

Než specifikujeme agendy malých obcí, je zapotřebí definovat samotný termín „malá obec“.

Podpůrným kritériem pro termín „malá obec“ může být počet obyvatel obce, počet nemovitostí na území obce či výměry katastrálního území obce. Alternativním kritériem může být výše ročního rozpočtu. Ta je dána rozpočtovým určením daní, ty jsou však z velké většiny odvozeny od výše uvedených kritérií – počtu obyvatel a výměry katastrálního území obce (§ 4 zákona č. 243/2000 Sb. o rozpočtovém určení daní).

Podle výše uvedeného kritéria se práce zaměřuje přibližně na obce do 3 000 obyvatel, jedná se však jen o podpůrné kritérium. Z pohledu agend vykonávaných obcí je významnější členění obcí podle výkonu tzv. přenesené působnosti. V tomto smyslu se obce rozdělují do tří skupin, tzv. stupňů (Kočí, 2012, s. 73):

- obec prvního stupně – jedná se o menší obce, na něž byl delegován pouze základní okruh přenesené působnosti, který je omezený a nepříliš rozsáhlý, a obec většinou nemá na výkon této přenesené působnosti vyčleněny příslušné zaměstnance obce (úředníky);
- obec druhého stupně – jde o obce s pověřeným obecním úřadem. Tyto obce již vykonávají relativně velkou část přenesené působnosti, a to většinou prostřednictvím profesionálních úředníků;
- obec třetího stupně – jde o obce s rozšířenou působností, které vykonávají poměrně rozsáhlý okruh přenesené působnosti, a to prostřednictvím specializovaných úředníků.

Je zjevné, že zaměřením této práce jsou obce prvního stupně. Množina těchto obcí se často kryje s kritériem „počet obyvatel do 3 000“, avšak kritérium stupně obce je exaktnější a rovněž lépe dovoluje zvolit vhodné agendy, kterými má smysl se zabývat.

Jak podotýká Kočí (2012, s. 74), „výkon přenesené působnosti je u těchto obcí velice problematický. Většinou není v možnostech malých obcí zaměstnat na výkon přenesené působnosti zaměstnance na požadované odborné úrovni a přenesenou působnost pak zajišťuje starosta obce či jiný zastupitel, což je minimálně z hlediska principů, na nichž je výkon státní správy založen (nestrannost, nezávislost, odbornost apod.) velice problematické."

Pravdivost uvedených slov může autor podpořit i díky své dlouholeté externí spolupráci s obcí Nelahozeves.

Z výše uvedeného je zjevné, že jsou to právě malé obce a zejména pak obce prvního stupně, jejichž agendy jsou primárním cílem této práce, protože kvalitní informační systém může u těchto obcí alespoň částečně vykompenzovat určitý deficit odborného personálního zázemí úřadu.

3.2.2 Přehled klíčových agend

Tato kapitola shrnuje klíčové agendy malých obcí. V úvodu jsou vyjmenovány všechny hlavní agendy včetně vysvětlení, jakým způsobem byly informace shromážděny.

Vzhledem k rozsahu práce není možné provést návrh zcela všech agend, které obce zajišťují, byť v předcházející kapitole jsme již omezili rozsah agend přibližně na obce prvního stupně.

V prvé řadě vyloučíme agendy, u nichž ani nedává implementace nového informačního systému smysl, protože potřebnou problematiku zajišťují již existující informační systémy veřejné správy – typicky jde například o výkon působnosti kontaktního místa Czech POINT, kde informační systém provozuje na základě zákonného pověření Ministerstvo vnitra (§ 8a odst. 6 zákona č. 365/2000 Sb. o informačních systémech veřejné správy). U těchto agend by samozřejmě bylo možné implementovat novou prezentační vrstvu, jejímž smyslem by bylo zajistit jednotný vzhled a ovládání napříč všemi aplikacemi (agendami). S příslušným informačním systémem veřejné správy by tato prezentační vrstva

komunikovala prostřednictvím API, které poskytuje jak informační systém Czech POINT[21], tak například Informační systém datových schránek (ISDS)[22]. kde je API poskytováno přímo na základě zákonného pověření (§ 14a zákona č. 300/2008 Sb. o elektronických úkonech a autorizované konverzi dokumentů) a řada komerčních subjektů jej ve svých aplikacích využívá, kdy svým zákazníkům nabízejí různé nadstavby nad standardní funkcionalitou ISDS[23]. Tato problematika ale přesahuje rozsah této práce.

Obecně lze říci, že pro výběr vhodných agend k implementaci specifické informační podpory by hlavním kritériem měla být přidaná hodnota pro uživatele (obec). Z tohoto úhlu pohledu lze říci, že budou vybírány primárně agendy, které:

- jsou rozsáhlé (velké množství zpracovávaných dat) a/nebo komplexní (velké množství prováděných různorodých úkonů), protože právě u takových lze nalézt největší potenciál pro optimalizaci činnosti a snížení rizika lidské chyby,
- jsou časté či pravidelné – budou tedy vybírány primárně agendy, které se provádějí relativně často (např. několikrát za týden), oproti agendám, které se provádějí spíše zřídka či nahodile.

Přehled relevantních agend byl získán s pomocí dvou základních zdrojů:

- **Katalogy agend a činností veřejné správy** (Katalogy agend a činností veřejné správy, 2009). Kompetence orgánů veřejné správy upravuje řada zákonů a dalších právních předpisů. Ministerstvo vnitra tedy shromáždilo na jedno místo (excelová tabulka) soupis jednotlivých činností určitého úřadu, včetně obcí, podle platných zákonů. Katalog je sice 13 let starý, ovšem vzhledem k charakteru informací a obecně malé dynamice změn legislativy (máme na mysli v podstatných ohledech, nikoliv dílčí novelizace příslušných zákonů, které jsou, jak známo, velmi časté) jde stále o použitelný zdroj.

[21] https://www.czechpoint.cz/public/vyvojari/informace-pro-vyvojare-aplikaci

[22] https://www.datoveschranky.info/technicke-pozadavky/pristupove-rozhrani

[23] Například https://chytradatovka.cz.

- **Účelová právní analýza samosprávných kompetencí obcí v oblasti plánování a realizace rozvojových aktivit obcí** (Účelová právní analýza samosprávných kompetencí obcí v oblasti plánování a realizace rozvojových aktivit obcí, 2012). Jde o dokument vypracovaný firmou GaREP v rámci projektu Ministerstva pro místní rozvoj (MMR) s názvem *Elektronická metodická podpora tvorby rozvojových dokumentů obcí.* Přehled agend v tomto dokumentu se částečně překrývá s katalogem Ministerstva vnitra, jsou zde ale uvedeny některé agendy v katalogu nezmíněné, například problematika územního plánování.

V dalším textu jsou jednotlivé agendy stručně popsány, je provedeno jejich základní hodnocení z pohledu výše uvedených dvou kritérií (velikost a frekvence), následně je provedeno závěrečné shrnutí a finální výběr.

3.2.2.1 Oznámení o shromažďování

Popis agendy: Obecní úřad přijímá oznámení o plánovaném shromáždění v katastrálním území dané obce. Úřad vede evidenci shromáždění, posuzuje přijatá podání a činí další nezbytné související úkony.

Související legislativa: zákon č. 84/1990 Sb. o právu shromažďovacím

Hodnocení: U malých obcí spíše marginální agenda s minimální frekvencí i administrativou. Agenda nebude zahrnuta do návrhu informačního systému.

3.2.2.2 Loterie

Popis agendy: Obecní úřad vydává na svém území povolení k provozování loterií, tombol a výherních automatů. Vypočítává správní poplatky spojené s povolením (výběr poplatku už ale zajišťují finanční úřady, které je distribuují obcím).

Související legislativa: zákon č. 202/1990 Sb. o loteriích a jiných podobných hrách

Hodnocení: U malých obcí spíše menší agenda se zanedbatelnou frekvencí a nepříliš rozsáhlou administrativou. Agenda nebude zahrnuta do návrhu informačního systému.

3.2.2.3 Místní referendum

Popis agendy: Obecní úřad přijímá návrh na konání místního referenda a provádí jeho formální posouzení. V případě vyhlášení místního referenda zabezpečuje obecní úřad toto místní referendum po organizační stránce (volební místnosti, činnost komisí apod.).

Související legislativa: zákon č. 22/2004 Sb. o místním referendu

Hodnocení: Komplexnost agendy není vysoká, a zejména jde o činnost, která se provádí velmi zřídka. Ve většině obcí ČR neproběhlo dosud žádné místní referendum.[24] Agenda nebude zahrnuta do návrhu informačního systému.

3.2.2.4 Obecní policie

Popis agendy: Obecní policie je orgánem obce, který zřizuje a zrušuje obecní zastupitelstvo obecně závaznou vyhláškou. Obecní policie je tvořena zaměstnanci obce zařazenými do obecní policie, obec zajišťuje personalistiku obecních strážníků a rovněž podává přihlášky ke zkouškám odborné způsobilosti. Obecní policii řídí starosta nebo jiný člen zastupitelstva obce pověřený zastupitelstvem obce. Obec je povinna nahradit škodu způsobenou strážníkem při výkonu jeho působnosti.

Související legislativa: zákon č. 553/1991 Sb. o obecní policii

Hodnocení: Středně komplexní agenda, jejíž podstatnou součástí je ale personalistika obecních strážníků, která je z velké části shodná s obecnou personalistikou. U malých obcí je počet obecních strážníků spíše nízký (do deseti osob). Činnost samotné obecní policie není předmětem této práce. Agenda tedy nebude zahrnuta do návrhu informačního systému.

[24] Například podle článku v Deníku Referendum z 14. 10. 2020 (https://denikreferendum.cz/clanek/31649-mistni-referenda-2020-sirsi-participaci-obcanu-brani-spatny-zakon) se v celé historii České republiky dosud konalo jen několik set referend.

3.2.2.5 Rozpočet obce

Popis agendy: Obec sestavuje návrh rozpočtu na následující rok v členění definovaném zákonem, na základě priorit aktuální politické reprezentace obce. Obecní úřad vede evidenci příjmů a výdajů pro jednotlivé rozpočtové kapitoly a kontroluje jejich nepřekračování. V případě potřeby jsou připravovány návrhy změn rozpočtu, tzv. rozpočtová opatření. Dobrou praxí je prezentace rozpočtu veřejnosti ve vhodné formě.

Související legislativa: zákon č. 250/2000 Sb. o rozpočtových pravidlech územních rozpočtů

Hodnocení: Velmi komplexní a specifická agenda, s každodenní frekvencí i u malých obcí, podléhající relativně přísné legislativě a častým kontrolám zejména z krajského úřadu. Agenda bude do návrhu informačního systému zahrnuta.

3.2.2.6 Poskytování informací, úřední deska

Popis agendy: Obecní úřad je povinným subjektem podle informačního zákona a je povinen poskytovat informace na žádost. Obecní úřad je současně povinen vést úřední desku a zveřejňovat na ní v požadovaných lhůtách řadu informací na základě pověření řady zákonů.

Související legislativa: zejména zákon č. 106/1999 Sb. o svobodném přístupu k informacím a zákon č. 500/2004 Sb., správní řád

Hodnocení: Až na naprosté výjimky je u malých obcí agenda poskytování informací na žádost zanedbatelná[25], tato agenda nebude do návrhu informačního systému zahrnuta. Významnější je agenda vedení úřední desky, která se provádí každodenně a informační systém zde může hrát důležitou roli zejména v souvislosti s hlídáním lhůt pro sejmutí dokumentu. Tato agenda bude do návrhu informačního systému zahrnuta.

[25] Například v obci Nelahozeves (cca 2 000 obyvatel) byly v roce 2022 podány pouze čtyři žádosti o informace, obdobně pak v předchozích letech (https://nelahozeves.cz/obcan/poskytovani-informaci/poskytnute-informace/).

3.2.2.7 Vlastnictví bytových jednotek

Popis agendy: Jedná se o agendu v samostatné působnosti obce, kdy obec pronajímá a spravuje bytové jednotky ve svém vlastnictví. Obec zde působí v roli běžného pronajímatele, jako jakákoliv jiná fyzická či právnická osoba.

Související legislativa: zákon č. 89/2012 Sb., občanský zákoník

Hodnocení: Malé obce zpravidla žádné bytové jednotky nevlastní, a pokud ano, tak ve velmi malém množství. Agenda není pro návrh informačního systému relevantní.

3.2.2.8 Evidence majetku – nemovitosti, dopravní značky, stožáry veřejného osvětlení apod.

Popis agendy: Obec je povinna vést evidenci svého majetku, ať už jde o nemovitosti, místní komunikace či další majetek, jako jsou například stožáry veřejného osvětlení.

Související legislativa: například zákon č. 13/1997 Sb. o pozemních komunikacích

Hodnocení: Množství nemovitého i movitého majetku ve vlastnictví obce nemusí být úplně zanedbatelné, jde však o agendu s velmi nízkou frekvencí změn. Agenda nebude do návrhu informačního systému zahrnuta.

3.2.2.9 Matrika, evidence obyvatel

Popis agendy: Obecní úřady vedou evidenci trvale žijících obyvatel na svém území a změny zapisují do příslušných registrů státní správy.

Související legislativa: zákon č. 301/2000 Sb. o matrikách a zákon č. 133/2000 Sb. o evidenci obyvatel

Hodnocení: Podstatná část agendy se odehrává v Informačním systému evidence obyvatel (ISEO)[26]. Obec však z mnoha důvodů potřebuje vést evidenci obyvatel trvale

[26] https://archi.gov.cz/nap:iseo

žijících na jejím území i pro své vlastní účely. Do návrhu informačního systému tedy bude zahrnuta agenda alespoň částečně.

3.2.2.10 Evidence adres (zákon o obcích, katastr)

Popis agendy: Obecní úřady vedou evidenci všech adres na svém území.

Související legislativa: zákon č. 111/2009 Sb. o základních registrech

Hodnocení: Stejně jako v předchozím případě se podstatná část agendy odehrává ve státním informačním systému, v tomto případě jde o registr územní identifikace, adres a nemovitostí (RÚIAN)[27]. I v tomto případě ale obec potřebuje registr adres i pro své vlastní účely. Agenda tedy bude do informačního systému částečně zahrnuta.

3.2.2.11 Záchranný systém a krizové řízení, hasičský sbor

Popis agendy: Obec má v době krizových stavů speciální postavení, může například vydávat nařízení sloužící k zajištění provedení krizových opatření v podmínkách obce. Obec může mít současně zřízenu jednotku sboru dobrovolných hasičů, z čehož jí plynou určité povinnosti, například poskytování náhrady ušlého výdělku členům jednotky v případě zásahu.

Související legislativa: Zákon č. 240/2000 Sb. o krizovém řízení, zákon č. 133/1985 Sb. o požární ochraně

Hodnocení: Agenda může být potenciálně velmi komplikovaná, ovšem její využití je omezeno na krizové stavy či jiné obdobné situace. Z tohoto důvodu nebude agenda do informačního systému zahrnuta.

[27] https://www.cuzk.cz/ruian

3.2.2.12 Přestupkové řízení

Popis agendy: Obecní úřad, má-li zřízenu přestupkovou komisi, vede přestupková řízení.

Související legislativa: zákon č. 250/2016 Sb., o odpovědnosti za přestupky a řízení o nich

Hodnocení: Nemá-li obec zřízenu obecní policii a neprovozuje-li kamerový radar, je množství přestupků velmi malé a taková agenda nevyžaduje specializovanou podporu informačního systému. Naopak v případě zřízení obecní policie či provozování radaru může počet přestupků rapidně vzrůst. Zde je však podpora informačního systému velmi úzce svázána s technologií samotného radaru či prostředky obecní policie. Z důvodu velké specifičnosti uvedené agendy nebude pro tuto oblast podpora informačního systému zajištěna.

3.2.2.13 Provoz vodovodu a kanalizace

Popis agendy: Obec může být vlastníkem či provozovatelem vodovodu nebo kanalizace. Z tohoto titulu vede agendu nezbytnou pro údržbu sítě a zároveň zajišťuje administrativu spojenou s prováděním odečtů odběru vody a výběrem vodného a stočného.

Související legislativa: zákon č. 274/2001 Sb. o vodovodech a kanalizacích

Hodnocení: Rozsah vodovodní či kanalizační sítě u malé obce nebývá velký, takže běžná údržba sítě podporu v podobě informačního systému nevyžaduje. Administrativně náročnější je ale problematika odečtů odběru vody a související úhrady vodného a stočného – tyto oblasti budou zahrnuty do návrhu informačního systému.

3.2.2.14 Veřejné zakázky, projektové řízení

Popis agendy: Obecní úřad vyhlašuje veřejné zakázky na stavební práce, dodávky a služby, které potřebuje pro svou činnost, a sleduje průběh veřejné zakázky i v době její realizace.

Související legislativa: zákon 134/2006 Sb. o zadávání veřejných zakázek

Hodnocení: Obecně jde o velmi komplikovanou agendu, která u větších zadavatelů vyžaduje podporu v podobě informačního systému. U malých obcí je ale vzhledem k jejich velmi omezenému rozpočtu vyhlašováno a realizováno veřejných zakázek relativně málo. Z tohoto důvodu nebude agenda do návrhu informačního systému zahrnuta.

3.2.2.15 Odpadové hospodářství

Popis agendy: Obec provozuje systém odpadového hospodářství na svém území. Zajišťuje svoz odpadů, vybírá za svoz poplatky od obyvatel či vlastníků nemovitostí.

Související legislativa: zákon 541/2020 Sb. o odpadech, zákon č. 565/1990 Sb. o místních poplatcích

Hodnocení: Rozsáhlá a komplexní agenda, neboť se týká všech obyvatel obce či vlastníků nemovitostí, přičemž sazebníky poplatků mohou být velmi variabilní a zároveň s agendou souvisí evidence množství odpadů svážených z jednotlivých nemovitostí. Agenda bude zahrnuta do návrhu informačního systému.

3.2.2.16 Poplatky za psa

Popis agendy: Obecní úřad vede evidenci psů a vybírá poplatky za psa.

Související legislativa: zákon č. 565/1990 Sb. o místních poplatcích

Hodnocení: Jednoduchá agenda, ale její náročnost je relativně vysoká z toho důvodu, že se týká (zejména na vesnicích) téměř všech obyvatel obce. Agenda bude zahrnuta do návrhu informačního systému.

3.2.2.17 Pronájem hrobových míst

Popis agendy: Obec vede evidenci hrobových míst, zajišťuje jejich pronájmy a výběr poplatků.

Související legislativa: zákon č. 256/2001 Sb. o pohřebnictví

Hodnocení: Na malých obcích nepříliš významná agenda, neboť poplatky za pronájem hrobových míst se obvykle vybírají na 10 let, takže frekvence agendy je velmi malá. Přesto bude agenda do návrhu informačního systému zahrnuta, neboť výběr poplatků je již beztak zahrnut v rámci agend popsaných v kapitolách 3.2.2.15 a 3.2.2.16.

3.2.2.18 Povolení kácení dřevin mimo les

Popis agendy: Obecní úřad vydává povolení kácení dřevin mimo les.

Související legislativa: zákon č. 114/1992 Sb. o ochraně přírody a krajiny

Hodnocení: Na malých obcích nefrekventovaná a jednoduchá agenda. Nebude zahrnuta do návrhu informačního systému.

3.2.2.19 Povolení zvláštního užívání komunikace

Popis agendy: Obecní úřad v roli silničního správního úřadu vydává povolení ke zvláštnímu užití komunikace (například stavební práce, umístění inženýrských sítí, provozování stánků, sportovní a kulturní akce apod.

Související legislativa: zákon č. 13/1997 Sb. o pozemních komunikacích

Hodnocení: U malých obcí nepříliš složitá a zejména málo frekventovaná agenda. Nebude zahrnuta do návrhu informačního systému.

3.2.2.20 Činnost zastupitelstva

Popis agendy: Jedná se o velmi specifickou agendu zahrnující zejména:

- svolávání veřejných zasedání zastupitelstva,
- předávání podkladů jednání zastupitelům a (v rámci dobré praxe) též obyvatelům obce,
- přípravu a následné zveřejnění zápisu ze zasedání zastupitelstva, případně též jeho online vysílání,
- odměňování zastupitelů za jejich činnost.

Související legislativa: zákon č. 128/2000 Sb. o obcích

Hodnocení: Jde o klíčovou agendu obce. V případě jejího vykonávání v minimalistické (minimálně zákonem vyžadované) podobě není podpora informačního systému zapotřebí. Moderní přístupy vedení obcí zahrnující silnější participaci občanů na činnosti samosprávy a její celkovou otevřenost a transparentnost vůči občanům však nároky na informační podporu zvyšují, a to zejména v oblasti online vysílání a možného propojení tohoto online vysílání s písemnými podklady. Tato oblast bude do návrhu informačního systému zahrnuta.

3.2.3 Výběr agend

Na základě stručného posouzení jednotlivých agend malých obcí se autor práce rozhodl, že analýza a návrh informačního systému bude provedena pro následující oblasti:

- Rozpočet obce
- Úřední deska
- Evidence trvalého pobytu
- Evidence adres
- Odpadové hospodářství
- Poplatky za psa
- Vodné a stočné
- Pronájem hrobových míst
- Činnost zastupitelstva

V praktické části práce je individuálně posouzena úroveň podpory pro jednotlivé oblasti, mj. i na základě konzultací s vedením obce Nelahozeves.

4 Praktická část

Praktická část se zabývá vlastní analýzou a návrhem informačního systému pro podporu agend malých obcí, vytipovaných v předcházející kapitole.

Tato část práce se opírá jednak o teoretická východiska formulovaná výše v této práci, jednak o vlastní odbornou zkušenost autora. Autor dlouhodobě pracuje jako softwarový analytik a současně působí jako externí konzultant pro oblast softwarového vývoje v obci Nelahozeves, kde žije. V rámci své konzultační činnosti autor v uplynulých letech přišel do úzkého styku s řadou obecních agend. K orientaci v činnosti místní samosprávy mu pomohla i funkce vedoucího redaktora Nelahozeveského zpravodaje, kterou zastává od roku 2019.

Vzhledem k tomu, že některé části práce by v budoucnosti měly najít využití přímo v obci Nelahozeves (ve formě zadání vývoje aplikace na zakázku některé softwarové firmě), podstatné vstupy pro analytické a návrhové práce poskytlo autorovi přímo vedení obce Nelahozeves, jmenovitě starosta Jakub Brynda a místostarosta Zdeněk Schneider, jakož i některé referentky obecního úřadu.

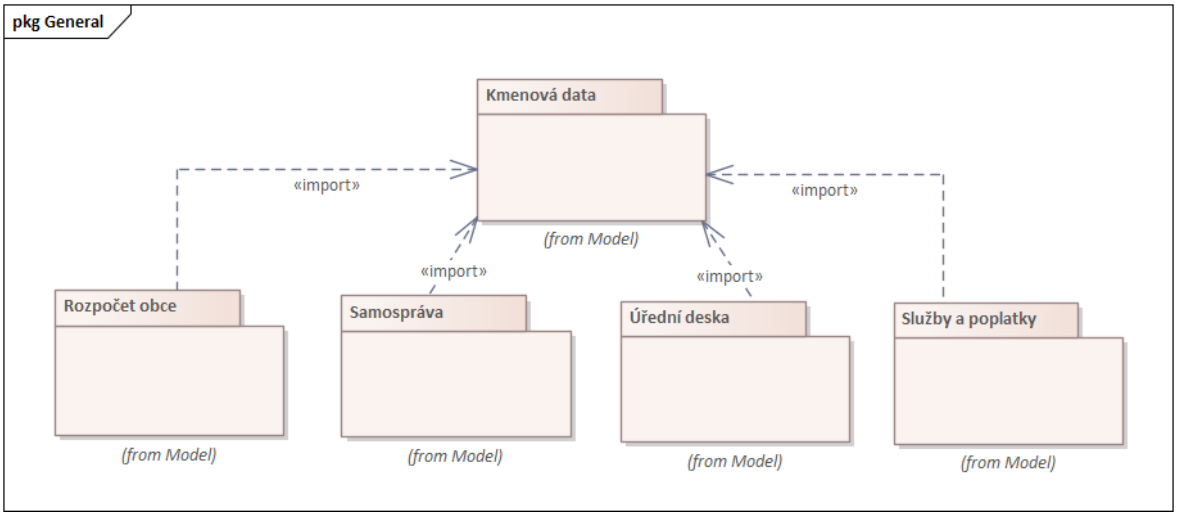
4.1 Členění kapitoly a základní dekompozice systému

Členění této kapitoly odpovídá teoretickým východiskům formulovaným výše. Autor se však musel na samém počátku vypořádat s obvyklým problémem při navrhování větších aplikací, kdy je třeba na počátku rozhodnout, zda první úroveň členění bude představovat business oblast (například správa úřední desky) či druh modelu (diagramu). Jinými slovy je třeba určit, zda aplikace bude popisována průřezově horizontálně, tzn. napříč všemi moduly, či vertikálně, tzn. každý modul samostatně. Oba přístupy mají své klady i zápory. Vzhledem k relativní nezávislosti jednotlivých modulů na sobě (každý modul je navrhován tak, aby byl schopen provozu relativně samostatně) byl zvolen přístup vertikální, v souladu s řadou odborných doporučení[28].

[28] Například https://www.modularmanagement.com/blog/software-modularity.

Kapitola je tedy členěna ve struktuře jednotlivých agend (modulů) a v jejich rámci jsou pak uvedeny jednotlivé potřebné diagramy a jejich specifikace.

Další otázkou k rozhodnutí je hlavní úroveň dekompozice systému. Nativním prostředkem dekompozice návrhu v UML jsou balíčky a členění systému do menších celků se modeluje pomocí diagramu balíčků (package diagram). V kapitole 3.2.3 byly vybrány agendy k dalšímu zpracování, to však nutně neznamená, že jedné vybrané agendě odpovídá jeden balíček; členění cílové aplikace je vždy třeba zvážit s ohledem na různá kritéria, přičemž jedním ze základních požadavků je „vysoký stupeň soudržnosti uvnitř balíčku a co nejmenší počet vazeb mezi balíčky“ (Arlow, 2007, s. 240). Na základě dalších úvah autora a konzultací s vedením obce Nelahozeves bylo rozhodnuto o členění systému, které znázorňuje Obrázek 14.



Obrázek 14: Základní členění systému na moduly vyjádření pomocí diagramu balíčků

Systém je rozdělen na pět modulů:

- **Kmenová data** (kapitola 4.2): Základní modul (balíček) obsahující problematiku evidence uživatelů, obyvatel, evidence pobytu a evidence adres. Některé z těchto funkcionalit jsou využívány dalšími moduly, což je v diagramu vyjádřeno vazbami. Modul též obsahuje několik dalších pomocných obecných funkcionalit.
- **Rozpočet** (kapitola 4.3): Zahrnuje problematiku správy obecního rozpočtu.

- **Samospráva** (kapitola 4.4): Zajišťuje evidenci samosprávných orgánů a jejich jednání, členů orgánů včetně jejich odměn.
- **Úřední deska** (kapitola 4.5): Zabezpečuje administraci záznamů na úřední desce včetně jejich publikace na externích platformách.
- **Služby a poplatky** (kapitola 4.6): Zahrnuje problematiku poplatků za odpady, psy, vodné a stočné a hrobová místa.

4.2 Kmenová data

4.2.1 Stručný popis modulu

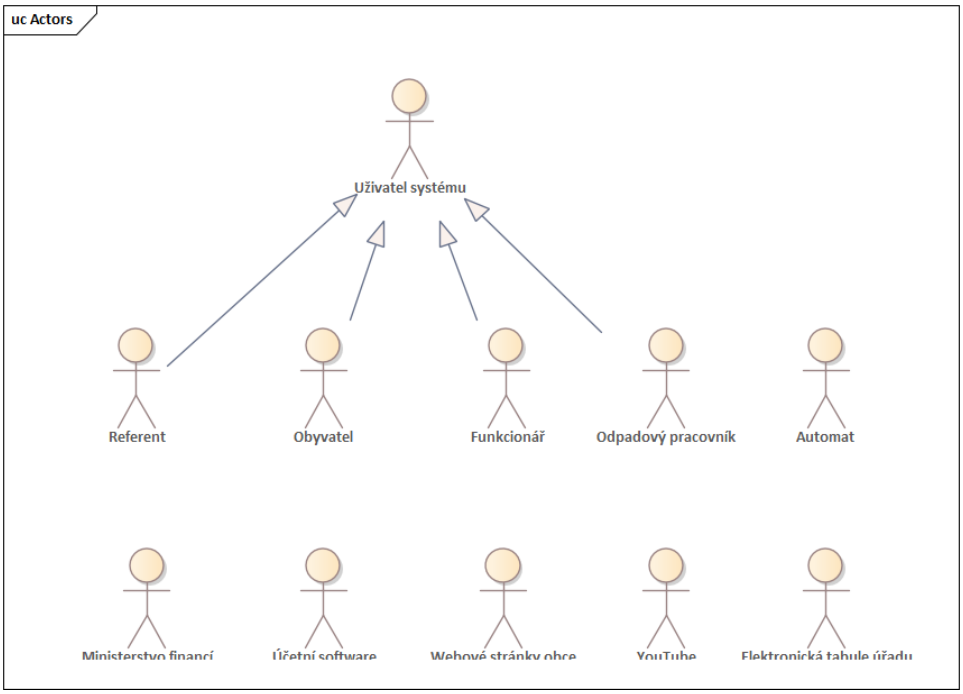
Modul *Kmenová data* zajišťuje administraci základních kmenových dat systému, mezi něž patří zejména osoby (v nějakém vztahu k obci), jednotky (tzn. byty, rodinné domy apod.) a vztahy mezi nimi.

4.2.2 Funkční požadavky

ID	*Název*	*Popis*	*Priorita*
KMN-RQ-01	Správa osob	Systém podporuje správu osob, které mají nějaký vztah k obci.	M
KMN-RQ-02	Správa jednotek	Systém podporuje správu jednotek (bytů, rodinných domů apod.) a vztahu osob k jednotkám. Vztahy mohou být různého typu a mohou se v čase měnit.	M
KMN-RQ-03	Synchronizace se systémem státní správy	Systém podporuje obousměrnou synchronizaci s Informačním systémem evidence obyvatel (ISEO).	M

4.2.3 Diagram případů užití

Pro potřeby diagramů v následujících kapitolách byl navržen jeden diagram znázorňující všechny aktéry systému (Obrázek 15).



Obrázek 15: Aktéři systému

Nejobecnějším aktérem je *Uživatel systému,* kterým může být kterýkoliv uživatel, i neregistrovaný. Neregistrovaný uživatel má však v systému velmi omezená přístupová práva, neboť hlavní zaměření systému je interní správa agend malé obce.

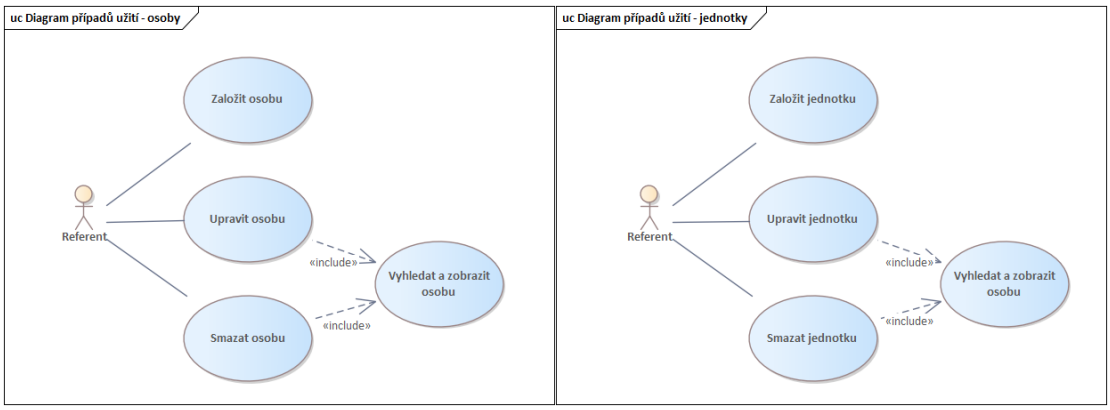
Z obecného uživatele jsou zděděni uživatelé *Referent*, *Obyvatel* a *Funkcionář.* Referent je uživatel, který v systému aktivně vykonává většinu podporovaných agend. Obyvatel má přístup k některým údajům vztahujícím se k jeho nemovitosti, typicky jde o komunální služby a jejich vyúčtování. Funkcionář má k dispozici materiály vztahující se k jednotlivým jednáním a též přehled svých odměn.

Dalším uživatelem, který má využití pouze v modulu *Služby a poplatky,* je Odpadový pracovník. Jeho role je popsána v kapitole 4.6.3.

Zvláštním uživatelem systému je *Automat.* Jde o automatického uživatele, který v předem daných časech provádí různé automatické operace. V literatuře se pro tohoto uživatele používají též termín *Timer.*

V dolní části diagramu jsou znázorněni pasivní aktéři. Jedná se o aktéry, kteří nepracují se systémem aktivně, ale systém jim předává různá data nebo zprávy. Význam těchto aktérů je vysvětlen vždy u příslušného modulu, kde se daný aktér používá.

Dále jsou v rámci modulu *Kmenová data* navrženy základní případy užití pro administraci osob a jednotek (viz Obrázek 16).



Obrázek 16: Diagramy případů užití pro administraci osob a jednotek

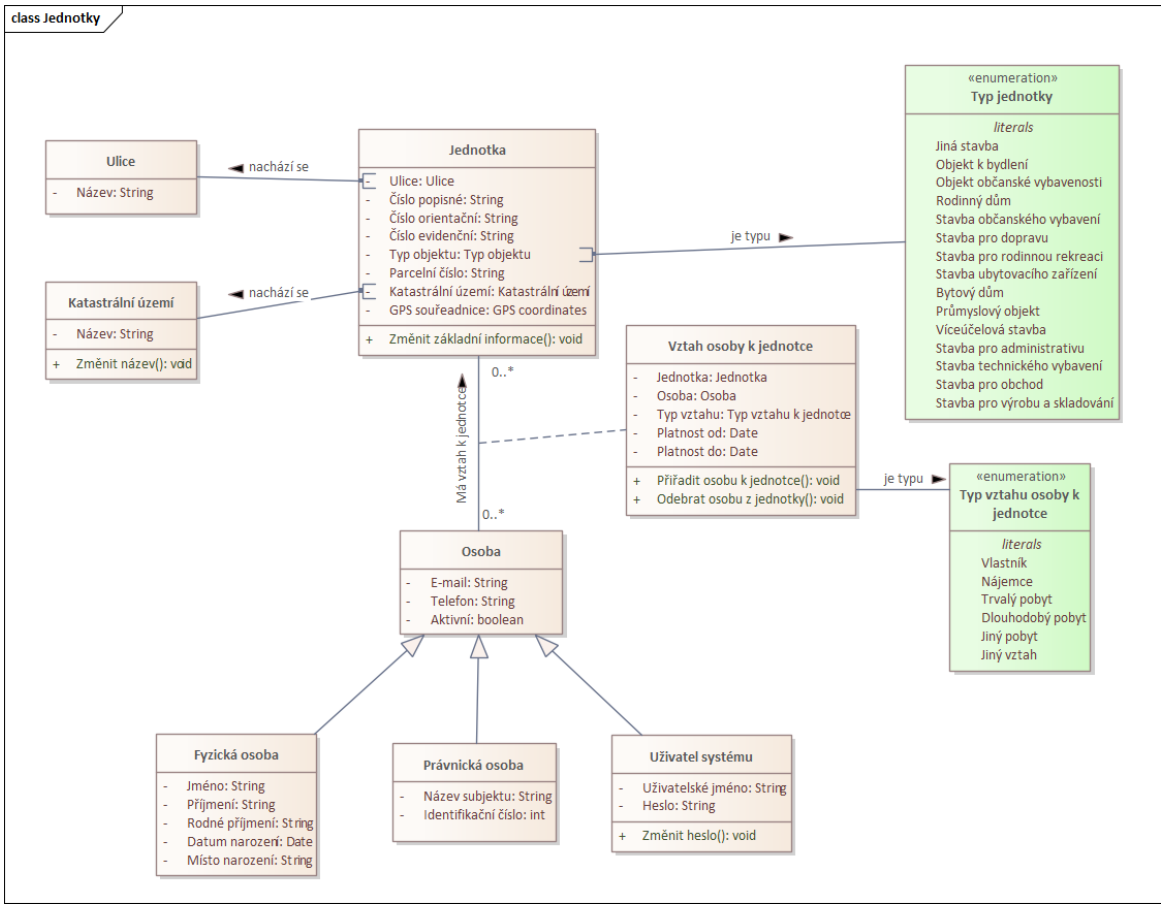
Logika navržených případů užití je velmi jednoduchá a přímočará. Za pozornost stojí vzájemné propojení osob a jednotek, které je realizováno v rámci případu užití *Upravit jednotku*. Ten je dále specifikován ve formě scénáře.

Název případu užití	Upravit jednotku
Identifikátor	KMN-UC-006
Stručný popis	Případ užití slouží k úpravě vlastností jednotky.
Aktéři	Referent
Vstupní podmínky	Žádné

Hlavní scénář	1. Uživatel vyhledá a zobrazí jednotku (KMN-UC-008). 2. Uživatel dle potřeby upraví vlastnosti jednotky (adresa, typ apod.). 3. Uživatel potvrdí změny.
Výstupní podmínky	Jsou provedeny požadované změny v jednotce.
Alternativní scénáře	a) V bodě 2 uživatel přistoupí k přiřazování osob k jednotce. 2.a.1. Uživatel vyhledá a zobrazí osobu. 2.a.2. Uživatel zvolí roli osoby ve vztahu k jednotce a její platnost. Výchozí „platnost od“ je dnešní den, výchozí „platnost do“ je neomezená. b) V bodě 2.a.1. uživatel nenalezl požadovanou osobu. Uživatel se rozhodne osobu vytvořit (KMN-UC-001). Po vytvoření osoby uživatel pokračuje v tomto případu užití. c) V bodě 2 uživatel přistoupí k odebírání osob z jednotky. 2.c.1. Uživatel vybere již přiřazenou osobu a odebere ji z jednotky.

4.2.4 Diagram tříd

Hlavní diagram tříd znázorňuje jednotky, osoby a jejich vzájemné vztahy (Obrázek 17).



Obrázek 17: Diagram tříd modulu Kmenová data

Modul *Kmenová data* obsahuje dvě hlavní třídy – *Jednotka* a *Osoba.* Osobou je libovolná osoba s nějakým vztahem k obci. Typicky se může jednat o vlastníka nemovitosti nebo o osobu s trvalým pobytem v obci.

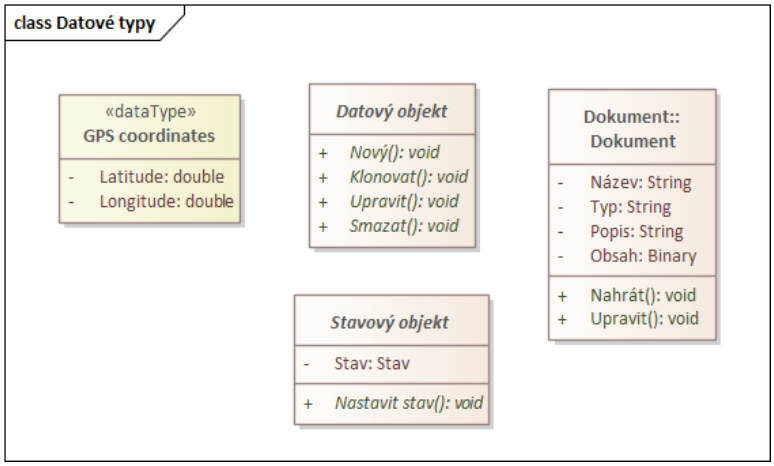
U jednotky je evidována ulice (z číselníku ulice v dané obci) a číslo (popisné a/nebo orientační a/nebo evidenční). K jednotce je dále evidován její typ (byt, rodinný dům apod.). Pro možné budoucí napojení na mapovou aplikaci je evidováno katastrální území a GPS souřadnice jednotky.

Třída *Osoba* obsahuje základní informace o osobě, která je v nějakém vztahu k obci. Z obecné osoby je zděděna *Fyzická osoba* a *Právnická osoba,* každá se svými specifickými atributy. Zároveň je z osoby zděděna třída *Uživatel systému,* která umožňuje dané osobě přihlásit se přímo do systému (s příslušnými přístupovými právy). Tato funkcionalita má

význam například ve vztahu k agendě služeb a poplatků (viz kapitola 4.6). Osoba, která není uživatelem systému, je pouze pasivním aktérem (subjektem), ale nemůže se přímo přihlásit do systému a používat jej. Autor zvažoval i možnost zdědění uživatele systému z fyzické osoby (a nikoliv z obecné), nakonec však zohlednil i možnost, aby se do systému přihlašovala i přímo právnická osoba (například v roli vlastníka nemovitosti). Kdyby byl přístup umožněn jen fyzickým osobám, musel by být model rozšířen o vazby fyzických osob na právnické (tzn. fyzická osoba s přístupem k datům právnické osoby – podobně jako například u Informačního systému datových schránek[29]).

Vazba mezi osobou a jednotkou je M : N – jedna osoba může mít vztah k více jednotkám a k jedné jednotce může mít vztah více osob. V datovém modelu by bylo nezbytné propojit osobu a jednotku prostřednictvím další tabulky, v diagramu tříd je tento typ vazby namodelován pomocí asociační třídy *Vztah osoby k jednotce*. Tato třída nese informace o platnosti a typu vztahu (například odkdy dokdy osoba vlastní jednotku).

V rámci modulu *Kmenová data* jsou, v souladu se základní dekompozicí systému (viz kapitola 4.1), dále navrženy pomocné třídy, které mají obecný charakter napříč business moduly. Tyto třídy znázorňuje Obrázek 18.



Obrázek 18: Obecné třídy

[29] § 8 zákona 300/2008 Sb. o elektronických úkonech a autorizované konverzi dokumentů.

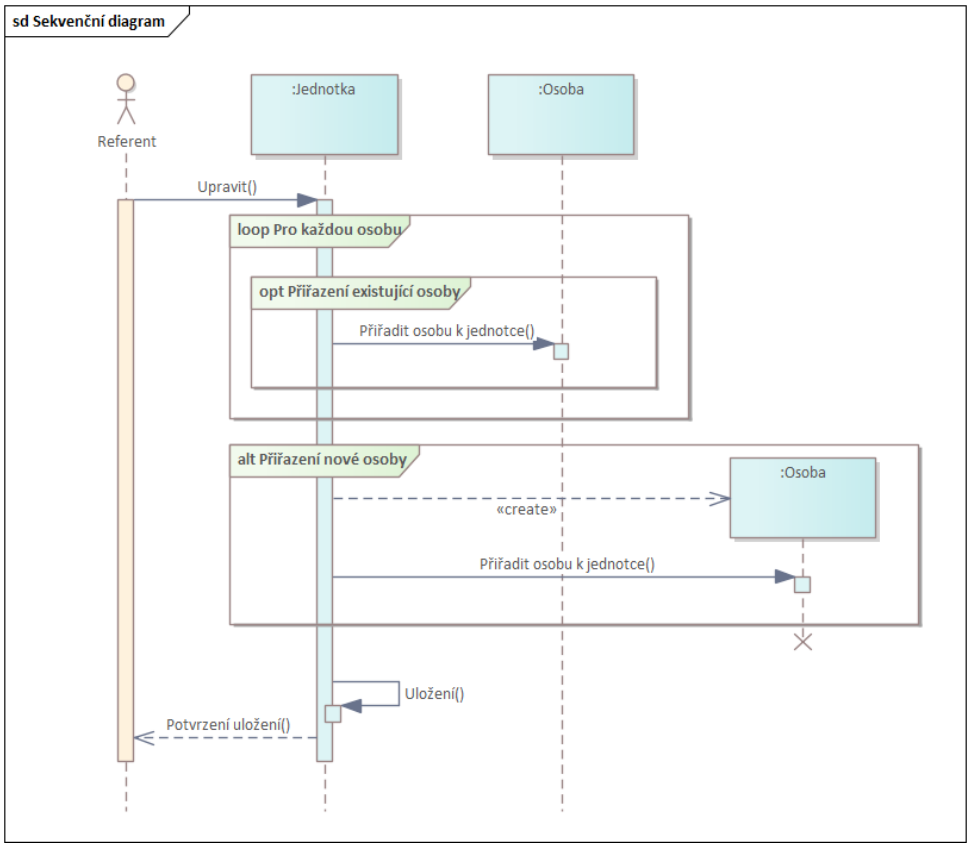
Datový objekt a *Stavový objekt* jsou abstraktní předci, jejichž jediným účelem je garantovat jednotné rozhraní všem třídám, které mají implementovat základní CRUD operace (vytvoření, klonování, úprava a smazání) či nastavování stavů. Třídy, které mají tyto operace podporovat, jsou pak pouze zděděny z tohoto abstraktního předka, čímž je dosaženo úspornosti zápisu a je eliminováno riziko nekonzistencí[30]. U jednotlivých tříd v diagramech tříd jednotlivých business modulů už pak z důvodu úspornosti a přehlednosti diagramu tuto dědičnost neuvádíme, ale pouze ji jednou větou zmiňujeme (jedinou výjimkou je modul *Rozpočet,* kde je generalizace znázorněna z ukázkových důvodů).

Třída Dokument je obecný dokument, který je dále využíván moduly *Samospráva* (přílohy k jednáním orgánů – viz kapitola 4.4) a *Úřední deska* (dokumenty vyvěšené na úřední desce – viz kapitola 4.5).

4.2.5 Sekvenční diagram

V rámci návrhu modulu *Kmenová data* je navržen sekvenční diagram případu užití, podrobněji ilustrující interakci mezi jednotlivými objekty v rámci případu užití *Upravit jednotku.*

[30] Doporučení viz například https://www.codemag.com/article/0201061/UML-Class-Diagrams.



Obrázek 19: Sekvenční diagram případu užití Upravit jednotku

Interakce znázorněná v sekvenčním diagramu odpovídá scénáři případu užití popsanému v kapitole 4.2.3. Znázorněné sekvence obsahují úpravu základních vlastností jednotky, jakož i přiřazení existující či nové osoby k jednotce.

4.3 Rozpočet

4.3.1 Stručný popis modulu

Modul *Rozpočet* slouží k administraci obecního rozpočtu. V modulu jsou udržovány jednotlivé verze rozpočtu od návrhu přes schválenou verzi až po skutečný stav. Základním legislativním podkladem pro práci s rozpočtem je zákon č. 250/2000 Sb. o rozpočtových pravidlech územních rozpočtů, respektive vyhláška č. 323/2000 Sb. o rozpočtové skladbě.

Obecní rozpočet se vztahuje vždy ke kalendářnímu roku a vzniká nejprve jako koncept, který se následně transformuje do oficiálního návrhu. Po schválení zastupitelstvem obce je rozpočet schválen. Rozpočet lze měnit i v průběhu roku pomocí takzvaných rozpočtových opatření, která rovněž schvaluje zastupitelstvo obce.

Rozpočet se skládá z položek, jejichž třídění je dáno výše uvedenou vyhláškou. Rozlišujeme položky příjmové a výdajové, přičemž příjmy i výdaje se třídí pomocí řady třídění. V rámci této aplikace bude podporováno třídění druhové a odvětvové, což jsou v praxi nejběžněji používané druhy třídění. V rámci daného třídění si může obec založit vlastní specifické podpoložky upřesňující členění dané položky.

Modul bude napojen na účetní software obce. Do budoucna může být celý systém rozšířen i o problematiku účetnictví obce.

Modul slouží k administraci rozpočtu ve všech jeho fázích a k přehledné prezentaci členům zastupitelstva a široké veřejnosti. Rozpočet se rovněž pravidelně odesílá Ministerstvu financí.

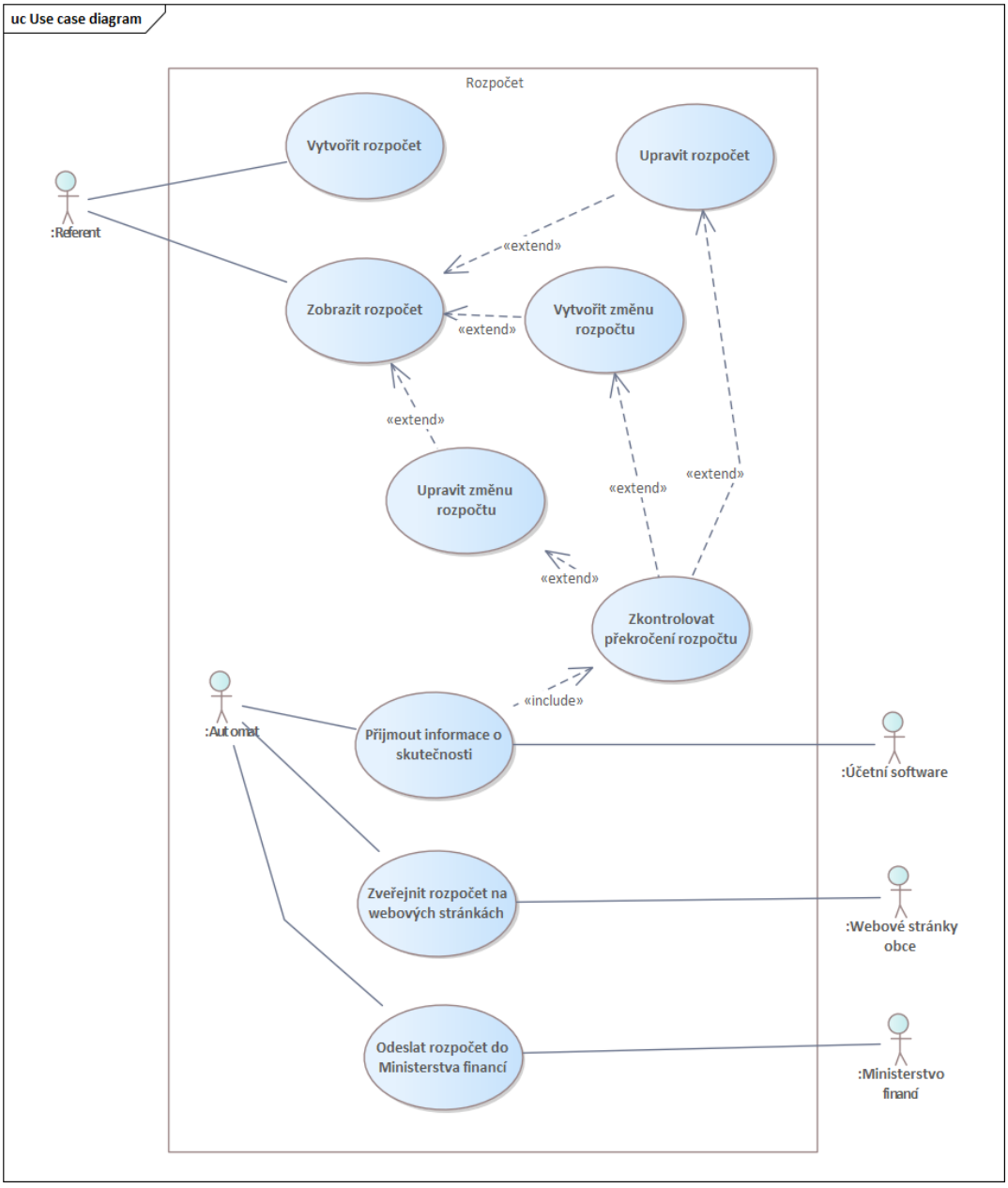
4.3.2 Funkční požadavky

ID	*Název*	*Popis*	*Priorita*
RZP-RQ-01	Správa rozpočtu	Systém podporuje správu obecního rozpočtu pro daný rok.	M
RZP-RQ-02	Životní tok rozpočtu	Systém podporuje životní tok rozpočtu od konceptu až po skutečný stav, jakož i proces změn rozpočtu v průběhu roku	M
RZP-RQ-03	Položky rozpočtu	Rozpočet se skládá z položek, podporováno je druhové a odvětvové třídění.	M
RZP-RQ-04	Prezentace rozpočtu	Systém podporuje přehlednou prezentaci rozpočtu pro členy zastupitelstva i širokou veřejnost.	M

ID	*Název*	*Popis*	*Priorita*
RZP-RQ-05	Aktualizace rozpočtu po změnách	Jednotlivé položky rozpočtu jsou automaticky přepočítány po schválení každé změny.	M
RZP-RQ-06	Aktualizace skutečného stavu	Skutečný stav rozpočtu bude získáván primárně propojením s účetním softwarem, z nějž budou přebírány data o uskutečněných příjmech a výdajích a budou agregovány na úroveň jednotlivých položek rozpočtu.	M
RZP-RQ-07	Varování při překročení výdaje	Systém bude varovat uživatele v případě, že výdajová položka je překročena (skutečné výdaje jsou vyšší než plánované).	M

4.3.3 Diagram případů užití

Na základě sesbíraných a zkonsolidovaných funkčních požadavků na modul byl navržen diagram případů užití.



Obrázek 20: Diagram případů užití modulu Rozpočet

Hlavním aktérem modulu je referent, který provádí veškeré základní manipulace s rozpočtem. Referent zakládá, upravuje či případně maže rozpočet a jeho položky, stejně jako rozpočtové změny. V případě změny s dopadem na saldo rozpočtu je vyvolána kontrola případného překročení rozpočtu.

Dalším aktérem je automat, který ve stanovených časech provádí požadované automatizované operace. Pasivními aktéry jsou:

- účetní software, z nějž jsou přijímány informace o skutečně provedených příjmech a výdajích
- webové stránky obce, na nichž je rozpočet publikován ve formě srozumitelné široké veřejnosti,
- Ministerstvo financí, jemuž je rozpočet pravidelně zasílán v požadované strukturované podobě.

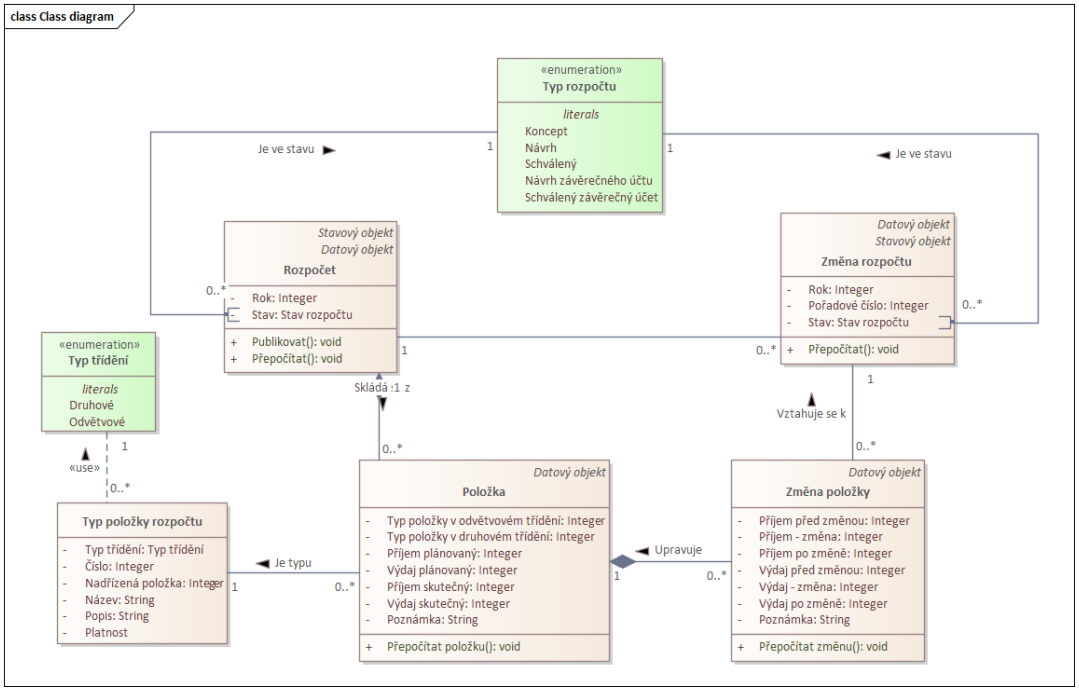
Dále v této kapitole jsou ve formě scénářů popsány klíčové případy užití.

Název případu užití	Vytvořit rozpočet
Identifikátor	RZP-UC-001
Stručný popis	Vytvořit manuálně rozpočet pro daný rok.
Aktéři	Referent
Vstupní podmínky	Žádné
Hlavní scénář	1. Uživatel zvolí rok, pro nějž chce vytvořit rozpočet. 2. Uživatel zvolí, že chce rozpočet vytvořit jako prázdný. 3. Uživatel potvrdí vytvoření rozpočtu.
Výstupní podmínky	Rozpočet na daný rok je vytvořen.

Alternativní scénáře	V bodě 2 uživatel zvolí, že chce rozpočet vytvořit klonem z jiného rozpočtu. Uživatel zvolí rok, z nějž chce rozpočet vytvořit. V bodě 3 je zjištěno, že rozpočet na daný rok již existuje. Rozpočet nelze vytvořit.

4.3.4 Diagram tříd

Obrázek 21 znázorňuje diagram tříd modulu *Rozpočet.*



Obrázek 21: Diagram tříd modulu Rozpočet

Diagram znázorňuje základní třídy navržené pro modul *Rozpočet*. V srdci modulu je stejnojmenná třída *Rozpočet*, která reprezentuje rozpočet na určitý rok. Rozpočet může být v určitém stavu, daném atributem *Stav*. K rozpočtu se vážou položky rozpočtu. Na rozpočet mohou být navázány změny, které mají rovněž své změnové položky.

Položka rozpočtu má definován svůj typ v daném typu třídění, podporovány jsou odvětvové a druhové třídění. Každý typ položky rozpočtu (kromě kořenového) má svou

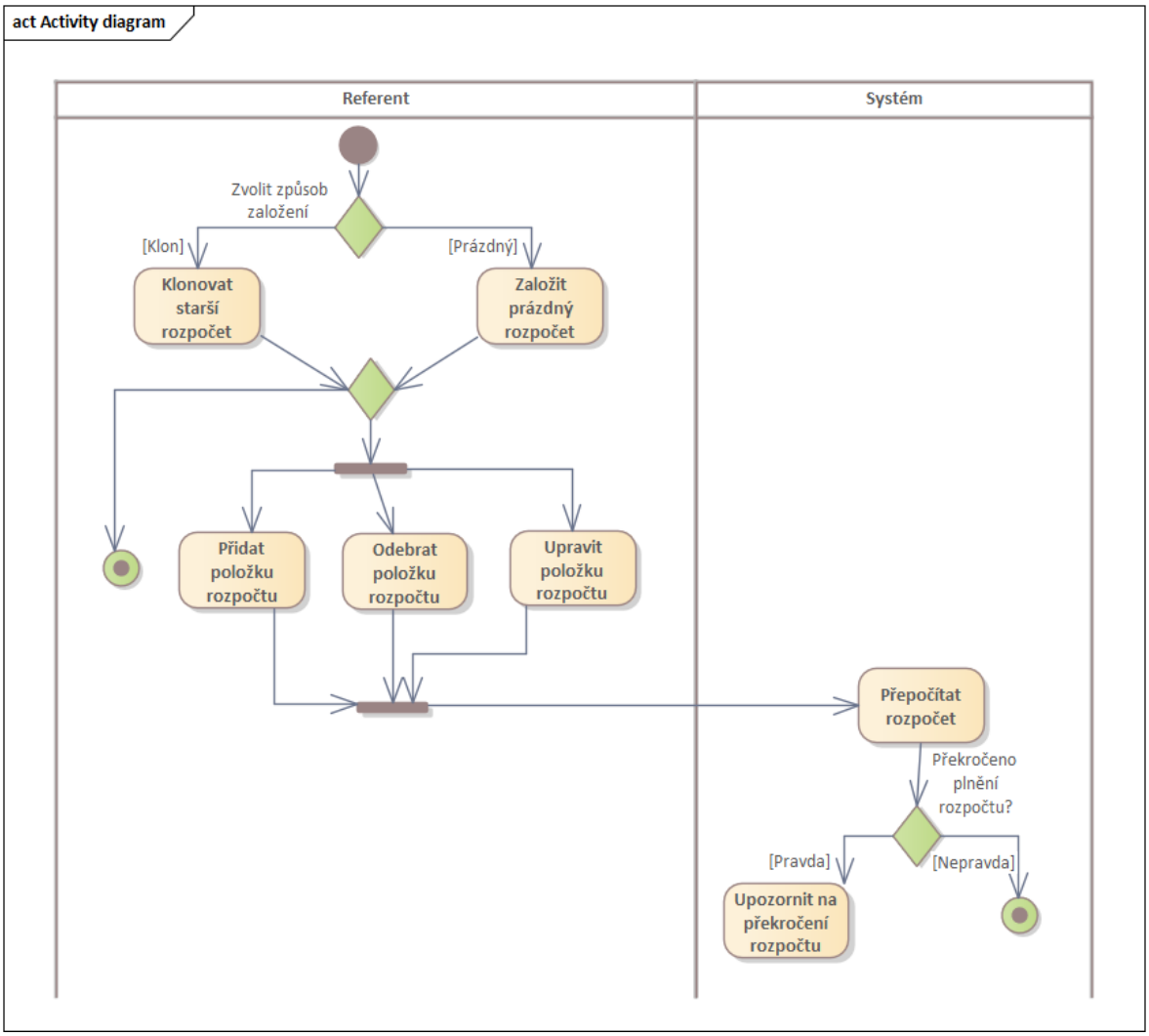
nadřízenou položku. Tímto způsobem je zajištěna hierarchie položek v souladu s rozpočtovou skladbou definovanou vyhláškou.

Každá položka má nejprve definovánu plánovanou hodnotu, později k ní přibude skutečná hodnota. Plánovaná hodnota může být buď příjmová, výdajová, nebo obojí (například u vydávání obecního zpravodaje jsou v položce kombinovány výdaje na tvorbu zpravodaje a příjmy z jeho prodeje a z inzerce).

Změna položky obsahuje původní hodnotu před změnou, samotnou změnu (navýšení či snížení) a hodnotu po změně. Tento způsob modelování přináší poměrně velkou redundanci, výrazně ale usnadňuje zobrazování dat a základní výpočty nad nimi.

4.3.5 Diagram aktivit

Obrázek 22 znázorňuje diagram aktivit pro založení nového rozpočtu.



Obrázek 22: Diagram aktivit pro založení rozpočtu

Založení rozpočtu, stejně jako jeho celou administraci, vykonává referent. Rozpočet může být založen buď jako nový (prázdný) nebo (obvykleji) jako klon některého předešlého (nejčastěji loňského) rozpočtu. Po založení rozpočtu má referent možnost rozpočet upravovat přidáváním, odebíráním a modifikací jednotlivých položek.

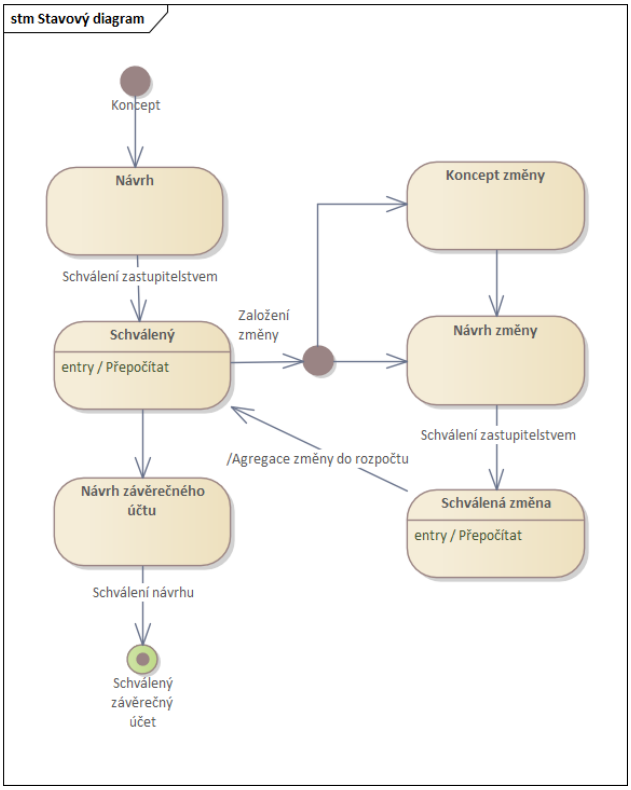
V případě manipulace s položkami, je zapotřebí rozpočet přepočítat, což představuje:

- přepočet všech nadřazených položek (položky jsou ve vzájemné hierarchii),
- přepočet celkových příjmů a výdajů (což je de facto tatáž operace jako předcházející, ale prováděná za rozpočet jako celek),

- kontrolu případného překročení plnění rozpočtu – jde o případ, kdy skutečné výdaje překračují plánované výdaje, což je z hlediska zákona nežádoucí situace, která vyžaduje přípravu změny rozpočtu. V případě založení nového rozpočtu ještě samozřejmě nedochází k jeho reálnému plnění, a tedy neexistují žádné skutečné výdaje, a tím pádem nemůže být plánovaný rozpočet překročen. Výše popsaný diagram aktivit je však s drobnou obměnou zamýšlen k použití i pro jakékoliv pozdější úpravy rozpočtu, k nimž může docházet i v průběhu roku.

4.3.6 Stavový diagram rozpočtu

Obrázek 23 znázorňuje stavový diagram rozpočtu.



Obrázek 23: Stavový diagram rozpočtu

Rozpočet na daný rok začíná vždy konceptem, kdy probíhají jeho první přípravy, zpravidla interně v rámci obecního úřadu. Následně se rozpočet uvede do stavu *Návrh,* který

je publikován na úřední desce. Po schválení zastupitelstvem je rozpočet ve stavu *Schválený*. Následně mohou probíhat změny rozpočtu, přičemž každá změna začíná opět buď konceptem nebo v tomto případě (v rámci zjednodušeného procesu) přímo návrhem. Schválená změna je zapracována do rozpočtu.

Po skončení roku je připraven a zveřejněn návrh závěrečného účtu a ten je nakonec schválen zastupitelstvem, čímž životní cyklus rozpočtu pro daný rok končí.

4.4 Samospráva

4.4.1 Stručný popis modulu

Modul *Samospráva* slouží k administraci činnosti samosprávy, zejména zastupitelstva obce, ale i dalších orgánů. Předmětem modulu je jednak evidence samotných samosprávných orgánů a jejich složení, jednak jejich jednání. Klíčovou funkcionalitou je administrace zasedání zastupitelstva, která jsou ze zákona veřejná a jedním z cílů modulu je zajistit transparentnost jednání zastupitelstva ve vztahu k široké veřejnosti.

4.4.2 Funkční požadavky

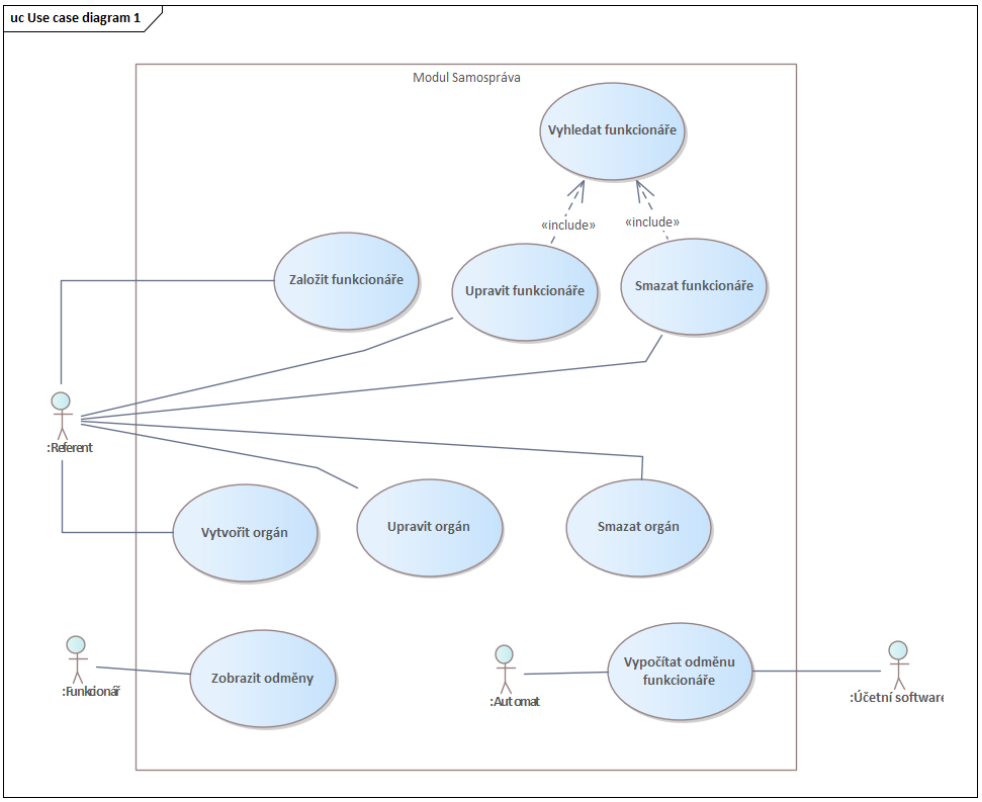
ID	*Název*	*Popis*	*Priorita*
SMS-RQ-01	Evidence orgánů a členů	Systém zajistí evidenci jednotlivých orgánů samosprávy a jejich složení v čase. V prvé řadě jde o zastupitelstvo obce, dále o výbory zastupitelstva, komise, ale i o pracovníky zařazené pod obecní úřad.	M
SMS-RQ-02	Evidence odměn	Systém umožní evidenci odměn funkcionářů.	M
SMS-RQ-03	Výpočet odměn	Bude automaticky odměny vypočítávat podle obecných pravidel.	M

ID	*Název*	*Popis*	*Priorita*
SMS-RQ-04	Prezentace odměn	Funkcionáři budou mít v systému k dispozici přehled odměn.	S
SMS-RQ-05	Předávání odměn	Přehled odměn funkcionářů bude automaticky předáván do účetního systému obce.	S
SMS-RQ-06	Evidence jednání	Systém umožní evidenci jednání orgánů obce.	M
SMS-RQ-07	Body programu	Ke každému jednání budou evidovány body programu a jejich podklady (přílohy).	M
SMS-RQ-08	Záznam	Systém umožní uchovat a přehrát obrazově-zvukový záznam k danému jednání.	M
SMS-RQ-09	Tagování záznamu	Záznam může být referentem obecního úřadu „otagován“, aby bylo možné přiřadit konkrétní pasáž záznamu konkrétnímu bodu programu.	S
SMS-RQ-10	Anonymizace záznamu	Vybranou část záznamu musí být možné anonymizovat, tzn. nahradit příslušnou zvukovou pasáž „pípáním“, aby byla zajištěna ochrana osobních údajů.	M
SMS-RQ-11	Pohodlné přehrávání	Systém umožní v záznamu „přeskočit“ na požadovaný bod programu.	S
SMS-RQ-12	Účast na jednání	Systém umožní evidenci účasti jednání funkcionářů na jednání orgánu.	M

ID	Název	Popis	Priorita
SMS-RQ-13	Tvorba zápisu	Systém umožní zapisovateli vytvořit zápis jednotlivých bodů programu a na základě těchto dílčích zápisů sestavit jeden konzistentní zápis, který bude možné ze systému vyexportovat ve formátech DOCX a PDF.	M

4.4.3 Diagram případů užití

Pro účely modulu *Samospráva* byly z důvodu přehlednosti navrženy dva diagramy případů užití: diagram pro práci s funkcionáři a orgány (Obrázek 24) a diagram pro práci s jednáními (Obrázek 25).



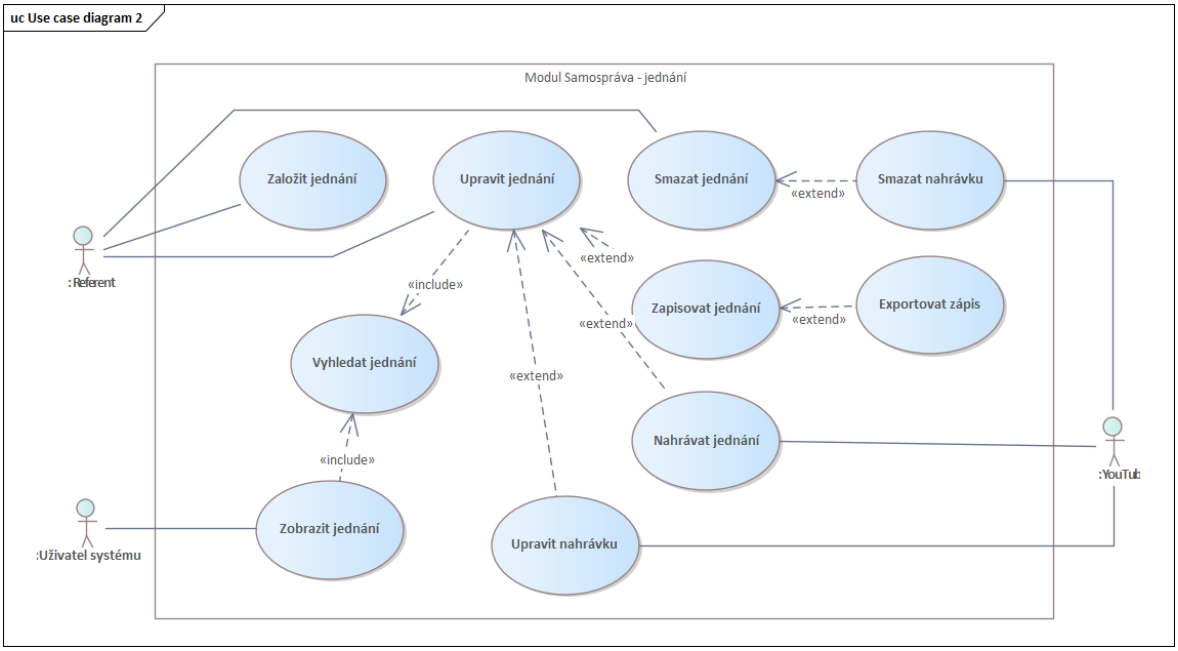
Obrázek 24: Diagram případů užití – funkcionáři a orgány

Aktérem pro správu funkcionářů je *Referent*. Referent může zakládat nové funkcionáře, provádět jejich úpravu nebo je mazat. Úpravě či smazání předchází vyhledání příslušného funkcionáře v seznamu.

Referent dále provádí potřebnou administraci orgánů a jejich obsazování funkcionáři.

Automat zajišťuje pravidelný výpočet odměn funkcionářů podle předpisu měsíčních odměn. Vypočítané odměny jsou předávány ve formě datové věty účetnímu softwaru obce.

Funkcionář má možnost zobrazit si své vypočítané odměny.



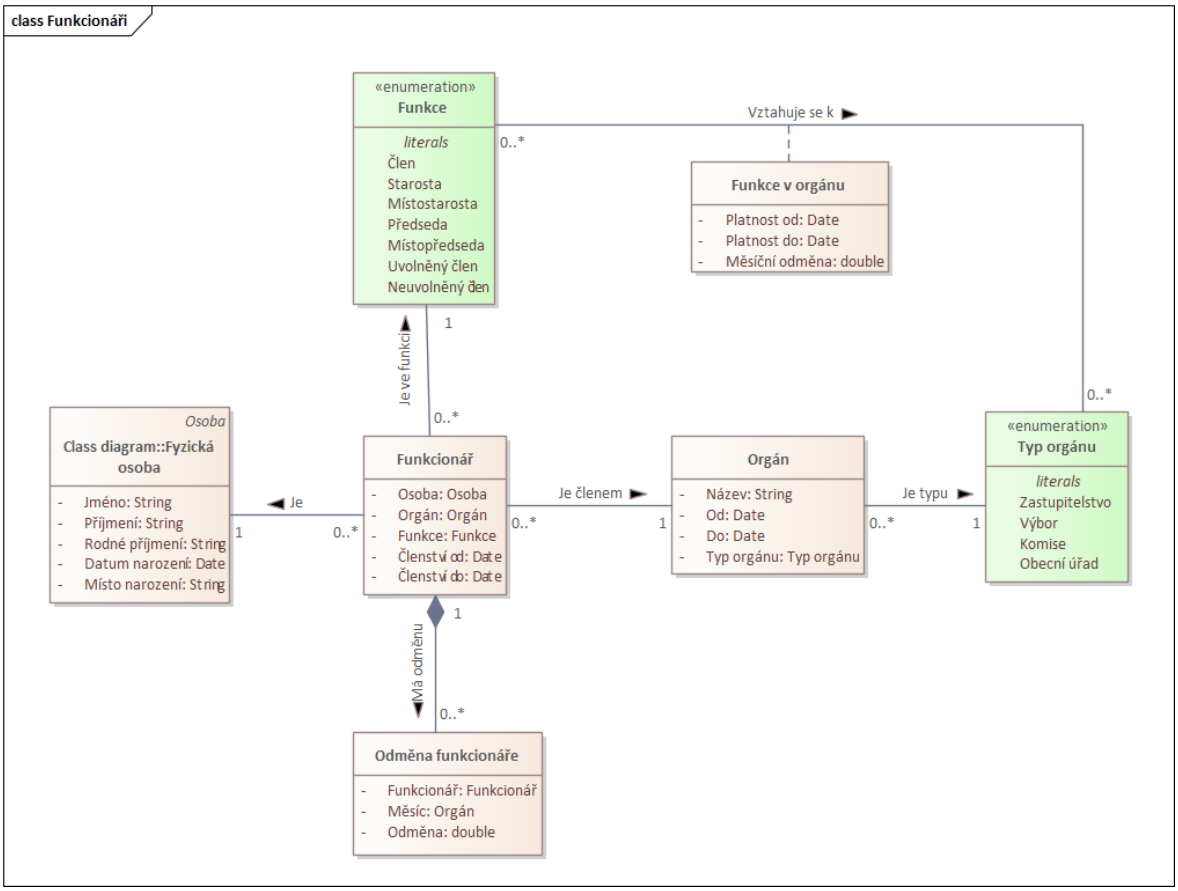
Obrázek 25: Diagram případů užití – jednání

Správu jednání zajišťuje rovněž referent. Zakládá jednání, provádí jejich potřebné úpravy. V rámci úpravy jednání může provádět též zápis nebo jednání nahrát (má-li k počítači připojeno potřebné nahrávací zařízení). Nahrávku může dodatečně upravit (tagování, anonymizace). Nahraný záznam se ukládá na platformu YouTube pomocí Data API.

4.4.4 Diagram tříd

Vzhledem k vyššímu rozsahu modulu je model tříd rozdělen do dvou diagramů – diagram *Funkcionáři* a diagram *Jednání.*

Diagram *Funkcionáři* znázorňuje Obrázek 26.



Obrázek 26: Diagram tříd Funkcionáři modulu Samospráva

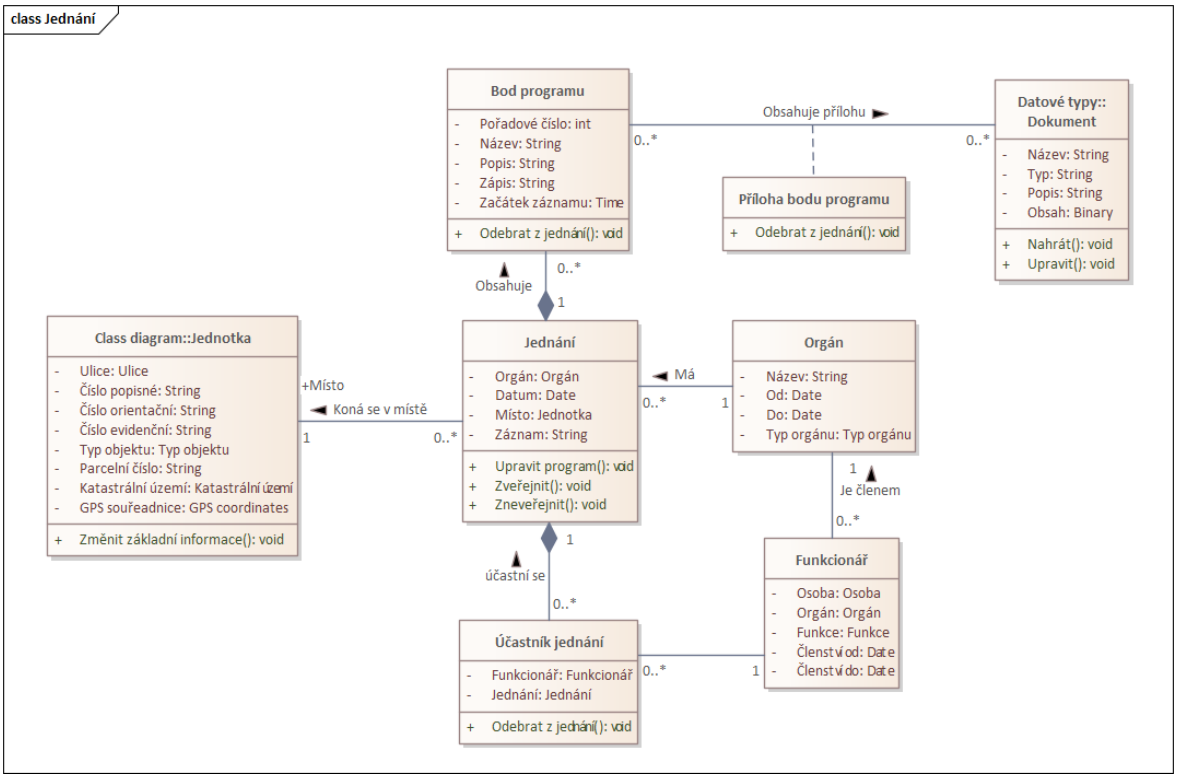
Klíčovou třídou diagramu je *Funkcionář,* která definuje konkrétního funkcionáře ve vztahu k dalším třídám. Funkcionář je vztažen k určité fyzické osobě (nemůže z ní být však zděděn, protože funkce má časovou platnost a je tedy spíše asociací než specializací) a zároveň je členem nějakého orgánu v určité funkci. Trojice fyzická osoba–orgán–funkce má společně stanovenou časovou platnost.

Další důležitou třídou je *Orgán,* kterým může být například zastupitelstvo, výbor či komise. Orgán je určitého typu (UML enumeration – výčet). Tento způsob namodelování je zcela záměrný, protože s výjimkou obecního úřadu mají orgány platnost časově vymezenou volebním obdobím zastupitelstev obcí. To je zajištěno evidencí platnosti od/do daného orgánu. Zároveň je ale třeba samostatně udržovat evidenci členství konkrétní osoby v orgánu, neboť členství nemusí pokrývat celou platnost existence orgánu. Takto navržený model umožní zachytit i relativně extrémní případ, kdy určitá osoba po nějakou dobu působí v určitém orgánu, poté v něm ukončí činnost a posléze se znovu stane jeho členem (v rámci téhož volebního období).

Diagram dále znázorňuje asociační třídu *Funkce v orgánu,* která omezuje přípustné kombinace orgánů a funkcí. Například tedy může existovat uvolněný člen zastupitelstva či neuvolněný člen zastupitelstva, ale už ne třeba předseda zastupitelstva. Přípustné kombinace pak kontrolují případy užití *Založit funkcionáře* a *Upravit funkcionáře.* Asociační třída je zároveň využita pro evidenci měsíčních odměn funkcionářů v daných orgánech. Na základě těchto „sazebníků“ se pak v rámci případu užití *Vypočítat odměnu funkcionáře* vypočítávají konkrétní měsíční odměny funkcionářů za jejich činnost.

Právě z důvodu uvedeného v předcházejícím odstavci nelze třídu *Funkcionář* přímo propojit s asociační třídou *Funkce v orgánu,* protože tato třída nese i informaci o měsíční odměně, která se může v čase měnit, aniž by to mělo mít jakýkoliv dopad na samotné členství dané osoby v orgánu.

Diagram *Jednání* znázorňuje Obrázek 27.



Obrázek 27: Diagram tříd Jednání modulu Samospráva

Druhým základním kamenem modulu *Samospráva* je třída *Jednání*. které ve formě atributů a asociovaných tříd zajišťuje evidenci podstatných informací o jednáních samosprávných orgánů obce.

Pro každé jednání je v systému evidován orgán, který jednání vede, datum a místo konání (místo je reprezentováno odkazem na jednotku, viz kapitola 4.2.3), a dále volitelně zvukově-obrazový záznam. V třídovém modelu je příslušný atribut typu *String*, systém ale předpokládá (a omezuje prostřednictvím případů užití, viz kapitola 4.4.3), že zde bude uložena URL adresa zvukového nebo obrazového záznamu, typicky na serveru YouTube.

Ke třídě *Jednání* se dále vztahuje (prostřednictvím kompozitní vazby) třída *Účastník jednání,* v níž je evidováno, kdo se jednání zúčastnil. Zvolený způsob modelování umožňuje, aby byl jako účastník jednání uveden i funkcionář, který není přímo členem jednajícího orgánu (například pokud se zúčastní jako host).

Jednání se skládá z bodů programu. Body programu lze zaevidovat ještě před samotným jednáním, kdy představují plánovaný program, a po skončení jednání (či ještě lépe online v jeho průběhu) lze pomocí atributu *Zápis* zapisovat, co bylo v rámci daného bodu programu projednáno. Model současně umožňuje ke každému bodu programu připojit libovolné množství příloh, ve formě odkazů na třídu *Dokument* (viz kapitola 4.2.4). V neposlední řadě lze u každého bodu programu evidovat počáteční čas příslušné sekvence audiovizuálního záznamu z příslušného jednání, což následně příslušnému případu užití (*Zobrazit jednání,* viz kapitola 4.4.3) umožní spustit uživateli záznam od konkrétního místa dle jeho zájmu.

4.5 Úřední deska

4.5.1 Stručný popis modulu

Modul slouží k administraci záznamů na úřední desce. Práce s úřední deskou je upravena zejména zákonem č. 500/2004 Sb., správní řád (ukládá povinnosti mít úřední desku a publikovat na ní dokumenty), a zákonem č. 106/1999 Sb. o svobodném přístupu k informacím (upravuje povinnost publikace úřední desky ve formě otevřených dat).

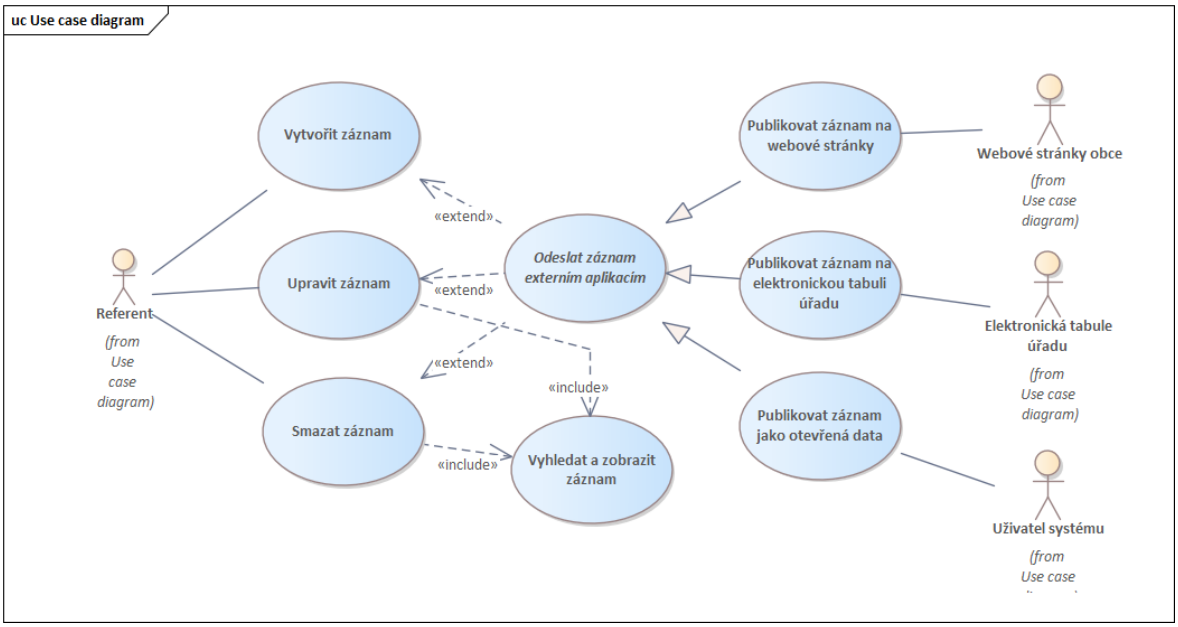
Modul umožňuje jednoduchou administraci záznamů na úřední desce – vytváření, úpravu a mazání. Záznamy se zveřejňují na vybraných místech v závislosti na tom, s jakými publikačními místy je systém integrován. Typicky se může jednat o webové stránky, elektronickou tabuli obecního úřadu nebo webový server, kde jsou záznamy úřední desky publikovány ve formě otevřených dat.

4.5.2 Funkční požadavky

ID	*Název*	*Popis*	*Priorita*
UDS-RQ-01	Správa záznamů	Systém umožní správu záznamů na úřední desce. Ke každému záznamu je evidován den zveřejnění a svěšení a dále dokumenty vztažené k záznamu.	M
UDS-RQ-02	Publikace záznamů – web a tabule	Systém umožní publikaci záznamů na webových stránkách obce a elektronické tabuli na budově obecního úřadu	M
UDS-RQ-03	Publikace záznamů – otevřená data	Systém umožní publikaci záznamů ve formě otevřených dat.	S

4.5.3 Diagram případů užití

Diagram případů užití znázorňuje Obrázek 28.



Obrázek 28: Diagram případů užití modulu Úřední deska

Základní manipulace s úřední deskou je velmi jednoduchá: Referent může prostřednictvím případu užití *Založit záznam* založit nový záznam na úřední desku. Existující záznam lze upravit pomocí případu užití *Upravit záznam,* případně záznam smazat pomocí případu užití *Smazat záznam.* Oběma těmto případům užití předchází vyhledání a zobrazení záznamu pomocí případu užití *Vyhledat a zobrazit záznam.*

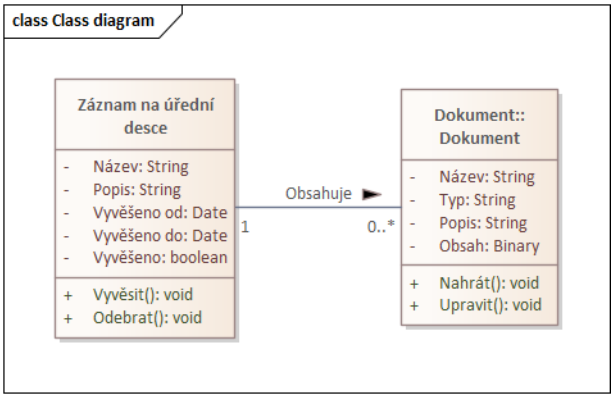
Na úpravu záznamu je pak navázán abstraktní případ užití Odeslat záznam externím aplikacím, který může být implementován řadou způsobů. V této verzi návrhu jsou podporovány tři:

- *Publikovat záznam na webové stránky* – přenese změněný záznam na webové stránky obce,
- *Publikovat záznam na elektronickou tabuli úřadu* – přenese změněný záznam na elektronickou tabuli úřadu (ve skutečnosti jde o jednoduchou aplikaci, která publikuje záznamy obvykle na monitoru umístěném na budově obecního úřadu),

- *Publikovat záznam jako otevřená data* – transformuje záznam do formátu XML a přidá jej na určené místo, odkud může být využíván aplikacemi třetích stran.

4.5.4 Diagram tříd

Diagram tříd je v základní verzi velmi jednoduchý, jak znázorňuje Obrázek 29.



Obrázek 29: Diagram tříd modulu Úřední deska

Třída *Záznam na úřední desce* obsahuje všechny požadované atributy a dále je k ní pomocí asociace připojena třída *Dokument*. K jednomu záznamu lze přiřadit více dokumentů.

Boolean atribut *Vyvěšeno* je duplicitní vůči atributům *Vyvěšeno od* a *Vyvěšeno do* a slouží zejména ke snazšímu filtrování záznamů, které mají být v daný den považovány za vyvěšené. Dotazování s pomocí dvou atributů vyžaduje na úrovni databáze spojení dvou indexů, což zvyšuje složitost dotazu a zejména s přibývajícími záznamy může zpomalovat získání výsledků.

4.6 Služby a poplatky

4.6.1 Stručný popis modulu

Modul slouží k evidenci vybraných služeb poskytovaných občanům ze strany obce. Jednotlivé služby jsou zpoplatněny a každá služba má obecně jiný způsob vyúčtování.

Modul zajistí evidenci podkladových dat potřebných pro vyúčtování, výpočet vyúčtování pro jednotlivé služby a jejich zákazníky (obvykle vlastníky nemovitostí, ale nikoliv vždy), a předání vyúčtování zákazníkům, kteří mají přímý přístup do systému. Modul eviduje též uskutečněné platby zákazníků a evidenci zbývajících dlužných částek. Modul podporuje vyúčtování následujících služeb a poplatků:

- svoz odpadu,
- vodné a stočné,
- pronájem hrobových míst,
- poplatek za psa (zde nejde přímo o službu, poplatek se hradí za vlastnictví psa).

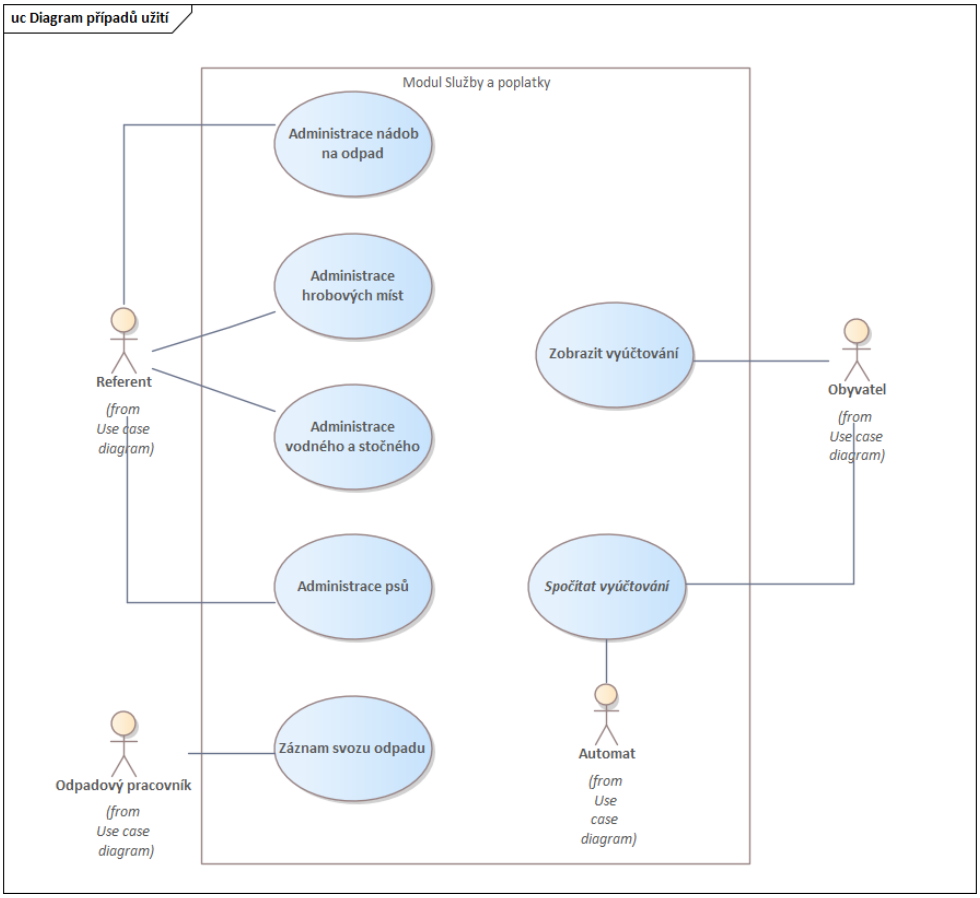
4.6.2 Funkční požadavky

ID	*Název*	*Popis*	*Priorita*
SPP-RQ-01	Výpočet poplatků	Systém podporuje výpočet poplatků za svoz odpadu, vodné a stočné, pronájem hrobových míst a vlastnictví psa, v souladu s platnou legislativou.	M
SPP-RQ-02	Podkladová data	Systém umožňuje evidenci všech podkladových dat, která jsou potřebná pro výpočet příslušných poplatků.	M
SPP-RQ-03	Informování zákazníků	Zákazníci jsou informováni o poplatcích za služby, které mají zaplatit.	M
SPP-RQ-04	Přístup zákazníků	Systém umožní přímý přístup zákazníků do systému, kde si budou moci prohlédnout aktuální vyúčtování včetně odkazu na přímou platbu.	S

ID	Název	Popis	Priorita
SPP-RQ-05	Přístup odpadového pracovníka	Odpadový pracovník bude schopen zaznamenávat uskutečněné svozy odpadů, a to prostřednictvím speciálního rozhraní, které umožní napojení systému přímo na svozový vůz, který pomocí RFID čipu rozpozná vyváženou nádobu a zjistí rovněž hmotnost odpadu v nádobě.	S

4.6.3 Diagram případů užití

Obrázek 30 znázorňuje diagram případů užití modulu *Služby a poplatky*.



Obrázek 30: Diagram případů užití modulu Služby a poplatky

Hlavním aktérem modulu je, podobně jako u ostatních modulů, referent. Jeho hlavní činností je administrace podkladů pro jednotlivé druhy vyúčtování, tzn. nádob na odpad, hrobových míst, vodného a stočného a vlastnictví psů. Každý takový případ užití je v diagramu znázorněn pouze souhrnně jako „administrace“, i když ve skutečnosti by patrně byla nezbytná dekompozice jednotlivých případů užití na menší celky. To by však bylo na úkor přehlednosti diagramu jako celku, proto se autor v tomto ohledu přidržel určité úrovně abstrakce.

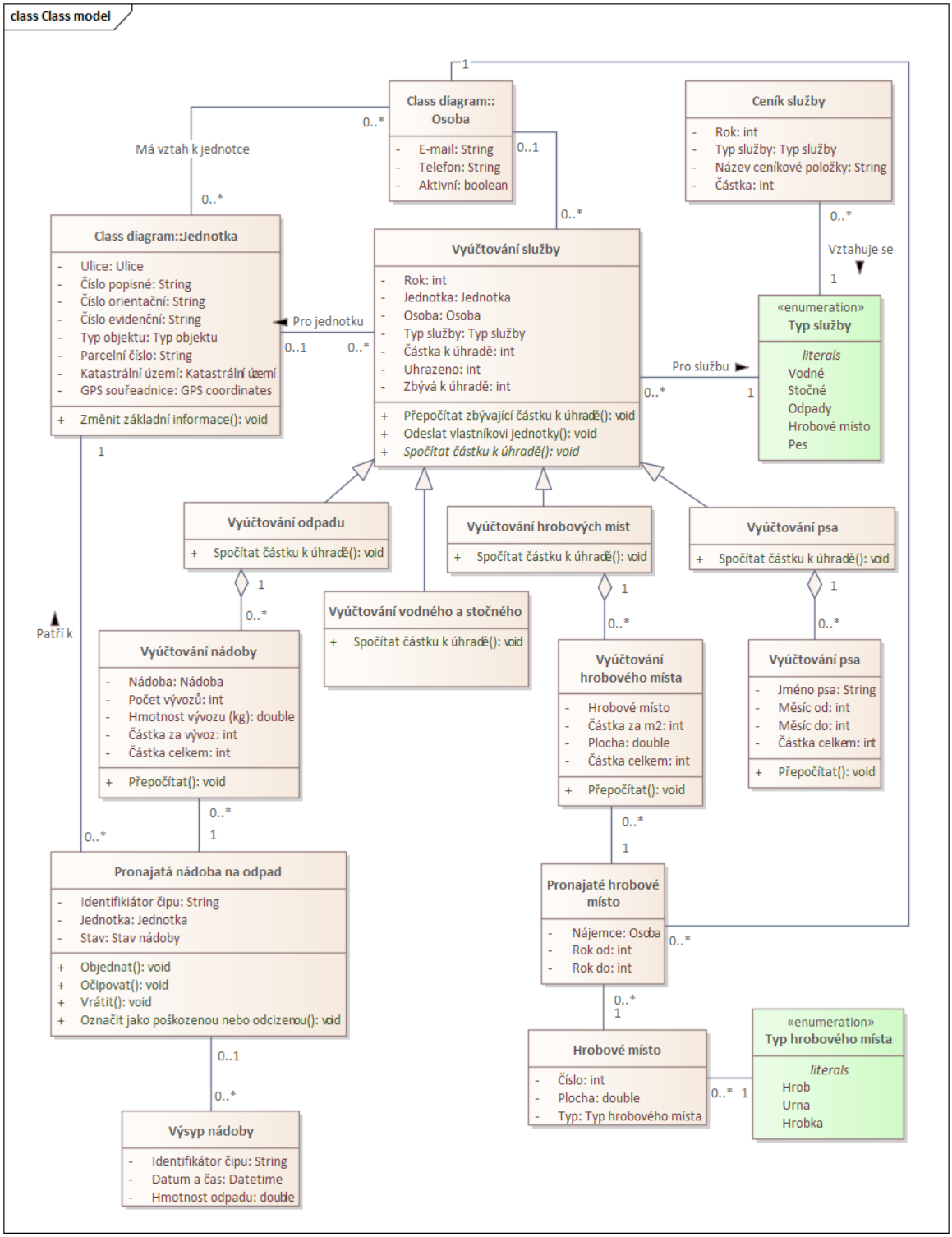
Specifickou roli v modulu zastává odpadový pracovník, který prostřednictvím specializovaného rozhraní může do systému zadat uskutečněný svoz, a to přímo ze svozového vozidla.

V předem definovaných intervalech, zpravidla jednou ročně, je provedeno vyúčtování jednotlivých typů poskytovaných služeb příslušným uživatelům (vlastníkům jednotek či dalším osobám). Vyúčtování zajišťuje případ užití *Spočítat vyúčtování,* který je volán speciálním aktérem *Automat* (funkce systému, která se sama spouští v nastavených časech). Jedná se o abstraktní případ užití (proto je v diagramu jeho titulek kurzívou), z něhož jsou zděděny výpočty pro jednotlivé typy služeb, neboť pro každý typ služby je výpočet obecně prováděn jinak. Toto dědění zde pro přehlednost není vizuálně znázorněno, je však dále zohledněno v diagramu tříd (viz kapitola 4.6.4). Pasivním aktérem tohoto případu užití je *Obyvatel,* který obdrží e-mailem provedené vyúčtování včetně instrukcí k platbě.

Obyvatelé mají rovněž přímý přístup do systému, v němž si mohou prohlížet provedená vyúčtování a uskutečněné platby, prostřednictvím případu užití *Zobrazit vyúčtování.*

4.6.4 Diagram tříd

Diagram tříd modulu Služby a poplatky znázorňuje Obrázek 31.



Obrázek 31: Diagram tříd modulu Služby a poplatky

Třída *Vyúčtování služby* obsahuje vyúčtování služeb za daný rok pro určitý typ služby. Podporované typy služeb jsou uvedeny ve výčtu *Typ služby.* (Je možné, že v rámci další detailizace návrhu by bylo zjištěno, že k jednotlivým službám je třeba evidovat doplňující informace, pak by stereotyp „výčet“ již nestačil a prvek by musel být upraven na běžnou třídu.)

Vyúčtování služby se vztahuje vždy buď k jednotce nebo k osobě, v závislosti na typu služby – většina služeb se vztahuje k nemovitosti, ale existují i výjimky (např. hrobová místa), kde se vyúčtování váže přímo na určitou osobu (např. nájemce hrobového místa). Prostřednictvím vazby mezi jednotkou a osobou v roli vlastníka jednotky je pak možné přiřadit konkrétní osobě i vyúčtování vztažené k jednotce. Autor v této souvislosti zvažoval, že přímo k osobě budou vztažena všechna vyúčtování bez rozdílu (čímž by bylo přímo dáno, kdo má příslušnou službu uhradit), vazbu na jednotku by však bylo přesto nutné ponechat a zároveň by bylo třeba ohlídat konzistenci vlastníka a jednotky v rámci jednoho vyúčtování.

Vyúčtování obsahuje abstraktní metodu *Spočítat částku k úhradě,* která je s využitím polymorfismu implementována až v potomcích třídy, specifických pro jednotlivé typy služeb. Návrh tak zohledňuje skutečnost, že výpočet se pro jednotlivé typy služeb obecně liší.

Diagram dále obsahuje třídy pro jednotlivé typy vyúčtování. V této verzi jsou zpracována vyúčtování pro odpady a pronájem hrobových míst. Vyúčtování pro vodné a stočné není zpracováno, neboť spolupracující obec Nelahozeves vodovod neprovozuje a způsob vyúčtování kanalizace procházel v době konzultací změnami a zadavatel nebyl schopen formulovat pro tuto oblast jednoznačné požadavky. Z tohoto důvodu model v této oblasti končí na úrovni potomka *Vyúčtování vodného a stočného* bez dalších podrobností.

Podpora vyúčtování odpadu vychází z platného zákona č. 541/2020 Sb. o odpadech. Zákon podporuje několik systémů plateb za odpady, z nichž byly v rámci konzultací vybrány dva reálně použitelné – platby podle počtu svozů nádoby a platby podle hmotnosti vyvezeného odpadu. Metoda *Přepočítat* v třídě *Vyúčtování nádoby* by měla podporovat oba

zmíněné mechanismy, pro něž jsou v třídě připraveny potřebné atributy. Alternativně by bylo možné zde použít opět polymorfismus. Vyúčtování nádoby je vázáno na konkrétní pronajatou nádobu (třída *Pronajatá nádoba na odpad)* a vychází z provedených výsypů, které jsou evidovány v třídě *Výsyp nádoby.* Vyúčtování nádoby pak bude provedeno buď na základě počtu výsypů nebo na základě celkové hmotnosti. Vazba mezi třídou *Výsyp nádoby* a třídou *Pronajatá nádoba na odpad* předpokládá i situaci, kdy je proveden výsyp nádoby, která není z jakéhokoliv důvodu registrována v systému (proto je násobnost na straně nádoby 0..1).

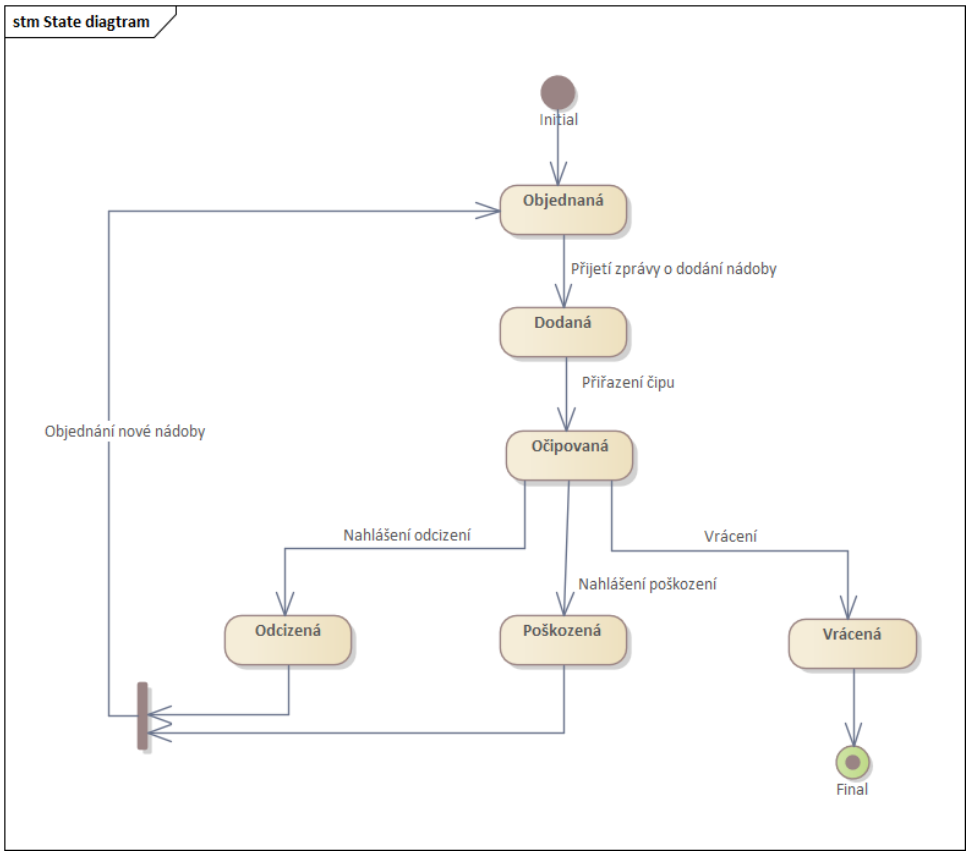
Problematika vyúčtování hrobových míst je výrazně jednodušší. Konkrétní hrobové místo (třída *Hrobové místo)* je jednoho ze tří podporovaných typů (třída stereotypu „výčet“) a lze si jej pronajmout na interval let. Na základě plochy daného hrobového místa je spočítán poplatek za daný rok. Problematiku upravuje zákon č. 256/2001 Sb. o pohřebnictví.

Ještě jednodušší je vyúčtování poplatků za psa. Poplatky se počítají na základě zákona 565/1990 Sb. o místních poplatcích, a to na základě počtu psů, přičemž zákon umožňuje určit jinou sazbu pro prvního psa a každého dalšího psa. Zdánlivě by tedy mělo stačit evidovat pro daného majitele nemovitosti počet psů v daném roce, ovšem zákon stanoví že „v případě trvání poplatkové povinnosti po dobu kratší než jeden rok se platí poplatek v poměrné výši, která odpovídá počtu i započatých kalendářních měsíců“. Z tohoto důvodu bylo zapotřebí navrhnout novou třídu *Pes,* v níž bude evidováno období poplatkové povinnosti za interval měsíců (1 až 12).

Poslední třídou navrženou v rámci diagramu je *Ceník služby* (na diagramu v pravém horním rohu). Ta umožňuje evidovat jednotkové ceny jednotlivých typů služeb podle jejich charakteru, vždy pro daný rok. Výpočty vyúčtování jednotlivých služeb pak využívají tento ceník.

4.6.5 Stavový diagram

Z hlediska toku stavů je užitečné znázornit stavový diagram odpadové nádoby – viz Obrázek 32.



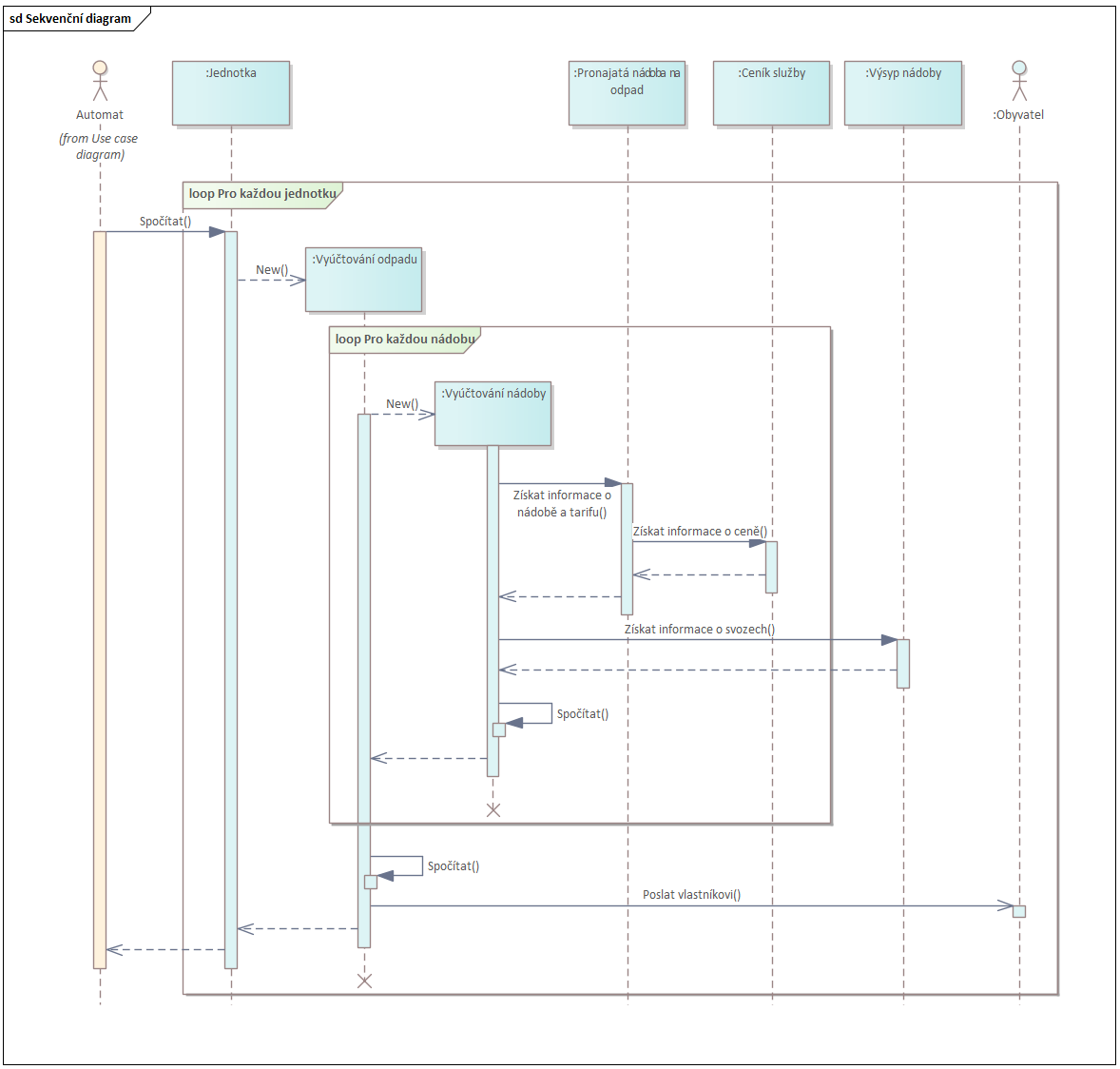
Obrázek 32: Stavový diagram pro nádobu na odpad

Odpadová nádoba je nejprve objednaná, tento stav zadává referent obecního úřadu. Následné stavy *Dodaná* a *Očipovaná* mohou být nastaveny zvnějšku, například mobilní aplikací odpadové firmy, která má implementováno rozhraní na systém. Ve stavu *Očipovaná* zůstává nádoba po dobu, kdy je využívána ke svozu odpadů. Standardní stavový tok končí vrácením nádoby, může ale dojít i k poškození nebo odcizení – v takovém případě referent obecního úřadu objednává nádobu novou.

4.6.6 Sekvenční diagram

Pro tento modul autor práce vyhodnotil jako přínosné navrhnout sekvenční diagram případu užití *Spočítat vyúčtování odpadu*, který je specializací abstraktního případu užití *Spočítat vyúčtování.* Výpočet vyúčtování odpadu je totiž, na rozdíl od ostatních typů služeb,

poněkud komplexnější úloha, je tedy užitečné ji alespoň v určité úrovni podrobnosti znázornit graficky. Sekvenční diagram znázorňuje Obrázek 33.



Obrázek 33: Sekvenční diagram případu užití Spočítat vyúčtování odpadu

Případ užití je spouštěn aktérem *Automat*. V rámci případu užití se pak v cyklu provádí výpočet pro všechny jednotky (byty, domy) v obci. Pro každou jednotku se vytvoří vyúčtování[31] a pro něj se pak opatřují potřebné informace. Nejprve je proveden výpočet pro

[31] V praxi by byl návrh o něco složitější, protože při požadavku na dobrý a robustní návrh systému musí být funkcionalita idempotentní, tzn. výpočet musí správně proběhnout i v případě, kdy je spuštěn

jednotlivé nádoby (k jedné jednotce může být evidováno více odpadových nádob). V rámci každé nádoby ne nejprve zjištěna cena svozu a dále jsou získány informace o provedených svozech. Poté je proveden samotný výpočet poplatku za danou nádobu a následně i za celou jednotku (součtem úhrad za jednotlivé nádoby). Na závěr je vyúčtování odesláno vlastníku nemovitosti. V tomto jediném případě je zpráva asynchronní (a je tedy použita obyčejná šipka a nikoliv šipka s vyplněným hrotem), protože jde o odeslání e-mailem a případ užití neočekává od obyvatele žádnou odezvu. Proto v tomto případě absentuje rovněž návratová zpráva.

opakovaně, přičemž při opakovaném spuštění by se již vyúčtování znovu nevytvářelo, ale aktualizovalo by se vyúčtování již dříve vytvořené.

5 Výsledky a diskuse

Tato kapitola se zabývá hodnocením samotné práce z pohledu autora a možnými alternativami řešení. Hodnocení je pochopitelně zatíženo subjektivním pohledem autora na vlastní práci, přesto se autor v této kapitole pokouší alespoň částečně o kritický odstup a nadhled.

Mezi klíčové faktory zhotovení správné a úplné analýzy a návrhu informačního systému není jen osoba analytika (modeláře), ale i přístup zákazníka. V tomto ohledu lze pozitivně hodnotit přístup (částečně fiktivního) zákazníka, referenční obce Nelahozeves, jejíž zástupci byli schopni v rozhovorech s autorem formulovat celou řadu konkrétních požadavků. Tyto požadavky byly následně konsolidovány a opatřeny prioritami. Na základě požadavků pak byly vytvořeny příslušné UML modely a jejich doprovodné specifikace, které byly následně znovu konzultovány se zákazníkem a iterativně zlepšovány.

V průběhu práce autor opakovaně řešil otázku, které modely a diagramy by měly být součástí návrhu a v jaké úrovni podrobnosti a s jak komplexními notacemi. Po úvaze byl zvolen přístup, kdy ke každému navrhovanému modulu byl vždy vytvořen minimálně diagram případů užití a diagram tříd. Další diagramy byly vytvářeny dle potřeby, tam kde to dávalo smysl a kde autor spatřoval vyšší přidanou hodnotu takových diagramů. Je možné, že kdyby na návrh navazovala vlastní implementace systému (což se v případě obce Nelahozeves může reálně stát, nicméně práce byla vytvářena nezávisle na případném budoucím rozhodnutí vedení obce ohledně implementace), bylo by nutné některé diagramy doplnit či případně některé existující diagramy zpřesnit.

Současně je zjevné, že v případě reálné analýzy a návrhu informačního systému by bylo nezbytné v některé projekční fázi navrhnout rovněž wireframy, čili návrhy obrazovek pro jednotlivé funkcionality. Návrh wireframů však není součástí jazyka UML a tato problematika tak přesahuje stanovený rozsah této práce.

Obecně se autor snažil řídit mottem britského softwarového inženýra Martina Fowlera, které formuloval v úvodu své knihy (využité i v této práci) *Destilované UML:*

„Od nejstarších dob znali nejtalentovanější architekti a nejnadanější designéři zákon šetrnosti. Ať už je uváděn jako paradox (‚méně je více‘) nebo jako koan (‚zenová mysl je mysl začátečníka‘), jeho moudrost je nadčasová. Omezit vše až na samu podstatu, a tím dosáhnout formy a funkce. Od pyramid k Opera House v Sydney, od von Neumannových architektur přes UNIX a Smalltalk, usilovali nejlepší architekti a designéři o následování všeobecných a věčných zásad. Uvědomuje si hodnotu řezu Occamovou břitvou, když navrhuji a čtu, hledám projekty a knihy zachovávající zákon šetrnosti.“

Diagramy prezentované v této práci jsou tedy vesměs jednoduché a intuitivní, což je dáno nejen omezeným rozsahem této práce, ale i snahou o zachování srozumitelnosti diagramů i pro čtenáře, kteří se v diagramech UML orientují jen částečně. Autorovi je z jeho praxe známo, že schopnost porozumět některým principům modelů může být poněkud komplikované nejen pro zákazníka, ale i pro některé návrháře a vývojáře. Pokročilejší modelovací techniky jsou samozřejmě užity tam, kde to je nezbytné.

Současně je třeba mít na zřeteli, že se jedná pouze o fiktivní projekt. Přestože spolupracující a referenční obec Nelahozeves má skutečně v úmyslu některé části aplikace perspektivně vyvinout, všechny diskuse o požadavcích na cílový systém byly vedeny s vědomím, že z nich nevyplývají žádné závazky ani pro zadavatele, ani pro řešitele. Je tedy možné, že při skutečně reálné analýze informačního systému by některé požadavky byly formulovány jinak, s jinými prioritami, a rovněž by bylo zohledňováno ekonomické hledisko, které při přípravě této práce zcela absentovalo. To se podepsalo i na výběru funkcionalit, které by mělo cílové řešení obsahovat (viz kapitola 3.2.3). Výběr provedl (po nezbytných konzultacích) sám autor práce, a přestože se při rozhodování snažil opřít o objektivizovaná a měřitelná kritéria, byl bezpochyby zatížen svým vlastním subjektivním pohledem na problematiku, a možná (přinejmenším podvědomě) zohledňoval i potenciální složitost analýzy a návrhu příslušné zvažované agendy. V reálném světě by o výběru agend rozhodoval zákazník ve spolupráci s odborným poradcem, kterým by, s cílem zamezení střetu zájmů, s vysokou pravděpodobností nebyla osoba, která je ekonomicky zainteresována na samotné dodávce řešení. Při výběru agend by byla bezpochyby zohledněna i stávající tržní situace, čili ještě před samotným rozhodnutím o vývoji nové

aplikace by byla podrobně analyzována řešení, která již na trhu aktuálně existují. Autor si je některých takových řešení vědom (například informační systém Munis[32]), cílem této práce ale nebyla rešerše dostupných řešení pro danou problematiku, která by bezpochyby sama o sobě mohla vydat na samostatnou diplomovou práci.

[32] https://www.munis.cz

6 Závěr

Tato diplomová práce si kladla za cíl analyzovat a navrhnout informační systém pro podporu agend malých obcí. Analýza a návrh byly v souladu s cíli práce provedeny v jazyce Unified Modelling Language (UML).

Modelovaný informační systém má sloužit k podpoře, zjednodušení a automatizaci vybraných agend malých obcí, které často provádějí řadu agend prostřednictvím tabulkového kalkulátoru či papírové formy. Důvodem volby tématu je skutečnost, že malé obce často nejsou personálně dostatečně vybaveny k tomu, aby efektivně zvládly všechny agendy, které jim svěřuje zákon či další podzákonné předpisy, a v případě použití informačního systému mohou dosáhnout kvalitativního zlepšení zpracování příslušných agend.

Při návrhu autor spolupracoval s vedením své domovské obce Nelahozeves, jehož starosta a místostarosta mu pomohli naformulovat základní požadavky na informační systém.

V první části práce byly teoreticky popsány postupy, standardy a nástroje analýzy a návrhu informačního systému, zejména nejlepší praktiky softwarového vývoje a základní principy modelování informačního systému pomocí jazyka UML. V rámci teoretické části se autor snažil držet faktů a citací použité literatury, v některých případech ale využil i svých znalostí a zkušeností z příslušné problematiky, zejména ve vazbě na kritické zhodnocení použitých zdrojů a postupů.

V rámci teoretické části byly rovněž pomocí transparentních postupů vybrány agendy, které by měly být součástí analyzovaného systému. Hlavním kritériem pro výběr agend byla vysoká přidaná hodnota, kterou obec jako provozovatel systému získá automatizací či obecně informační podporou dané agendy v podobě informačního systému.

V praktické části byla provedena samotná analýza a návrh informačního systému v podobě návrhu UML diagramů a jejich doprovodných popisů. Bylo identifikováno celkem šest oblastí, reprezentovaných moduly systému, které byly (při zohlednění vzájemných vazeb) analyzovány samostatně. Pro každý modul byl v prvé připraven katalog požadavků,

na jehož tvorbě se výrazně podíleli představitelé spolupracující a referenční obce Nelahozeves. Na základě katalogu požadavků byl sestaven diagram případů užití a diagram tříd. Tam, kde to bylo vhodné a účelné, byly připraveny další diagramy, jako je diagram aktivit, stavový diagram nebo sekvenční diagram. Při rozhodování, které diagramy budou v daném kontextu vytvořeny, byly vždy zvažovány přínosy daného diagramu pro zvolené řešení, či, jinými slovy, jejich přidaná hodnota.

S výslednými diagramy a jejich doprovodnými popisy byli seznámeni představitelé obce Nelahozeves a na základě jejich zpětné vazby byly některé aspekty diagramů aktualizovány a doplněny.

Zhotovený návrh aplikace je způsobilý pro další detailizaci a následný vývoj.

7 Seznam použitých zdrojů

ARLOW, Jim a Ila NEUSTADT. 2007. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press. ISBN 978-80-251-1503-9.

BOOCH, Grady, James RUMBAUGH a Ivar JACOBSON. 2005. *The Unified Modelling Language User Guide*. 2. Upper Saddle River, New Jersey, USA: Pearson Education. ISBN 0-321-26797-4.

BRUCKNER, Tomáš, Jiří VOŘÍŠEK, Alena BUCHALCEVOVÁ, Iva STANOVSKÁ, Dušan CHLAPEK a Václav ŘEPA. 2012. *Tvorba informačních systémů: principy, metodiky, architektury*. 1. Praha: Grada. Management v informační společnosti. ISBN 978-802-4741-536.

CONSTANTINE, Larry. 1968. *Control of Sequence and Parallism in Modular Programs*. 1. AFIPS '68 (Spring): Proceedings of the April 30--May 2, 1968, spring joint computer conference. Dostupné na internete: https://www.semanticscholar.org/paper/Control-of-sequence-and-parallelism-in-modular-Constantine/8cc389626af8a8e02c40f595528e308d776e4fb9

FOWLER, Martin. 2009. *Destilované UML*. 1. Praha: Grada. Knihovna programátora (Grada). ISBN 978-80-247-2062-3.

KADLEC, Václav. 2004. *Agilní programování: metodiky efektivního vývoje softwaru*. 1. Brno: Computer Press. ISBN 80-251-0342-0.

Katalogy agend a činností veřejné správy. 2009. *Ministerstvo vnitra* [online]. Praha: Ministerstvo vnitra [cit. 2022-01-18]. Dostupné na internete: https://www.mvcr.cz/sluzba/docDetail.aspx?docid=21229859&docType=ART

KOČÍ, Roman. 2012. *Obecní samospráva v České republice: praktická příručka s judikaturou*. 1. Praha: Leges. Praktik (Leges). ISBN 978-80-87576-28-1.

KRAVAL, Ilja. 2010. *Analytické modelování informačních systémů pomocí UML v praxi*. 1. Lipina: Object Consulting. ISBN 978-80-254-6986-6.

KRISHNAMURTHY, Ganesh. 1995. CASE Tools: Adoption and Relevance. *University of Missouri–St. Louis* [online]. St. Louis: University of Missouri [cit. 2022-02-07]. Dostupné na internete: https://www.umsl.edu/~sauterv/analysis/F08papers/View.html

PAGE-JONES, Meilir. 2001. *Základy objektově orientovaného návrhu v UML*. 1. Praha: Grada. Moderní programování. ISBN 80-247-0210-X.

RAMAKRISHNAN, Sreekanth. 2012. System Analysis and Design. *Journal of Information Technology & Software Engineering* [online]. **02**(05) [cit. 2021-11-20]. DOI: 10.4172/2165-7866.S8-e001. ISSN 21657866. Dostupné na internete: http://www.omicsgroup.org/journals/system-analysis-and-design-2165-7866.S8-e001.php?aid=12686

Softwarové inženýrství. 2001. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-11-20]. Dostupné na internete: https://cs.wikipedia.org/wiki/Softwarov%C3%A9_in%C5%BEen%C3%BDrstv%C3%AD#Softwarov%C3%A1_krize

Účelová právní analýza samosprávných kompetencí obcí v oblasti plánování a realizace rozvojových aktivit obcí. 2012. *Ministerstvo pro místní rozvoj* [online]. Praha: Ministerstvo pro místní rozvoj [cit. 2022-01-18]. Dostupné na internete: http://www.mmr.cz/getmedia/def4e459-425d-44d8-bde1-062e3ba1c737/1-a2-ucelova-pravni-analyza_ariel

VRANA, Ivan. 2008. *Projektování informačních systémů s UML*. Praha: Česká zemědělská univerzita, Provozně ekonomická fakulta. ISBN 978-80-213-1817-5.

WIEGERS, Karl Eugene. 2008. *Požadavky na software*. Brno: Computer Press. ISBN 978-80-251-1877-1.

8 Seznam obrázků

9 Přílohy

[A] Soubor UML diagram použitých v této práci ve formátu qea (Enterprise Architect)