AMBIS vysoká škola, a.s.

Katedra ekonomie a managementu

Metody analýzy a návrhu informačních systémů

Bakalářská práce

Autor: **Karel Růžička**

Ekonomika a management

Vedoucí práce: **doc. RNDr. Juraj Pančík, CSc.**

Praha **2023**

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a v seznamu uvedl veškerou použitou literaturu.

Svým podpisem stvrzuji, že jsem seznámen se skutečností, že práce bude zpřístupněna třetím osobám prostřednictvím informačního systému AMBIS vysoké školy, a.s.

V Praze, dne ……. Karel Růžička

Poděkování

Na tomto místě bych rád poděkoval doc. RNDr. Jurajovi Pančíkovi, CSc. za inspiraci, ochotu a trpělivost, kterou mi věnoval. Dále bych chtěl poděkovat personálu studijního oddělení a rektorce Dr. Martině Mannové za umožnění dodatečného odevzdání práce.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Akademický rok: 2020/2021

Student:	Karel Růžička
UČO:	38825
Program:	Ekonomika a management
Studijní obor:	Ekonomika a management podniku
Téma:	Metody analýzy a návrhu informačních systémů
Topic:	Methods of analysis and design of information systems
Vedoucí bakalářské práce:	doc. RNDr. Juraj Pančík, CSc.
Cíl práce:	**Cíl práce:** Cílem práce je navrhnout konkrétní informační systém s využitím metod analýzy a návrhu informačních systému, s využitím nástroje CASE, modelovacího jazyka UML a procesního modelovaní v jazyce BPMN. Navrhnutý informační systém bude použit pro výukové účely. **Použité metody:** Rešerše odborné literatury, analýza informačního systému. **Struktura práce:** Úvod Teoretická část - vymezení pojmů IS, UML, BPMN, CASE Analytická část - analýza informačního systému, vytvoření vývojové dokumentace Návrhová část - případová studie analýzy, návrh IS pro výukové účely Závěr
Základní prameny a odborná literatura:	ÖZKAYA, Mert. ANALYSING UML-BASED SOFTWARE MODELLING LANGUAGES. *Journal of Aeronautics & Space Technologies*. online, 2018, s. 119-133, 11 s. ISSN 1304-0448. DEITEL, Paul a Harvey DEITEL. *C# 2010 for Programmers*. 4. vyd. New Jersey: Prentice Hall, 2010. ISBN 978-0-13-265739-6. SCHMULER, Joseph. *Myslíme v jazyku UML : knihovna programátora*. 1. vyd. Praha: Grada Publishing, 2001. 359 s. ISBN 80-247-0029-8. ARLOW, Jim a Ila NEUSTADT. *UML 2 and the Unified Process*. Computer Press, 2007. 567 s. 2.vydání. ISBN 978-80-251-1503-9. KANISOVÁ, Hana a Miroslav MÜLLER. *UML srozumitelně*. Vyd. 1. Brno: Computer Press, 2004. 158 s. ISBN 8025102319.

Anotace

Bakalářská práce se v teoretické části zaměřuje na vymezení základních pojmů souvisejících s vývojem informačních systémů. Důraz je kladen na životní cyklus softwarových aplikací, obecný vizuální modelovací jazyk UML a procesně orientovaný jazyk BPMN. Cílem je navrhnout konkrétní informační systém s využitím metod analýzy a návrhu informačních systémů. Navrhnutý informační systém bude sloužit pro výukové účely.

Klíčová slova

informační systém, UML, BPMN, analýza a návrh, C#, objektově orientovaný

Annotation

In the theoretical part, the bachelor's thesis focuses on the definition of basic terms related to the development of information systems. Emphasis is placed on the life cycle of software applications, the general visual modeling language UML, and the process-oriented language BPMN. The goal is to design a specific information system using methods of analysis and design of information systems. The proposed information system will be used for educational purposes.

Key words

information systém, UML, BPMN, analysis and design, C#, object oriented

Obsah

Úvod

Cílem bakalářské práce je navržení konkrétního informačního systému s využitím metod analýzy a návrhu informačních systémů, s využitím nástroje CASE a modelovacího jazyka UML. Navrhnutý informační systém bude použit pro výukové účely.

Práce je členěna na tři hlavní části: teoretická, analytická a návrhová část. Na začátku teoretické části jsou vymezeny relevantní pojmy, se kterými se pracuje v dalších částech práce. Součástí teoretické části je kapitola životní cyklus, která popisuje několik přístupů k vývoji aplikací a vývojové postupy. Dále jsou v teoretické části popsány vybrané aspekty objektově orientovaného modelovacího jazyka UML a procesně orientovaného jazyka BPMN.

V analytické části je čtenář obeznámen s dokumentem požadavků budoucího informačního systému. V této části jsou podrobně popsány jednotlivé části systému a jsou definovány funkční a nefunkční požadavky. Následuje analýza informačního systému pomocí postupů a diagramů modelovacího jazyka UML

Cílem návrhové části práce je implementovat systém ve vybraném programovacím jazyku. Vzhledem k nedostatku vhodných příkladů pro výukové účely budou postupy vybrané části zdrojových kódů popsány.

1 Vymezení pojmů

1.1 Informační systém

Dříve než můžeme definovat informační systém, je důležité definovat systém jako takový. Definic systému existuje několik. Dle slovníku anglického jazyka dictionary.com (System Definition & Meaning) slovo systém znamená *„soubor věcí, které spolupracují jako součást mechanismu“* či *„komplexní celek“*.

Další definice z knižních zdrojů definují systém jako *„celek, tvořený jednak svou celistvostí (danou obvykle cílem či účelem) a jednak souhrnem částí (komponentů, prvků) a jejich vzájemným, často dynamických vztahů (interakcí)“* (Bruckner, 2012). Gála (2009) systém definuje jako *„účelově definovaná neprázdná množina prvků a množina vazeb mezi nimi, přičemž vlastnosti prvků a vazeb mezi nimi určují vlastnosti (chování) celku.“*

K dispozici jsou definice od mnoha autorů, které se však příliš neliší. Můžeme tedy říct, že se definice shodují na tom, že systém je celek, soubor věcí, či množina prvků. Tyto prvky nejsou nahodilé, ale jsou mezi sebou provázané a spolupracují ke společnému cíli.

Pokud hovoříme o systému v prostředí informatiky, nazýváme jej informačním systémem. Účelem IS je zajistit vhodné vyjádření informací, zpracování těchto informací a přenos v rámci daného systému. IS je tvořen lidmi, vhodnými nástroji a metodami, které se seskupují do tří hlavních komponent (Gála, 2009).

1.1.1 Komponenty

Jak již vyplívá z definice, IS se skládá z komponent či prvků. *„Komponenta je tvořena jedním nebo více prvky“* (Gála, 2015). Většina autorů udává 5 základních komponent, avšak ne všichni se shodují v tom, které to jsou. Některé zdroje uvádí 6, nebo i 7 komponentů. Pro účely této práce se obeznámíme s následujícími komponenty: lidé, (transformační) proces, informační technologie (hardware, software, data, telekomunikace).

Lidská složka (Peopleware) – Při úvahách o informačních systémech je snadné se zaměřit na technologické komponenty a zapomenout se podívat i mimo tyto nástroje, abyste plně pochopili jejich integraci do firmy. Gála (2015) rozděluje tento prvek na dvě základní kategorie – uživatelé a IT personál. Uživatelé jsou vně systému, využívají jeho výsledků (informací) a také se podílejí na formulaci jeho účelu. Zaměření se na lidi, kteří se nějakým způsobem na informačních systémech podílejí, je dalším krokem. IT personál jsou lidé, kteří mají specifické dovednosti a znalosti o tvorbě, nasazení a provozu informačních technologií. Od předních

pracovníků uživatelské podpory, přes systémové analytiky, vývojáře až po vedoucího informačního oddělení (CIO, nebo ředitel IT) jsou lidé zabývající se informačními systémy základním prvkem (Bourgeois, 2019).

Proces je řada kroků podniknutých k dosažení požadovaného výsledku nebo cíle. Informační systémy se stále více integrují s organizačními procesy, což těmto procesům přináší větší produktivitu a lepší kontrolu. *„Transformační proces podnikového informačního systému je aplikace informačních technologií"* (Gála, 2015). Což můžeme označit pojmy IT aplikace nebo aplikační software (ASW). ASW plní funkce v závislosti na kontextu (např. ve financích poskytuje uživatelům funkce zaúčtování dokladu, v kontextu prodeje poskytuje funkci zaevidování zákazníka apod.). Dále manipuluje s daty (např. je zpracuje, zobrazí, přenese apod.) za pomoci lidí, hardwaru a softwaru. Podniky, které doufají, že získají konkurenční výhodu nad svými konkurenty, se na tuto složku informačních systémů silně zaměřují (Gála, 2015).

„Informační a komunikační technologie (Information and comunication technologies, ICT) jsou hardwarové a softwarové prostředky pro sběr, přenos, uchování, zpracování a poskytování informací a pro vzájemnou komunikaci lidí a technologických komponent IS" (Voříšek, 2015).

Technické prostředky (hardware) - je hmatatelná, fyzická část informačního systému – část, které se můžeme dotknout. Počítače, klávesnice, pevné disky a routery jsou příklady systémového hardwaru (Bourgeois, 2019).

Programové prostředky (software) - zahrnuje sadu pokynů, které hardwaru říkají, co má dělat. Software není hmatatelný – nelze se ho dotknout. Programátoři vytvářejí software tak, že napíšou sérii pokynů, které hardwaru říkají, co má dělat. Dvě hlavní kategorie softwaru jsou: operační systémy a aplikační software. Software operačních systémů poskytuje rozhraní mezi hardwarem a aplikačním softwarem. Příkladem operačních systémů pro osobní počítač může být Microsoft Windows. Aplikační software umožňuje uživateli provádět úkoly, jako je vytváření dokumentů, zaznamenávání dat do tabulky nebo posílání zpráv příteli (Bourgeois, 2019).

Další součástí informačních technologií jsou data. Data jsou formalizovaný záznam lidského poznání (informací). Např. řetězce „8AB 50-84" a „731548852" jsou data. Jde tedy o nějakou sekvenci znaků, které lze přenášet, uchovávat, interpretovat a zpracovávat (Voříšek, 2015). Informace jsou potom data, která jsou obohaceny o jejich význam. Např. pokud k výše uvedeným řetězcům dodáme, že se jedná o státní poznávací značku automobilu a telefonní

číslo. Kusy nesouvisejících dat nejsou příliš užitečné. Ale agregovaná, indexovaná a společně uspořádaná do databáze se mohou stát mocným nástrojem pro firmy. Analýza dat se pak používá ke zlepšení výkonnosti organizace (Bourgeois, 2019).

Informační systém může existovat i bez možnosti komunikace – první osobní počítače byly samostatné stroje, které neměly přístup k internetu. V dnešním hyper-propojeném světě je to však velmi vzácné a naprostá většina počítačů je připojena k internetu, nebo k jinému zařízení. Technicky vzato je síťová komunikační komponenta tvořena hardwarem a softwarem, ale je to natolik základní vlastnost dnešních informačních systémů, že jí můžeme brát jako vlastní kategorii (Bourgeois, 2019).

1.1.2 Požadavky

Požadavky na systém jsou popis toho, co by měl systém dělat, nebo jaké vlastnosti a funkce by měli být jeho součástí. Požadavky jsou většinou specifikovány budoucími uživateli. *„Rozlišujeme přitom dva základní typy požadavků:*

- *Funkční – specifikují požadavky na funkčnost systému.*
- *Nefunkční – specifikují jisté vlastnosti systému, případně podmínky omezující fungování systému“* (Kanisová, 2006).

„Inženýrství požadavků (Requirements engineering) je termín používaný k popisu aktivit zapojených do zjišťování, dokumentování a údržby množiny požadavků na softwarový systém. Zastupuje odhalování způsobů, jak a k čemu uživatelé daný systém potřebují“ (Arlow, 2007).

Požadavky jsou nesmírně důležitou součástí všech projektů a efektivní inženýrství požadavků je klíčovým faktorem vývoje. Mezi hlavní příčiny neúspěchu softwarových projektů se často řadí právě nedostatečně specifikované požadavky. Další významnou příčinou selhání je nedostatečné zapojení uživatelů do tvorby požadavků. (Arlow, 2007; Kanisová, 2006)

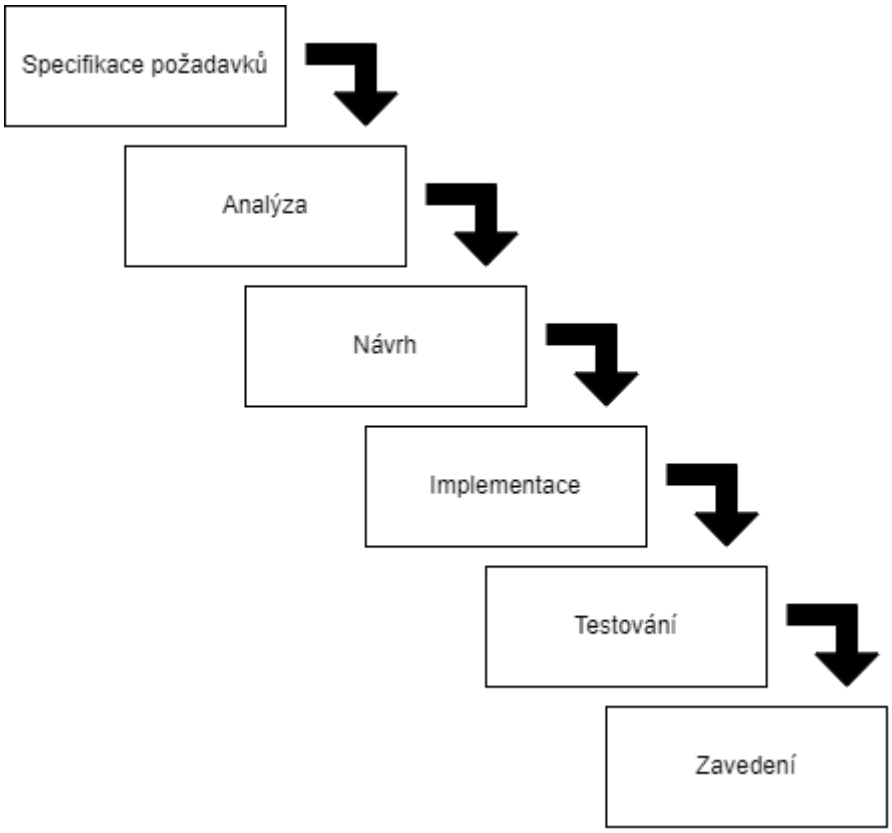
1.2 Životní cyklus

Vývoj, modelování a správa požadavků se realizuje během celého životního cyklu aplikace podnikové informatiky. Ve fázi analýzy v životním cyklu aplikace jde o vývoj požadavků – identifikaci, sběr, dokumentaci a kontrolu požadavků. Během návrhu aplikace jsou požadavky analyzovány a rozpracovány do detailnější podoby v podobě předpisu k implementaci. Během analýzy, implementace a testování aplikace mohou být doladěny případné nesrovnalosti požadavků, vznikají změny v požadavcích, které je nutno řádně zadokumentovat. Do provozu

je uvedena výsledná funkčnost aplikace, přičemž každá funkčnost je podložena požadavkem, resp. požadavky uživatele. Během provozu aplikace postupně vznikají nové požadavky na aplikaci a ve fázi rozvoje, resp. optimalizace aplikace na základě rozhodnutí o osudu nových požadavků se může přistoupit k fázi vzniku nové specifikace požadavků – tedy definování rozsahu nového zadání v podobě množiny požadavků a jejich zadokumentování a kontrole (Kanisová, 2006).

1.2.1 Vodopádový model

Vodopádový model (Waterfall Model) byl prvním procesovým modelem, který byl představen. Byl vyvinutý v 60. letech pro správu velkých softwarových projektů spojených s firemními systémy běžícími na sálových počítačích. V době svého vzniku představoval významný pokrok, jelikož rozdělil celý proces vývoje do samostatných fází. Vyžaduje jasné a předem srozumitelné chápání toho, co má software dělat a není přístupný návrhovým změnám. Vodopádový model ilustruje proces vývoje softwaru v lineárním sekvenčním toku. To znamená, že jakákoli fáze vývojového procesu začíná pouze tehdy, je-li předchozí fáze dokončena. Tento přístup je zhruba podobný procesu montážní linky, kde je všem zúčastněným stranám jasné, co by měl konečný produkt dělat a že velké změny je obtížné a nákladné realizovat. I přes častou kritiku se vodopádový model v praxi stále používá. Především v případech, kdy je možné přesně definovat všechny požadavky na produkt ve fázi specifikace požadavků. Pro případy, kdy není možné specifikovat všechny požadavky na začátku projektu, nebo vznikají časté změny je tento model nevhodný (Bruckner, 2012).



Obrázek 1 - Vodopádový model
Zdroj: vlastní zpracování dle (Bruckner, 2012)

Hlavní výhody vodopádového modelu:

- Jednoduchý, pochopitelný a snadno použitelný model.
- Snadná správa díky neohebnosti modelu. Každá fáze má specifické výstupy a proces revize.
- Jednotlivé fáze se zpracovávají a dokončují postupně.
- Dobře funguje u menších projektů, kde jsou požadavky velmi dobře stanoveny a pochopeny (LEARN SDLC, 2017).

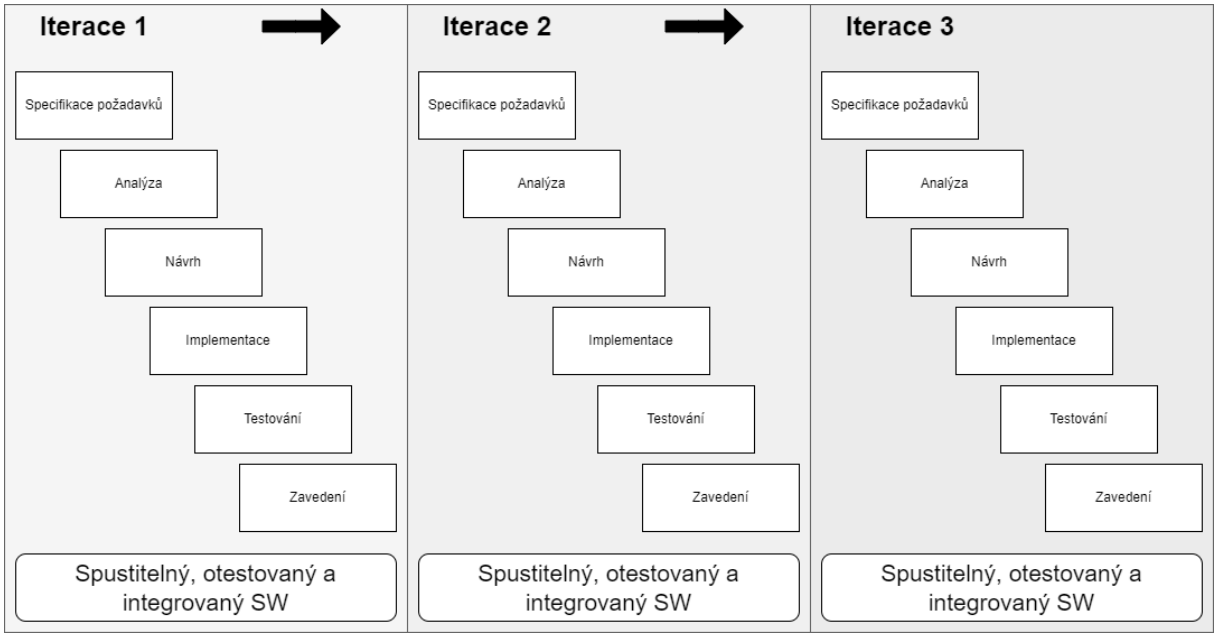
Hlavní nevýhody vodopádového modelu:

- Zákazník je do vývoje zapojen především na začátku a konci procesu, nemá tudíž takovou kontrolu nad průběhem projektu.
- Funkční software je k dispozici až ke konci vývoje.
- Problémy, které vyžadují změny návrhu vedou k celkovému zpoždění projektu.
- Nelze vyhovět měnícím se požadavkům.

- Nevhodný model pro dlouhé a stále se rozvíjející projekty (LEARN SDLC, 2017).

1.2.2 Modely pro iterativní vývoj

Myšlenka iterativního vývoje zakládá na tom, že člověk řeší menší projekty mnohem lépe než jeden velký projekt. Proto se celý projekt rozloží na řadu dílčích projektů – iterací. Každou iteraci si můžeme představit jako malý vodopádový model, který obsahuje všechny fáze vývoje. Výsledkem každé iterace je již funkční část systému (Bruckner, 2012).



Obrázek 2 - Iterativní vývoj
Zdroj: vlastní zpracování dle (Bruckner, 2012)

Model iterativního životního cyklu se nesnaží začít s úplnou specifikací všech požadavků. Místo toho začíná vývoj specifikací a implementací jen části softwaru, která je pak přezkoumána pro identifikaci dalších požadavků. Tento proces se pak opakuje a na konci každé iterace vznikne nová verze softwaru (LEARN SDLC, 2017).

Hlavní výhody iterativního modelu:

- Některé funkce mohou být vyvinuty rychle a brzy v životním cyklu.
- S každou iterací je dodán funkční produkt.
- Lze naplánovat paralelní vývoj.
- Méně nákladná změna požadavků/rozsahu.
- Testování a ladění jednotlivých iterací je snadnější (LEARN SDLC, 2017).

Hlavní nevýhody iterativního modelu:

- Může být zapotřebí více zdrojů.
- Je nutná větší pozornost a zapojení managementu.
- Mohou vzniknout problémy s architekturou nebo designem systému, protože ne všechny požadavky jsou shromážděny na začátku celého životního cyklu.
- Obtížnější stanovení termínu dokončení projektu, což je rizikové (LEARN SDLC, 2017).

Existuje několik moderních metodologií, které vycházejí z iterativního vývoje, nebo v sobě kombinují přístupy iterativního vývoje s dalšími metodologiemi např. inkrementální, nebo spirálový přístup.

Metodologie založená na rychlém vývoji aplikací (RAD, Rapid Application Development). Jedná se o novější třídu metodologií vývoje systémů, které se objevily v 90. letech 20. století. Obecně platí, že přístupy RAD k vývoji softwaru kladou menší důraz na plánování a větší důraz na adaptivní proces. RAD je zvláště vhodný pro vývoj softwaru, který je řízen požadavky na uživatelské rozhraní. Metodologie založené na RAD se pokoušejí řešit slabé stránky metodologií strukturovaného návrhu úpravou fází SDLC, tak aby se některé části systému vyvinuly co nejrychleji a dostali se do rukou uživatelů. Uživatelé tak mohou lépe porozumět systému a navrhnout revize, které systém přiblíží k požadovanému stavu (Dennis, 2015).

Agilní SDLC model je kombinací iterativních a inkrementálních procesních modelů se zaměřením na procesní adaptabilitu a spokojenost zákazníka rychlým dodáním funkčního softwarového produktu. Agilní metody rozdělují produkt na malé inkrementální buildy (sestavení). Tyto buildy jsou poskytovány v iteracích. Každá iterace obvykle trvá přibližně jeden až tři týdny. Každá iterace zahrnuje více účelové týmy pracující současně na různých oblastech projektu, jako je:

- plánování;
- analýza požadavků;
- návrh;
- programování (implementace);
- testování a akceptační řízení (LEARN SDLC, 2017; Dennis, 2015).

Gála člení fáze životního cyklu aplikace následovně:

- *„plánování a příprava aplikace;*
- *analýza a návrh aplikace;*
- *implementace aplikace;*
- *zavedení do provozu, migrace;*
- *provoz a užití aplikace;*
- *rozvoj a optimalizace aplikace“* (Gála, 2015).

1.2.3 Plánování a příprava aplikace

První úloha, která spadá do této fáze životního cyklu je vstupní analýza. Její úlohou je posouzení myšlenky o potencionálním zahájení nového projektu vůči představám budoucích uživatelů i celkové koncepci informační strategie rozvoje podniku. Posuzuje se nejen představa na danou aplikaci, ale také její vazby na ostatní aplikace v podniku. Provedení vstupní analýzy ještě nemusí znamenat, že se projekt v budoucnu opravdu uskuteční. Cílem vstupní analýzy je jasně formulovat priority a specifikace projektu (Gála, 2015).

Důležitou otázku, kterou si musí odpovědět vedení podniku je to, zdali se bude systém řešit dodavatelským způsobem (outsourcing), nebo ho firma zadá internímu týmu. V současné době převládá volba outsourcingu, a to jak rozvoje, tak i provozu. Je samozřejmé, že sebou outsourcing nese určitá rizika, specializace IT firem se však tak zefektivnila, že pozitivní přínosy např. (špičková kvalita specializovaného dodavatele, nejnovější technologie, snížení nákladů díky sdílení zdrojů) převažují ty negativní.

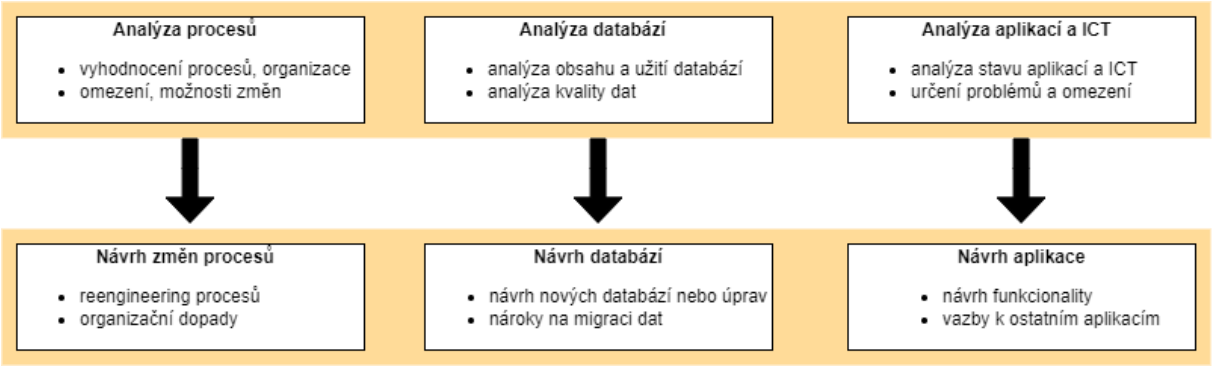
V tomto stádiu by také měla vzniknout představa, zda bude projekt realizován typovým aplikačním softwarem (TASW), nebo individuálním aplikačním softwarem (IASW). Zvolení varianty IASW znamená, že aplikace bude navržena a vytvořena zcela na míru dle potřeb podniku, pro který je určena. Výhodou této varianty je to, že podnik může lépe podpořit svoje specifické procesy a tím lépe dosáhnout svých cílů na trhu. Nevýhodou je pak vyšší cena a většinou i pomalejší vývoj. TASW je vytvořena a dále rozvíjena specializovanou firmou, která tento hotový software nabízí? Celkové náklady TASW jsou sice vyšší oproti IASW, cena pro zákazníka je nižší, z důvodu rozpuštění nákladů mezi více zákazníků. Velkou výhodu TASW je značně kratší doba nasazení, jelikož se aplikace nemusí vyvíjet od začátku, ale použije se již hotový software, který se případně upraví pro potřeby společnosti. Nevýhodou je naopak to, že

se firma musí nějakým způsobem přizpůsobit hotovému softwaru, který nebyl navržen přímo pro její potřeby (Voříšek, 2015; Bourgeois, 2019).

Dalším krokem je vypracování úvodní studie, jejíž smyslem je stanovení celkové koncepce řešení v rámci celkového rozvoje podnikové informatiky. Obsahem studie je definice cíle projektu, promítnutí projektu do aplikační architektury podniku, vymezení funkcí, které projekt pokrývá a které nikoli. Studie dále musí specifikovat organizaci projektových činností – určení pravidel komunikace, určení řídícího výboru, principy řízení projektu, personální obsazení pracovních týmů (Gála, 2015).

1.2.4 Analýza a návrh aplikace

Analýza aplikace představuje souhrn činností, jejichž cílem je do detailu popsat funkcionalitu budoucího systému, data, se kterými bude systém pracovat, identifikovat podnikové procesy které má aplikace podporovat. Analýza klade důraz spíše na zkoumání problému a požadavků než na řešení. Návrh klade důraz spíše na koncepční řešení, které splňuje požadavky než na jeho realizaci. Například popis databázového schématu a softwarových objektů. V konečném důsledku lze návrhy realizovat (Gála, 2015).



Obrázek 3 - Fáze analýza a návrh
Zdroj: vlastní zpracování dle (Gála, 2015)

Úlohou analýzy podnikových procesů je získat přehled o současném stavu řízení podniku v oblastech, které má plánovaná aplikace řešit, tj. nákup, výroba, prodej atd. Dále se musí odhalit, kde se nacházejí problémy v řízení a požadavky na jeho další rozvoj. Rozsah analýzy se liší podle řešené aplikace, od řízení vztahů k zákazníkům u aplikací CRM, až po analýzu celopodnikových aplikací ERP (Gála, 2015).

Z předchozí analýzy vychází návrh změn podnikových procesů, které má aplikace podporovat. Změny podnikových procesů se často provádějí způsobem tzv. procesního reengineeringu (Business process re-engineering BPR).

Stejně tak jako se analyzují podnikové procesy se musí provést i analýza stávajících databází. Vyhodnocuje se jejich obsah, rozsah, kvalita a způsob využívání. Cílem této analýzy databází je zhodnotit stav a kvalitu všech databází se kterými má aplikace pracovat (v případě aplikací ERP se může jednat i o všechny databáze společnosti) pro přípravu jejich migrace do nových databázových struktur (Gála, 2015). Po provedení analýzy má podnik představu o časové náročnosti sestavení plánu migrace dat do nových databázových struktur.

Návrh báze dat a jejího obsahu souvisí s typem aplikačního softwaru. Pro typové aplikace jsou databáze jasně definovány, přípustné jsou jen dílčí změny. Pokud je očekáváním zákazníka, že aplikace bude vzájemně propojena s jinými aplikacemi, které jsou v podniku aktuálně provozovány, analýza vyžaduje vyhodnotit technologické možnosti takové integrace. Cílový systém má být navržen tak, aby obsahoval snadno integrovatelné prvky. Cílem integrace je, aby se kombinace nebo propojení různorodých komponentů vzájemně uživateli jevilo jako jednotný systém (Gála, 2015).

Většina aplikací není zcela izolována; spolupracují s jinými aplikacemi nebo sdílejí datové zdroje s jinými aplikacemi. Proto je první formou analýzy aplikací vysoce abstraktní pohled na aplikační prostředí. Tato úroveň analýzy se dívá na aplikaci v kontextu celé organizace, nebo její IT části. Systémové informace jsou shromažďovány na velmi vysoké úrovni. Klíčové je zde pochopit, které aplikace existují, jak na sebe vzájemně působí a zjištěnou hodnotu požadované funkce. S tímto typem analýzy mohou organizace spravovat celkové strategie migrace a identifikovat klíčové aplikace, které jsou dobrými kandidáty pro migraci (Application Analysis, 2012).

Návrh řešení samotné aplikace se dělí na dvě úrovně – logickou a fyzickou. Logická úroveň zahrnuje následující činnosti – návrh funkcí a funkcionality ve strukturované formě, specifikace obsahu podle jednotlivých modulů, návrh výstupních informací (formuláře, výkazy apod.), specifikace interních a externích (s ostatními aplikacemi) vazeb. Fyzická úroveň představuje technologické nároky aplikace, které jsou identifikovány při definování potřebné technologické architektury, síťové konfigurace, specifikaci přístupů uživatelských rolí apod (Gála, 2015).

1.2.5 Implementace a vývoj aplikace

„Implementace zahrnuje přesnou specifikaci jednotlivých programových modulů, tvorbou tzv. prototypů, a následně dle konkrétního řešení buď kastomizaci funkcí typového aplikačního softwaru, nebo vývoj či dovývoj specializovaných, tedy nestandardních programových modulů. To představuje již technologickou realizaci navržených řešení“ (Gála, 2015).

Fáze implementace v životním cyklu aplikace vyžaduje splnění několika úloh. Analytický tým nejdříve musí zpracovat detailní specifikaci jednotlivých modulů a požadovaných úprav. Součástí je detailní specifikace obsahu, změny v uspořádání polí, struktury jednotlivých obrazovek, obsah a struktura reportů, úpravy číselníků a další. V některých projektech je vysokou přidanou hodnotou zpracování návrhu pro vytvoření prototypu, který je možné přímo konzultovat s budoucím uživatelem. Zpracování prototypu dokáže snížit riziko omylů či nedorozumění při formulaci požadavků na funkcionalitu aplikace (Gála, 2015).

Další krok implementace se liší podle toho, zda se jedná o typový software TASW nebo se se jedná o software na míru (IASW)

Pokud se jedná o typový software TASW, následuje kastomizace – nastavení jednotlivých parametrů systému podle podmínek daného prostředí, test upravených modulů a zpracování dokumentace. V druhém případě, se vývoj aplikace realizuje prostřednictvím zvolených vývojových prostředků (programovacích jazyků) ze strany vývojového týmu. To zahrnuje nejen vývoj jednotlivých modulů aplikace, ale i realizaci datových rozhraní s ostatními existujícími aplikacemi v rámci podniku (Gála, 2015; Poláková, 2011).

Každou elementární činnost aplikace je nutné detailně otestovat, přičemž se výsledky testů zdokumentují a vytvoří se z nich protokoly o průběhu a výsledcích testů. Tyto protokoly potom slouží jako podklady o korektnosti funkcionality aplikace oproti jejímu návrhu a zpravidla vniká i uživatelská dokumentace (Gála, 2015; Poláková, 2011).

Akceptační řízení se může uskutečnit pro jednotlivé milníky projektu, nebo jako poslední konečná fáze před uvedením aplikace do provozu. Sestává z instalace jednotlivých modulů do testovacího prostředí, přípravy testovacích dat odpovídajících reálné situaci, kontroly dokumentace z interních testů funkcionality. Na testování funkcionality ze strany uživatelů by měl být adekvátně sestavený tým odborně vybavených a kompetentních osob. Výstupem testování jsou protokoly o průběhu a výsledcích testů (Gála, 2015; Schmuller, 2001).

1.2.6 Zavedení do provozu, migrace

Příprava na zavedení do provozu se liší podle jednotlivých typů aplikace. V případě že se jedná o přechod na provoz nové aplikace, je potřeba zpracovat detailní plán postupu přechodu v závislosti na zvolené variantě – či už byl zvolený okamžitý přechod, anebo varianta postupného přechodu. Plán uvedení aplikace do provozu zahrnuje detailní postup přechodu z původní aplikace na novou., včetně plánu migrace dat. Migrace dat znamená zahrnuje vytvoření prvotních databází pomocí konverze z původních databází, případně vytvoření zcela

nových databází, pokud to projekt vyžaduje. Tato migrace dat je přímo ovlivněna kvalitou provedené analýzy dat z kapitoly 1.2.4 (Gála, 2015).

Uvedení aplikace do provozu dále zahrnuje instalaci technologické infrastruktury, tj. instalaci základního softwaru a ostatních technických zařízení, instalaci aplikačního softwaru na servery a klientské stanice, vytvoření uživatelských profilů, nastavení přístupových práv apod. Časová náročnost tohoto kroku je závislá na počtu budoucích uživatelů a fyzickém rozmístění/rozsah jejich pracovišť (Gála, 2015).

Neméně důležitým krokem je včasné naplánování školení budoucích uživatelů. Zejména v případě mnoha dislokovaných pracovišť se jedná o organizačně a technicky náročnou činnost. Po organizační stránce se zabezpečuje vydání nových, resp. aktualizovaných interních předpisů a směrnic (Bruckner, 2012).

Posledním krokem, který se provádí před uvedením aplikace do provozu je předávací řízení. Obě zúčastněné strany, tedy zákazník a dodavatel si vzájemně s předávacím protokolu odsouhlasí požadovanou funckionalitu, provozní charakteristiky aplikace a uživatelskou dokumentaci (Voříšek, 2015; Gála, 2015).

1.2.7 Provoz aplikace

Samotný provoz aplikace spočívá v podpoře uživatelů, konzultačních službách, správě infrastruktury, monitorování provozu a dalších operativních zásahů do aplikace. Bezprostředně po uvedení aplikace do provozu má poměrně významnou úlohu kontaktní místo, kam se uživatelé aplikace mohou obrátit s jakýmkoli problémem. Help-desk je místem, kde zaškolené osoby odpovídají na dotazy uživatelů, případně analyzují a přeposílají přijaté podněty kompetentním osobám. Aplikace se během svého provozu monitorují – zachycují se informace o vytížení, poruchách a vzniklých chybách způsobených technologiemi, na kterých je aplikace provozována. Statistiky a veškeré výstupy, které vznikají během monitoringu slouží jako podklad pro další rozvoj aplikace (Voříšek, 2015; Gála, 2015).

1.2.8 Další rozvoj a optimalizace

Rozhodnutí o dalším kroku v rozvoji aplikace vyplívá ze změnového řízení, které určí budoucí optimalizaci aplikace – částečných úprav v podobě nové verze aplikace, anebo zásadních změn celého řešení, což by znamenalo zadání nového projektu. Podkladem pro rozhodnutí o novém vývoji jsou požadavky uživatelů, postupně sbírané v průběhu provozu aplikace. Podnik stanovuje základní pravidla pro sběr a vyhodnocení požadavků, které určují:

- Osoby oprávněné k formulování nových požadavků.
- Jak a kde se požadavky shromažďují.
- Kdo a jakým způsobem požadavky posuzuje.
- Jakým způsobem je uživatel informován o výsledku posouzení požadavku (Gála, 2015).

Pokud některé přijaté požadavky překračují rámec běžné údržby, nebo jde o tak zásadní změnu, která by vyžadovala specifikaci nového projektu, zváží kompetentní tým.

1.3 Unifikovaný modelovací jazyk

Softwarové architektury mohou být modelovány z různých úhlů za pomocí softwarových modelovacích jazyků, což mohou být jazyky popisu architektury (ADL, Architecture description language), doménově specifické jazyky (DSL, Domain-specific language) a UML. Mezi nimi je UML považován za nejčastěji používaný jazyk pro modelování softwarových architektur (Ozkaya, 2020).

1.3.1 Co je to UML

UML (unifikovaný modelovací jazyk, Unified Modeling Language) je obecný vizuální modelovací jazyk, který se používá pro specifikaci, vizualizaci, konstrukci a dokumentaci artefaktů softwarového systému. Zachycuje rozhodnutí a porozumění o systémech, které musí být zkonstruovány. Používá se k porozumění, návrhu, procházení, konfiguraci, údržbě a řízení informací o takových systémech. Je určen pro použití se všemi vývojovými metodami, stadií životního cyklu, aplikačními doménami a médii. Modelovací jazyk má sjednotit dosavadní zkušenosti s modelovacími technikami a začlenit současné softwarové osvědčené postupy do standardního přístupu. UML zahrnuje sémantické koncepty, notaci a pokyny. Má statickou, dynamickou, environmentální a organizační část. Má být podporován interaktivními vizuálními modelovacími nástroji (CASE), které mají zabudované generátory kódu a automatické hlášení chyb. Specifikace UML nedefinuje standardní proces, ale má být užitečná s procesem iterativního vývoje. Má podporovat většinu stávajících objektově orientovaných vývojových procesů (Rumbaugh, 2010).

Jedna z výhod UML spočívá v tom, že je čitelný a srozumitelný pro lidi, ale zároveň snadno interpretovaný pro programy CASE. UML samo o sobě nedefinuje metodiku modelování, pouze poskytuje vizuální syntaxi. Metodikou často spojovanou s jazykem UML je Unified Process,(UP = volně šiřitelná verze rup) která již udává přesný pracovní postup. Jazyk UML

tedy není přímo vázán na UP, je to pouze preferovaná metoda užití, kdy UP využívá jazyk UML jako syntaxi pro vizuální modelování. UML tedy podporuje i další metody např. OPEN (Object-oriented Process, Environment and Notation), nebo Select Perspective (Arlow, 2007).

Podle Kanisové (2006) je UML *„souhrnem především grafických notací k vyjádření analytických a návrhových modelů.“* Formální syntaxe UML se nemění v závislosti na velikosti a složitosti projektu, což znamená vysokou kompatibilitu mezi návrháři systémů. Jednotlivé diagramy nezachycují systém jako celek, ale zaměřují se vždy na jeden pohled na vyvíjený systém. (Kanisová, 2006).

1.3.2 Historie UML

UML bylo vyvinuto ve snaze zjednodušit a konsolidovat velké množství objektově orientovaných vývojových metod, které se začali objevovat. Komerční aplikace se spíše zdráhaly přijmout rozsáhlé CASE systémy a vývojové metody. Většina podniků vyvíjela software interně pro své vlastní potřeby, bez nepřátelského vztahu mezi zákazníkem a dodavateli, který charakterizoval velké vládní projekty. Rozmanitost modelovacích jazyků a metod znamenala vysokou nekompatibilitu mezi projekty napříč firmami (Rumbaugh, 2010).

V roce 1994 vznikla první snaha o sjednocení dosud velmi chaotického prostředí několika soupeřících jazyků pro vizuální modelování. Nastala situace kdy si zhruba polovinu trhu rozdělili dvě nejpoužívanější metody – Booch (Grady Booch) a OMT (Object Modeling technique, James Rumbaugh). První pokus o sjednocení trhu s názvem Fusion (sloučení Booch, OMT a Objectory) nebyl úspěšný, jedním z faktorů bylo to, že se na jejím vývoji nepodíleli autoři původních metod. Fusion byl rychle překonán, když v roce 1994 Booch a Rumbaugh vstoupili do Rational Corporation, aby pracovali na UML. Nový jazyk, který přejímá to nejlepší z předchozích jazyků se stal velice rychle otevřeným průmyslovým standardem (Arlow, 2007).

V roce 1996 vydala Object Management Group (OMG) žádost o návrhy standardního přístupu k objektově orientovanému modelování. Autoři UML Booch, Jacobson a Rumbaugh začali spolupracovat s metodiky a vývojáři z jiných společností, aby vytvořili návrh atraktivní pro členství v OMG, stejně jako modelovací jazyk, který by byl široce přijímán výrobci nástrojů, metodiky a vývojáři, kteří by byli případní uživatelé. Od roku 1997 OMG převzala odpovědnost za další vývoj standardu UML (Rumbaugh, 2010).

Od roku 1997 se UML stále rozvijí o nové funkce. Nejvýznamnější aktualizací byla verze UML 2.0, která přišla v roce 2005. V aktualizované verzi se opravilo spoustu chyb odhalených zkušenostmi s používáním a zároveň přibyli nové prvky vizuální syntaxe, které byly

požadovány v některých aplikačních doménách. Aktuální verze je UML 2.5 z roku 2015, která přináší hlavně drobné revize (Walker, 2022).

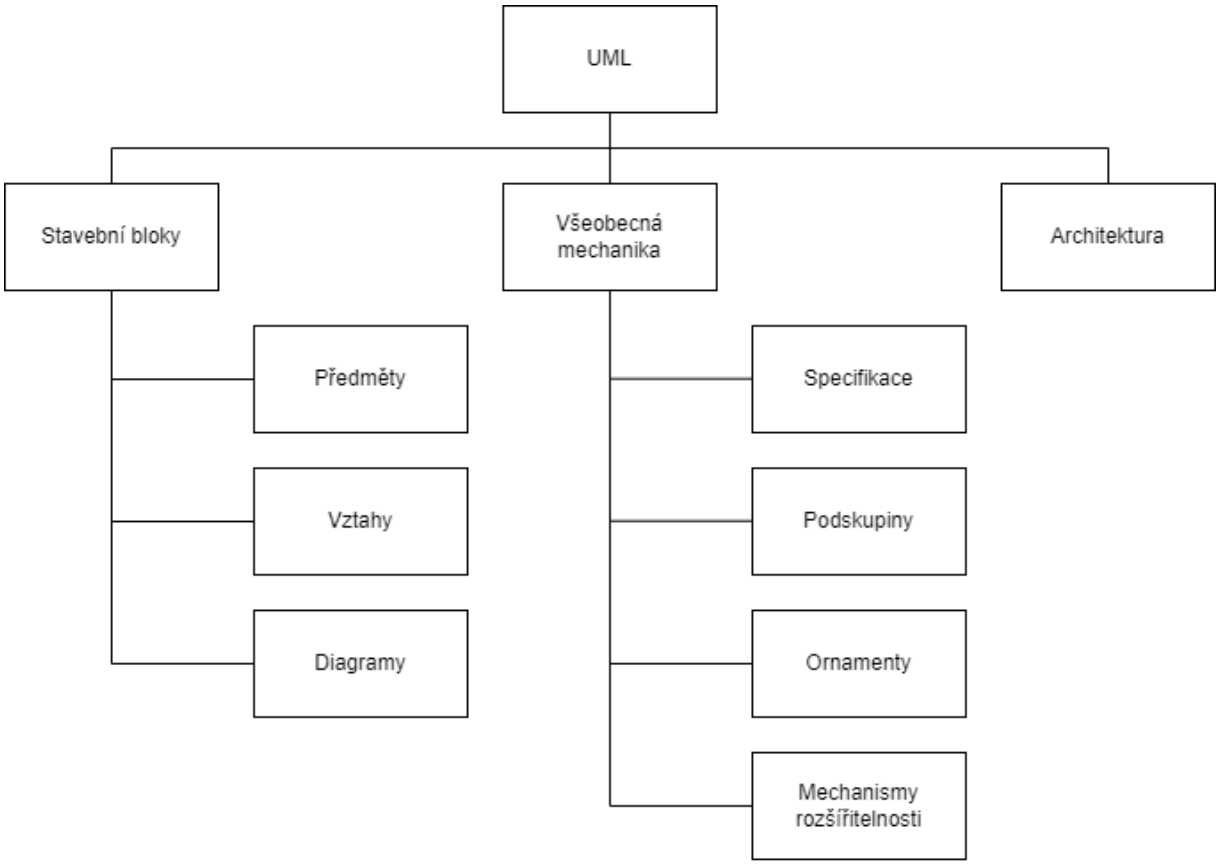
1.3.3 Struktura UML

Abychom pochopili funkci jazyka UML, je důležité se povídat na jeho strukturu. Struktura jazyka UML se skládá ze třech součástí:

- Stavební bloky – základní prvky modelu, relace a diagramy.
- Společné mechanismy – obecné způsoby, kterými se v jazyku UML dosahuje specifických cílů.
- Architektura – pohled v jazyku UML na architekturu navrhovaného systému (Arlow, 2007).

UML se skládá ze tří hlavních stavebních bloků:

- Předměty (Things, nebo Classifiers) – představují samotné prvky modelu.
- Vztahy, relace (Relationships) – propojení mezi předměty, určují jak spolu dva nebo více předmětů sémanticky souvisí.
- Diagramy (Diagrams) – kolekce předmětů, vizualizace toho, co bude systém dělat, nebo jak to bude dělat (Arlow, 2007).



Obrázek 4 - Struktura UML
Zdroj: vlastní zpracování dle (Arlow, 2007)

Předměty se dále dělí na:

- Strukturální abstrakce (structural things) – takzvaná podstatná jména modelu UML. Definují statickou část modelu a představují fyzické a koncepční prvky. Patří sem třída (class), rozhraní (interface), spolupráce (collaboration), případ užití (use case), aktivní třída (active class), komponenta (component) a uzel (node).
- Abstrakce chování (behavioural things) – dynamické části, či slovesa modelu UML. Skládá se z interakcí (interaction) aktivit (aktivity) a stavu (state machine).
- Abstrakce seskupení (grouping things) – organizační část modelu UML. Jde o balíčky používané k seskupování sémanticky souvisejících prvků dohromady.
- poznámky (annotational things) – anotace, mechanismus k zachycení poznámek, popisů a komentářů, které lze k modelu připojit (LEARN UML, 2017).

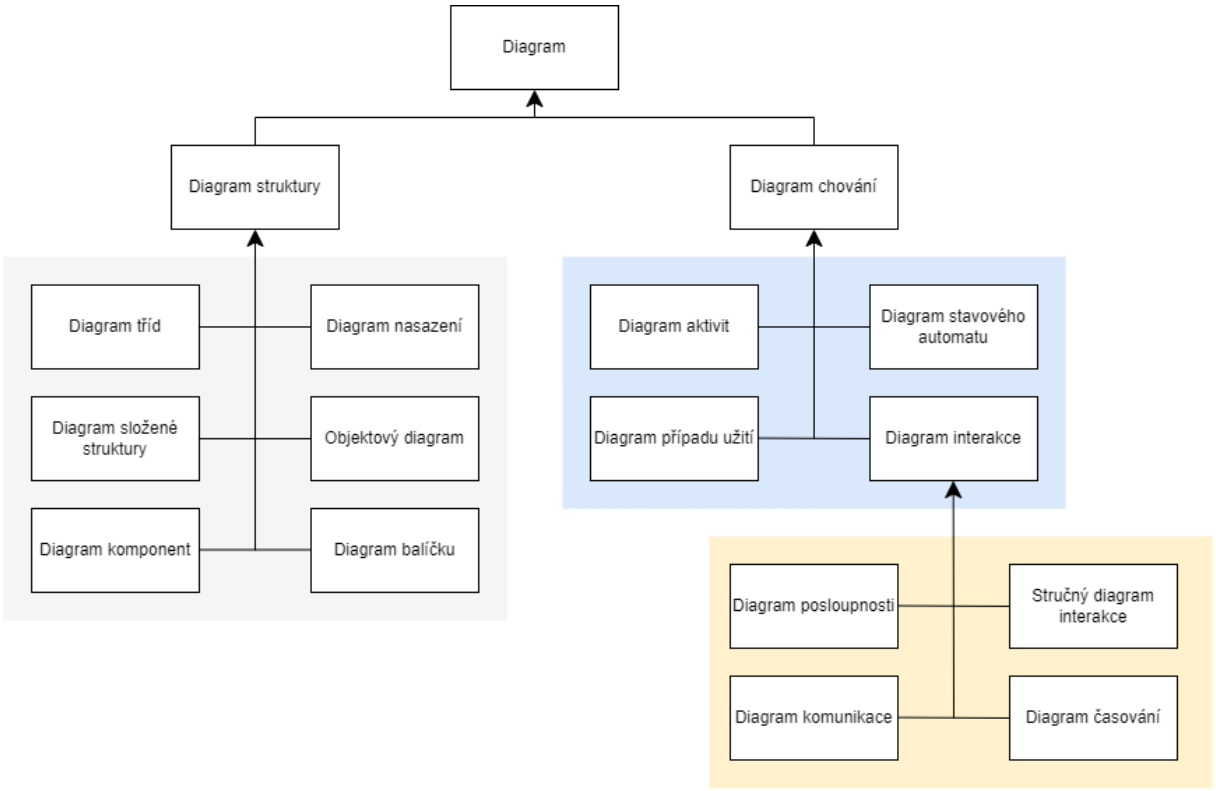
Relace slouží k určení vztahu mezi dvěma a více předměty. Relace se dají přirovnat k vztahům v rodinnému stromu, kde každé spojení znamená určitý druh vztahu mezi členy rodiny např. potomek, sourozenec, strýc apod. Relace tedy zachycují sémantický vztah mezi předměty. V tabulce č.1 je stručný popis čtyřech základních typů relace (Arlow, 2007).

Tabulka 1 – UML relace
Zdroj: vlastní zpracování dle (Arlow, 2007) a (Rumbaugh, 2010)

Typ relace	Popis	Syntaxe Zdroj cíl
Asociace (Association)	Popis množiny spojení mezi objekty. Popis spojení mezi instancemi tříd.	
Závislost (Dependency)	Zdrojový předmět závisí na cílovém předmětu a jeho význam může být ovlivněn jeho změnami.	
Generalizace (Generalization)	Zdrojový prvek je specializací cílového obecnějšího prvku a může jím být nahrazen.	
Realizace (Realization)	Zdrojový klasifikátor zaručuje splnění dohody, kterou specifikuje cílový klasifikátor.	

1.3.4 UML diagramy

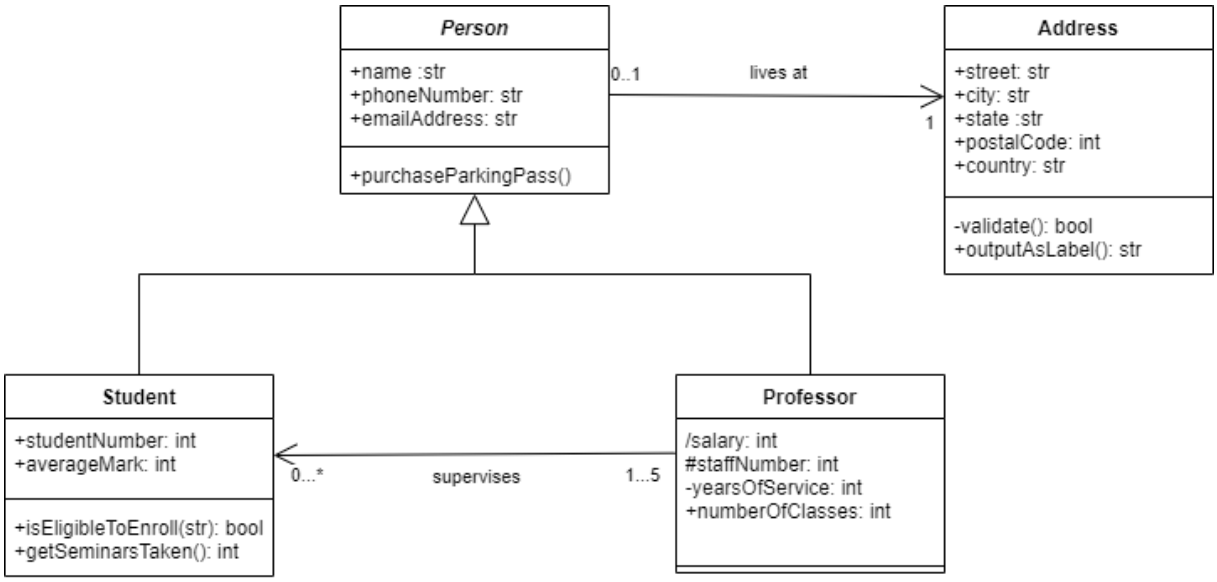
Zatímco model je *„repositářem všech předmětů a relací vytvořených k tomu, aby popisovaly požadované chování softwarového systému“* (Arlow, 2007). Diagram tedy není stejný pojem jako model, ale je to určitý pohled na model. Diagramy jsou grafické prezentace kolekce prvků modelu, nebo také pohledy na model a slouží k lepšímu porozumění, budoucím změnám a údržbě systémů (Rumbaugh, 2010).



Obrázek 5 - struktura UML
Zdroj: vlastní zpracování dle (Arlow, 2007)

Diagram tříd (class diagram)

Diagram tříd je grafická prezentace statického zobrazení, která zobrazuje kolekci deklarativních (statických) prvků modelu, jako jsou třídy, typy a jejich obsah a vztahy. Diagram tříd se nepoužívá pouze pro vizualizaci, popis a dokumentaci různých aspektů systému, ale také pro konstrukci spustitelného kódu softwarové aplikace. Diagram tříd popisuje atributy a operace třídy a také omezení kladená na systém. Diagramy tříd jsou široce používány při modelování objektově orientovaných systémů, protože jsou jedinými diagramy UML, které lze přímo mapovat pomocí objektově orientovaných jazyků. Je také známý jako strukturální diagram (LEARN UML, 2017).

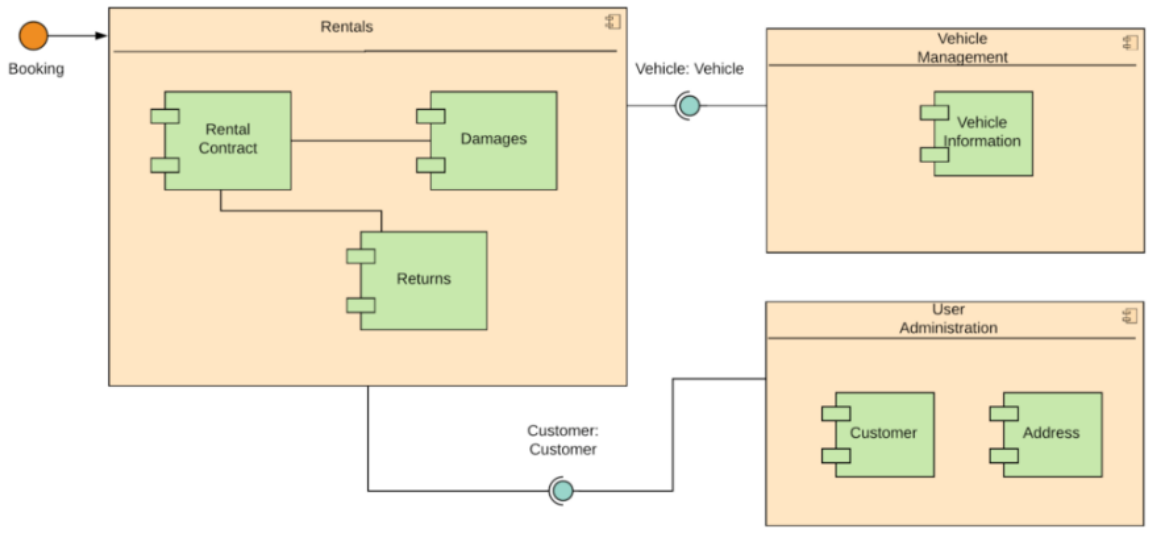


Obrázek 6 - UML diagram tříd
Zdroj: vlastní zpracování dle (Diagrams.net, 2022)

Na obrázku je příklad jednoduchého diagramu tříd. V horní části každé třídy je její jméno, v prostřední části jsou atributy třídy (data/proměnné) a dolní obsahuje metody (operace/funkce) třídy. Třídy jsou propojeny různými typy relací. Symboly před názvy atributů a metod označují jejich viditelnost.

Diagram složené struktury (composite structure diagram)

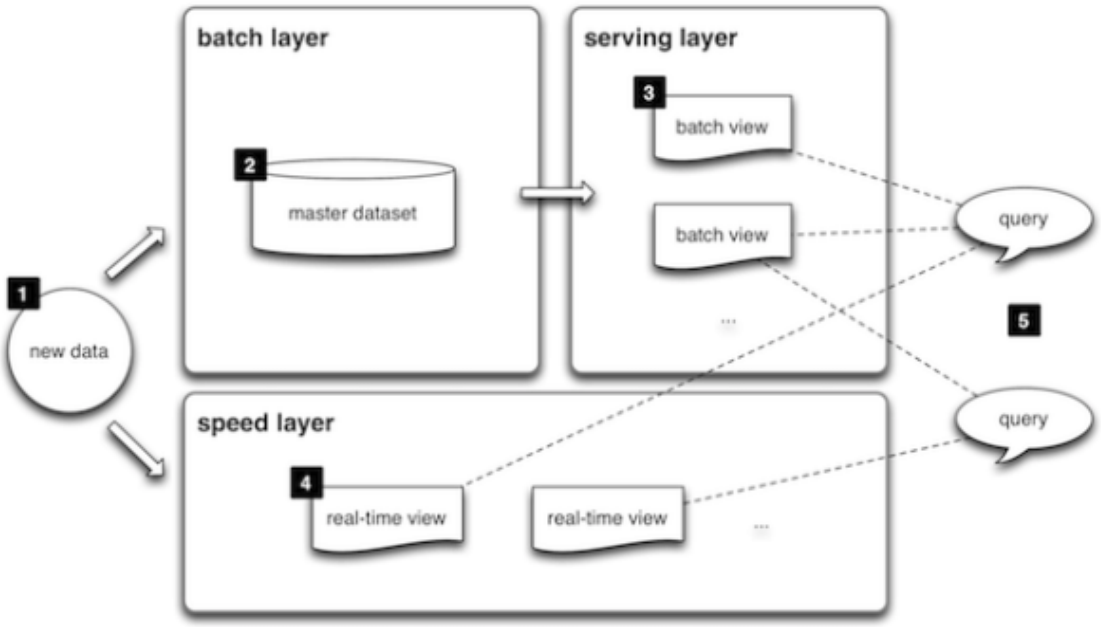
Diagram složené struktury znázorňuje vnitřní strukturu strukturovaných předmětů pomocí součástí, portů a konektorů. Strukturovaný předmět definuje implementaci předmětu a může zahrnovat třídu, komponentu, nebo uzel nasazení. Diagram složené struktury můžeme použít k zobrazení vnitřních podrobností předmětu a k popisu objektů a rolí. Diagram složené struktury je podobný diagramu tříd, místo celých tříd však zobrazuje jednotlivé části (Rational Software Architect, 2016).



Obrázek 7 – Diagram složené struktury
Zdroj: (Tallyfy.com, 2018)

Diagram komponent (component diagram)

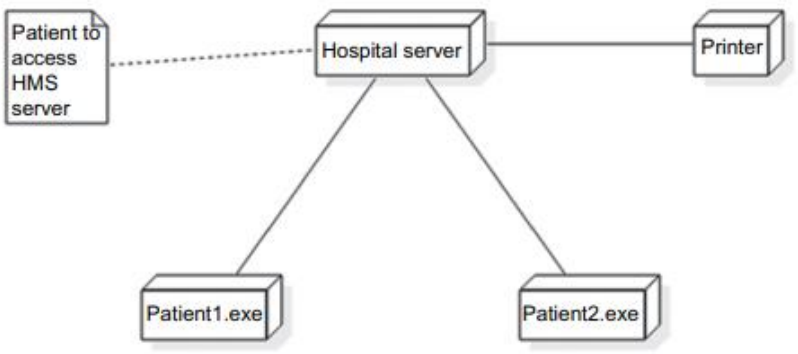
Diagramy komponent se používají k modelování fyzických aspektů systému. Fyzické aspekty jsou prvky, jako jsou spustitelné soubory, knihovny, soubory, dokumenty atd. Komponenty mohou být taky vnořeny do jiných komponent. Diagramy komponent se používají k vizualizaci organizace a vztahů mezi komponentami v systému. Tyto diagramy se také používají k vytváření spustitelných systémů (LEARN UML, 2017).



Obrázek 8 – Diagram komponent
Zdroj: (Tallyfy.com, 2018)

Diagram nasazení (deployment diagram)

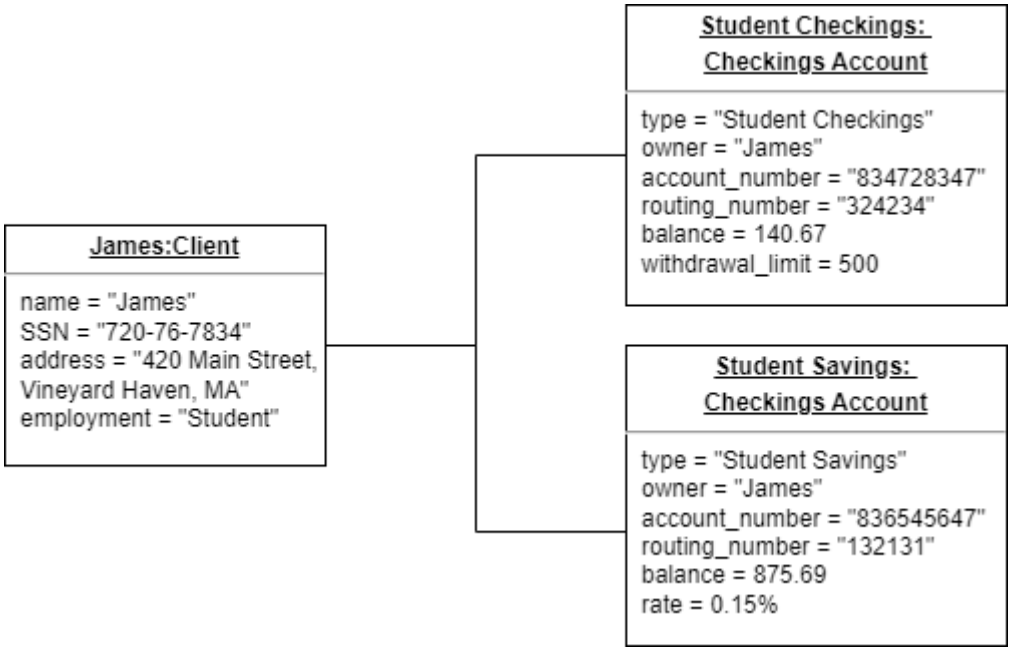
Diagramy nasazení se používají k vizualizaci topologie fyzických komponent systému, kde jsou softwarové komponenty nasazeny. Diagramy nasazení se skládají z uzlů a jejich vztahů. Samotný termín nasazení popisuje účel diagramu. Diagramy nasazení se používají pro popis hardwarových komponent, kde jsou nasazeny softwarové komponenty. Diagramy komponent a diagramy nasazení spolu úzce souvisí. Diagramy komponent se používají k popisu komponent a diagramy nasazení ukazují, jak jsou nasazeny v hardwaru (LEARN UML, 2017).



Obrázek 9 - diagram nasazení
Zdroj: (Arlow, 2007)

Objektový diagram (object diagram)

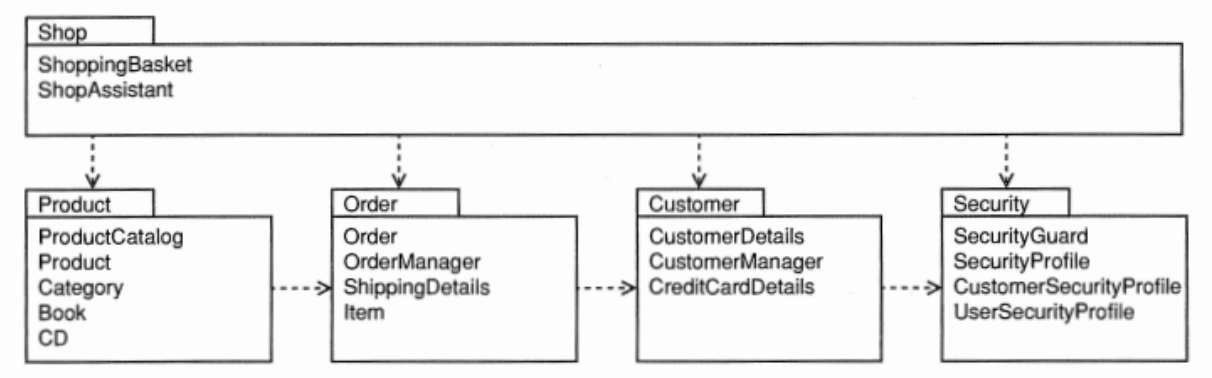
Objektové diagramy jsou odvozeny od diagramů tříd a jsou na nich závislé. Diagram, který ukazuje objekty a jejich vztahy v určitém okamžiku. Objektový diagram lze považovat za speciální případ diagramu tříd, ve kterém mohou být zobrazeny specifikace dané instance, stejně tak jako třídy. Diagramy objektů se používají k vykreslení sady objektů a jejich vztahů jako instance (Rumbaugh, 2010).



Obrázek 10 - UML objektový diagram
Zdroj: vlastní zpracování dle tallyfy

Diagram balíčku (package diagram)

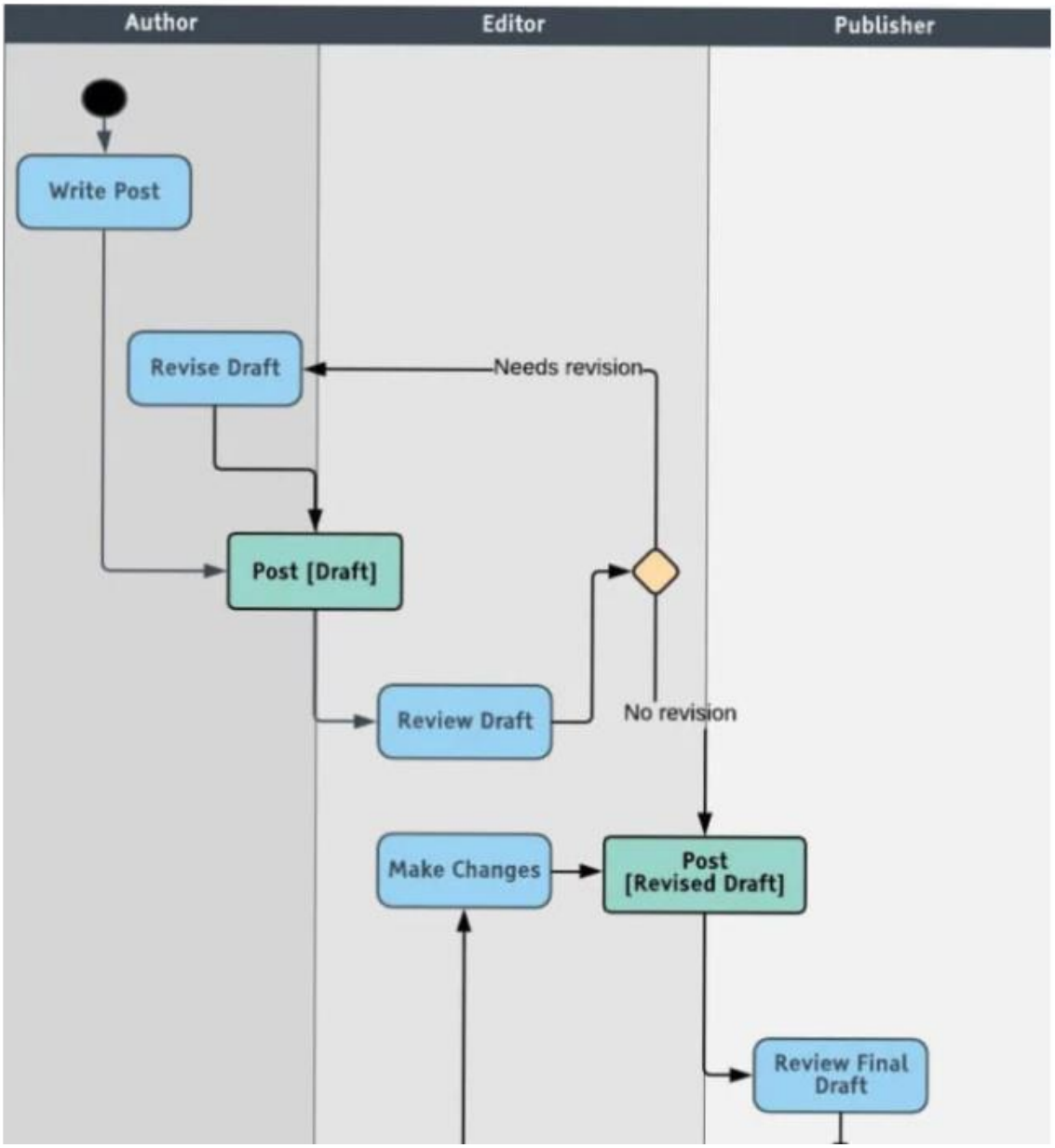
Balíčky primárně představují velkou a soudržnou část systému (subsystém). Z hlediska modelování jsou tyto digramy považovány za organizační. Balíček je sbírka logicky soudržných artefaktů a diagramů UML. Diagramy balíčku proto ukazují pouze pohled shora – pohled z ptačí perspektivy – na organizaci systému. Diagramy balíčků mohou také zobrazovat závislosti mezi balíčky. Diagram balíčku lze použít ke zjednodušení komplexních diagramů tříd, může seskupovat třídy do balíčků (Unhelkar, 2018).



Obrázek 11 - Diagram balíčku
Zdroj: (Arlow, 2007)

Diagram aktivit (activity diagram)

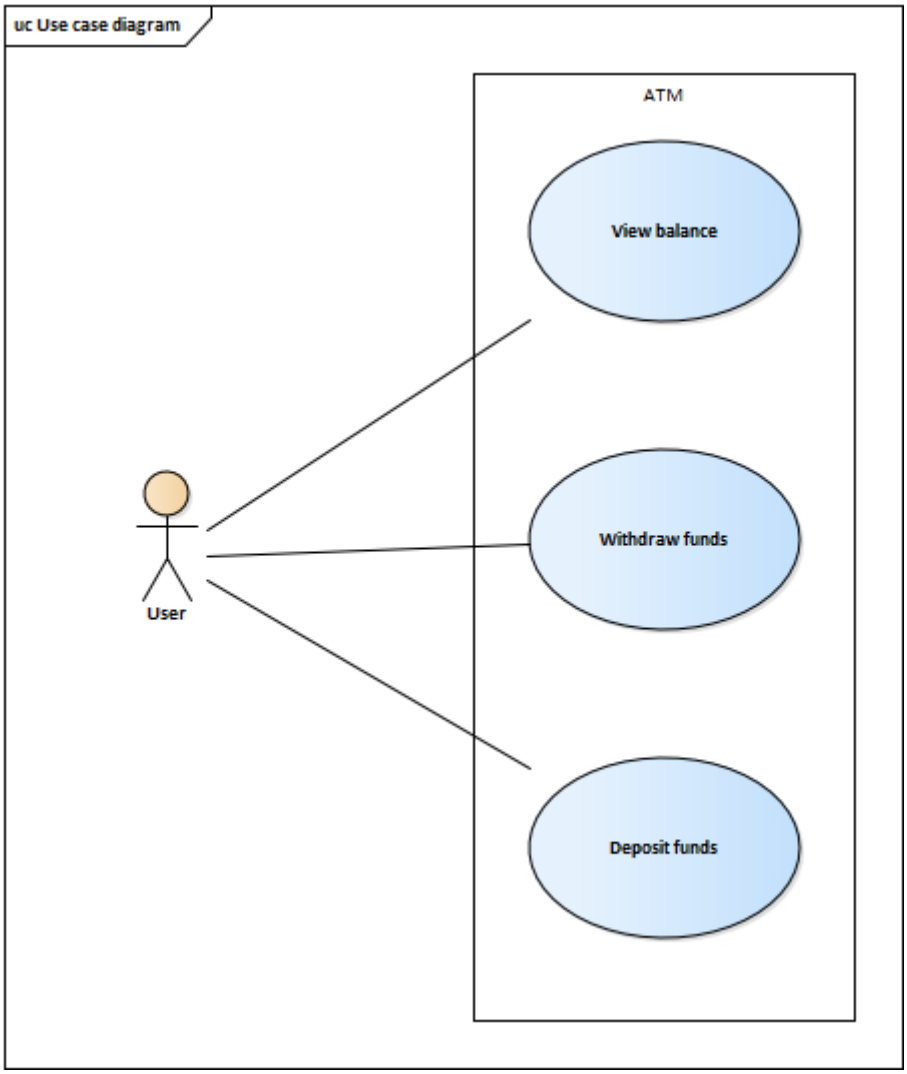
Diagram aktivit je v podstatě flowchart, který znázorňuje tok, či proces z jedné aktivity do další aktivity. Aktivita může být popsána jako operace v rámci systému. Aktivity mohou být na obchodní úrovni, nebo na podrobné technické úrovni. Diagramy aktivit dokumentují interní chování v rámci případů užití, mezi případy užití nebo celého podniku. Další důležitou vlastností diagramů aktivit je schopnost ukázat závislost mezi aktivitami. Diagramy aktivit také pomáhají při mapování aktivit k odpovídajícím aktérům v rámci systému (Unhelkar, 2018).



Obrázek 12 - Diagram aktivit
Zdroj: (Tallyfy.com, 2018)

Diagram případu užití (use case diagram)

Účelem diagramu případu užití je zachytit dynamický aspekt systému. Diagramy případů užití se používají ke shromáždění požadavků na systém včetně vnitřních a vnějších vlivů. Tyto požadavky jsou většinou požadavky na design. Vnitřní a vnější činitelé jsou známí jako aktéři. Diagramy případů užití se skládají z aktérů, případů užití a jejich vztahů. Když je tedy systém analyzován za účelem shromáždění jeho funkcí, jsou připraveny případy použití a identifikováni aktéři (LEARN UML, 2017; Schmuller, 2001).

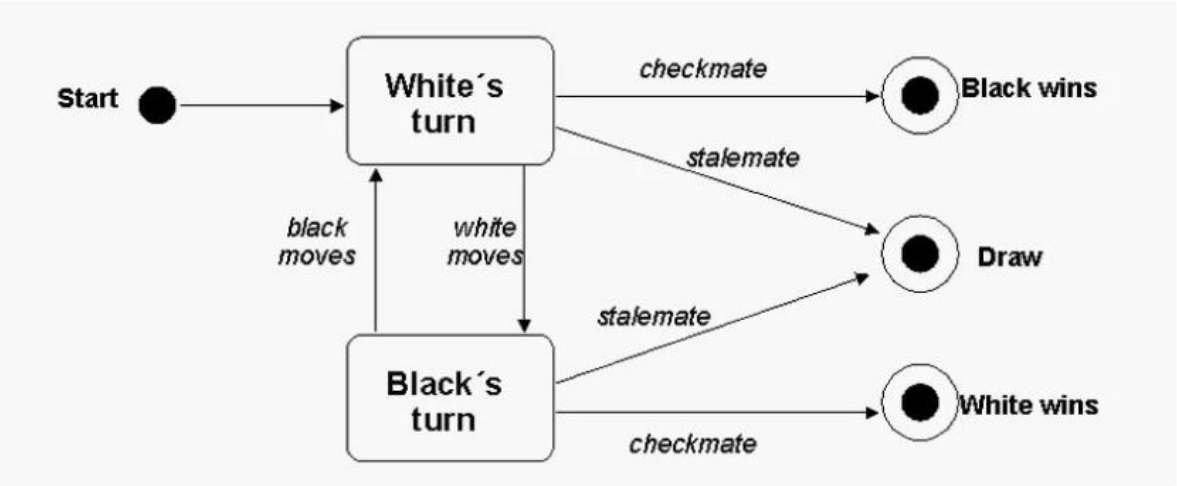


Obrázek 13 - UML diagram případu užití
Zdroj: vlastní zpracování

Figurka znázorňuje externího aktéra v našem případě uživatele bankomatu. Subjekt diagramu (ATM) je ohraničený a obsahuje tři případy užití. Aktér a případy užití jsou spojeny asociací.

Stavový diagram (state machine diagram) (diagram stavového automatu)

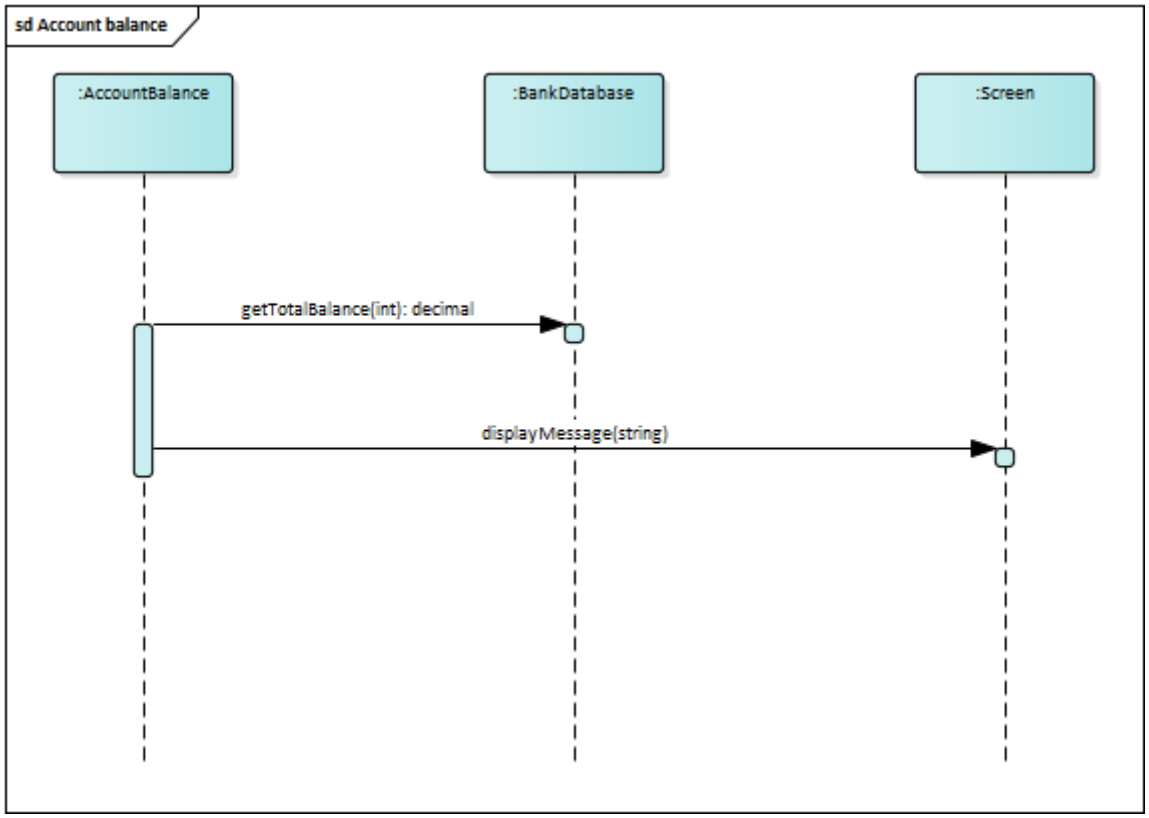
Diagram stavového automatu (také nazývaný stavový diagram) označuje různé stavy, ve kterých se může nacházet objekt, případ použití nebo celý systém. Ve srovnání s objektovým diagramem (který ukazuje „verzi instance“ nebo „snímek“ systému) zobrazuje stavový diagram všechny možné stavy objektu – zobrazuje tím celý životní cyklus objektu, který mění svůj stav v reakci na zprávy, které přijímá. Výsledkem je, že se jedná o diagram chování, který nemá téměř žádný strukturální obsah (Unhelkar, 2018).



Obrázek 14 - UML stavový diagram
Zdroj: (Tallyfy.com, 2018)

Sekvenční diagram (sequence diagram) (diagram posloupnosti)

Je první z dvou interakčních diagramů, které znázorňují, jak mezi sebou spolupracují objekty. Je to diagram, který znázorňuje posloupnost zpráv mezi objekty v interakci. Sekvenční diagram se skládá ze skupiny objektů, které jsou reprezentovány čarami života, a zpráv, které si během interakce postupně předávají. Zprávy jsou znázorněny šipkou mezi čarami života dvou objektů. Zprávy jsou odesílány v pořadí směrem shora dolů, v souladu se scénářem případu užití (Kanisová, 2006).

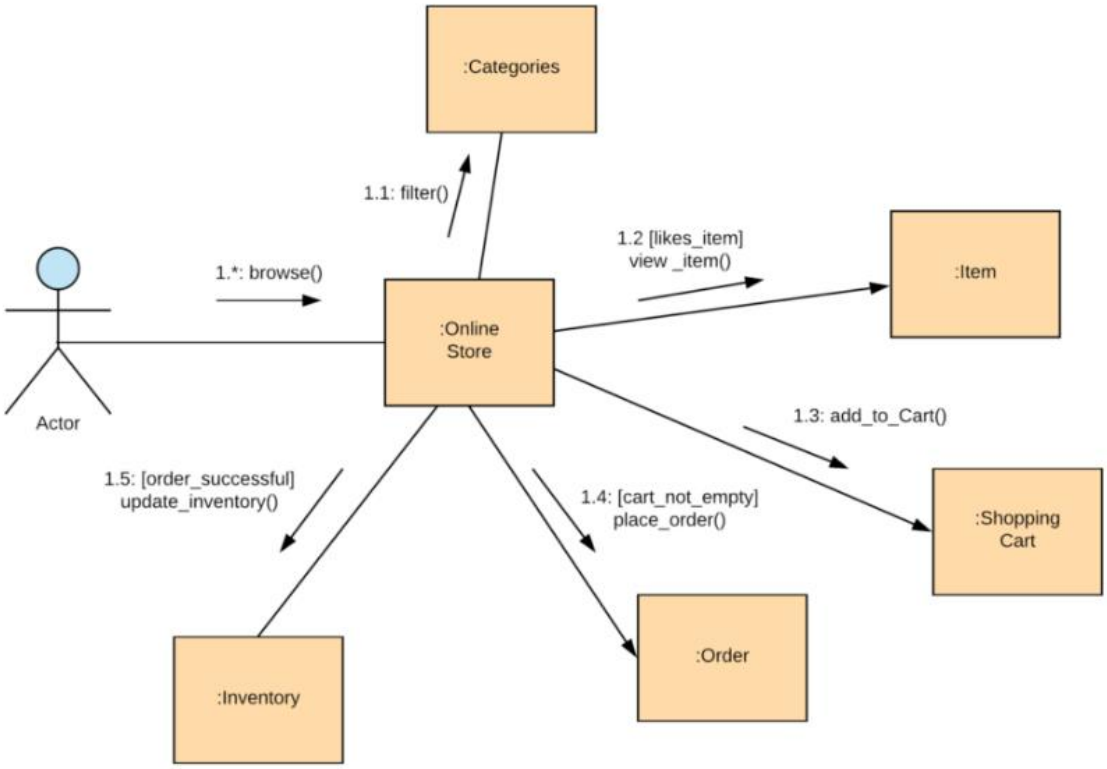


Obrázek 15 - UML sekvenční diagram
Zdroj: vlastní zpracování

Sekvenční diagram znázorňuje objekt AccountBalance, který během své čáry života pošle dvě zprávy objektům BankDatabase a Screen.

Diagram komunikace (communication diagram)

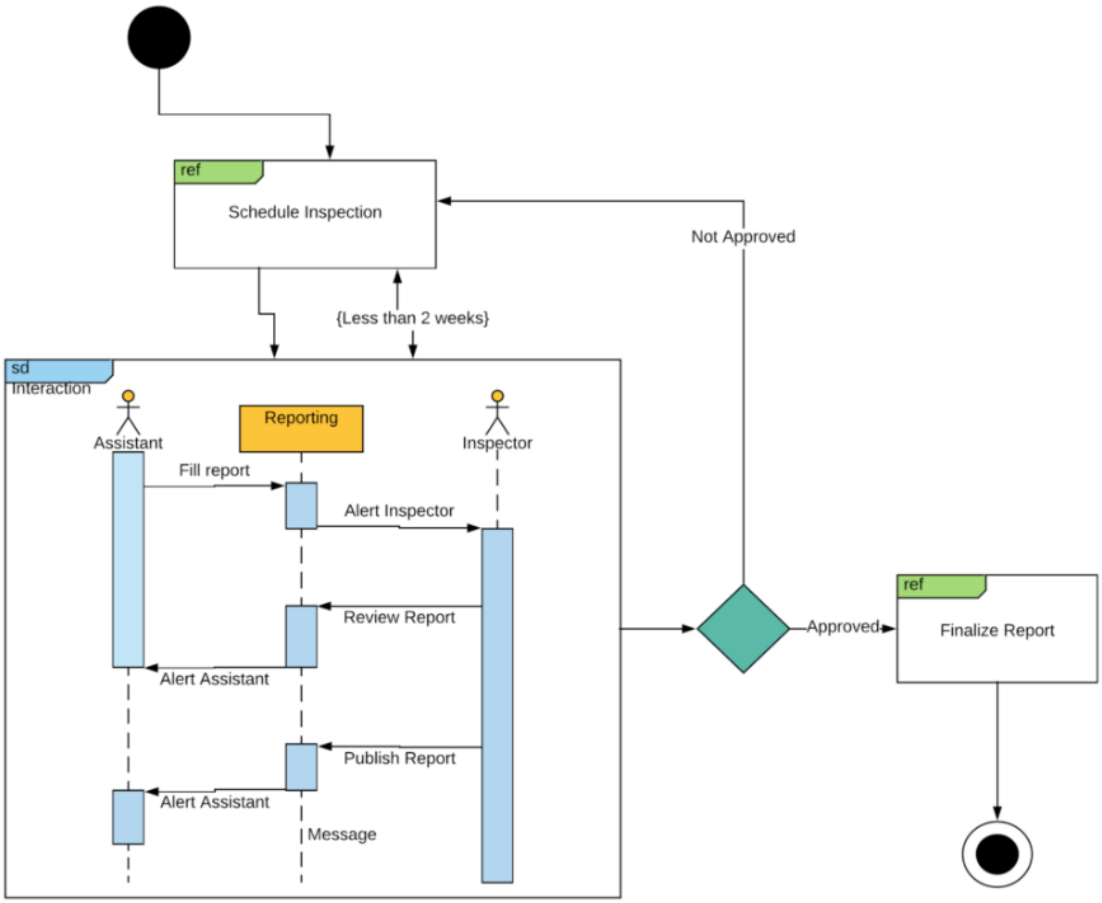
Diagramy (objektové) komunikace jsou druhou formou interakčních diagramů a můžeme říct, že se jedná o alternativu sekvenčních diagramů. Oproti sekvenčním diagramům nejsou na první pohled tak přehledné, výhoda ale spočívá v prostorovém vyjádření a lepším znázorněním spojení objektů v rámci jejich spolupráce. Vzhled těchto diagramů také může vysvětlit strukturu balíčků. Sekvence zpráv je v tomto případě značena číslováním (Kanisová, 2006; Schmuller, 2001).



Obrázek 16 - Diagram komunikace
Zdroj: (Tallyfy.com, 2018)

Stručný diagram interakce (interaction overview diagram)

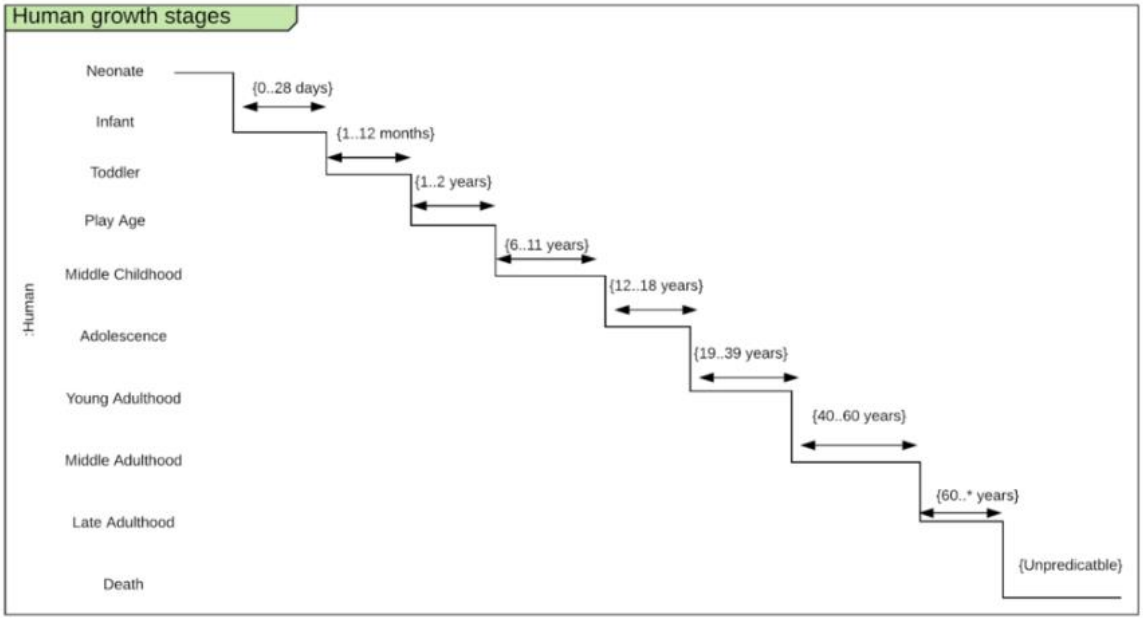
Jak již napovídá název, diagramy poskytují přehled o interakcích probíhajících v systému. Protože jsou tyto interakce nejlépe znázorněny pomocí sekvenčních (nebo alternativně komunikačních) diagramů, stručné diagramy interakce obsahují vnořené sekvenční diagramy, nebo na ně odkazují. Můžeme je brát jako variace diagramu aktivity (mohou znázorňovat rozhodnutí „if-then-else“) zahrnující fragmenty sekvenčního diagramu spolu s tokem kontrolních konstruktů (Unhelkar, 2018).



Obrázek 17 - Stručný diagram interakce
Zdroj: (Tallyfy.com, 2018)

Diagram časování (timing diagram)

Alternativní způsob zobrazení sekvenčního diagramu, který explicitně ukazuje změny stavu na životní čáře v časových úsecích. Nezajímá nás, jak objekty interagují nebo se navzájem mění, ale spíše chceme znázornit, jak objekty a aktéři jednají podél lineární časové osy (tallyfy.com).



Obrázek 18 - Diagram časování
Zdroj: (Tallyfy.com, 2018)

1.3.5 Využití UML pro analýzu IS

Analytické modelování odpovídá na otázky, kdo bude systém používat, co bude systém dělat a kde a kdy bude používán. Během analýzy jsou identifikovány podrobné požadavky a je vytvořen návrh systému. Tým poté vytvoří funkční model (diagram případů užití, diagramy aktivit a popisy případů užití), strukturální model (karty CRC a diagram tříd a objektové diagramy a modely chování (sekvenční diagramy, komunikační diagramy, stavové diagramy) (Dennis, 2015).

Při analýze požadavků probíhá komunikace se zákazníkem, od kterého se nedájí očekávat znalosti UML. Podstatou komunikace je použití náčrtů jako pomůcky pro zlepšení komunikace o nápadech a alternativách budoucího řešení. Pokud je zákazník laik, vnímá nejspíše systém jako souhrn evidovaných pojmů a jejich výskytů v daném systému. V praxi to znamená, že má zákazník přehled o tom, co systém zpracovává, eviduje nebo realizuje. Myšlenky na této úrovni abstrakce mají povahu modelu informačního systému a právě k tomu se efektivně využívá zápis v UML (Kanisová, 2006; Ozkaya, 2020).

Při komunikaci se zákazníkem se často používá zjednodušená neformální forma UML zápisu. Takový zápis je v pořádku a má své místo např. při pracovních schůzkách a slouží zejména k zaznamenání hlavních myšlenek. V takovém případě se může jednat o návrhy vytvořené pomocí jednodušších kreslících nástrojů (Microsoft Visio), nebo i ručně načrtnuté návrhy na

papír nebo tabuli. Na odbornější úrovni např. při pracovních schůzkách v rámci skupiny vývojářů se již konzultují důležité aspekty projektu a je již na místě pracovat s plnohodnotnými UML diagramy za pomocí CASE nástrojů (Kanisová, 2006; Ozkaya, 2020).

Jako první krok samotné analýzy projektový tým shromáždí požadavky od uživatelů. Pomocí shromážděných požadavků pak projektový tým identifikuje obchodní procesy a jejich prostředí pomocí případů užití a diagramů případů užití. Dále uživatelé úzce spolupracují s týmem na modelování obchodních procesů ve formě diagramů aktivit a tým dokumentuje obchodní procesy popsané v diagramech případů užití a diagramech aktivit vytvořením popisu případu užití pro každý případ užití. Nakonec tým ověří a potvrdí obchodní procesy tím, že zajistí, aby všechny tři modely (diagram případů užití, diagram(y) aktivit a popisy případů užití) vzájemně souhlasily. Jakmile je současné chápání podnikových procesů zdokumentováno ve funkčních modelech, je tým připraven přejít ke strukturálnímu modelování. (Dennis, 2015; Ozkaya, 2020).

Případy užití tvoří základ, na kterém je vytvořen podnikový informační systém. Z hlediska architektury zaměřené na strukturální modelování podporuje vytvoření vnitřního strukturálního nebo statického pohledu na podnikový informační systém tím, že ukazuje, jak je systém strukturován, aby podporoval základní podnikové procesy (Dennis, 2015).

Jedním z primárních účelů strukturálního modelu je vytvořit slovní zásobu, kterou mohou používat analytici i uživatelé. Strukturální modely představují předměty, myšlenky nebo koncepty obsažené v oblasti problému a znázorňují jejich vztahy. Vytvořením strukturálního modelu problémové oblasti vytváří analytik slovník nezbytný pro efektivní komunikaci mezi analytikem a uživateli. Strukturální modely jsou obvykle znázorněny pomocí karet CRC, diagramů tříd a v některých případech objektových diagramů (Dennis, 2015; Schmuller, 2001).

Během analýzy používají analytici diagramy chování k zachycení nezbytného porozumění dynamických aspektů základního obchodního procesu (Dennis, 2015).

Pomocí interakčních diagramů (sekvenční a komunikační diagramy) a stavových automatů je možné poskytnout úplný pohled na dynamické aspekty vyvíjejícího se podnikového informačního systému (Dennis, 2015).

Jedním z primárních účelů behaviorálních modelů je ukázat, jak budou základní objekty v problémové oblasti spolupracovat při vytváření součinnosti na podporu každého z případů užití. Zatímco strukturální modely představují objekty a vztahy mezi nimi, modely chování zobrazují

vnitřní pohled na obchodní proces, který případ užití popisuje. Proces lze znázornit interakcí, která se odehrává mezi objekty, které spolupracují na podpoře případu užití pomocí diagramů interakce (sekvence a komunikace). Je také možné ukázat účinek, který má množina případů užití, které tvoří systém, na objekty v systému pomocí stavových automatů. (Ozkaya, 2020).

Vytváření modelů chování je iterativní proces, který se opakuje nejen přes jednotlivé modely chování [např. interakční (sekvenční a komunikační) diagramy a stavové automaty], ale také přes funkční (functional modeling) a strukturální modely. Při vytváření modelů chování není neobvyklé provádět změny ve funkčních a strukturálních modelech (Dennis, 2015).

1.3.6 Využití UML pro návrh IS

Nejpoužívanější diagramy v části návrhu projektu jsou:

- Diagramy tříd a jejich vzájemné spojení.
- Sekvenční diagramy.
- Diagramy balíčků.
- Stavové diagramy tříd se složitým životním cyklem.
- Diagramy nasazení s fyzickým upořádáním SW.

Účelem analýzy je zjistit, jaké jsou potřeby podniku. Účelem návrhu je rozhodnout, jak vybudovat systém, který tyto potřeby bude uspokojovat. Návrhář tento detailní návrh předá programátorům, kteří dle návrhu systém naprogramují. V první fázi je obvyklé vytvoření více variant návrhů, ze kterých se po konzultacích vybere ten nejlepší, který se zpracuje do detailů. Definování hranic pro zpracování detailního návrhu souvisí s rozdělením zodpovědnosti a hranic kompetence v rámci vývojového týmu (Kanisová, 2006).

Důležitou počáteční částí návrhu je prozkoumat několik strategií návrhu a rozhodnout, která bude použita k sestavení systému. Jak již bylo zmíněno v kapitole 1.2 systémy mohou být vyvinuty od nuly (IASW), zakoupeny a přizpůsobeny (TASW) nebo převedeny na jiné subjekty (outsourcing) a projektový tým musí prozkoumat životaschopnost každé alternativy. Toto rozhodnutí ovlivňuje úkoly, které mají být během návrhu splněny.

Zároveň musí být ještě dokončen detailní návrh jednotlivých tříd a metod, které se používají k mapování základních součástí systému a způsobu jejich uložení. Techniky, jako jsou karty CRC, diagramy tříd, specifikace kontraktu, specifikace metod a návrh databáze, poskytují

konečné detaily návrhu v rámci přípravy na fázi implementace a zajišťují, že programátoři mají dostatek informací pro efektivní vybudování systému (Dennis, 2015).

Kroky návrhu jsou úzce propojeny a stejně jako u kroků v analýze se mezi nimi analytici často pohybují tam a zpět.

Nejdůležitějším krokem fáze návrhu je návrh jednotlivých tříd a metod. Objektově orientované systémy mohou být poměrně složité, takže analytici potřebují vytvořit instrukce a pokyny pro programátory, které jasně popisují, co systém musí dělat. Prakticky řečeno, návrh tříd a metod je oblast projektu, kde se odvede naprostá většina práce během návrhu. Bez ohledu na to, na kterou vrstvu se zaměřujete, musí být navrženy třídy, které budou použity k vytvoření systémových objektů. (Dennis, 2015; Voříšek, 2015)

1.4 Jazyk pro popis obchodních procesů

1.4.1 Co je to BPMN

Business Process Modeling Notation (BPMN) je standardním jazykem pro modelování obchodních procesů. Poskytuje grafickou notaci pro specifikaci obchodních procesů v Business Process diagramech (BPD), na základě technik vývojových diagramů, velmi podobné diagramům aktivit, které známe z UML. BPMN vytváří standardizovaný můstek, který vyplňuje mezeru mezi analýzou procesu a její následnou implementací (Von Rosing, 2015).

„Hlavním rozdílem mezi UML a BPMN je v perspektivě – UML je objektově orientovaný, zatímco BPMN je procesně orientovaný. Díky tomu je BPMN široce použitelný jak pro IT, tak pro podnikání, zatímco UML je vhodnější pro vývoj IT systémů a méně vhodný pro zlepšování procesů“ (Investskills.com, 2021).

Cílem BPMN je srozumitelnost pro všechny zúčastněné, což znamená, že jazyk používá intuitivní (ale zároveň dostatečně komplexní) notaci srozumitelnou všem zúčastněným aktérům. Mezi ně patří procesní analytici, kteří vytvářejí a zdokonalují procesy, vývojáři za jejich implementaci, obchodní manažeři i pracovníci, kteří budou denně systém využívat. Standard také poskytuje mapování mezi grafickou notací a ostatními spustitelnými jazyky, zejména Business Process Execution Language for Web Services (BPEL4WS) (Kanisová, 2006).

Specifikace BPMN 1.0 byla vydána pro veřejnost v roce 2004 skupinou Business Process Management Initiative (BPMI). Primárním cílem specifikace bylo poskytnout notaci, která je snadno srozumitelná všemi firemními uživateli. Od roku 2006 udržuje a vyvíjí standard

konsorcium Object Management Group (OMG). Pod jejím dohledem došlo k několika aktualizacím, poslední z nich je verze 2.0 vydaná v roce 2011. Cíl verze 2.0 je být jedinou notací pro tvorbu modelů podnikových procesů. Nejnovější aktualizace 2.0.2 vydaná v roce 2014 přináší pouze drobné úpravy a opravy překlepů. Hlavní přínosy, které s sebou přinesla verze 2.0 jsou:

- sjednocení s metamodelem BPDM;
- zobrazení odlišných pohledů na procesní model;
- model choreografie;
- model spolupráce;
- model komunikace (Von Rosing, 2015).

BPMN tedy poskytuje podnikům schopnost porozumět jejich interním obchodním postupům v grafickém zápisu a umožňuje organizacím tyto postupy dále komunikovat standardním způsobem (Object Management Group, 2013).

1.4.2 Syntaxe BPMN

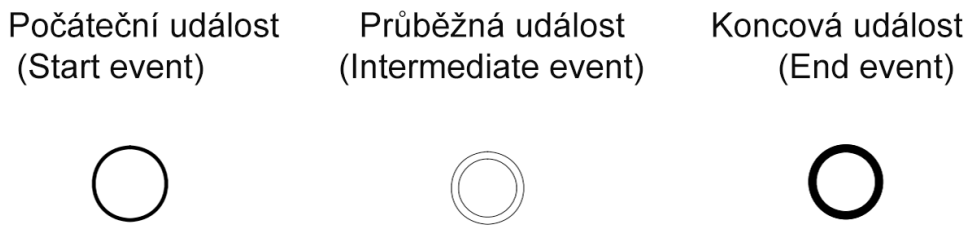
Rozeznáváme pět základních kategorií grafických prvků BPMN:

- tokové objekty (Flow Objects);
- data;
- spojovací objekty (Connecting Objects);
- plavecké dráhy (Swimlanes);
- artefakty (Artifacts) (Object Management Group, 2013).

Tokové objekty jsou hlavními grafickými prvky, které definují chování obchodního procesu. Existují tři tokové prvky (objekty):

- Události (Events) – Událost je něco, co se „stane“ v průběhu podnikového procesu nebo choreografie. Události přímo ovlivňují tok modelu a obvykle mají příčinu (trigger) nebo důsledek (result), které se značí obrázkem uvnitř kruhu. Události se značí kruhem a podle vztahu k procesu se rozlišují se tři hlavní typy:
    - Počáteční událost (Start event) – spouštěč procesu, označuje, kde konkrétní proces začíná.

- Průběžná událost (Intermediate event) – událost, která se vyskytuje mezi počáteční a koncovou událostí, ovlivňuje proces, ale nespouští ho ani ho neukončuje.
- Koncová událost (End event) – označuje, kde bude proces ukončen, představuje výsledek procesu (Object Management Group, 2013).



Obrázek 19 – BPMN události
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

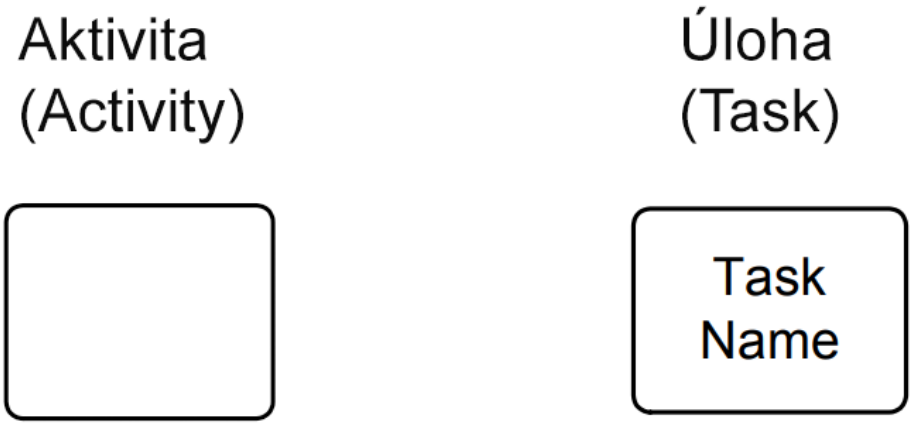
Typy událostí

Na obrázku č.20 jsou všechny druhy událostí. Piktogramy uvnitř kruhu definují příčinu (u počátečních, případně průběžných) nebo důsledek (u koncových) událostí.

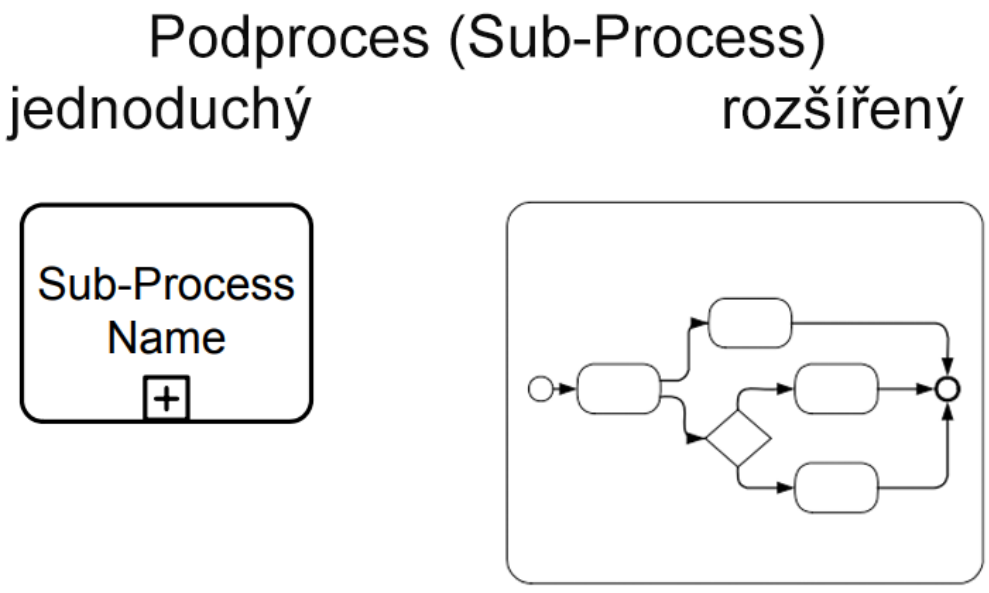
Obrázek 20 – typy BPMN událostí
Zdroj: (Object Management Group, 2013)

- Aktivity (Activities) - znázorňují místa v procesu, kde se vykonává nějaká práce. Jsou reprezentovány obdélníkem se zaoblenými rohy. Aktivity se rozlišují na:
  - Úloha (Task) – práci v procesu již nelze členit nebo rozložit na dílčí části, je atomická.

- Podproces (Sub-Process) – aktivita složena z aktivit, bran, událostí a sekvenčních toků. Používá se při potřebě složení více aktivit do jednoho celku a pro skrytí dalších úrovní procesu (Object Management Group, 2013).



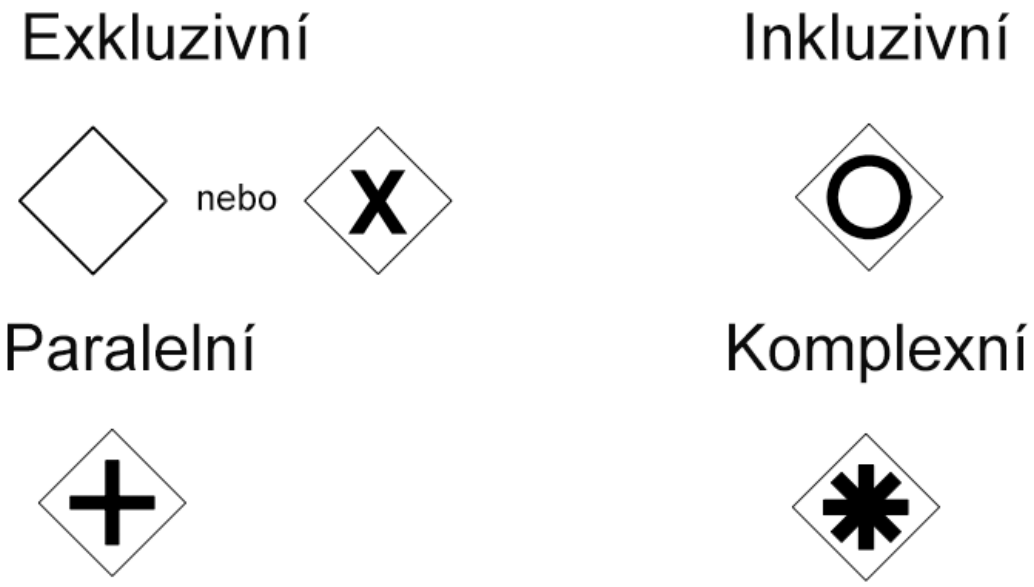
Obrázek 21 – BPMN aktivita a úloha
Zdroj: vlastní zpracování dle (Object Management Group, 2013)



Obrázek 22 – BPMN podproces
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

- Brány (Gateways) – používají se pro kontrolu divergence (rozdělení) a konvergence (sloučení). Vnitřní značky označují typ kontroly toku.

- Exkluzivní brány – rozhodnutí, proces může pokračovat pouze jednou ze dvou a více možných cest. Používá se i exkluzivní konvergující (sbíhající se) ke sloučení alternativních cest do jedné.
- Inkluzivní – divergenční brány slouží k výběru jedné anebo více cest, vytváří tedy možnost paralelních cest. Na rozdíl od exkluzivních bran jsou vyhodnoceny všechny možné cesty a ty, které splňují podmínku pro pokračování procesu nevylučují další takové větve. Inkluzivní konvergující brána může synchronizovat kombinaci alternativních a paralelních cest.
- Paralelní – vytváří paralelní cesty, aniž by kontrolovaly podmínky. Konvergující varianta brány počká na všechny příchozí toky, před spuštěním odchozího toku.
- Komplexní – mohou být použity pro rozdělení či sloučení komplikovaného scénáře procesního toku. V BPMN se moc často nepoužívají, protože výše zmíněné brány zvládají většinu situací a jsou jednodušší a srozumitelnější. Komplexní brána využívá „aktivační podmínku“, která může vyjadřovat jakékoli pravidlo např: brána se aktivuje po přijetí tří tokenů. Tokeny, které dorazí se zpožděním můžou bránu resetovat (Object Management Group, 2013).



Obrázek 23 - BPMN brány
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

Data jsou reprezentována čtyřmi prvky:

- Datové objekty (Data Objects) – poskytují informace o tom, co aktivity vyžadují, aby bylo provedeno anebo co vytvářejí.
- Datové vstupy (Data Inputs) – znázorňují požadavky na data, která aktivity a procesy potřebují pro vykonávání činnosti.
- Datové výstupy (Data Outputs) – zachycují data, která jsou aktivitou nebo procesem produkovaná.
- Datové sklady (Data Stores) – způsob, kterým aktivity získávají a aktualizují informace uložené mimo hranice procesu (Object Management Group, 2013).

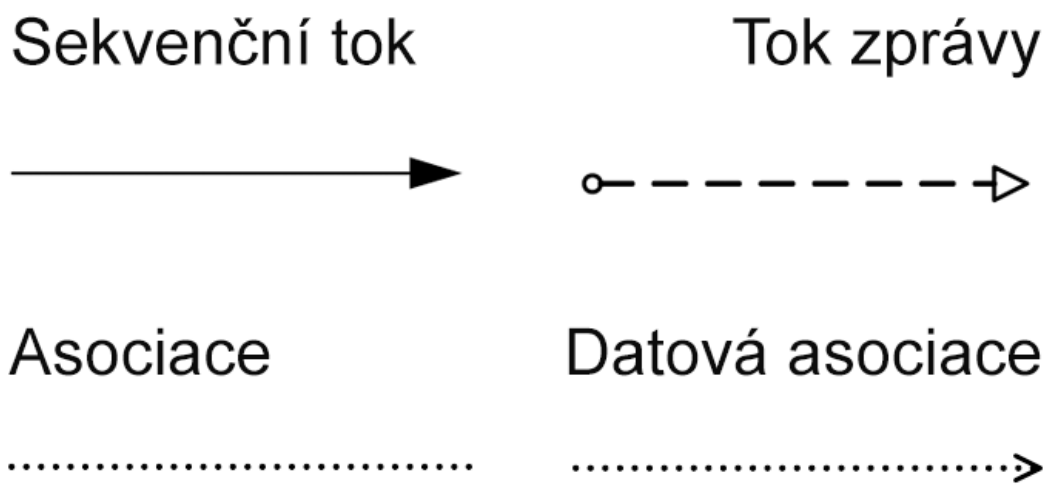
Obrázek 24 – BPMN data
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

Spojovací objekty slouží k vzájemnému propojování tokových objektů, případně dalších informací a tím tvoří základní strukturu (kostru) podnikového procesu. Rozpoznáváme čtyři typy:

- Sekvenční toky (Sequence Flows) – určují pořadí v jakém budou jednotlivé události, aktivity a brány procesu vykonávány. Každý tok má vždy jen jeden zdroj a jeden cíl a nesmí překročit hranice bazénu. Sekvenční toky se značí nepřerušovanou čarou s vyplněnou šipkou, která udává směr na jednom konci.
- Toky zpráv (Message Flows) – znázornění přenosu zpráv mezi dvěma účastníky. Zprávy proudí vždy přes hranice bazénu, a odesílat a přijímat je mohou buď tokové

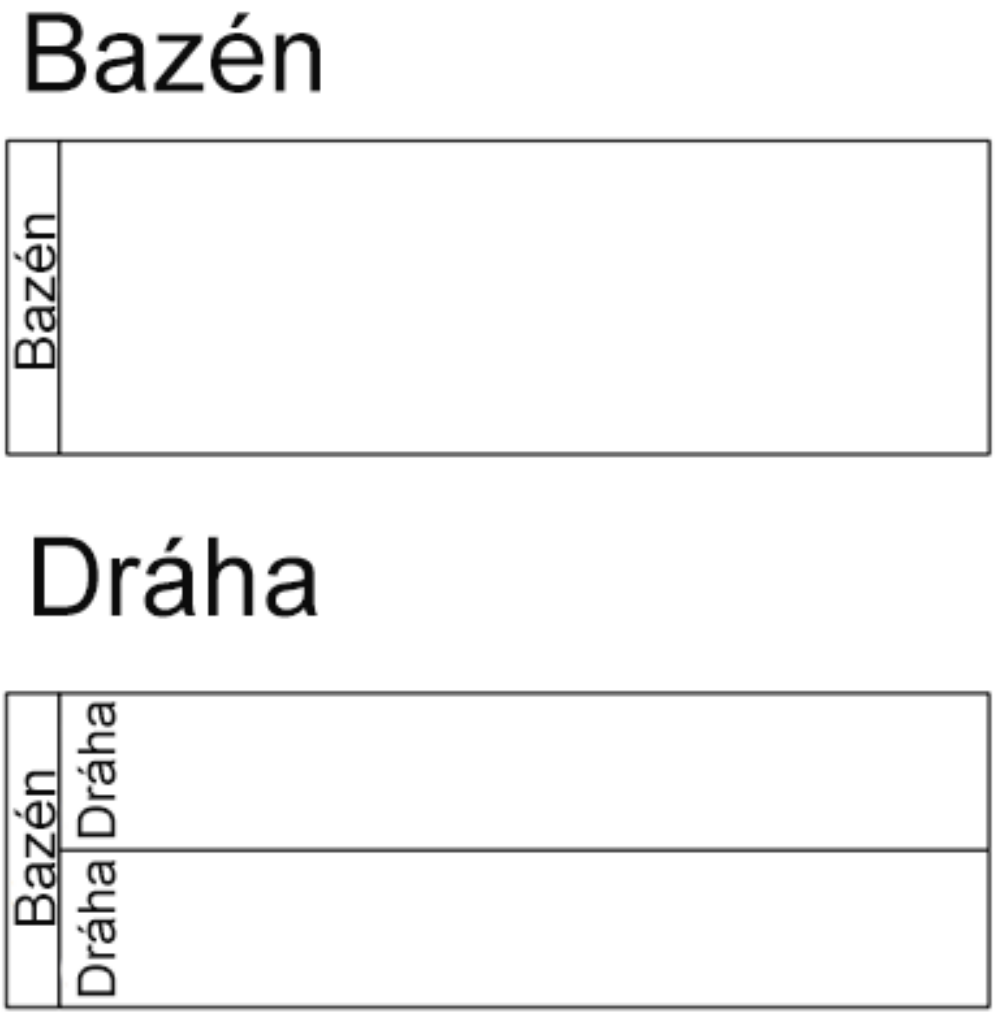
objekty, nebo samotné bazény. Jsou reprezentovány přerušovanou čarou s prázdnou šipkou na jednom konci a prázdným kruhem na začátku.

- Asociace (Associations) – používá se k propojení tokových objektů s dodatečnou informací, textem, artefaktem. Od verze BPMN 2.0 se asociace značí přerušovanou čarou bez šipky. Notaci směrové asociace s otevřenou šipkou na konci, používané v předchozích verzích využívá od verze 2.0 datová asociace.
- Datové asociace (Data Associations) – slouží k přenosu dat mezi datovými objekty, výstupy a vstupy aktivit i celými procesy. Značí se přerušovanou čarou s otevřenou šipkou na konci (Object Management Group, 2013).



Obrázek 25 – BPMN spojovací objekty
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

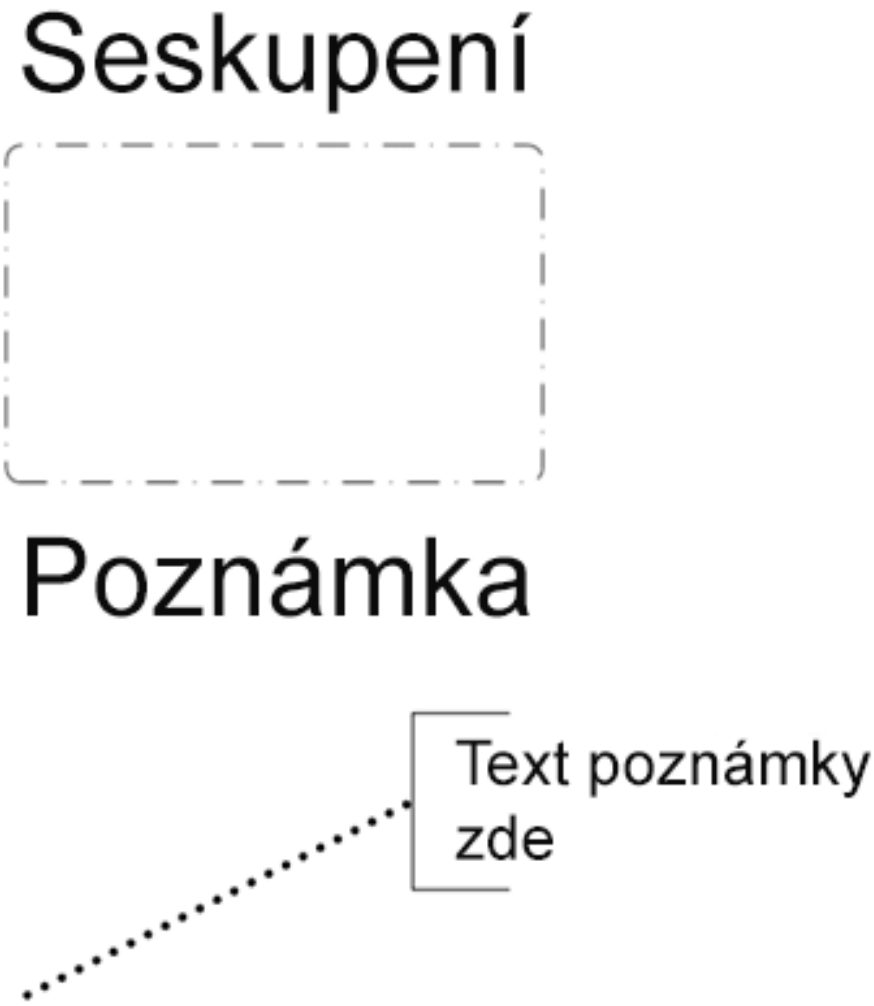
- Bazény (Pools) – grafická reprezentace účastníka procesu v rámci kolaborace. Roli účastníka většinou hraje nějaký business partner např. prodejce, nakupující, nebo výrobce. Bazén má jednu nebo více drah a v jednom bazénu se nachází pouze jeden konkrétní proces. Sekvenční toky mohou křížit jednotlivé dráhy, ale nesmí opustit hranice bazénu.
- Dráhy (Lanes) – podmnožina bazénu (tedy i procesu). Používá se k organizaci aktivit uvnitř bazénu na základě rolí a funkcí.



Obrázek 26 – BPMN bazén a dráhy
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

Artefakty poskytují dodatečné informace o procesu. Existují dva standardizované artefakty, nicméně vývojáři mohou v případě potřeby přidat další. Aktuální sada artefaktů zahrnuje:

- Seskupení (Group) – mechanismus pro neformální seskupení prvků diagramu, nemá žádný vliv na sekvenční toky. Nejčastěji se používají ke zvýraznění určitých částí diagramu např. pro potřeby analýzy výkazů.
- Poznámka (Text Annotation) – způsob jakým může tvůrce diagramu poskytnout doplňující informace pro čtenáře (zákazník, management apod.). Anotace může být připojena ke konkrétnímu objektu pomocí asociace. Stejně jako seskupení neovlivňuje tok procesu (Object Management Group, 2013).



Obrázek 27 – BPMN artefakty
Zdroj: vlastní zpracování dle (Object Management Group, 2013)

Procesy se dělí na tři základný typy:

- Soukromé spustitelné (vnitřní) podnikové procesy – interní specifické organizaci. Tyto procesy se běžně nazývají „workflow“ nebo „BPM procesy“, v oblasti webových služeb také „orchestrace“. Spustitelné procesy jsou modelovány za účelem spustitelnosti, tedy že je bude možné po dokončení modelu spustit.
- Soukromé nespustitelné (vnitřní) podnikové procesy – vnitřní proces, který je modelován za účelem dokumentace chování procesu na úrovni detailů definované modelářem. Tedy informace potřebné pro spuštění, jako například výrazy formálních podmínek obvykle nejsou zahrnuty v nespustitelném procesu.

- Veřejné procesy – zobrazují interakce mezi soukromými podnikovými procesy a jinými účastníky nebo procesy. Ve veřejných procesech jsou zahrnuty pouze ty aktivity, které komunikují s dalšími účastníky, všechny ostatní (interní) aktivity jsou soukromé, tudíž se nezahrnují. Tyto aktivity jsou takzvané „body doteku“, mezi účastníky. Veřejné procesy ukazují vnější pohled na zprávy a jejich pořadí, které jsou potřebné při komunikaci s konkrétním byznys procesem. Veřejné procesy se často modelují jako součást kolaborace, ale není to podmínkou (Von Rosing, 2015).

Choreografie je typ procesu, liší se ale v účelu a v chování od standardního procesu. Zatímco standardní proces je běžně známější a definuje tok aktivit organizace, choreografie formalizuje způsob, jakým účastníci koordinují své interakce. Důraz není kladen na orchestraci práce, kterou tito účastníci odvádějí, ale spíše na to, jak mezi sebou komunikují, přijímají a odesílají zprávy (Object Management Group, 2013).

Kolaborace znázorňuje interakce mezi dvěma nebo více subjekty. Obvykle obsahuje dva nebo více bazénů, reprezentující účastníky kolaborace. Výměna zpráv mezi účastníky je znázorněna toky zpráv, které spojují bazény, nebo objekty uvnitř bazénů mezi sebou. Kolaborace může být zobrazena jako dva a více veřejných a/nebo soukromých procesů, které spolu komunikují. V kolaboracích jsou povoleny všechny možné kombinace bazénů, procesů a choreografií (Von Rosing, 2015).

2 Analytická část – analýza informačního systému, vytvoření vývojové dokumentace

Analýza je realizovaná technikami a postupy vysvětlených v předchozí části práce. K realizování vývojových diagramů byl použit CASE nástroj Enterprise Architect. Informační systém bankomatu, který je v této práci analyzován a později i navrhnut je inspirovaný knihou (Deitel, 2010). Vhodných příkladů pro studenty není mnoho a zjednodušený systém skutečného bankomatu poskytuje perfektní možnost se naučit a prakticky si osvojit všechny aspekty vývoje informačních systémů.

2.1 Popis informačního systému/dokument požadavků

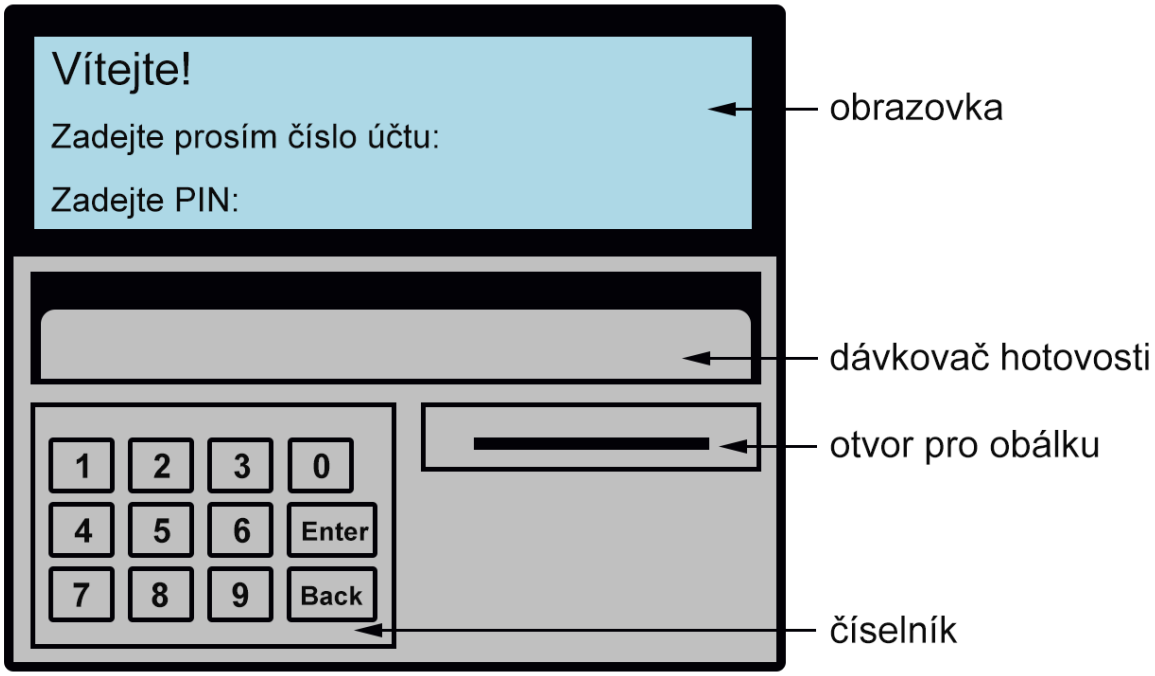
Následující kapitola představuje popis informačního systému a zjednodušený dokument požadavků. Takový dokument je obvykle výsledkem důkladného procesu shromažďování požadavků, který může zahrnovat rozhovory s potenciálními uživateli systému a specialisty v

oblastech souvisejících se systémem. Pro účely této práce je dostačující obecnější popis požadavků.

Banka plánuje instalaci nového bankomatu (ATM), který klientům banky umožní provádět základní finanční operace. Uživatelé bankomatu by měli mít možnost zobrazit zůstatek na účtu, vybrat hotovost a vložit hotovost.

Uživatelské rozhraní bankomatu obsahuje následující hardwarové komponenty:

- Obrazovka, která zobrazuje zprávy a tím obstarává komunikaci s uživatelem.
- Číselník, který přijímá numerický vstup od uživatele.
- Dávkovač na vybírání hotovosti.
- Otvor pro vložení hotovosti.



Obrázek 28 - Rozhraní bankomatu
Zdroj: vlastní zpracování dle (Deitel, 2010)

Vzhledem k omezenému rozsahu této případové studie některé prvky zde popsaného bankomatu zjednodušují různé aspekty skutečného bankomatu.

- Obsluha naplní bankomat každé ráno hotovostí, a to v podobě pět-seti bankovek, každé v hodnotě dvaceti dolarů.

- Každý uživatel může mít pouze jeden účet.
- Vkládaná hotovost se musí nejdříve vložit do obálky.
- Bankomat nedisponuje funkcí, která načte číslo bankovního účtu z vložené karty, uživatel tedy musí číslo účtu zadat pomocí číselníku.
- Bankomat neumí tisknout účtenky, všechen výstup se zobrazí pouze na obrazovce.
- Banka plánuje postavit pouze jeden bankomat, tudíž do databáze banky nemůže přistupovat více bankomatů v jednu chvíli.
- Banka neprovádí změny informací v databázi bankovních účtů, ve stejnou chvíli, kdy se bankomat k databázi připojuje.

Relace bankomatu začíná autentizací uživatele (tj. prokázání identity uživatele) na základě čísla účtu a osobního identifikačního čísla (PIN), po které následuje vytvoření a provedení finančních transakcí. K ověření uživatele a provádění transakcí musí bankomat spolupracovat s databází bankovních účtů banky. Pro každý bankovní účet je v databázi uloženo číslo účtu, PIN a peněžní zůstatek.

Po přistoupení k bankomatu by měl uživatel zaznamenat následující sled událostí:

1. Na obrazovce se zobrazí uvítací zpráva a vyzve uživatele k zadání čísla účtu
2. Uživatel zadá pomocí klávesnice pětimístné číslo účtu
3. Pro účely ověření se na obrazovce zobrazí výzva k zadání kódu PIN spojeného se zadaným číslem účtu
4. Uživatel zadá pomocí klávesnice pětimístný PIN
5. Pokud uživatel zadá platné číslo účtu a správný PIN k tomuto účtu, na obrazovce se zobrazí hlavní nabídka
6. Pokud uživatel zadá neplatné číslo účtu nebo nesprávný PIN, na obrazovce se zobrazí příslušná zpráva a bankomat se vrátí ke kroku 1 a restartuje proces ověřování.

Poté, co bankomat autentizuje uživatele, zobrazí se hlavní nabídka s očíslovanými možnostmi pro každý ze tří typů transakcí: dotaz na zůstatek (možnost 1), výběr hotovosti (možnost 2), vklad (možnost 3). Poslední možnost umožňuje uživateli opustit systém (možnost 4). Uživatel si pak vybere, zda provede transakci (stisknutím kláves 1, 2 nebo 3) nebo opustí systém

(stisknutím 4). Pokud uživatel zadá neplatnou možnost, na obrazovce se zobrazí chybová hláška a poté se znovu zobrazí hlavní nabídka.

Obrázek 29 - Hlavní menu
Zdroj: vlastní zpracování dle (Deitel, 2010)

Pokud uživatel stiskne klávesu 1 pro zjištění zůstatku, na obrazovce mu zobrazí zůstatek na jeho učtu. K provedení této operace se musí bankomat připojit k databázi banky a z ní načíst zůstatek z příslušného učtu.

K následujícím akcím dojde, pokud uživatel stiskne klávesu s číslicí 2:

1. Na obrazovce se zobrazí menu obsahující běžně vybírané částky: $20 (možnost 1), $40 (možnost 2), $60 (možnost 3), $100 (možnost 4) a $200 (možnost 5). Poslední možností v menu je možnost 6, která transakci zruší.
2. Uživatel vybere pomocí číselníku jednu z možností (1-6).
3. Pokud uživatel zvolí částku, která je vyšší než zůstatek na jeho účtu, zobrazí se na obrazovce zpráva, která ho na to upozorní a bankomat se vrátí k prvnímu kroku. Pokud je zvolená částka výběru nižší nebo rovna zůstatku na účtu uživatele (tj. přijatelná částka výběru), bankomat přejde ke kroku 4. Pokud se uživatel rozhodne zrušit transakci (možnost 6), bankomat zobrazí hlavní menu a čeká na zadání uživatele.

4. Pokud má bankomat dostatek hotovosti pro uspokojení požadavku, přejde na další krok. V opačném případě se na obrazovce zobrazí zpráva informující uživatele o nedostatku bankovek s prosbou, aby zvolil menší částku výběru. Bankomat se poté vrátí k prvnímu kroku.
5. Bankomat odečte částku výběru ze zůstatku účtu uživatele v databázi banky.
6. Bankomat vydá požadovanou částku uživateli pomocí dávkovače.
7. Na obrazovce se zobrazí zpráva připomínající uživateli, aby si vzal své peníze.

Následující akce nastanou, když uživatel zvolí vklad hotovosti (možnost 3 z hlavního menu):

1. Na obrazovce se zobrazí výzva k zadání částky vkladu, nebo zadání nuly pro ukončení transakce.
2. Uživatel zadá částku, kterou chce vložit, případně nulu pro zrušení.
3. Pokud uživatel správně zadal částku k vložení, bankomat pokračuje na krok číslo 4. Pokud uživatel zadal nulu, bankomat zobrazí hlavní menu.
4. Obrazovka zobrazí výzvu k vložení obálky s hotovostí do vkládacího otvoru.
5. Pokud bankomat obdrží obálku s hotovostí do dvou minut, připíše se částka vkladu na účet uživatele v databázi banky. (Banka samozřejmě musí ověřit, jestli částka v obálce odpovídá částce zadané uživatelem, proto by v takovém případě uživatel mohl vloženou částku vybrat až po ověření bankou. K tomuto kroku dochází nezávisle na systému bankomatu, proto se jím nebudeme zabývat.) Pokud bankomat neobdrží vkladovou obálku do dvou minut, na obrazovce se zobrazí zpráva o zrušení transakce u důvodu nečinnosti. Bankomat se poté vrátí do hlavní nabídky.

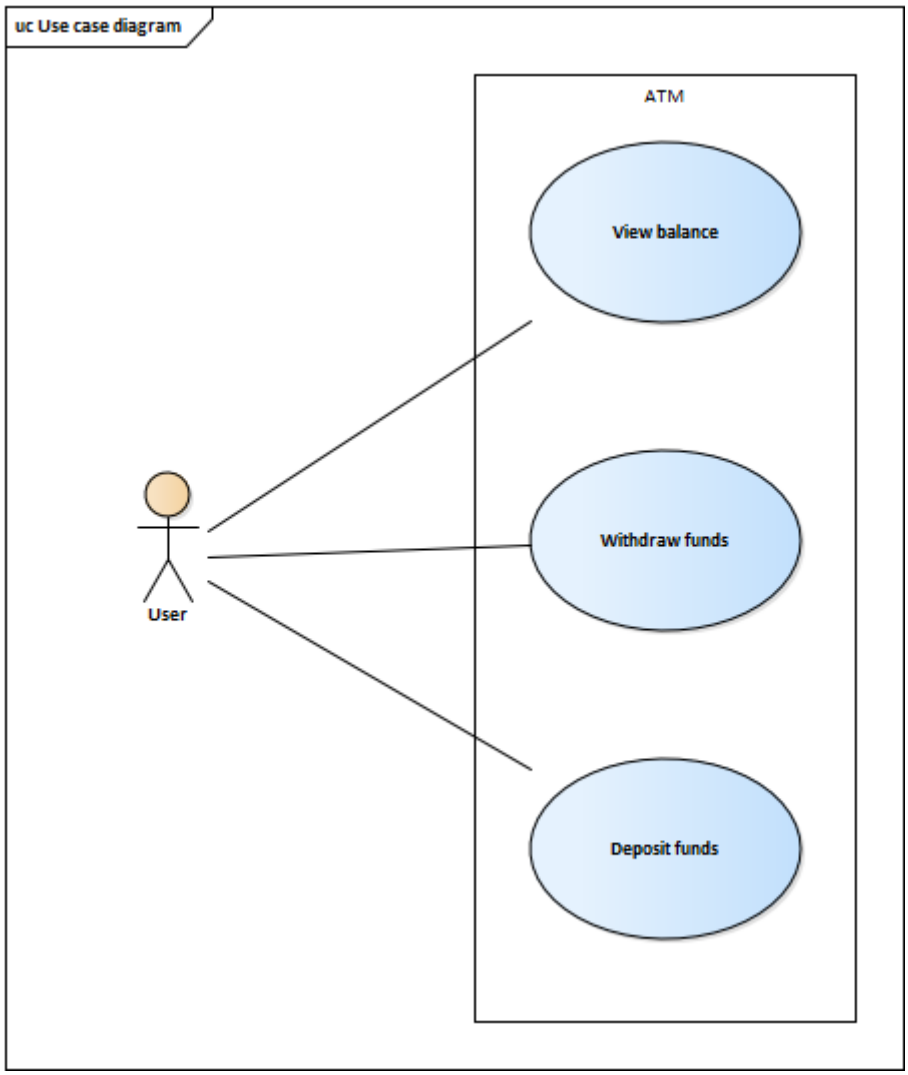
Poté, co systém úspěšně provede transakci, zobrazí znovu hlavní menu, aby mohl uživatel provádět další operace. Pokud se uživatel rozhodne ukončit relaci (zadáním možnosti 4), na obrazovce se zobrazí zpráva s poděkováním a potvrzením odhlášení. Po několika vteřinách se zobrazí uvítací zpráva pro příštího uživatele.

2.2 UML modely

2.2.1 Diagram případu užití

Obrázek č.30 zobrazuje diagram případu užití našeho bankomatu. Figurka znázorňuje aktéra (v našem případě zákazníka banky), v roli vnějšího činitele, který interaguje se systémem

bankomatu. Náš bankomat má tři základní případy užití – zobrazit zůstatek, vybrat hotovost a vložit hotovost. Cílem je ukázat, jak uživatel interaguje se systémem bez poskytnutí podrobností – ty jsou vykresleny v dalších diagramech.



Obrázek 30 - Diagram případu užití
Zdroj: vlastní zpracování dle (Deitel, 2010)

2.2.2 Diagram tříd

Abychom mohli sestavit diagram tříd, musíme nejprve třídy identifikovat, a to tak, že z dokumentu požadavků vybereme klíčová podstatná jména a fráze. V tomto kroku se může stát, že některá podstatná jména či fráze neodpovídají částem systému, a proto nebudou vůbec modelována. Ne všechny třídy můžou být identifikovány v tomto kroku, takže není neobvyklé, pokud se na další třídy narazí až v dalších krocích návrhu systému. Z některých pojmů nevzniknou samotné třídy, ale atributy tříd.

Tabulka 2 – fráze, třídy
Zdroj: vlastní zpracování dle (Deitel, 2010)

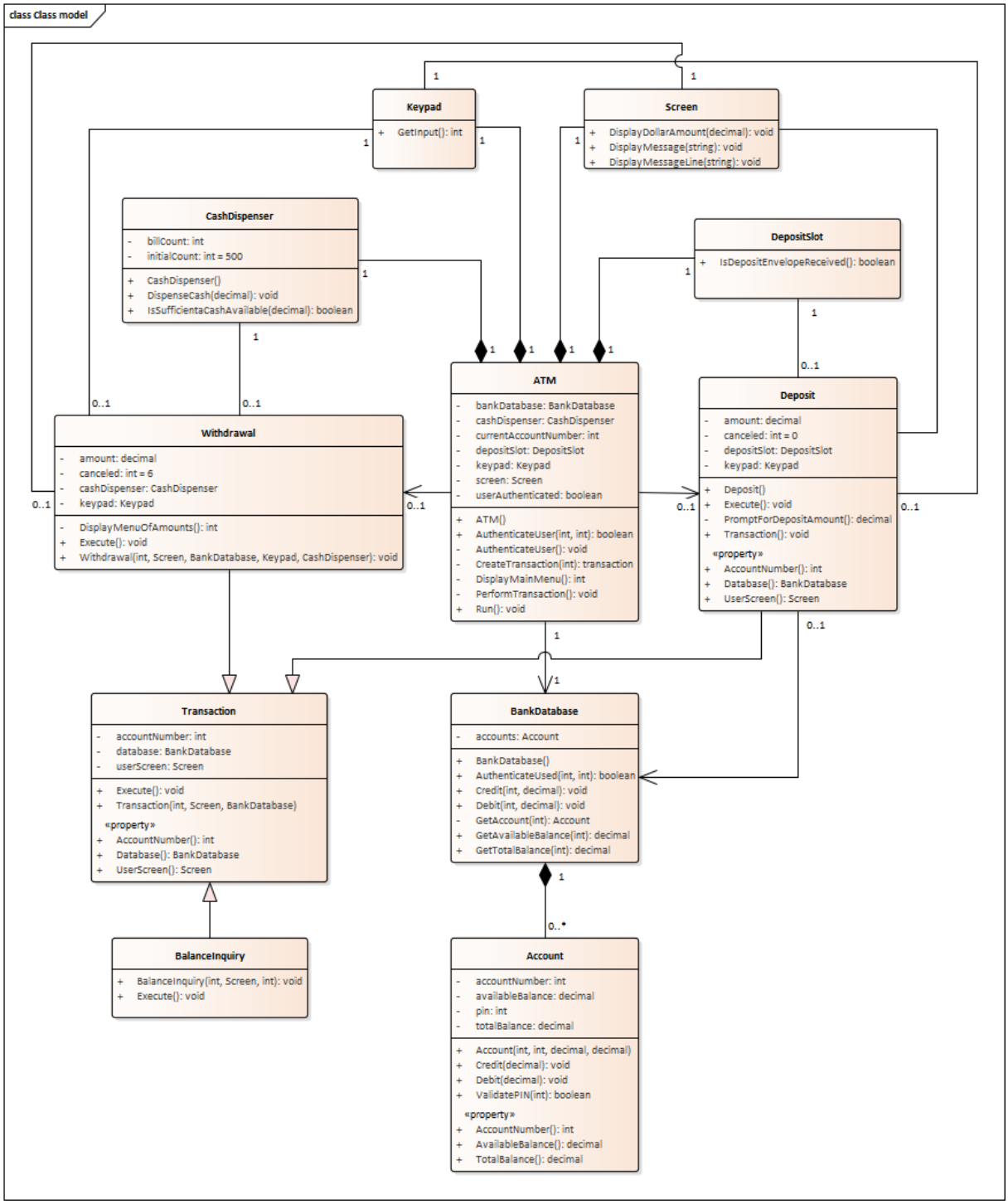
Bankovní účet	bankomat	hotovost
banka	obrazovka	uživatel
PIN	číselník	zákazník
Výběr	Vklad	Dávkovač na hotovost
Transakce	Databáze banky	Otvor pro vklad
Bankovní účet	$20 bankovka	peníze
Zůstatek na účtu	Obálka	Dotaz na zůstatek

V tabulce jsou zvýrazněna podstatná jména, která mají pro náš systém bankomatu význam a z kterých vytvoříme třídy.

Některé pojmy v tabulce jsou sice relevantní pro náš systém bankomatu, ale nehodí se pro vytvoření třídy např. banka, uživatel, zákazník (nejsou součástí systému bankomatu), $20 bankovka, obálka (nejsou součástí toho, co se snažíme automatizovat). Pojmy jako PIN a číslo účtu budou ideální atributy třídy bankovní účet.

Každá třída má své atributy (data) a operace (chování). Dalším krokem tedy bude identifikace a přiřazení atributů. Jednoduchý způsob, identifikace atributů spočívá v hledání popisných slov a frází v dokumentu požadavků. Pro každé takové slovo, či frázi můžeme rovnou vytvořit atribut a přiřadit ho k jedné, nebo více třídám. Vytváříme také atributy, které reprezentují jakákoli další data, které může třída potřebovat.

Posledním krokem je identifikace operací, potřebné pro zavedení systému. Operace je něco služba, kterou objekty třídy poskytují klientům třídy. Mnoho operací tříd v našem bankomatu můžeme odvodit zkoumáním sloves a slovesných frází v dokumentu požadavků. Každou z nich pak přiřadíme ke konkrétním třídám v našem systému.

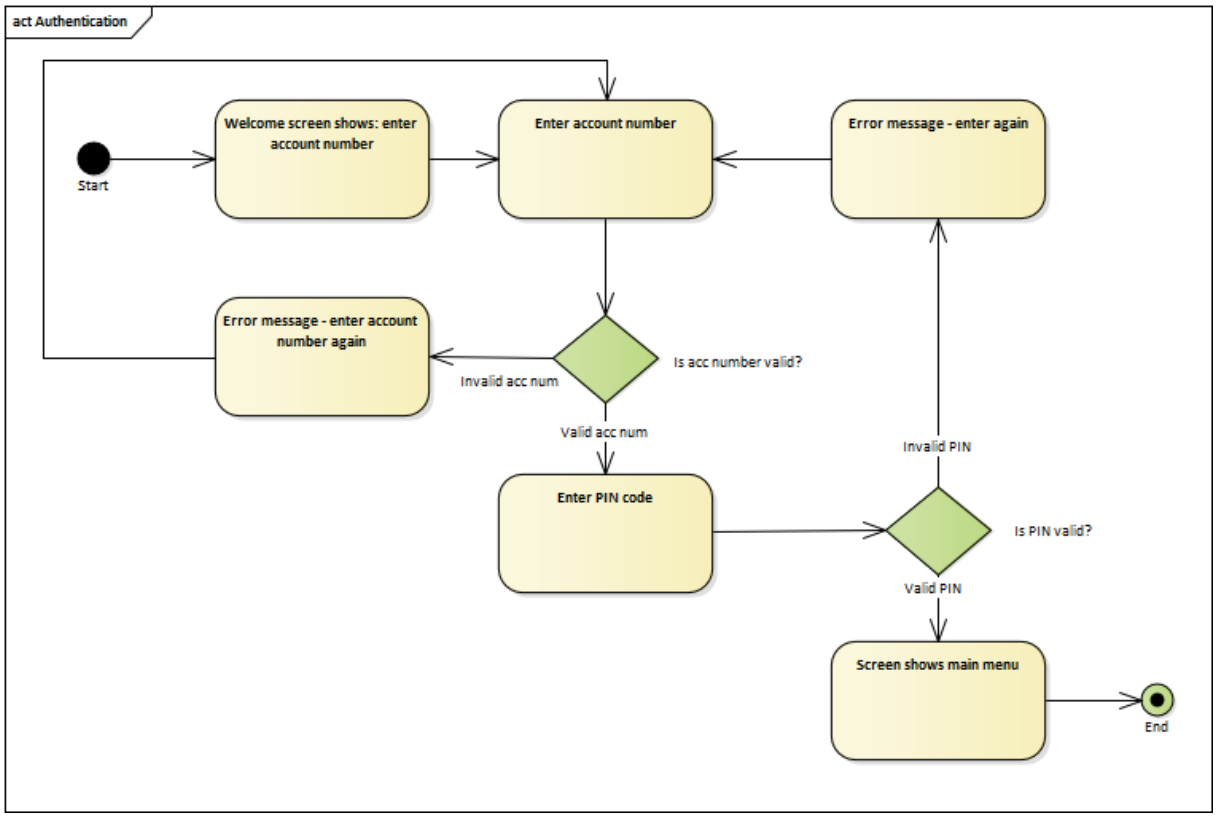


Obrázek 31 - Diagram tříd
Zdroj: vlastní zpracování dle (Deitel, 2010)

2.2.3 Diagramy aktivit

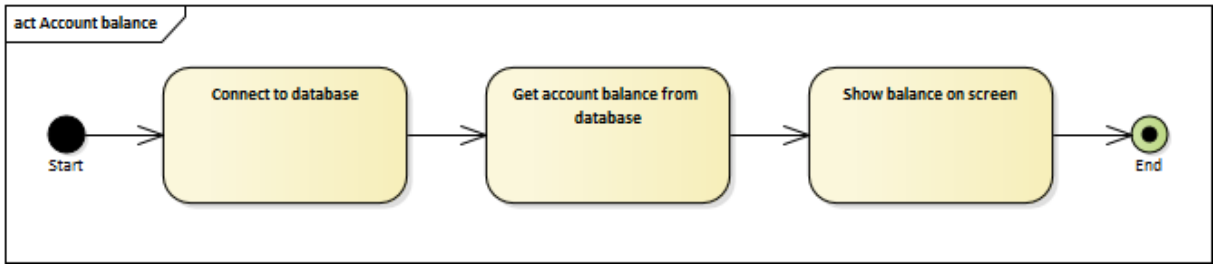
Diagram ověření uživatele znázorňuje sekvenci akcí, ke kterým musí dojít pro vpuštění uživatele do hlavního menu bankomatu. První aktivitou procesu je zobrazení uvítací zprávy na obrazovce a výzva k zadání osobních údajů. Po zadání čísla účtu uživatel zadá PIN, což posune proces do kroku ověření správnosti PINu a čísla účtu. V tomto kroku tedy dojde k rozhodnutí,

pokud jsou údaje správné, pokračuje proces do poslední aktivity – zobrazení hlavního menu. Pokud je PIN (nebo číslo účtu) chybný, následuje aktivita, který zobrazí chybovou hlášku, proces se částečně opakuje.



Obrázek 32 – Diagram aktivit Authentication
Zdroj: vlastní zpracování dle (Deitel, 2010)

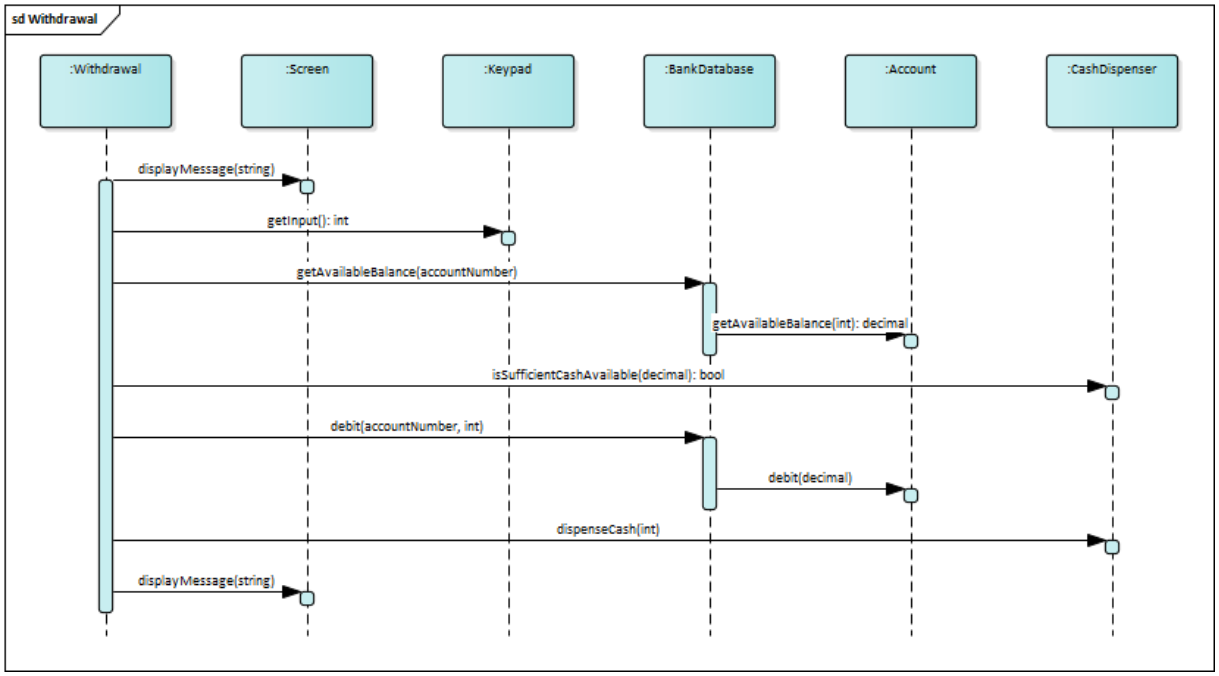
Dotaz na zůstatek je zdaleka nejjednodušší ze tří transakcí našeho bankomatu. Skládá se pouze ze tří aktivit a nedochází k rozvětvení. V první aktivitě se bankomat připojí v databázi banky. Následně z databáze získá informaci o zůstatku, kterou v posledním kroku zobrazí na obrazovce.



Obrázek 33 – Diagram aktivit Account balance
Zdroj: vlastní zpracování dle (Deitel, 2010)

2.2.4 Sekvenční diagramy

Sekvence zpráv na obrázku č.33 začíná, když objekt Withdrawal vyzve uživatele k vybrání částky, odesláním zprávy objektu Screen. Další zprávou odeslanou objektu Keypad obdrží vstup od uživatele. Po obdržení částky výběru odešle objekt Withdrawal zprávu objektu BankDatabase, která obratem odešle zprávu objektu Account. Za předpokladu že má účet uživatele dostatečný zůstatek k provedení transakce, objekt Withdrawal pošle zprávu objektu CashDispenser zda je v bankomatu dostatek bankovek. Pokud ano objekt Withdrawal odešle zprávu objektu BankDatabase a ten zprávou objektu Account odečte peníze z účtu. Konečně objekt Withdrawal odešle zprávu objektu CashDispenser, který umožní výběr hotovosti. Poslední zpráva směruje objektu Screen, který zobrazí potvrzení pro zákazníka.



Obrázek 34 – Sekvenční diagram Withdrwal
Zdroj: vlastní zpracování dle (Deitel, 2010)

3 Návrhová část – případová studie analýzy, návrh IS pro výukové účely

Tato část práce obsahuje kompletní funkční implementaci systému bankomatu naprogramovanou v jazyce C#. Celý projekt se skládá z 11 tříd, které jsme identifikovali v analytické části a necelých 500 řádků kódu. Základem oporou procesu programování systému je navržený diagram tříd. Rozvinout obsah tříd nám pomůžou především diagramy aktivit a sekvenční diagramy. Náš návrh projektu nespecifikuje veškerou logiku programu a nemusí specifikovat všechny atributy a operace potřebné k dokončení implementace funkčního

bankomatu. Toto je normální součást procesu objektově orientovaného návrhu. Při implementaci systému dokončujeme programovou logiku a přidáváme atributy a chování podle potřeby pro konstrukci systému bankomatu specifikovaného v dokumentu požadavků.

Vzhledem k omezenému rozsahu práce jsou v této kapitole popsány vybrané tři třídy: ATM, Screen a Account. Celý projekt je k dispozici jako příloha.

3.1 Třída ATM

Třída ATM reprezentuje bankomat jako takový a je to zároveň nejobsáhlejší třída. Řádky 3–9 implementují atributy třídy. Řádky 10–16 deklarují výčet, který odpovídá čtyřem možnostem hlavního menu. Na řádcích 17–26 je konstruktor třídy ATM, který inicializuje atributy třídy, tedy nastavuje jejich počáteční hodnoty.

```
1  public class ATM
2  {
3     private bool userAuthenticated;
4     private int currentAccountNumber;
5     private Screen screen;
6     private Keypad keypad;
7     private CashDispenser cashDispenser;
8     private DepositSlot depositSlot;
9     private BankDatabase bankDatabase;
10    private enum MenuOption
11    {
12       BALANCE_INQUIRY = 1,
13       WITHDRAWAL = 2,
14       DEPOSIT = 3,
15       EXIT_ATM = 4
16    }
17    public ATM()
18    {
19       userAuthenticated = false;
20       currentAccountNumber = 0;
21       screen = new Screen();
22       keypad = new Keypad();
23       cashDispenser = new CashDispenser();
24       depositSlot = new DepositSlot();
25       bankDatabase = new BankDatabase();
26    }
```

Obrázek 35 - Třída ATM (1.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Řádky 27–41 implementuje veřejnou metodu třídy ATM, která umožňuje externí třídě ATMCaseStudy spustit bankomat. Metoda používá nekonečný cyklus k opakovanému přivítání uživatele, pokusu o ověření uživatele a, pokud ověření uspěje, umožní uživateli provádět

transakce. Uvnitř nekonečného cyklu je podmínka, která opakovaně vítá a snaží se ověřit uživatele, dokud uživatel není ověřen (řádky 31–35).

```
27    public void Run()
28    {
29       while ( true )
30       {
31          while ( !userAuthenticated )
32          {
33             screen.DisplayMessageLine( "\nVítejte!" );
34             AuthenticateUser();
35          }
36          PerformTransactions();
37          userAuthenticated = false;
38          currentAccountNumber = 0;
39          screen.DisplayMessageLine( "\nDěkujeme! Na shledanou!" );
40       }
41    }
```

Obrázek 36 - Třída ATM (2.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Metoda AuthenticateUser na řádcích 42–55 obsahuje kroky nutné k ověření uživatele před umožněním transakce. Uživatel je vyzván k zadání čísla účtu a PIN pomocí metody DisplayMessage třídy screen a jeho vstup je zachycen metodou GetInput třídy keypad (řádky 44–47). Řádky 48 a 49 se pokoušejí ověřit uživatele tím, že předají zadané údaje metodě AuthenticateUser třídy bankDatabase. Pokud ověření proběhne, hodnota atributu userAuthenticated se změní na true. Pokud je tato hodnota true, řádky 50–54 uloží zadané číslo účtu do atributu currentAccountNumber, se kterým pracují další metody této třídy.

```
42    private void AuthenticateUser()
43    {
44       screen.DisplayMessage( "\nZadejte prosím Vaše číslo účtu: " );
45       int accountNumber = keypad.GetInput();
46       screen.DisplayMessage( "\nZadejte Váš PIN: " );
47       int pin = keypad.GetInput();
48       userAuthenticated =
49          bankDatabase.AuthenticateUser( accountNumber, pin );
50       if  ( userAuthenticated )
51          currentAccountNumber = accountNumber;
52       else
53          screen.DisplayMessageLine(
54             "Chybně zadané číslo účtu nebo PIN, zkuste to znovu." );
55    }
```

Obrázek 37 - Třída ATM (3.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Řádky 56–82 zobrazují metodu PerformTransaction, která zahrnuje akce pro ověřeného uživatele. Řádek 58 deklaruje lokální proměnnou currentTransaction, která reprezentuje jednu ze tří zrovna zpracovávaných transakcí. V dalším řádku deklarujeme proměnnou userExited, která sleduje, zda uživatel zvolil možnost opustit systém. Tato proměnná řídí smyčku na řádcích 60–81 a umožňuje uživateli provádět transakce, dokud se nerozhodne pro ukončení. Řádek 62 uvnitř smyčky zobrazí hlavní menu zavoláním metody DisplayMainMenu (definovanou na řádcích 83–92 a získá volbu uživatele, kterou uloží do proměnné mainMenuSelection. Řádky 63–80 reprezentují switch (přepínač), který dle výběru uživatele provede příslušnou akci.

```
54              "Chybně zadané číslo účtu nebo PIN, zkuste to znovu." );
55      }
56      private void PerformTransactions()
57      {
58         Transaction currentTransaction;
59         bool userExited = false;
60         while ( !userExited )
61         {
62            int mainMenuSelection = DisplayMainMenu();
63            switch ((MenuOption)mainMenuSelection)
64            {
65               case MenuOption.BALANCE_INQUIRY:
66               case MenuOption.WITHDRAWAL:
67               case MenuOption.DEPOSIT:
68                  currentTransaction =
69                     CreateTransaction( mainMenuSelection );
70                  currentTransaction.Execute();
71                  break;
72               case MenuOption.EXIT_ATM:
73                  screen.DisplayMessageLine("\nSystém se ukončuje...");
74                  userExited = true;
75                  break;
76               default:
77                  screen.DisplayMessageLine(
78                     "\nNevybrali jste platnou položku. Zkuste to
                     znovu." );
79                  break;
80            }
81         }
82      }
```

Obrázek 38 - Třída ATM (4.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Při výběru jedné ze tří transakcí se vyvolá metoda CreateTransaction (definovaná na rádcích 93–112) a zároveň metodu Execute, čímž se provede zvolená transakce. Pokud uživatel zvolí možnost 4, systém se ukončí (72–75). Řádky 76–79 řeší situaci, kdy vstup neodpovídá žádné z možností menu.

```
83     private int DisplayMainMenu()
84     {
85        screen.DisplayMessageLine( "\nHlavní menu:" );
86        screen.DisplayMessageLine( "1 - Zobrazit zůstatek" );
87        screen.DisplayMessageLine( "2 - Vybrat hotovost" );
88        screen.DisplayMessageLine( "3 - Vložit hotovost" );
89        screen.DisplayMessageLine( "4 - Odejít\n" );
90        screen.DisplayMessage( "Vyberte položku z menu: " );
91        return keypad.GetInput();
92     }
93     private Transaction CreateTransaction( int type )
94     {
95        Transaction temp = null;
96        switch ( ( MenuOption ) type )
97        {
98           case MenuOption.BALANCE_INQUIRY:
99              temp = new BalanceInquiry( currentAccountNumber,
100                screen, bankDatabase);
101             break;
102          case MenuOption.WITHDRAWAL:
103             temp = new Withdrawal( currentAccountNumber, screen,
104                bankDatabase, keypad, cashDispenser);
105             break;
106          case MenuOption.DEPOSIT:
107             temp = new Deposit( currentAccountNumber, screen,
108                bankDatabase, keypad, depositSlot);
109             break;
110       }
111       return temp;
112    }
113 }
```

Obrázek 39 - Třída ATM (5.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

3.2 Třída Screen

Třída Screen představuje obrazovku bankomatu a zahrnuje všechny aspekty zobrazování výstupu uživateli. Řádky 5–8 implementují metodu DisplayMessage, která vypíše hodnotu argumentu (string) na obrazovku. Další metoda (9–12) je velmi podobná té první, s tím rozdílem, že po vypsání zprávy se kurzor posune na další řádek. Poslední metoda (13–16) pracuje slouží k vypsání částky v dolarovém formátu.

```
1  using System;
2
3  public class Screen
4  {
5     public void DisplayMessage( string message )
6     {
7        Console.Write( message );
8     }
9     public void DisplayMessageLine( string message )
10    {
11       Console.WriteLine( message );
12    }
13    public void DisplayDollarAmount( decimal amount )
14    {
15       Console.Write( "{0:C}", amount );
16    }
17 }
```

Obrázek 40 – Třída Screen
Zdroj: vlastní zpracování dle (Deitel, 2010)

3.3 Třída Account

Třída Account reprezentuje bankovní účet. Z diagramu tříd vyplívá, že má každý účet 4 atributy, ty jsou implementovány na řádcích 3–6. Konstruktor (řádky 8–15) bere tyto 4 atributy jako argumenty. Řádky 11–14 přiřazují tyto hodnoty atributům třídy.

```
1  public class Account
2  {
3     private int accountNumber;
4     private int pin;
5     private decimal availableBalance;
6     private decimal totalBalance;
7
8     public Account( int theAccountNumber, int thePIN,
9        decimal theAvailableBalance, decimal theTotalBalance )
10    {
11       accountNumber = theAccountNumber;
12       pin = thePIN;
13       availableBalance = theAvailableBalance;
14       totalBalance = theTotalBalance;
15    }
```

Obrázek 41 - Třída Account (1.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Vlastnost AccountNumber (16–22) poskytuje přístup k proměnné accountNumber. Tuto vlastnost zařazujeme do naší implementace, aby klient třídy (např. BankDatabase) mohl identifikovat konkrétní účet. Například databáze BankDatabase obsahuje mnoho objektů Account a má přístup k této vlastnosti na každém ze svých objektů Account, aby nalezla ten s

konkrétním číslem účtu. Stejným způsobem jsou konstuovány vlastnosti AvaibleBalance (23–29) a TotalBalance (30–36).

```
16     public int AccountNumber
17     {
18        get
19        {
20           return accountNumber;
21        }
22     }
23     public decimal AvailableBalance
24     {
25        get
26        {
27           return availableBalance;
28        }
29     }
30     public decimal TotalBalance
31     {
32        get
33        {
34           return totalBalance;
35        }
36     }
```

Obrázek 42 - Třída Account (2.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Metoda ValidatePIN na řádcích 37–40 určuje, zda uživatelem zadaný PIN odpovídá PINu přidruženému k správnému účtu. Pokud PIN odpovídá, metoda vrací hodnotu true, v opačném případě vrací hodnotu false. Metoda Credit (41–44) přidává částku peněz (určenou parametrem amount) proměnné totalBalance. Poslední metoda Debit (45–49) odečítá peněžní částku jako součást transakce výběru. Na rozdíl od předchozí metody, transakce výběru odečítá částku od proměnné totalBalance i avaibleBalance.

```
37      public bool ValidatePIN( int userPIN )
38      {
39         return ( userPIN == pin );
40      }
41      public void Credit( decimal amount )
42      {
43         totalBalance += amount;
44      }
45      public void Debit( decimal amount )
46      {
47         availableBalance -= amount;
48         totalBalance -= amount;
49      }
50  }
```

Obrázek 43 - Třída Account (3.)
Zdroj: vlastní zpracování dle (Deitel, 2010)

Závěr

Cílem této bakalářské práce bylo navržení konkrétního informačního systému s využitím metod analýzy a návrhu informačních systémů, s využitím nástroje CASE a modelovacího jazyka UML.

Práce byla členěna na teoretickou, analytickou a návrhovou část. Teoretická část práce seznamuje čtenáře s problematikou informačních systémů, vymezuje pojmy a popisuje teorii vývoje informačních systémů. Analytická a návrhová část potom už pracují s konkrétním informačním systémem. Čtenář má tedy možnost projít si všechny kroky vývoje až po finální implementaci.

Výstupem práce je spustitelný program, který simuluje rozhraní bankomatu. Tento program je v současné fázi vhodný především pro testování a pro výukové účely.

Literatura

Application Analysis, 2012. In: *ScienceDirect* [online]. [cit. 2022-12-01]. Dostupné z: https://www.sciencedirect.com/topics/computer-science/application-analysis

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.

BOURGEOIS, David, 2019. *Information Systems for Business and Beyond* [online]. Kalifornie: Biola University [cit. 2023-02-02]. ISBN 9781304943484. Dostupné z: https://digitalcommons.biola.edu/cgi/viewcontent.cgi?article=1000&context=open-textbooks

BRUCKNER, Tomáš, 2012. *Tvorba informačních systémů: principy, metodiky, architektury*. 1. vyd. Praha: Grada. Management v informační společnosti. ISBN 978-80-247-4153-6.

DEITEL, Paul a Harvey DEITEL, 2010. *C# 2010 for Programmers*. 4. New Jersey: Prentice Hall. ISBN 9780132657396.

DENNIS, Alan, Barbara WIXOM a David TEGARDEN, 2015. *System Analysis and Design: An Object-Oriented Approach with UML*. Fifth Edition. New Jersey: Wiley. ISBN 978-1-118-80467-4.

Diagrams.net [online], 2022. Northampton: JGraph Ltd [cit. 2023-02-05]. Dostupné z: diagrams.net/blog

GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ, 2009. *Podniková informatika*. 2., přeprac. a aktualiz. vyd. Praha: Grada. Expert (Grada). ISBN 978-80-247-2615-1.

GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ, 2015. *Podniková informatika: počítačové aplikace v podnikové a mezipodnikové praxi*. 3., aktualizované vydání. Praha: Grada Publishing. Management v informační společnosti. ISBN 978-80-247-5457-4.

Investskills.com [online], 2021. London: CATWOE [cit. 2023-02-10]. Dostupné z: https://www.inveskills.com/bpmn/bpmn-vs-uml/

KANISOVÁ, Hana a Miroslav MÜLLER, 2006. *UML srozumitelně*. 2., aktualiz. vyd. Brno: Computer Press. ISBN 80-251-1083-4.

LEARN SDLC, 2017. In: *Tutorialspoint.com* [online]. [cit. 2022-11-20]. Dostupné z: https://www.tutorialspoint.com/sdlc/index.htm

LEARN UML, 2017. In: *Tutorialspoint.com* [online]. [cit. 2022-11-20]. Dostupné z: https://www.tutorialspoint.com/uml/uml_building_blocks.htm

OBJECT MANAGEMENT GROUP, 2013. *Business Process Model and Notation (BPMN)* [online]. [cit. 2023-02-04]. Dostupné z: doi:10.3403/30270926

OZKAYA, Mert, 2020. A survey on the practical use of UML for different software architecture viewpoints. *Information and software technology* [online]. **121**(1), 106275 [cit. 2020-05-03]. ISSN 0950-5849. Dostupné z: https://msfx.lib.cas.cz/sfxlcl3?ID=doi:10.1016%2Fj.infsof.2020.106275&genre=article&atitle=A%20survey%20on%20the%20practical%20use%20of%20UML%20for%20different%20software%20architecture%20viewpoints&title=Information%20and%20Software%20Technology&issn=09505849&isbn=&volume=121&issue=&date=20200501&aulast=Ozkaya,%20Mert&spage=&pages=&sid=EBSCO:ScienceDirect:S0950584920300252

POLÁKOVÁ, Danica, 2011. *Správa a modelovanie požiadaviek zákazníka*. Banská Bystrica. Bakalářská práce. Bankovní institut vysoká škola Praha zahraničná vysoká škola Banská Bystrica. Vedoucí práce Doc. RNDr. Juraj Pančík, CSc.

Rational Software Architect, 2016. In: *Ibm.com* [online]. [cit. 2022-12-30]. Dostupné z: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-composite-structure

RUMBAUGH, James, Ivar JACOBSON a Grady BOOCH, 2010. *The Unified Modeling Language Reference Manual*. Second Edition. Boston: Addison-Wesley. ISBN 0-321-24562-8.

SCHMULLER, Joseph, 2001. *Myslíme v jazyku UML: knihovna programátora*. 1.vyd. Praha: GRADA. Myslíme v .. ISBN 80-247-0029-8.

System Definition & Meaning. In: *Dictionary.cz* [online]. [cit. 2022-11-15]. Dostupné z: https://www.dictionary.com/browse/system

Tallyfy.com [online], 2018. St. Louis: Tallyfy, Inc [cit. 2023-02-09]. Dostupné z: https://tallyfy.com/uml-diagram/

UNHELKAR, Bhuvan, 2018. *Software engineering with UML*. Boca Raton: Taylor & Francis Group, LLC. ISBN 978-1-138-29743-2.

VON ROSING, Mark, Stephen WHITE, Fred CUMMINS a Henk DE MAN, 2015. Business Process Model and Notation—BPMN. *The Complete Business Process Handbook* [online]. Elsevier, 433-457 [cit. 2023-02-04]. ISBN 9780127999593. Dostupné z: doi:10.1016/B978-0-12-799959-3.00021-5

VOŘÍŠEK, Jiří, 2015. *Principy a modely řízení podnikové informatiky*. Vydání druhé. Praha: Oeconomica, nakladatelství VŠE. ISBN 978-80-245-2086-5.

WALKER, Alyssa, 2022. UML Diagrams: History, Types, Characteristics, Versions, Tools. In: *Guru99* [online]. online [cit. 2022-12-09]. Dostupné z: https://www.guru99.com/uml-diagrams.html#3

Seznam obrázků a tabulek

Přílohy

Příloha 1 – UML diagramy

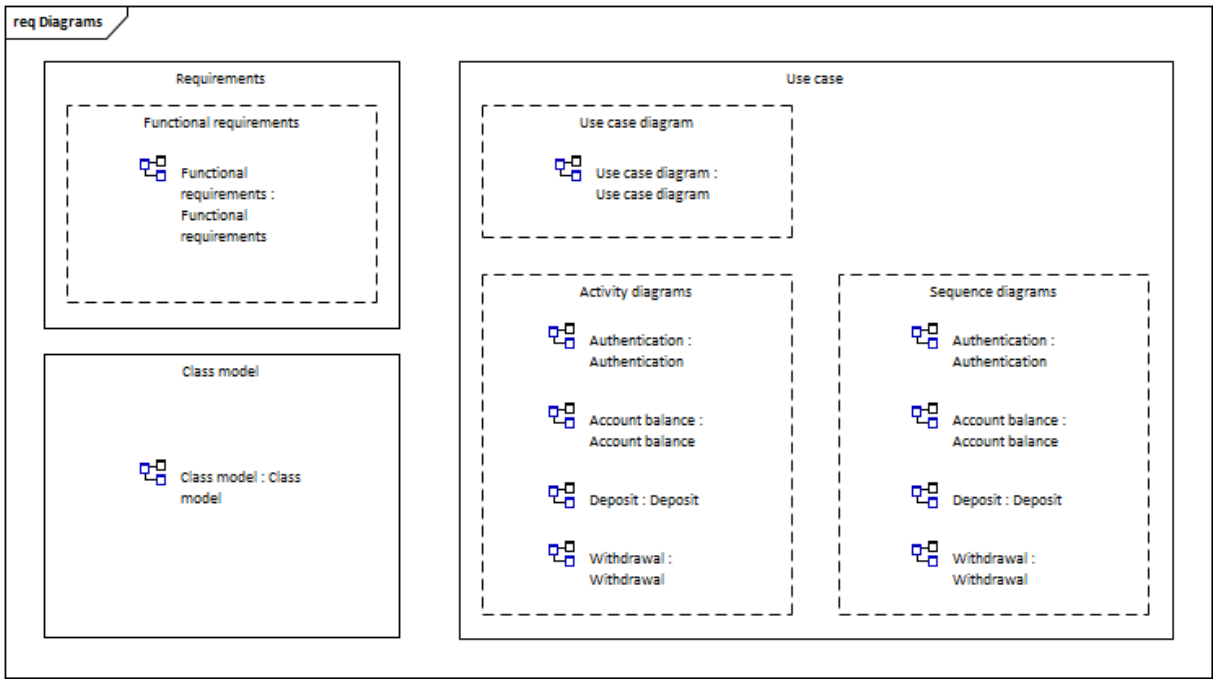
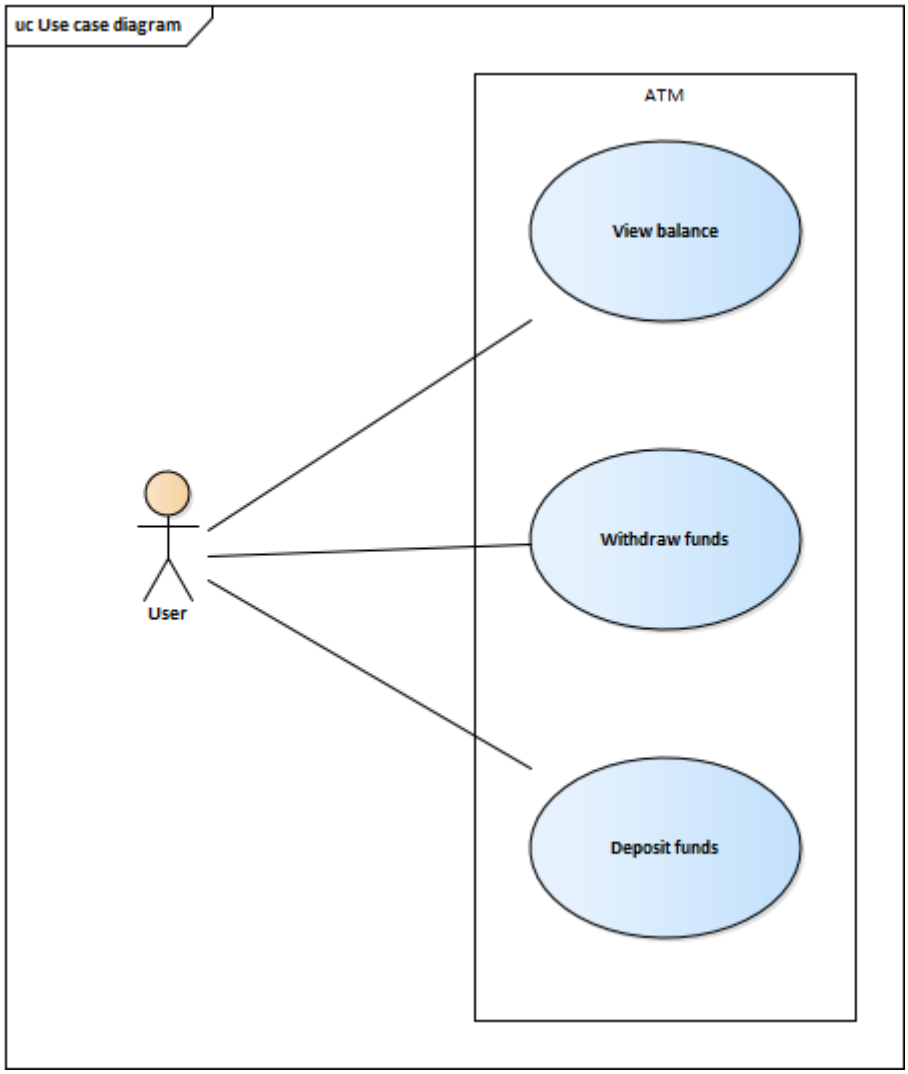
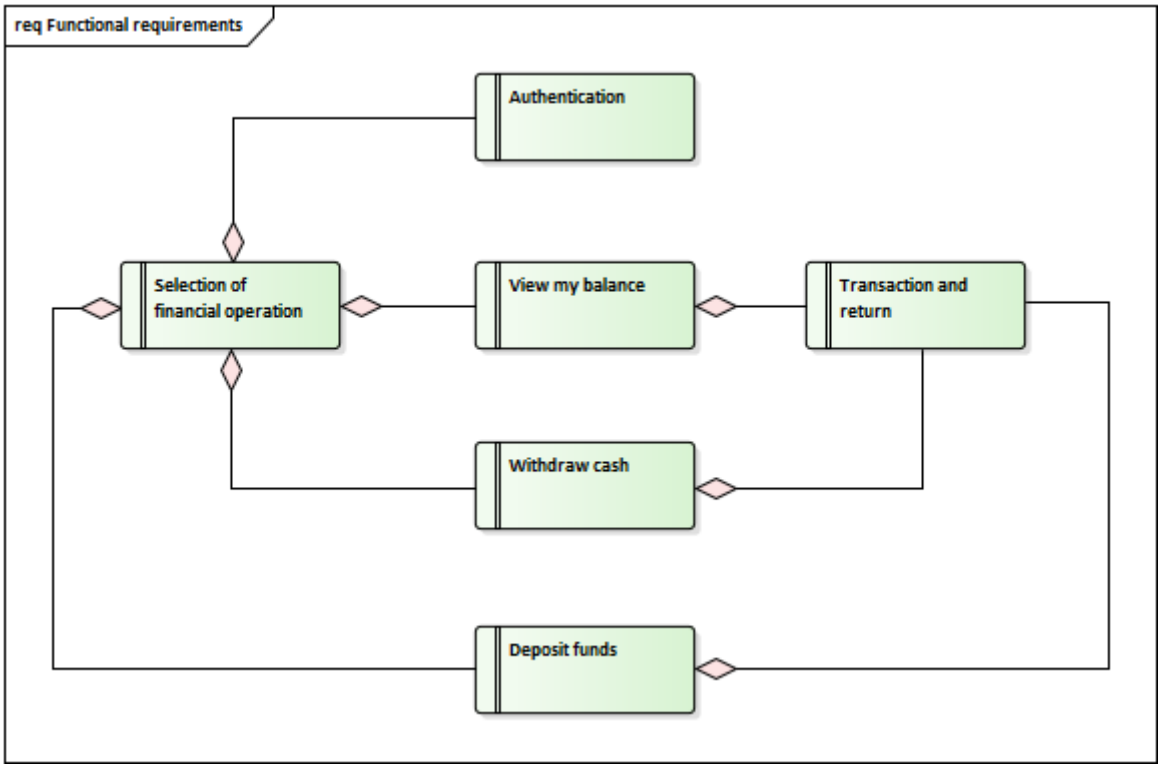


Diagram případu užití

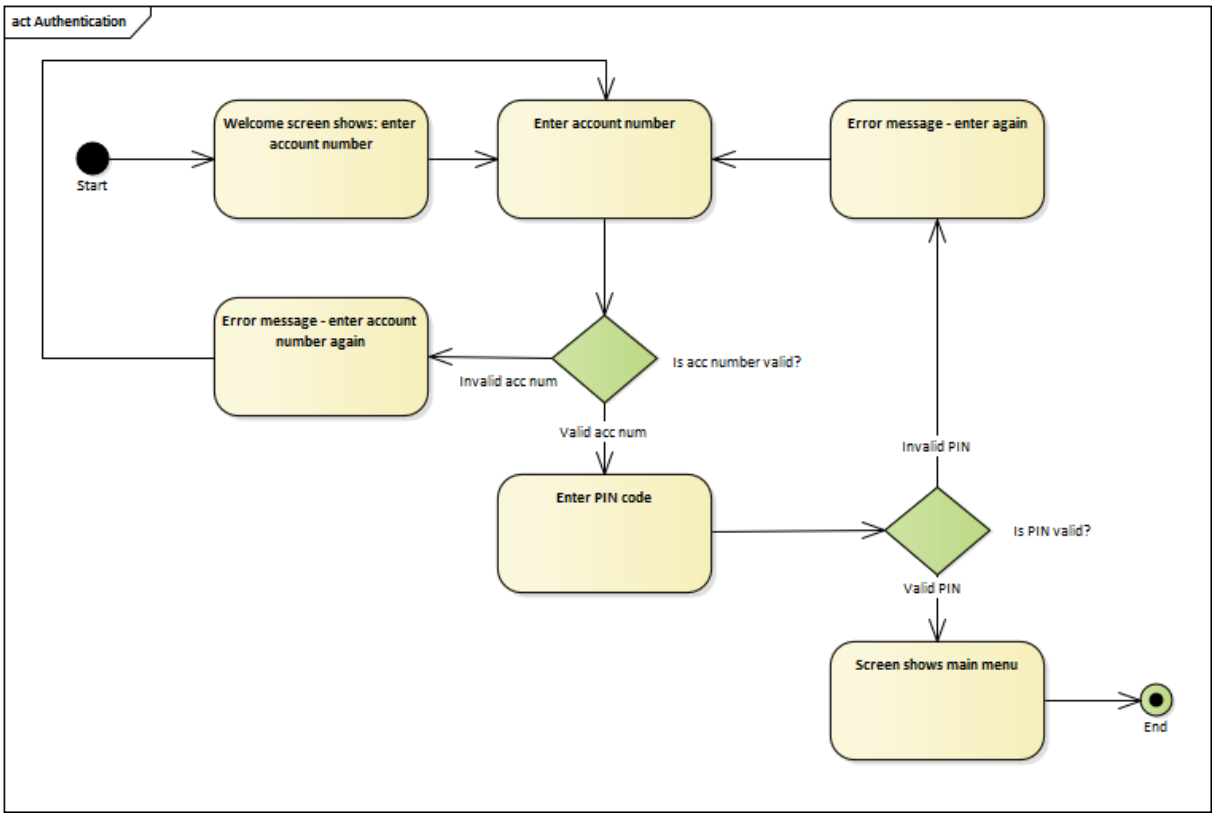


Funkční požadavky

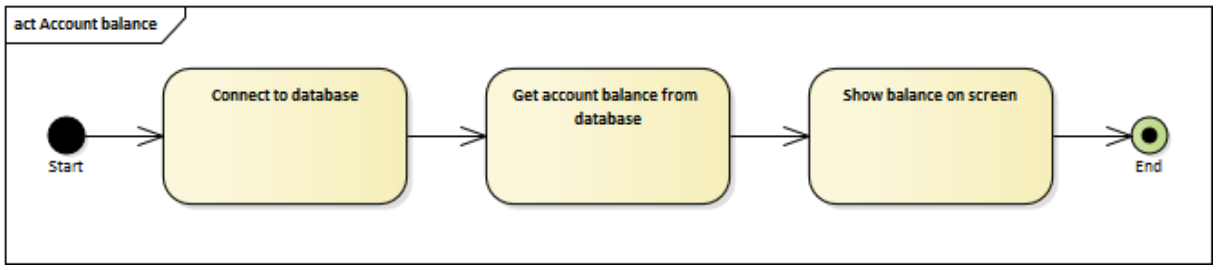


Diagramy aktivit

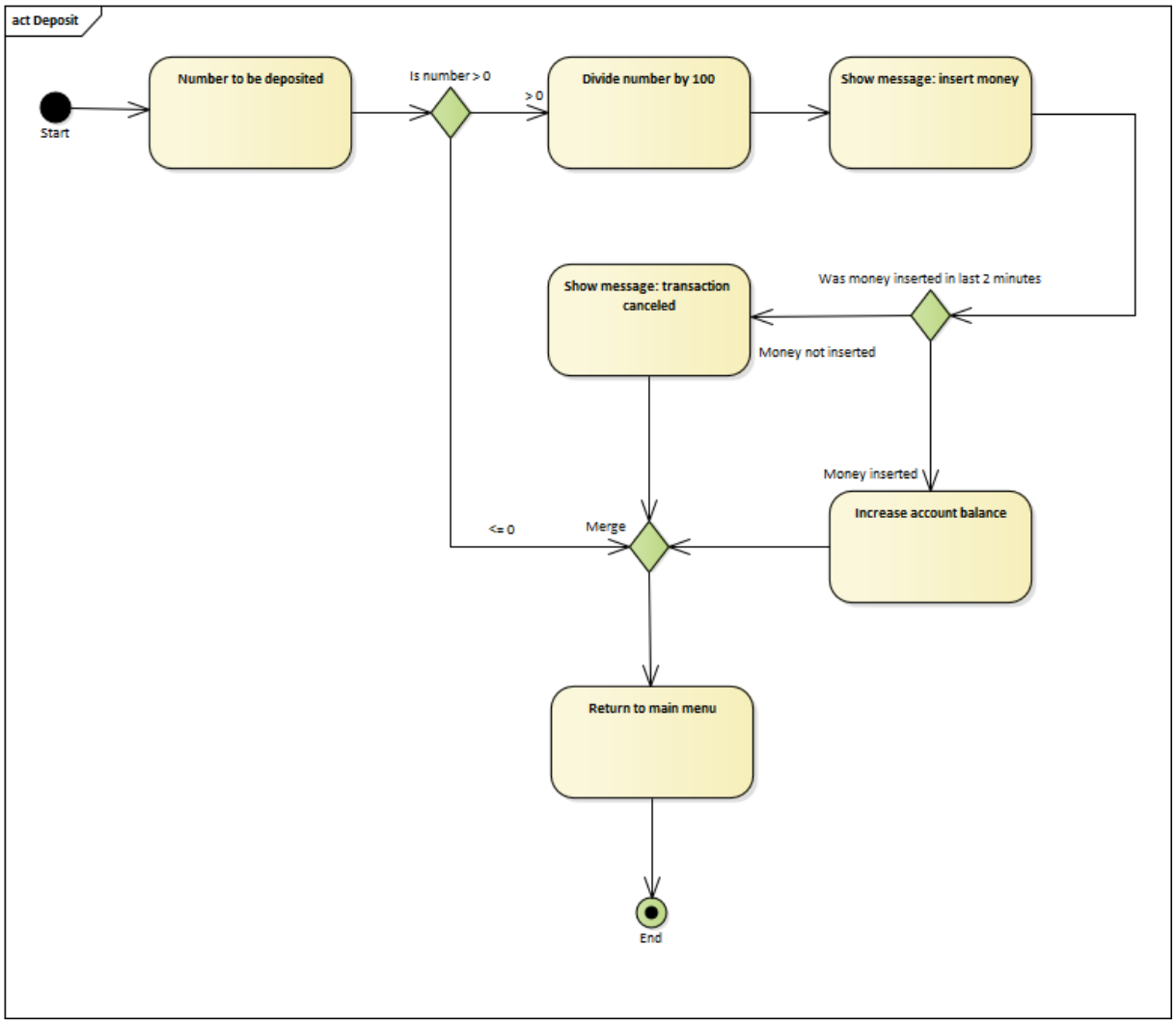
Authentication



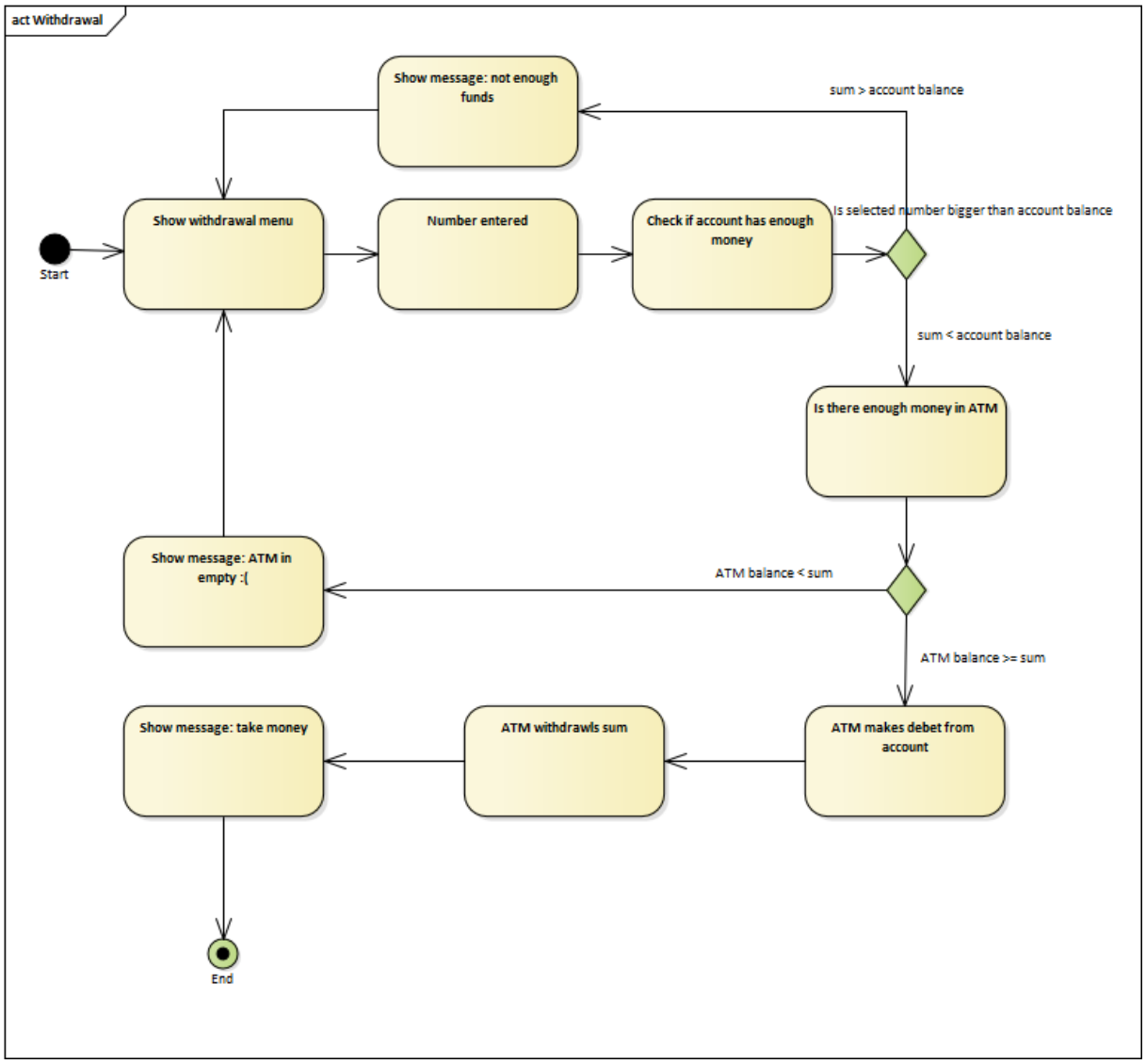
Account balance



Deposit

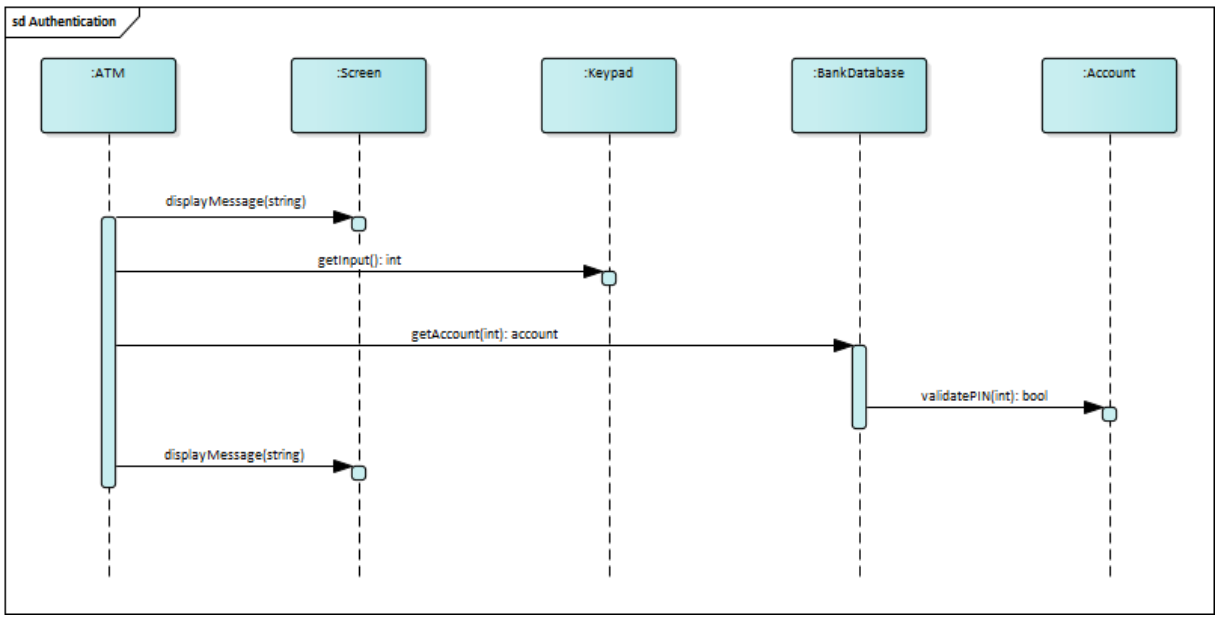


Withdrwal

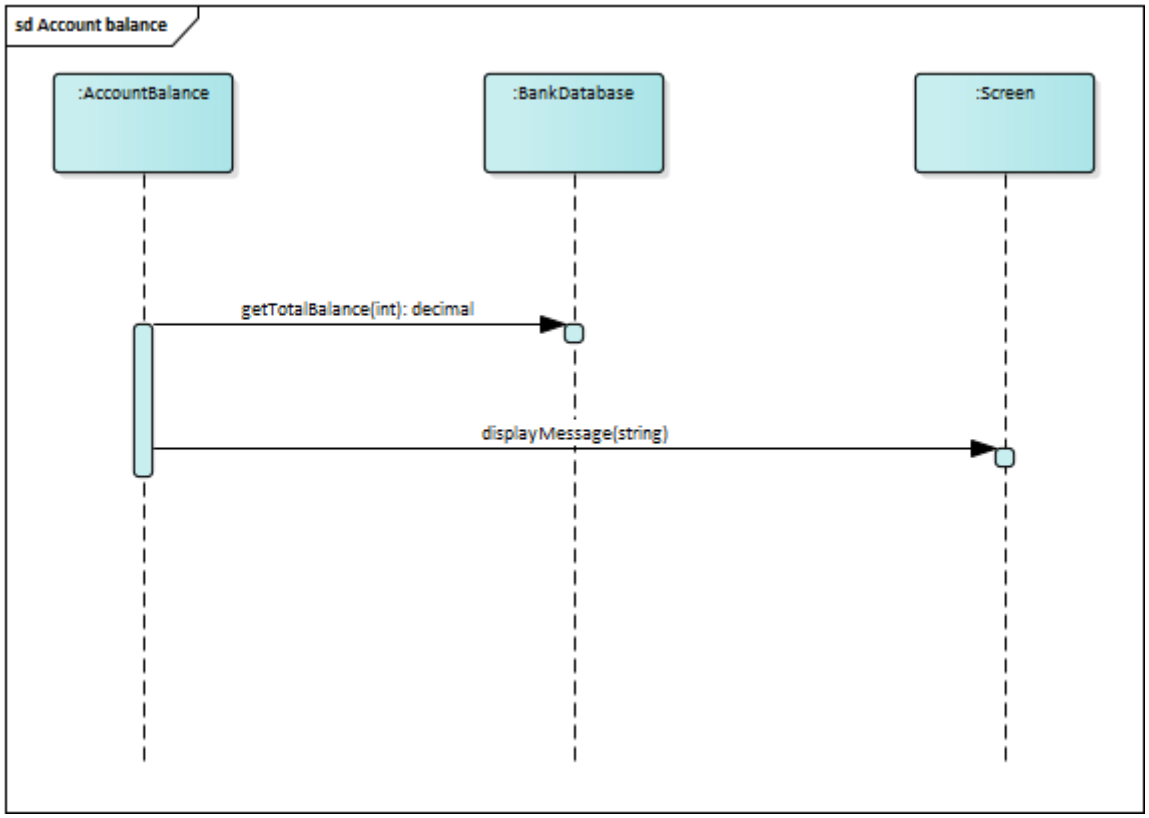

act Withdrawal
Show message: not enough funds
sum > account balance
Show withdrawal menu
Number entered
Check if account has enough money
Is selected number bigger than account balance
Start
sum < account balance
Is there enough money in ATM
Show message: ATM in empty :(
ATM balance < sum
ATM balance >= sum
Show message: take money
ATM withdrawls sum
ATM makes debet from account
End

Sekvenční diagramy

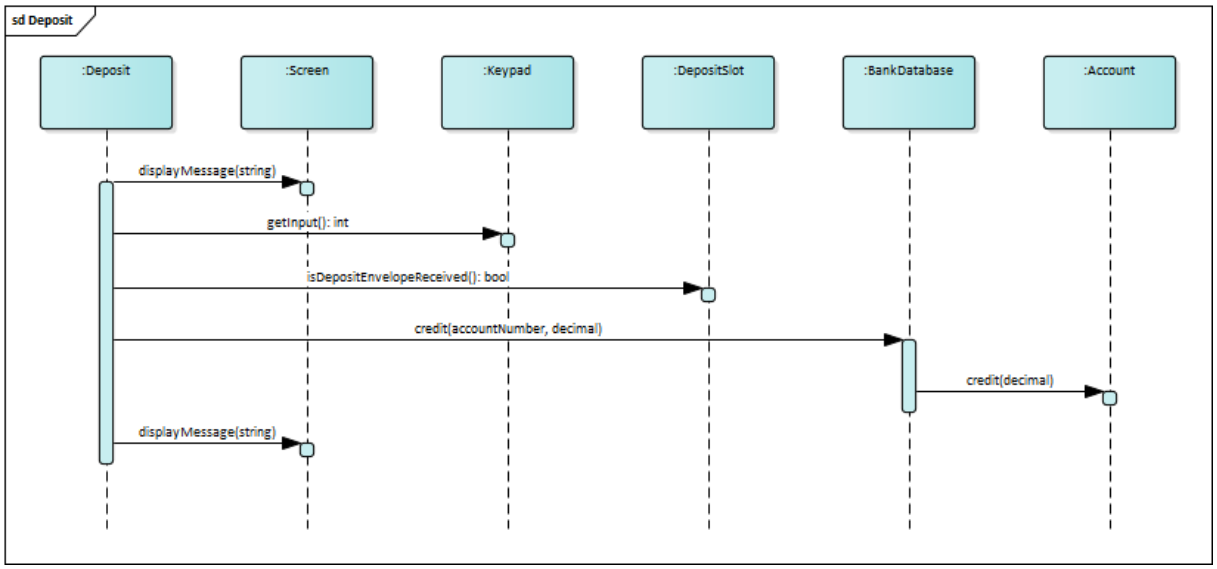
Authentication



Account balance



Deposit



Withdrawal

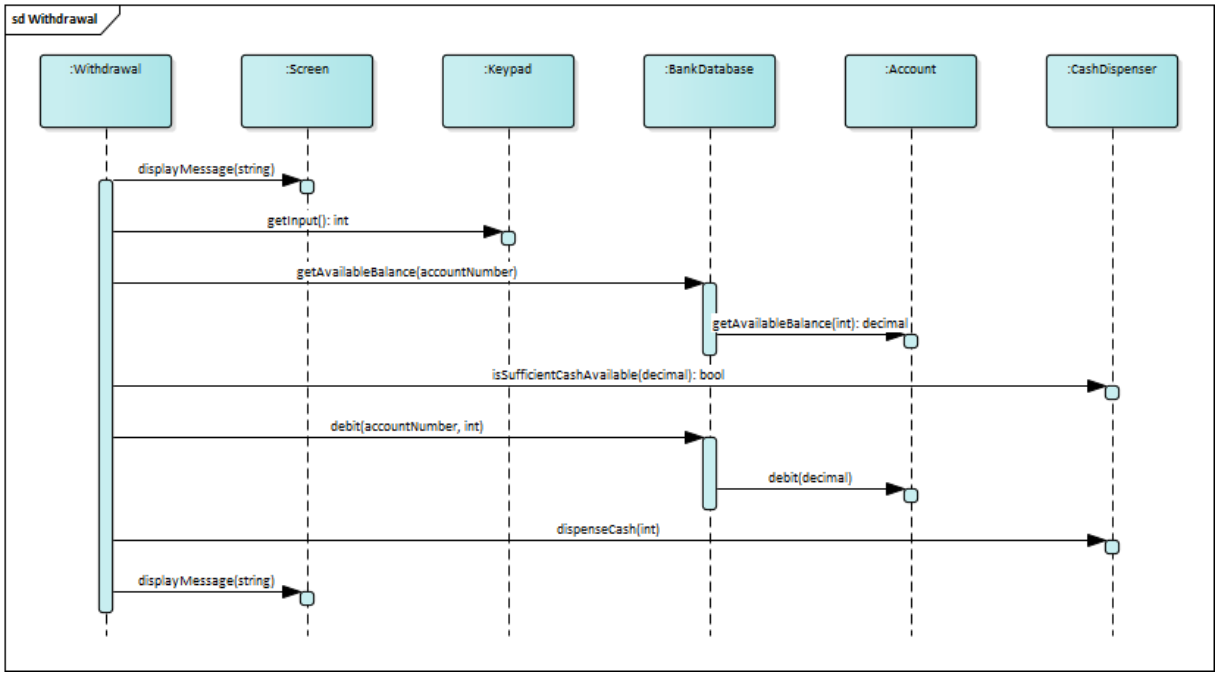
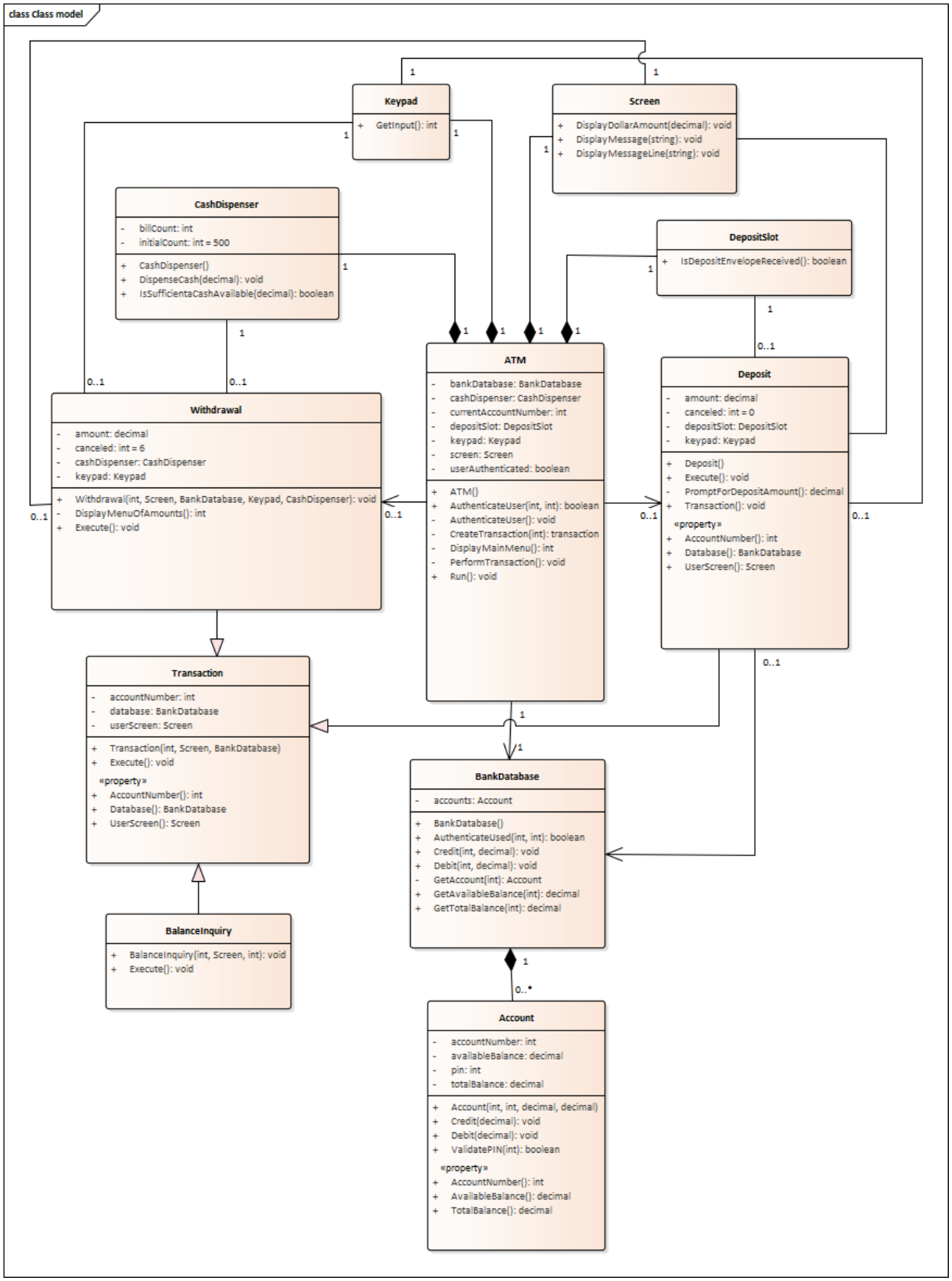


Diagram tříd



Příloha 2 – Implementace systému v C#

Třída ATM

```
1  public class ATM
2  {
3     private bool userAuthenticated;
4     private int currentAccountNumber;
5     private Screen screen;
6     private Keypad keypad;
7     private CashDispenser cashDispenser;
8     private DepositSlot depositSlot;
9     private BankDatabase bankDatabase;
10    private enum MenuOption
11    {
12       BALANCE_INQUIRY = 1,
13       WITHDRAWAL = 2,
14       DEPOSIT = 3,
15       EXIT_ATM = 4
16    }
17    public ATM()
18    {
19       userAuthenticated = false;
20       currentAccountNumber = 0;
21       screen = new Screen();
22       keypad = new Keypad();
23       cashDispenser = new CashDispenser();
24       depositSlot = new DepositSlot();
25       bankDatabase = new BankDatabase();
26    }
27    public void Run()
28    {
29       while ( true )
30       {
31          while ( !userAuthenticated )
32          {
33             screen.DisplayMessageLine( "\nVítejte!" );
34             AuthenticateUser();
35          }
36          PerformTransactions();
37          userAuthenticated = false;
38          currentAccountNumber = 0;
39          screen.DisplayMessageLine( "\nDěkujeme! Na shledanou!" );
40       }
41    }
42    private void AuthenticateUser()
43    {
44       screen.DisplayMessage( "\nZadejte prosím Vaše číslo účtu: " );
45       int accountNumber = keypad.GetInput();
46       screen.DisplayMessage( "\nZadejte Váš PIN: " );
47       int pin = keypad.GetInput();
48       userAuthenticated =
49          bankDatabase.AuthenticateUser( accountNumber, pin );
50       if  ( userAuthenticated )
51          currentAccountNumber = accountNumber;
52       else
53          screen.DisplayMessageLine(
```

```
54              "Chybně zadané číslo účtu nebo PIN, zkuste to znovu." );
55     }
56     private void PerformTransactions()
57     {
58        Transaction currentTransaction;
59        bool userExited = false;
60        while ( !userExited )
61        {
62           int mainMenuSelection = DisplayMainMenu();
63           switch ((MenuOption)mainMenuSelection)
64           {
65              case MenuOption.BALANCE_INQUIRY:
66              case MenuOption.WITHDRAWAL:
67              case MenuOption.DEPOSIT:
68                 currentTransaction =
69                    CreateTransaction( mainMenuSelection );
70                 currentTransaction.Execute();
71                 break;
72              case MenuOption.EXIT_ATM:
73                 screen.DisplayMessageLine("\nSystém se ukončuje...");
74                 userExited = true;
75                 break;
76              default:
77                 screen.DisplayMessageLine(
78                    "\nNevybrali jste platnou položku. Zkuste to
                      znovu." );
79                 break;
80           }
81        }
82     }
83     private int DisplayMainMenu()
84     {
85        screen.DisplayMessageLine( "\nHlavní menu:" );
86        screen.DisplayMessageLine( "1 - Zobrazit zůstatek" );
87        screen.DisplayMessageLine( "2 - Vybrat hotovost" );
88        screen.DisplayMessageLine( "3 - Vložit hotovost" );
89        screen.DisplayMessageLine( "4 - Odejít\n" );
90        screen.DisplayMessage( "Vyberte položku z menu: " );
91        return keypad.GetInput();
92     }
93     private Transaction CreateTransaction( int type )
94     {
95        Transaction temp = null;
96        switch ( ( MenuOption ) type )
97        {
98           case MenuOption.BALANCE_INQUIRY:
99              temp = new BalanceInquiry( currentAccountNumber,
100                screen, bankDatabase);
101             break;
102          case MenuOption.WITHDRAWAL:
103             temp = new Withdrawal( currentAccountNumber, screen,
104                bankDatabase, keypad, cashDispenser);
105             break;
```

```
106           case MenuOption.DEPOSIT:
107              temp = new Deposit( currentAccountNumber, screen,
108                 bankDatabase, keypad, depositSlot);
109              break;
110        }
111        return temp;
112     }
113  }
```

Třída Screen

```
 1  using System;
 2
 3  public class Screen
 4  {
 5     public void DisplayMessage( string message )
 6     {
 7        Console.Write( message );
 8     }
 9     public void DisplayMessageLine( string message )
10     {
11        Console.WriteLine( message );
12     }
13     public void DisplayDollarAmount( decimal amount )
14     {
15        Console.Write( "{0:C}", amount );
16     }
17  }
```

Třída Keypad

```
1  using System;
2
3  public class Keypad
4  {
5     public int GetInput()
6     {
7        return Convert.ToInt32( Console.ReadLine() );
8     }
9  }
```

Třída CashDispenser

```
1  public class CashDispenser
2  {
3     private const int INITIAL_COUNT = 500;
4     private int billCount;
5     public CashDispenser()
6     {
7        billCount = INITIAL_COUNT;
8     }
9     public void DispenseCash( decimal amount )
10    {
11       int billsRequired = ( ( int ) amount ) / 20;
12       billCount -= billsRequired;
13    }
14    public bool IsSufficientCashAvailable( decimal amount )
15    {
16       int billsRequired = ( ( int ) amount ) / 20;
17       return ( billCount >= billsRequired );
18    }
19 }
```

Třída DepositSlot

```
1  public class DepositSlot
2  {
3     public bool IsDepositEnvelopeReceived()
4     {
5        return true;
6     }
7  }
```

Třída Account

```
1  public class Account
2  {
3     private int accountNumber;
4     private int pin;
5     private decimal availableBalance;
6     private decimal totalBalance;
7
8     public Account( int theAccountNumber, int thePIN,
9        decimal theAvailableBalance, decimal theTotalBalance )
10    {
11       accountNumber = theAccountNumber;
12       pin = thePIN;
13       availableBalance = theAvailableBalance;
14       totalBalance = theTotalBalance;
15    }
16    public int AccountNumber
17    {
18       get
19       {
20          return accountNumber;
21       }
22    }
23    public decimal AvailableBalance
24    {
25       get
26       {
27          return availableBalance;
28       }
29    }
30    public decimal TotalBalance
31    {
32       get
33       {
34          return totalBalance;
35       }
36    }
37    public bool ValidatePIN( int userPIN )
38    {
39       return ( userPIN == pin );
40    }
41    public void Credit( decimal amount )
42    {
43       totalBalance += amount;
44    }
45    public void Debit( decimal amount )
46    {
47       availableBalance -= amount;
48       totalBalance -= amount;
49    }
50 }
```

Třída BankDatabase

```
1  public class BankDatabase
2  {
3     private Account[] accounts;
4     public BankDatabase()
5     {
6        accounts = new Account[ 3 ];
7        accounts[ 0 ] = new Account( 123456, 1234, 100.00M, 120.00M );
8        accounts[ 1 ] = new Account( 000000, 0000, 200.00M, 200.00M );
9        accounts[ 2 ] = new Account( 1, 1, 2000.00M, 3000.00M );
10    }
11    private Account GetAccount( int accountNumber )
12    {
13       foreach ( Account currentAccount in accounts )
14       {
15          if ( currentAccount.AccountNumber == accountNumber )
16             return currentAccount;
17       }
18       return null;
19    }
20    public bool AuthenticateUser( int userAccountNumber, int userPIN)
21    {
22       Account userAccount = GetAccount( userAccountNumber );
23       if ( userAccount != null )
24          return userAccount.ValidatePIN( userPIN );
25       else
26          return false;
27    }
28    public decimal GetAvailableBalance( int userAccountNumber )
29    {
30       Account userAccount = GetAccount( userAccountNumber );
31       return userAccount.AvailableBalance;
32    }
33    public decimal GetTotalBalance( int userAccountNumber )
34    {
35       Account userAccount = GetAccount(userAccountNumber);
36       return userAccount.TotalBalance;
37    }
38    public void Credit( int userAccountNumber, decimal amount )
39    {
40       Account userAccount = GetAccount( userAccountNumber );
41       userAccount.Credit( amount );
42    }
43    public void Debit( int userAccountNumber, decimal amount )
44    {
45       Account userAccount = GetAccount( userAccountNumber );
46       userAccount.Debit( amount );
47    }
48 }
```

Třída Transaction

```
1  public abstract class Transaction
2  {
3     private int accountNumber;
4     private Screen userScreen;
5     private BankDatabase database;
6     public Transaction( int userAccount, Screen theScreen,
7        BankDatabase theDatabase )
8     {
9        accountNumber = userAccount;
10       userScreen = theScreen;
11       database = theDatabase;
12    }
13    public int AccountNumber
14    {
15       get
16       {
17          return accountNumber;
18       }
19    }
20    public Screen UserScreen
21    {
22       get
23       {
24          return userScreen;
25       }
26    }
27    public BankDatabase Database
28    {
29       get
30       {
31          return database;
32       }
33    }
34    public abstract void Execute();
35 }
```

Třída BalanceInquiry

```
1  public class BalanceInquiry : Transaction
2  {
3     public BalanceInquiry( int userAccountNumber,
4        Screen atmScreen, BankDatabase atmBankDatabase )
5        : base( userAccountNumber, atmScreen, atmBankDatabase ) {}
6     public override void Execute()
7     {
8        decimal availableBalance =
9           Database.GetAvailableBalance( AccountNumber );
10       decimal totalBalance = Database.GetTotalBalance( AccountNumber );
11       UserScreen.DisplayMessageLine( "\nInformace o zůstatku:" );
12       UserScreen.DisplayMessage( " - Dostupný zůstatek: " );
13       UserScreen.DisplayDollarAmount( availableBalance );
14       UserScreen.DisplayMessage( "\n - Účetní zůstatek: " );
15       UserScreen.DisplayDollarAmount( totalBalance );
16       UserScreen.DisplayMessageLine( "" );
17    }
18 }
```

Třída Withdrawal

```
1  public class Withdrawal : Transaction
2  {
3     private decimal amount;
4     private Keypad keypad;
5     private CashDispenser cashDispenser;
6
7     private const int CANCELED = 6;
8
9     public Withdrawal( int userAccountNumber, Screen atmScreen,
10       BankDatabase atmBankDatabase, Keypad atmKeypad,
11       CashDispenser atmCashDispenser )
12       : base( userAccountNumber, atmScreen, atmBankDatabase )
13    {
14       keypad = atmKeypad;
15       cashDispenser = atmCashDispenser;
16    }
17    public override void Execute()
18    {
19       bool cashDispensed = false;
20       bool transactionCanceled = false;
21       do
22       {
23          int selection = DisplayMenuOfAmounts();
24          if ( selection != CANCELED )
25          {
26             amount = selection;
27             decimal availableBalance =
28                Database.GetAvailableBalance( AccountNumber );
29             if ( amount <= availableBalance )
30             {
31                if ( cashDispenser.IsSufficientCashAvailable( amount ) )
32                {
33                   Database.Debit( AccountNumber, amount );
34                   cashDispenser.DispenseCash( amount );
35                   cashDispensed = true;
36                   UserScreen.DisplayMessageLine(
37                      "\nVezměte si prosím svoji hotovost." );
38                }
39                else
40                   UserScreen.DisplayMessageLine(
41                      "\nNedostatek hotovosti v bankomatu." +
42                      "\n\nZvolte prosím nižší částku." );
43             }
44             else
45                UserScreen.DisplayMessageLine(
46                   "\nNa vašem účtu není dostatek hotovosti." +
47                   "\n\nZvolte prosím nižší částku.");
48          }
49          else
50          {
51             UserScreen.DisplayMessageLine( "\nZrušení transakce..." );
52             transactionCanceled = true;
53          }
```

```
54          } while ( ( !cashDispensed ) && ( !transactionCanceled ) );
55       }
56       private int DisplayMenuOfAmounts()
57       {
58          int userChoice = 0;
59          int[] amounts = { 0, 20, 40, 60, 100, 200 };
60          while ( userChoice == 0 )
61          {
62             UserScreen.DisplayMessageLine( "\nMožnosti výběru:" );
63             UserScreen.DisplayMessageLine( "1 - $20" );
64             UserScreen.DisplayMessageLine( "2 - $40" );
65             UserScreen.DisplayMessageLine( "3 - $60" );
66             UserScreen.DisplayMessageLine( "4 - $100" );
67             UserScreen.DisplayMessageLine( "5 - $200" );
68             UserScreen.DisplayMessageLine( "6 - Zrušit transakci" );
69             UserScreen.DisplayMessage(
70                "\nVyberte možnost (1-6): " );
71             int input = keypad.GetInput();
72             switch ( input )
73             {
74                case 1: case 2: case 3: case 4: case 5:
75                   userChoice = amounts[ input ];
76                   break;
77                case CANCELED:
78                   userChoice = CANCELED;
79                   break;
80                default:
81                   UserScreen.DisplayMessageLine(
82                      "\nNeplatný výběr. Zkuste to znovu." );
83                   break;
84             }
85          }
86          return userChoice;
87       }
88    }
```

Třída Deposit

```
1  public class Deposit : Transaction
2  {
3     private decimal amount;
4     private Keypad keypad;
5     private DepositSlot depositSlot;
6
7     private const int CANCELED = 0;
8
9     public Deposit( int userAccountNumber, Screen atmScreen,
10       BankDatabase atmBankDatabase, Keypad atmKeypad,
11       DepositSlot atmDepositSlot )
12       : base( userAccountNumber, atmScreen, atmBankDatabase )
13    {
14       keypad = atmKeypad;
15       depositSlot = atmDepositSlot;
16    }
17    public override void Execute()
18    {
19       amount = PromptForDepositAmount();
20       if ( amount != CANCELED )
21       {
22          UserScreen.DisplayMessage(
23             "\nVložte prosím vkladovou obálku obsahující " );
24          UserScreen.DisplayDollarAmount( amount );
25          bool envelopeReceived = depositSlot.IsDepositEnvelopeReceived
              ();
26          if ( envelopeReceived )
27          {
28             UserScreen.DisplayMessageLine(
29                "\nVaše obálka byla přijata.\n" +
30                "Peníze nebudou k dispozici dokud " +
31                "neověříme \nvloženou hotovost, " +
32                "případně vložené šeky." );
33             Database.Credit( AccountNumber, amount );
34          }
35          else
36             UserScreen.DisplayMessageLine(
37                "\nNevložil/a jste obálku, bankomat " +
38                "zrušil Vaši transakci" );
39       }
40       else
41          UserScreen.DisplayMessageLine( "\nZrušení transakce..." );
42    }
43    private decimal PromptForDepositAmount()
44    {
45       UserScreen.DisplayMessage(
46          "\nZadajte prosím částku (v formátu včetně haléřů), kterou
             chcete vložit:\n" +
47          "Pro zrušení stiskněte 0: ");
48       int input = keypad.GetInput();
49       if ( input == CANCELED )
50          return CANCELED;
51       else
```

```
52            return input / 100.00M;
53        }
54    }
```

Třída ATMCaseStudy

```
1  public class ATMCaseStudy
2  {
3     public static void Main( string[] args )
4     {
5        ATM theATM = new ATM();
6        theATM.Run();
7     }
8  }
```

Příloha 3 – seznam požadavků

Seznam nefunkčních požadavků

ID	název	popis požadavku
NFR01	Bank database	Při provádění transakce se bankomat připojí do vzdálené databáze, kde pomocí vloženého čísla účtu uživatelem získá informace o zůstatku. Bankomat tedy nedrží informace o účtech lokálně.
NFR02	Keypad	Na klávesnici bankomatu jsou číslice 0 - 9, tlačítko ENTER a tlačítko BACKSPACE.
NFR03	Deposit slot	Otvor pro vkládání hotovosti má rozměry standardní poštovní obálky. Bankovky tedy nejsou vložit samostatně, pouze pokud jsou vložené uvnitř obálky.

Seznam funkčních požadavků

ID	název	popis požadavku
FR01	Authentication	Pokud uživatel vloží správné číslo účtu a odpovídající PIN, bankomat uživatele ověří a pustí do hlavního menu. V případě, že uživatel zadá špatné číslo účtu, či PIN, zobrazí se informativní text, který vyzve uživatele k opětovnému zadání údajů.
FR02	Deposit funds	Vklad na účet: 1. Na obrazovce se zobrazí výzva k zadání částky pro vklad na klávesnici. Pro zrušení transakce uživatel zadá nulu. 2. Uživatel zadá částku v dolarech včetně centů, bez desetinné čárky, která na klávesnici není. 3. Na obrazovce se zobrazí výzva k vložení obálky s penězi.

		4. Pokud bankomat obdrží obálku během dvou minut, připíše se částka na uživatelův účet. Vyprší-li časový limit, na obrazovce se zobrazí zpráva o zrušení transakce a zobrazí se hlavní menu.
FR03	Selection of financial operation	Výběr operace: Na hlavní obrazovce je číslovaný seznam operací, ze kterých může uživatel vybírat stisknutím příslušného čísla na klávesnici. 1. View my balance - zobrazit zůstatek. 2. Withdraw cash - vybrat hotovost. 3. Deposit funds - vložit hotovost. 4. Exit
FR04	Transaction and return	Po úspěšném vykonání transakce, nebo ukončení zobrazí systém hlavní menu, případně uživatele odhlásí.
FR05	View my balance	Zobrazení zůstatku na účtě po stisknutí klávesy 1. Zůstatek se uživateli zobrazí na obrazovce, ten získá přístupem do databáze.
FR06	Withdraw cash	Výběr hotovosti: 1. Na obrazovce se zobrazí menu, které obsahuje časté sumy k výběru. Ty jsou: 20$, 40$, 60$, 100$, 200$. Poslední položkou je zrušení transakce. 2. Uživatel vybere částku, nebo zrušení pomocí klávesnice. 3. Pokud je vybraná částka vyšší než zůstatek na účtu, zobrazí se zpráva o nedostatečném zůstatku a návrat k bodu 1. Pokud je částka nižší, nebo rovna zůstatku, proces pokračuje. 4. Bankomat zjistí, jestli je v zásobníku dostatečný počet bankovek. Pokud ne, zobrazí se zpráva o nedostatečném počtu bankovek a návrat k bodu 1. Pokud ano proces pokračuje. 5. Bankomat sníží zůstatek na účtu uživatele o vybranou částku. 6. Bankomat vysune bankovky. 7. Na obrazovce se zobrazí text s upozorněním pro vybráním hotovosti.