

Sem vložte zadání Vaší práce.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
KATEDRA SOFTWAREVÉHO INŽENÝRSTVÍ



Bakalářská práce

Desktopový nástroj pro správu skupin lidí

Jakub Štercl

Vedoucí práce: Ing. Jan Baier

4. května 2017

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Janu Baierovi za jeho cenné rady, poznatky a připomínky.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2017

.....

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2017 Jakub Štercl. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Štercl, Jakub. *Desktopový nástroj pro správu skupin lidí*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá vývojem desktopové aplikace pro správu skupin lidí a jejich rozdělování do týmů. Součástí práce je celý cyklus vývoje, od analýzy požadavků, přes návrh, implementaci až po testování, případně nasazení.

Klíčová slova Desktopová aplikace, Python, Qt, správa skupin

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords

Obsah

Úvod	1
1 Cíl práce	3
1.1 Existující řešení	3
1.2 Analýza požadavků	3
2 Návrh řešení	9
2.1 Volba technologií	9
2.2 Doménový model	11
2.3 Proces rozdělení skupiny	12
3 Implementace	15
Závěr	17
Literatura	19
A Seznam použitých zkratk	21
B Obsah přiloženého CD	23

Seznam obrázků

1.1	Diagram případů užití	6
2.1	Doménový model	12
2.2	Diagram aktivit znázorňující proces vytvoření rozdělení	13

Úvod

Moderní technologie a výpočetní technika nám čím dál více usnadňují život. Mnohdy dokáží stroje rozhodovat rychleji, lépe nebo třeba spravedlivěji než člověk. Toto platí hlavně pro řešení složitějších úkolů. Jedním z takových úkolů, před které jsou mnohdy postaveni učitelé, organizátoři různých akcí nebo projektoví vedoucí, je rozdělení lidí (žáků, účastníků, apod.) do týmů na základě určitých požadavků. Jelikož neexistuje uspokojivý nástroj, který by toto umožňoval, rozhodl jsem se jej v rámci své bakalářské práce vytvořit.

Cíl práce

Cílem práce je tedy na základě analýzy požadavků navrhnout a posléze naimplementovat aplikaci, která bude organizátorům soustředění korespondenčního semináře FIKS (Fitácký informatický korespondenční seminář)¹ sloužit při tvorbě týmů pro jednotlivé hry. Hlavní motivací, proč organizátoři takovouto aplikaci chtějí je, aby se účastníci mezi sebou poznali, tj. každou hru hráli, pokud možno, s novými spoluhráči.

1.1 Existující řešení

Vzhledem k tomu, že s podobným problémem se organizátoři po celém světě samozřejmě setkávají poměrně často, existují různé nástroje, které tvorbu skupin umožňují. Většinou se ale jedná buď o webové nebo mobilní aplikace. Jednou z takových webových aplikací je například Team Maker², podobná mobilní aplikace je Teamistr³. Tyto aplikace ale neumožňují vytvoření a uložení seznamů lidí (např. paralelky), tudíž neukládají ani historii vytvořených týmů, neumožňují zobrazení, s kým už daná osoba ve skupině byla a neumožňují ani zvolit, zda preferujeme rozdělení, kde lidé v jednotlivých týmech spolu již spolupracovali, nebo naopak. Proto nejsou zdaleka dostačující a jejich jediný účel je rozdělovat skupiny do týmů „spravedlivě“.

1.2 Analýza požadavků

Prvním krokem jakéhokoli úspěšného softwarového projektu je pochopit, co od výsledného systému zákazník (zadavatel) chce, co od něj požaduje. K tomu slouží tzv. analýza požadavků (requirements analysis).

¹<https://fiks.fit.cvut.cz/>

²<http://chir.ag/projects/team-maker/>

³<https://itunes.apple.com/us/app/teamistr/id609410714?mt=8> pro iOS
<https://play.google.com/store/apps/details?id=com.brownapps.sortr> pro Android

Nejdůležitější částí analýzy požadavků je samotný sběr požadavků, který probíhá spoluprací se zástupci zadavatele. V ideálním případě se do diskuse zapojují zástupci všech skupin budoucích uživatelů. Například není vhodné, aby u systému, který budou využívat žáci i učitelé, proběhla diskuse o požadavcích pouze se zástupci učitelů. Mohlo by se totiž stát, že ačkoli výsledný systém splňuje všechny vytyčené požadavky, nebude systém pro žáky použitelný.

1.2.1 Funkční požadavky

Požadavky sebrané v analýze (příp. sběru) požadavků se nejčastěji dělí na funkční a nefunkční požadavky. Samozřejmě existuje mnoho jiných způsobů jejich kategorizace, ale pro tento projekt je toto rozdělení naprosto dostačující. Funkčním požadavkem je popis požadované funkce systému – popis toho, co by měl systém umět.[1]

Na základě diskuse se zadavatelem jsem dospěl k následujícím funkčním požadavkům:

- **Správa skupin** – Aplikace bude umožňovat vytvořit, přejmenovat a upravovat skupiny lidí. Úpravou skupiny se myslí možnost přidávat členy, případně členy odebírat a přejmenovávat.
- **Rozdělení skupiny do týmů** – Aplikace bude umět navrhnout rozdělení skupiny do týmů podle zadaných parametrů. Těmito parametry může být minimální/maximální počet lidí v týmu, počet týmů v rozdělení a vynucení/zakázání určitých dvojic (skupin) lidí ve stejném týmu. Dále bude aplikace umožňovat zvolit, zda preferujeme takové rozdělení do týmů, ve kterém lidé, kteří spolu již v týmu spolupracovali, budou opět ve stejném týmu, či naopak. V případě, že požadované rozdělení neexistuje, aplikace navrhne takové rozdělení, které porušuje co nejméně požadavků.
- **Vizualizace rozdělení** – Aplikace bude umět vytvořené rozdělení vizualizovat pro potřeby tisku.
- **Export rozdělení** – Aplikace bude umět vytvořené rozdělení exportovat do textového souboru. Soubor se bude skládat jednotlivých týmů, každý na samostatné řádce. Každá řádka bude začínat jménem týmu, následovaným jmény členů týmu, oddělenými čárkou.
- **Zobrazení historie rozdělení** – Aplikace bude umět pro každého člena skupiny umět zobrazit s kterými členy skupiny již byl v týmu a kolikrát. Zároveň bude aplikace umět pro každou skupinu zobrazit uložená rozdělení.

1.2.2 Nefunkční požadavky

Jako nefunkční požadavky označujeme všechny požadavky, které přímo neříkají, co má systém umět. Nefunkční požadavek je v podstatě omezující nebo upřesňující podmínka uvalená na daný systém.[1]

Pro tento projekt jsem identifikoval tyto nefunkční požadavky:

- **OS Linux** – Aplikace musí fungovat na operačním systému Linux.
- **Velikost skupiny** – Aplikace bude využívána pro skupiny o velikosti přibližně 24 členů.

O validitě posledního nefunkčního požadavku (*Velikost skupiny*) by se dalo polemizovat, jelikož se vlastně nejedná o požadavek, nýbrž o informaci o tom, jak bude aplikace používána. Přesto jsem se rozhodl začlenit tuto cennou informaci přímo do požadavků, jelikož takováto informace bude hrát roli při návrhu uživatelského rozhraní, testování a podobně.

1.2.3 Případy užití

Případy užití přesně zachycují funkčnost, která bude budoucím systémem pokryta a vymezují tak jednoznačně rozsah prací. Každý případ užití popisuje jeden ze způsobů použití systému, popisuje tedy jednu jeho požadovanou funkčnost.[?]

Případ užití je vlastně seznam kroků, který definuje jednu konkrétní interakci uživatele se systémem. Ačkoli struktura případu užití (popis, jak by vlastně takový případ užití měl vypadat) není standardizována, tak UML nabízí tzv. diagram užití (use case diagram). Jedná se vlastně o přiřazení jednotlivých činností k aktérům a případné znázornění souvislostí jednotlivých případů užití (relace «*extend*» a «*include*»). Samotný scénář případu užití (posloupnost jednotlivých kroků potřebných pro vykonání případu užití) už není součástí diagramu.

Pro tento projekt jsem dospěl k následujícím případům užití znázorněným v diagramu 1.1. Dále jsou uvedeny scénáře případů užití.

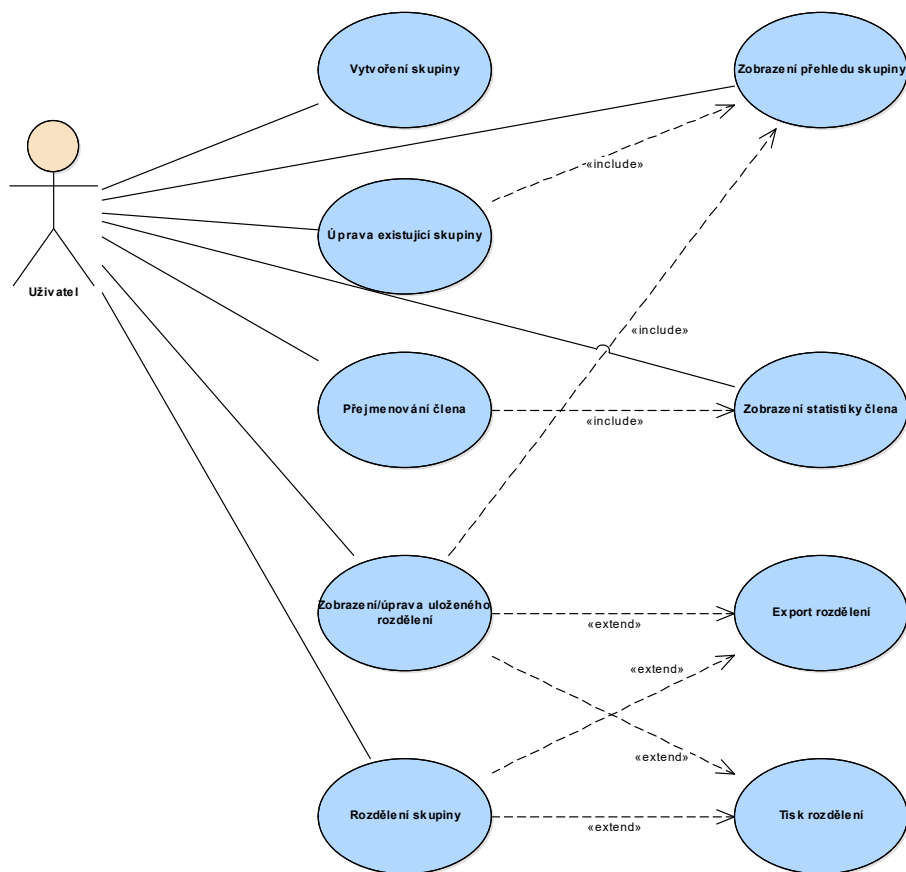
- **Vytvoření skupiny**

1. Uživatel stiskne tlačítko *Vytvořit skupinu*.
2. Aplikace zobrazí obrazovku s přehledem nově vytvořené skupiny.

- **Zobrazení přehledu skupiny**

1. Uživatel vybere ze seznamu skupinu.
2. Aplikace zobrazí obrazovku s přehledem členů, s možností přepnutí na přehled uložených rozdělení.

- **Úprava existující skupiny**



Obrázek 1.1: Diagram případů užití

1. Případ užití *Zobrazení přehledu skupiny*.
2. Uživatel přidá člena, vybere ze seznamu člena, kterého chce ze skupiny odebrat, nebo zvolí nové jméno skupiny.
3. Aplikace zkontroluje, zda je takováto změna možná a změny uloží.

- **Zobrazení statistiky člena**

1. Případ užití *Zobrazení přehledu skupiny*.
2. Uživatel vybere ze seznamu člena.
3. Aplikace zobrazí obrazovku s přehledem člena.

- **Přejmenování člena**

1. Případ užití *Zobrazení statistiky člena*.

2. Uživatel stiskne tlačítko pro úpravu jména člena.
3. Uživatel vyplní nové jméno.
4. Aplikace zkontroluje unikátnost jména ve skupině a jméno uloží.

- **Rozdělení skupiny**

1. Uživatel vybere ze seznamu skupinu, kterou si přeje rozdělit.
2. Uživatel nastaví parametry rozdělení (omezení velikostí týmů, vynucení/zakázání dvojic, apod.)
3. Aplikace vytvoří rozdělení a zobrazí jej uživateli.
4. Uživatel může rozdělení upravit, případně uložit.

- **Zobrazení/úprava uloženého rozdělení.**

1. Příklad užití *Zobrazení přehledu skupiny*
2. Uživatel vybere ze seznamu rozdělení.
3. Aplikace zobrazí uživateli uživateli rozdělení.
4. Uživatel rozdělení upraví a uloží, nebo se vrátí zpět na přehled skupiny.

- **Export rozdělení**

1. Příklad užití *Zobrazení/úprava uloženého rozdělení*, nebo případ užití *Rozdělení skupiny*.
2. Uživatel zvolí *Export*.
3. Uživatel zvolí soubor, do kterého chce skupinu exportovat.
4. Aplikace provede export do zvoleného souboru.

- **Tisk rozdělení**

1. Příklad užití *Zobrazení/úprava uloženého rozdělení*, nebo případ užití *Rozdělení skupiny*.
2. Uživatel zvolí *Tisk*.
3. Aplikace převede rozdělení do tisknutelné podoby a provede tisk.

Po vytvoření případů užití se doporučuje namapovat je na funkční požadavky, a to zejména z kontrolních důvodů, abychom odhalili případné nesrovnalosti. **[[tabulka namapování]]**

Návrh řešení

2.1 Volba technologií

Logika a struktura jakékoli aplikace je úzce spjata s výběrem technologií, pomocí kterých je vystavěna. Vzhledem k tomu, že v zadání není určeno jaké technologie má implementace využívat, bude nutné je zvolit.

Pro tvorbu desktopových aplikací se nabízí celá řada možností a jazyků. Vždy je ovšem potřeba myslet na to, že bude nutné vytvořit nejen logickou část aplikace, ale také GUI, proto je musíme zvolit také technologii pro jeho tvorbu.

2.1.1 Volba programovacího jazyka

Jedním z nejpoužívanějších jazyků pro tvorbu desktopových aplikací je jednoznačně C++[?]. Mezi jeho výhody patří vysoká efektivita vedoucí k rychlejším běhům programů. To je ovšem vykoupeno větší náročností pro programátora, jelikož se musí starat o uvolňování a alokování paměti apod. Proto se podle mého názoru nehodí pro tento projekt.

Další možností by bylo využití jazyku Java a tvorba GUI v JavaFX. Java řeší mnoho nevýhod C++ například integrovaným garbage collectorem, který se stará o uvolňování paměti. To s sebou samozřejmě přináší cenu v podobě rychlosti.[?] Java se ovšem v poslední době přesunuje spíše na servery a vývoj GUI je poměrně náročný. Další nevýhodou pro tento projekt je nutnost instalace Java virtual machine (JVM) na uživatelské stroji. JVM sice nabízí naprostou přenositelnost mezi platformami, to ale není pro tento projekt vůbec přínosem, jelikož cílem projektu je vytvořit aplikaci čistě pro operační systém Linux.

Nakonec jsem se rozhodl pro programovací jazyk Python. Python je na rozdíl od obou zmíněných jazyků tzv. dynamicky typovaný (někdy také jen dynamický) jazyk. Dynamicky typované jazyky se liší od staticky typovaných jazyků tím, že proměnná není vázána na typ, jen na objekt. To prakticky zna-

mená, že v průběhu programu se v jedné proměnné může vystřídat více typů (např. integer a string).[?] Tato vlastnost může, ale nemusí, některé operace a úkony usnadnit a v některých případech také značně zlepšit čitelnost kódu. Na druhou stranu může docházet k chybám za běhu, které by se u staticky typovaných jazyků projevíly již při kompilaci.

2.1.2 Python

U Pythonu je důležité, jakou verzi budeme používat, jelikož na konci roku 2008[?] byla vydána verze 3.0, která již není zpětně kompatibilní. To samozřejmě vedlo k částečnému rozdělení Python komunity, protože někteří programátoři musí stále pracovat se starší verzí Pythonu, například protože rozšiřují již existující aplikaci nebo pracují s knihovnou, která operuje se starší verzí Pythonu, a také se vždy najdou tací, jímž se změny nelíbí a proto budou pokračovat s prací se starší verzí. Dle ankety z roku 2014[?] by téměř polovina dotazovaných započala nový projekt v Pythonu verze 2.x, spíše než v některé z 3.x verzí. Zároveň anketa ukazuje, že nejpoužívanější verzí Pythonu byla verze 2.7, následovaná v té době aktuální, verzí 3.4. V anketě také můžeme vidět, že nejčastějším (59%) důvodem pro setrvání v Pythonu 2.x byly závislosti na různých knihovnách a balíčcích, z kterých mnoho není do dnešní doby aktualizováno do Pythonu 3. Předpokládám, že kdyby anketa proběhla v dnešní době, tato čísla a celkové vyznění ankety, by se příliš nezměnilo. Jenom nejpoužívanějším Pythonem 3.x by snad byl nejnovější Python 3.6.

Pro tento projekt jsem zvolil Python 3.6, jelikož se starším Pythonem nemám zkušenost a ani nepotřebuji využívat žádnou knihovnu, která by nebyla k dispozici pro tuto verzi. Jediná nevýhoda verze 3.6, tentokrát oproti starší verzi 3.4 je absence knihovny PySide, která slouží jako „vylepšení“ knihovny PyQt, která zprostředkovává napojení Pythonu na GUI framework Qt. Výhody PySide jsou ale, podle mého názoru, téměř zanedbatelné, proto mi pro tento projekt plně stačila jen knihovna PyQt5.

2.1.3 GUI

Jak jsem již naznačil v předchozí sekci, pro tvorbu GUI jsem se rozhodl využít knihovnu PyQt5 a tudíž i framework Qt.

Qt je multiplatformní framework pro vytváření uživatelského rozhraní. Ačkoli je to knihovna pro C++, existují implementace pro různé jazyky jako například Python, Ruby, nebo C# a další. Qt podporuje lokalizaci, správu vláken, nabízí předpřipravená řešení pro časté případy (dialogy pro otevírání souborů, tisk apod.) a přejímá nativní vzhled operačního systému. Mezi další výhody patří velmi dobrá a přehledná dokumentace[?] a kvalitní vývojové programy jako QtCreator pro návrh, nebo Linguist pro překlad. Aktuální verze Qt je verze 5.8.

2.1.4 Persistence dat

Poslední důležité rozhodnutí byla volba způsobu zachování dat mezi jednotlivými běhy aplikace – persistence dat. Vzhledem k povaze projektu se jedná o zápis na pevný disk uživatelského počítače a nabízí se v podstatě dvě možnosti, serializace nebo databáze.

Serializace označuje proces, při kterém je instance objektu nějakým způsobem převedena do uložitelné podoby. Zároveň může být cílem serializace také minimalizace velikosti dat, z důvodu úspory místa na disku, nebo rychlejšího přenosu po síti.[?] Opačným procesem je deserializace, čili získání těchto dat zpět. Python nabízí pro serializaci řadu modulů, například modul *pickle*.[?]

Jako databáze se dá využít SQLite – knihovna napsaná v jazyce C, která poskytuje jednoduchou databázi umístěnou na pevném disku, pro Python dostupná například pomocí modulu *sqlite3*.[?] Přístup k datům v databázi je realizován pomocí SQL dotazů.

Pro tento projekt jsem zvolil databázový přístup, jelikož relační databáze ze své podstaty usnadňuje vyhledávání dat (pomocí dotazů), dále také není při každém běhu programu nutné načíst všechna uložená data do paměti ve formě objektů Pythonu, což by při použití serializace nejspíše bylo nutné.

2.2 Doménový model

Doménový model slouží k pochopení vztahů, konceptů a objektů v cílové business doméně (oblasti). Snaží se dekomponovat doménu do objektů a vztahů mezi nimi.[?] Přestože je doménový model nezávislý na platformě a objekty v tomto modelu nejsou softwarové objekty, je doménový model také základem pro design softwaru.

Doménový model 2.1 pro tento projekt je celkem jednoduchý, obsahuje pouze čtyři logické objekty:

- **Group** – skupina

Skupina je vlastně agregace (sjednocení) členů. Každá skupina má jméno.

- **Member** – člen

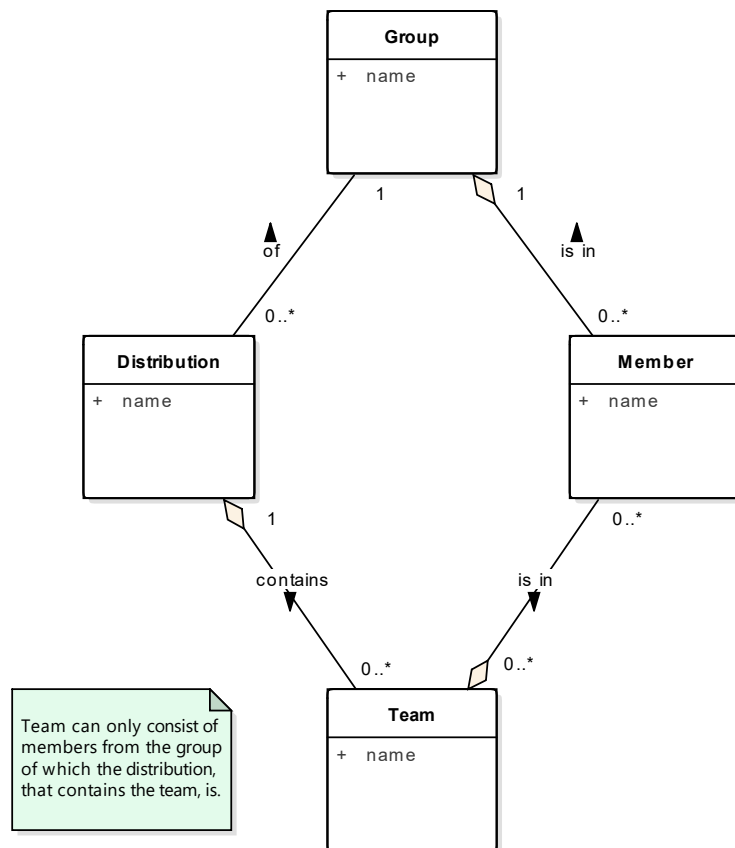
Člen má jméno a patří právě do jedné skupiny.

- **Distribution** – rozdělení

Rozdělení symbolizuje výsledek procesu rozdělení skupiny do týmů, je to agregace týmů. Rozdělení se váže právě k jedné skupině. Zároveň má také jméno (kvůli uložení).

- **Team** – tým

Tým je agregace členů, jak můžeme vidět na poznámce v diagramu 2.1, tým může obsahovat pouze členy ze skupiny svého rozdělení. To znamená, že pokud rozdělení *dist* patří (rozděluje) skupinu *A*, pak každý



Obrázek 2.1: Doménový model

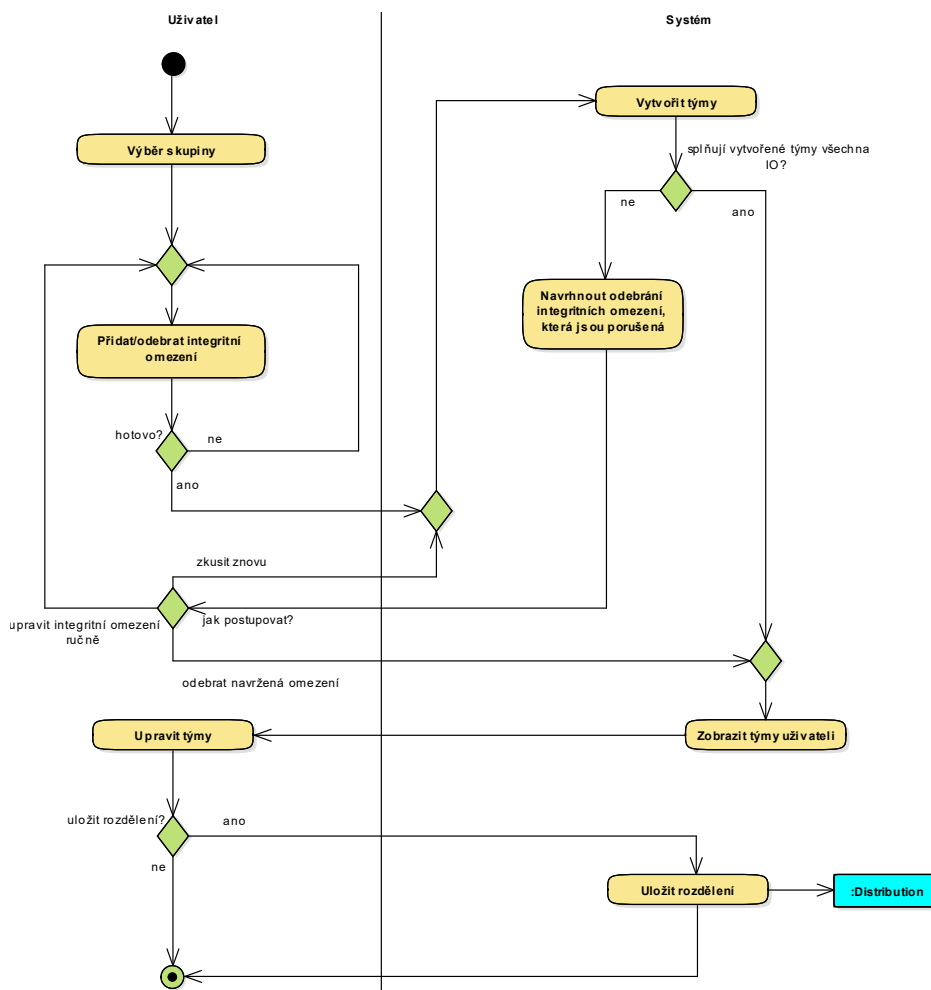
tým z rozdělení *dist* může obsahovat pouze členy skupiny *A* a žádné jiné. Tým má opět své jméno, abychom mohli rozlišit týmy v rozdělení.

2.3 Proces rozdělení skupiny

Nejdůležitějším a zároveň také nejkomplikovanějším procesem v aplikaci je vytvoření rozdělení, tj. rozdělení skupiny do týmů. Proto jsem se rozhodl při návrhu jej popsat a ujasnit, jak bude vlastně takové rozdělení probíhat. Pro popis tohoto procesu jsem využil diagram aktivit (activity diagram).

Diagram aktivit slouží právě k popisu procesů. Ať už se jedná o bussiness procesy nebo proces přímo v systému, obojí můžeme snadno a přehledně po-

2.3. Proces rozdělení skupiny



Obrázek 2.2: Diagram aktivit znázorňující proces vytvoření rozdělení

psat právě diagramem aktivit. Dle [1] „nabízí univerzální mechanismus pro modelování chování, který můžeme využít, kdykoli se nám to bude hodit“ a právě při popis procesu rozdělení skupiny se tento diagram náramně *hodí*.

Při pohledu na diagram 2.2 může vyvstat otázka, proč, pokud systém nenajde vhodné rozdělení do týmů, má uživatel možnost opakovat hledání a proč by to chtěl udělat. Tuto možnost jsem do diagramu, a také do aplikace, doplnil až v průběhu implementace a souvisí to se způsobem (algoritmem), který se používá pro nalezení vyhovujícího rozdělení. Více o tomto algoritmu v sekci [\[\[odkaz na sekci s algoritmem\]\]](#).

Implementace

Závěr

Literatura

- [1] Jim Arlow, I. N.: *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, a.s., 2007.

Seznam použitých zkratk

GUI Graphical user interface

UML Unified modeling language

JVM Java virtual machine

Obsah přiloženého CD

	readme.txt.....	stručný popis obsahu CD
	exe	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis	zdrojová forma práce ve formátu L ^A T _E X
	text	text práce
	thesis.pdf	text práce ve formátu PDF
	thesis.ps	text práce ve formátu PS