VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro vizuální modelování pomocí jazyka modelování i-star

i-Star Modeling Tool

2018

Lukáš Blahut

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Blahut**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroj pro vizuální modelování pomocí jazyka modelování i-star
i-Star Modeling Tool

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit nástroj pro modelování požadavků-cílů. Nástroj bude podporovat modelování v jazyce i*.

1. Seznamte se s jazykem i-star.
2. Seznamte se s případnými stávajícími nástroji pro jazyk i*.
3. Navrhněte a implementujte online nástroj.
4. Vyzkoušejte funkčnost nástroje na ukázkovém příkladu.

Seznam doporučené odborné literatury:

[1] Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. Software Engineering: Theory and Practice: Prentice Hall, ISBN 0136061699
[2] Pressman, Roger S. 2010. Software Engineering : A Practitioner's Approach. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977
[3] Sommerville, Ian. 2010. Software Engineering. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151
Další literatura podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2018 .....

Rád bych na tomto místě poděkoval vedoucímu mé bakalářské práce Ing. Svatoplukovi Štolfovi, Ph.D. za poskytnuté rady, čas strávený konzultacemi a také příležitost pro vypracování této práce.

Abstrakt

Náplň této bakalářské práce spočívá v návrhu a implementaci online softwarového nástroje pro vizuální modelování diagramů modelů cílů, a to pomocí jazyka modelování i-star. Primární funkcionalita, kterou nástroj uživateli zpřístupňuje, je tvorba těchto modelů, jejich editace a následovné vyexportování celého modelu pro budoucí užití. Nástroj podporuje tvorbu modelů cílů na základě dvou metod. Hlavní metodikou je modelování využívající právě jazyka i-star, druhotnou pak tvorba standartních digramů případů užití na základě notace UML. Práce se skládá ze dvou částí, kde obsah první časti se zabývá popisem jazyka i-star, cílového modelování a také srovnáním již existujících nástrojů podporující práci s jazykem i*. Druhá část se pak zaměřuje na návrh a implementaci online nástroje s využitím technologií třetích stran.

Klíčová slova: online, případ užití, jazyk modelování i-star, cílové modelování

Abstract

The goal of this bachelor thesis was to design and implement an online software tool, dedicated for working with goal modeling diagrams, using the i-star modelling laguage. Primary functionality of implemented application lies in creation and modification of goal models as well as in ability to export and import existing diagrams. Tool supports modelling based on two distinct methods. The main method is using the i-star modelling language while the other one is based on using standart UML use-case diagrams. Bachelor thesis can be divided into two main parts. The first one deals with the description of i-star language, after that a detailed goal modeling description as a whole and the comparison of the existing i-star capable tools is also mentioned. Design and implementation details of created online application are discussed in the second part of this paper.

Key Words: online, use-case, modeling language i-star, goal modeling

Obsah

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
CSS	–	Cascading Style Sheets
DIV	–	Document Division
DOM	–	Document Object Model
HTML	–	Hyper Text Markup Language
JDK	–	Java Development Kit
JSON	–	JavaScript Object Notation
MDA	–	Model Driven Architecture
UML	–	Unified Modeling Language
URL	–	Uniform Resource Locator
SVG	–	Scalable Vector Graphics

Seznam obrázků

Seznam tabulek

1 Úvod

Před samotnou implementací jakéhokoliv počítačového programu je třeba zvážit všechny možné aspekty ovlivnující návrh, údržbu a tvorbu takového systému. Touto činností se zabývá disciplína softwarového inženýrství. Hlavními nástoji, které nám pomáhají při těchto analýzach jsou grafické jazyky pracující s modely, znázorňující různé aspekty, které ovlviňují vývoj takového softwaru. UML je jedním takovýmto grafickým jazykem, jehož účelem je vizualiazce, specifikace, navrhování a dokumentování programových systémů. Lze s jeho pomocí zachytit jednak konceptuální prvky či systémové funkce, ale také konkrétní prvky jako jsou příkazy programovacích jazyků, databázové schémata a znovupoužitelné programové komponenty. Nejčastěji využívané modely jsou případy užití, třídní diagramy, sekvenční diagramy a mnoho dalších. Existují ale také jiné pohledy na modelování popisující chování systému, kde se snažíme zachytit cíle a úkoly navrhovaného systému.

Tato bakalářská práce popisuje návrh a implementaci webové aplikace pro vizuální tvorbu právě jednoho z těchto přístupů pro popis chování softwarového produktu, konkrétně cílového modelování. Aplikace podporuje tvorbu těchto modelů cílů dle notace modelovacího jazyka i-star, ale i standartní diagramy případů užití. První kapitola je věnovaná popisu modelovacího jazyka i-star, který je základem logiky celé aplikace. Druhá kapitola se zaměřuje na srovnání několika existujících nástrojů, které podporují prácí s jakyzem i-star. Následuje třetí, tedy poslední kapitola, kde je popsán návrh a implementace celé webové aplikace spolu s javascriptovými knihovnami třetích stran, bez kterých by vznik aplikace nebyl možný.

2 Popis jazyka i-Star

Primárním zdrojem informací pro celou kapitolu 2, kromě již v textu zmíněných, je online dokumentace jazyka iStar, dostupná ze zdroje: [1].

Modelovací jazyk I-star byl vytvořen, aby vyplnil mezeru ve spektru konceptuálních modelovacích jazyků, se zaměřením na úmyslné, sociální a strategické stránky. Tedy otázky proč, kdo a jak.

i-Star nachází své využití v mnoha oblastech, jako například zdravotnictví, analýzy bezpečnosti nebo elektronickém obchodování. Jazyk i* má mnoho akademických aplikací. Diverzita a množství variací jeho rozšíření může ale pro uživatele, který se s tímto modelovacím jazykem setkává poprvé, znamenat dosti náročné používání. Naštěstí existuje několik zdrojů, popisujících logiku jazyka i*.

2.1 Historie

První verze jazyka i* vznikla v druhé polovině devadesátych let, a to v podobě modelovacího frameworku, který se zaměřoval na popis objektů a jejich cílů. Základem frameworku byl modelovací jazyk spolu s technikami pro analýzu vytvořených modelů. Netrvalo dlouho a jeho existence upoutala pozornost skupin v oblastech byznys modelování a inženýrství požadavků. Účelová otevřenost jazyka vedla ke vzniku řady návrhů pro rozšíření nebo předefinování existujících konstrukcí. Větsinou toho bylo dosáhnuto rozšířením specifikací některých sémantických problému, které nebyly v klíčovém návrhu řádně řešeny, popřípadě konkrétní návrhy nových konstrukcí pro specifické oblasti.

Flexibilita jazyka i* navíc pro vývojáře a výzkumníky poskytovala možnost zakomponovat úpravy, jež nejlépe splňovaly jejich potřeby. Bohužel tento přístup nesl své nevýhody. Tou nejzávažnější nevýhodou byl problém rozšíření frameworku mimo komunitu odborníků, což vedlo k sérii komplikací:

- Obtižné zaškolení pro nově příchozí uživatele.
- Školitelé postrádají jednotné jádro znalostí pro výuku.
- Praktikům nejsou poskytnuty stanovené reference pro využívání jazyka i* ve svých projektech.
- Poskytovatelé technologií nemohou jednoznačně určit, které z konstrukcí jsou klíčové pro implementaci, a které techniky by tedy měly být na jejich základě postaveny.

Starší verze jazyka i* tedy jednoznačně nebyla dokonalá. Bylo potřeba dosáhnout jisté rovnováhy v otevřenosti frameworku. Výzkumné komunity zabývající se jazykem i* proto začali s iniciativou pro identifikaci dohodnutých souborů klíčových konceptů jazyka. Cílem těchto komunit bylo

zachování možnosti přizpůsobit si framework na základě svých potřeb, zatímco však budou dodrženy pravidla týkající se úpravy základních konstrukcí. Konečným výsledkem byla první iterace standartizačního procesu jazyka i*. Pro jednoznačné rozlišení od svých předchůdců byla tato iterace pojmenována iStar 2.0.

2.2 Cílové modelování

Model cíle můžeme chápat jako element inženýrství požadavků, který také může být použit v rozšířené podobě pro byznysově zaměřené analýzy [2]. Na definované cíle lze pohlížet jako na jednotlivé úkoly, kterých by systém v rámci svého chování měl dosáhnout. Konkrétně využitím kooperací aktérů v zamýšleném softweru a daném prostředí. Největší přínos nabízí cílové modelování v prvnotních fázích projektu.

Projekty mohou zvážit jakým způsobem zamýšlený systém splňuje definované organizační cile. Poskytuje tak jednoduchou odpověd na otázku, proč je v první řadě systém vůbec zapotřebí a jak jsou řešeny zájmy zůčastněných stran.

Model cíle představuje schopnost vyjádřit vztahy mezi systémem a jeho prostředím a to nejen v ohledu na to, co by systém měl dělat, ale také proč [2]. Pochopení důvodů proč je systém zapotřebí, je užitečné kromě očividného opodstatnění vzniku systému rovněž v kontextu návrhové trendu dnešní doby. Vyvíjené systémy se stále častěji využívají k zakomponování zásadních změn v obchodních procesech, než k automatizaci dlouhotrvajících procesů.

Další funkcionalitou modelů cílů je objasňování požadavků. Specifikace cílů vede k řadě otázek. Zajímá nás totiž proč tyto požadavky vznikly, jak je možné jich dosáhnout, popřípadě jestli existují alternativní přístupy. V rámci tohoto procesu často bývají odhaleny požadavky zůčastněných stran, což vede k minimalizaci rizik buďto přehlédnutí nebo přehnaného specifikování těchto požadavků. Tyto modely také umožnují analyzovat jednotlivé cíle systému. Na základě výsledků těchto analýz můžeme přeorganizovat rozsáhlé cíle do menších, dílčích cílů.

Zabývá se také řešením konfliktů výkonu, fexibility, bezpečnosti, nákladů a dalších podstatných askeptů návrhu systému, které cílové modelování může odhalit. Příkladem konfilktu můžou být prodichůdné zájmy jednotlivých zůčastněných stran. Konflikty vzniklé splněním cíle, který svou existencí zasahuje do schopnosti plnění jiných cílů. Mímo již zmíněné výhody cílového modelování, dokáže měřit úroveň požadované úplnosti. Požadavky systému můžeme obecně považovat za splněné, pakliže splňují všechny cíle v modelu.

Existuje řada notací, které se používají pro modely cílů při vývoji softwaru. Patří mezi ně například i*, KAOS nebo UML diagram případu užití.

Notace modelování cílů jazyka i* poskytuje dva typy diagramů:

- Strategická závislost, definující vztahy mezi rolemi v kontextu konkrétních cílů, kdy jedna role závisí na jiné roli, která funguje jako poskytovatel této závislosti.
- Strategické odůvodnění, analyzující cíle identifikované modelem strategických závislostí a jejich rozdělení na dceřiné cíle a úkoly.

Jazyk i* reprezentuje každého aktéra, ať už se jedná o jakýkoliv typ, jako objekt obsahující cíle, úkoly a zdroje, které vlastní. Důkladnější popis notace pokračuje v následujících podkapitolách.

2.3 Aktéři a jejich variace

Aktéři jsou vzhledem k sociální povaze modelovacího jazyka i* základním prvkem, s kterým pracujeme. Představují aktivní, autonomní entity, které se snaží dosáhnout svých cílů tím, že uplatní osobní prostředky ve spolupráci s ostatními aktéry. V aktuální verzi jakyza iStar 2.0 rozlišujeme dva typy aktérů.

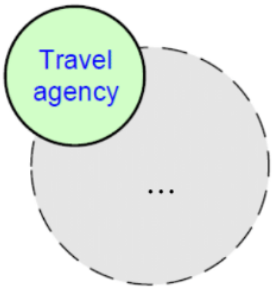
- *Role:* abstraktní popis chování sociálního aktéra v rámci specializovaného kontextu nebo doméně, kterou je součástí. Příkladem může být Student, Profesor apod.
- *Agent:* aktér s pevnou, fyzickou manifestací, jakou představuje kupříkladu konkrétní jedinec, organizace či oddělení ve společnosti. Příkladem může být Vysoká Škola Báňská, Jan Novák, Cestovní kancelář apod.

Nastane-li situace, kdy na typu využívaného aktéra, ať už z důvodu daného scenáře nebo fáze modelování nezáleží, existuje obecný aktér bez bližší specializace. Scénář, kde lze využít tohoto obecného aktéra je například situace, kdy při modelování prozatím nevíme, jestli bude aktér zastávat funkci role nebo agenta.

Pro grafickou reprezentaci aktérů využíváme kruh. V případě agentů je pro jednoznačné určení navíc přidaná přímka v horní části kruhu. Role pro změnu používá křivku v spodní části kruhu, jak můžeme vidět na obrázku 1. Samotné grafické notace se řídí pokyny, které si původní jazyk i* nadefinoval.

Obrázek 1: Grafické znázornění typů aktérů v pořadí: Aktér, Role a Agent (Zdroj: [1])

Rámec úmyslů aktéra je vyjádřen explicitně pomocí aktérovy hranice, což je grafický kontejnér pro prvky úmyslů (Sekce 2.5) a jejich vzájemná propojení (Sekce 2.7). Grafické znázornění této aktérovy hranice můžeme vidět na obrázku 2. Prvky a jejich vzájemná propojení jsou umístěny v šedě vyznačené oblasti.

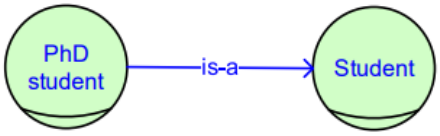


Obrázek 2: Grafické znázornění hranice aktéra (Zdroj: [1])

2.4 Asociační propojení aktérů

Aktéři často bývají vzájemně propojení. V jazyce iStar 2.0 existují asociace, které definují nebo popisují tyto vztahy. Tyto asociace mají binární podobu, což znamená, že vytvoření tohoto propojení je možné pouze mezi jednotlivými odlišnými aktéry. Používají se dva typy těchto propojení:

– *Aktér-je (is-a):* reprezentuje v jazyce iStar 2.0 koncept generalizace - specializace. Role mohou být specializací jiných rolí, stejně tak obecný typ aktéra může být specializací jiného obecného aktéra. Kupříkladu PhD student (role) může být případem specializace Studenta (jiné role), tak, jak vidíme na obrázku 3. Jelikož agenti reprezentují konkrétní entity, nemá mezi nimi vznik této vazby logické opodstatnění (např., Jan Novák nemůže být Petr Dostál).



Obrázek 3: Grafické znázornění vztahu: Aktér-je (Zdroj: [1])

– *Účast-aktéra (participates-in):* reprezentuje jakýkoliv jiný vztah mezi dvěma odlišnými aktéry, než již zmíněný: generalizace - specializace. Nejsou zavadené žádné restrikce určující, které typy aktérů mohou být tímto vztahem propojeny. V závislosti na typu propojených aktérů, tento vztah nabývá různých významů. Jedna z typických situací je zachycená na obrázku 4, kde zdrojem je agent a cílem je role, chápeme v kontextu: "agent hraje roli"(např., Mike White hraje roli PhD studenta).

Asociační propojení aktérů využívá grafického znázornění šípky, která je orientovaná ve směru od zdroje k cíli tohoto spojení. Je potřeba, aby spojení také obsahovalo korespondující štítek, jednoznačně určující o který typ spojení se jedná.

Obrázek 4: Grafické znázornění vztahu: Účast-aktéra (Zdroj: [1])

2.5 Prvky úmyslů

Prvky úmyslů představují záležitosti modelovaného systému, které jsou součástí aktérů a které samotný aktér pro realizaci své práce potřebuje. Jsou tedy klíčovými objekty jakyza iStar 2.0. Prvek úmyslu obsažen v aktérově hranici reprezentuje obecně něco, čeho chce aktér dosáhnout. Každý z těchto prvků se také může vyskytovat mimo aktérovu hranici, konkrétně součástí sociální závislosti mezi dvěma aktéry (Sekce 2.6). V této části budou prvky úmyslů popsány z hlediska první varianty, tedy jakožto součást aktéra.

V jazyku jsou definovány následující prvky úmyslů:

- *Cíl:* stav, kterého chce aktér dosáhnout a který má jasné kritéria pro jeho splnění.
- *Kvalita:* atribut, od kterého aktér očekává určitou úroveň uspokojení. Mějme vyvíjený systém, reprezentující entitu, kde kvalita představuje výkon této entity. Jiný příklad entity by mohl být byznys, který je analyzován a jeho sledovanou kvalitou jeho roční profit. Úroveň uspokojení je buďto přesně definována nebo ponechána bez bližšího určení. Kvality je možné chápat jako způsob hledání alternativních cest, jak dosáhnout splnění cíle.
- *Úkol:* reprezentace činností, po kterých aktér požaduje jejich provedení, většinou za účelem splnění cíle.
- *Prostředek:* fyzická či informační entita, ke které potřebuje mít aktér přístup, aby mohl provést určitý úkol.

Pro grafickou reprezentací cílů používáme ovály, zatímco kvalita využívá zakřivený tvar připomínající oblak. Úkoly jsou pak znázorňovány pomocí hexagonu pro zvýraznění jejích strukturovanější definice z hlediska procesu, který je třeba dodržovat. Prostředky používají obyčejného tvaru obdelníku. Ukázku jednotlivých tvarů je možné si prohlédnout na obrázku 5).

Obrázek 5: Grafická podoba prvků úmyslů v pořadí: cíl, kvalita, úkol a prostředek (Zdroj: [1])

2.6 Sociální závislosti

Závislosti v jazyce iStar 2.0 reprezentují sociální vztahy. Na základě předpokladu, že aktéři mohou zastávat logickou roli jednotlivých jedinců, organizací, technických systémů nebo jakoukoliv jejich kombinaci, činí iStar 2.0 sociálně-technickým modelovacím jazykem. Sociální závislost je definována jako vztah s pěti argumenty:

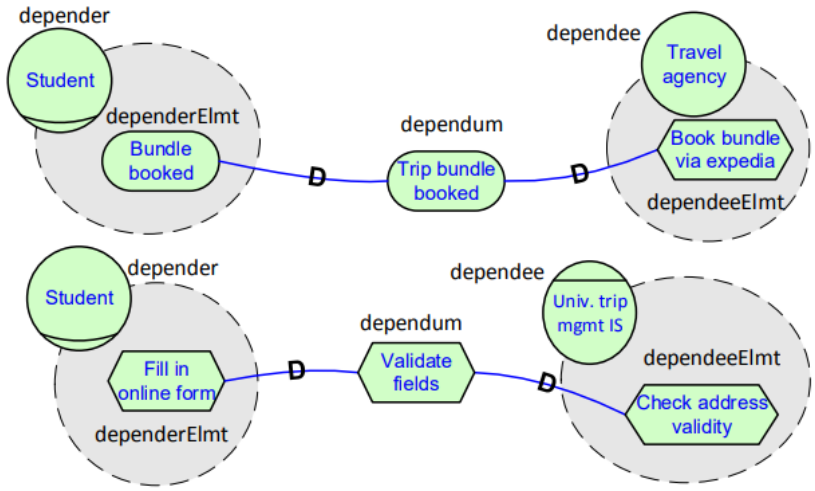
- *Závisející (depender)* je aktér, který závisí na nečem, (Předmět závislosti) co mu má být poskytnuto.
- *prvek Závisejícího(dependerElement)* je prvek úmyslu umístěn v hranici závisejícího aktéra odkud tato sociální závislost vzniká, což vysvětluje proč vlastně existuje.
- *Předmět závislosti (dependum)* je prvek úmyslu, který představuje kontext a význam samotné závislosti.
- *Poskytovatel (dependee)* je aktér, jehož úkolem v této zavislosti je tento předmět závisejícímu aktérovi poskytnout.
- *prvek Poskytovatele (dependeeElement)* je prvek úmyslu, který vysvětluje jakým způsobem chce závislost poskytující aktér závislému aktérovi předat.

Příklad sociální závislosti můžeme vidět na obrázku 6. Konkrétně tedy propojuje prvek závisejícího aktéra k předmětu vzniklé závislosti, a ten se zase odkazuje na prvek poskytujícího aktéra. Propojení nese štítek "D"nahrazující roli šipky, směřující od prvků úmyslů obou zučastněných aktérů, nebo mezi nimi samotnými, pakliže jsou prvky úmyslů vynechány.

Oba prvky úmyslu lze při vytváření sociální závislosti vynechat. Propojujeme tedy jen samotné aktéry. Možnost vynechání osobních prvků úmyslů závisejícího/poskytujícího aktéra lze využít při vytváření počátečního modelu - Strategické Závislosti (Sekce 2.8). Další situací je například, když nemáme dostatečné znalosti, tedy nevíme proč závisející aktér požaduje vznik této závislosti, nebo jakým způsobem by poskytující aktér zajišťoval její splnění.

Existuje taktéž varianta, kde je specifikován pouze jeden z prvků úmyslů, jak je vidět na obrázku 7. Typy objektů jsou totožné s typy, které používají prvky úmyslů. Reprezentují předmět závislosti a zároveň poskytují bližší informace of sémantice vztahu:

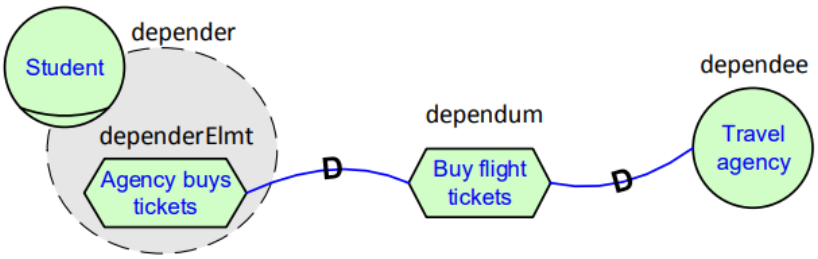
- *Cíl:* od poskytujícího aktéra je očekáváno, že splní tento cíl, má však volnou ruku v podobě způsobu, kterým toho dosáhne.
- *Kvalita:* od poskytujícího aktéra je očekáváno, že dostatečně uspokojí požadovanou kvalitu, dle prostředků, které si může sám zvolit.
- *Úkol:* od poskytujícího aktéra je očekáváno, že splní úkol předepsaným způsobem.



Obrázek 6: Příklad vztahů závislostí 1 (Zdroj: [1])

– *Prostředek:* od poskytujícího aktéra je očekáváno, že zpřístupní prostředek závisejícímu aktérovi.

Na základě použitého typu předmětu závislosti mají závisející a poskytující aktéři různou úroveň volnosti jak s touto vazbou zacházet. Cíle a kvality poskytují největší volnost, následují úkoly, kde je tato úroveň výrazně menší a nakonec prostředky, které nepovolují skoro žádnou volnost.



Obrázek 7: Příklad vztahů závislostí 2 (Zdroj: [1])

2.6.1 Pravidla a restrikce

Závisející aktér nemá možnost si diktovat, jakým způsobem mu poskytovatel tuto závislost zajistí, včetně jeho interní agendy - konkrétní způsob, za užití jeho osobních prvků úmyslů, který poskytovatel využil.

Může nastat situace, kde jeden nebo více elementů zachycující sociální závislost mají stejné jméno. Pořád ale i v takovémto případě jsou všechny tyto elementy jednoznačně rozlišitelné, protože reflektují odlišný pohled na sociální závislost mezi dvěma aktéry. Jednotlivé vztahy

závislosti by neměli sdílet stejný předmět závislosti, jelikož každý takový předmět je z konceptuálního úhlu pohledu odlišný objekt. V některém případě by předmět závislosti v jedné konkrétní sociální závislosti splněn byl, zatímco v jiném tomto vztahu ne. Jinými slovy tedy aktér nemůže být závislý v rámci jednoho konkrétního předmětu závislosti na více než jednom odlišném aktérovi, nebo také dva aktéři nemohou být závislí na stejném předmětu pocházející od dalšího aktéra.

2.7 Propojení prvků úmyslů

Existují čtyři druhy propojení mezi prvky úmyslů: *upřesnění (refinement)*, *potřeba (needed-by)*, *přínos (contribution)* a *způsobilost (qualification)*. Jednotlivé typy jsou podrobnější popsány v následujících sekcích. Sumarizace podporovného užívání propojení můžeme vidět v křížové tabulce 1.

Tabulka 1: Možnosti vzájemného propojení (Zdroj: [1])

Zdroj / Cíl propojení	Cíl	Kvalita	Úkol	Prostředek
Cíl	Upřesnění	Přínos	Upřesnění	N/A
Kvalita	Způsobilost	Přínos	Způsobilost	Způsobilost
Úkol	Upřesnění	Přínos	Upřesnění	N/A
Prostředek	N/A	Přínos	Potřeba	N/A

2.7.1 Upřesnění

Notace jazyka i*, standart iStar 2.0, nabízí generický vztah označený jako: upřesnění, který hierarchicky propojuje cíle a úkoly. Upřesnění je vztah, který vzájemně propojuje jednoho rodiče k 1-n potomkům. prvek úmyslu může být rodičem v nanejvýš jednom takovémto vztahu. Využívají se dva typy upřesnění, které můžeme aplikovat na jakéhokoliv rodiče (cíl/úkol). Oba typy definují logický operátor, kterým spojuje rodiče s potomkem:

- *AND:* splnění úkolů/cílů všech n potomků, (kde n≥2) vede k splnění úkolu/cíle rodiče.
- *OR:* splnění úkolu/cíle alespoň jednoho z n potomků, (kde n≥2) vede k splnění úkolu/cíle rodiče.

Rodič může být spojen se svými potomky buďto typem AND-upřesnění nebo OR-upřesnění, nelze použít oba typy pro jednoho rodiče najendou. Na základě propojených prvků má vztah upřesnění odlišný význam:

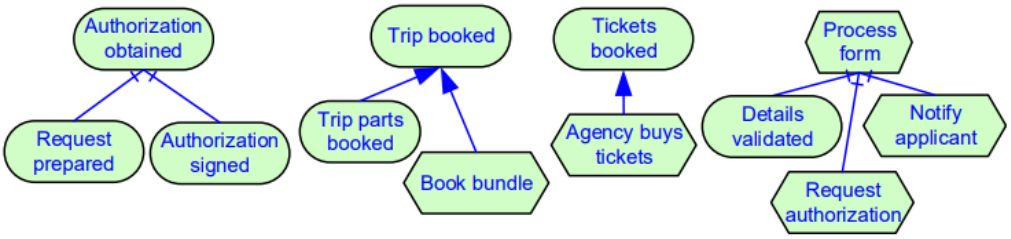
- Pokud je rodičem cíl:
  - v případě, že používáme AND typ upřesnění, potomek typu cíl funguje v roli dílčího cíle, který je součástí cíle rodiče. Zatímco potomek typu úkol je brán jako dílčí úkol rodiče, který musí být splněn.

- v případě, že používáme OR typ upřesnění, představuje potomek typu úkol konkrétní způsob pro splnění cíle rodiče. Potomek typu cíl působí jakožto dílčí cíl, který může být splněn, jehož důsledkem je následovné splnění i cíle rodiče.

– Pokud je rodičem úkol:

- v případě, že používáme AND typ upřesnění, funguje potomek typu úkol jako součást úkolu rodiče. Je-li potomek typu cíl, jeho význam je odhalen analýzou úkolu rodiče.
- v případě, že používáme OR typ upřesnění, potomek typu cíl je cíl, jehož význam je odhalen při analýze úkolu rodiče a může nahradit původní (rodičův) úkol. Potomek typu úkol je konkrétním možným způsobem, jak provést rodičův úkol.

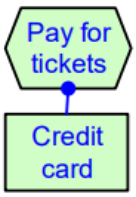
Vztahy upřesnění nediktují přesně striktní přístup zhora dolů. Mohou být sestaveny konkrétním procesem nebo dokonce smíšeným přístupem. Graficky jsou tyto vztahy znázorněny jako čáry vedené od myšlených potomků k jejím rodičům. Pro AND typ upřesnění se využívá šipka ve tvaru znaku "T", zatímco pro typ upřesnění OR se používá standartní vyplněná šipka. Ukázky obou typů refinementu můžeme vidět na obrázku 8.



Obrázek 8: Příklad vztahů typu upřesnění (Zdroj: [1])

2.7.2 Potřeba

Vztah potřeby spojuje úkol s prostředkem, což indikuje potřebu aktéra mít k dispozici tento prostředek, aby byl schopen provest daný úkol. Samotný vztah neodpovídá na otázku proč vznikl: spotřeba, čtení, modifikace, vytvoření apod. Graficky je vztah znázorněn s vyplněnou tečkou na straně úkolu, jak můžeme vidět na obrázku 9.



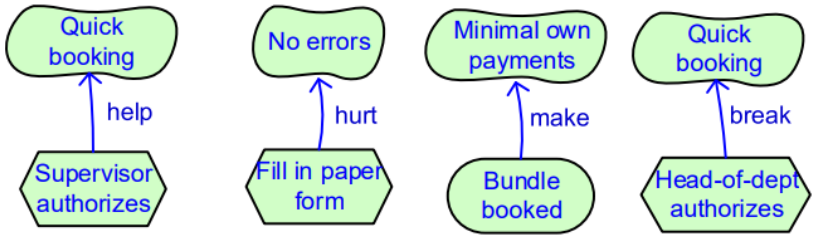
Obrázek 9: Příklad vztahu potřeby (Zdroj: [1])

2.7.3 Přínos

Vztah přínosu určuje efekt prvků úmyslů na prvky kvalit, což z nich dělá podstatný rozhodovací faktor při analyzování a rozhodování výběru alternativních přístupů k řešení úkolů a cílů. Jednotlivé přínosy vedou k zajištění důkazů o splění/selhání konkrétní kvality. Říkáme tedy, že kvalita je buďto splněna nebo uspokojena, poskytující dostačující pozitivní důkaz o splnění kvality. Opačným případěm je odepření kvality, tedy silným negativním důkazem indikující její selhání. Přínosy jsou definovány jako vztahy jdoucí od zdroje (prvek úmyslu) k cílové kvalitě. Existují čtyři typy přínosů:

- *Provedení:* zdroj poskytuje dostatečný pozitivní důkaz pro uspokojení cílové kvality.
- *Pomoc:* zdroj poskytuje slabý pozitivní důkaz pro uspokojení cílové kvality.
- *Ublížení:* zdroj poskytuje slabý negativní důkaz proti uspokojení (odepření) cílové kvality.
- *Rozbití:* zdroj poskytuje dostatečný negativní důkaz proti uspokojení (odepření) cílové kvality.

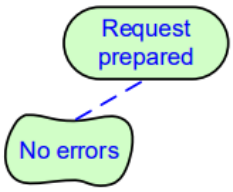
Grafická reprezentace přínosů má otevřenou šipku ve směru k cílovému objektu (kvalitě) spolu s adekvátním textovým štítkem pro jednoznačnou identifikaci typu. Na obrázku 10 můžeme vidět příklady těchto vztahů. Zdrojové objekty mohou samozřejmě být rovněž další kvality nebo prostředky.



Obrázek 10: Příklad vztahů přínosu (Zdroj: [1])

2.7.4 Způsobilost

Vztah způsobilosti vytváří propojení kvality k objektu, ke kterému se váže. Příkladem může být kvalita "bezchybný přenos"vázaná k cíli "přenos dat". Vymezuje jakým způsobem by mělo být zajištěno splnění operace nebo funkcionality daného cílového objektu. Obdobně jak je tomu v situaci, kterou vidíme na obrázku 11, kde kvalita "bez chyb (no errors)"naznačuje možnost vzniku chyby při plnění cíle "požadavek připraven (request prepared)", ke kterému se vztahuje. Pro grafické znázornění této vazby se používá přerušovaná čára.

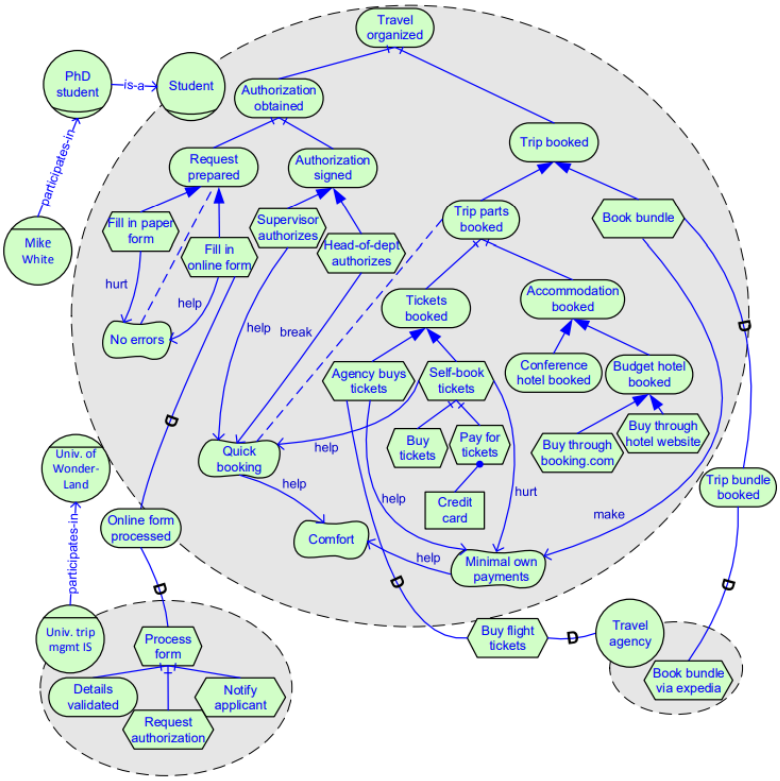


Obrázek 11: Příklad vztahu způsobilosti (Zdroj: [1])

2.8 Pohledy modelu

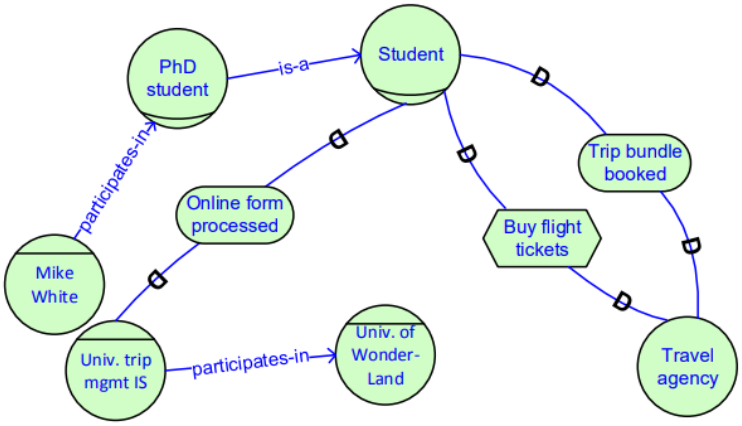
Práce s jazykem iStar 2.0 začiná, když analytik vytvoří model. Tento model může byt vizualizován prostřednictvím více pohledů nebo zobrazení modelu. Existují tři typy pohledů pocházející z originálního jazyka i*, spolu s několika rozšířeními. Jedná se o pohledy: zobrazení Strategického Odůvodnění, Strategické Závislosti a pohled Hybridní.

Strategické Odůvodnění (SO) vyzobrazuje všechny zachycené detaily, včetně aktérů, sociálních závislostí, asociačních spojení aktérů a prvky úmyslů včetně jejich propojovacích vazeb. Uživatelé mohou vidět strategická odůvodnění v rámci hranic každého aktéra. Příklad SO modelu dostupný na obrázku 12. Strategické Závislosti (SZ) vyzobrazují všechny aktéry, asociační spojení těchto aktérů a sociální závislosti, vztahující se přímo k jednotlivým aktérům. Jak můžeme vidět na obrázku 13.

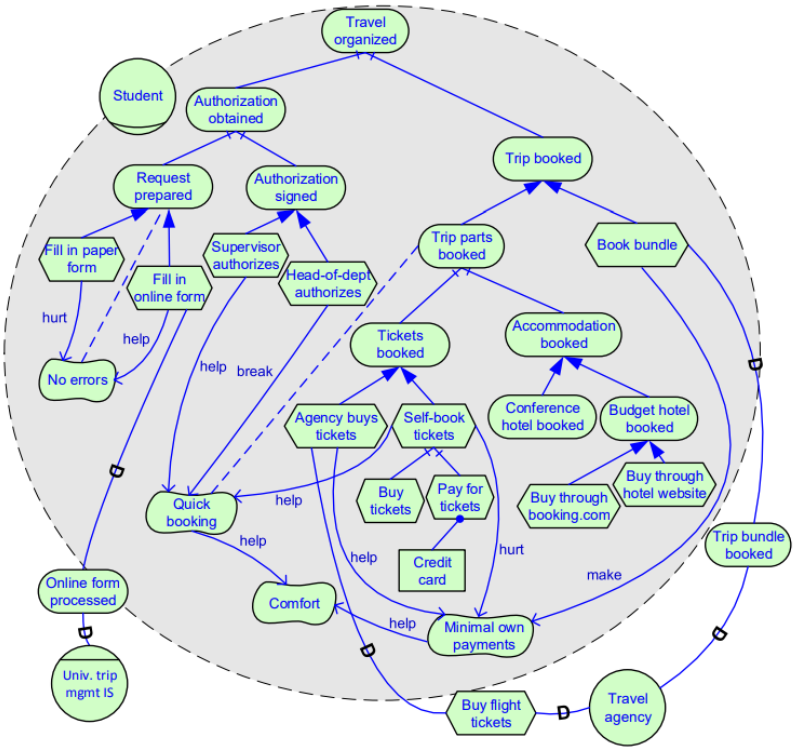


Obrázek 12: Příklad SO pohledu (Zdroj: [1])

Hybridní pohled je rovněž často využívaný, protože kombinace SO/SZ pohledů nabízí rozšířený způsob vnímání celého systému, kde některé jeho částí mají být přístupné a jiné zase ne. Pokud jsou aktéři otevření, používají aktivně své hranice, znamená to, že se zaměřují na strategická odůvodnění, zatímco někteří aktéři mohou své hranice kompletně skrýt a zachycovat čistě strategické závislosti. Příklad hybridního pohledu můžeme vidět na obrázku 14.



Obrázek 13: Příklad SZ pohledu (Zdroj: [1])



Obrázek 14: Příklad Hybridního pohledu (Zdroj: [1])

3 Popis a srovnání nástrojů podporující práci s jazykem i-Star

Při vývoji softwarových produktů je důležité věnovat adekvátní čas fázím návrhu a analýzy. Předcházíme tak, nebo výrazně změnšujeme šanci výskytů možných budoucích problémů při implementaci [3].

Primárním účelem modelování situací je hlubší pochopení návrhovaného systému, vedoucí k dobře definovaném, škálovatelném návrhu architektury celé aplikace. Jak už bylo jednou v této práci zmíněno, existujé celá řada pohledů zachycující chování jednotlivých komponent systému. Ať už sledujeme možné případy užití, třídních, sekvenčních a další diagramy zachycující rúzné úhly pohledu.

3.1 Srovnání stávajících nástrojů

Existuje řada modelovacích nástrojů podporující tvorbu výše zmíněných diagramů a modelů. Samotné nástroje jsou dostupné v různých podobách. Od desktopových aplikací, rozšířujících pluginů pro již existující nástroje, po webové nebo dokonce mobilní aplikace. V následujících sekcích budou uvedeny některé z těchto nástrojů, podporující modelování cílů na základě notace i*, aktuálních nebo starších verzí.

3.1.1 OME

Autorem prvního nástroje je skupina z Laboratoře Řízení Znalostí na Torontské Univerzitě. Projekt začal v létě roku 1998 a jeho nejaktuálnější verze byla zveřejněna v červenci roku 2000. Nástroj bohužel není volně šířitelný a k jeho získání je třeba získat licenci, kterou poskytuje na vyžádání vedoucí projektu, Prof. Eric Yu, v zmíněné době pracující na Torontské Univerzitě.

3.1.1.1 Popis nástroje

OME je nástroj poskytující operace modelování a následovné analýzy. Pracuje s přístupy modelování aktérů a cílu separátně, nebo ve spojení. Uživatelům poskytuje grafické prostředí, ve kterém mohou vyvíjet své modely. Poskytuje užitečnou základnu informací, která přináší sofistikovanou počítačem-řízenou analýzu vytvořených modelů. Hlavní funkcionalita spočívá v zajištění jasné spojitosti mezi fázemi vývoje požadavků, specifikací a návrhem architetůry pro softwarové vývojáře[4].

Nejnovější verze OME podporuje návrh modelů s využíváním i* frameworku. Pohledy modelů Strategického Odůvodnění a Strategické Závislosti, které i* využívá, jsou v tomto nástroji sloučené do jednotného modelu. Aktéři mohou být rozšířeni o vnitřní prvky a jejich propojení dodatečně. Veliké pozornosti se mu také dostává v oblasti rekonstrukce podnikových procesů.

3.1.1.2 Technické prostředky

OME je implementován jako desktopová aplikace, určená pro operační systém Windows. Aplikace byla implementovaná programovacím jazykem Java. Pro používání nástroje není třeba žádných dalších technologií. Jakákoliv další funkcionalita může být přidaná formou pluginů.

3.1.1.3 Popis klíčových funkcionalit

Aplikace podporuje vytváření modelů pomocí grafického editoru. Poskytuje funkci kliknutí a přetáhnutí požadovaného elementu do kreslící oblasti na pozici, kde si přejeme objekt umístit. Pro spojování objektů pomocí vzájemných vazeb je třeba kliknout na korespondující vazbu a následovné kliknutí na zdrojový a cílový prvek, umístěné v kreslící oblasti. Nástroj bohužel nepodropuje tvorbu modelů na základě textových vstupů.

Uživatel má možnost kdykoliv změnit velikost, jméno nebo typ elementu. Dále má k dispozici funkce pohybu, přeorganizování, mazaní, kopirování a následovné vložení těchto elementů. Všechny zmíněné operace jsou prováděné na kreslící oblasti využíváním příkazů, které jsou k dispozici v panelu nástrojů nebo skrze kontextové okno.

Jednotlivým spojům sociálních závislostí může uživatel měnit velikost, stejně tak je může přeorganizovávat, posouvat, mazat a duplikovat. Pro grafickou reprezentaci těchto spojů nástroj využivá přímých čar, které mohou být dle potřeby ručně zakřiveny, nebo zpětně vráceny do původní podoby.

Dalšími klíčovými funkcionalitami OME je možnost skrytí a zpětné zobrazení elementů, vložení i výjmutí elementů z jiného, rozšiřitelného elementu nebo přidání doplňujících popisků pro elementy [5].

3.1.2 OpenOME

Nástroj OpenOME, stejně jako jeho desktopová verze OME, byl vytvořen rovněž skupinou z Laboratoře Řízení Znalostí na Torontské Univerzitě. Neni přesně známo, kdy byla vydaná první verze nástroje, ale jeho poslední oficiální verze k tomuto dni byla zveřejněna v roce 2011. Vedoucím projektu byl Dr. Yijun Yu, který stejně jako tvůrce nástroje OME v daném období pracoval na Torontské Univerzitě.

3.1.2.1 Popis nástroje

Vzhledem k faktu, že OpenOME je založen na stejných principech jako tomu bylo v jeho desktopové verzi: OME, i tento nástroj pracuje s přístupy modelování aktérů a cílu. Návíc integruje několik vylepšení OME v podobě používání přístupů, které podporují modelování dalších aspektů, ovlivnující inženýrské požadavky ve vývoji softwaru, konceptuální modelování nebo jiných prostředků použitelných v prostředích pro editování grafů [6].

3.1.2.2 Technické prostředky

OpenOME je implementován jako plugin pro existující vývojová prostředí, jako je například Eclipse, Protége nebo Microsoft Visio. Plugin byl implementován pomocí programovacího jazyku Java. Práce s nástrojem je podporovaná na operačních systémech Windows, Linux a Mac OS.

3.1.2.3 Popis klíčových funkcionalit

Plugin podporuje vytváření modelů pomocí grafického editoru v totožné podobě, jako tomu je v prostředí OME 3.1.1.3. Plugin kromě grafického editoru navíc podporuje tvorbu na základě textových vstupů. Tyto vstupy jsou definované v rámci textového formátu, pojmenovaného Q7, který byl vytvořen samotnými autory. Plugin obsahuje vlastní parser, který s tímto formátem umí pracovat.

Uživatel má totožné možnosti manipulace nad jednotlivými elementy a sociálními závislostmi jako v případě nástroje OME. Tedy funkcionality pro změnu jména, smazání, pohyb objektů, změna velikosti, kopírování apod.

Dalšími klíčovými funkcionalitami OpenOME je předběžná a zpětná analýza dodržení notace jazyka i* nebo možnost vložení a výjmutí elementů do jiných, rozšiřitelných elementů [7].

3.1.3 TAOM4E

Autorem tohoto nástroje je skupina Software Inženýru z FBK, Centra pro Informační Technologie, Trento, Itálie. Demo aplikace bylo prezentováno na konferenci CAiSE 2006 v Lucemburku, která se konala v roce 2006. Poslení úpravy na nástroji k tomuto dni byly provedeny v roce 2010. Distribuce nástroje spadá pod licenci GPL v2. Je tedy možné nástroj volně využívat, kopírovat, měnit a rozšiřovat, stejně tak je možné využití v oblasti studia.

3.1.3.1 Popis nástroje

TAOM4E podporuje tvorbu modelů na základě agentů, řídící se metodologií Tropos, tedy metodologie, která užívá konceptu vzoru agenta po celou fázi procesu vývoje [8]. Při modelování a analýze softwarových požadavků se používají pojmy agent, cíl, úkol a sociální závislosti. Je to tedy metodologie v tomto ohledu totožná s metodologií i*. Byla navrhnuta s přihlédnutím na doporučení MDA. Předchůdcem tohoto nástroje je TAOM Modeler, z kterého si TAOM4E vypůjčil základní funkcionalitu a rozšířil ji, spolu s přidáním již zmíněné metodologie Tropos [9].

3.1.3.2 Technické prostředky

TAOM4E je implementován jako plugin pro vývojového prostředí Eclipse. Nejnovější verzi pluginu lze využít ve verzích Eclipse 3.6 a 3.7. Instalace je možná za pomocí správce aktualizací vývojového prostředí Eclipse. Plugin byl implementován v programovacím jazyce Java. Jelikož se jedná o plugin, je potřeba, aby uživatel měl nainstalované funkční vývojové prostředí Eclipse verze 3.7 nebo nižší, spolu s aktuálním JDK.

3.1.3.3 Popis klíčových funkcionalit

Plugin podporuje tvorbu modelů pomocí grafického editoru, kde jednotlivé prvky, které jsou k dispozici v panelu objektů, mohou být přidány na kreslící plochu kliknutím na zvolený element a přetazením na kreslící plochu. Nástroj nepodporuje tvorbu modelů na základě textových vstupů.

Uživateli jsou poskytnuty funkcionality pro přejmenování, změnu velikosti nebo barvy objektů. Rovněž je možné tyto objekty přesouvat, mazat, kopírovat a vkládat. Přístup k těmto příkazům je buďto v kontextovém okně nebo v panelu nástrojů. V diagramu mohou být použity různé částečné pohledy na model. Vazby mezi objekty mohou být taktéž přesouvány, kopírovány a vkládány a zatímco mohou být přeorganizovány, nemůže být jejich cesta, kterou vedou, zakřivena.

Další klíčové funkcionality, které nástroj nabizí je například operace kopírování a vkládání skupiny elementů, vrácení předešlé operace nebo její znovu vykonání [10].

3.1.4 piStar

Nástroj vznikl spoluprácí Federální univerzity v Pernambuco a Federální Venskovské univerzity v Pernambuco. Nástroj byl vyvíjen od roku 2017, kdy nejnovější úpravy byly provedeny v březnu roku 2018. Využivání nástroje nespadá pod žádní licence a je ho tedy možné volně využívat, šířit a přidávat další funkcionality.

3.1.4.1 Popis nástroje

piStar je modelovací nástroj, který odpovídá standartu iStar 2.0. Vytvořené modely mají vysokou vizuální věrnost, což tyto modely dělá vhodnými pro následovné použití v dokumentacích nebo na webových strankách. Díky volné licenci může být nástroj jakkoliv upraven. Vývojáři tedy mohou nástroj aplikovat jako základ pro vytvoření jejích vlastních aplikací se specifickými účely. Hlavní uplatnění nástroje je určeno pro výzkumné pracovníky a praktiky inženýrství požadavků, kteří běžně potřebují vytvářet a udržovat i* modely.

3.1.4.2 Technické prostředky

piStar je implementován jako webová aplikace pracující výhradně na straně klienta, tedy v prohlížeči. Žádné informace nejsou přenášeny na server. Stránka také nepracuje se soubory cookies, aby jejím uživatelům poskytovala maximální zabezpečení osobního soukromí. Vzhledem k faktu, že se jedná o webovou aplikaci, je třeba využívat aktuálních webových prohlížečů, jako jsou například Google Chrome nebo Mozzila Firefox. Aplikace byla implementovaná programovacím jazykem JavaScript, spolu s využitím HTML a CSS.

3.1.4.3 Popis klíčových funkcionalit

Webová aplikace poskytuje uživateli grafický editor, kde v panelu objektů jsou k dispozici jednotlivé prvky popsány jazykem i* 2. Operace kreslení aktérů a vnitřních prvků uživatel docílí kliknutím na příslušné tlačítko a následovně kliknutím na pozici v kreslící oblasti. Při kreslení

vazeb mezi objekty musí obdobně uživatel nějdříve kliknout na tlačítko, zvolit druh vazby a následovně kliknout na zdrojový a cílový objekt, mezi kterými má vazba vzniknout.

Nástroj nepodporuje vytváření modelů skrze textový editor. Je zde možnost manuální úpravy vyexportovaných modelů v textové podobě, ale vzhledem k možnosti poškození souboru se takový zásah nedoporučuje.

Uživatel má k dispozici funkcionality, které umožnují pohyb, mazaní a změnu jména jednotlivých elementů. Taktéž může uživatel přidat doplňkové informace pro konkrétní prvek formou dalších popisných vlastností. Vazby sociálních závislostí mohou být mazány, a ikdyž jsou modelovány jako přímo jdoucí čáry, lze vytvářet zakřivení kliknutím na bod vedené čáry a následovným posunem na požadovanou pozici.

Další specifické funkcionality, které nástroj poskytuje je možnost změny velikosti kreslící oblasti nebo změna pozadí aktérovy hranice [11].

3.2 Srovnání nástrojů a této práce

Při srovnávání existujících nástrojů jsem vzal v potaz několik kličových faktorů, které ovlivnují vhodnost využívání těchto nástrojů. Jednotlivé aspekty jsou rozděleny do dvou skupin, kde první popisuje vhodnost tohoto nástroje pro cílové modelování, zatímco druhá skupina poukazuje na jednotlivé rozdíly v nabízených funkcionalitách.

3.2.1 Vhodnost pro i* modelování

V rámci vytváření nového modelu dle notace jazyka i* můžeme hodnotit schopnosti nástroje v mnoha ohledek, ovlivňující kvalitu a korektnost výsledného modelu, viz. tabulka 2. Využívaná verze jazyka i* je rozhodně také rozhodovací faktor.

Primárně nas zajímají podporované pohledy modelu, stejně tak nás ale zajímá i kontrola správného využívání těchto pohledů. Kontrola může probíhat formou zpětné analýzy nebo již při samotném vytváření. Pro jednoduché rozlišení bude tato práce v porovnání označena názvem: "myStar".

3.2.2 Nabízené funkcionality

Dalším způsobem podle kterého můžeme posuzovat kvalitu jednotlivých nástrojů je nabízená funkcionalita pro práci s jednotlivými modely. Podstatné podporované funkcionality pro dříve zmíněné nástroje jsou uvedeny v tabulce 3. Rozdíly jsou většinou nepatrné, lišící se v přístupu pouze dle daného typu objektu.

Tabulka 2: Srovnání vhodnosti nástrojů (Zdroj: [12])

	OME	OpenOME	TAOM4E	piStar	myStar
Podpora modelového pohledu strategického odůvodnění	Ano	Ano	Ano	Ano	Ano
Podpora modelového pohledu strategických závislostí	Ano	Ano	Ano	Ano	Ano
Podpora tvorby modelů graficky	Ano	Ano	Ano	Ano	Ano
Podpora tvorby modelů textově	Ne	Ano	Ne	Ne	Ne
Kontrola modelového pohledu strategického odůvodnění	Ne	Ano	Ano	Ano	Ano
Kontrola modelového pohledu strategických závislostí	Ne	Ano	Ne	Ano	Ano
Možnost práce s oběma pohledy zároveň (Hybridní)	Ano	Ano	Ano	Ano	Ano

Tabulka 3: Poskytované funkcionality nástrojů (Zdroj: [12])

	OME	OpenOME	TAOM4E	piStar	myStar
Pohyb	✓	✓	✓	✓	✓
Přejmenování	✓	✓	✓	✓	✓
Mazání (Prvků/ Sociálních Závislostí)	✓/✓	✓/✓	✓/✗	✓/✓	✓/✓
Změna velikosti (Prvků/ Sociálních Závislostí)	✓/✓	✓/✗	✓/✗	✗/✗	✗/✗
Změna typu (Prvků/ Sociálních Závislostí)	✓/✓	✓/✗	✓/✗	✗/✗	✗/✗
Kopírování a vložení (Prvků/ Sociálních Závislostí)	✓/✓	✓/✓	✓/✓	✗/✗	✗/✗
Export a import	✓	✓	✓	✓	✓
Doplňkové atributy	✓	✓	✓	✓	✓

3.3 Zhodnocení

Přestože existuje velká řada nástrojů pro modelování cílů, válná většina je implementovaná jako plugin pro již existující vývojové prostředí nebo existující nástroje, které svou funkcionalitou pouze rozšiřují. Nepříznivým faktorem je i to, že jejich další vývoj většinou do dnešního dne už nepokračuje. Jednalo se totiž většinou o nástroje, které vznikaly na začátku 21. století. Jak bylo zmíněno v kapitole 2.1, nejnovějším standartem je v dnešní době iStar 2.0. Je tedy logické, že tyto nástroje můžeme použít pro tvorbu cílového modelování, ale nemůžeme očekávat nejak-

tuálnější přístup. Jedinou vyjímkou je webový nástroj piStar, jehož tvorba začala na začátku roku 2017. Stejně tak tato bakalářská práce byla navržena s důrazem na dodržování nejnovějšího standartu modelovacího jazyka i*.

V předešlých kapitolách byly uvedeny srovnávací tabulky, které zachycovali jednotlivé rozdíly porovnaných nástrojů spolu s touto prací. Důvodem, proč jednotlivé rozdíly byly tak nepatrné je, že prostor pro odlišné přístupy řešení podmínek jednotlivých standartů a pravidel, kterými se nástroj musí řidit, je velmi malý. Stejně tak u většiny grafických editoru, uživatel požaduje většinou stejnou řadu operací pro práci s graficky reprezentovanými modely.

Hlavní výhodou a specifikem této práce je její aktuální a korektní přistup pro modelování cílů dle nejaktuálnější notace, iStar 2.0, spolu s nabízenou funkcionalitou oddálení/ přiblížení a celkového posunu vykreslovací plochy, popsané v sekci 4.3.1.

4 Návrh a implementace webové aplikace

Jádrem celého nástroje je grafický editor, tvořený jednotlivými operacemi spolu s modelovací oblastí, který je přístupný uživateli prostřednictvím grafického rozhraní. Nástroj je realizován formou webové aplikace s veškerou funkcionalitou na straně klienta, tedy přímo v prohlížeči uživatele a to bez jakékoliv potřeby instalace nebo jiného nastavování. Nástroj je tedy možný využívat na všech standartních a nejnovějších internetových prohlížečích.

4.1 Knihovny třetích stran

V následujících kapitolách budou rozvedeny klíčové JavaScript knihovny třetích stran, které výrazně zjednodušili celkou implementaci nástroje. Kromě nich, byla využita rovněž knihovna Bootstrap, aktivně upravující vzhled webových stránek. Přináší sebou navíc vlastní CSS soubor, který může být modifikován. Taktéž v této práci využívám knihovnu lodash, upravující a přidávající možnosti práce s nativním JavaScriptem. Celkový seznam použitých knihoven je k dipozici v příloze C.

4.1.1 JointJS

Knihovna pro tvorbu diagramů JointJS dovoluje tvorbu plně interaktívních diagramových nástrojů s podporou všech moderních prohlížečů. Architektura knihovny odděluje modely grafu, elementů a vazeb od jejich samotného zobrazování na kreslící ploše. Přínosem je tak například možnost jednoduchého ukládání modelu, obsahující data všech prvků, na server. Samotná knihovna vznikla s využitím knihovny Backbone, která bude popsána následovně.

Výhodou využívaní této knihovny je i fakt, že se nijak nesnaží měnit směr vývoje zavedených technologií, které jsou pro webové vývojáře běžnou praxí. Můžeme tedy říci, že informace, které získame v rámci této konkrétní knihovny, mohou být užitečné i v jiných případech [13].

API poskytované touto knihovnou definuje tři základní komponenty: *joint*, *Vectorizer* a *Geometry*. Modul *joint* obsahuje veškeré myslitelné objekty, které vývojář potřebuje pro tvorbu diagramu. Modul *Vectorizer* je rozměrově malá pomocná SVG knihovna. Důvodem proč ji tvůrci nezakomponovali do modulu *joint*, je ten, že knihovnu v případě potřeby lze využívat samostatně. Stejný případ je modul *Geometry*, další pomocná knihovna vestavěná do JointJS. Její funkcí je poskytnutí mnoha užitečných geometrických operací [14].

Existuje také řada pluginu přidávající různe sety objektů a funkcionalit, většinou jsou tyto pluginy bohužel licencované a nelze je tedy volně využívat.

4.1.2 Backbone

Při práci na webové aplikaci se neobejdeme bez JavaScriptu. Ačkoliv svázání dat aplikace s jednotlivými DOM elementy je možné a nepřínaší žádné problémy, vedou se jisté spory, proč by se tento přístup neměl praktikovat. DOM by obecně měl být pohled na data. Takže jednotlivé vlastnosti těchto DOM elementů by měly sloužit jako metadata a né obsahovat data z modelu. Navíc takto nelze dosáhnout struktury objektů s identifikátory, což na základě funkcionality aplikace může být kritické.

Backbone reprezentuje data jako modely, které mohou být vytvořeny, ověřený, zničeny nebo uloženy. Kdykoliv aktivita na uživatelském rozhraní způsobí například změnu atributu konkrétního modelu, tento model vyvolá událost. Všechny pohledy, které vyzobrazují daný model pak můžou být informovány o této změne a adekvátně zareagovat, tedy překreslit model na základě nových informací [15].

Hlavní snahou knihovny Backbone je snaha o objevení minimální sady primitivních prvků strukturace dat (modelů a kolekcí) a uživatelského rozhraní (pohledů a URL), které jsou obecně užitečné při vytváření webových aplikací, které pracují s JavaScriptem.

Tím nejdůležitějším přínosem knihovny je schopnost pomoct vývojáří oddělit byznys logiku od uživatelského rozhraní.

4.1.3 jQuery

jQuery je JavaScript knihovna, která se snaží dosáhnout psaní méně obsáhlych zdrojových kodů, zatímco výsledek zůstane stejný. Hlavním účelem je tedy zjednodušit programování v JavaScriptu pro webovou stránku.

jQuery se zaměřuje na spoustu standartních úloh, jejichž realizování v běžném nativním JavaScriptu zabírá velké množství řádků kodu a slučuje tyto operace do metod, které mohou být volány v rámcí jediného příkazu/řádku. Stejně tak zjednodušuje mnoho komplikovaných JavaScript funkcí. Například volání AJAX metod nebo DOM manipulace. Ikdyž existuje celá řada JavaScript frameworku, jQuery je jedním z nejpopulárnejším a také nejjednodušeji rozšířitelným o další funkce [16].

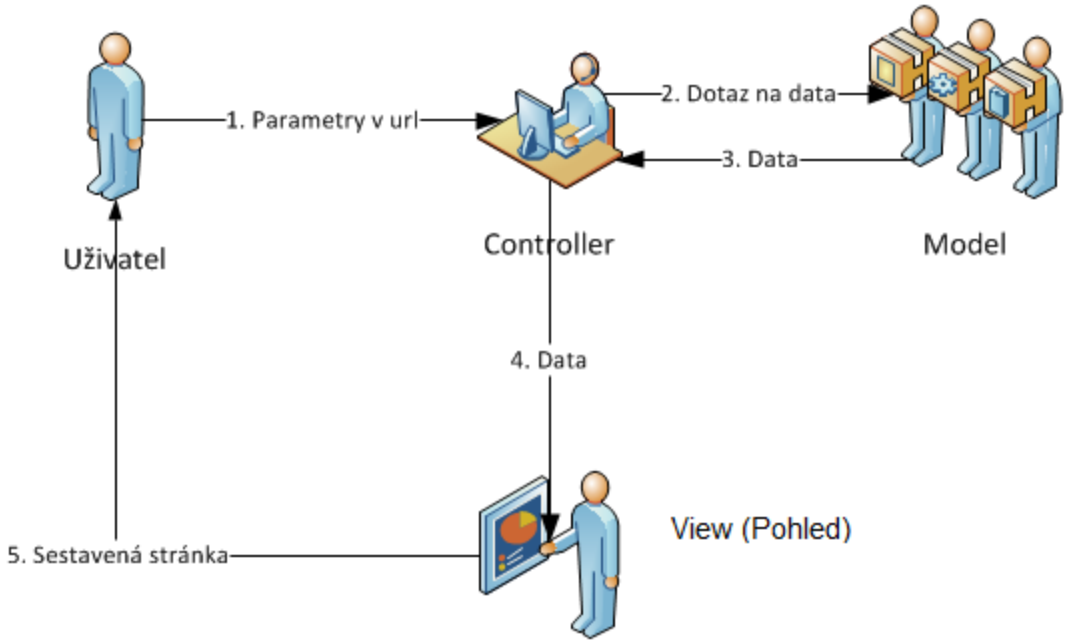
Základní podoba jQuerry podporuje následující funkcionality:

- HTML/DOM/CSS manipulaci
- HTML metody událostí
- Efekty a animace
- AJAX a další utility

4.2 Architektura aplikace

Při vytváření nástrojů s grafickými editory se nejčastěji váže návrhový vzor MVC, který slouží k rozdělení datového modelu aplikace, uživatelského rozhraní a řídící logiku. Všechny tyto kompotenty jsou realizovány nezávysle, důsledkem čehož jakékoliv provedené změny na jedné komponentě ovlivní ostatní jen minimálně [18]. Ačkoliv je MVC chápan primárně jako návrhový vzor, jeho zásah do architektury aplikací je rozsáhlejší než tomu bývá u jiných vzorů, takže se někdy označuje za architektonický vzor [17]. Ukázku jednotlivých části vzoru MVC a jejich interakci můžeme vidět na obrázku 15. Popis jednotlivých části vzoru:

- **Model** obsahuje logiku, spolu se vším co do ní spadá. Všechny jeho funkce zahrnují přijetí parametrů, dat nebo příkazů a následovné operace nad nimi, popřípadě jejich vrácení v pozměněné podobě.
- **View** zastává funkci zobrazení výstupů uživateli. Převádí tedy data, které obsahuje model, do podoby vhodné pro vykreslení uživateli.
- **Controller** (řadič) je jakési propojení, jehož funkce spočívá ve schopnosti reagovat na událostí, nejčasteji vycházející z interakce uživatele s rozhraním a zajištění, že se vykonají adekvátní změny v modelu nebo pohledu.

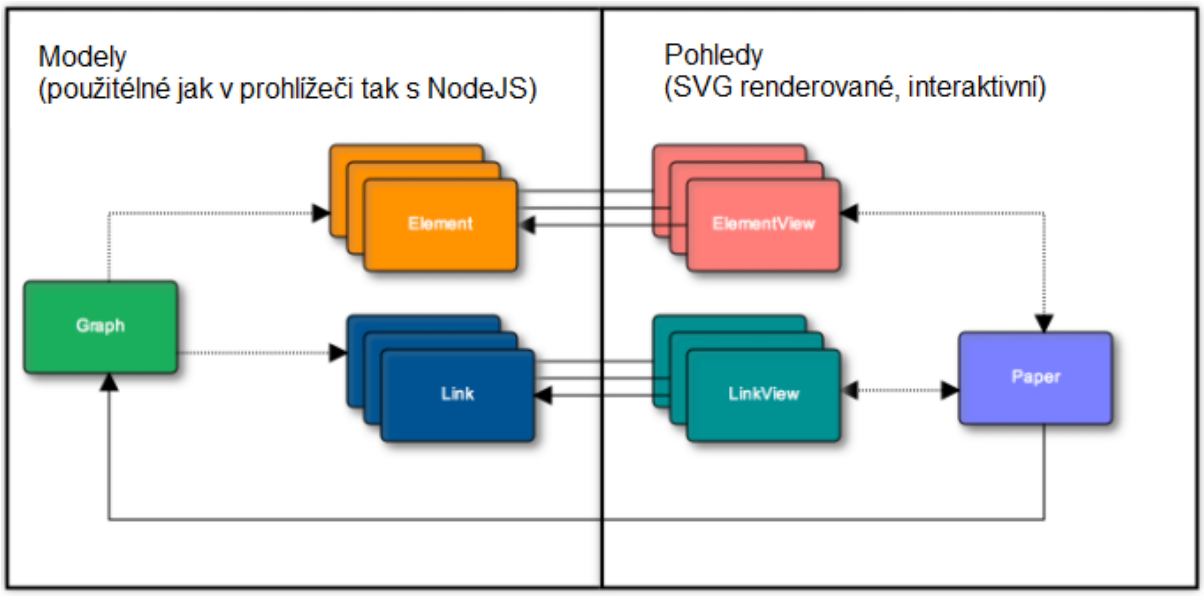


Obrázek 15: Ukázka komponent MVC (Zdroj: [18])

Přestože skutečná architektůra této aplikace pracuje reálně pouze s modely a pohledy, práce kontroleru je simulovaná v rámci grafického rozhraní s kterým pracuje uživatel a existující interní

funkce knihovny JointJS, která definuje pro každy prvek view funkci, která vytváří propojení mezi modelem a pohledem. Při interakci s pohledem proběhne notifikace modelu, který na základě konkrétních změn provede korekci/úpravu dat korespondujících objektů a zpětně informuje o své změne pohled, což zapříčiní překreslení. Konkrétní propojení jednotlivých částí můžeme vidět na obrázku 16.

Model tedy pracuje s *grafem*, který se dělí na elementy a vazby, jsou to tedy JavaScript objekty, ve kterých jsou uchovávany data o konkrétních prvcích. Druhou částí je takzvaný *papír*, na němž se realizují všechny operace spjaté s pohledem, který se zase rozděluje na elementView a linkView. Poskytuje tedy separátní výchozí rozložení pro obě tyto kategorie. *Papír* je tedy reprezentací části pohledu v návrhovém vzoru, zatímco *graf* obsahuje všechny data existujících prvků.



Obrázek 16: Propojení Modelu s Pohledem (Zdroj: [13])

Při inicializaci aplikace se vytvoří nová instance *papíru*, které do konstruktoru přiřadíme oblast pro vykreslování, v tomto případě konkrétní DIV naší HTML stránky, dále vybranou instanci *grafu*, tedy model a několik dalších parametrů, jako je například maximální povolená šířka a výška papíru, skutečné vykreslovací plochy. Poté je již možné editovat obsah zvlášť modelu a prvků, které se mají vykreslovat uživateli.

4.2.1 Struktura objektů

Vzhledem k faktu, že aplikace nepoužívá soubory cookies, ani žádné propojení na server, všechny data jsou ukládány a mazány přes JavaScript objekty. Jednotlivé objekty o sobě udržují všechny podstatné informace v podobě atributů. Obecně v aplikaci rozděluji objekty na elementy, tedy aktéry, prvky úmyslů a předměty sociálních závislostí a na vztahy, které spojují dle logiky notace iStar 2.0 elementy mezi sebou. Jsou to asociační vazby aktérů, vazby propojení sociální závislosti a propojení prvků úmyslů.

Při definování obou skupin využívám možnostni rozšíření existujících generických objektů, zastávající roli společného předka, který zastupuje funkci kostry, se kterou můžeme pracovat - přidávat a upravovat atributy. Rozdíly mezi elementy a vztahy nejsou až tak rozsáhle. Ty, které odlišné jsou, určují způsob vyskreslování uživateli na kreslící plochu.

4.2.1.1 Elementy

Při vytvoření objektu je mu vygenerováno a přiřazeno id, které funguje jako jednoznačný identifikátor, podle kterého ho lze později dohledat nebo reagovat na jeho změny ať už v modelu nebo pohledu.

Prvním důležitým atributem, který určuje jeho podporované chování na kreslící ploše je *markup*, který obsahuje reference na pomocné funkce knihoven *Geometry* a *Vectorizer*, zmíněny v sekci 4.1.1. Rovněž v něm definuji příslušné SVG atributy, které budou součástí grafické podoby objektu. Každý takový element může být tvořen celou řadou SVG objektů. Při vytváření mých objektů standartně obsahuje definice SVG tvar, či více tvarů a text.

Dalším atributem je *type*, který nám zase pomáhá rozlišovat mezi konkrétními prvky. Následuje atribut *size*, kde uvádíme výšku a šířku prvku. Jedná se o rozměry, které mu budou přiřazeny v době vzniku, jestliže v atributu *markup* je použita na konkrétní vykreslovanou část funkce knihovny *Geometry* "scalable", může se tento atribut v průběhu existence objektu změnit.

Poté následuje atribut *attrs*, který obsahuje všechny ostatní podstatné informace, spolu s vnitřními atributy jednotlivých SVG, které jsme si nadefinovali v *markup*. Pro tyto SVG prvky, což jsou konkrétní objekty, jako je čtverec, kruh apod., můžeme editovat barvu, kterou budou vyplněny, barvu a šířku obtáhnutí objektu, jejich výšku a šířku relativní k velikosti celého objektu. Jendotlivé atributy, které můžeme využívat záleží od typu SVG objektu. U SVG objektu text, tedy například pracujeme s atributem velikostí, typem a barvou písma, zatímco u SVG objektu kruh, je potřeba uvést refereční středový bod a poloměr.

Navíc u všech elementů sleduji jednotlivé vazby relevatní k typu daného objektu, spolu s vazbami sociálních závislostí, díky čehož je relativně jednoduché zajištění korektní funkcionality a logiky propojování jednotlivých objektů na bázi notace iStar 2.0. Speciální atribut, který využívají aktéři, zachycuje stav jejich hranic, která může být viditelná nebo schovaná, což logicky ovlivňuje viditelnost i vnitřních elementů aktéra. Při samotném vytváření objektu se zasílá konkrétní

pozice, kde na vykreslovácí ploše má být objekt vložen, což je realizováno přes údálosti kliknutí na vykreslovací plochu.

4.2.1.2 Vazby

Obdobně jako u elementů i při vytváření vazeb je vygenerováno a přiřazeno id, které jezdnoznačně rozlišuje jednotlivé instance vazeb.

Pro splnění stejné funkce, jako tomu je u elementů, je definován atribut *type*, který rozlišuje jednotlivé druhy vazeb. Následují dva nepovinné atributy, které upravují styl vykreslování vazby. Atribut *router*, obsahující také interní algorytmus, který se snaží detekovat kolize vedení vazby s již existujícími objekty a vytváří tak ohyby pro vedení vazby. Druhým atributem *connector*, který přímo navazuje na *router* a modifikuje způsob vykreslení oblasti ohybu vazby. Zde je možnost ponechat původní ostrý úhel nebo použít zakulacení. Pakliže jsou tyto atributy vynechány, používá se standartní přímočaré vykreslení, nejkratší cesty mezi cílovým a zdrojovým objektem.

Stejně jako u elementů pracujeme s atributem *attrs*, který ale obsahuje informace o cílovém a zdrojovém objektu, mezi kterými vzniká vazba, spolu s informacemi o textu, ten však není povinný. Jak u zdroje, tak u cíle můžeme upravit tvar znaku pomocí SVG cesty. Jak bylo řečeno, SVG poskytuje základní tvary nejčastěji používané při běžné práci. Pomocí cesty lze realně zrekonstruovat nebo vytvořit jakýkoliv tvar, ikdyž je to značně pracnější. Dále je možné upravit barvu výplně a ohraničení, pakliže v námi nadefinované cestě spojujeme jednotlivé body do ploch místo úseček.

Vznik vazby doprovází dodatečné přidání identifikátorů elementů do korespondující části *attrs* mezi kterými vazba vzniká. Před vytvořením vazby nejdřív dochází k validaci typu objektů, aby byly dodrženy pravidla definované notací iStar 2.0. Konkrétně se jedná o limitaci existence jednotlivých propojení poposivaných v příslušných sekcích kapitoly 2. Vazba navíc nemůže vzniknout mezi cílovým a zdrojovým objektem, kde už jedna vazba existuje, a to ani jiného typu.

4.2.2 Vizualizace objektů

Pro vizualiazci jednotlivých objektů modelu je potřeba vytvořený prvek přiřadit do grafu, čímž dáváme najevo, že tento konkrétní prvek má být vykreslen, jak jde vidět na obrazku 16. Pro práci s objekty na kreslícím plátnu je poskytnuta celá řada událostí, reagující například na kliknutí, pohyb, posun nebo přejetí přes buňku nebo prázdné místo na kreslící ploše. Za buňku považujeme element nebo vazbu. Skrze tyto události můžeme zachytit i související aktivity, jako je připojení nebo odpojení vazby.

Operace nad prvky nebo samotnou vykreslovací plochou vyvolávají změny atributů těchto prvku, kde konkrétní řadič zajistí přenos dat do modelu.

JointJS knihovna pracuje i s konceptem vnoření objektů ve vztahu rodič a potomek, vztah, který je zachycen v modelu pomocí atributu *embeds* obsahující identifikátory jednotlivých po-

tomků a atribut *parent* obsahující identifikátor svého rodiče, kterého je součástí. Této vazby v aplikaci využívám při práci s aktéry a jejich hranicemi. Při vykreslování je zachyceno chování tohoto vnoření objektů specifickým způsobem. Pohyb rodiče vyvolává relativní pohyb potomků, smazání rodiče způsobuje smazaní i všech jeho potomků.

Změny vyvolané interakcí uživatele s kreslicí plochou tedy aktivně ovlivňují model obsahující data aplikace. K existujicím událostem máme možnost reagovat a využivat doplňkové události, reagující na změny pozice, úhlu, nebo velikosti objektů, popřípadě konkrétních atributů nebo přídání potomka rodiči.

Odebrání objektu z *grafu* neznamená jeho destrukci, objekt stále existuje, dokud není odstraněn interní logikou správy objektů JavaScriptu. Není již nadále ale vykreslován a není možné s ním pro uživatele tedy pracovat ani ho zpětně přidat do *grafu*.

4.3 Klíčové funkcionality

Jednotlivé funkcionality, které tento nástroj nabízí a byly avizovány v kapitole 4 budou nyní v následujících sekcích popsány a vysvětleny jejich interakce s pohledem a modelem.

4.3.1 Posun vykreslovací plochy a funkce přiblížení/oddálení

Aplikace neposkytuje možnost změny velikosti vykreslovací plochy. Navíc je vykreslovací plocha omezená velikostí DIV sekce HTML stránky, kde je umístěna. Je však logické předpokládat, že uživatel bude vytvářet modely, které nepůjde v této omezené ploše zachytit celé.

Proto aplikace využívá funkcionality nabízene knihovnou třetí strany: SVG-pan-zoom, která je volně šířitelná pod licencí, splňující požadavky zmínění jejího autora. Autor je implicitně zmíněn v zdrojovém souboru. Odkaz na knihovnu můžeme najít v sekci C příloh.

Poskytnuta funkcionalita skrze API přidává operace přiblížení/oddálení. Jejich funkcí je změna velikosti, tedy měřítka vykreslovaných objektů, které v aplikaci využívám pro až dvojnásobné přiblížení nebo oddálení, spolu s vyresetováním měřitka na původní hodnoty. Sekundární poskytnutá funkcionalita řeší problém velikosti vykreslovácí plochy, a to funkci "pan"- hromadného posunu vykreslovaných objektů a přepočítaní souřadnic i do mínusových hodnot plochy pro vykreslování. Uživatel si tedy kdykoliv může oddálit pohled, přesunout a přiblížit na konkrétní část modelu. Efekt na pohled je řízen přes zmíněné operace, ovlivnující *papír*. Ten zase v rámcí svých událostí uvědomí model o změnách pozice jednotlivých objektů.

4.3.2 Přidání poznámky

Jedná se o jednoduchý objekt ve tvaru obdelníku, reprezentující prostor pro přidání poznámky k konkrétním prvkům modelu nebo celého modelovaného systému. Jak uživatel využije této mož-

nosti je již na něm. Tato možnost byla přidaná s přihlédnutím na možnost rozsáhlého rozložení modelu a důsledné ztráty jednoduchého pochopení zachycených cílů modelu.

Jednotlivé objekty nesou svůj název a logika vazeb a propojení je dostatečně vysvětlena a němělo by tedy docházet k nedorozuměním. Funkcionalita tedy uživateli poskytuje pouze možnost preference přístupu zachycení logiky modelovaného systému.

4.3.3 Editace textu objektu

Jak název této funkcionality napovídá, jejím účelem je možnost změny textu objektu. Při vytvoření a vložení objektu do *grafu* mu je přiřazen popis, který pouze identifikuje jeko typ. Uživatel pak má možnost přejmenovat prvek na konkrétní entitu, s kterou si přeje pracovat v konxtextu systému, jež má model popisovat.

Funkce se aktivuje při kliknutí na konkrétní prvek. Uživateli se zobrazí malý formulář s textovým polem. Následovně může změnu potvrdit, nebo operaci zrušit. Delká použitého názvu neni omezená, ale pakliže přesáhne oblast velikosti objektu, kterého je součástí, nebude jeho celá podoba viditelná.

4.3.4 Přepínaní viditelnosti hranic aktéra

Sekce 2.8 byla zaměřena na popis různých druhů pohledů na model popsané jazykem i*. Aplikace byla navržena, aby podporovala všechny tři typy pohledů, a proto implementuje možnost zobrazení/schování hranic aktérů, obsahující prvky úmyslů a jejich vzájemne propojení.

Operace spočívá v změně atributu viditelnosti aktéra, všech jeho potomků a souvisejících sociálních závislostí, které byly vedeny od těchto potomků. Vešketé objekty jsou stále součásti grafu, ale tento atribut upozorňuje *papír*, že si nepřejeme je vykreslovat ani reagovat na jakékoliv události, které běžne aplikace zachycuje. Uživateli stačí vybrat tuto operaci a poté kliknout na konkrétního aktéra, jeho hranice bude schovaná. Opětovným kliknutím na aktéra se zpětně všechny ovlivněné prvky začnou znovu vykreslovat a reagovat na události.

Nedochází tedy k mazaní (destrukci) ani odstranění těchto objektů z *grafu*, protože by si uživatel mohl změnu pohledu rozmyslet a nemusí tak opakovaně vytvářet všechny, jinak ztracene, prvky úmyslů a jejích propojení.

4.3.5 Mazání objektů

Pokuď si přejeme trvale smazat část modelu, můžeme využít funkcionality mazaní. Jako u většiny funkcionalit nástroje se standartně zvolí tato operace a poté uživatel klikne na vybraný objekt určený pro smazání. Zde přichází na scénu podstata důležitosti uchovávaní indentifikátorů všech možných vztahů pro každý element. Pokud by byl cíl smazaní objekt bez jakýkoliv vazeb na ostatní prvky v *grafu*, nevznikl by žádný problém, ale vzhledem k požadavkům notace

iStar 2.0 by taková situace nastala jen zřídka. Pracujeme s aktéry, kteří mezi sebou využívají asociační spoje, dále obsahují zmíněné prvky úmyslů, které zase mají vlastní logiku propojování a v poslední řadě jsou zde sociální závislosti, které mohou propojovat jendotlivé aktéry, ale i konkrétní prvky úmyslů v rámci různých rodičů (aktérů).

Proto je třeba dosažení korektního způsobu kaskádového mazaní objektů. V praxi aplikace využívá následující postupy, dle typu objektu:

- **Aktér a jeho vazby:** Při mazání asociačního spojení dochází pouze kromě samotného smazání vazby k odstranění identifikátoru smazané vazby z atributu vazeb u cílového a zdrojového aktéra. Smazání samotného aktéra je už složitější. Neexistuje údálost, která by byla schopná zachytit chystající se operaci smazání konkrétního prvku - aplikace nemá jak zjistit, na který prvek uživatel klikne. Existuje ale naštěstí událost, zachycující provedenou akci smazaní "on:remove", skrze kterou jsme se schopni dostat k smazanému prvku ještě předtím, než je reálně smazán i z modelu.

  Můžeme si tedy získat atributy, jako jsou konkrétní asociační vazby nebo vazby sociálních závislostí, které se k němu vztahovaly. Stejně tak máme k dispozici seznam prvků úmyslů, které při své existenci obsahoval. Totožným způsobem se pak lehce dostaneme k jendotlivým propojením a sociálním zavislostem, kterých byly prvky úmyslů zůčastněny. Poté už jen stačí postupně procházet tyto kolekce objektů a mazat je.

- **Prvky úmyslů a jeho vazby:** Při mazání samostatných vazeb, spojující prvky úmyslů, stačí odstranit referenci mazané vazby z atributu v cílovém a zdrojovém objektu.

  Smazaní samotného prvku úmyslu znamená vytažení kolekce vztahů propojení mezi ostatními prvky úmyslů a samozřejmě i příslušných sociálních závislostí. Obdobně jak při mazání aktéru pak procházíme tyto kolekce referencí, na jejichž základě mažeme příslušné objekty.

- **Sociální závislosti:** Při mazání, ať už spojů jdoucích od zdroje nebo cíle sociální závislosti, popřípadě předmětu závislosti, aplikace reaguje totožně, protože při smazání jakékoliv části sociální závislosti (sekce 2.6), existence ostatních částí postráda logiku.

  Jak bylo zmíněno, vazby obsahují reference na zdrojový a cílový objekt. Stejně tak předmět závislosti, jako každý jiný element obsahuje atribut, zachycující reference na něj připojené vazby. V obodu připadech tedy není problém vyhledat a odstranit tyto reference u poskytujícího a závisejícího (popřípadě jejích prvků) aktéra. Následuje už jen odstranění vazeb/předmětu závislostí a operace je dokončena.

4.3.6 Export a import

Nástroj je využíván pro tvorbu a editaci modelů, ale vzhledem k tomu, že pracuje pouze na straně klienta (uživatele), by po vypnutí prohlížeče, ve kterém webová aplikace pracuje o všechny data uživatel přišel a pokud by si přál se zpětně k tomuto modelu vrátit a dále ho rozšiřovat nebo editovat, musel by ho začít kreslit znova od začátku.

Z tohoto důvodu aplikace podporuje možnost vyexportování modelu v textové podobě formátu JSON, který si uživatel může uložit na jakékoliv externí zařízení. Zpětně pak uživatel může přistoupit ke svému modelu skrze funkcionalitu importu zdrojového souboru, kde aplikace dekóduje a zpětně zrekonstruje model do podoby při jeho exportu. Tuto operaci může provádět opakovaně a může ji využívat odkudkoliv - není limitován na zařízení, kde byl model původně vytvořen.

4.3.7 Metodika modelování

Kromě primární funkcionality modelování cílů dle notace standartu jazyka i*, iStar 2.0, nástroj nabízí možnost tvorby klasických případů užití dle notace UML, které popisují chování systému z hlediska pohledu uživatele. Nabízená funkcionalita pro tvorbu a prací s případy užití je totožná s primárním zaměřením nástroje. Mimo specifickou funkcionalitu pro modely iStar 2.0, jako je přepínaní viditelnosti hranic aktérů.

4.3.8 Nové kreslící plátno a ukázky

Funkce vytvoření nového souboru spočívá pouze ve vyčištění stavajících objektů, které byly obsaženy v modelu. Ukázka je přítomná jako šablona pro rychle otestování aplikace, bez ztráty času vytvářením ukázkové situace.

5 Závěr

Cílem této bakalářské práce bylo navrhnout a vytvořit webovou aplikaci dovolující tvorbu a editaci diagramů pro modelování cílů dle notace modelovacího jazyka i*, které jsou základem pro popis cílů a úkolů navrhovaných systémů.

Práci lze rozdělit na dva úseky. Sekundarní, který se zaměřoval na popis a pochopení celé notace modelovacího jazyka i*, a to včetně jeho vývoje do dnešního dne a srovnání existujících nástrojů, které rovněž podporují práci na bázi tohoto jazyka. Jednotlivé nástroje, spolu s výsledkem této práce, jsou porovnány v dvou relevatních ohledech, zachycující vhodnost na základě specifických atributů, ovlivňujících tuto podporu a nástroji nabízené funkcionality.

Primárním zaměřením pak byla samotná implementace a návrh celé aplikace. Hlavní náplní je popis použité architektury, struktura objektů a celkové řešení implementace. Dále jsou posány klíčové funkcionality grafického editoru, které jsou poskytnuty uživateli pro jeho tvorbu a editaci modelů. Rovněž jsou popsány nejdůležitější knihovny třetích stran, bez kterých by vytvoření aplikace bylo mnohonásobně těžší.

Při samotné implementaci jsem nenarazil na žádné rozsáhlejší problemy, a to hlavně díky kvalitně zpracovaném API, které knihovny třetích stran pro vytvoření grafického editoru poskytovaly. Jakéhokoliv budoucího rozšiřovaní nabízené funkcionality tohoto nástroje by tedy bylo jednoduché dosáhnout. Užitečným rozšířením nástroje by kupříkladu mohlo být přidání podpory tvorby dalších typů diagramů.

Na závěr bych také rád uvedl, že téma bakalářské práce považuji za velice přínosné, jelikož vedlo k mému zdokonalení ve vytváření webových aplikací, tedy práce s JavaScriptem, HTML a CSS. Dalším přinosem bylo přisvojení správného využívání a začlenění nabízených knihoven třetích stran a dalších webových technologií.

Literatura

[1] Fabiano Dalpiaz, Xavier Franch a Jennifer Horkoff. *iStar 2.0 Language Guide* [online]. [cit. 2018-04-15]. Dostupné z: https://arxiv.org/pdf/1605.07767v3.pdf

[2] Goal modeling. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Goal_modeling

[3] Dunstan Thomas. *Why Use UML?* [online]. [cit. 2018-04-16]. Dostupné z: http://dthomas-software.co.uk/resources/frequently-asked-questions/why-use-uml-2/

[4] *Organization Modelling Environment* [online]. [cit. 2018-04-16]. Dostupné z: http://www.cs.toronto.edu/km/ome/

[5] i* wiki. *OME* [online]. [cit. 2018-04-16]. Dostupné z: http://istarwiki.org/tiki-index.php?page=OME

[6] OpenOME. *OpenOME, an open-source requirements engineering tool* [online]. [cit. 2018-04-16]. Dostupné z: https://se.cs.toronto.edu/trac/ome/wiki

[7] i* wiki. *OpenOME* [online]. [cit. 2018-04-16]. Dostupné z: http://istarwiki.org/tiki-index.php?page=OpenOME

[8] *Tool for Agent Oriented Modeling* [online]. [cit. 2018-04-16]. Dostupné z: http://selab.fbk.eu/taom/

[9] *Tropos* [online]. [cit. 2018-04-16]. Dostupné z: http://www.troposproject.eu/

[10] i* wiki. *TAOM4E* [online]. [cit. 2018-04-16]. Dostupné z: http://istarwiki.org/tiki-index.php?page=TAOM4E

[11] i* wiki. *piStar* [online]. [cit. 2018-04-16]. Dostupné z: http://istarwiki.org/tiki-index.php?page=piStar

[12] i* wiki. *Comparing the i* Tools* [online]. [cit. 2018-04-17]. Dostupné z: http://istarwiki.org/tiki-index.php?page=Comparing+the+i%2A+Tools&structure=i%2A+Wiki+Home

[13] JointJS. *Getting Started* [online]. [cit. 2018-04-18]. Dostupné z: https://resources.jointjs.com/tutorial

[14] JointJS. *Joint API* [online]. [cit. 2018-04-18]. Dostupné z: https://resources.jointjs.com/docs/jointjs/v2.0/joint.html

[15] BackboneJS. *Getting Started* [online]. [cit. 2018-04-18]. Dostupné z: http://backbonejs.org/#Getting-started

[16] w3schools. *jQuery Introduction* [online]. [cit. 2018-04-18]. Dostupné z: `https://www.w3schools.com/jquery/jquery_intro.asp`

[17] Model-view-controller. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-18]. Dostupné z: `https://en.wikipedia.org/wiki/Goal_modeling`

[18] David Čapka. *1. díl - Popis MVC architektury* [online]. [cit. 2018-04-18]. Dostupné z: `https://tinyurl.com/ydgyntf2`

Seznam příloh

A Tabulka překladů

Důvodem vzniku této překladové tabulky, byla potřeba nadefinování českého překladu pro pochopení významu těchto pojmů, protože pro ně neexistuje formální podoba překladu.

Tabulka 4: Tabulka překladů s referencemi na definice

Původní název	Český překlad	Význam
Depender	Závisející	Aktér, který závisí na nečem, (Dependum) co mu má být poskytnuto.
DependerElement	Prvek Závisejícího	Prvek úmyslu umístěn v hranici závisejícího aktéra odkud tato sociální závislost vzniká
Dependum	Předmět Závislosti	Prvek úmyslu, který představuje kontext a význam samotné závislosti
Dependee	Poskytovatel	Aktér, jež by měl Předmět Závislosti poskytnout.
DependeeElement	Prvek Poskytovatele	Prvek úmyslu, který vysvětluje jakým způsobem chce tento aktér Předmět Závislosti poskytnout.

B Uživatelská příručka

Uživatelské příručka detailně popisuje interakci uživatele s nástrojem a je k dispozici na přiloženém optickém médiu, konkrétně v adresáři "manual".

Tabulka 5: Struktura optického média

Adresář	Obsah
\src	Projekt webové aplikace se všemi zdrojovými soubory
\doc	Dokumentace práce v PDF formátu
\manual	Uživatelská příručka v PDF formátu

C Knihovny třetích stran

- JointJS - `https://www.jointjs.com/`
- Backbone - `http://backbonejs.org/`
- Lodash - `https://lodash.com/`
- SVG-Pan-Zoom - `https://github.com/ariutta/svg-pan-zoom`
- jQuery - `https://jquery.com/`
- Font Awesome - `https://fontawesome.com/v4.7.0/`
- Bootstrap - `https://getbootstrap.com/docs/4.0/getting-started/download/`