Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Srovnání metodik pro oblast multi-agentových systémů

Bakalářská práce

Autor: Eliška Hegrová
Studijní obor: Informační management

Vedoucí práce: doc. RNDr. Petr Tučník, Ph.D.

Hradec Králové 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne 15.8.2022 Eliška Hegrová

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Petru Tučníkovi za cenné rady, věcné připomínky a vstřícnost při konzultacích.

Anotace

Bakalářská práce se zaměřuje na srovnání vybraných metodik pro oblast multiagentových systémů. Seznamuje čtenáře s teorií agentů a jejich zasazením do multiagentních systémů. V práci jsou popsány metodiky ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MESSAGE a Prométheus, které byly vybrány vzhledem k jejich četnému využívání v rámci vývoje multiagentních systémů. U každé metodiky je znázorněna jejich vnitřní struktura, obecný popis fungování a jejich slabé a silné stránky. V návaznosti na tento teoretický popis obsahuje práce případové studie z různých oblastí vývoje pro srovnání vybraných metodik z praktického hlediska. V závěru práce dochází k posouzení kladů a záporů každé metodiky. V závěru práce se rozebírá potencionální vývoj této oblasti do budoucna.

Klíčová slova: metodologie, metodika, multi-agentní systém, ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MESSAGE, Prométheus

Annotation

Title: Comparison of Multi-agent Based Methodologies

This Bachelor Thesis is focused on the comparison of selected methodologies for the field of multi-agent systems. It introduces the reader to the theory of agents and implementation of agent-oriented methodologies in multi-agent systems context. The Thesis describes the methodologies ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MESSAGE and Prométheus, which were chosen due to their numerous applications in the development of multi-agent systems. For each methodology is provided, its internal structure, a general description of how it works, and its strengths and weaknesses. Following this theoretical description, the Thesis contains case studies from various areas of development, used for comparison of the selected methodologies from a practical point of view. The pros and cons of each methodology are assessed in concluding discussion. The conclusion provides overview of results and further potential development of this area in the future.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

ABMS	Agent-Based Modelling and Simulation
ACL	Agent Communication Language
ADELFE	Soubor nástrojů pro navrhování softwaru s novými funkcionalitami emergence
AMAS	Autonomní multiagentní systémy
AUP	Agile Unified Process
BDI	Belife-desire-intentions
BMC	Business Modeling Canvas
BSPL	Blindingly Simple Protocol Language
GAIA	Geometrical Analysis for Interactive Aid
HAPL	Hierarchical Agent Protocol Notation
IAF	Agent Framework
IDK	INGENIAS Devepment Kit
KQML	Knowledge Query Manipulation Language
MAL	Multi-Agent Learning
MAS	Multiagentní systém
MASE	Multiagent Systems Engineering
MDE	Model-Driven Engineering
PASSI	Process for Agent Societies Specification and Implementation
RAD	Rapid Application Development
RUP	Rational Unified Process
SPEM	Software Process Engineering Metamodel

1 Úvod

Vzhledem k současnému technologickému pokroku se aplikace založené na teorii multiagentních systémů stávají standardem v různých oblastech technologického vývoje. Setkáváme se s nimi v telekomunikacích, logistice, ve výrobě, ale i ve zdravotnictví. Multiagentní systémy jako takové často pomáhají při vývoji nových distribuovaných systémů, které jsou zaměřené přímo na člověka ve smyslu zjednodušené komunikace. Usnadňují člověku bez předchozích technických znalostí práci s potřebnou technologií. V návaznosti na tento fakt se tak multiagentním systémům dostává velké popularity nejen v praxi, ale i na akademické půdě.

Tato práce zahrnuje v teoretické části obecný popis agenta a jeho začlenění do různých typů prostředí. Stejně jako prostředí, i agenti mohou mít různé podoby. Ty nejběžnější typy agentů jako jsou reaktivní, deliberativní a sociální jsou popsány v práci. Následuje obecný úvod do problematiky multiagentních systémů a jakým způsobem jsou agenti začleněni do multiagentní komunity. Vzhledem k návaznosti na systémové inženýrství a zasazení agenta do prostředí je v práci uvedena kapitola, která stručně pojednává o modelování agentů a jejich dorozumívání. Tato bakalářská práce je primárně zaměřena na popsání metodik. Proto je v textu stručné vymezení pojmu metodika a metodologie. Následuje popis konkrétních metodik ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MESSAGE a Prométheus. Mnohé z uvedených metodik přímo vyházejí či se inspirují metodikami RUP, KGR a AUP. Proto jsou RUP, KGR a AUP popsány v teoretické části pro bližší nahlédnutí do vývoje. V praktické části jsou uvedeny konkrétní případy využití vypsaných metodik vyjma metodik AUP a RUP, které slouží pouze pro teoretické porovnání vývoje. Na základě popisu a uvedených případových studií je v rámci práce vytvořeno přehledné srovnání kladů a záporů všech metodik. Na konci teoretické části se nachází tabulka s přehledem funkcionalit daných metodik.

Multiagentní systémy jsou vzhledem k rychlosti aktuálního vývoje velmi probíranou oblastí. Smart cities, automatizované výrobní linky, monitoring továren, robotika, virtuální asistence, kyberbezpečnost atd. To vše dokáže usnadnit život jedincům ale i firmám. Multiagentní systémy jsou tak velmi obsáhnou oblastí, která se vyvíjí závratnou rychlostí a rozhodně stojí za bližší prozkoumání.

2 Cíle práce

Cílem práce je zhodnotit silné a slabé stránky metodik ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MAS-CommonKADS a MESSAGE v kontextu praktické použitelnosti v rámci konkrétních případových studií. Výstupem práce je přenést uvedené výsledky do tabulky s vybranými výzkumnými otázkami. Uvedená tabulka bude sloužit jako rozcestník při hledání vhodné metodiky pro zkoumanou oblast.

3 Teoretická část

V teoretické části této práce jsou vysvětleny pojmy agent a multiagentní systém. Jsou zde popsány nejběžnější typy agentů a typy prostředí, do kteterých může být agent zasazen. V této části se popisuje, jakým způsobem se agent začleňuje do multiagentní komunity. Popisují se zde vybrané metodiky ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MAS-CommonKADS a MESSAGE z teoretického hlediska. V návaznosti na tento výčet jsou zde popsány i metodiky RUP a AUP, ze kterých mnohé z uvedených metodik vycházejí.

3.1 Agent

Multiagentní systémy jsou sestaveny ze společenství různých entit neboli agentů. Kubík (2004) popisuje agenta takto: „*Agent je entita zkonstruovaná za účelem kontinuálně a do jisté míry autonomně plnit své cíle v adekvátním prostředí na základě vnímání prostřednictvím senzorů a prováděním akcí prostřednictvím aktuátorů. Agent přitom ovlivňuje podmínky v prostředí tak, aby se přibližoval k plnění cílů*“.

Z této definice přímo vyplývají vlastnosti, které jsou pro agenty stěžejní. První z nich je autonomnost. Agent je entita, která je schopna proaktivní činnosti mířené na řešení úloh. K tomuto řešení nemusí potřebovat součinnost okolních agentů. Pakliže tuto pomoc od ostatních entit potřebuje, měl by být schopen vstupovat do komunity za účelem získání informací. Dobrovolnost vstupu a výstupu z komunity je také stěžejní.

Další vlastností, která definuje agenta, je jeho reaktivita. Agent, který existuje v prostoru je totiž aktivován okolní událostí, na kterou musí reagovat. Tato reakce se odehrává v reálném čase. Při reagování na události by měl mít agent na paměti své dlouhodobé cíle. Tato schopnost se nazývá intencionalita a zahrnuje i organizaci chování agenta k dosažení těchto cílů. Agent musí využívat svého úsudku v hodnocení událostí a jednoznačně formulovat vlastní plány.

Tím, že jsou agenti součástí společenství neboli komunity, musí mít osvojenou schopnost sociálního chování. Toto sociální chování je jim prospěšné pro naplňování vlastních cílů. Pro správné fungování ve skupině si tak každý agent musí osvojit sdružování do určitého druhu koalic, ze kterých mu bude plynout prospěch. Do této schopnosti náleží i

udržování informací o ostatních agentech ve společenství, se kterými přijde agent do kontaktu.

Pro co nejlepší funkčnost a jednoduchost by měl agent mít pouze nezbytné vlastnosti, které ho budou přímo a jednoznačně definovat. To samé platí i o vnímání prostředí. Agent vnímá prvky okolí, na které je vybaven díky svým snímačům. Získané informace tak může zpracovat a následně převzít iniciativu (Prýmek, 2008). Na událost zareagovat pomocí aktuátorů. Získané údaje o prostředí si ukládá, vytváří z nich posloupný záznam, který mu slouží pro rozhodování v budoucnu. Tímto způsobem se vzdělává ze svého okolí. Formálně tak lze agenta popsat pomocí jeho funkcí neboli tabulkou (Russell, 2020). Označení tabulka může být lehce zavádějící. Jedná se o abstraktní matematický popis agenta.

Mařík a spol. ve své práci (2001) rozdělují architekturu agentů do čtyř vnitřních modulů a jednoho vnějšího. Prvním je vstupní rozhraní agenta. Ten vnímá okolí pomocí svých senzorů. Senzorem se rozumí softwarové i hardwarové vybavení, které umožňuje agentovi sbírat data ze svého okolí pro následnou analýzu.

Druhý vnitřní model je model zastoupení paměti a prostředků pro zpracování informací, které agent získal. Tyto informace si pak třídí a zpracovávají, takže z nich vzniká nová znalost prostředí. Prýmek ve své práci (2008) uvádí, že má znalost prostředí dva prvky, které je nutno rozlišovat. Znalost pasivního prostředí agenta a znalost o okolních agentech. Pasivní znalost, jak název vypovídá, je vědění o okolních předmětech (rozměry, umístění, rozloha celé místnosti atd.). Znalost o okolních agentech v sobě zahrnuje pokročilejší stupeň inteligence. Agent by měl být schopný predikovat chování jiného agenta, jeho potencionální vlastnosti a jak by se dal daný agent využít.

Třetím vnitřním modelem je modul řízení systému v reálném čase. V tomto modulu se nastavují cíle agenta, průběžně se kontroluje jejich plnění, odvozování pravidel atd. V této fázi se určuje, zda se skutečně jedná o racionálního agenta.

Při prvotním seznámení s agenty je patrné, že má agent sadu nástrojů, které mu pomáhají přistupovat k prostředí a tím i dosahovat svého cíle. Dalším modulem je tak model aktuátorů. Aktuátor je nástroj mechanického rázu. Často se jedná o mechatronickou soustavu, jako jsou například mechanické paže, které odebírají vzorky půdy nebo metodu pro zápis na disk. Díky těmto nástrojům může agent ovlivňovat okolní prostředí, ale i vlastní působení.

Tyto moduly ukončují vnitřní architekturu agenta. Podstatnou částí pro fungování je ale i vnější prostředí. Vnější prostředí sice není součástí agenta v pravém slova smyslu, ale často

obsahuje řadu komunikačních kanálů mezi agenty. Jedná se o část, která je navázaná určitou formou na agenta, a proto nelze opomenout.

Při modelování agentů si musí tvůrce položit otázku, co konkrétně odlišuje dobře namodelovaného agenta od špatně namodelovaného. Odpověď se mu dostane po analýze výsledků agenta. Tedy jeho výkonu. Vzhledem k různým typům agentů se nemůže předepsat jeden způsob měření úspěšnosti. Můžou se ovšem definovat obecné konvence, které jsou vhodné při vyhodnocování dodržovat.

Vyhodnocování míry úspěšnosti se nedoporučuje nechávat v rukách samotného agenta. Mělo by se pro účely vyhodnocování zavést tzv. objektivní míra úspěšnosti, kdy agenta pozoruje například programátor.

Dalším předpokladem pro určení míry úspěšnosti je i komplexnost pozorování. Lze to vidět na příkladu agenta, který zprostředkovává těžbu uhlí. Ten je schopný zpracovat za určitý čas určité množství horniny. Zjištění toho, kolik je agent schopný vytěžit za jednu hodinu, je pro posuzování stěžejní. Do hodnocení je třeba zahrnout i další aspekty, jako je například spotřeba energie, poruchovost atd.

Otázkou také je, v jaký čas míru výkonu posuzovat. U některých agentů se pozorování provádí na začátku jejich práce, u jiných zase v průběhu či na konci. Každý agent je jiný, takže některý vykonává zadanou práci stále konstantně, jinému klesá postupem času účinnost. Je důležité se při tvorbě zaměřit na to, jak vývojář chce, aby vypadalo prostředí agenta po vykonané práci spíše, než jaké očekává chování.

3.1.1 Prostředí agenta

Pakliže se na prostředí nahlíží jako na část modelu, definuje se agent jako prvek agentního systému a prostředí jako subsystém (Zbořil, 2006). Z pohledu agenta je tak prostředí jedním prvkem. Tento prvek může nabývat různých stavů prostředí, která mohou být různě komplikovaná. Prvním typem je prostředí plně pozorovatelné. Takové prostředí není pro agenta žádnou výzvou, jelikož se zde neskrývají žádná úskalí v mapování okolí. V praxi je samozřejmě běžnější prostředí částečně pozorovatelné, kde bude mít agent na začátku bádání pouze omezené informace.

Při modelování systému je žádoucí agenta vybavit všemi senzory a aktuátory tak, aby mohl efektivně plnit svůj zadaný cíl. V mnoho případech, a zvláště v reálném světě, ovšem nelze předpovídat v určité oblasti všechny možné formy okolí. Agent se tak dostává

do prostředí, které je buď deterministické nebo stochastické. Podle Přibylové (2015) je deterministické prostředí takové, které nemění spontánně svůj stav a je vázáno stanovenými vztahy. Stav prostředí není tak náhodný – určuje ho spousta proměnných, jako jsou dané podmínky a určené vztahy. Vztáhnuto na prostředí agenta je jeho budoucí stav dán stavem aktuálním nebo případnou akcí prováděnou agentem.

Stochastické prostředí je oproti tomu takové, které je tvořeno nahodilými jevy. Tyto jevy nelze předpovědět na základě předešlé návaznosti, ale lze je předvídat pomocí statistiky.

Mnoho autorů uvádí členění na epizody nebo sekvence. U prostředí epizodického je z názvu patrné, že člení určitý celek na epizody. V tomto případě prostředí, ale z pohledu agenta, a ne přímo děje okolí (jaro, léto). Epizody posluhují jako druh vzpomínky v porovnání s člověkem. Ke vzpomínkám se lze vracet a brát z nich ponaučení a zkušenosti. Podobně je tomu tak i u epizod agenta. Každá epizoda obsahuje různý počet jednotlivých akcí, které agent vykonal. Akce jsou závislé na epizodě, ve které je agent vykonal, ale epizody na sobě závislé nejsou. U sekvenčního zpracování agent neuchovává oddělené vzpomínky, a tak musí myslet kvůli návaznosti stále „dopředu“ (Brom, 2013).

U všech těchto případů se pozorovalo prostředí samostatně a zkoumalo se, jak ho vnímá agent. Při dosahování cílů ale agent do prostředí zasahuje, a tím ho modifikuje. Na toto navazuje i členění na dynamické a statické prostředí. Dynamické prostředí se totiž mění i bez zásahu agenta a statické pouze po zásahu. U statického prostředí se tak nemusí entita orientovat v čase a při provádění úkonu zůstává okolní stav stále stejný, takže agent nemusí neustále pozorovat své okolí. Mnoho zdrojů uvádí i prostředí semi-dynamické, kdy se prostředí jako takové nemění, takže je statického rázu, ale výkonost agenta se zvyšuje.

Při tvorbě agentů a jejich prostředí lze narazit na oblast, kdy bude prostředí definováno jako diskrétní nebo spojité. Z hlediska pravděpodobnosti nabývá náhodná veličina hodnot diskrétního nebo statického rozdělení. Diskrétní rozdělení může nabývat mnoha hodnot, které představují definované body na dané ose. Tyto body jsou izolované. Oproti tomu spojité rozdělení může nabývat všech hodnot na určeném intervalu. Ve skutečnosti tedy až nekonečno (Přibylová, 2015). Když se aplikuje tento statistický model na prostředí agentů, je patrné, že je diskrétní prostředí takové, které má omezený počet vnímání a akcí. Příkladem mohou být šachy. Každá figurka má na své pozici omezený počet tahů, které může v danou chvíli udělat. Oproti tomu prostředí spojité může nabývat nekonečně mnoha hodnot. Příkladem může být řízení automobilu. Automobil může sám o sobě nabývat různých rychlostí a lokací stejně tak jako ostatní automobily na silnici.

Obecně se považuje za nejjednodušší prostředí pro zpracovávání plně pozorovatelné, statické, epizodické, deterministické, diskrétní a s jedním agentem. Takováto kombinace reálně v praxi nastane s nulovou pravděpodobností. Často se tak naráží na dynamické a nepřístupné prostředí, se kterým se musí „poprat“ více než jeden agent. Velký „zádrhel“ je také určení, zda je prostředí opravdu deterministické. Toto určit je často velmi sporné, tudíž se automaticky nahlíží na prostředí jako nedeterministické.

Definování prostředí agenta je stěžejní pro další manipulaci s návrhem. V objektově orientovaném programování se lze setkat se dvěma významy prostředí. Jedním z nich je prostředí pro správu agentů. To zajišťuje agentovi jeho hladký běh procesů. Například komunikaci mezi ostatními agenty atd.

Pod pojmem prostředí si lze představit i specializovaného agenta typu Environment. Tento agent obsahuje mechanismy pro běh prostředí jako takového. Tím, že koordinuje okolní prostředí agentů ve výsledku spouští i jejich operace a předává jim informace. Tyto programové věci jsou ale nad rámec, proto se o nich práce zmiňuje pouze okrajově.

3.1.2 Reaktivní agent

Jak je již z názvu patrné, tento agent je založen na principu zachycení podmětu a následné reakce. Postrádá tak komplexnější zvažování svých aktivit – plánování. Tento agent nemá prostředky pro uchovávání aktuálního nebo minulého stavu okolního prostředí. Na zachycené podměty reaguje bezprostředně, čemuž odpovídá i jeho architektura – senzory a aktuátory jsou napojeny napřímo (Cimr, 2016).

Při popisování agentů se uvádí, že by měl být agent schopný určitého racionálního či inteligentního chování. Na první pohled ale tento agent vypadá jako neinteligentní entita. Jeho inteligence spočívá v tom, že i přes neznalost okolního prostředí dokáže vykonávat správná rozhodnutí. Dobrým příkladem je robotický vysavač, který pro fungování nepotřebuje obsluhu člověka. Vysavač je schopný rozpoznávat překážky ve své trajektorií jako jsou stěny či dřevěná noha židle, a těmto překážkám přizpůsobuje svůj směr. Tato schopnost je značně pozoruhodná vzhledem k tomu, že vysavač nezná danou místnost jako celek.

3.1.3 Deliberativní agent

Deliberativní, nebo někdy také intencionální agent, má v porovnání s reaktivním agentem komplikovanější strukturu jednání. Po zachycení podmětu z prostředí si agent sestaví několik možných způsobů, jak na daný podmět reagovat. Možnosti si určuje dle databáze svých předchozích znalostí a vnitřních stavů. Pro rozhodování pak využívá principů umělé inteligence. Dle Chainbi (1998) je okolní prostředí zaznamenáváno pomocí symbolické reprezentace. Se symboly se pak dále manipuluje v rámci rozhodovacích stromů či modální logiky.

Deliberativní agenti se obecně udávají jako nejefektivnější v dobře specifikovaném prostředí, které není proměnlivé a cíl je jasně vymezen.

3.1.4 Sociální agent

Dalším pomyslným evolučním krůčkem v oblasti vývoje agentů je agent sociální. Deliberativní agent zdokonaloval reaktivního přidáním vnitřní struktury pro plánování. Dokázal nově zaznamenávat stavy okolí. Problém ovšem nastával, když byl do modelovaného prostředí zasazen další agent, který byl takové vnímán jako součást prostředí, a ne jako další entita schopna spolupráce. Jakým způsobem ale namodelovat agenta, který je schopný spolupracovat s ostatními, i když nemají přímý společný cíl? Konstrukce agenta, který by byl schopný ve vnitřní struktuře obsahovat všechny případné převzaté informace od ostatních agentů by byla až zbytečně složitá. Zároveň by se nedodržela zásada abstrakce od nepodstatných detailů. V nepředvídatelném prostředí ani není reálné dopředu počítat se všemi typy problémů. Řešením je zanechávání značek v prostředí. Na základě těchto jednoduchých lokálních reakcí vzniká racionální chování všech entit ve společenství (Prýmek, 2008).

3.2 Začlenění agenta do multiagentní komunity

Každý agent, který existuje v určité multiagentní komunitě, musí dodržovat stanovená pravidla skupiny. Pravidla jsou zde nastavena tak, aby bylo vzájemné sdílení informací a nabízené služby přístupné pro každého příchozího za stejných podmínek. Každý agent musí

plnit své závazky a uzpůsobovat svou činnost v souladu s globálním cílem dané skupiny. Tento princip se nazývá kooperace. Kubík (2004) jej definuje takto: „*Kooperace je řízenou formou koordinace s účelovým uspořádáním agentů ve skupině s cílem dosáhnout společného řešení problému nebo konfliktu. Společenství může být řízeno centrálně nebo naprosto decentralizovaně s autonomními prvky řídícími se svou funkcí užitečnosti, která následně ovlivňuje strategie jejich chování. Každý agent má přesně určenou roli, kterou musí plnit a také vztahy s ostatními agenty k tomu, aby bylo dosaženo globálního cíle.*“ Kooperativní síť tak sdružuje agenty, kteří mají společné cíle. Tomuto sdružení se v multiagentním světě přezdívá koalice. Každá koalice by měla mít definovaná pravidla o svém vzniku i zániku a určené mechanismy pro řešení případných konfliktů mezi agenty. Předpokládá se, že do koalice vstoupí pouze takový agent, který je schopný splňovat pravidla skupiny a dostávat svým závazkům. Pakliže by agent tyto pravidla neplnil, můžou na něj být uvaleny sankce v podobě odebrání zdrojů, informační izolace, odmítání komunikace atd.

V multiagentním světě ale mohou vznikat i vztahy typu koordinace a konfliktu. Koordinace v tomto případě znamená, že existují takoví agenti, kteří mají ve společenství cíle odlišné než skupina. Dokážou ovšem s ostatními spolupracovat, aby dosáhli svého záměru efektivní cestou. Pro tuto spolupráci potřebují agenti vybudovat vzájemnou dohodu fungování. Tato dohoda začíná ve fázi, kdy si všichni agenti uvědomí, že dosažení cíle není možné právě bez zmíněné spolupráce. Následně si agenti domluví podmínky fungování a pomocí daného komunikačního jazyka sestaví efektivní plán pro naplnění cílů. V principu kooperace často dochází v tzv. dražbám, kdy agent nabízí a poptává potřebné zdroje.

Vztah typu konflikt vzniká za předpokladu, kdy je užitek pro vstupujícího agenta mizivý až nulový za přítomnosti agenta spolupracujícího. Tyto konflikty vznikají například v rámci exklusivních zdrojů (Ipser, 2008). Agent, který daný zdroj potřebuje ke svému úspěšnému splnění cíle, bude v konfliktu s agentem, který ohrožuje přístup k danému zdroji. Řešením daného konfliktu se tak může jevit forma útěku, kdy by ustoupili oba agenti či radikálnější varianta útoku. Silnější agent zničí svého oponenta, což není vhodné pro zachování efektivního společenství. Další variantou by byl vstup do subsumpční architektury, kdy je jeden z agentů podmaněn druhým (Ipser, 2008).

Agenty tedy můžeme dělit podle cílů, jak bylo popsáno výše. Při fungování ve společenství jsou ale tři věci, které musí mít agenti společné. Těmito společnými oblastmi je vnímání pojmů, pravidla vzájemné komunikace a jazyk (Netrvalová, 2010).

Na počátku komunikace mezi agenty stál jazyk KQML neboli Knowledge Query Manipulation Language. Jedná se o velmi obecnou koncepci, která je nezávislá díky své multifunkčnosti na konkrétním nasazením. Lze si na něm proto ukázat základy komunikace mezi agenty. Mařík (2001) přirovnal členění KQML k vrstvám ISO/OSI, které se používají v počítačových sítích. Na první vrstvě je fyzická úroveň, kde probíhá přenos pomocí bytů. Druhou vrstvou je vrstva transparentní, která zprostředkovává kódování. Následuje úroveň komunikační architektury. Tato úroveň zodpovídá za předávání zpráv, například na základě volání procedur atd. Čtvrtá úroveň je úroveň ACL neboli Agent Communication Language. Tato vrstva uchovává citlivé informace o adresátu zprávy, příjemci, ale také jazyk, ve kterém je zpráva napsána. Poslední vrstvou komunikačního modelu je úroveň, která uchovává konkrétní obsah zprávy. Tento obsah je již dekódován použitým jazykem, takže je agentovi přímo čitelný. Koncepce KQML tak plní funkci určitého obalu. Tento obal tak obsahuje údaje potřebné k doručení zprávy a za jakým účelem je poslána a předána k vyřízení. Může tak obsahovat různé per formativy jako ask-if, tell či archive. Mezi novější jazyky, které pro komunikaci využívají síť protokolů, si můžeme uvést HAPN či BSPL (Chopra, 2020). Každý jazyk je specifikován určitou sémantikou, syntaxí a abecedou. Ve výsledku kopíruje koncept řečového aktu. Tento řečový akt se dělí do tří úrovní podle složitosti na syntaktickou, sémantickou a pragmatickou úroveň. V první syntaktické úrovni je daný problém, se kterým se agent musí potýkat jednoduchý, takže můžeme syntaxi snadno vymezit. U sémantické úrovně se agent potýká se složitějším problémem k řešení, takže si koncipuje vlastní znalostní ontologie. Na třetí pragmatické úrovni je agent vystaven problému vyšší složitosti. Hodnotí, s kým má vlastně o problému hovořit, jak takového agenta ve společenství najít, a jak navázat komunikaci. Při těchto aktech se rozeznává několik typů dialogů, které agenti při komunikaci využívají. Agent může použít dialog dotazování. Ptá se na takovém místě, kde očekává, že by danou informaci mohl získat. Při jednoduchých problémech a malém počtu agentů ve společenství tato cesta není náročná. Dalším typem dialogu je vyjednávání. Agent nebo skupina agentů vyjednávají o podmínkách spolupráce tak, aby byly všichni ve skupině spokojení. Dochází ke konzultaci míry sdílení informací a využívání společných prostředků. Zajímavým typem dialektu je dialekt eristický. Tento druh argumentace je expresivního rázu a může tak připomínat hádku. Účelem je v tomto případě získat agentovu pomoc při plnění úkolů za každou cenu.

3.3 Multiagentní systémy

„Multiagentní systémy jsou zvláštním typem distribuovaných inteligentních systémů, ve kterých autonomní agenti obývají svět bez globální kontroly nebo globálně konzistentních znalostí. Na multiagentní systém tak můžeme pohlížet jako na společnost jedinců, kteří se vzájemně ovlivňují výměnou znalostí a interakcí, aby dosáhli vlastních zájmů či zájmů skupiny.“ Takto uvádí multiagentní systémy Oprea ve své práci (2004). Jedná se rozhodně o rozsáhlou definici, která nicméně obsahuje všechny důležité prvky definice multiagentních systémů neboli MAS. Každý agent v tomto společenství disponuje pouze omezenou informovaností a schopnostmi pro řešení daného problému. Vzhledem k tomu, že jsou agenti uspořádáni do daného společenství, neexistuje žádný řídící prvek, který by všechny agenty koordinoval. Je zde absence globálního řízení celého systému. Důležitým aspektem multiagentních systémů je i jejich celková decentralizace. V průběhu vývoje byly systémy tak složité, že jejich tvorba a udržování vyžadovalo paralelní zpracování dat a procesů. Musí tak fungovat ve složitých, a hlavně dynamických doménách. Při zdokonalování těchto systémů se tak řešila a stále řeší otázka, jak musí být agenti integrováni, aby zvládali koordinovat své aktivity a řešit složité problémy. Kromě těchto specifik se očekává, že se MAS vyrovná nejen s nepředvídatelnými změnami v okolí, ale i s vnitřními poruchami. Další podmínkou MAS je tak určitá míra odolnosti. V tomto případě se za odolnost považuje schopnost vykonávat a poskytovat služby a plnit úkoly navzdory všem změnám. Agenti i společenství tak rozpoznají změny v okolní struktuře a přizpůsobí se jim. Jeden z mechanismů, který nám tuto odolnost zajišťuje, je dynamická rekonfigurace (Inna Vistbakka, 2021).

S pohledem do historie zjistíme, že jsou počátky tvorby multiagentních systémů úzce spjaty s oborem distribuované umělé inteligence. Brooks se o toto téma zajímal již v 80. letech. V roce 1987 začal propagovat nový přístup k umělé inteligenci zvaný Nouvelle, který později blíže popsal ve své knize (Brooks, 1989). Klasická umělá inteligence je koncipovaná jako soustava algoritmů, které reagují s okolním světem nepřímo pomocí klávesnice či monitoru. To přišlo Brooksovi nedostatečné pro využití v praxi, a tak se snažil ve svých akademických pracích o převedení umělé inteligence do fyzické podoby pomocí robotů. Tito roboti tak mohli v reálném světě vnímat a reagovat na své prostředí a přizpůsobovat se mu. Popsal tak jako první princip reaktivního agenta (Copeland, 2007). Na konci 20. století vyvstala při tvorbě modulárních systémů otázka, jakým způsobem posunout vývoj těchto

systémů, a jak zvýšit autonomní jednání jednotlivých aktérů v rámci jedné architektury. Dalším žádoucím úsekem vývoje bylo i celkové zvýšení flexibility systému a volná integrace jednotlivých modulů pomocí zasílání zpráv (Zbořil, 2004). Na základě těchto úvah a celkového zkoumání v této oblasti se uskutečnilo i nespočet konferencí. Mezi významnější bychom mohli zařadit IJCAI workshop v Sydney na téma: Theoretical and Practical Design of Rational Agents.

Postupem času se multiagentní systémy začleňovaly do všech odvětví vědy. Přes robotiku až po zdravotnické systémy. Zdokonalování těchto systémů ovšem neustalo a stále na akademické a firemní půdě dochází ke zdokonalování procesů.

Pro správné fungování celého multiagentního systému je stěžejní v prvotní fázi nastavit globální cíle. Určují se funkční a nefunkční cíle, kterých by měl systém za svého fungování dosáhnout. Tyto cíle je vhodné rozdělit na podmnožiny, které můžeme hierarchicky uspořádat. Podmnožiny cílů pak můžeme delegovat na agenty, takže s nimi manipulujeme, jako by byli nezávislé. Při velkému objemu agentů v jednom systému můžeme rozdělit i agenty do skupin podle jejich schopností pro přehlednější manipulaci.

3.4 Modelování

Agentově orientované modelování je jednou z výpočetních forem vědeckého zkoumání a zobrazování systémů. Softwarové inženýrství se využívá na různé oblasti – přes modelování biologických systémů po robotiku. Není tak překvapením, že se tyto systémy odlišují ve svém uvažování, autonomností a inteligencí. V literatuře má multiagentní modelovaní s autonomními entitami různé názvy. Nejvíce hledanému vyjádření odpovídá Agent-Based Modelling and Simulation neboli ABMS. Jedná se o agentově orientovaný přístup, který má základ v dynamice, multiagentních systémech a robotice. Z názvu je patrná důležitost simulace. Při simulaci modelu je snaha napodobit stavy nebo skutečnosti z reálného světa. Často se modeluje pouze oblast pozorování, která souvisí se zkoumaným problémem, aby model odstínil od nepotřebných detailů (Bittner, 2002). Pro otestování konzistentnosti a stability modelu se využívá experimentů. V modelu se provede určitá cílená změna a sleduje se, jak se tato změna projeví. Porovnává se změněný systém se systémem izolovaným (systém před změnou).

Uvádějí se dva přístupy pro modelování agentních systémů: bottom-up a top-down (Husáková, 2013). Podle bottom-up přístupu se model tvoří od detailu k celku. Tím, že se prochází přes mikro úroveň na makro úroveň je tento přístup ideální pro tvorbu multiagentních systémů. Může tak dojít z jednoduchého plánování agentů k emergenci principu, který je uveden výše. Detailnější modelování s sebou nese předpoklad vyšší výpočetní náročnosti.

Aby bylo modelování přehledné a zahrnovalo všechny potřebné aspekty, musí se oblasti zájmu rozdělit podle způsobu, jakým se oblast zkoumá. K tomuto účelu slouží rozsáhlá škála diagramů. Nejdůležitější budou uvedeny v rámci dalších kapitol.

Správné multiagentní systémy by měly být adaptabilní ve smyslu uvažování nepředvídatelné události v rychle se měnícím prostředí. Společenství je sice složeno z inteligentních autonomních agentů, ale často se naráží na hranice ve struktuře. Nové již silné adaptační schopnosti pro MAS jsou tak nejen novinkou, ale i výzvou v programování. V návaznosti na tuto výzvu vzniká teorie AMAS, tedy autonomní multiagentní systémy. Tato teorie neříká, že bychom měli kódovat globální funkci systému uvnitř agentů, ale že tato silná adaptabilnost pochází z globálního chování různých činitelů. Výsledkem by tak měla být snaha vytvořit multiagentní systém, který má vlastní organizaci, kde agenti sledují svůj vlastní cíl, ale zároveň udržují kooperativní vztahy s okolními agenty. Podle (Chainbi, 1998) je AMAS charakterizován těmito boty: je ponořen do prostředí, skládá se ze vzájemně závislých agentů, každý agent vykonává dílčí funkci a organizace agentů za běhu umožňuje systému realizovat emergentní funkci.

Systém schopný dynamicky měnit svoje chování a učit se z okolních akcí představuje velký potenciál při řešení problémů, které jsou nové. U nového typu se neví, jak budou reagovat ostatní činitelé v prostředí. To dodává systému na složitosti.

V návaznosti na tento problém vznikl MAL neboli Multi-Agent Learning. MAL umožňuje navrhnout pravidla, díky kterým agenti využívají dynamiku okolního prostředí ve svůj prospěch, aby se mu přizpůsobili. Pakliže do prostředí zasahuje více agentů najednou, je pro entitu obtížnější udělat rozhodnutí. Okolní agenti se totiž nemusí řídit pravidly prostředí, takže mohou být nedeterministickými (Julian, 2019).

3.5 Metodologie

Současný výzkum v oboru MAS si klade otázky, jak musí agenti vzájemně spolupracovat, aby mohli koordinovat svoje aktivity a řešit složité problémy. Řešení tohoto problému s sebou nese spoustu abstrakcí a problémů s vývojem a designem ve srovnání s tradičními přístupy k vývoji softwaru. Výzkum se tak zaměřuje na návrh vhodných koordinačních mechanismů pro řízení společenství agentů. V předešlé kapitole byl popsán význam modelování a programování těchto agentů.

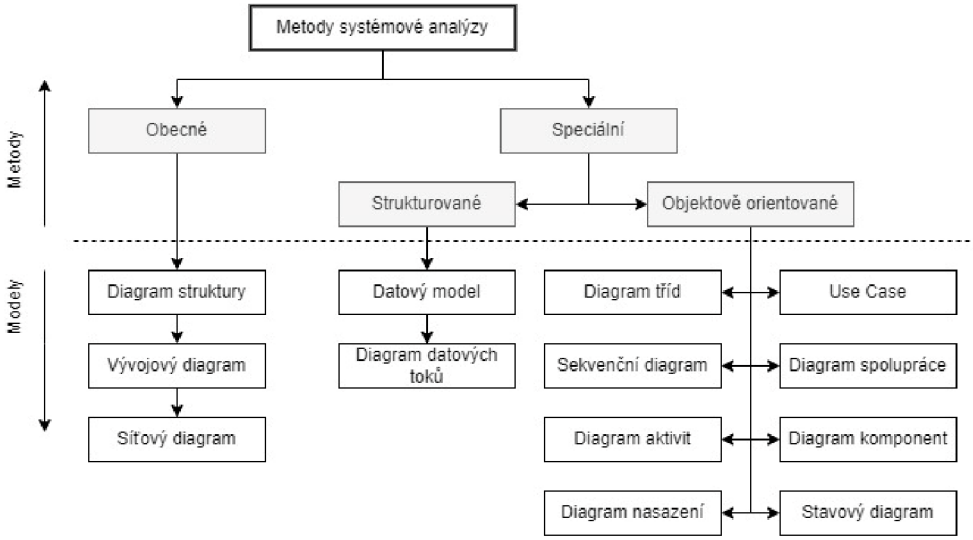
Metodologie hlídá, aby byly pokryty všechny důležité aspekty při tvorbě těchto softwarů. Jedná se o celou vědní disciplínu, která se zabývá souhrnem metod určité vědní oblasti a jejich tvorbou. Ochrana (2009) definuje metodologii v návaznosti na metodu takto: „*Metoda je nástrojem ke zkoumání daného výzkumného předmětu. Je to způsob a aplikace postupu, tak abychom dosáhli stanovený výzkumný cíl. Použití metody při vědeckém zkoumání předpokládá znát postup, jak metodu použít. Tento postup má rysy záměrnosti (vztahuje se k výzkumnému cíli) a systematičnosti (metoda je uplatňovaná v rámci teoreticky zdůvodněného postupu). Východiska ke zdůvodnění postupu dává metodologie. Metodologie má klíčové místo pro zaměření vědeckého zkoumání a pro volbu vědeckých výzkumných metod. Metodologie je tak vědní disciplína, která se zabývá metodami, jejich tvorbou a aplikací.*“ V rámci multiagentních systémů tak metodologie hlídá vývoj všech nezbytných procesů, které jsou v různých fázích.

Metodologie v rámci multi-agentních systémů dělí na agentově orientované, metodologie, které vycházejí z požadavků na programové vybavení, metodologie, které vycházejí ze znalostí inženýrství a metodologie, které se inspirovali konkrétní platformou.

3.6 Metodika

V rámci vědecké práce se využívá i pojem metodika. Ochrana (2009) definuje metodiku takto: „*Metodika nepatří do oblasti metodologie. Metodika výzkumné práce je postup (návod, „recept"), jak v praxi postupně realizovat výzkumné procedury vztahující se k realizaci výzkumného cíle. Metodický postup můžeme formálně ztvárnit např. ve vývojovém diagramu či v jiném formalizovaném schématu.*“ Metodika je tak pracovní postup nebo

nauka o metodě. V rámci navrhování systémů je tak metodika stěžejní v rámci navrhování frameworků či jiných pracovních postupů. Metoda je v rámci kontextu konkrétní postup tvorby. Dělení metod je znázorněno na Obrázek 1.



Obrázek 1: Dělení metod, volné zpracování autorky dle (Ochrana, 2009)

Framework se často v rámci kompletačního týmu využívá pro pracovní postup při tvorbě nového systému. První framework byl vytvořen již v roce 1960 a nesl název SDLC neboli Systems development life cycle. Podle Elliotta (2004) bylo hlavním cílem „*směřovat vývoj informačních systémů účelným, strukturovaným a metodickým způsobem, což vyžadovalo, aby všechny fáze vývoje od nápadu až po dodání finálního systému byly provedeny postupně a vždy ve stejné formě*.“ Je tedy patrné, že dané pracovní postupy obsahují různé fáze přes návrh, plánování a řízení až po samotnou implementaci a následnou údržbu. V rámci vývoje docházelo i k odlišným názorům na vývoj systémů, takže se přirozeně vyvinulo několik vývojových přístupů.

Jedním z těchto postojů je vodopádový přístup. Jedná se o model, který své procesy řadí sekvenčně. Fáze projektu jsou tak rozděleny na dílčí části, které na sebe přímo navazují. V určitých případech se mohou navzájem překrývat. Obecně je v tomto přístupu šest dílčích fází. Analýza požadavků na systém, následný návrh, implementace, testování systému, integrace a údržba. Každá fáze má jako výstup textový dokument či model. Klade se důraz na celkovou dobu životnosti systému.

Dalším přístupem, který je s tvorbou systémů spjatý, je přístup přírůstkový. V literatuře je znám i pod názvem inkrementální. Tento přístup, stejně jako vodopádový, rozděluje

systém na menší celky pro přehlednost. Rozdílem ale je, že přírůstkový přístup dělí systém na malé segmenty, které jsou více separované od celku z důvodu zjednodušení případných úprav v systému. Struktura v těchto segmentech je sekvenčního typu. Přírůstkový model může v rámci své struktury kombinovat sekvenční a iterační metodiky vývoje systémů.

Třetím využívaným přístupem je přístup spirálový. Jedná se o proces, který spojuje prvky prototypového a designového přístupu. Kombinuje tak takzvaný koncept bottom-up a bottom-down. Spirálový přístup má čtyři kvadranty. V prvním kvadrantu je analýza, kde se stanovují cíle hlavní a alternativní. Druhý kvadrant zastupuje vyhodnocení řešení rizik. Ve třetím a čtvrtém je vývoj, kontrola výsledků a plánování interakcí. V každém cyklu si systém stanoví vstupní subjekty a podmínky na ně pro úspěšné splnění.

Mezi další klasické přístupy náleží přístup RAD neboli Rapid Application Development. Tento přístup vytváří iterativní prototypy systémů. Z názvu je patrné, že tento koncept cílí na rychlý vývoj systémů s cílem co nejnižších počátečních nákladů. I přes tyto počáteční ambice je snaha o co nejkvalitnější systém. Konceptuální úroveň je zde zvolena za účelem snížení projektových rizik. Systém je rozdělen na menší segmenty, které umožňují snadnější změny v oblasti vývoje. V rámci těchto prototypových systémů je snazší začlenění zadavatele do aktivní spolupráce.

3.6.1 RUP

RUP neboli Rational Unified Process je metodika pro vývoj softwaru různého typu. Vzhledem k rozsáhlosti životních cyklů a fází se ovšem doporučuje převážně pro rozsáhlejší projekty. Je specifická tím, že rozděluje životní cyklus do čtyř fází, ve kterých proběhne šest stěžejních vývojových disciplín. Mezi tyto disciplíny jsou podle Tia (2020) řazeny: obchodní modelování, předběžné a konečné požadavky, analýza a návrh, implementace, testování a nasazení. Zmíněné procesy nemají pochopitelně stejnou časovou náročnost. Některé mají vyšší prioritu a odehrávají se stěžejně v odlišných disciplínách. Obchodní modelování a předběžné požadavky, jak již název vypovídá, se budou odehrávat převážně v raných fázích oproti testování atd. Každá ze čtyř fází má svůj cíl, který je potřeba dokončit, aby mohl projekt pokročit do další fáze.

Tato metodika upřednostňuje spokojenost uživatelů a je tak žádoucí, aby vývojář aktivně komunikoval se zadavatelem, aby byl zadavatel přímo zapojen do procesu vývoje systému. V první fázi se tak od zákazníka zjistí požadavky na systémové procesy a

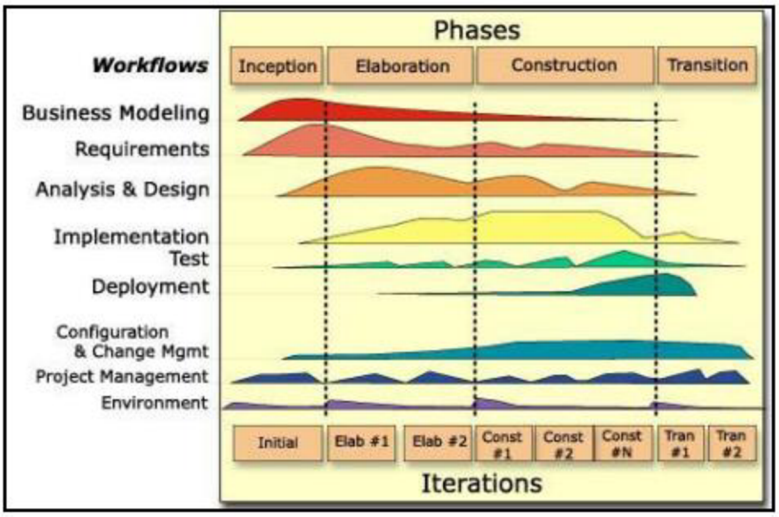
požadavky. Pro zpracování obchodních informací získaných od zákazníka se využívá Business Modeling Canvas neboli BMC (Sudarma, 2021). Pro setřídění požadavků na systém se v rámci RUP vytváří tabulky funkčních a nefunkčních požadavků společně s údajem o agentovi, na kterého bude požadavek přidělen. Na základě této analýzy se konstruuje návrh. V rámci analýzy se využívá Use Case diagram pro znázornění propojenosti aktérů. Následuje fáze vypracování konkrétních prototypů a fáze konstrukce, ve které již dochází k programovému zpracování prototypů. Následuje zpracování navrženého předběžného modelu pomocí vybraného programového kódu. Konečnou fází je samotné nasazení systému. Všechny fáze a jejich provázanost s náročností, jsou zobrazeny na obrázku.

3.6.2 AUP

AUP neboli Agile Unified Process je typem metodiky, která staví na základech RUP (Edeki, 2013). AUP byla představena roku 2005 Scottem Amblerem. Zaměřuje se převážně na podnikové softwary, které se snaží srozumitelným způsobem přetvořit pomocí agilních technik. AUP využívá agilní techniky jako je řízené testování TDD a agilní modelování AM. Testem řízení vývoj přidává do této metodiky fakt, že jsou požadavky převedeny na testovací případy a všechny prvky se opakovaně testují pro odhalení nesourodostí. Agilní modelování se od klasického liší vyšší flexibilností, tudíž se hodí více do prostředí, které se dynamicky mění.

AUP metodice schází oblast procesní dokumentace. Předpokládá se totiž, že vývojáři a zaměstnanci organizace vědí, co od systému chtějí a nepotřebují tyto zdlouhavé záznamy. Tento potencionální nedostatek ale vynahrazuje velmi propracovaná síť odkazů, kde uživatel najde rady různého rázu. AUP není závislá na nástroji a v rámci tvorby se uvažují pouze problémy a činnosti, které podle vývojáře skutečně nastanou. Model je tak odstíněn od různých podproblémů a má jednu dějovou linii.

AUP mapuje sedm fází vývoje, jak je vidět na Obrázek 2. V první fázi modelu se řeší pochopení požadavků na systém, problémové domény atd. Následuje fáze implementace, kdy se modely realizují do konkrétního kódu. Další fází je hodnocení pro zajištění kvality. Odhalují se závady ve fungování systému a ověření, zda systém splňuje vstupní požadavky. Po úspěšném testu se může systém zpřístupnit uživateli. Nad rámec této systémové oblasti dochází ještě k následné konfiguraci systému, řízení aktivit v rámci projektového management a konzultace se zadavatelem nad správnou volbou nástrojů pro implementaci.



Obrázek 2: Fáze vývoje metodiky AUP (Edeki, 2013)

3.7 ADELFE

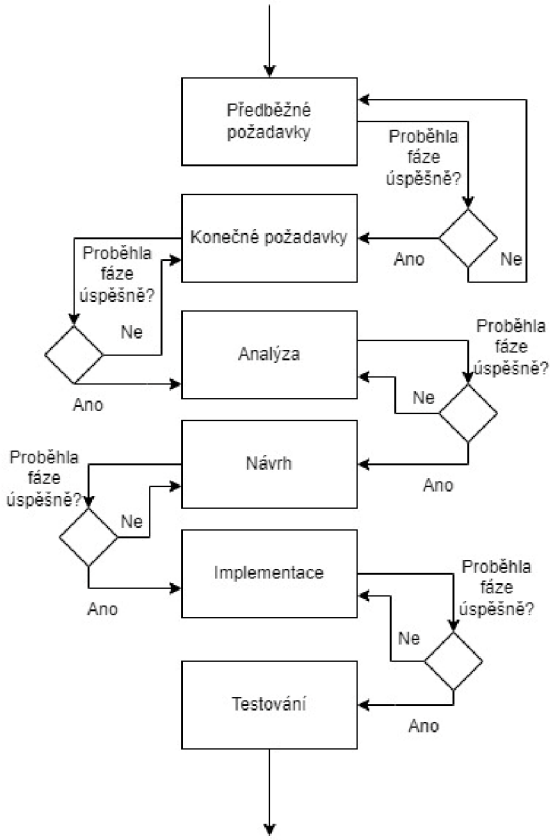
ADELFE se od roku 2003 používá v průmyslové i akademické oblasti. Na bázi této metodiky vzniklo přes dvacet systémů a stále jich přibývá. Stále se zdokonaluje, aby odpovídala současné integraci v oblasti Multi-Agent Oriented Software Engineering.

Metodika ADELFE si odnesla svou zkratku z francouzštiny. V překladu se jedná o „Soubor nástrojů pro navrhování softwaru s novými funkcionalitami emergence“. Tato metodika byla vytvořena s cílem vést designera během vývoje multiagentního systému. Tento koncept je tak vhodný i pro začátečníky v této oblasti. Předpokládá se ale, že bude designer obeznámen se speciálním slovníkem, který je zde používán.

ADELFE je založena nejen na objektově orientovaných metodologií, ale řídí se i Rational Unifield Process (RUP) a zápisy se provádí v UML a AUML. Tato metodologie se

skládá z několika aktivit, které jsou věnovány AMAS. Jedná se tak o pomyslný výklenek v klasické metodologii. Díky mechanismu spolupráce můžeme rozlišit nominální a kooperativní chování agentů. Nominální implementuje jejich funkce a kooperativní řeší všechny problémy, kterým agent čelí. Tyto dva typy chování organizuje ADELFE postupně v oddělených činnostech. U kooperativního chování dochází k problematice z hlediska nepředvídatelných nebo nespolupracujících situací. Vývojáři ADELFE vyzdvihují, že se díky této metodice sníží doba vývoje systému a tím i náklady vzhledem ke včasnému rozpoznání chybných návrhů a částí modelu. Díky technik založených na simulaci se mají zvýšit i adaptační schopnosti (Mefteh, 2015).

ADELFE kompletuje tvorbu systému do šesti fází: předběžné požadavky, konečné požadavky, analýza, návrh, implementace a testy. Tyto fáze na sebe sice postupně navazují, ale mezi každou fází je důležitý krok ověřování, jak je vidět na Obrázek 3. V tomto ověřování může dojít k situaci, kdy je potřeba znovu fázi projít. Každá fáze obsahuje výstup v podobě příslušného diagramu či obyčejného textu.



Obrázek 3: Struktura ADELFE, volné zpracování autorky dle (Mefteh, 2015)

Ve fázi předběžných požadavků se zjišťují od zákazníka požadavky na daný systém, jeho potřeby. Specifikují se očekávání zákazníka, ale i návrháře a dochází k obsáhlým konzultacím, jak by měl systém vypadat pro uživatele. Na základě této konzultace si obě strany určují očekávání, co by měl daný systém umět, a jaká by měl mít omezení.

Při určování předběžných požadavků vývojář vytvoří výstup v podobě textového souboru. Úkolem konečných požadavků je tento popis dostat do strojově srozumitelné podoby pomocí diagramu Use Case. V tomto diagramu se specifikují případy užití a aktéři, kteří mohou s těmito případy pracovat. Namodeluje se a objasní propojenost sekvenčních diagramů. V rámci nich se dohledají chyby ve spolupráci pomocí diagramu spolupráce a namodelují se scénáře. Závěrem je tak vytvořená charakteristika prostředí systému a oprava chybných interakcí.

Ve fázi analýzy se popisují vytvořené entity. Určuje se, zda jsou aktivní nebo pasivní. V rámci entit se modeluje diagram tříd, ve kterém jsou znázorněny všechny atributy a metody daného agenta. V rámci metodiky ADELFE se zaznamenávají i role agentů. Všechny modely jsou následně implementovány.

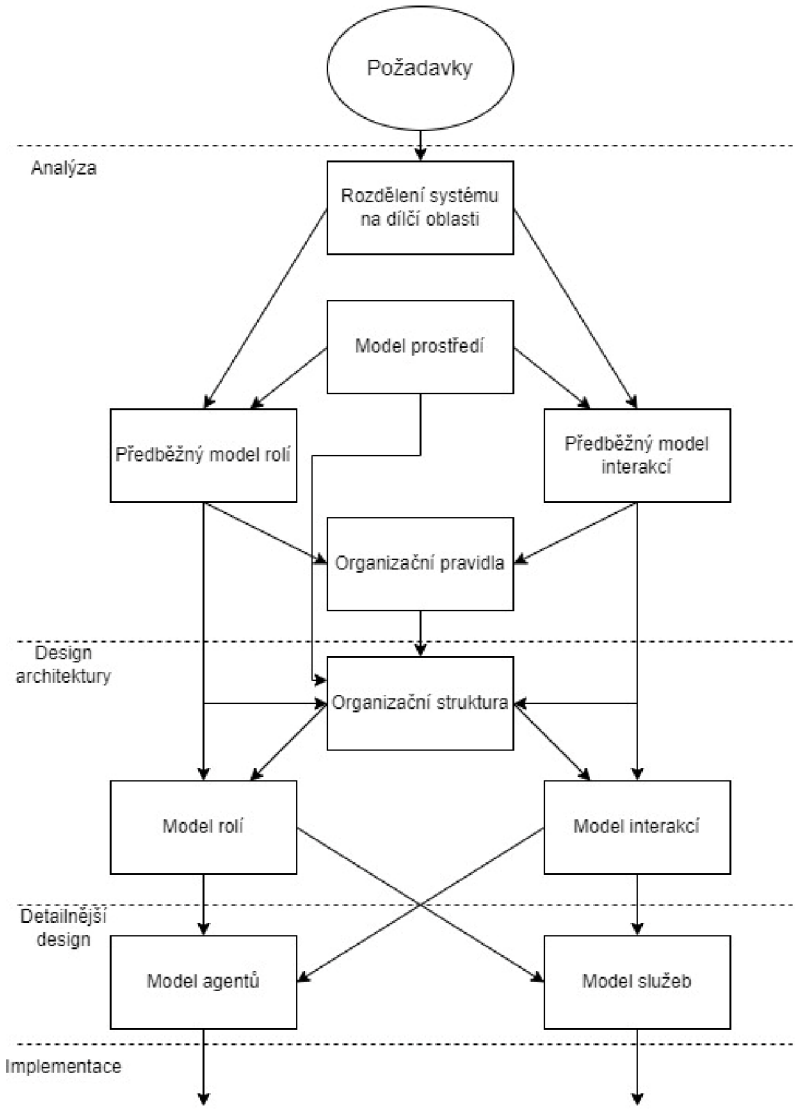
Na základě vytvořeného modelu systému se v konečné fázi na základě testování rozhoduje, zda je zvolený postup vhodný pro zkoumanou doménu. Pakliže ano, dochází k upravování zaznamenaných chyb v rámci běhu systému.

3.8 GAIA

Další metodikou, kterou si zde rozebereme je GAIA. Název vzešel ze sousloví Geometrical Analysis for Interactive Aid a v roce 1988 ji představili profesoři Jean-Pierre Brans a Bertrand Mareschal. Tato metodika je podle Pokorného (2014) užitečná díky své schopnosti grafického znázornění vícekriteriálního rozhodovacího problému do dvourozměrného prostoru. GAIA je vhodná pro řešení úloh malého až středního rozsahu (zhruba do 100 agentů). Schopnosti těchto agentů a jejich služeb jsou statické, jelikož se za běhu nemění. Tato metodika se zabývá makro i mikro pohledem na daný systém. Modeluje jak agenty jako takové, tak jejich zasazení do společenstva. V tomto společenstvu se ovšem neuvažují žádné konflikty mezi agenty. Předpokládá se, že budou agenti jednat ku prospěchu společného cíle. Agenti zasazení do společenství nejsou heterogenní v tom ohledu, že nemají stejné architektury ani programovací jazyky. V GAIE se nedělají žádné závěry ohledně

platformy, na kterou má být následný systém nasazen. Tento aspekt se nebere v úvahu. Podle Wooldridge (2000) je GAIA vhodná pro využití například v systémech ADEPT a ARCHON, což jsou systémy, které mají svůj původ v unixu.

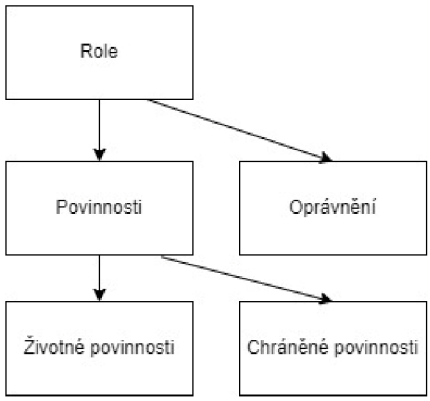
GAIA je svým návrhem hodně podobná PROMETHEU, což se taky odráží v její struktuře, která je vidět na Obrázek 4. Tato struktura se skládá ze čtyř fází, které obsahují několik vývojových kroků.



Obrázek 4: Struktura GAIA, volné zpracování autorky dle (Wooldridge, 2000)

Na Obrázek 4 je patrné, že se analytik přesouvá od abstraktního vymezení ke konkrétnějším konceptům. Vymezení požadavků není přímo závislé na paradigmatu používaném pro analýzu a návrh. To umožňuje analytikovi přejít od požadavků hned k návrhu, který je pak možné ihned implementovat. GAIA obsahuje entity, které by se daly rozdělit do dvou kategorií: abstraktní a konkrétní (2003). Abstraktní jsou takové, které se používají v rámci analýzy a nemusí tak být nutně realizované v rámci konečného systému. Oproti tomu konkrétní entity, jak název vypovídá, v rámci systému realizované jsou.

Pojďme se zaměřit oblast analýzy. Již dříve jsme si ujasnili, že ve fázi analýzy se snažíme rozvinout naše porozumění systému a jeho struktuře. Stanovujeme si cíle na danou strukturu a očekávané globální chování. Na danou strukturu pohlížíme jako na soubor rolí, přičemž každá role je definována čtyřmi atributy: povinnosti, oprávněním, aktivitami a protokoly, jako tomu je na Obrázek 5.



Obrázek 5: Atributy rolí v GAIA, volné zpracování autorky

Povinnost určuje funkčnost a jako taková je klíčová pro spojení s rolí. Dělí se na dva typy: životné povinnosti a chráněné povinnosti (Wooldridge, 2000). Životné povinnosti popisují stavy věcí, které musí agent vyvolat za určitých podmínek v prostředí. Oproti tomu chráněné povinnosti říkají, že se „neděje nic špatného“ a agent může udržovat současný chod věcí. Příkladem může být tepelná regulace, kdy je nastavená hodnota mezi 0-10 stupni. Chráněná povinnost tedy kontroluje, zda je teplota v místnosti pod 30 stupňů a udržuje nastavenou povinnost v platnost.

Jako další atribut jsme si uváděli oprávnění. To nám stanovuje, jaké zdroje máme k dispozici pro plnění povinností a jaké limity se k oprávnění stahují. Často se jedná například o přístupy ke složkám atd. V GAIE máme prostředky, které jsou typu informace. Tato informace má tři druhy zpracování a to: reads, generates, changes.

Role jako taková může generovat informace, tedy vytvářet aktivity. Tyto aktivity jsou soukromé pro daného agenta. Jsou spojené s rolí a agent je může provádět bez interakce s jiným agentem. Posledním uvedeným atributem rolí jsou protokoly. Ty definují způsob integrace s jinými rolemi. U těchto protokolů víme určité informace, kterou jsou pro vymezení protokolu stěžejní. První informací je, k čemu je protokol určen. Jedná se o krátký slovní popis. Dále je určený iniciátor. Tedy který protokol zahajuje rozhovor. Ve spojitosti s touto informací tak víme o všech, kteří se rozhovoru účastní. Při rozhovoru se zaznamenává, jaké informace má zahajující článek v okamžiku vzniku komunikace a jaké informace si navzájem vymění v průběhu. Celý proces ukončuje fáze zpracovávání, ve které máme zaznamenány procesy použité v hovoru, dané argumenty atd. Je důležité si uvědomit, role jako takové jsou v tomto smyslu chápány jako určité abstraktní šablony. Při převzetí role daným agentem se v daném okamžiku vytvoří instance této role.

Fáze analýzy obsahuje i model prostředí. Tento model si můžeme představit jako abstraktní reprezentaci okolí, do kterého bude daný MAS zasazen. Na obrázku si můžeme všimnout, že po fázi analýzy, kde jsme si předběžně nastínili role a interakce, následuje fáze designu architektury. Ta obsahuje již modelování rolí a interakcí do modelu a nastínění celkové struktury systému. V detailnější fázi návrhu architektury se již soustředíme na vytváření agentů a jejich poskytovaných služeb. Za touto fází následuje již samotná implementace.

Mnoho autorů, stejně jako Wooldridge, porovnává návrhový přístup v GAIE ke KGR přístupu. Metodika GAIA se nesnaží sjednotit analýzu a abstraktní návrh multiagentního systému s jeho konkrétním návrhem a implementací s konkrétní technologií agenta. Výstup procesu se bere tradiční formou na nižší úrovni. KGR je oproti tomu silně architektonicky zavázána k BDI architektuře, u které se předpokládá zdokonalení procesu návrhu tak, že bude specifikovaný agent přímo spustitelný (Chen, 2021).

Další fází GAII je fáze designu. Ve fázi analýzy jsme ze získaných informací vytvořili předběžné návrhy potřebných modelů. Ve fázi designu se ve většině případů tyto předběžné modely komplementují tak, aby vznikl předobraz s dostatečně nízkou úrovní abstrakce, aby mohly být tyto modely snadno implementovány. Metodika GAIA se specificky soustředí na takový design, aby byla možná implementace tradičním způsobem i s objektově orientovanými prvky. GAIA je tak zatížena na řešení problému, jak realizuje každý agent své cíle na systémové úrovni. Rozebírá, jaké úkoly musí každý agent vykonat pro plnění těchto cílů. Zvláštností tak je, že již neřeší, jak konkrétně bude agent tyto služby poskytovat.

Řešení tohoto úhlu pohledu se nechává na konkrétní doméně, na které je systém implementován.

Ve fázi designu je několik modelů, které pomáhají zpřesnit mapovanou strukturu. Patří mezi ně model rolí, agentů, služeb a interakcí. Model rolí popisuje z abstraktního hlediska očekávané funkce entity. Identifikujeme tak klíčové role v systému, které mají dva typy atributů. Těmito atributy je oprávnění neboli práva spojená s rolí a odpovědnost role. Práva spojená s rolí definují, které typy a množství zdrojů lze použít s danou rolí. Tento aspekt je zachycen v atributu oprávnění role. Oproti tomu odpovědnost role znázorňuje funkčnost. V tomto atributu máme uvedeno, jaké zdroje můžeme legitimně používat v rámci role a jaké máme limity pro zpracování těchto zdrojů.

Dalším modelem, který v GAIE najdeme, je model agentů. Jeho úkolem je mapovat zaprvé různé druhy agentů, které budou v systému využívány a zadruhé instance těchto agentů. Konkrétní typ agenta je určen jeho rolemi, které je schopen vykonávat. Agent může vykonávat různé množství rolí v systému. V modelu se tak orientujeme podle kardinality zobrazení, rolemi a typy. Obvykle je mezi kardinalitou a rolemi a typy nastavena hodnota 1:1. V modelu pak můžeme určit, s kolika agenty určitého typu se pracuje ve smyslu instancí. Můžeme nastavit přesný počet instancí N či rozsah instancí mezi M a N. Pro situaci, kdy můžeme mít v daném použití instanci 0 a více, použijeme symbol hvězdičky a pro označení pro jednu a více instancí klasické plus. Oprati jiným metodologiím se v GAIE neuvažuje nad dělením typů agentů.

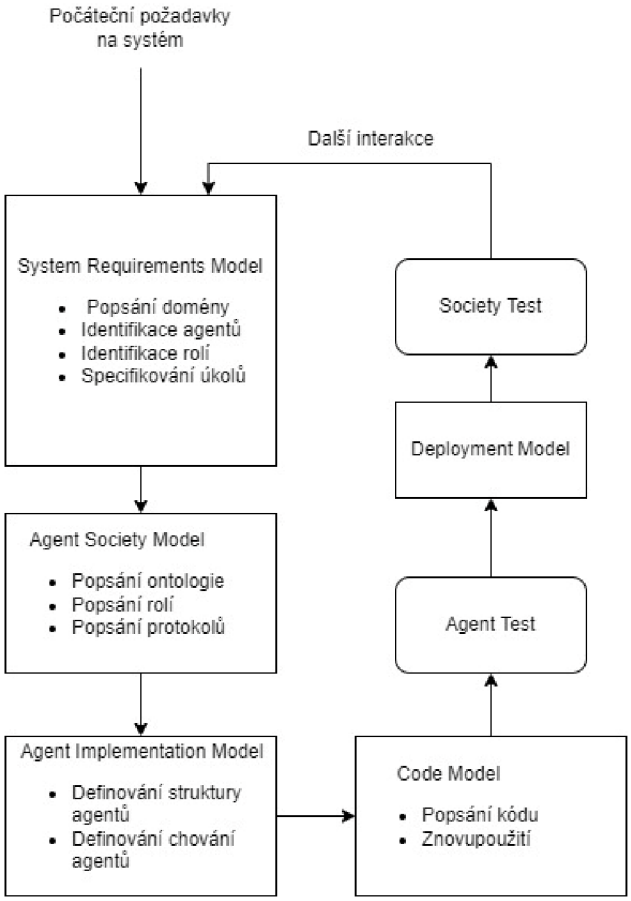
Po modelu agentů je třeba sestavit model služeb. Tento model určuje všechny služby, které jsou spojené s rolí agentů a specifikuje jejich vlastnosti. Tato služba je v přeneseném slova smyslu funkce agenta. Jedná se o určitý koherentní blok činností, který je dostatečně jednoduchý na to, aby se do něj mohl agent zapojit. Každá aktivita, která se tak zde nastaví, odpovídá konkrétní službě. Oproti tomu každá služba neodpovídá aktivitě. V této fázi vývoje si určujeme všechny předběžné podmínky pro danou oblast, výstupy, vstupy a potřebné protokoly. Podmínky jak předběžné, tak následné, představují omezení pro danou službu. Souvisí tak s bezpečnostní vlastností každé role. Služby odvozujeme ze sestaveného seznamu protokolů. Zajímavostí je, že z pohledu objektově orientovaného programování může služba odpovídat metodě, a to vzhledem k blokovému přístupu. Vývojář si tak může ve výsledku vybrat, jakým způsobem bude tento model služeb implementovat.

Další model, který se v GAIE vytváří ve fázi designu, je model-dostupnosti. Dalo by se říci, že se řadí k nejjednodušším v této metodologii vzhledem ke grafickému ztvárnění.

Tento model zaznamenává komunikační spojení mezi agenty. Jedná se o orientovaný graf, kde hrany představují komunikační kanály mezi uzly. Tyto uzly představují agenty. Model dostupnosti nedefinuje, jaké zprávy se odesílají. Neříká ani, kdy se odesílají. Znázorňuje pouze komunikační cesty a jeho hlavním účelem je odhalit úskalí v potencionálních smyčkách komunikace za běhu. Komunikační kanály a uzly vytváříme na základě modelu agentů či modelu služeb. V některých případech to může být obráceně. Máme dán model dostupnosti a na jeho základě přetváříme model agentů a služeb, či se rovnou vrátíme do fáze analýzy.

3.9 PASSI

PASSI neboli Process for Agent Societies Specification and Implementation je metodologie vhodná pro postupný vývoj více agentů. Zakládá se na teorii MAS, využívá UML modelovací jazyk, architekturu FIPA pro implementaci agentů a XML pro komunikaci a mezi agenty (Cossentino, 2012). Popis metodiky PASSI se skládá z pěti komponent. Systémových požadavků, navržení agentů, implementace agentů, vytvoření kódu a nasazení. V rámci návaznosti na SPEM neboli Software Process Engineering Metamodel lze říct, že se proces skládá z procesních komponent, přičemž je každá komponenta tohoto procesu tvořena fázemi, které jsou rozložitelné na konkrétní kroky. Jeden proces PASSI odpovídá jedné komponentě SPEM. Rozložení je zobrazeno na Obrázek 6.



Obrázek 6: Struktura PASSI, volné zpracování autorky dle (Cossentino, 2012)

Při sestavování systému dle metodologie PASSI je jako u ostatních metodologií třeba zjistit počáteční požadavky na systém. V rámci těchto informací se sestavuje model systémových požadavků, který se skládá ze čtyř fází. První fází je popis požadavků na doménu. Tyto požadavky je žádoucí vyjádřit pomocí konvenčních diagramů případů použití. Po vytvoření konvenčních diagramů nastává otázka, jaké agenty bude systém konkrétně obsahovat. Identifikují se tak konkrétní agenti. Vytvořenému agentovi se definuje odpovědnost, která je vyjádřena dle balíčků UML. Následně se zavedou role pomocí sekvenčních diagramů. V rámci těchto sekvenčních diagramů je pro větší konkrétnost a hloubku modelu vhodné navrhnout konkrétní scénáře. Čtvrtou fází v rámci systémových požadavků je fáze specifikace úkolů, které se vytvářejí za pomoci diagramu činností.

Dalším modelem pro práci je model společnosti agentů, ve kterém se určují interakce a závislosti mezi agenty. Prvním krokem v tomto modelu je popis ontologie. Pro ontologii se využívá diagram tříd a různá omezení OCL pro popis znalostí a komunikačních dovedností.

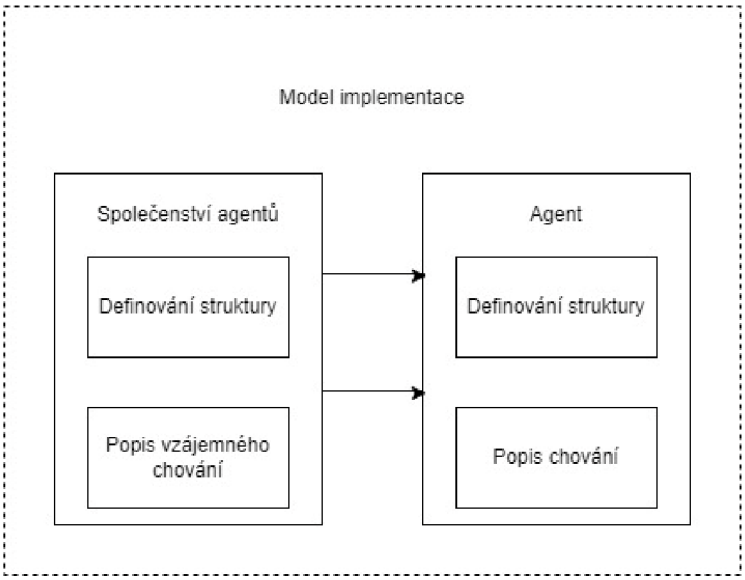
Diagramy tříd se využívají také k popisům rolí a schopností agentů. V modelu společenství agentů se popisují protokoly za využití sekvenčních diagramů.

Dalším krokem v rámci vytváření systému je vytvoření modelu implementace pro agenty. V tomto modelu se řeší architektura z hlediska tříd a metod pro agenty. Rozdílem oproti klasickému přístupu v objektově orientovaném modelování je přístup k abstrakci. Na model se nahlíží buď ze sociálního globálního hlediska nebo z pohledu jednotlivých agentů.

V rámci dokončování sktruktury agentů, jejich oprávnění atd., se v další fázi vytváří konkrétní kód. Aby bylo vytváření kódu přehledné a rychlé, využívá se generování z modelu agentů pomocí doplňků přímo pro metodologii PASSI. V rámci těchto doplňků tak může programátor generovat základní kostry modelu. Pomocí ukládání do knihoven lze části kódu využívat opakovaně.

Jako poslední model se v rámci PASSI vytváří model distribuce částí navrženého systému mezi hardwarovými jednotkami. Konfiguruje se nasazení, testuje se činnost agentů a ověřují se celkové výsledky integrací.

V rámci metodiky PASSI lze nahlížet na agenta dvojím pohledem. V prvních krocích je agent vnímán jako autonomní entita, která sleduje svůj cíl a je schopna se začlenit do sociálních vztahů. Entita může přijímat v průběhu sledování svého cíle různé role a plnit různé úkoly. Úkolem se v tomto ohledu myslí jednotka interaktivního a individuálního chování. Každý agent má při tom uchovanou reprezentaci světa pomocí ontologie, na kterou se odkazuje ve všech zprávách, které si agenti posílají a které tvoří komunikaci. Druhým pohledem na agenta je tak aspekt sociální, v rámci kterého se určuje vnitřní struktura společenstva a chování agentů mezi sebou, jak můžeme vidět na Obrázek 7.

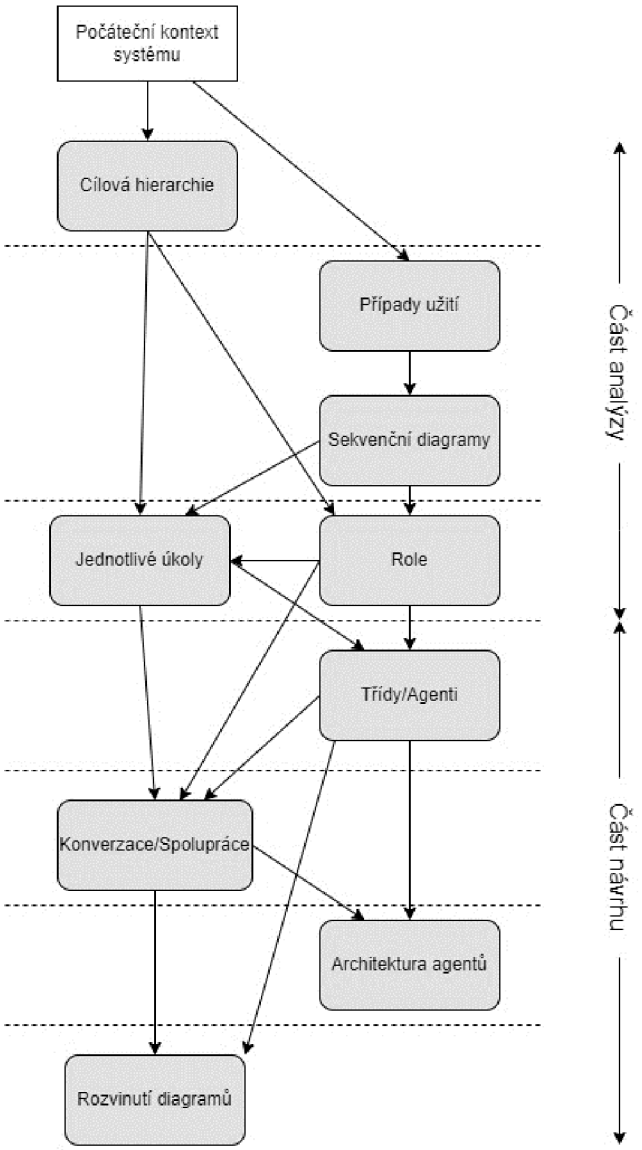


Obrázek 7: Model implementace v PASSI, volné zpracování autorky dle (Cossentino, 2012)

3.10 MASe

Multiagent Systems Engineering neboli MASE je metodika kompletního životního cyklu multiagentních systémů. Můžeme na tuto metodiku nahlížet jako na abstrakci objektově orientovaného paradigma, kde jsou agenti souborem objektů. Z tohoto hlediska si tak ulehčíme od problémů s určováním, co je a není agent. Primárním cílem MASE je tak pomoci konstruktérovi nastavit počáteční sadu požadavků, zanalyzovat, navrhnout a implementovat fungující multiagentní systém. Byla založena institucí Air Force a zapadá mezi tradičně koncipované metodiky. Kroky a fáze MaSE jsou znázorněny na Obrázek 8.

Na obrázku můžeme vidět dvě části: analýza a návrh. Ve fázi analýzy máme zachycení cíle, použití případů užití a upřesnění rolí. Fáze návrh má oproti tomu kroky čtyři: vytváření tříd, konverzací, sestavení architektury agentů a rozvinutí diagramů neboli návrh samotného systému.



Obrázek 8: Struktura MASE, volné zpracování autorky dle (Alvin, 2019)

Záměrem této metodiky je, aby mohl návrhář volně přecházet mezi jednotlivými kroky a fázemi volně. S každou úpravou se tak zdokonalují podrobnosti až do té fáze, kdy je vytvořen komplexní a konzistentní systém. Při tvorbě můžeme sledovat změny v průběhu procesu, což je velkou výhodou.

Cílem prvotní fáze analýzy je vytvoření sady rolí, díky kterým popíšeme, co musí systém umět, aby splnil požadavek klienta. Pro splnění požadavků musíme od zadavatele zjistit množství informací. Potřebujeme zjistit jaká jsou očekávání od systému a získat hlavní údaje potřebné pro procesní výzkum. Údaje můžeme získat mnoha způsoby. Nejčastější je ovšem sbírání údajů klasickým pozorováním, rozhovorem s konkrétním zadavatelem, nebo

teoretickým studiem oblasti. Získaných údajů je nepřeberné množství. Informace si musíme pro modelovací účely setřídit tak, abychom mohli postupně modelovat strukturu systému. U prvotních požadavků na systém zohledňujeme požadavky nejen klienta, ale i návrháře. Musíme si určit, co je pro obě strany podstatné, co by měl daný systém umět, a jaká by měl mít omezení. Při určování požadavků je nutno si uvědomit, zda jde o funkční nebo nefunkční. Funkční požadavky říkají, co by měl systém umět. Například že má na stránku zobrazovat maximálně 30 záznamů z databáze. Nefunkční požadavky říkají, jaký by měl systém být. Například doba odezvy, požadavky na RAM, atd. Požadavky jako takové mají svou konečnou fázi. Při určování předběžných požadavků jsme si vytvořili jako výstup detailní popis, například v textovém souboru. Úkolem konečných požadavků je tento popis dostat do strojově srozumitelné podoby, kterou pak bude možné popsat formou diagramů.

Po určení požadavků je nutné si stanovit cíle a správně je strukturovat dle důležitosti do grafu. Ve většině případů užíváme stromovitý graf, kde zadáváme cíle dle důležitosti a znázorňujeme jejich provázanost.

Po ujasnění požadavků na systém a jeho cílů je na řadě dostat tento popis do strojově přijatelné podoby. K tomuto účelu nám slouží množství diagramů. Prvním podstatným diagram ve fázi analýzy je Use Case neboli diagram případů užití. Objasňuje soubor funkcí, na které musí systém reagovat. Bittner a Spence ve své knize uvádějí (Bittner, 2002), že pakliže se chceme dostat k jádru toho, co musí systém vykonávat a umět, musíme se nejdříve zaměřit na to, kdo jej bude používat nebo kdo jím bude využíván. Na základě toho jsou v Use Case diagramu dva prvky: actors a use case. Aktér představuje objekt, který určitým způsobem komunikuje s případy užití. Entity jsou mimo systém a jsou napojeny s případy užití pomocí vztahů.

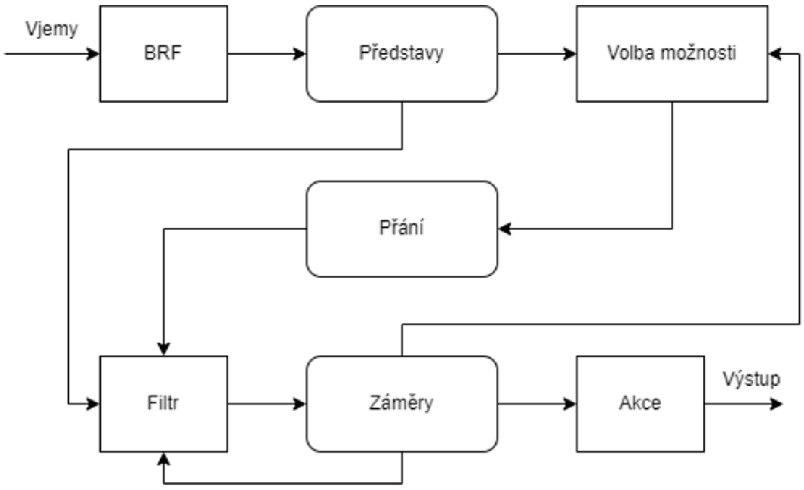
Diagram případů užití je silně provázán se sekvenčními diagramy. Ty nám vizuálně reprezentují objektové interakce a tím obsahují informace potřebné pro pochopení programu či jeho ladění (Alvin, 2019). V diagramu můžeme vidět časovou posloupnost, v jaké zasílají objekty či uživatelé zprávy. Můžeme zde zachytit i spolupráci několika objektů na jednom cíli.

Po namodelování těchto dvou zmíněních diagramů můžeme pokročit k detailnějšímu rozpracování. Přecházíme tak do fáze návrhu. Každý systém se skládá z mnoha entit, které mají určité vlastnosti a atributy. Cílem diagramu tříd je tyto prvky systému popsat a propojit s prvky, ke kterém určitým způsobem přistupují. Popis třídy obsahuje její název, atributy a metody či operace třídy

3.11 KGR

Metodologie KGR je zaměřena převážně na práci a návrh systémů s agenty BDI. BDI agenti jsou kombinovaným typem agenta deliberativního a racionálního. V překladu zkratka BDI znamená beliefs, desires a intentions. Fungování tohoto typu agenta se tak vyznačuje jeho představami, přáními a záměry. V literatuře se můžeme setkat s tímto pojmem v rámci softwarového modelu Belief-Desire-Intentions, který je inspirovaný teoriemi od Bratmana. Celý koncept agentů BDI leží na praktickém usuzován v rámci dosažení zvolených cílů. Tito agenti si naplánují postup realizace cílů a v průběhu času ho doplňují.

BDI agenti mají pět hlavních vlastností, které je definují. Důležité je vnímání okolního prostředí. Tito agenti si uchovávají obraz okolního prostředí, což je odlišuje od obyčejných reaktivních agentů. Jsou racionální a jejich akce jsou řízeny podle logické vnitřní kalkulace. Jejich chování a cíle jsou definované na základě deklarativního, symbolického programovacího jazyka. Struktura BDI agentů má tři hlavní datové struktury: představy, přání, záměry (Hájková, 2012).



Obrázek 9: Struktura BDI agenta, volné zpracování autorky dle (Zbořil, 2006)

Na Obrázek 9 je znázorněna vnitřní struktura BDI agenta. Každý agent přijímá z okolního prostředí pomocí senzorů vjemy. Tyto vjemy následně zpracovává do své představy o vnímání okolního stavu světa. Tyto informace jsou ukládány symbolicky. Následuje zpracování na základě nastavených filtrů neboli omezení. Po zpracování informací dochází k vytvoření několika možností jednání, ze kterých si agent následně

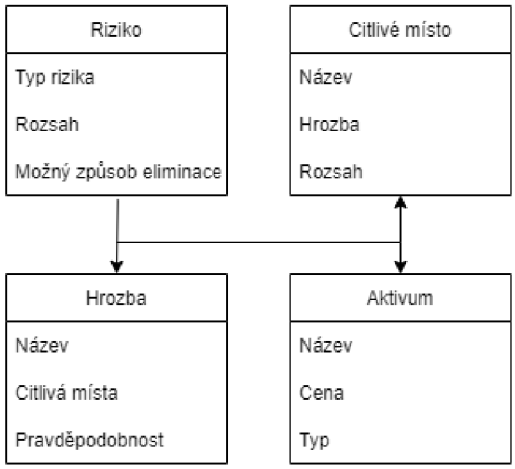
vybírá. Do tohoto kolotoče zpracovávání přichází agentovy přání. Definují, čeho by chtěl agent dosáhnout, jaké stavy by chtěl uskutečnit. Nastavené cíle agenta mohou být krátkodobého i dlouhodobého rázu. Předpokládá se, že agent nesplní všechny cíle, které má nastavené. Tyto cíle jsou také v procesu schvalování. Následuje oblast záměrů. Záměry jsou určitým typem závazku, který může a měl by vést ke splnění nastavených cílů. Tyto záměry mohou být mezi agenty sdíleny, pakliže má více než jeden agent stejný cíl. Celá struktura je doplněna o plánovač, který na základě všech uvedených datových struktur vytváří přesný plán, jak by měl agent dosáhnout cíle. Poslední oblastí ve struktuře BDI agenta je již samotná akce a následný výstup této akce.

BDI agenti mají rozhodovací mechanismus řízený BDI logikou, která je ovšem odlišná u různých druhů implementace. Rozhodování tak může být rozšířeno o temporální logiku, která se větví v čase neboli CTL. Tedy computer tree logic. Z názvu je patrné zachycení větvení v čase pomocí stromové struktury. Rozhodovací mechanismus může obsahovat i modální logiku. Tato logika stojí na odlišném konceptu, kdy máme soubor možných reprezentací takzvaných světů pro koncept přání, záměrů a představ.

Přístup KGR byl vyvinut, aby naplnil potřebu přistupování ke specifickým komplexním multiagentním systémům, které jsou založené na technologii BDI agentů, procedurálně řízených systémech a distribuovaných multi-agentních systémech. KGR má silně spjatou architekturu s architekturou BDI agentů. Návrh je tak na takové úrovni dokonalosti, aby posléze došlo k vytvoření přímo spustitelného agenta. Na úrovni agenta se tak modelují představy, cíle a plány agenta. Oproti tomu na multiagentní úrovni se modelují přímo hierarchické struktury daných agentů. Model se tak rozšíří o abstraktní třídy neboli role a instance těchto tříd.

Strukturu KGR tvoří několik modelů: model představ, model cílů, model plánů a agentní model (Zbořil, 2006). Model představ se i na základě struktury BDI modeluje jako první. Jedná se o množinu objektových diagramů, na základě kterých, se tvoří konkrétní kolekce možných variant představ pro konkrétního agenta. Diagram tak znázorňuje aktuální stavy, které jsou převzaty z pozorování a zobrazuje vztahy mezi agenty. Dochází ke zvažování převzatých informací, přičemž je toto zvažování chápáno jako množina funkcí, které jsou používány pro transformaci představ. Na základě toho KGR dělí představy do tří oblastí: pragmatické, vyčíslitelné a pomíjivé. V rámci pragmatických představ se v literatuře setkáme s označením extensional. Jedná se o funkce, které pracují s predikáty. Tento predikát je obsažen v odvozovaném systému agenta. Představy vyčíslitelné mají

v anglických zdrojích označení evaluable. Jejich určení vzniká na základě nastaveného algoritmu, který pracuje v určitém programovacím či odvozeném jazyce. Posledním typem představ jsou představy pomíjivé neboli ephemeral. Tyto představy jsou specifické svou životností. Existují právě v daná okamžik používání či zachycení a nejsou dále ukládány do báze představ. Mají tak velmi pomíjivou životnost. Jednoduchý model představ máme znázorněn na Obrázek 10.



Obrázek 10: Model představ BDI, volné zpracování autorky dle (Zbořil, 2004)

Dalším modelem, který se v rámci KGR metodiky využívá je model cílů. V tomto znázornění uvažujeme tři typy cílů: achieve, verify, test. Každý tento cíl je odlišný svým typem operátoru, který využívá v rámci agenta. Achieve využívá symbol pro vykřičník a značí dosažení stavu predikátu. Zda došlo či nedošlo k naplnění cíle. Verify oproti tomu využívá symbol pro otazník. Jedná se o dotazovací strukturu, kde zjišťujeme pravdivost predikátu. Posledním typem cíle je test, který má značku amerického dolaru. Z názvu je patrné, že testuje, zda agent ví o pravdivosti predikátu. V rámci cílů rozlišujeme, jaké jsou úrovně a na základě čeho vznikají. Cíle vyšší úrovně se v zásadě generují během chodu vykonávání plánů. Cíle nižší úrovně se vytváří v rámci aktivovaných událostí.

Dalším modelem v rámci KGR je model plánů. V tomto modelu vytváříme množinu plánů, kde má každý plán svoje konkrétní schéma. Toto schéma se skládá z jména plánu, grafu, aktivační události a kontextu této události, stavu a podmínek dané akce. Ve vnitřní struktuře dochází k dělení na podmínky while jako v programování. Pomocí while stanovujeme různé podcíle nebo podgrafy. Celá vnitřní struktura je grafického rázu. Například v rámci vnitřního stavu je pasivní stav označen bílým kolečkem. V rámci

podmínek jsou označeny červenou a zelenou barvou různá vlákna na základě úspěšnosti. Plán navíc obsahuje parametry typu příznak, priorita a pořadí.

Posledním modelem, který se v této metodologii rozpracovává je model agentů. V tomto modelu jsou znázorněny agentní třídy a jejich vztahy v rámci agentních tříd. Těmito vztahy propojujeme abstraktní třídy a třídy konkrétní. V rámci modelu agentů máme i model agentních instancí, ve kterém se znázorňují vztahy instancí všech agentů. V tomto modelu máme zaznamenanou celkovou hierarchii, ve které je patrná dostupnost a stavy těchto instancí.

Oproti ostatním metodikám umožňuje KGR dědičnost v rámci agentních tříd. Agenti jsou schopni dědit svoje cíle, představy a plány od předků. V rámci dědění jsou definovány určitý omezení a koncepty, které se vytváří automaticky. Například u dědění diagramů pro představy se dědí i predikát a funkce dané představy. Mohou být ovšem ve třídě, která je od nich odvozená předefinovány. Dědění cílů s sebou nese i dědění aktivační události za předpokladu, že se dědí i představy. V plánu se předpokládá dědičnost, pakliže se dědí i dané cíle, představy ale i události.

KGR má v rámci agentních tříd specifickou vlastnost, kdy za předpokladu více asociovaných tříd umožňuje vytvoření jedné velké nové agentní třídy. V porovnání se třídou, která by vznikla formou několikanásobné dědičnosti, nemohou agenti upravovat svoje stavy. Vnitřní úpravu svého stavu tak můžou zajistit pomocí asynchronní komunikace. Ve výsledku je tak agent, který vznikl na základě agregace, vždy agent logický.

Při popisování modelů se využívá pojmu instanciování. Tento název pod sebou obsahuje stanovení vztahů mezi třídou a svými instancemi. Těchto instancí může být na jedné straně více. Rozlišujeme dva druhy instancí: statické a dynamické. Tyto druhy se od sebe liší podle toho, v jaké fáze byly navrženy. Statické instance si určíme ve fázi návrhu, dynamické vznikají posléze po spuštění systému.

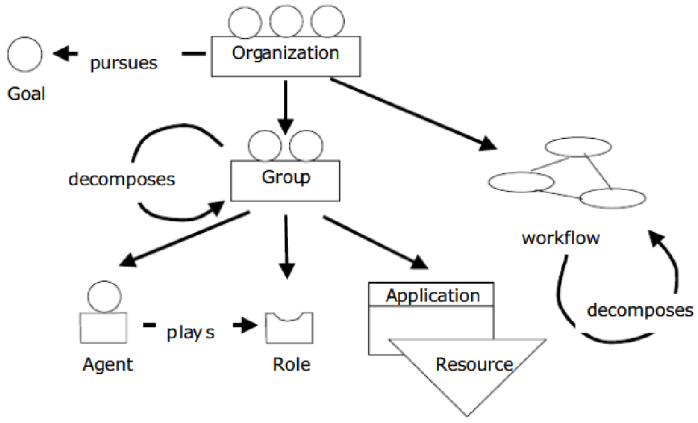
Přístup KGR má vzhledem ke svému obecnému pojetí značné výhody v oblasti návrhu. Jedná se o metodologii, která je vhodná pro obsáhlejší systémy. Máme zde možnost dobrého grafického znázornění všech modelovaných atributů. Nevýhodou však je, že nemůže poskytnout sadu modelů, abstrakcí a také terminologie, které by šly jednotně využívat v průběhu životního cyklu systému.

3.12 INGENIAS

INGENIAS je metodika pro vývoj softwaru multiagentních systémů, která je založena na paradigmatu MDE neboli Model-Driven Engineering. INGENIAS zpracovává jak oblast analýzy, tak návrhu. Tato metodika je určena pro obecné využití. Kromě MDE má své základy i v procesu RUP a UML diagramech. Díky tomu značně usnadňuje její používání. INGENIAS se stále vyvíjí. Vývojáři se snaží tuto metodologii co nejvíce přizpůsobit pro realizaci MAS, což se jim v posledních letech povedlo. Do INGENIAS byl například přidám Agent Framework neboli IAF, který má své základy v platformě JADE (Corchado, 2011).

Vzhledem k návaznosti na paradigma MDE je přístup INGENIAS založený na dvou hlavních komponentách. Modelovacím jazyku a softwarovým nástroji. V metamodelu metodologie máme specifikovaný jazyk, pomocí kterého definujeme koncepty, vztahy a s nimi spojené vlastnosti a omezení. Vzhledem k použitelnosti pro modelování MAS systémů se do modelu zahrnují koncepty agenta a celých skupin agentů. Agent má v rámci struktury INGENIAS velmi podobné vlastnosti jako u předchozích metodologií. Je vymezený svými cíli a disponuje schopnostmi, které má k dosažení těchto cílů. V modelu máme v rámci skupin nejen agenty, ale i zdroje z okolního prostředí, které mohou agenti využívat. Ve struktuře lze zaznamenat interakce mezi agenty, jejich směrovost. Specifikací této metodologie je i možnost zahrnout do modelu část kódu pro bližší specifikaci. Můžeme tak k uzlům přidat algoritmy na základě kterých funguje daný spoj. Tato funkcionalita je vhodná využít hlavně na nižší úrovni zkoumání. Pro vytváření modelů a práci s nimi byl v rámci INGENIAS vytvořen softwarový nástroj INGENIAS Devepment Kit neboli IDK. Tento nástroj podporuje distribuci konkrétních modelů a vytváření k nim patřičné dokumentace. Je zde ovšem podmínka dostupnosti šablon pro danou oblast práce, kterou si musí vývojář stáhnout a nahrát do programu.

Návrhový proces má v případě INGENIE dvě hlavní fáze (Li, 2007). První fází je identifikace klíčových konceptů domény, druhou fází je popis interakcí domény specifikované v prvním kroku. Metamodel obsahuje prvky, které jsou primitivního konceptu, jak je vidět na obrázku.



Obrázek 11: Prvky metamodelu INGENIAS (Pavón, 2005)

V INGENIAS máme pouze dvě složky, které mohou vykonávat úkoly. První složkou jsou agenti, kteří jsou vnímáni jako proaktivní entity. Mohou tak inicializovat dané úkoly podle vlastního úsudku a agendy. Druhou složkou jsou samotné aplikace, které jsou prováděny skrze agenty. Vyvolávají události v rámci změn okolního prostředí. Agenti, kteří sdílí tyto aplikace, jsou zařazeny do jedné skupiny. V této skupině platí pravidla sdílení postupů a znalostí mezi agenty (Bordini, 2009).

INGENIA si na úrovni modelování zakládá na zdravé vnitřní struktuře organizace. V této metodice je tak přidána funkcionalita pro správu agentů v rámci jejich přidávání a odebírání ze struktury. V rámci této funkcionality můžeme dané agenty sledovat. Jedná se o určitý druh kolaborativního filtrování, kdy se v rámci skupiny zvažuje agentův přínos společenstvu (With). Tento způsob řešení s sebou tak nese nízké výpočetní náklady. V rámci společenství jsou tak filtrování agenti, kteří řádně nespolupracují, mají nízkou produktivitu či se vyhýbají plnit nastavené cíle skupiny. Agenti, kteří jsou takto vyloučeni, nemají možnost se do struktury znovu vrátit.

V Tabulka 1 je nastíněna vnitřní struktura, která je rozdělena na fázi analýzy a designu.

Tabulka 1: Struktura INGENIAS, volné zpracování autorky dle (Pavón, 2005)

		Fáze		
	Analýza	Počátek	Zpracování	Konstrukce
		Případy užití. Struktura organizace. Prostředí modelů.	Detailnější zpracování agentního modelu.	Zpřesnění modelů pro pokrytí případů užití.
	Design	Generování prototypů za pomoci nástrojů Zeus a AgentTool. Generování kódu z IDK.	Zpřesnění pracovního postupu. Interakční modely. Zajištění dosáhnutí cílů agentů.	Generování nových modelů. Definování sociálních vztahů, které regulují chování v organizaci.

INGENIAS se řadí mezi metodiky, které řeší implementaci systému. Návrh implementace opírá o IDK, kde se mapují specifické prvky do entit v rámci modulů. Tyto moduly jsou pak zodpovědné za generování kódu.

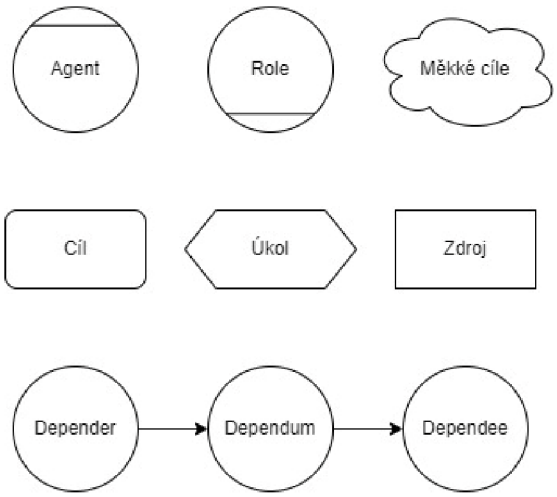
3.13 Tropos

Tropos je metodologií softwarového inženýrství, která je silně orientovaná na agenty. Svůj původ má na Univerzitě Trento. Jeho vývoj se velmi prolíná s jiným projektem pod kódovým označením I*. Tyto dva projekty mají odlišné terminologie, ale stejné zaměření (Giorgini, 2004).

Tropos je s ostatními metodologiemi podobný ve vnímání agenta. Stejně jako metodologie INGENIAS obsahuje agenty typu BDI. Tedy Belief-Desire-Intention. Agenti neboli aktéři, kteří mají vytvořené vlastní strategické cíle a zájmy. Tyto cíle přitom nemusí mít jasně stanovená kritéria či výstup, zda byl cíl splněn či nikoliv. Oproti tomu obsahuje agent i tvrdé cíle, které jasně stanovují například podmínky světa, které musí nastat, aby byl daný cíl splněn. Každý agent má navíc definované úkoly, které musí splnit a zavedené závislosti na ostatních agentech. Tyto závislosti vyznačují, jestli agent potřebuje ke splnění cíle dalšího agenta či nikoliv, například pro dodání zdrojů či úkolů.

Struktura Tropos je založena na dvou stěžejních funkcí. První je pojmenování samotného agenta. Pojem agent je zde využíván od prvotního fáze vývoje požadavků až po konečnou realizaci. Druhá stěžejní funkce pro systém je, že metodologie Tropos klade velký důraz na fázi předběžných požadavků. Oproti jiným metodologiím kladení takové pozornosti činnostem, které předcházejí specifikaci normativních požadavků na systém není obvyklé. Tento přístup tak více rozvíjí funkční a nefunkční požadavky.

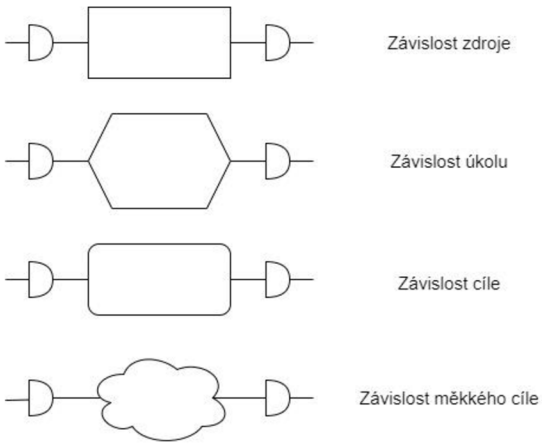
Struktura metodologie Tropos je členěna do čtyř fází vývoje: předběžná analýza požadavků, pozdní analýza požadavků, architektonický návrh a design. První fází, kterou je třeba při navrhování systému zdokumentovat, je fáze předběžných požadavků. V této fázi je stěžejní pochopení problému zkoumáním organizační struktury daného systému. Vývojář či projektový vedoucí se bedlivě ptají zadavatelů na funkce, které by měl mít očekávaný systém. Dokumentujeme cíle systému a rozkládáme je na dílčí podcíle. Na základě toho se v dokumentaci objeví i řada alternativních řešení. Identifikace samotných plánů vede ke stanovení závislostí v systému. Takzvané měkké, kvalitativní cíle zavádějí cestu k nefunkčním požadavkům. Metodologie Tropos má vlastní specifickou notaci. V této notaci odlišujeme měkké cíle od cílů globálních. Pomocí této notace určujeme i jakým způsobem jsou propojeny závislosti jednotlivých prvků systému, jako tomu je na Obrázek 12.



Obrázek 12: Notace prvků Tropos (Giorgini, 2004)

Po fázi předběžných požadavků následuje fáze konečných požadavků na systém. Tato fáze se zaměřuje více na zasazení modelovaného systému do prostředí. Specifikujeme jeho funkce a kvality. Řeší se provázanost systému s aktéry z předběžných požadavků.

Výsledkem tak je soubor funkčních a nefunkčních požadavků na systém. S fází konečných požadavků je svázán model strategického zdůvodnění požadavků neboli strategic rationale model. Tento model určuje na základě analýzy prostředků a cílů, jak mohou být nastavené cíle splněny s využitím všech definovaných aktérů. Tento model se skládá ze čtyř hlavních typů uzlů, kterými jsou cíle, úkoly, zdroje a měkké cíle. Tyto uzly jsou dekomponované cíle. Navíc jsou zde odkazy dvou typů. Means-ends link a odkazy rozkladu. Propojování uzlů mezi sebou je již otázka specifikací v rámci modelu agentů neboli Strategic Dependency. V grafu tak lze namodelovat vztahy mezi cíli jednotlivých agentů a závislosti mezi nimi, jak můžeme vidět na (Giorgini, 2004). Tento krok již spadá do fáze architektonického návrhu.



Tabulka 2: Vztahy cíli agentů (Giorgini, 2004)

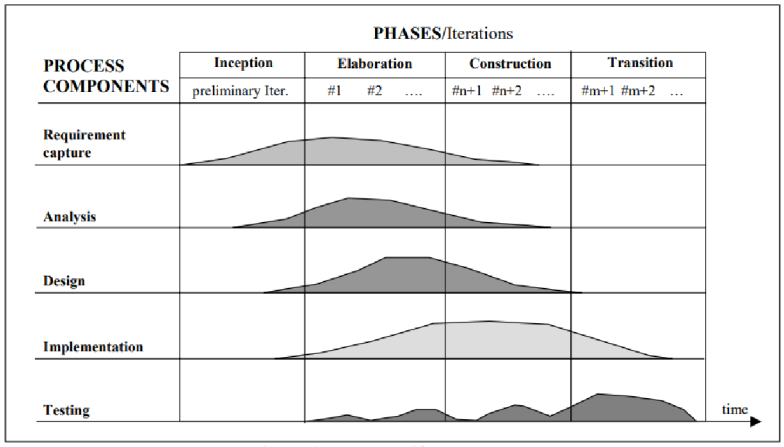
Fáze architektonického návrhu má za úkol určit globální architekturu systému a rozdělit tento celek na jednotlivé subsystémy. Tyto subsystému neboli aktory pak propojujeme závislostmi, které mohou mít datovou nebo kontrolní povahu. Architektonický návrh sestává ze tří kroků. Prvním je editování diagramu entit na základě požadavků. Druhým krokem je přidělení schopností aktérům a jako poslední určení sady komponent.

Po namodelování všech žádaných diagramů se vybírá platforma pro implementaci. Design jako takový bude přímo zasazen v rámci kódu. Kód je vyhotoven ve vybraném prostředí. Tropos nemá konkrétní pravidla v tomto ohledu.

3.14 MESSAGE

Metodika MESSAGE byla v prvopočátku vyvinuta převážně pro potřeby telekomunikačního průmyslu. Je vhodná pro tvorbu komplexních systémů s modulárními komponentami. V rámci této metodiky se ale nevylučuje ani použití pro menší systémy či různé organizace různé od telekomunikačních. Stejně jako spousta ostatních metodik má MESSAGE základy v RUP.

Tato metodika využívá pro jazykovou reprezentaci vlastní jazyk ale i UML. Vzhledem k značnému využívání UML v rámci multiagentních systémů je to tak velkým přínosem. UML využívá metamodel definovaný v rámci jazyka MOF. Jazyk v MESSAGE a UML tak mají stejný meta-modelovací jazyk a meta-modelové koncepty, které prohlubují znalostní úroveň. MESSAGE se skládá z pěti fází životního cyklu, jak můžeme vidět na Obrázek 14.



Obrázek 14: Fáze MESSAGE (Evans, 2001)

Vzhledem k návaznosti na RUP je v prvotní fázi kladen důraz na zjištění předběžných požadavků na systém. Již v této fázi si vývojář promýšlí případné entity v systému. Agent jako takový je vzhledem k návaznosti na MAS vnímán jako autonomní entita. Funkční vlastnosti a schopnosti tohoto agenta jsem zachyceny službou. Pojem služba má přitom návaznost na metamodel znalostí. Každý agent má motivaci plnit cíle neboli svůj účel. V rámci MESSAGE lze propojovat agenty do organizací a definovat role. Specialitou v rámci rolí je fakt, že je role vnímána odděleně od agenta. Jeden agent tak může mít více rolí a více agentů může využívat jednu roli.

Analytický model v MESSAGE je sestaven z několika pohledů na modelované společenství entit. Je tu pohled na organizační strukturu, pohled na cíle a úkoly agentů, pohled na jednotlivé agenty a interakční pohled. V rámci strukturování je stěžejní najít konzistentnost mezi těmito pohledy.

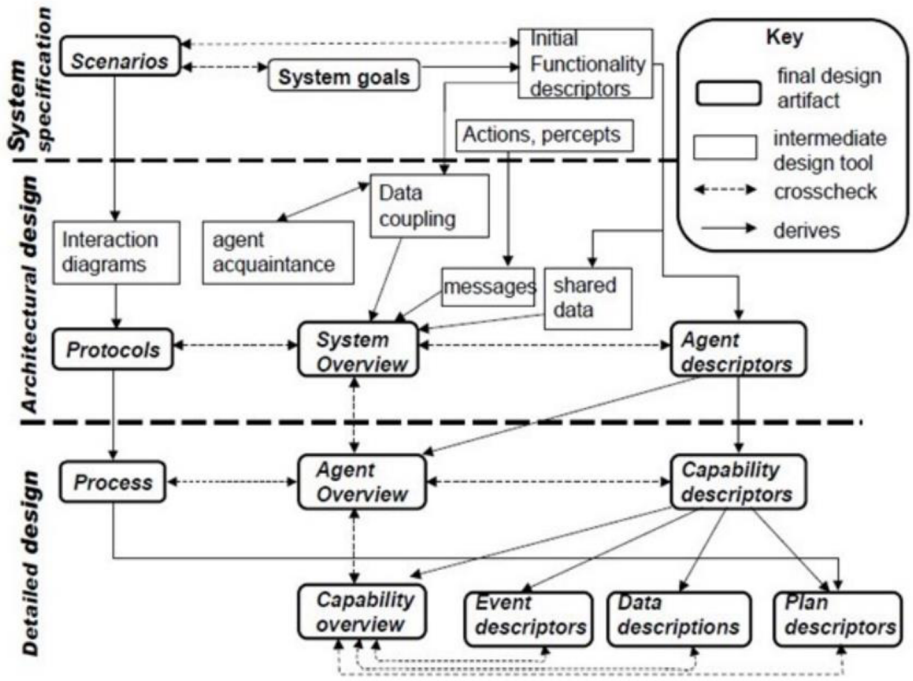
Fáze analýzy začíná na úrovni 0. Vývojář se dívá na systém jako na uzavřenou černou skříňku. Zkoumá se tak prvotně okolní prostředí a vztahy mezi entitami pomocí pohledu organizace. Metodika MESSAGE má vlastní notaci, která je odlišná od předchozích metodik, které jsou v práci uvedeny. V rámci analýzy na nulté úrovni se řeší i zobrazení cílů.

Po přechodu do úrovně analýzy 1 se modeluje pohled na organizaci, jejích požadované funkce a vnitřní struktura jednotlivých agentů. V MESSAGE je možnost zaznamenání dědičnosti mezi agenty. Posledními dvěma pohledy v této úrovni je interakční pohled a zobrazení domény.

V rámci procesu návrhu namodelované struktury se vzhledem k vytvořeným pohledům může vývojář zaměřit na dva přístupy návrhu. Jedním je přístup, který je spjatý s organizací MAS a vnitřní architekturou agenta a společenství. Druhý přístup je zaměřen více na platformu agenta a tříd.

3.15 Prométheus

Prométheus je metodika, která mnoho konceptů přejímá ze softwarového inženýrství. Základy této metodiky pocházejí z rukou Padghama a Winikoffa (Larioui, 2020). Je vhodná pro systémy středního rozsahu a v době svého největšího ohlasu se využívala převážně pro popsání systémů spjatých s topografiemi atd. Metodika Prométheus obsahuje tři stěžejní fáze: specifikace systému, architektonický design a detailní návrh viz. obrázek. Ve všech fázích je agent specifikován pomocí BDI náhledu.



Obrázek 15: Struktura Prométhea (Larioui, 2020)

Na Obrázek 15 můžeme vidět, že tato metodika pokrývá v rámci svých fází vše přes agentní modely, zprávy, role, protokoly po model cílů. V rámci Prométhea může vývojář modelovat systém pomocí platformy JADE, která je s metodikou Prométheus úzce spjatá. Návrhářským nástrojem je zde Prometheus Design Tool, což je grafický editor pro pokrytí konstrukčních úloh. Jsou uvedeny i různé postupy pro tvorbu konceptů atd.

4 Praktická část

V praktické části jsou uvedeny případové studie vybraných metodik ADELFE, GAIA, PASSI, MASSE a MESSAGE. V rámci případových studií jsou uvedeny i stěžejní diagramy a popis modelace pomocí dané metodiky.

4.1 Zdravotnický systém podle ADELFE

Metodika ADELFE byla využita pro projekt ze zdravotnictví, který nesl název GuardianAngel. GuardianAngel je projekt, který se specializoval převážně na chronická onemocnění pacientů, kteří trpěli diabetem, hypertenzí či závislostí na inzulinu. Na tomto projektu a na rámci hodnocení se podíleli převážně Yu a Cysneiros (2002). Realizovaný systém měl vykazovat stejné prvky jako mají centralizované systémy v nemocnicích. Měla zde být možnost ukládat zprávy z vyšetření, osobní údaje pacienta společně s jeho recepty. V rámci první interakce se zadavateli vývojáři sestavili soupis předběžných a konečných požadavků, které se měly se zadavatelem diskutovat. Na základě konzultace tak vznikly v předběžné fázi požadavků tyto funkční a nefunkční prvky.

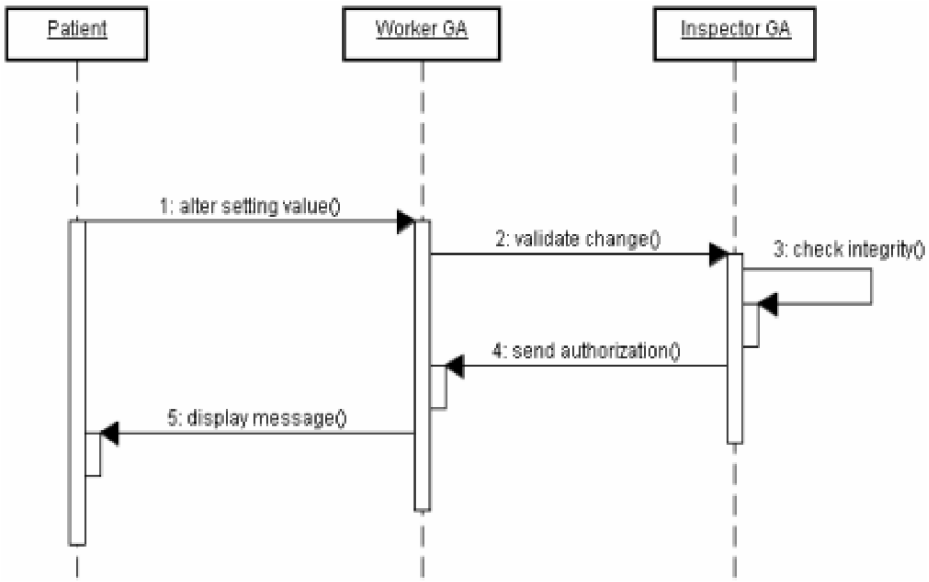
Tabulka 3: Předběžné požadavky ADELFE v případové studii, dle volného zpracování autorky

Předběžné požadavky	
Funkční	Nefunkční
Umožnění uživateli provádět dotazy v databázi.	Systém bude umět ukládat informace, které mají velkou datovou náročnost.
Monitorovat zdravotní stav pacienta.	Systém bude k dispozici 24 hodin denně po celý týden.
Pravidelné ověřování integrity dat.	Systém umožňuje odpovídat na několik dotazů najednou (multitasking).
…	…

Popis zkoumané domény je stěžejní také díky definování klíčových slov, která jsou vodítkem pro budoucí případy užití a třídy. Mezi klíčová slova patřilo: pacient, GA,

monitorování, zdraví, komunikace atd (Werneck). V rámci konečných požadavků se vytvořili již názvy konkrétních entit jako je knihovna, pobyt v nemocnici, nemoc atd.

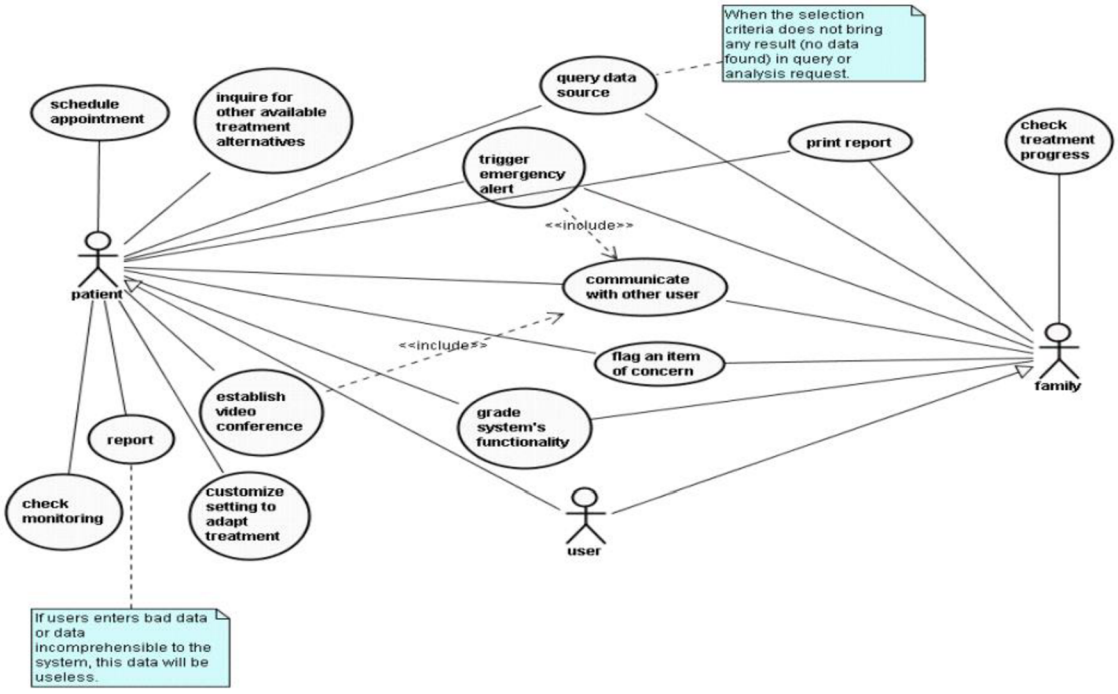
Po určení prostředí se definovaly Use Case diagramy, které vývojáři rozčlenili do skupin dle oblastí: pacient, instituce, administrativa, GA doména a služba. Následně byly vytvořeny sekvenční diagramy. Příklad sekvenčního diagramu, který je specializovaný pro oblast systému léčby pacienta, je na Obrázek 16.



Obrázek 16: Sekvenční diagram ADELFE pro případovou studii (Yu, 2002)

V rámci metodiky ADELFE je specifické zavádění uživatelského rozhraní UI a grafického rozhraní GUI. Grafické rozhraní obsahuje funkční a nefunkční prvky. Na obrázku je sekvenční diagram, který vznikl jako jeden z výstupu fáze předběžných a konečných požadavků.

Další fází vývoje je část analýzy, kde se vytvářel diagram tříd. Identifikovali se agenti a interakce mezi nimi prostřednictvím diagramů spolupráce a sekvencí. Na Obrázek 17 můžeme vidět přiklad kooperace mezi pacientem, uživatelem a rodinou.



Obrázek 17: Kooperace entit v ADELFE v rámci případové studie (Yu, 2002)

V části Domain Analysis se našly nové pasivní entity: terapie, pobyt v nemocnici, bakterie onemocnění a příčina onemocnění. V diagramu tříd pak byly tyto třídy: lidé, uživatel, pacient, rodina, zdravotník, opatrovník, lékař, zdroj dat, klinika, pojistitel, world wide web, knihovna, vláda, laboratoř, nemocnice, faktor, příčina onemocnění, terapie a pobyt v nemocnici.

Rámec hodnocení této metodiky, navrhnutý Cysneiros s Yu, byl důležitý vzhledem k praktickému otestování metodiky a ověření její schopnosti simulování reálných situací. V průběhu hodnocení se ovšem ukázalo, že ne všechny scénáře byly úplné. Metodika ADELFE úplně opomíjí zjišťování požadavků a či jejich změn v návaznosti na design. Metodice úplně chybí druh projektového managementu. Není zde možnost různých úrovní abstrakce či identifikací účastníků v doméně. Analýza domény tu sice má své malé místo, ale oproti jiným metodikám nezabírá takovou část, která by byla potřeba. Silnými stránkami ADELFE je ale rozhodně autonomnost uvažování, úvahy o různých nefunkčních aspektech a proaktivita.

Modelovaný projekt tak sice dospěl zdárného konce, ale v rámci vývoje chyběl obecnější přehled o systému a jeho okolí. ADELFE není koncipovaná pro sledování cílů agentů. V rámci dalších úprav na této metodice je tak prostor pro vylepšování fáze požadavků, zlepšení fáze nasazování a přidání lepších mechanismů pro sledování systému.

4.2 Systém objednávek podle GAIA

V roce 2014 byl v rámci metodologie GAIA realizován projekt SFC neboli shop floor control, který rozebírali Araúzo a Olmo v rámci své studie (Araúzo, 2014). Tento systém je využívaný k odesílání, plánování a kontrole objednávek. Tyto objednávky si daný závod vytvářel sám. V prvotní fázi zjišťování požadavků obdrželi vývojáři požadavky na zohlednění technologických možností a kapacit strojů, možnost operací s položkami a možnosti volby přednostního zpracování, a nakonec možnost evidovat u každé objednávky datum splatnosti. Systém měl být schopný přidělovat a plánovat úkoly na konkrétních strojích, reagovat na poruchy a sledovat celkový stav závodu. Přidělování zakázek ke konkrétními stroji mělo probíhat na základě dostupnosti a účinnosti konkrétních strojů v nabídce. V rámci tohoto projektu se přidělování koná na základě aukce, a ne tradičním způsobem.

Na základě požadavků, které byly zjištěny od zadavatele, vývojáři identifikovali čtyři externí a devět systémových rolí v systému. Tyto role jsou popsány v Tabulka 4: Model rolí v rámci GAIA

.

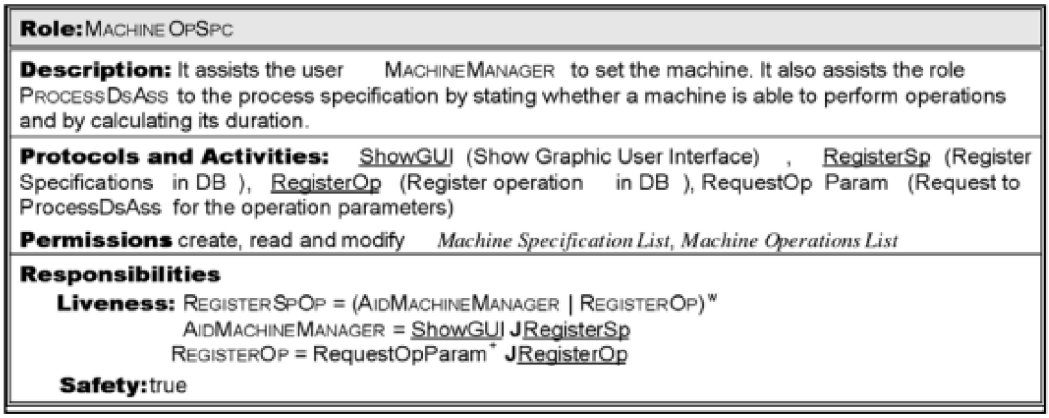
Tabulka 4: Model rolí v rámci GAIA (Araúzo, 2014)

Role	Popis
MACHINEMANAGER	Nastavuje strojní parametry: kapacitu stroje, technické možnosti a specifikace.
PROCESSDESIGNER	Nastavuje produkty, které se mohou v systému vyrábět a pro každý produkt navrhuje výrobní procesy (operace a vztahy).

PLANNER	Definuje a odesílá objednávky. Analyzuje efektivitu systému.
SHOPFLOORMANAGER	Nastavuje a monitoruje systém.
MACHINEOPSPC	(Machine Operation Specification) Pomáhá uživateli MACHINEMANAGER nastavovat stroj. Pomáhá i PROCESSDSASS tím, že uvádí, zda je stroj schopen provádět operaci a zadává dobu trvání operace.
PROCESSDSASS	(Process Design Assistant) Pomáhá definovat proces výroby. Žádá roli MACHINEOPSPC o informace o možnosti provádění operací na strojích a době trvání operací.
PLANNERASS	(Planer Assistant) Pomáhá uživateli PLANNER definovat výrobní zakázky. Vypočítává parametry pro měření výkonu systému.
ORDERDISPATCHER	(Order Dispatcher) Odesílá objednávky definované PLANNEREM.
DISPLAYER	Zobrazuje stav systému (objednávky a stav stroje) a jeho vývoj.
PRICECALCULATOR	(Order Manufacturing Manager). Vypočítává cenu využití zdrojů v rámci jednotek horizontu plánování.
LOCALSCHEDULER	Plánuje operace objednávek podle cen, které vypočítal uživatel PRICECALCULATOR.
ORDMANUFMANAGER	(Order Manufacturing Manager). Řídí provádění operací každé výrobní zakázky podle harmonogramu, který navrhuje role LOCALSCHEDULER. Když je třeba

	provést operaci, tato role požádá roli OPEXMANAGER o provedení. Tato role je také zodpovědná za dodržování prioritních omezení během fáze provádění.
OPEXMANAGER	(Operation Execution Manager). Je odpovědný za provedení operace. Ovládá fyzicky stroj.

Všechny tyto role by měly mít popis oprávnění, odpovědností, aktivit a protokolů. Vizuální znázornění takového popisu můžeme vidět na obrázku, který popisuje tyto parametry u role typu MACHINEOPSPC.



Obrázek 18: Konkrétní pospi role v GAIA

Po určení oprávnění, odpovědností, aktivit a protokolů se v rámci GAIA metodologie přechází k fázi designu, kde se modeluje model agenta, model služeb a model spolupráce. V Tabulka 5 jsou znázorněny role, které jsou přiřazeny k agentům.

Tabulka 5: Znázornění protokolů a iniciátorů (Araúzo, 2014)

Protokol	Iniciátor/Respondent	Záměr iniciátora
RequestOpParam	MACHINEOPSPC/PROCESSDSASS	Znalost parametrů, které definují operaci k návrhu provedení a výpočet doby procesu.

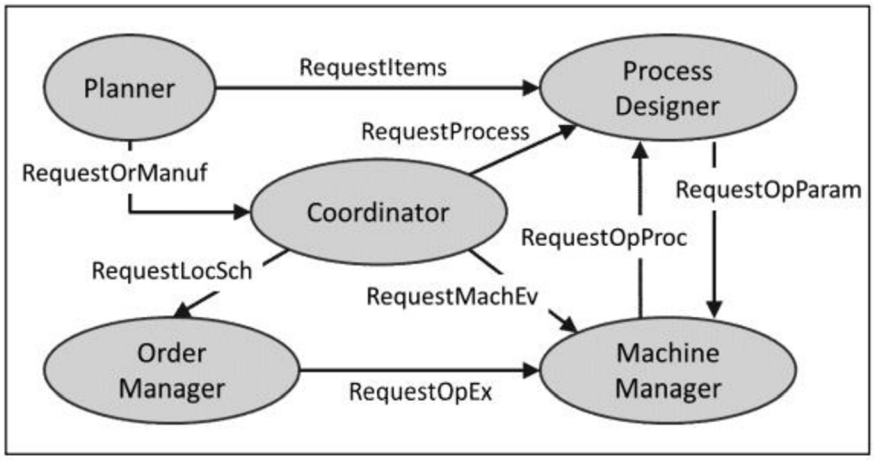
RequestOpProc	PROCESSDSASS / PROCESSDSASS	Generování procesu provádění operace ve stroji, zná dobu procesu.
RequestItems	PLANNER / PROCESSDSASS	Znát položky s definovaným procesem vytváření výrobní zakázky.
RequestOrManuf	PLANNER / ORDERDISPATCHER	Zavádění ORDMANUFMANAGER pro správu výrobní zakázky.
RequestProcess	ORDERDISPATCHER/ PROCESSDSASS	Zná výrobní proces objednávky pro vytvoření ORDMANUFMANAGER.
RequestOpEx	ORDMANUFMANAGER/ OPEXMANAGER	Provede operaci.
RequestLocSch	PRICECALCULATOR/ LOCALSCHEDULER	Podle navržené metody plánování se vypočítávají ceny za použití zdrojů. S tímto protokolem dostane role PRICECALCULATOR plány.
RequestMachEv	DISPLAYER / OPEXMANAGER	Zná změny stavu strojů pro sledován systému. DISPLAYER musí znát čas zahájení operace a čas závěru.

Po vyjádření modelu interakcí je nutné vytvořit i model agentů, který je zachycen v tabulce. V Tabulka 5jsou k agentům uvedeny i možné role, které mají pro přiřazení specifická pravidla. Používají se zde kritéria v rámci interakcí s konkrétní uživatelskou rolí či umístění ve stejné síti, kde mají přístup ke stejným informacím.

Tabulka 6: Model agentů v GAIA (Araúzo, 2014)

Agent	Počet instancí	Role
Planner	1	PLANNERASS
Process Designer	1	PROCESSDSASS
Coordinator	1	DISPLAYER ORDERDISPATCHER PRICECALCULATOR
Order Manager	Počet objednávek v systému	ORDMANUFMANAGER LOCALSCHEDULER
Machine Manager	Počet strojů	OPEXMANAGER MACHINEOPSPC

Jako poslední model, který je v rámci GAIA pro systém SFC realizován, je model „známostí“ neboli acquaintance model, který je znázornění na Obrázek 19: Acquaintance model v GAIA .



Obrázek 19: Acquaintance model v GAIA (Araúzo, 2014)

V konečné fázi se v rámci projektu řešilo zpracování scénářů. Celý projekt se implementoval pomocí JADE. Návrháři hodnotili konečný projekt jako agilní a robustní. Při následném testování se dospělo k výsledku, že navrhovaný postup byl pro systém vhodný a výsledky jsou tak kladné. Daný přístup funguje v navrhovaných scénářích správně a celkový přístup zajistil vysokou flexibilitu.

Metodologie GAIA je koncipovaná do to-down přístupu. Tomu také odpovídá celý koncept analýzy a designu. Při modelování si vývojář může povšimnou určitých nedostatků,

které jsou stěžejní při rozhodování, jakou metodologii použít pro určitý druh modelované domény.

První nedostatek může analytik pocítit již při modelování jednotlivých agentů nebo respektive tříd agentů. Při tvorbě jsou zde patrné velké podobnosti mezi agentem a objektem a souborem atributů a metod, což není vhodné vzhledem k příliš jemné abstrakci, která může být umístěna do nevhodné úrovně. Vzhledem k této podobnosti nevykazuje objektový model mnoho užitečných informací. Úplně se odprošťuje například od dědičnosti či agregace, takže je model ochuzen o část důležité reprezentace. Wooldringe uvádí (2000), že má tento problém původ ve faktu, že je paradigma agenta založeno na výrazně silnějším pojetí zapouzdření než na paradigma objektu. Co se týká chování agenta, často není při vykonávání akce oznámeno, že jde agent nějakou akci vykonávat. Správně by měla vzniknout dohoda, jak bude daný úkol proveden. Z tohoto pohledu se tak komunikační kanál chová asynchronně. Agent může kdykoliv iniciovat interní či externí chování.

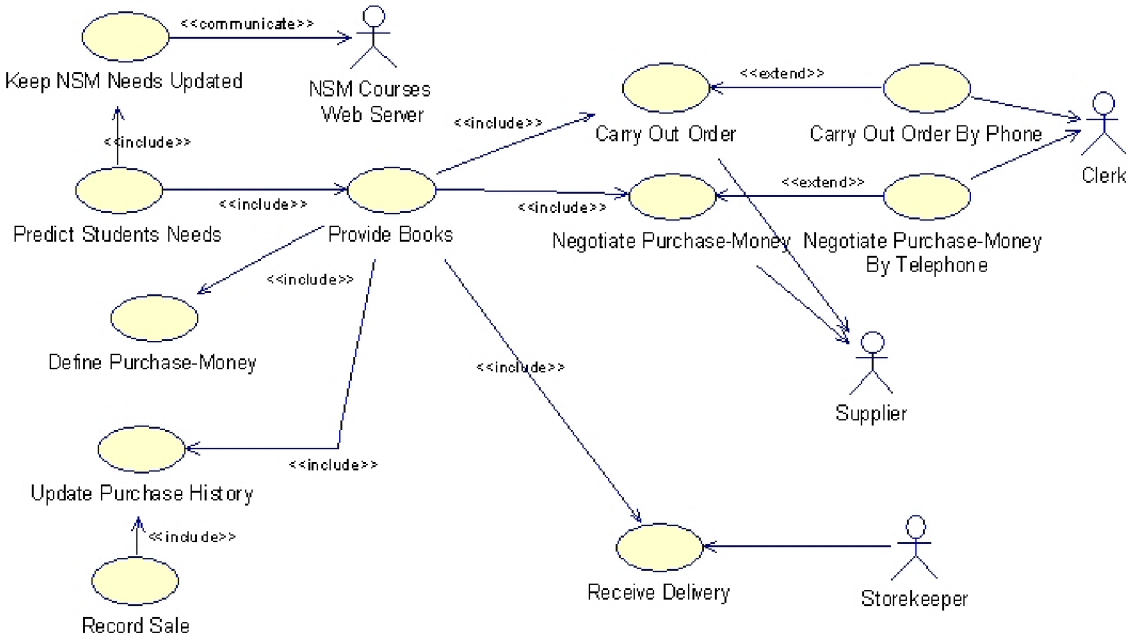
Jedním z problémům pro budoucí práci s GAIOU je i fakt, že se tato metodologie nepokouší zachytit práci agentů na společných cílech. Multiagentní systémy by ke správnému fungování měly obsahovat funkcionality pro naplňování společných cílů agentů. Tento nedostatek se tak jeví jako vážný problém pro budoucí práci. S tím také souvisí jistá ochuzenost v zastoupení protokolů spolupráce mezi agenty.

Metodika GAIA má ovšem dobře vyvinutou sémantiku, která je jednoznačná a formální. Nejedná se tak o metodologii, která by měla pouze pragmatickou hodnotu. GAIA nebyla navržena na základě konkrétních standardů pro komunikaci agentů, jako je například komunikační jazyk fipa atd. Tento fakt můžeme vnímat jako prostor pro přizpůsobení, aby byla s danými standardy kompatibilní.

4.3 Knihkupectví podle PASSI

Jackson (2001) ve své knize specializující se na popis problémových fází vývoje softwaru studoval metodiku PASSI postupně v rámci všech jejích diagramů. Celý postup modelování koncipoval na rámec knihkupectví. V rámci této publikace se zaměřil převážně na jeden scénář. Detailněji popisoval, co se stane pokaždé, když knihkupectví dokupuje do svého sortimentu knížky ze skladu.

V rámci zjišťování předběžných požadavků na systém vedení knihkupectví uvedlo, že se obchod specializoval převážně na knihy pro vysokoškolské studenty. Rozhodování o nákupu nových knih tak začíná každý semestr, nebo když určitá fakulta změnila požadovanou literaturu k předmětu. Změnil se tak například u konkrétního předmětu stav knihy z doporučené na „povinné“. Knihkupectví se tak snaží předvídat potřeby studentů. Zároveň si přeje uchovávat předchozí nákupy a dodavatele, a mít celkový přehled o objednávce.

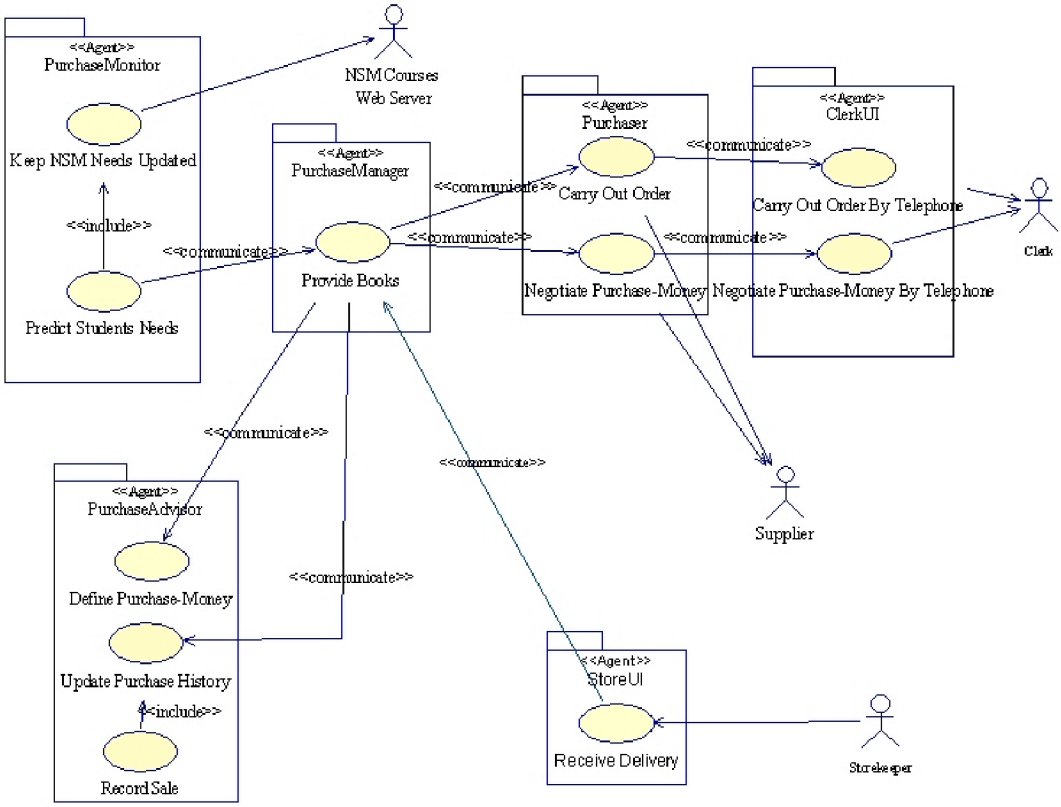


Obrázek 20: Diagram požadavků (Jackson, 2001)

Za pomoci diagramu požadavků tak vznikl tento nákres. Diagramy u metodiky PASSI jsou vytvořené na základě UML. V rámci diagramu požadavků UML zadáváme uživatele, kteří systém používají v návaznosti na funkce, které zde mohou provádět. V obrázku lze doplnit i jaké aktivity mohou nahlížet do okolních v rámci vztahu <<include>> a <<extended>>. Nárok na uchovávání informací je zabalen do Define Purchase-Money use case. Zákazník si přál celkový přehled o objednávce. Na základě toho je v diagramu vytvořená aktivita pro vyjednávání o ceně, realizace objednávky atd. V rámci modelu požadavků může návrhář doplnit model o kardinality.

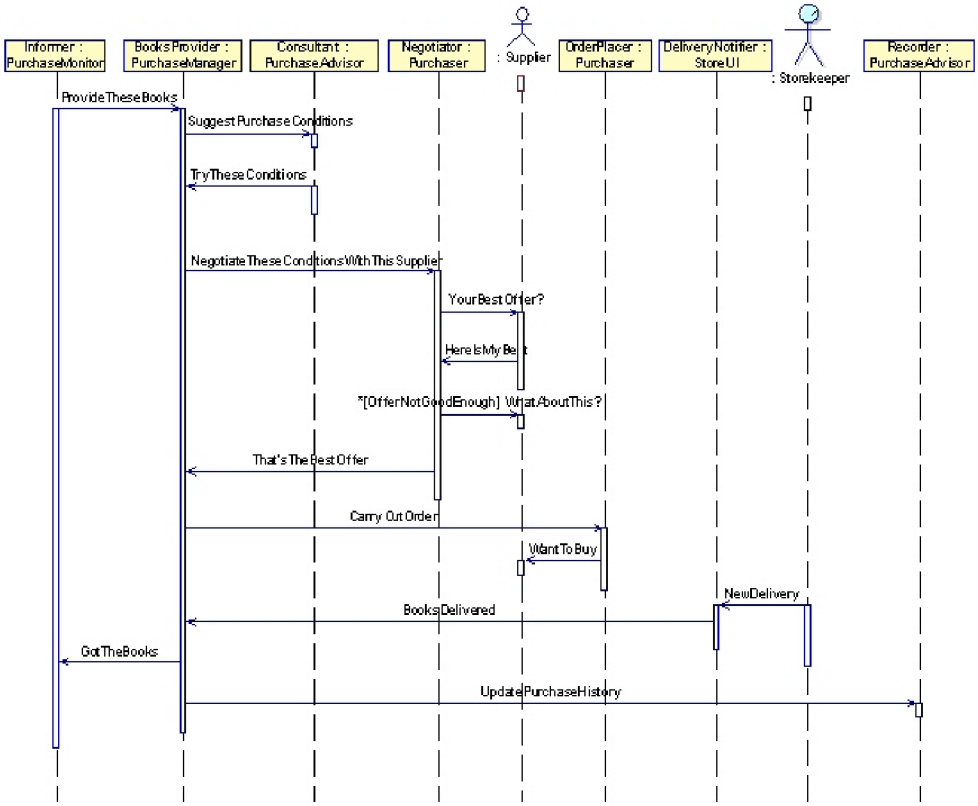
Identifikace agentů začala u sestavení případů užití pomocí diagramu Use Case v předchozím kroku. Jako další fáze zdokonalování diagramu agenta je tak zabalení entit

použitých v Use Case do balíčků. Každý tento balíček definuje funkce konkrétního agenta. V rámci tohoto diagramu se změní i typ vztahu na <<comunicate>>.



Obrázek 21: Use Case a balíčky v PASSI (Jackson, 2001)

V rámci úkolu ProvideBooks a Receive Delivery dochází ke změně směrovosti. V oblasti funkčnosti v organizaci tato změna dává smysl z pohledu manažera, který se nemusí stále obtěžovat s dodáním věcí na sklad. Může tak pokračovat v práci a nezabývat se informacemi o doručení. Ve fázi analýzy požadavků se dále řeší chování agenta. Definují se tak jeho role, kdy se prozkoumávají cesty interní komunikace. Každá komunikační cesta obsahuje scénář, který se znázorňuje pomocí sekvenčních diagramů. Diagram na obrázku č. 19 je velmi podobný znázornění pomocí UML. Syntaxe je ale jiná. Pro zápis se využívá syntaxe <role_agenta>:<jméno_agenta>.



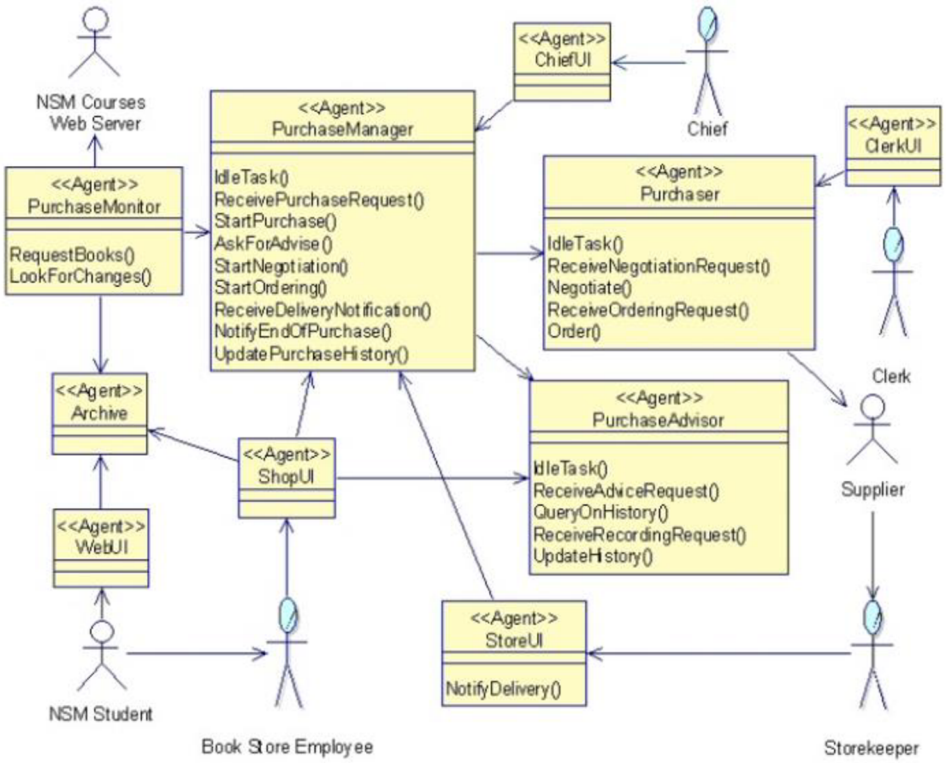
Obrázek 22: Sekvenční diagram PASSI (Jackson, 2001)

V rámci PASSI může agent přistupovat k různým scénářům a v každém tomto scénáři mít odlišné role. Sekvenční diagram v této metodice začíná vždy akcí konkrétního agenta. Může se tak spustit scénář a ostatní aktéři do něj mohou vstoupit později. Zprávy, které jsou vidět na obrázku, vyznačují nejen komunikaci mezi rolemi, ale mohou znázorňovat i vnější událost. Zprávy, které se v rámci tohoto diagramu vytvářejí, obsahují data a údaje o tom, co má agent vykonat či poskytnout.

Po návrhu diagramu Use Case se v rámci kolektivního fungování ve skupině řešil návrh úkolů a jejich provázanost v rámci chování celku. Tyto úkoly zahrnují funkce, které ve výsledku tvoří celkový obraz fungování společenstva. Pro každou entitu modelu se tak vytvořil diagram specifikace úloh. V rámci metodik je takovýto popis úloh nadstandartní.

V dalším popisování vnitřní struktury knihkupectví se vytvořil diagram ontologie. Diagram ontologie v tomto případě zobecňuje vztah mezi dvěma agenty a určuje asociace vztahů mezi těmito prvky. Ještě více se tak vyjasňuje struktura systému. V tomto diagramu lze najít i prvky agregace, které jsou použity v rámci výskytu množin.

Metodika PASSI obsahuje více diagramů, které specifikují strukturu nejen agenta samotného, ale i celkový multiagentní systém, jak můžeme vidět na Obrázek 1.



Obrázek 23: Diagram tříd v PASSI (Jackson, 2001)

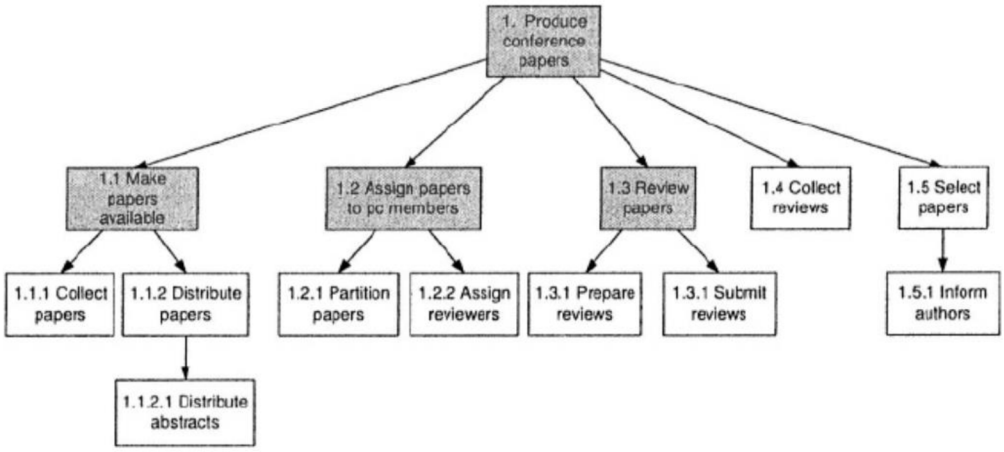
V rámci tvorby knihkupectví se v rámci PASSI vytvořil kvalitní návrh systémové struktury. Dosáhlo se tak požadovaného výstupu a navržená struktura byla konzistentní. Metodika PASSI je hojně využívaná vzhledem k notaci UML. Diagramy, které jsou zde využívány, jsou přehledné a díky logickým balíčkům a strukturám je tak celkový návrh modelovaného systému intuitivní. PASSI je založena na principu koherence a koheze. Pakliže tedy při vývoji specifikace agenta nepřinese tížený výsledek, je zde možnost změny směrovosti atd. Tento fakt s sebou nese velkou flexibilitu systému. Velkou výhodou je detailní zaměření na rozbor agentů, jejich role a specifika. Tato metodika řeší nejen agenty jako jedince, ale i jejich chování ve společenstvu, což je pro tvorbu multiagentních systémů žádoucí. Metodika PASSI se jako jedna z mála zabývá ontologickou částí vývoje.

4.4 Konference podle MASE

Scott Deloach v rámci své knihy (Bergenti, 2004) zkoumal aplikaci metodiky MASE pro systém realizace mezinárodních konferencí. Systém měl být schopný posílat podklady

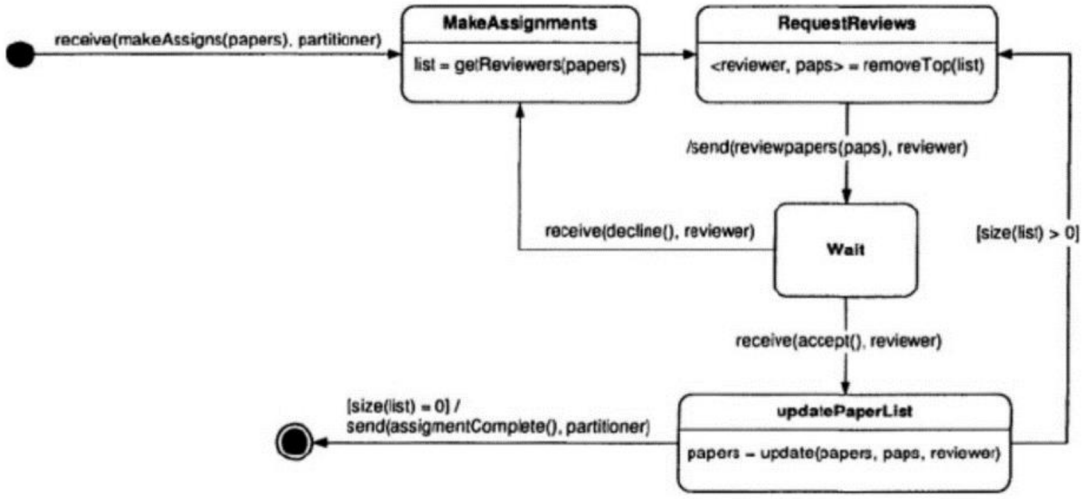
pro konferenci elektronicky a následně ukládat tyto podklady a články do databáze. Účastníci konference by měli mít možnost vytvářet recenze a odesílat je na sběrné místo v databázi. V rámci systému se bude řešit členství a příspěvky za toto členství.

V návaznosti na těchto požadavcích se jako první řešila hierarchie cílů.



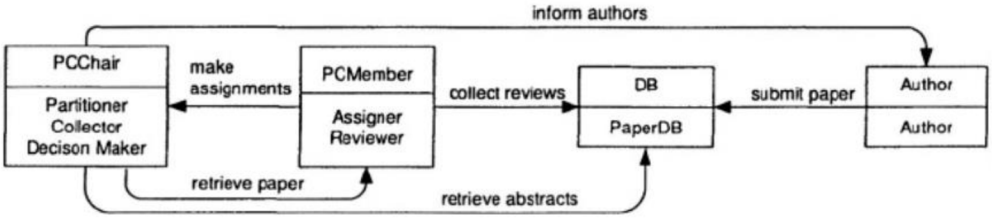
Obrázek 24: Dekompozice cílů v rámci MASE (Bergenti, 2004)

Na Obrázek 24 je vidět postupná dekompozice. Jako další se vytváří diagram případů užití a sekvenční diagramy. V návaznosti na tento diagram se cíle komponují do rolí, které se znázorňují do modelu rolí a určují se komunikační cesty. Ve fázi analýzy se jako poslední určovaly souběžné úlohy pro každý úkol, jako tomu je na Obrázek 25.



Obrázek 25: Úkoly v rámci MASE (Bergenti, 2004)

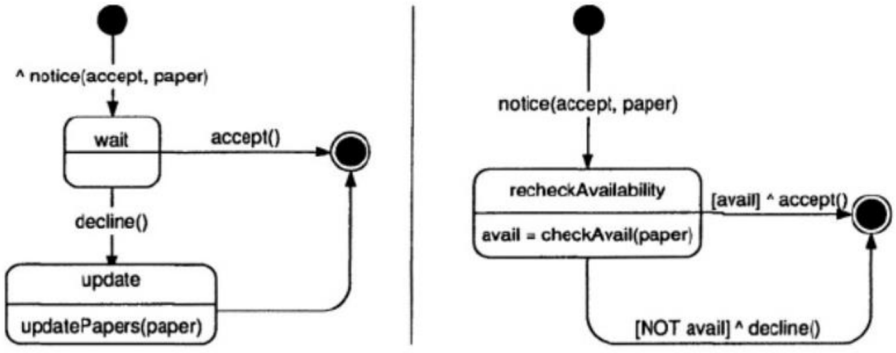
Metodika MASE obsahuje čtyři kroky ve fázi designu. V prvním kroku se vytváří diagram tříd agenta. V tomto smyslu je třída šablonou pro agenta. V příkladu konference se v systému pohybují čtyři stěžejní agenti a PCChair, PCMember, DB a Author. Všem třídám jsou přiřazeny role a asociace, jak je vidět na Obrázek 26.



Obrázek 26: Přidělení rolí v rámci MASE (Bergenti, 2004)

Vytváření konverzací mezi agenty je krokem, který musí vývojář řešit hned po sestavení diagramu tříd. Konverzace jsou definovány pomocí komunikačních diagramů, do kterých vstupuje vždy iniciátor a respondent. Konverzace obsahuje koordinační protokoly, které se nesmějí v tomto ohledu opomíjet. Příklad komunikačního protokolu je na obrázku.

Konkrétní úkol lze přitom mapovat na jednu konkrétní konverzaci. Souběžné úkoly ale mohou konverzací vytvářet více. V rámci struktury se doplní i vnitřní struktura agenta a vývojář může přejít ke konečné struktuře systému pomocí diagramů nasazení.



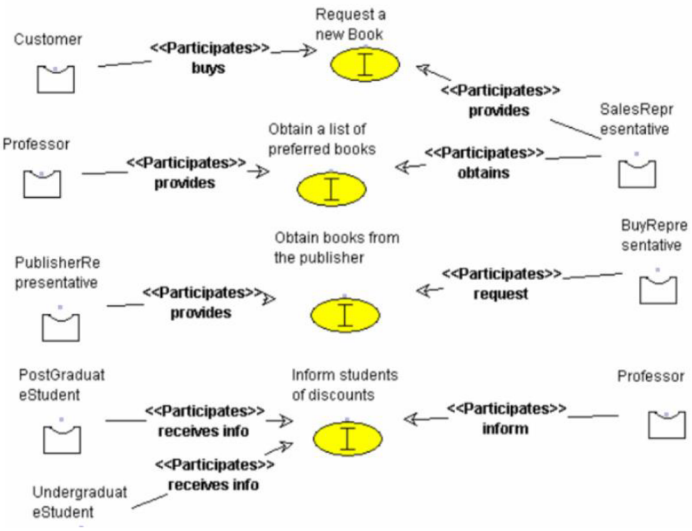
Obrázek 27: Diagram nasazení v MASE (Bergenti, 2004)

Metodika MASE má pro vývojáře velký přínos vzhledem k pokrytí celého životního cyklu pro analýzu, návrh a vývoj. MASE obsahuje spoustu graficky založených modelů, které mají původ v UML, takže je celková kompozice velmi přehledná. Na základě MASE vzniklo mnoho projektů. Bohužel postrádá rozšířenější ontologické prvky a více autonomní generování kódu.

4.5 Knihkupectví podle INGENIAS

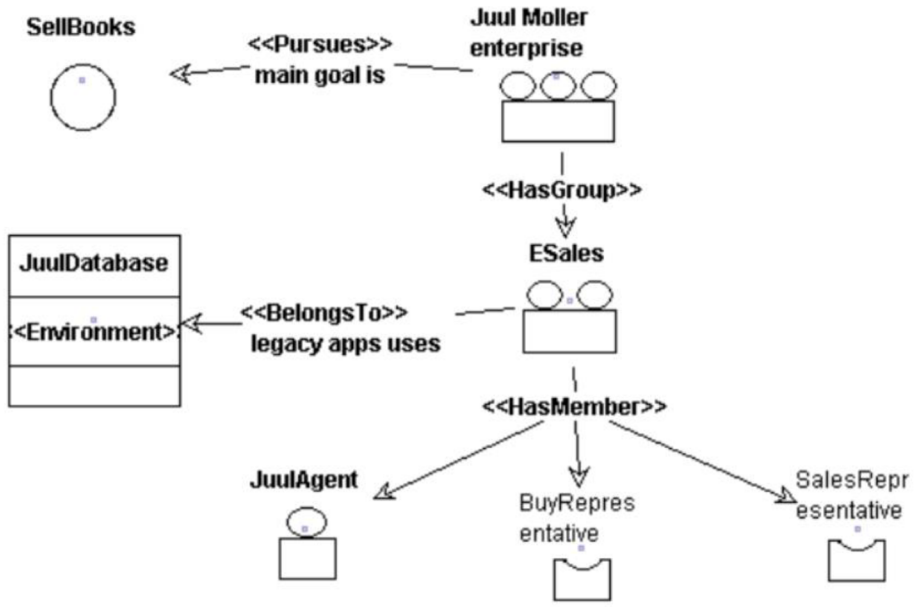
Strukturou metodiky INGENIAS se zabýval i Andreson (Andersen, 2001) v rámci popsání knihkupectví Juul Møller Bokhandel A/S. Knihkupectví se zabývá prodejem knih vysokoškolským studentům. Vedení knihkupectví má přehled o knihách, které studenti potřebují, a tak je zařazuje do svého sortimentu na stálo či nárazově s ohledem na požadavky studentských kurzů. Samotný prodej může probíhat konvenčně v rámci pobočky či na internetu. S ohledem na tento fakt je tak knihkupectví modelováno do dvou částí: logistika a eObchod.

V prvotní fázi dochází k popsání stěžejních funkcí na systém v rámci případů použití. Hlavní funkcionalitou je prodání knihy, doplnění knih na sklad od dodavatele a zajištění dostupnosti těchto knih. Do systému by mělo jít přidávat slevy a na webových stránkách se musí zobrazovat dostupnost knih. Diagram případů užití je znázorněný na Obrázek 28. V rámci knihkupectví jsou čtyři případy užití: žádost o novou knihu, získání seznamu preferovaných knih, získání knih z nakladatelství a informování studentů o výprodeji.



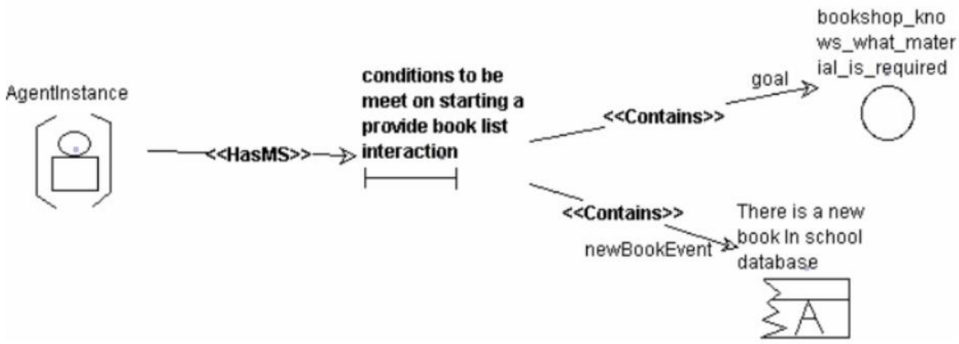
Obrázek 28: Případy užití (Bokhandel, 2006)

V návaznosti na případy užití vznikl počáteční návrh struktury, který je vidět na Obrázek 29.



Obrázek 29: Struktura knihkupectví v INGENIAS (Andersen, 2001)

Na Obrázek 29 je patrné značení v rámci INGENIAS. Samotné nakladatelství je označeno obdélníkem se třemi kruhy na vrchní hraně. Jednotlivé skupiny organizace, jako je odbor distribuce prostřednictvím webu, jsou označeny obdélníkem se dvěma kruhy na vrchní hraně. Cíle organizace, jako je prodej knih, jsou vyznačeny obyčejným kruhem bez výplně nebo jiného přidaného symbolu. Pracovní postupy, které označují složitější vnitřní strukturu, jsou označeny pomocí spojených elips. V návrhu jsou symboly čtverců s vrchním zakřivením. Tento symbol označuje role.



Obrázek 30: Duševní stav v rámci INGENIAS (Andersen, 2001)

Z počáteční fáze analýzy se vývojáři přesunuli do fáze designu. Z počátku se zde zkoumají prototypy a konkrétní aspekty problémových oblastí a jejich testování na různých platformách, jako je JADE atd. V pozdější fázi designu dochází k přeměňování případů použití a případně se objevují nové. Tyto případy použití se zpřesňují a spojují se s konkrétními interakcemi. Zavádí se model pro vztahy mezi cíli a úkoly v rámci entit. Jako jeden z posledních kroků v rámci designu je nastavení duševního stavu entit. Na Obrázek 30 je znázorněno nastavení duševního stavu, který je potřebný pro zahájení interakce pro získání seznamu knih.

Poslední fází metodiky INGENIAS je fáze implementace, která byla v případě knihkupectví realizovaná na platformě JADE.

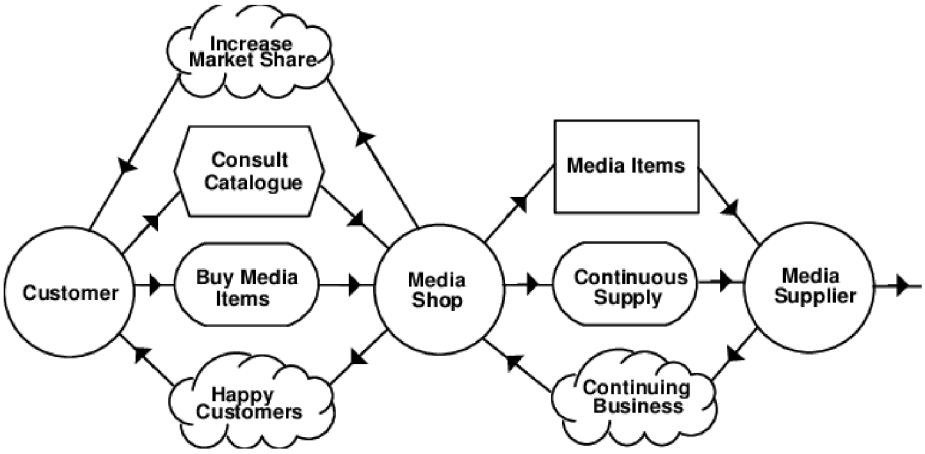
Metodika INGENIAS vedla k úspěšné tvorbě vnitřní struktury knihkupectví. Vývojáři při práci ocenili možnost analýzy, designu, testová i následné implementace kódu. INGENIAS vhodně zasazuje cíle do modelů a rozmanitě popisuje interakce entit pomocí diagramu. Výhodou je možnost konvertovat vybrané diagramy do XML. INGENIAS se od ostatních metodik liší v oblasti koncepčního rámce, kde umožňuje rozdělit výzkumné přístupy. Pomocí meta-modelu lze tyto přístupy rozšířit i v průběhu o další.

Nedostatky v rámci této metodiky je oblast algoritmů. INGENIAS tyto algoritmy zapouzdřuje na úkoly a popisuje pouze vstupy a výstupy. Řešením by tam mohlo být pouze vytvoření vlastních algoritmů a následné zasazení do systému. Další nevýhodu je komplikované generování kódu a složité zajišťování soudržnosti v rámci správnosti specifikací systému.

4.6 Media Shop podle Tropos

Giorginy a spol. se ve své práci (2004) zabývali zkoumáním vhodnosti použití metodiky Tropos pro Media Shop. Media Shop je obchod, který se zaměřoval na prodej různých video kazet, CD a ostatních mediálních položek. Tento ochod využívá v rámci svého působení

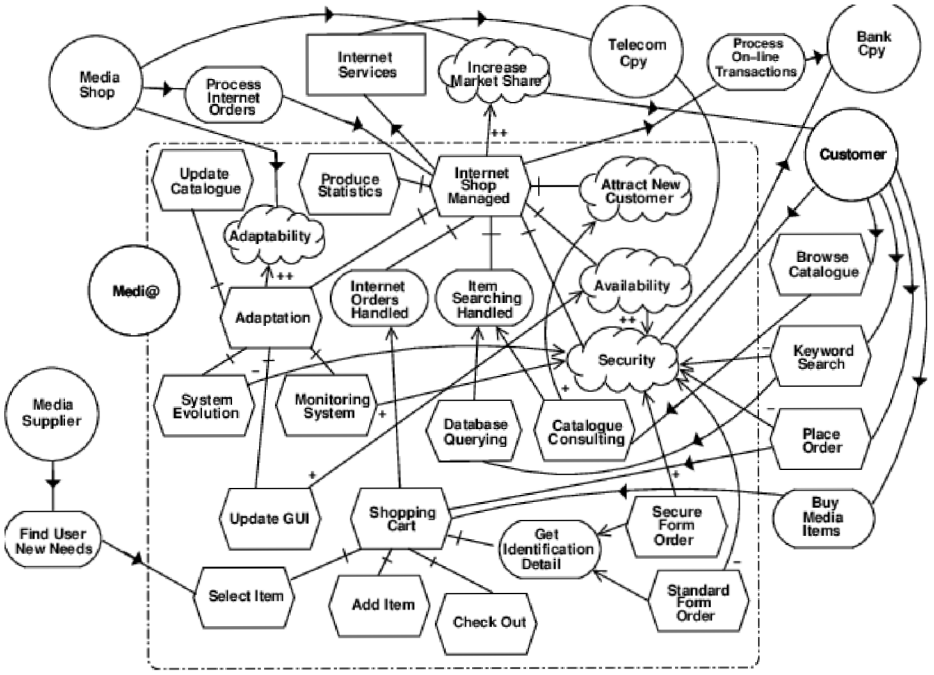
katalog dostupných položek Media Supplier. Vedení Media Shopu se snažilo o expanzi na trh, a tak došlo k prvotnímu nápadu vytvoření internetového eshopu, pro distribuci zboží.



Obrázek 31: I model pro Media Shop (Giorgini, 2004)*

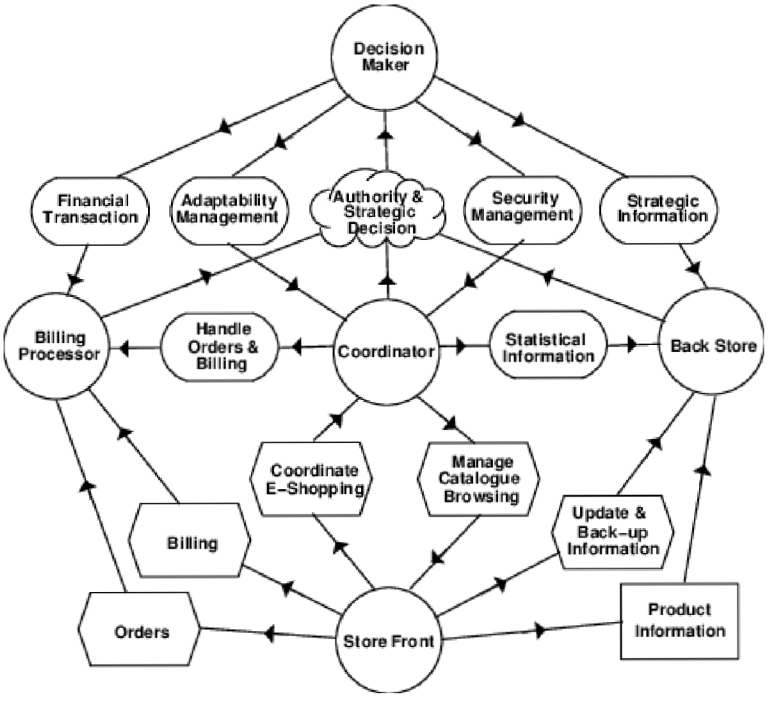
Na základě konzultace se zadavatelem se definovaly hlavní entity systému a to: zákazník, obchod a dodavatel. V návaznosti na notaci Tropos jsou zde uvedeny úkoly, zdroje a měkké cíle entit. Typem zdroje jsou v tomto případě mediální položky, cílem je nákup těchto položek a měkkým cílem spokojenost zákazníka. Všechny znázorněné aspekty mají definované závislosti. Tento model ukončuje předběžné požadavky na systém.

Fázi konečných požadavků zakončuje Strategic Rationale Model, který je zobrazen na Obrázek 31. Model navazuje na i* model vytvoření v prvotní fázi požadavků a analyzuje, jak mohou být nastavené cíle splněny s použitím všech zúčastněných aktérů.



Obrázek 32: Strategic Rationale Model v rámci Tropos (Giorgini, 2004)

V této fázi se mohli vývojáři plně soustředit na popis a zavedení konkrétních tříd systému. Třídám doplnili atributy a metody. V rámci ulehčení další fáze můžeme v Tropos exportovat tuto strukturu pomocí vložených nástrojů pro zjednodušení kompletace fáze architektury, která je zobrazena na Obrázek 33.



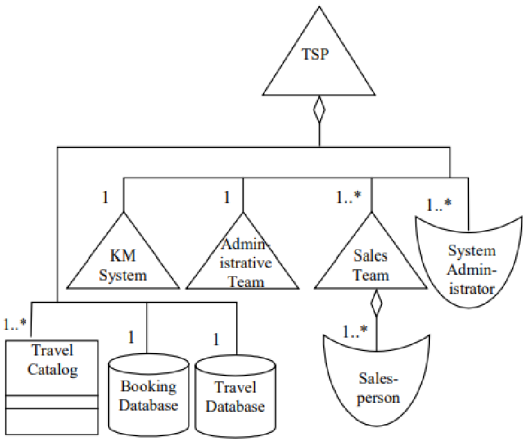
Obrázek 33: Struktura konference v rámci Tropos (Giorgini, 2004)

Specifikací této metodiky je tvorba takzvaných sociálních vzorů. Tyto sociální vzory určují, jak mají být cíle delegovány na konkrétní aktéry systému. Bohužel jsou tyto vzory znatelné převážně ve fázi implementace. Cíle je zde možné znázornit do modelu, který lze vytvářet automatizovaným způsobem, kdy se vztahy AND a OR přiřazují na základě algoritmu.

V rámci komplementace návrhu Media Shopu vývojáři uvítali orientaci metodiky převážně na agenty a jejich cíle. Je zde kladen velký důraz na analýzu požadavků, což je žádoucí vzhledem ke snaze předejít nesrovnalostem mezi zadavatelem a vývojářem. Nevýhodou v rámci projektu bylo, že Tropos postrádá nástroje, které podporují přechody mezi jednotlivými fázemi vývoje. Media Shop byl navíc projekt, který měl prvky multiagentního systému, ale nejedná se o rozsáhlý systém. Metodika Tropos není vhodná pro rozsáhlejší projekty.

4.7 Aplikace pro ulehčení cestování dle MESSAGE

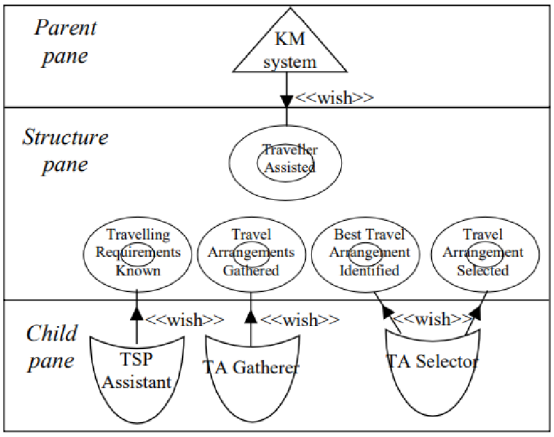
V rámci projektu EURESCOM P907 vznikla pro popsání fungování metodiky MESSAGE aplikace Universal Personal Assistant for Travel (Evans, 2001). Tento systém měl sloužit jako elektronický osobní asistent pro pomoct s cestovními aktivitami typu: letenky, hotely a ostatní cestovní služby. V předběžných požadavcích na systém tam byla možnost rezervace různých lístků, pomoc v rámci registrace do cestovních služeb, zobrazování upozornění ohledně vytíženosti plánované trasy atd. Vývojáři začali v rámci analýzy vytvářet organizační model, ve kterém byly zavedeny i strukturální vztahy.



Obrázek 34: Organizační model v MESSAGE (Evans, 2001)

Na Obrázek 34 je například vidět systém KM, který má dvě role a spolupracuje s dvěma externími zdroji. Na této úrovni se bere v potaz i návaznost na budoucí databáze. Na úrovni 0 analýzy se vytvářeli v rámci cestovní aplikace i diagramy implikací a pracovních postupů, do kterých se nastínila dědičnost a směrovost.

V rámci analýzy úrovně jedna je v MESSAGE zvláštnost v podobě diagramu struktury delegování, který můžeme vidět na obrázku.



Obrázek 35: Diagram struktury delegování (Evans, 2001)

Na základě sestavování diagramů pro cestovní systém se ukázalo, že je metodika MESSAGE vhodná pro rozsáhlejší projekty vzhledem ke všem svým modelům, které umožňují specifikovat různé oblasti systému pro co největší přehlednost.

4.8 Porovnání jednotlivých metodologií

V praktické části byly uvedeny metodiky ADELFE, GAIA, PASSI, MASE, INGENIAS, TROPOS, MESSAGE a Prométheus. Tyto metodiky se liší použitými diagramy a nástroji pro podporu, jak je vidět v Tabulka 7.

Tabulka 7: Srovnání metodik na základě diagramů a nástrojů pro podporu

Metodologie	Použité diagramy	Nástroje pro podporu
ADELFE	Use Case Diagram, Collaboration Diagram, Protocol Diagram Class Diagram	CASE OpenTool
GAIA	Acquaintance Diagram Agent Diagram Role schema Diagram Protocol Definitions Diagram Service Diagram	MASSDK
MaSE	Goals Diagram Organizational Diagram Role Diagram Protocol Diagram Agent Class Diagram	MASSDK
PASSI	Domain Description Diagram Agent Identification Diagram Roles Identification Diagram Protocol Description Diagram Task Specification Diagram Roles Description Diagram Structure Definition Diagram Domain Ontology Diagram	PTK
INGENIA	Use Case Diagram Class Diagram Agent Diagram	AgentTool Zeus IDK

	Cooperation Diagram	
TROPOS	Srategis Rational Model Strategic Dependency Model	TAOM, OpenOME
MESSAGE	Organisation Diagram Goal/Task Implication Diagram Workflow Diagram Delegation Structure Diagram Agent/Role Schema Interaction Diagram Domain Information Diagram	MOF
Prométheus	Goal Overview Diagram Agent Acquaintance Diagram Sequence Siagram Negocation Protocol Diagram	Prometheus Design Tool Jack Development Environment

Tabulka č. 7 popisuje vybrané aspekty metodik. Hodnotící kritéria se odvíjejí od vnitřní struktury metodik, přístupu k agentům, flexibility úprav atd.

	ADELFE	GAIA	PASSI	MASE	INGENIAS	TROPOS	MESSAGE	Prométheus
Objektově orientované	✓	✓	✓	✓	✓	✓	✓	✓
Vycházejí z požadavků na programové vybavení	-	-	-	-	✓	✓	-	✓
Vycházejí ze znalosti inženýrství	✓	✓	✓	✓	✓	✓	✓	✓
Inspirace konkrétní agentovou platformou	-	-	-	-	✓	-	-	✓

Servisně orientovaná metodika	-	✓	-	-	-	-	-	✓
Proaktivita agentů	✓	✓	✓	✓	✓	✓	✓	✓
Autonomní uvažování agentů	✓	✓	✓	✓	✓	✓	✓	✓
Různé úrovně abstrakce	✓	✓	✓	✓	-	-	✓	-
Předběžné požadavky na systém	✓	✓	✓	✓	✓	✓	✓	✓
Konečné požadavky na systém	✓	-	-	-	-	-	-	-
Analýza domény	✓	✓	✓	✓	✓	✓	✓	✓
Zaznamenání cílů agenta	✓	✓	✓	✓	✓	✓	✓	✓
Zahrnutí prostředí do modelu	-	✓	✓	-	✓	-	✓	✓
Model rolí	✓	✓	✓	✓	✓	✓	✓	✓
Znázornění interakcí	✓	✓	✓	✓	✓	✓	✓	✓
Nastavení pravidel společenství	✓	✓	✓	✓	✓	✓	✓	✓

Znázornění organizační struktury	✓	✓	✓	✓	✓	✓	✓	✓
Zaznamenání vnitřní struktury agenta	✓	✓	✓	✓	✓	✓	✓	✓
Uvažování služeb agenta ostatním agentům	✓	✓	✓	✓	✓	✓	✓	✓
Zaznamenání chování agentů	✓	✓	✓	✓	✓	✓	✓	✓
Implementace	✓	✓	✓	-	-	-	✓	-
Popsání kódu	-	-	✓	-	✓	-	-	-
Ontologie	-	-	✓	-	-	-	-	-
Testování	✓	-	✓	-	-	-	✓	-
Ověřování správného průběhu jednotlivých fází	✓	-	-	✓	✓	✓	✓	-
Návrh databáze	-	-	-	-	-	-	✓	-
Návrh uživatelského rozhraní	-	-	-	-	-	-	-	-
Sledování změn v požadavcích	✓	✓	✓	✓	✓	✓	✓	✓

Následná podpora nástroje	-	✓	✓	-	✓	✓	-	✓
Vhodná pro rozsáhlé projekty	-	✓	✓	-	-	-	✓	✓

Tabulka 8: srovnání metodik

5 Závěr

Cílem práce bylo detailně popsat metodiky pro oblast multiagentních systémů jak z teoretického, tak praktického hlediska. Práce seznamuje čtenáře s teorií agentů, jejich typy a zasazením do prostředí. V návaznosti na teorii agentů jsou v práci popsány multiagentní systémy a stručně jejich modelování vzhledem k návaznosti na systémové inženýrství. U vybraných metodik ADELFE, GAIA, PASSI, MASE, Tropos, INGENIAS, MESSAGE a Prométheus se podařil vypracovat teoretický popis a zhodnocení slabých a silných stránek každé metodiky. Teoretický a praktický popis se nepodařil vytvořit u metodiky RAP vzhledem k nedostatku zdrojů a u MAS-CommonKADS vzhledem k více strojařskému zastoupení případových studií. V návaznosti na tuto teoretickou část se podařilo ukázat na konkrétních případech použití uvedených metodik v praxi. Vybrané praktické studie byli převzaty z různých oblastí působnosti, takže se jedná o rozmanitý výběr. U každé praktické studie bylo nastíněno prostředí a typ multiagentního systému, který se měl realizovat na základě zvolené metodiky. Z případových studií je tak patrné, pro jaké oblasti vývoje je každá z metodik vhodná. Praktická část je doplněna o množství diagramů, které blíže seznamují čtenáře s konkrétní notací každé z metodik. Závěrem práce bylo úspěšné srovnání konkrétních metodik v rámci testu a pro usnadnění v rámci rozsáhlé tabulky. Kritéria srovnávací tabulky se dělí do několika oblastí. Ukazují, jakým způsobem jsou uvedené metodiky orientované. Jaké diagramy mají metodiky pro svůj vývoj k dispozici. Jestli se jedná o metodiky vhodné pro rozsáhlejší projekty či nikoliv. Či jak do hloubky se k modelování pomocí této metodiky přistupuje. V rámci srovnávací tabulky je dobře patrná rozdílnost či naopak podobnost uvedených metodik.

Multiagentní systémy jsou využitelné v různých oblastech vývoje. Přes zdravotnický, robotiku, adaptivní systémy, logistiku až po autonomní vozidla. Do budoucna se dá předpokládat, že se bude multiagentním systémům dostávat ještě více pozornosti a financování napříč firmami. Jedná se o východisko při zlepšování bezpečnosti na pracovištích a výkonosti firem. Dá se tak počítat s dalším zdokonalováním aktivních metodik na trhu. Tato práce je využitelná jako rozcestník při rozvaze o volbě metodiky pro konkrétní projekt vzhledem k uvedení popisu a uvedení silných a slabých stránek metodik.

V rámci této práce se dá uvažovat o přidání dalších metodik pro multiagentní systémy a o rozšíření zkoumaných otázek. Práce by se mohla ubírat více programovacím

směrem. Jaké jsou úskalí zavedení konkrétních metodik do praxe či jak detailně probíhá modelování diagramů. Lze uvažovat i zvolení více ekonomického směru. Jak jsou dané metodiky náročné z finančního hlediska atd.

6 Citovaná literatura

ALVIN, C., 2019. Static generation of UML sequence diagrams. *International Journal on Software Tools for Technology Transfer volume* [online]. International Journal on Software Tools for Technology Transfer, 31-53 [cit. 2022]. Dostupné z: https://link.springer.com/article/10.1007/s10009-019-00545-z

ANDERSEN, 2001. *Juul Møller Bokhandel A/S* [online]. In: . [cit. 2022]. Dostupné z: https://espen.com/papers/jme.pdf

ARAÚZO, Olmo, 2014. Gaia Analysis and Design of a Multi-agent-based Shop Floor Control System. In: *Direccion y Organizacion*. Universidad de Burgos, s. 26-52. Dostupné také z: https://doi.org/10.37610/dyo.v0i54.457

BERGENTI, Gleizes, 2004. The MaSE Methodology. In: DELOACH, Scott. *Methodologies and Software Engineering for Agent Systems*. Kansas: Springer, s. 107-125. Dostupné také z: https://doi.org/10.1007/1-4020-8058-1_8

BITTNER, Spence, 2002. *Use Case modeling*. 1. Boston: AddisonWesley Professional, s. 368.

BOKHANDEL, Juul, 2006. Requirements Elicitation for Agent-Based Applications. In: *Agent-Oriented Software Engineering* [online]. Madrid: Lecture Notes in Computer Science, s. 43-50 [cit. 2022]. Dostupné z: https://doi.org/10.1007/11752660_4

BORDINI, Dastani, 2009. Programming Multi-Agent Systems. In: *Third International Workshop, ProMAS*. Berlín: Springer, s. 267. Dostupné také z: https://doi.org/10.1007/11678823

BROM, Kadlec, 2013. DyBaNeM: Bayesian Episodic Memory Framework for Intelligent Virtual Agents. In: *Intelligent Virtual Agents*. Springer, s. 15-23. Dostupné také z: https://doi.org/10.1007/978-3-642-40415-3_2

BROOKS, 1989. *He Industrialization of Intelligence*. 0815349378. Routledge.

CIMR, 2016. *Agentová simulace efektivního rozmístění aktivních prvků protivzdušné obrany*. Hradec Králové. Dostupné také z: https://theses.cz/id/3zu3or/STAG85985.pdf. Diplomová práce. Univerzita Hradec Králové.

COPELAND, Jack, 2007. Artificial Intelligence: A Philosophical Introduction. *Alanturing* [online]. 1. Canterbury: Wiley-Blackwell, s. 429-482 [cit. 2022]. Dostupné z: https://doi.org/10.1016/B978-044451540-7/50032-3

CORCHADO, Rodríguez, 2011. *International Symposium on Distributed Computing and Artificial Inteligence*. Berlin: Springer, s. 443. Dostupné také z: https://doi.org/10.1007/978-3-642-19934-9

COSSENTINO, Massimo, 2012. *From Requirements to Code with the PASSI Methodology*. Italian National Research Council. Dostupné také z: https://doi.org/10.4018/978-1-59140-581-8.ch004

EDEKI, Charles, 2013. Agile Unified Process. *IJCSMA*. Brooklyn: International Journal of Computer Science and Mobile Applications, (2321-8363), 5.

ELLIOTT, Geoffrey, 2004. *Global Business Information Technology: an integrated systems approach*. 978-0321270122. Boston: Addison Wesley.

EVANS, Kearney, 2001. *MESSAGE: Methodology for Engineering System of Software Agents*. EURESCOM. Dostupné také z: https://people.ucalgary.ca/~far/Lectures/SENG697/PDF/references/P06.pdf

GIORGINI, Kolp, 2004. The Tropos Methodology. In: *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Toronto: Springer. Dostupné také z: https://doi.org/10.1007/1-4020-8058-1_7

HÁJKOVÁ, 2012. *BDI Agenti*. Praha: Masarykova univerzita. Dostupné také z: https://nlp.fi.muni.cz/uui/referaty2012/veronika_hajkova/referat.pdf. Seminární práce.

HUSÁKOVÁ, 2013. Multi-agentové systémy pro modelování biologických fenoménů. In: *Kognice a umělý život*. Hradec Králové. Dostupné také z: https://www.researchgate.net/publication/259932244_Multi-agentove_systemy_pro_modelovani_biologickych_fenomenu

CHAINBI, 1998. Conception, Behavioural Semantics and Formal Specification of Multi-agent Systems. In: *Multi-Agent Systems. Theories, Languages and Applications*. Berlin: Springer, s. 16-28. Dostupné také z: https://doi.org/10.1007/10693067_2

CHEN, Deng, 2021. Neural symbolic reasoning with knowledge graphs: Knowledge extraction, relational reasoning, and inconsistency checking. *Fundamental Research*. National Natural Science Foundation of China, (1), 565-573. Dostupné také z: https://doi.org/10.1016/j.fmre.2021.08.013

CHOPRA, Christie, 2020. *An Evaluation of Communication Protocol Languages for Engineering Multiagent Systems*. Lancaster: Journal of Artificial Intelligence Research, (69). Dostupné také z: https://doi.org/10.1613/jair.1.12212

INNA VISTBAKKA, Elena, 2021. *Modelling resilient collaborative multi-agent systems*. s. 103. Dostupné také z: https://doi.org/10.1007/s00607-020-00861-2

IPSER, Jaroslav, 2008. *Informační portál pro multiagentní technologie*. Brno, s. 40. Dostupné také z: https://is.muni.cz/th/uowq4/final_vz6gu.pdf. Bakalářská práce.

JACKSON, 2001. *Problem frames : analysing and structuring software development problems*. Addison Wesley.

JULIAN, Botti, 2019. *Multiagentní systémyUPRAAAAAAVITTT*. Valencie: Katedra informačních systémů a výpočetní techniky.

KUBÍK, 2004. *Inteligentní agenty: tvorba aplikačního softwaru na bázi multiagentních systémů*. Brno: Computer Press, s. 280.

LARIOUI, El, 2020. *Multi-Agent System Architecture Oriented Prometheus*. International Journal of Emerging Trends in Engineering Research. Dostupné také z: https://www.warse.org/IJETER/static/pdf/file/ijeter105852020.pdf

LI, Daoliang, 2007. *COMPUTER AND COMPUTING TECHNOLOGIES IN AGRICULTURE, VOLUME II*. Wuyishan, s. 719. Dostupné také z: https://link.springer.com/content/pdf/10.1007/978-0-387-77253-0.pdf

MAŘÍK, Pěchouček, 2001. *Multiagentní systémy: Principy komunikace a základní formální architektury*. 80-200-0502-1. Praha: Academia.

MEFTEH, 2015. *Simulation Based Design for Adaptive Multi-Agent Systems with the ADELFE Methodology*. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, s. 36-38. Dostupné také z: https://doi.org/10.1109/WETICE.2013.60

NETRVALOVÁ, 2010. *Úvod do problematiky multiagentních systémů*. Plzeň: ZČU v Plzni, 28. Dostupné také z: https://www.kiv.zcu.cz/~netrvalo/phd/MAS.pdf

OCHRANA, František, 2009. *Metodologie vědy: úvod do problému*. Praha: Karolinum.

OMIDSHAFIEI, Papadimitriou, 2019. *α- Rank: Multi-Agent Evaluation by Evolution*. In: . Dostupné také z: https://doi.org/10.1038/s41598-019-45619-9

OPREA, Michaela, 2004. *Applications of Multi-Agent Systems*. Ploiesti: University of Ploiesti, s. 239–270.

PAVÓN, Gómez-Sanz, 2005. The INGENIAS methodology and tools. In: *Agent-Oriented Methodologies*. Idea Group. Dostupné také z: https://doi.org/10.4018/978-1-59140-581-8.ch009

POKORNÝ, 2014. *Grafické řešení úloh vícekriteriálního hodnocení variant*. Praha: Vysoká škola ekonomická. Dostupné také z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=45577. Bakalářská práce.

PRÝMEK, 2008. *Multiagentní systémy v praxi* . s. 59. Dostupné také z: https://is.muni.cz/th/eqn7d/prymek_dp_final.pdf. Diplomová práce. MASARYKOVA UNIVERZITA.

PŘIBYLOVÁ, 2015. *Deterministické modely* [online]. In: . [cit. 2022]. Dostupné z: https://is.muni.cz/do/rect/el/estud/prif/ps15/determ/web/docs/deterministonline.pdf

RUSSELL, Norvig, 2020. *Artificial Intelligence: A Modern Approach*. Second Edition. Londýn: Prentice Hall, s. 113. Dostupné také z: https://ktiml.mff.cuni.cz/~bartak/ui/lectures/lecture02.pdf

SUDARMA, Ariyani, 2021. *Implementation of the Rational Unified Process (RUP) Model in Design Planning of Sales Order Management System*. Denpasar: Udayana University. Dostupné také z: https://doi.org/10.29407/intensif.v5i2.15543

TIA, Nuryasin, 2020. *Model Simulasi Rational Unified Process (RUP) Pada Pegembagan Perangkat Lunak*. Journal Repositor. Dostupné také z: https://www.researchgate.net/publication/343202411_Model_Simulasi_Rational_Unified_Process_RUP_Pada_Pegembagan_Perangkat_Lunak

V. MAŘÍK, O., 2001. *Multiagentní systémy: Principy komunikace a základní formální architektury*. Praha: Academia, **189-236**.

WERNECK, Kano,. *Evaluating ADELFE Methodology in the Requirements Identification*. Toronto: York University. Dostupné také z: http://www.yorku.ca/cysneiro/articles/Adelfe-WER07-FinalVersion-1.pdf

WITH, Implementing. *Implementing Multi-agent Systems Organizations with INGENIAS*. Universidad Complutense de Madrid. Dostupné také z: https://link.springer.com/content/pdf/10.1007/11678823.pdf

WOOLDRIDGE, Jennings, 2000. *The Gaia Methodology for Agent-Oriented*. Netherlands: Kluwer Academic Publishers, 285–312. Dostupné také z: https://doi.org/10.1023/A:1010071910869

YU, E., 2002. *Agent-Oriented Methodologies-Towards a Challenge Exemplar*. Toronto: Bi-Conference Workshop on Agent-Oriented, s. 47-63. Dostupné také z: https://www.cs.toronto.edu/pub/eric/AOIS02.pdf

ZAMBONELLI, Jennings, 2003. *Developing Multiagent Systems: The Gaia Methodology*. ACM Transactions on Software Engineering and Methodology, (12), 54. Dostupné také z: https://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/tosem2003.pdf

ZBOŘIL, 2004. *Plánování a komunikace v multiagentních systémech*. Brno: FIT VUT, s. 105. Dostupné také z: http://www.fit.vutbr.cz/~zborilf/PhD/thesis.pdf. Disertační práce.

ZBOŘIL, 2006. Agentní systémy, základní architektury, jejich modely a realizace. In: *Podklady k přednáškám kurzu AGS* [online]. [cit. 2021]. Dostupné z: http://www.fit.vutbr.cz/~zborilf/study/AGS/AGS02_Zakladni_modely.pdf

ZBOŘIL, 2006. *Návrh a modelování MAS* [online]. [cit. 2022]. Dostupné z: http://www.fit.vutbr.cz/~zborilf/study/AGS/AGS12_Navrh_a_modelovani.pdf

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Eliška Hegrová**

Studium: I2100507

Studijní program: B0688A140001 Informační management

Studijní obor: Informační management

Název bakalářské práce: **Srovnání metodik pro oblast multi-agentových systémů**

Název bakalářské práce AJ: Comparison of Multi-agent Based Methodologies

Cíl, metody, literatura, předpoklady:

Cílem práce bude porovnání vybraných metodik pro oblast multi-agentových systémů Tropos, MAS-CommonKADS, PASSI, Prometheus, Gaia, ADELFE, MESSAGE, INGENIAS, MASE, RAP, případně dalších na základě výzkumné rešerše. Práce poskytne popis jejich základních principů a srovnání jejich klíčových charakteristik. Osnova:

1. Úvod
2. Cíle práce
3. Teoretická část – popis metodik
4. Praktické aplikace a využití metodik
5. Shrnutí
6. Závěr
7. Použitá literatura

Garantující pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: doc. RNDr. Petr Tučník, Ph.D.

Datum zadání závěrečné práce: 15.10.2021