UNIVERZITA KARLOVA
KATOLICKÁ TEOLOGICKÁ FAKULTA
Katedra filozofie a práva

Jiří Fiala

Metafyzika Tomáše Akvinského a objektově orientované paradigma

Bakalářská práce

Vedoucí práce: doc. Mgr. David Svoboda, Ph.D.

Praha 2024

Prohlášení

1. Prohlašuji, že jsem předkládanou práci zpracoval samostatně a použil jen uvedené prameny a literaturu.

2. Prohlašuji, že práce nebyla využita k získání jiného titulu.

3. Souhlasím s tím, aby práce byla zpřístupněna pro studijní a výzkumné účely.

V Svitavách dne 14. 4. 2024 Jiří Fiala

Bibliografická citace

Metafyzika Tomáše Akvinského a objektově orientované paradigma: bakalářská práce / Jiří Fiala; vedoucí práce: doc. Mgr. David Svoboda, Ph.D. -- Praha, 2024. -- 46 s.

Anotace

Tato práce se zabývá inspirací božím stvořením v informatice a zejména pak tzv. objektově orientovaným paradigmatem (OOP), které je základem pro objektově orientované programování. OOP není však jedinou informatickou inspirací ve stvoření, ale je součástí širší skupiny oblastí, které se snaží rozkrývat a využívat inteligentních principů obsažených v božím stvoření. V této práci se pak především snažíme poukazovat na skutečnost, že OOP je svou abstraktní reprezentací v určité oblasti blízké k abstrakci stvoření v metafyzice, resp. ontologii. Přestože se nejedná o zjištění nové skutečnosti, tato problematika je v oblasti informatiky, filozofie i teologie poměrně málo diskutována. Proto se zde tímto tématem zabýváme. Na základě dostupné literatury i vlastní studie jsou zde předkládány některé praktické příklady, se kterými se můžeme v OOP setkat, a které jsou v určité podobnosti s ontologií dle Tomáše Akvinského. S odkazem na dostupné publikace připojujeme také ostatní dosud známé poznatky k této problematice.

Klíčová slova

Metafyzika, Tomáš Akvinský, objektově orientované paradigma, design systém, přirozená teologie

Abstract

This thesis deals with the inspiration from god's creation in computer science and mostly in object-oriented paradigm (OOP), which is the basis of object-oriented programming. We also note that OOP is not just one inspiration used in computer science from the creation, but is a part of wider group, where we can discover an intelligent principle encapsulated in God's creation. In this thesis we mostly note the fact that OOP is like Thomas Aquinas metaphysis in terms of its abstraction of creation. Although this fact is not new, this issue is not discussed enough in computer science and in theology vice versa. Hence that is why we are discussing this issue. Based on available sources and own studies we offer practical examples used in OOP and comparing them with adequate ontology of Thomas Aquinas. With respect to available publications, we also remark nowadays knowledge to this issue.

Keywords

Metaphysics, Thomas Aquinas, object-oriented paradigm, design system, natural theology

Počet znaků: 71 840

Poděkování

Děkuji p. docentu Davidu Svobodovi za trpělivost a odborné rady poskytnuté při vedení této práce. Děkuji také p. doktorce Lence M. Demartini za zpětnou vazbu během čtení prvopisu této práce. Současně děkuji i katolické církvi a jejím představitelům, kteří mě duchovně vedli a motivovali při mém studiu a psaní této práce. V neposlední řadě poděkování patří i členům mé rodiny, neboť bez jejich vstřícnosti by tato práce nemohla vzniknout.

Obsah

Úvod ... 6
1. Informatické inspirace ve stvoření ... 9
   1.1. Motivace ... 9
   1.2. L-systém ... 10
   1.3. Neuronové sítě ... 11
   1.4. Evoluční algoritmy ... 12
   1.5. UX design ... 13
2. Historie programovacích jazyků a jejich modelů ... 15
3. Objektově orientované paradigma ... 17
   3.1. Úvod do OOP ... 17
   3.2. OOP v přehledu ... 18
4. Srovnání filozofických systémů (ontologií) ... 24
5. Metafyzika Tomáše Akvinského ... 26
6. Srovnání Tomášovo metafyziky a OOP ... 28
7. Ukázka přínosu na životním cyklu SW ... 30
   7.1. Inspirace skrze návrhové principy či vzory ... 31
   7.2. Člověk jako návrhový vzor pro své programy ... 32
   7.3. Od člověka přes návrhové principy až k SW ... 33
   7.4. Zhodnocení uvedeného inspirovaného přístupu ... 37
8. Zhodnocení přínosu z pohledu teologie ... 39
Závěr ... 42
Seznam použitých zkratek ... 44
Použitá literatura ... 45

Úvod

V oblasti informatiky v různých situacích odkládáme naše vlastní sofistikované postupy jako neoptimální pro řešení v určitých úlohách a s nadšením přejímáme inspirace inteligentních principů, které nacházíme ve stvoření. Zapomínáme se ale někdy zamýšlet na tím, proč tyto inteligentní principy optimálně fungují, a kde se vlastně vzaly. V této práci se snažíme poukázat na dvě podstatné skutečnosti.

Za prvé zde ukazujeme, že vyšší míra inspirovanosti nás může přivádět k lepším informatickým výsledkům, což je sice zajímavý poznatek, ale jeho význam je spíše v rovině informatiky.

Za druhé chceme ukázat, že i při informatické činnosti jako je např. vývoj softwaru, zejména tam kde se dotýkáme inspirovaných oblastí, můžeme ve světle víry spatřovat v objevených inteligentních principech otisk boží péče o své stvoření. Z pohledu teologie je to odkrývání božího sebesdělení, a to i skrze tyto technické vědní obory. Tento poznatek je už více teologický, neboť tím, jak Boha takto lépe ve světle víry poznáváme, se rozšiřuje i naše rozumové poznání, které pak recipročně dává opět růst i naší víře. V uvedené práci se celkově dotýkáme různých informatických inspirací, nejvíce však tzv. objektově orientovaného paradigmatu (OOP). Zde se pak především snažíme poukazovat na skutečnost, že OOP je svou abstraktní reprezentací blízké k abstrakci stvoření v metafyzice, což podtrhuje i míru inspirovanosti OOP. Nevědomě či podvědomě se zde tak nutně dotýkáme oblastí, jako je právě ontologie. Přestože se nejedná o zjištění nové skutečnosti, tato problematika je v oblasti informatiky, filozofie i teologie poměrně málo diskutována. Proto se zde tímto tématem zabýváme a na několika ilustrativních příkladech a případové studii ukazujme, jak skrze OOP můžeme rozkrývat odpovědi na obě výše uvedené podstatné skutečnosti.

V případě OOP se jedná o zcela jiný způsob myšlení, kterými v informatice můžeme symbolicky vyjádřit (modelovat neboli také simulovat) entity jako je člověk, živé i neživé věci kolem nás. Nevědomě či podvědomě se zde tak nutně dotýkáme oblastí jako je metafyzika či ontologie[1]. Při těchto myšlenkových abstrakcích OOP se tak můžeme ocitnout na rovině myšlení, které přesahuje běžné technické procesy informatiky. Můžeme dokonce získat dojem, že jsme se octli ve zcela jiné mimo-informatické oblasti. Vyjádřením této myšlenky jinými slovy lze následně říct, že skrze OOP lze tak poznávat

[1] V této práci budeme pracovat s pojmy metafyzika a ontologie jako se synonymy.

řád božího stvoření a skrze poznávání stvořeného, inteligence stvořitele tak docházet i k poznání jeho stvořitele – Boha.

K tomuto poznatku můžeme zpravidla dospět buďto na základě vlastní prožité zkušenosti, nebo skrze zdokumentovaná svědectví jiných. V této práci bychom chtěli podpořit výše uvedený poznatek. Cílem se tak stává názorně demonstrovat podobnost mezi myšlenkovými abstrakcemi OOP a metafyzikou dle Tomáše Akvinského, a to prostřednictvím vhodných příkladů a zvoleného případu vývoje reálného programu v OOP včetně odkazu na jiné vhodné práce, které se tohoto tématu dotýkají[2].

Z pohledu terminologie budeme z hlediska metafyziky pracovat s pojmy dle učení Tomáše Akvinského (filozofie zde bude vystupovat jako nástroj teologie), z hlediska informatiky budeme OOP reprezentovat skrze tzv. modelovací jazyk UML[3] (*Unified Modeling Language*) a pseudokód založený na jazyku Java, resp. C# (informatika zde bude opět jen v roli nástroje pro teologii).

V první kapitole se nejprve celkově podíváme na přehled oblastí, které jsou v informatice výrazněji inspirované božím stvořením, a to i s odkazem na jiné související práce. Vycházíme zde z předpokladu, že boží stvoření, které je smysly poznatelné, nám dává skrze vědu odkrývání svých zákonitostí a my tak docházíme k poznání velmi inteligentních systémů, principů, nad kterými můžeme žasnout, a které můžeme s nemalým užitkem zpětně uplatnit i v našich dalších vědních oborech.

Ve druhé kapitole se zastavíme u přehledu vývoje paradigmat pro abstrakce nad výpočetním procesem, respektive nad vývojem programovacích jazyků jako takových.

Ve třetí kapitole se už následně zaměříme na oblast OOP, kterou ještě blíže popíšeme včetně nezbytných odborných termínů.

Čtvrtá kapitola se zaměří na přehled ontologických systémů. V páté kapitole blíže popíšeme klíčové pojmy z Tomášovo metafyziky, abychom se v následné šesté kapitole mohli podívat na možnou podobnost OOP s jednotlivými pojmy metafyziky dle Tomáše. Některé vzájemné vztahy se zde pokusíme také ilustrovat skrze vhodné diagramy jazyka UML.

V další již sedmé kapitole se pak budeme zabývat přínosem tohoto pohledu pro informatiku, a to na konkrétní případové studii s návrhem programu dle OOP, ve které

---

[2] TARKO, Vlad. *The Metaphysics of Object Oriented Programming* (28. 5. 2006) [2022-1-1]. <https://news.softpedia.com/news/The-Metaphysics-of-Object-Oriented-Programming-24906.shtml>.

[3] OESTEREICH, Bernd. *Developing Software with UML: Object-oriented Analysis and Design in Practice*. Amsterdam: Addison-Wesley, 2002.

budeme moci ještě názorněji představit souvislost mezi zde navrženými objekty, jejich atributy a reálnými jsoucny s jejich akcidenty. Cílem této případové studie je také snaha demonstrovat, že souvislost mezi termíny OOP a ontologií zde není uměle vytvářena, ale že zcela přirozeně vyvstává v běžném návrhu programu dle OOP. Jedná se tedy sice o informatickou činnost, které je ale podstatně inspirována řádem božího stvoření a myšlenkovými procesy z oblasti ontologie, které toto uchopují.

V osmé kapitole se naopak zaměříme na přínos tohoto přístupu z pohledu teologie. Na závěr shrneme výstupy této práce a zhodnotíme pozitiva přístupu přetaveného v této práci.

1. Informatické inspirace ve stvoření

1.1. Motivace

V informatice se můžeme setkat s řadou různých aplikací, ve kterých můžeme spatřovat inspiraci v boží stvoření. V této kapitole si postupně představíme několik základních reprezentativních oblastí, které této inspiraci odpovídají. Výjimkou zde bude oblast OOP, která je ústředním tématem této práce, a proto ji blíže rozvedeme až v následné samostatné kapitole. Za zmínku stojí skutečnost, že právě i skrze tyto inspirace můžeme zpětně dospívat k poznání stvořeného, k inteligenci tvůrce, a tedy v důsledku tak dospívat i k poznání jejich stvořitele – Boha. Použitím informatické metafory bychom tak mohli hovořit o těchto přístupech jako o reverzním inženýrství s tím rozdílem, že cílem zde není získání zdrojového kódu z kompilovaného programu, ale jde o získání poznání o Stvořiteli z jeho stvoření. Použitím teologické metafory dle Órigena a jeho stupňů poznání pak můžeme o tomto přístupu hovořit také jako o kontemplaci Boha ve stvoření.

Nakonec výše uvedené můžeme završit symbolickou abstrakcí hermeneutického kruhu. Ten lze zde popsat jako postupné poznání stvořeného, skrze které se tak můžeme i v informatice setkat s Bohem, tím se rozšíří naše rozumové poznání o nové světlo, kterým můžeme nahlédnout za hranici čistě informatického rozumového chápání a pak nově nebo lépe vidět ve světle víry přítomnost Boha za tímto stvořením, nebo v tomto stvoření, viz také encyklika Jana Pavla II *Fides et Racio*[4].

Současně zde také vycházíme z pramenů v církvi (např. Jan Pavel II[5], papež František[6]), ale i sekundární literatury (např. prof. Pospíšil[7]), dle kterých lze o evoluci uvažovat jako o inteligentním nástroji božího záměru, kterým Bůh uskutečňuje svůj plán pro stvoření člověka a ostatního tvorstva.

---

[4] JAN PAVEL II. *Fides et ratio*, encyklika ze dne 14. 9. 1998 – český překlad. Praha: Zvon, 1999.

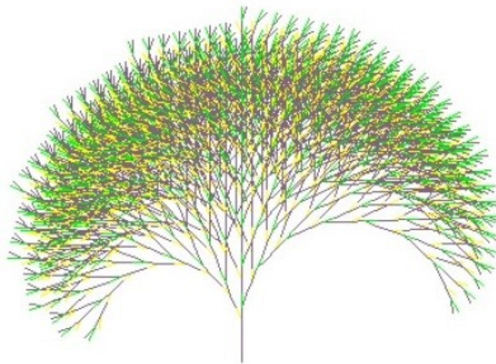
[5] JAN PAVEL II. *Poselství Jana Pavla II. adresované grémiu Papežské akademii věd na téma evoluce*, proneseno dne 22. října 1996. In: EV 17. Bologna: EDB, 1999, č. 1346–1354.

[6] FRANTIŠEK. *Evolution … is not inconsistent with the notion of creation* (27. 10. 2014) [2022-1-1]. <https://religionnews.com/2014/10/27/pope-francis-evolution-inconsistent-notion-creation/>.

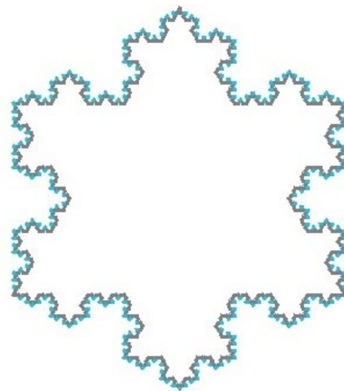
[7] POSPÍŠIL, Ctirad V. *Zápolení o pravdu, naději a lidskou důstojnost: Česká katolická teologie 1850–1950 a výzvy přírodních věd*. Praha: Karolinum, 2017.

1.2. L-systém

L-systém neboli také Lindermayerův systém je druh formálních gramatik, které jsou předmětem zájmu především teoretické informatiky. V roce 1968 maďarský botanik Aristid Lindenmayer objevil zákonitost, dle které lze generovat různé druhy rostlin včetně jejich tvaru i barev, obr. 1.1. Tato zákonitost je pak zachytitelná v pravidlech tzv. multigramatik, které jsou druhem formální gramatiky dle Chomského hierarchie gramatik. Pro svá specifika je tato multigramatika označována jako samostatný systém tzv. L-systém. Pomocí tohoto systému lze pak vytvářet také některé druhy fraktálů jako je např. Kochova vločka (viz obr. 1.2).



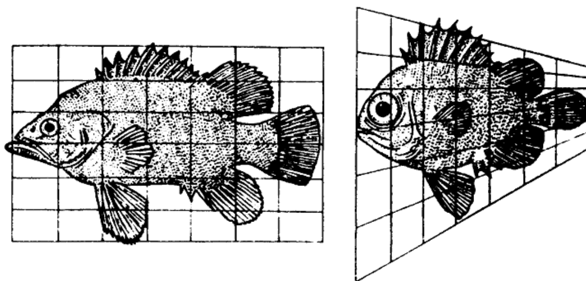
Obr. 1.1: rostlina miříku střapatého generovaná programem v jazyce Java na základě vstupní gramatiky obsahující základní abecedu symbolů a pravidel jejich posupného zřetězení (vlastní studie).



Obr. 1.2: tzv. Kochova vločka generované v témže programu na základě své gramatiky obsahující základní abecedu symbolů a pravidel jejich posupného zřetězení (vlastní studie).

V širším slova smyslu bychom zde mohli hovořit o nekonečném zdroji matematických funkcí, kterými je stvoření protkáno a jehož postupným poznáváním můžeme poznávat vhodnost různých inspirací z těchto funkcí pro naše vlastní řešení informatických úloh, ale i pro jiné technické obory jako je třeba stavebnictví (např. tvar konstrukce vysutého mostu se podobá stavbě páteře dinosaurů). Jedním z matematiků, kteří poukazovali na souvislosti mezi zákonitostmi stvoření a matematikou byl i biolog a matematik prof. D'Arcy Thompson, který se tak stal zakladatelem nového oboru tzv. „*biomatematiky*". Ve své knize *On Growth and Form*[8] ukazuje například souvislosti mezi matematickými transformacemi a proměnami napříč mezitřídními živočišnými druhy tedy, že rozmanitost tvarů je možné redukovat použitím transformací v různých geometriích (viz obr. 1.3). V kontextu Thompsonovo publikace může považovat i výše uvedený L-systém za podsystém z této biomatematické množiny funkcí.

$$u = (a_{10} + a_{11}x + a_{12}y)/(a_{00} + a_{01}x + a_{02}y)$$
$$v = (a_{20} + a_{21}x + a_{22}y)/(a_{00} + a_{01}x + a_{02}y)$$

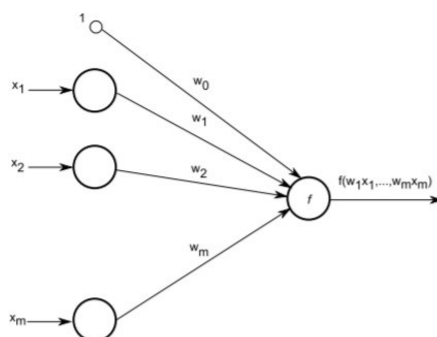


Obr. 1.3: redukce tvarů transformační funkcí projektivní geometrie, která vede k proměně skrze mezitřídními druhy ryb (převzatá ilustrace z uvedené Thompsonovo publikace).

1.3. Neuronové sítě

Jedná se o umělé neuronové sítě, které jsou využívány zejména v oblasti umělé inteligence. Základem je jednoduchý model neuronu tzv. perceptron (viz obr. 1.4), ten je v praxi běžně propojen s dalšímy umělými nerony přes své vstupy a výstupy, a to jak rovině horizontální, tak i vertikální. Vzniká tak rozsáhlá síť o několika vrstvách. Ta může být využívána v aplikacích tzv. strojového učení (*machine learning*) pro rozpoznávání (např. rozpoznávání určitých vzorů od jednoduchých tvarů až po SPZ v mýtných branách), algoritmech komprese digitálních obrazů, nebo jako funkce u spam filtrů aj.

[8] Thompson, A. *On Growth and Form*, the complete revised edition. New York: Dover Publications, 1992.



Obr. 1.4: model umělého neuronu se vstupními hodnotami x, váhami w a výstupní funkcí f (vlastní studie).

Znovu se tak setkáváme s oblastmi praktického uplatnění, které bychom bez inspirace v části stvoření nebyly schopni řešit potřebné úlohy efektivně s dostatečnou přesností v reálném čase na současně dostupném HW. Znovu si tedy můžeme položit otázku po původu inteligence, která byla schopna takových rozumových úsudků, které předčí naše vlastní sofistikované algoritmy založené na tradičních přístupech.

1.4. Evoluční algoritmy

Myšlenka evolučních algoritmů pochází podobně jako v případě neuronových sítí ze zákonitostí pozorovaných ve vývoji stvoření. Vlastní autorství evolučního procesu lze opět jako v ostatních případech připisovat Bohu – Stvořiteli, neboť jak jsme již uvedli v motivaci této kapitoly, evoluční proces se ukazuje být božím nástrojem celého stvoření[9].

Výpočetní procesy (evoluční procesy), které jsou postaveny na elementárních operacích (jednoduchých změnách ve vývoji), se mohou ve svém důsledku jevit jako velmi inteligentní, pokud mají dostatečný výpočetní výkon (populaci).

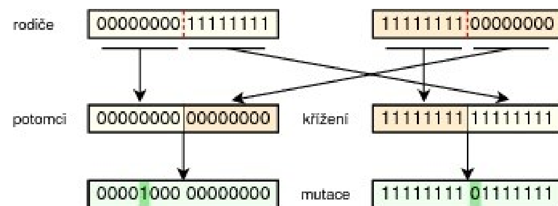
Mezi tyto elementární operace můžeme zařadit operace jako dědičnost, přirozený výběr a křížení nebo mutace, viz obr. 1.5. Tímto „*šlechtěním*“ je dosahováno změny u jedinců v populaci, kde jedinec reprezentuje jedno možné řešení úlohy a dostatečně počet kvalitních řešení v populaci pak zpravidla znamená ukončení procesu, tj. dosažení očekávaného výsledku. Nutno poznamenat, že otázka vyhodnocení je zde řízena tzv. ***fitness funkcí,*** tj. algoritmem, který obsahuje informaci o očekávané kvalitě (naplnění jistých parametrů) a každý jedinec je v daném mezikroku posouzen touto funkcí, aby mohlo být ověřeno, zda se již nenaplnily parametry očekávaného řešení.

---

[9] POSPÍŠIL, Ctirad V. *Zápolení o pravdu, naději a lidskou důstojnost: Česká katolická teologie 1850–1950 a výzvy přírodních věd*. Praha: Karolinum, 2017.

Ve výsledku jsem tak v informatice díky těmto evolučním algoritmům schopni efektivně nacházet řešení úloh, pro které nejsou algoritmické postupy vůbec známé nebo mají přílišnou časovou či prostorovou složitost (tj. nejsou prakticky použitelné).

Na závěr můžeme ještě přidat jedno základní zamyšlení. Jestliže jsme jako lidé obdaření rozumem nebyli sami schopni dojít k řešení těchto úloh vlastní logickou cestou, můžeme se ptát, jakou inteligencí disponuje Ten, od kterého jsme se rozhodli „*opisovat*“ jeho principy.



Obr. 1.5: ukázka principu elementárních operací jednobodového křížení a mutace (převzato z práce Z. Vašíčka[10])

1.5. UX design

Human Computer interaction (HCI) je vědní obor, který přesahuje také běžný koncept informatiky a vyžaduje další specifické znalosti z oblastí jako je zejména filozofie, psychologie a sociologie. Zabývá se návrhem uživatelského rozhraní (nejen grafického rozhraní označovaného jako GUI), tak aby bylo dobře přístupné a použitelné pro cílové uživatele. Samotné termíny přístupnosti (***accessability***) a použitelnosti (***usability***) zde nejsou jen v roli obecných pojmů, ale v roli měřitelných metrik, které mají v tomto oboru klíčovou roli. V minulé době býval tento obor překládán jako „*styk člověk stroj*“, tento překlad se však neujal. Dnes se proto spíše i u nás setkáme se zaužívanou zkratkou HCI nebo také s termínem ***UX design*** (zkratka z angl. ***User eXperience design***), což je praktický název pracovního oboru s odborností v HCI.

V porovnání se zahraničním systémem vzdělávání není tento obor u nás bohužel běžně zakotven, určitou výjimkou je u nás fakulta elektrotechniky (FE ČVUT), kde lze v rámci katedry *Počítačové grafiky a interakce* studovat tento obor v rovině magisterského studia od ak. roku 2016/2017[11]. Jiným příkladem a přístupem může být i Filosofická fakulta

[10] VAŠÍČEK, Zdeněk. *Biologií inspirované počítače – kartézské genetické programování* [2022-1-1]. < https://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/>.

[11] OUKOPEC, Jindřich. *ČVUT otevírá nový obor, studenty naučí interakčnímu designu* (28. 9. 2016) [2022-1-1]. <https://www.czechdesign.cz/temata-a-rubriky/cvut-otevira-novy-obor-studenty-nauci-interakcnimu-designu>.

Masarykovi univerzity (FI MUNI), kde se tento obor nachází pod *Katedrou informačních studií a knihovnictví* pod názvem „*Design informačních služeb*".[12]

Člověk je zde ve středu celého návrhu a jeho zhodnocení, pracujeme zde proto s také termíny jako jsou uživatelské příběhy (tzv. ***user stories***, ***scenarios***), které jsou určitým textovým popisem toho, co a jak s daným systémem či nástrojem děláme, resp. potřebujeme dělat. Zpravidla pak následují tzv. diagramy případu použití (***use case diagram*** neboli také USD), které více formálně vystihující různé způsoby používání. Obvykle je tento soubor požadavků převeden do formy tzv. drátového modelu ***wireframe***, což odpovídá zachycení první grafické podoby návrhu. Později přichází přesnější návrh s možností určité interakce, tento model označovaný jako tzv. ***mockup***. Nejblíže konečné podoby jsou už pak prototypy, které nejenže nabízejí nezbytnou interakci, ale celkově se svým vzhledem i obsahem snaží připomínat možné konečné řešení.

Vlastních způsobů, jak vyhodnotit jednotlivé návrhy z pohledu UX designu je hned několik. Běžně se můžeme setkat s ***inspekčními metodami*** kam spadá i heuristická evaluace. Ta je založena na tzv. heuristikách neboli ověřených principech, které vycházejí z osvědčené praxe. Asi nejznámější jsou tzv. Nielsenovo[13] heuristiky, které obsahují 10 základních principů pro dobrý návrh uživatelského rozhraní (viz ukázka na obr. 1.6).

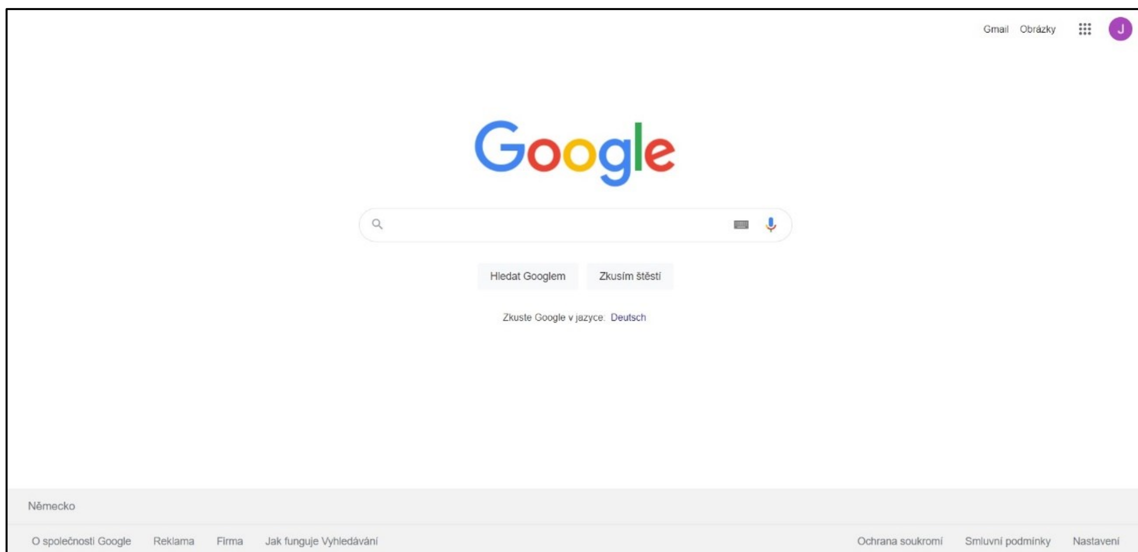
Jinou možností ověření vhodnosti daného návrhu jsou ***experimentální metody***, mezi které můžeme zařadit i testování použitelnosti s reálnými uživateli (tzv. ***usability testing***). Zde je opět koncový uživatel vtažen do nitra návrhu. Specifickým, ale poměrně rozšířeným druhem tohoto testování je např. tzv. ***guerrilla test***, který pracuje s koncepcí přirozeného a náhodného oslovení předem nepřipraveného uživatele s cílem provedení několika kroků v daném testovaném prostředí. Takové oslovení se děje zpravidla náhodně např. v kavárně nebo knihkupectví a může být odměněno např. kávou nebo knihou zdarma.

S uvedeným pojmem ***UX design*** pak úzce souvisí i další termíny jako je tzv. ***Design system*** a ***UX driven software development***, o kterých se více zmíníme až v dalších podkapitolách 7.1 a 7.3.

---

[12] URBAN, Tereza. *Vyhlašují boj chaosu na úřadech. Nový studijní program vychová designéry, kteří dokážou z úřadů vytvořit srozumitelné místo*. (15. 8. 2022) [2023-1-1]. <https://www.czechdesign.cz/temata-a-rubriky/vyhlasuji-boj-chaosu-na-uradech-novy-studijni-program-vychova-designery-kteri-dokazou-z-uradu-vytvorit-srozumitelne-misto>.

[13] Jakob Nielsen a Donald Norman jsou klíčové osoby při zrodu oboru HCI během 80. let min. století, za zmínku zde stojí zejména Normanova publikace „***The Design of Everyday Things***", která poprvé vyšla už v roce 1988.



*Obr. 1.6: ukázka 8. principu „**Aesthetic and minmalistic design**“, dle Nielsenovo heuristiky, která je např. naplněna i u nejrozšířenějšího vyhledávače společnosti Google.*

Záměrně jsme představení HCI oboru (resp. UX designu) ponechali až na konec této kapitoly, protože právě zde je inspirace člověkem snad nejvíce patrná. Ostatně je více než zřejmé, že bez inspirace v řádu lidského stvoření, bychom v tomto oboru příliš nepořídili. Přestože je řada informatických zájmů jinak orientována na parametry různých neživých technologií, v této oblasti je zde orientace přímo na člověka. Důvodem je zejména prostá skutečnost, že dříve nebo později jsme nakonec nuceni si připustit, že koncovým beneficientem jakékoliv informatické technologie musí být vždy člověk. HCI, resp. UX design nás tedy vrací zpět na zem a připomíná nám, že jakákoliv technologie, která se ale míjí s lidskými možnostmi, je určena k zániku pro svou nesmyslnost, resp. nepoužitelnost.

2. Historie programovacích jazyků a jejich modelů

Programové paradigma je v informatice způsob abstraktního myšlení o výpočetním procesu, kterým můžeme nahlížet na různé způsoby programování. Asi nejznámější je tzv. procedurální paradigma, které má své počátky už v 50. letech minulého století, a které se vyvinulo z obecnější formy tzv. imperativního programování. Tento způsob znamenal již jistou abstrakci a značný posun od tzv. strojových jazyků, které byly svázány se strojovým kódem daného výpočetního zařízení.

Program je v případě procedurálního paradigmatu strukturován do tzv. procedur (funkcí), které operují nad výpočetními hodnotami (proměnnými daného datového typu).

Typickými zástupci jsou zde programovací jazyky jako např. Fortran, Cobol, Algol, Basic, Pascal a jazyk C, který je z této rodiny jazyků dosud nejvíce zaužíván, viz ukázka na obr. 2.1. Způsob procedurálního programování převládal až do 80. let minulého století, kdy se začaly postupně ukazovat výhody tzv. objektově-orientovaného paradigmatu. Ačkoliv se způsob procedurálního programování zachoval i do dnešní doby, jeho uplatnění je dnes spíše pro tvorbu menších systémů, proto se dodnes tento přístup používá pro tzv. vestavěné systémy (zaměřují se na řízení jednoúčelového zařízení limitovaného svým rozsahem) a také tam, kde potřebujeme nižší rovinu programování s ohledem na druh čipu a jeho požadavky.

Přestože i v procedurálních programovacích jazycích jako je např. jazyk C můžeme vedle práce s jednoduchými datovými typy vytvářet i složené datové typy (označované jako tzv. struktury), které nám zvyšují rovinu abstrakce nad výpočetním procesem, zůstáváme jinak stále omezeni celkovým procedurálním paradigmatem, a to se ukázalo do budoucna jako samo o sobě neudržitelné.

```
/*
 * Funkce Max
 * fuknce vrati maximum ze dvou zadanych cisel
 */

int max (int x, int y) {
      if (x > y) {
            return x;
      } else {
            return y;
      }
}
```

Obr. 2.1: ukázka jednoduché funkce (procedury) v jazyce C (vlastní studie).

Proto v 80. letech minulého století zejména s rozvojem programů i do dalších oblastí a s ohledem na požadavky vytvářet více sofistikované systémy dochází k většímu rozšíření programovacích jazyků založených na tzv. objektově orientovaném paradigmatu (OOP), které se ve své abstrakci snaží více přiblížit realitě našeho světa. Existující entity tohoto světa zde popisujeme jako tzv. objekty, které mají své charakteristické vlastnosti a

metody (chování), kterými mezi sebou vzájemně interagují a vyvíjejí se. Typickými zástupci programovacích jazyků dle OOP jsou např. jazyky jako C++, Java nebo C#. S rozvojem OOP a tím i s rozvojem objektově orientovaných programovacích jazyků vzniká také více potřeba vyvářet tzv. meta popis nezávislý na daném programovacím jazyce a současně umožňující zachytit kategorie, hierarchie, vazby mezi objekty a jejich změnami. Postupně tak ve 2. polovině 90. let minulého století vzniká standard (meta-jazyk) dnes označovaný jako UML (*unified modelling language*), který nám umožní graficky modelovat a znázornit to, o čem OOP ve svých abstrakcích hovoří, aniž bychom museli hned přistoupit k realizaci v konkrétním objektově orientovaném jazyce. Více se tohoto tématu dotkneme v následné kapitole zaměřené právě na OOP.

3. Objektově orientované paradigma

Zatímco jsme v 1. kapitole představili pro lepší kontext ostatní jiné představitele informatických inspirací v řádu božího stvoření, zde se nyní zaměříme na ústřední inspiraci, kterou je oblast objektově orientovaného paradigmatu. V této kapitole připomeneme nejprve základní pojmy z OOP, abychom se pak v pozdější 6. kapitole mohli podívat přehledově na jednotlivé podobnosti s ontologií, které zde můžeme spatřovat.

3.1. Úvod do OOP

Pokud se podíváme zpět na oblast procedurálního programování, setkáme se zde s pojmy, jako jsou proměnné určitého datového typu, které nám podle povahy obsahu rozlišují data na různé typy a mohou sloužit jako nositelé hodnot (tj. kvantity a kvality). Jejich proměnlivost je pak zpravidla nastavována v rámci určité procedury. Už zde lze vytušit jistý pokus o abstrakci reality stvoření do virtuálního informatického prostředí. Avšak proměnné obsahující jen omezený rozsah svého obsahu, jako je tomu u primitivních datových typů (např. číselné, znakové či řetězcové datové typy), nám nemohou poskytnout potřebnou mohutnost abstrakce reality stvoření. O něco blíže se ocitáme u složeného datového typu tzv. struktur (např. v jazyce C typ *struct*), který není jen o jednom primitivním datovém typu, ale může sám obsahovat více těchto různých datových typů. Avšak i zde stále narážíme na mantinely dané samým konceptem procedurálního programování, kde procedury (funkce) nejsou přímo propojeny s těmito složenými datovými typy, ale stojí jen vedle zvlášť. To ale neodpovídá našemu řádu stvoření, kde se změny (procesy či procedury) dějí jako součást určitého jsoucna, resp. bytí, nebo jako

součást jiných jsoucen, které určité jsoucno ovlivňují. Z tohoto důvodu nejsou také procedurální programovací jazyky vhodné pro popis rozsáhlejších systémů, neboť je zde obtížnější vyjádření abstrakce (reálných entit a jejich vztahů), se kterými se při reprezentaci určité reality setkáváme.

S příchodem OOP v 80. letech minulého století se ale vše obrací k lepšímu. Není nám známo, do jaké míry byli sami tvůrci OOP obeznámeni se základy filozofie nebo dokonce s Tomášovo metafyzikou, ale je více než pravděpodobné, že tento koncept nevzniká zcela náhodně, ale staví na jistém stavu předporozumění – poznání či abstrakci řádu stvoření a snaží se mu přiblížit více, než toho byl schopen koncept do té doby zaužívaného procedurálního paradigmatu.

Nyní si postupně představíme základní pojmy objektově orientovaného paradigmatu, které budeme posléze potřebovat při porovnání s termíny Tomášovo metafyziky.

3.2. OOP v přehledu

Základním pojmem v objektově orientovaném programování je termín **třída** (*class*). **Třídu** můžeme chápat jako definici složeného datového typu, která má navíc vedle svých proměnných (mohou být také různého datového typu, a to i složeného) také procedury, resp. funkce. Proměnné třídy pak označujeme jako tzv. **členské proměnné** (*member variables*) nebo také jako tzv. **atributy** či **vlastnosti** (*properties*). Funkce ve třídě pak nazýváme **metodami** a hovoříme o nich jako o schopnostech či dovednostech. Je však třeba mít na paměti, že třída zůstává sama osobě jen jakousi šablonou či definicí složeného datového typu, takže sama o sobě není ničemu prospěšná, dokud není vytvořena její tzv. **instance** (jde o obdobu deklarace proměnné k určitému datovému typu), kdy dochází k vytvoření **objektu**, který je typem (instancí) dané třídy.

Při vytvoření instance třídy (objektu) dochází totiž k přidělení času a prostoru (paměti a případně dalších systémových prostředků PC), podobně jako při **deklaraci**[14] resp. **definici** proměnné typu **int**. Specifickou roli metody zde zastává tzv. konstruktor.

Jedná se o metodu třídy (zpravidla hned na začátku po výčtu vlastností třídy), která se volá při instancování dané třídy, tj. hned při vytváření objektu a slouží zpravidla k počátečním krokům, které má daný objekt hned při vzniku vykonat (např. nastavení hodnot pro své vlastnosti). Ukázka kódu definice třídy pro zamýšlený objekt živočicha je v objektově orientovaném jazyce Java na obr. 3.1.

[14] Deklarací rozumíme v počítači přidělení prostoru paměti např. příkazem „int číslo;" nedochází zde ale k přiřazení konkrétní hodnoty, která se děje až při jeho definici tj. např. příkazem „int číslo = 5;".

```
Public class Zivocich {
      // vlastnosti zivocicha
      int druh;
      String jmeno;

      // konstruktor zivocicha, nastavi prvni vlastnosti
      Zivocich (int druh, String jmeno) {
            // nastav parametry zivocicha
            this.druh = druh;
      }
      // metoda pro vypis jmena
      Public void vypisJmeno () {
            System.out.println(
            "Jméno živočicha je: " + this.jmeno);
      }
      // metoda pro vyvoj zivocicha
      Public void vyvoj () {
            // zde by mel byt kod pro obecny vyvoj zivocichu
      }
      // dale bychom mohli definovat i dalsi metody
}
```

Obr. 3.1: ukázka definice třídy pro zamýšlený objekt živočicha (vlastní studie).

Vytvoření objekt dané třídy pak probíhá pomocí klíčového slova ***new***[15] (společné pro většinu objektově orientovaných jazyků jako např. C++, Java či C#). Ukázka vytvoření objektu živočicha s konkrétním jménem dle definice třídy *Zivocich* je pak na obr. 3.2.

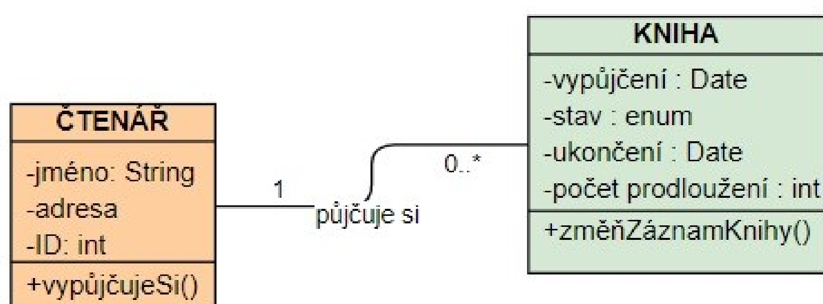
---

[15] Opakem projevu klíčového slova ***new*** je zánik objektu, když už objekt nemá další využití. Na rozdíl od jazyka C++ jsou tyto otázky u moderních objektově orientovaných jazyků jako je Java nebo C# řešeny automaticky skrze tzv. ***Garbage collector***, tedy systémově.

```
// vytvoreni noveho zivocicha do promenne mujZivocich
Zivocich mujZivocich = new Zivocich(2, "Myšák Stuart Little");
// vypsani jmena zivocicha
mujZivocich.vypisJmeno();
```

*Obr. 3.2: vytvoření objektu třídy Zivocich do proměnné mujZivocich a nastavení počátečních hodnot konstruktorem, kde např. hodnota 2 může představovat kód pro savce a řetězcový parametr (tj. text) např. s hodnotou "Myšák Stuart Little" může představovat vlastní jméno tohoto živočicha. Skrze tzv. tečkovou notaci přistupujeme k metodě objektu, a tak se nám po volání metody vypisJmeno(); vypíše tento text: "**Jméno živočicha je: Myšák Stuart Little**" (vlastní studie).*

Mezi třídami, resp. objekty mohou existovat různé závislosti (**vztahy**), tak jako mezi jsoucny reálného světa. Například v případě systému knihovny budeme mít instance tříd *KNIHA* a *ČTENÁŘ*. *ČTENÁŘ* bude mít metodu *Vypůjčuje_si,* která se bude vztahovat k vypůjčení dané knihy reprezentované objektem třídy *KNIHA*. Bude tedy platit, že daný čtenář si vypůjčuje žádnou, jednu nebo více určitých knih a dále také, že danou knihu může mít ale vždy jen jeden návštěvník. Mezi třídami *ČTENÁŘ a KNIHA* bude tedy existovat vztah 0 : N (viz obr. 3.3). V jiném slova smyslu můžeme také chápat vztahy mezi objekty jako jejich vzájemnou komunikaci, v OOP proto někdy hovoříme o tzv. **zasílání zpráv** mezi objekty, kde existují určité vztahy.



Obr. 3.3: ukázka vztahu mezi třídami ČTENÁŘ a KNIHA. Identický vztah bude pak existovat i mezi objekty instancovaných na těchto třídách. Zachyceno skrze diagram tříd dle standardu UML (vlastní studie).

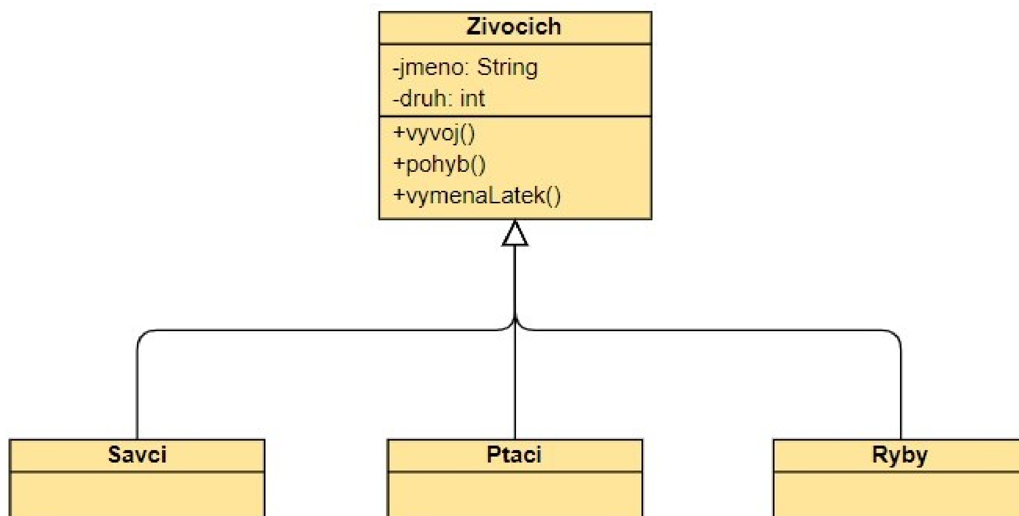
Jednotlivé třídy, které mají určitý společný zájem, resp. potřebu vzájemného provázání můžeme seskupovat do tzv. **balíčků** (*package*). To nám za pomoci tzv. specifikátorů dovolí udělovat určitá přístupová práva pro přístup k atributům a metodám jiných tříd, které budou ve stejném prostoru – balíčku. Tyto specifikátory se pak nemusí vztahovat jenom k jednotlivým třídám, ale také i k určitým atributům a metodám.

Obecně rozlišujeme v OOP následující specifikátory přístupu:

- **Public** – takto označená třída je viditelná pro všechny jiné, tzn. jiné třídy si mohou vyžádat přístup k jejím atributům a metodám, pokud nemají nastaven svůj vlastní restriktivnější specifikátor jako např. *private*.
- **Protected** – takto označená třída je viditelná pro všechny jiné, které jsou s ní ve stejném balíčku.
- **Private** – takto označená třída není viditelná pro ostatní třídy ani ze stejného balíku. Zpravidla je tento specifikátor používán ale častěji u třídy, a to spíše pro atributy nebo metody dané třídy, které chceme uchránit před změnami (zásahy z jiných objektů). Tyto metody a atributy tak budou viditelné (přístupné, tj. i ovlivnitelné) pouze skrze atributy a metody uvnitř této třídy (hovoříme zde o tzv. zapouzdření dat – *data encapsulation*).

Vedle specifikátorů existují také ještě tzv. modifikátory (***asbtract***, ***final***), které nám zasahují do specifického druhu vztahu, kterému říkáme **dědičnost**. Dědičnost je takový vztah mezi třídami, kdy chceme, aby třída potomka obsahovala vše, co obsahovala třída rodiče a zároveň chceme potomku ponechat možnost něco stávajícího změnit (hovoříme zde o tzv. **překrývání metod**), nebo doplnit. Ukázka dědičnosti je na obr. 3.4.

Pokud se podíváme blíže např. na metodu *Vyvoj* (obr. 3.1), je zde zcela jisté, že budeme potřebovat tuto metodu blíže zpřesnit. Můžeme klidně zachovat vnější podobu této metody stejnou (tzv. hlavička metody), ale její vnitřní podobu bude zcela jistě potřeba dále upravit dle specifického vývoje savců. Bude se tedy jednat o tzv. překrytí rodičovské metody.



Obr. 3.4: ukázka vztahu dědičnosti u živočišných druhů. Rodičovská třída Zivocich předává své metody a atributy svým potomkům, zde v podobě tříd Savci, Ptáci, Ryby. Zachyceno skrze diagram tříd standardu UML (vlastní studie).

Podobně se pak bude postupovat u ostatních druhů živočichů. V našem případě bychom mohli ukázat dědičnost s překrytím metod rodičovské třídy např. pro savce. Ukázka takové dědičnosti pro savce v jazyce Java s použitím klíčového slova ***extends*** (v jazyce Java používáme pro vyjádření dědičnosti klíčové slovo ***extends***[16]) je na obr. 3.5.

```
Public class Savci extends Zivocich {
      // konstruktor savce, nastavi prvni vlastnosti
      Savci (int druh, String jmeno) {
            //kontruktor rodice tj. z tridy Zivocich
            super(druh, jmeno);
      }
      // metoda pro vypis jmena, jedna se o prekryti metody
      Public void vypisJmeno () {
            System.out.println("Jméno savce je: " +
      this.jmeno);
      }
      // metoda pro vyvoj savce, jedna se o prekryti metody
      Public void vyvoj () {
            // zde by mel byt kod pro specificky vyvoj savce
      }
}
```

Obr. 3.5: ukázka definice třídy potomka pro zamýšlený objekt savce, (vlastní studie).

Tím se dotýkáme dalšího klíčovému pojmu OOP a tím je tzv. ***polymorfismus***. Tzv. polymorfismem vyjadřujeme, že určitá instance třídy potomka je zároveň instancí své rodičovské třídy. V některých situacích může být také vhodné hovořit o instanci třídy rodiče a ponechávat si tak prostor k vyjádření společného pro všechny instance třídy potomka (například můžeme chtít souhrnně používat společné označení živočich a až v určitém okamžiku nechat odhalit o jaký druh živočicha se v danou chvíli za tímto obecným označením skrývá). Polymorfismus je právě o takovém způsobu práce s objekty, kdy stejné volání metody nad danou proměnnou (objektová reference) může vést k různým projevům, viz ukázka kód na obr. 3.6.

---

[16] V jazyce C++ a v jazyce C# je odvození neboli dědičnost vyjádřena jiným způsobem než klíčovým slovem ***extends*** a sice klíčovým symbolem **dvojtečky**. Zápis naší třídy *Savci* z obr. 3.4 by pak vypadal např. takto: „**class Savci : Zivocich**“.

```
// zavedeni atributu – tj. promenne typu Zivocich
Zivocich mujZivocich;

// vytvoreni noveho objektu tridy Savci
mujZivocich = new Savci(2, "Myšák Stuart Little");

// stejny zapis jako u obr. 3.5, ale vypise jmeno savce
mujZivocich.vypisJmeno ();
```

*Obr. 3.6: ukázka projevu polymorfismu. Atribut mujZivocich je zaveden jako proměnná typu Zivocich. Přesto však po přistoupení k metodě vypisJmeno(); vypíše jiný text, než by vypsala metoda z třídy Zivocich, a sice tento text: "**Jméno savce je: Myšák Stuart Little**". Srovnejme si tento výpis s výstupem na obr. 3.2 (vlastní studie).*

V souvislosti s dědičností jsme výše zmínily tzv. modifikátory[17] ***asbtract*** a ***final***. Ty mají svůj význam i při aplikaci polymorfismu. Modifikátor ***abstract*** u třídy nám hovoří o tom, že z dané třídy nemůže vzniknout přímo objekt, ale může existovat potomek dle pravidel dědičnosti, z něhož už může vzniknout určitý objekt. Pro představu si můžeme takovou abstraktní třídu představit jako třídu *Zivocich*. Přestože tato třída nám v kontextu s naší realitou stvoření dává smysl a je důležitá, v praxi nebude ale nikdy existovat živočich, který by byl jen živočichem sám o sobě (instancí třídy *Zivocich*), ale bude vždy i případem nějakého daného durhu, tj. instancí nějaké konkrétnější třídy jako např. *Savci*.

Naproti tomu nám pak modifikátor ***final*** u třídy vyjadřuje skutečnost, že z této třídy není možné už odvozovat další třídy potomka, taková třída je tedy konečná. To má samozřejmě smysl i v kontextu naší reality stvoření, kdy určitý vývojový druh už považujeme za konečný a nemá v naší současnosti další vývojové druhy (v případě člověka bychom mohli např. hovořit o končené třídě pro druh *Homo sapiens sapiens*).

Pro přehled základních pojmů OOP zde můžeme ještě zmínit tzv. tečkovou notaci. Jde o způsob, jak přistupovat čili pracovat s metodami a atributy objektů, tak aby bylo zřejmé, k jakému objektu se daná metoda nebo atribut váže. Zpravidla má tečková notace tento předpis: „*Název_objektu.název_Atributu_či_Metody*".

V případě, kdy máme k dispozici jen objekt rodiče a chceme ale pracovat s atributy (např. se jménem) potomka, je třeba k němu nejprve přistoupit opět přes tečkovou notaci

---

[17] Zde budeme modifikátory pro určité zjednodušení chápat jen ve spojení s třídami. Budeme zde tedy hovořit např. jen o abstraktních (*abstract*) nebo končených (*final*) třídách, a nikoliv však o takových metodách nebo atributech, ačkoliv se s nimi mohou tyto modifikátory také pojit.

tedy např. takto: „*rodic.potomek.jmeno*“. Příklad tečkové notace je také zřejmý i v ukázce kódu na obr. 3.6. Současně abychom mohli zde uvedené pojmy OOP určitým způsobem jednoznačně schematicky vyjadřovat, používáme k tomu už zmíněný modelovací jazyk UML, kde dle určitého druhu pohledu pak vytváříme specifické diagramy, jako již výše zmíněný diagram tříd na obr. 3.4. Dalším případem je pak také např. diagram objektů.

4. Srovnání filozofických systémů (ontologií)

Ontologie neboli také metafyzika je obecnou teorií bytí a je tedy legitimní součásti studia filozofie. Hlavní úlohou ontologie je vytvořit obraz světa, tak jaký doopravdy je. Pokud bychom se měli zabývat důsledným srovnáním různých ontologií, taková studie by jistě vydaly na nejednu diplomovou či disertační práci, proto se pro potřeby této práce zaměříme na pohled srovnání dřívějších tzv. klasických ontologií se současným pohledem.

Klasickou ontologií rozumíme nauku o bytí, které zde byla od dob Aristotela až po Tomáše Akvinské a dále i do doby neoscholastiky. Většina filozofů až po Kanta se snažila jistým způsobem o vlastní komplexní systém filozofie. Zlomovým bodem je zde Kantova osobnost, neboť v jeho důsledku končí doba tzv. klasické ontologie a otvírá se cesta nové.

Jako zástupce nové ontologie navazující na Kantovu kritiku[18] můžeme zmínit osobnost filozofa Nicolaie Hartmanna (1882-1950), který se staví ostře proti těm filozofickým systémům, které nemají základy postavené na nauce ontologie. Dle jeho argumentace jde totiž o klíčovou roli, dle které je možné odkrýt principy člověka a celého světa a díky ní se filozofie stává bližší reálnému životu. Hartmann ve své práci v návaznosti na Kanta přichází s touto odvážnou tezí:

> *„Kategorie bytí nejsou apriorními principy. Apriorní mohou být jen vhledy, poznatky, úsudky; celý protiklad apriorního a aposteriorního existuje jen v teorii poznání. V ontologii však nejde o poznání, ale o předmět poznání.“*[19]

[18] KANT, Immanuel: Kritika čistého rozumu. Praha: Oikoymenh, 2001.
[19] HARTMANN, Nicolai: Nové cesty ontológie. Bratislava: Pravda, 1976

Vlastní tezi nové ontologie pak Hartmann formuluje následovně:

> *„Cestou nové ontologie je tedy kategoriální analýza – postup, který nesplývá ani s indukcí, ani s dedukcí, a který nevyvrátí ani čistě apriorní a ani čistě aposteriorní poznání. Předpokládá celou šířku zkušenosti, tedy i každodenní zkušenost, i zkušenost praktického života a vědy.“*[20]

Hartmannova hypotéza tak obsahuje na první pohled určitou iracionalitu. Ta spočívá v tom, že principy bytí jsou svým způsobem zakódovány v jsoucnech, a proto se nám nejeví samozřejmé. Přesto je zapotřebí vnořovat se do jsoucen, abychom mohli najít klíč a tyto principy odhalit.

O něco takového se pokouší svým způsobem i OOP, když studuje nám známá jsoucna a základě poznání pak dochází k jejich meta-popisu, tedy ke vzniku jistého paradigmatu, který není sám o sobě konkrétním programovacím jazykem, ale je obecnějším principem, z něhož lze objektově orientovaný jazyk v dané konkrétní podobě utvořit. I proto se v této práci nezabýváme toliko konkrétními objektově orientovanými jazyky, ale právě více jejich společným paradigmatem, abychom mohli ukázat na jeho odvozenost ze jsoucen a podobnost tedy s existujícími ontologiemi (více o tomto pojednáme v 6. kapitole).

Dle Hartmanna také můžeme říct, že tzv. substanční paradigma v klasické ontologii nám může mechanizovat pohled na náš svět, což by nás mohlo vést ke změně paradigmatu.

Vedle Hartmanna se k tomuto výkladu kloní také filozof A. N. Whitehead[21]. Hartmann jde ještě dál a hovoří o nové ontologii jako každodenní realitě, neboť se nachází ve své časovosti a individuálnosti, což není stejné jako v klasické ontologii, kterou lze vidět jen jako rovinu abstrakce. Samotné plynutí, tj. projevy individualit v časovosti, je pro Hartmanna v nové ontologii klíčové. Whitehead zde dále navazuje a dokonce říká, že substanční paradigma by mělo být nahrazeno tzv. událostním paradigmatem, neboť základní jednotkou dle Whiteheada[22] přestává být substance a stává se jí událost jako elementární prvek reality.

---

[20] HARTMANN, Nicolai: Nové cesty ontológie. Bratislava: Pravda, 1976

[21] WHITEHEAD, North, Alfred: Process and Reality. Londýn: Free Press, 2010.

[22] tamtéž

Zde by se po přečtení výše uvedeného mohla nabízet otázka, proč se v této práci (jak z jejího názvu také plyne) držíme i přes výše napsané právě srovnání OOP s tzv. klasickou ontologií se substančním paradigmatem. Jako odpověď můžeme jednoduše říct, b že i přes Hartmannovu a Whiteheadovu kritiku klasické ontologie, je to právě ta klasická ontologie, jež oslovuje některé lidské snahy inspirovat se. A právě v OOP můžeme vidět takovou snahu inspirovat se abstrakcí jsoucen, která je snadno uchopitelná.

Protože se samo OOP drží přímých abstrakcí (zřejmě z důvodu jisté přímosti a srozumitelnosti), tak že jsou jeho modely snadno odvoditelné od tohoto světa, existuje zde právě podobnost (inspirovanost) mezi metafyzikou Tomáše a OOP, resp. jistá podobnost OOP k hierarchii kategorií.

Dalším důvodem, proč srovnáváme právě Tomášovu metafyziku s OOP lze spatřovat i v tom, že Tomáš ve své metafyzice současně říká, jak Bůh vidí a tvoří tento svět. My lidé se dle toho inspirujeme a právě např. skrze OOP pak tvoříme i náš virtuální svět podobným způsobem. Z těchto důvodů si dovolujeme zde porovnat OOP právě s Tomášovo metafyzikou.

5. Metafyzika Tomáše Akvinského

Jsoucno a bytí, jsou dva základní termíny v oblasti ontologie. Řecký filozof Aristoteles používá pro vyjádření ontologie pojem první filozofie a zahrnuje ji do obecnější oblasti tzv. metafyziky, tj. oblasti zabývající se otázkou přesahující jen smyslové poznávání.
Tyto starověké termíny metafyziky vzkřísil znovu pro středověkou i současnou teologii zejména sv. Tomáš Akvinský, který ale také poznání dle Aristotela rozvádí a opravuje ještě dále, aby se přiblížil více skutečnému. O jsoucnu a bytí píše Tomáš Akvinský už ve svém raném díle „*O jsoucnu a bytnosti*“ roku 1255 (český překlad vyšel u nás pod názvem „*O bytí a bytnosti*“[23]).

Zatímco **jsoucnem** rozumíme „něco co, je“ (obecně můžeme myslet jsoucno reálné nebo nereálné – abstraktní), **bytím** pak rozumíme akt (dění), který díky své přítomnosti ve stvořených jsoucnech dává těmto jsoucnům to, že existují. Bytí je tedy neoddělitelnou součástí stvořených jsoucen a nelze je od těchto jsoucen oddělit.

Jinými slovy můžeme také říci, že jsoucno je to, co má své bytí (má jednotu své podstaty a existence), proto je každé bytí pro dané jsoucno také odlišné.

---

[23] AKVINSKÝ, Tomáš. *O bytí a bytnosti*, český překlad. Praha: Nákladem dědictví sv. Prokopa, 1887.

Substance – stabilní stránka jsoucna neboli také podstata. O substanci platí, že nemá žádného dalšího nositele, že existuje sama o sobě. Dále zde rozeznáváme dle Tomáše:

- **první substanci** (konkrétní individuální podstata, která je jen vlastní), ta se dále skládá z **látky** a **formy**.
- **druhou substanci** neboli druhou podstatu (myšleno druhová nebo rodová přirozenost věci). Ta je pro daná jsoucna společná. Např. z věty „*Můj kamarád Pavel je také člověk*" můžeme získat informaci o příslušnosti Pavla k jeho vlastní individuální podstatě (první substanci), ale i informaci o jeho příslušnosti ke společné lidské přirozenosti (druhé substanci).

Můžeme si představit substanci na příkladu konkrétního člověka. Ten je příkladem první substance a má také druhovou přirozenost (druhou substanci). První substance je jeho individuální podstatná složka, která bude vlastní jen jemu. Druhá část substance bude pak představovat lidskou podstatu, která je nám lidem společná. Ta je rozumem abstrahovatelná a jako abstrahovaná existuje jen jako pojem v rozumu, tj. pomyslně.

K pojmu **druhé substance** se od dob Porfyriových[24] dále pojí také otázka tzv. **univerzálií**. Pojem univerzálií Tomáš čerpá z Aristotela a dále jej opracovává. Univerzália jsou dle Tomáše (shodně také dle dalších realistů jako Johna Scota, Anselma z Canterbury a Alberta Velikého) chápány jako obecniny, které ale reálně existují a nejsou jen součástí konkrétních věcí. Podle Tomáše tak existují tři typy univerzálií:

- univerzália jako pojmy „před konkrétními jsoucny, tj. věcmi" (lat. *ante res*), které chápeme jako ideové předobrazy o jednotlivých bytí v božském intelektu,
- univerzália jako substanční formy jednotlivých bytí „v konkrétních jsoucnech, tj. věcech " (lat. *in rebus*),
- univerzália jako pojmy „po konkrétních jsoucnech, tj. po věcech" (lat. *post res*), které vznikají dodatečně rozumovou abstrakcí jednotlivých lidských bytí.

Protikladem univerzálií je pohled na konkrétní jsoucna, a to živá či neživá, které nazýváme **jednotlivinami**. Dle Tomáše univerzália jakožto Boží ideje existují nezávisle na jednotlivinách, pokud jimi však myslíme obecné pojmy, pak jsou na nich závislé.

Vedle pojmu substance je třeba zmínit také pojem akcident. **Akcident** je takové jsoucno, díky němuž se první substance proměňuje, např. přijetím moudrosti se Sokratés změní, stane se moudrým. Tím, že první substance ztrácí nebo získávají nějaké akcidenty, dochází tak k jejich změnám. Současně je podstatné říct, že akcidenty v přirozeném řádu nemohou existovat samy o sobě, ale jsou závislé na svém nositeli – substanci.

---

[24] Filozof Porfyrios z Tyru (232–304 n.l.)

U těchto akcidentů se můžeme dále zabývat otázkou jejich množství nebo jakosti, ve smyslu množství hovoříme o kvantitě a ve smyslu jakosti pak zase o kvalitě.

Nakonec se u akcidentů ke slovu dostávají také **vztahy**, které nám ukazují vazby na jiná jsoucna, bez kterých se ale může jsoucno obejít, aniž by ztratilo svou podstatu.

6. Srovnání Tomášovo metafyziky a OOP

V tomto srovnávacím přehledu zkusíme vedle sebe postavit některé klíčové pojmy OOP a ontologie a podíváme se, zda mezi některými nemůže být vztah podobnosti. V diagramech jazyka UML můžeme skutečnosti dle OOP vyjadřovat buďto rovinou pojmových obecnin, tomu odpovídá např. diagram tříd, nebo skrze pohled už na existující konkrétní objekty, tomu odpovídá např. diagram objektů. První případ nám ve vyjádření v podobě **tříd**[25] umožňuje hovořit o něčem, co sice už možná existuje, ale modelujeme zde jen popisující myšlenky jako obecniny, nikoliv však konkrétní věci vzniklé z těchto myšlenek. Dle terminologie Tomáše se tak jedná o druhou substanci (druhová nebo rodová přirozenost věci). Dá se tedy napsat následující vztah:

$$\text{třída} \approx \text{druhová přirozenost dle Tomáše}$$

Rovnice 6.1: vztah mezi třídou v OOP a kategoriemi dle Tomáše (vlastní studie).

V určitých případech bychom mohli obdobně jako u vztahu 6.1. také hovořit o podobnosti mezi pojmem třída dle OOP a tzv. třetí typ univerzálií (lat. *post res*), viz podkapitola 7.3.

Při pohledu na konkrétní věci čili objekty dle OOP už ale musí z třídy existovat určitá instance čili objekt (něco muselo z myšlenky už povstat). Pak tedy můžeme napsat, že instance třídy čili objekt odpovídající určité věci je už určitým konkrétním jsoucnem, tj. **jednotlivinou**. Můžeme tedy uvést následující vztah:

$$\text{objekt} \approx \text{jednotlivina}$$

Rovnice 6.2: vztah mezi objektem v OOP a kategoriemi dle Tomáše (vlastní studie).

O akcidentech jsme řekli, že jde o jsoucna, která vyžadují svého nositele, a díky nimž se mění např. daný člověk (jako např. Sokrates, Tomáš apod.). Obdobně v OOP mohou

[25] Pojem třída (angl. *class*) má v OOP už odlišný význam, než jak jej známe jen v matematice. Ne náhodou je zde určitá podobnost s pojmem tříd z oboru biologie (viz pojmy dědičnost, předek či potomek).

objekty mít své atributy, které nazývány také jako proměnné třídy. Tyto atributy nemohou také existovat sami o sobě, ale jsou vždy spojeny s nějakým objektem. Podobně jako akcidenty proměňují svého nositele, děje se tak i s atributy u objektů, můžeme toto tedy vyjádřit následujícím vztahem:

$$\text{atribut objektu} \approx \text{akcident}$$

Rovnice 6.3: vztah mezi atributem objektu v OOP a kategoriemi dle Tomáše (vlastní studie).

Pokud jde o **kvantitu** daného akcidentu, tu můžeme vyjádřit hodnotou přiřazenou do proměnné v daném rozsahu. V případě vyjádření **kvality** bychom zde potřebovali druhý rozměr dané proměnné, to nám ale jednoduché datové typy pro čísla, znaky a řetězce neumožnují. Na druhé straně nám ale v OOP nic nám nebrání v tom, abychom takový atribut vyjádřili ne skrze základní datové typy, ale skrze vlastní třídu, resp. objekt, který může v sobě obsahovat další proměnné, které budou vypovídat o kvantitě i kvalitě daného atributu.

Při pohledu na **vztahy** mezi jsoucny nám nic nebrání vyjádřit tyto vztahy způsobem vlastním OOP. Na rovině diagramu tříd nebo diagramu objektů dle notace UML můžeme tak vyjadřovat různé případy vztahů, které mohou mezi objekty jakožto jsoucny vyvstávat, viz např. obr. 3.3. resp. obr. 3.4.

Dále se pokusíme pomocí OOP vyjádřit i další pojmy z ontologie. Zde nemůžeme už ale tvrdit, že daný model OOP může vést k danému termínu ontologie, ale můžeme tvrdit, že daný pojem z ontologie lze v OOP vyjádřit podobným modelem.

Představme si situaci, kdy budeme chtít ochránit některé jsoucno nebo jeho část před změnou od jiných jsoucen. Toto lze v OOP vyjádřit např. pomocí specifikátorů **private,** resp. **protected** (např. v případě možných změn jen v rámci vlastního druhu).

V závěru této kapitoly se ještě zastavíme u pojmu **dědičnosti**. Tu můžeme demonstrovat na příkladu božské ideje o naší přirozenosti a naší reálnou přirozeností. Božskou ideu o lidské přirozenosti můžeme modelovat jako rodičovskou třídu a lidskou reálnou přirozenost pak jako z ní odvozenou třídu, která reprezentuje lidskou přirozenost už poznamenanou dědičným (prvotním) hříchem. Přestože není v boží ideji o lidské přirozenosti obsažen prvotní hřích, principem dědičnosti z této boží ideje můžeme dle OOP odvodit lidskou přirozenost, která už je zasažena dědičným hříchem. Neboť princip dědičnosti v OOP u potomka dovoluje, aby obsahoval něco jinak, než jak bylo obsaženo

v rodiči (tzn. potomkovi může i např. chybět něco, co u rodiče bylo). Dle UML tuto skutečnost znázorňujeme na příkladu lidské přirozenosti na obr. 6.4.

Obr. 6.4: ukázka dědičnosti z BOŽSKÉ IDEJE O LIDSKÉ PŘIROZENOSTI. (vlastní studie).

7. Ukázka přínosu na životním cyklu SW

V této kapitole se znovu dotkneme ústředního tématu této práce. Na konkrétním případu budeme demonstrovat podobnost metafyziky dle Tomáše a terminologie v OOP. Tyto pojmy se pokusíme také vyjádřit odpovídající reprezentací v modelovacím jazyce UML. Vycházíme z předpokladu o existence podobnosti mezi pojmy OOP a pojmy ontologie, který jsme se pokusili nastínit v minulé kapitole. V takovém případě budou abstrakce reálných jsoucen tvořené během návrhu programů podobné abstrakcím těchto jsoucen skrze ontologii. Můžeme zde spatřovat nejen podobnost ale i obdobný cíl, v návrhu programu dle OOP se totiž snažíme porozumět daným jsoucnům, abychom je mohli modelovat a tímto modelem skrze daný program věrohodně ve virtuálním prostoru výpočetní techniky pracovat s těmito jsoucny. Podobný cíl můžeme také spatřovat i v případě ontologie, kde nám jde také o věrohodné a logické popisování systému jsoucen, navíc ještě s odkazem na svého stvořitele.

V následující případové studii se pokusíme ukázat na praktickém příkladu způsob návrhu konkrétního programu dle OOP, ve kterém jsou zřejmé podobnosti s cíli a pojmy ontologie. Můžeme zde tedy hovořit opět o „*opisování*" nebo také o inspiraci v principech stvoření.

Současně můžeme také předpokládat, že čím věrnější popis systému jsoucen bude (čím přesnější bude naše inspirovanost), tím věrnější bude i implementovaný model v daném

programu, a tím spíše lze očekávat větší přínos i pro koncové uživatele. Ne náhodou se tak situace podobá situacím u evolučních algoritmů, kde je určité řešení běžnou algoritmickou cestou nedosažitelné nebo neoptimální.

7.1. Inspirace skrze návrhové principy či vzory

Před uvedením naší případové studie si ve stručnosti doplníme ještě dva klíčové termíny z OOP. Oba níže uvedené klíčové pojmy jsou součástí slovníku OOP a pracuje se s nimi na rovině návrhu SW, který může být řízena různými strategiemi. V různých návrhových strategiích pak využíváme popisu v jazyce UML nebo slovního neformální popisu. Se slovním neformálním popisem se více setkáme u strategie v podobě UX řízeného vývoje SW, kde je vývoj zaměřen na práci a výstupy kolem UX designu a vzniká přitom jádro celého procesu, které označujeme jako DESIGN SYSTEM (DS[26]). Do životního cyklu DS jsou zapojeni i vývojáři a ostatní odbornosti. Tento DS se pak stává podkladem pro vznik koncového SW (s určitou nadsázkou můžeme říct, že daný SW je specifickým způsobem generován z DS).

První z těchto doplňujících pojmů je tzv. **návrhový vzor** (*design pattern*). Návrhovým vzorem rozumíme takový předpis – vzor, který je složen z množiny tříd neboli entit s jejich vzájemnými vztahy, a který je jako určitý celek řešení daného problému či situace opakovaně použitelný. Cílem je zde vytvářet znovupoužitelné vzory, které budou již připravenými stavebními kameny pro opakované situace napříč různými programy.

Druhý z těchto termínů je tzv. **návrhový princip** (*design principle*). Zatímco návrhový vzor nám představuje sice opakovaně použitelnou ale již konkrétní formu možného řešení (zachytitelnou např. ve formě UML diagramu tříd nebo kódu daného programovacího jazyka), návrhový princip je obecnější pojem, neboť je jistou rovinou abstrakce nad návrhovým vzorem. Protože samotný návrhový principy je také již jistou rovinou abstrakce, můžeme o návrhovém principu také hovořit jako o návrhové meta-abstrakci, nebo jako o návrhovém meta-vzoru. Takový meta-vzor je však těžko uchopitelný v daném programovacím jazyce nebo diagramech UML, proto jej lze zpravidla vyjádřit jen skrze textový popis, kde se snažíme používat alespoň vhodná klíčová slova.

---

[26] Viz např. design system společnosti **Emplifi** označovaný příznačně jako **Soul**, prezentace tohoto DS je dostupná pod tímto odkazem: https://soul.emplifi.io/latest/soul-design-system-vp5F4DxK

Tyto pojmy jako návrhový princip či vzor nám budou následně vyvstávat během níže prezentované případové studie podobně jako jiné pojmy z oblasti OOP, které jsou také blízké terminologii ontologie.

7.2. Člověk jako návrhový vzor pro své programy

V již dříve publikované studii[27] jsme měli možnost se zabývat potřebami osob s mentálním postižením ve smyslu dostupnosti a přístupnosti programů, které by byly pro tyto uživatele dobře použitelné a přínosné. Ve spolupráci se skupinou speciálních pedagogů tak došlo k definici potřebných programů pro tablety (tzv. apek), které nebyly tehdy na trhu dostupné, anebo nebyly dostupné v očekávané kvalitě. Následně vyvstala otázka, jak tyto apky navrhnout a naprogramovat (implementovat), aby splňovaly očekávaný přínos pro své uživatele. Tento úkol byl pro nás výzvou, abychom se již tehdy v dané situaci zabývali otázkou dobrého návrhu inspirovaného v samotném stvoření. Myšlenka zde vycházela z prostého výše uvedeného předpokladu, čím lépe budou tyto apky inspirovány svými vlastními uživateli (budeme se nořit do těchto jsoucen), jejich životem, tím přesněji budou moci odpovídat jejich požadavkům v celkovém slova smyslu. Využili jsme tedy maximálně možných prostředků OOP a vhodné vývojové strategie, abychom tak mohli vytvořit očekávanou věrnou reprezentaci reálných jsoucen s jejich substancemi i akcidenty (tj. objekty, atributy a vztahy).

Během několikaleté práce s uživateli s mentálním postižením jsme postupně identifikovali celkem 14 návrhových principů, které by měly být promítnuty v každé dobrém návrhu apky pro mentálně postižené uživatele, aby byly optimálně zachovány parametry přístupnosti a použitelnosti u dané aplikace měřitelné skrze pohled HCI. Tyto návrhové principy se nám staly základem pro náš design system označovaný jako „***computer as therapy design system***“ (*i-CT* DS) na jehož podkladě pak probíhal vývoj (dle vývojové strategie v podobě UX řízeného SW vývoje) vlastních apek, jak si ukážeme dále.

Abychom mohli nacházet vhodné návrhové principy či vzory pro aplikace mentálně postižené uživatele, ukázala se zde také vhodné být s těmito uživateli v jejich přirozeném prostředí a nutnost stát se zde dobrými pozorovateli (viz otázka zkušenosti a noření se do jsoucen). Poukázali jsme tak na skutečnost, že nejlepším návrhovým principem pro

[27] FIALA, Jiří. – KOČÍ, Radek. Počítačová terapie jako koncept nové formy terapie pro osoby s mentálním postižením: teorie i praxe. *Journal of Technology and Information Education* 2014, roč. 6, č. 1, s. 89–103.

aplikace postižených uživatelů jsou tito uživatelé sami o sobě, viz také dřívější publikace na toto téma.[28] Zkusme se nyní na tyto výsledky podívat podrobněji ve světle zaměření této práce.

7.3. Od člověka přes návrhové principy až k SW

Abychom mohli začít vytvářet návrhové principy popisující jsoucna uživatelů a jejich ostatních entit, musíme si nejprve určit, na co se v popisu budeme zaměřovat, neboť není v našich silách uchopit celé lidské bytí.

V našem případě se snažíme postihnout dvě podstatné skutečnosti, jednak skutečnost zachycující odlišné vzory ve způsobu uvažování chování mentálně postižených a za druhé pak skutečnost zachycující naše vlastní myšlení i chování, protože každá apka obsahuje jak část práce kolem potřeb postiženého uživatele, tak i část pro práce asistenta nebo pedagoga. Současně se zaměřujeme jen na ty vzory chování a způsoby uvažování kolem potřeb, které se dotýkají cílů dané apky (jako může např. být trénování grafomotoriky). Výstupem je pak popis procesu kolem daných potřeb, přizpůsobování našeho myšlení a jednání vůči těmto uživatelů v dané oblasti potřeb.

Tento výstup lze v jisté rovině abstrakce vyjádřit tzv. semi-formálním popisem (blízké k tzv. *scenarios* v HCI a diagramu případu použití), který už je jen krok od vytvoření návrhového principu. Zde popisujeme uživatele a jeho uvažování o daných potřebách a způsobech, jak s těmito potřebami pracuje. Z pohledu ontologie, bychom zde řekli, že nás zde zajímají jednotliviny a jejich hierarchie a vztahy. Ze semi-formálních popisů pak můžeme extrahovat zobecňující principy, které lze rozčlenit do jednotlivých kategorií dle toho, čeho se dotýkají. Ve smyslu ontologie bychom zde hovořili o meta-universaliích 3. typu.

Pokud chceme následně směřovat k cílové apce, je třeba tento návrhový princip vyjádřit jednou z jeho možných realizací a sice konkrétním návrhovým vzorem. Daný návrhový vzor lze pak formálně popsat např. diagramem tříd (model abstrakce entit s jejich atributy a vztahy) zachytitelného např. skrze modelovací jazyk UML. Na obr. 7.1 je zachycen návrhový vzor asistence a správy, který je jednou z možných realizací obecnějšího stejnojmenného návrhového principu.[29]

---

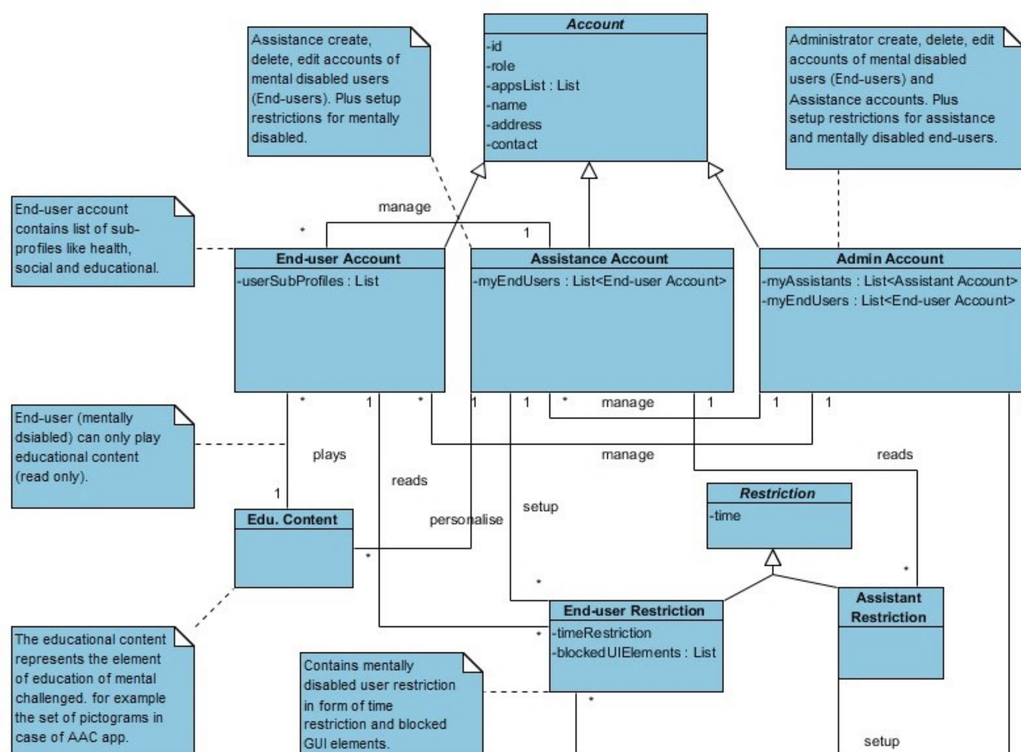
[28] FIALA, Jiří. – ZENDULKA, Jaroslav. Mentally challenged as design principles and models for their applications. *Applied Computer Science* 2016, roč. 12, č. 4, s. 28–48.

[29] Návrhový princip **asistenčního a správcovského režimu** je důsledkem potřeby správy a pomoci skrze centrální správce – pedagogy či jejich asistenty, kteří by měli mít v dané apce možnost spravovat účty uživatelů, aby mohli ukládat změny, provádět nastavení, které přesahují možnosti uživatele s postižením.

I v tomto případě bychom z pohledu ontologie mohli hovořit universaliích 3. typu (přesněji o rodině univerzálií), avšak už bez přídomku meta. Při detailním pohledu pak zde můžeme vidět konkrétní třídy, jejichž instance (objekty) korespondují s jednotlivinami. Dále pak také atributy těchto tříd, které zase odpovídají akcidentům těchto jednotlivin. Nakonec pak lze rozpoznávat i vztahy mezi třídami odpovídající vztahům mezi vlastními jednotlivinami.

Z takového návrhové vzoru vyjádřeného např. skrze diagram tříd jazyka UML lze pak v některých vývojových prostředích přímo generovat zdrojový kód např. v jazyce Java, který se pak po doprogramování nezbytných náležitostí mění v koncovou apku. Tento přístup bývá někdy označován jako tzv. *MODEL DRIVEN DEVELOPEMENT*, avšak v případě, kdy jsou modelem podklady na rovině DS z UX designu, je přesnější hovořit o tzv. *UX DRIVEN DEVELOPMENT* (strategie vývoje SW UX procesem).

Současně je podstatné uvést, že v našem případě získala implementovaná apka podobu aplikační nadstavby pracovně označované jako tzv. *i-CT Framework*.



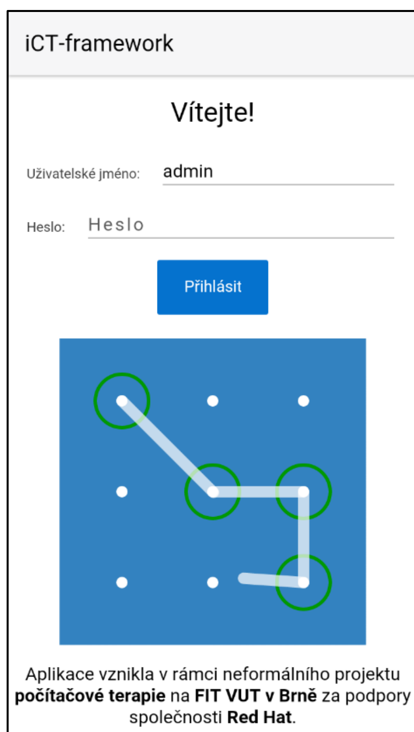
Obr. 7.1: ukázka možné realizace návrhového principu správcovského a asistenčního režimu jako diagram tříd v UML. Zde je třída pro postižené uživatele, třída pro správce a třída pro pedagoga (asistenta). Všechny tyto třídy mají společného předka, proto jsou odvozeny od třídy účet. Dále jsou zde metody a atributy tříd a vazby (vlastní studie).

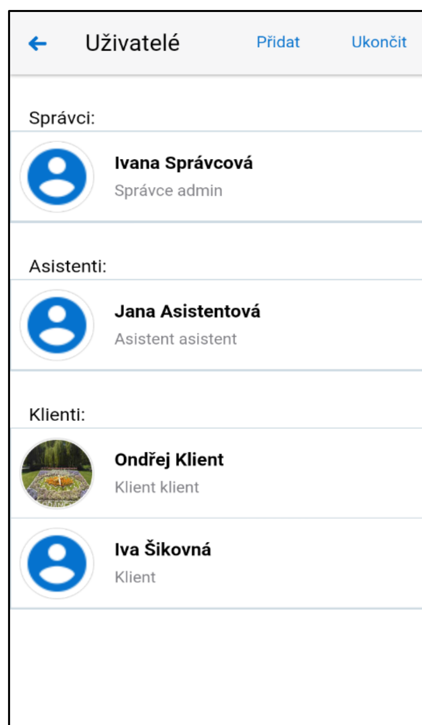
Tento *i-CT Framework*[30] je pak schopen pracovat nad jednotlivými menšími aplikacemi, a tak jim poskytovat potřebná rozšíření, která by jim jinak scházela.

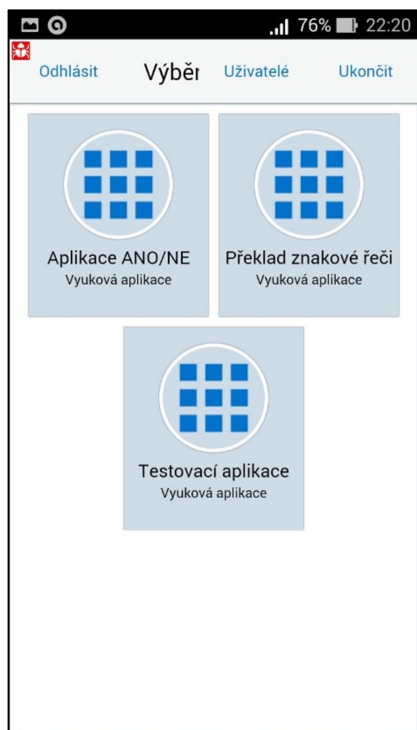
V rámci naší implementace bylo také vytvořeno i několik menších ukázkových aplikací pro výuku postižených uživatelů. Tyto ukázkové apky jsou zde schopny s touto aplikační nadstavbou komunikovat skrze předdefinovaný protokol, aby jim tak mohly být poskytovány potřebné služby odpovídajícím jednotlivým nezbytným návrhovým principům.

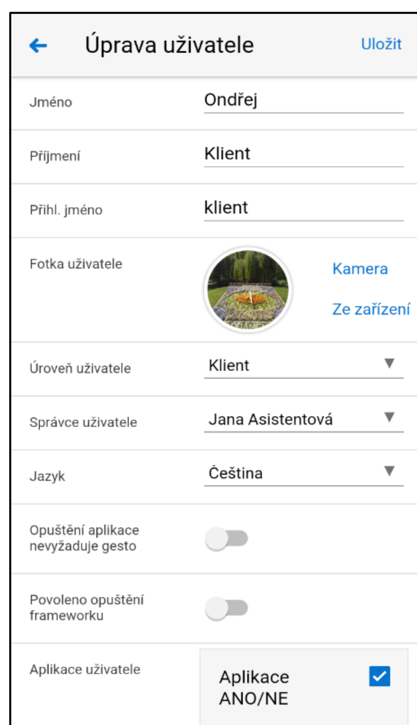
Výsledná apka v podobě nadstavbové aplikace byla implementována pro platformu iOS (tablety iPad) i pro druhou velmi rozšířenou platformu s operačním systémem Android, abychom mohli takto oslovit co nejvíce postižených uživatelů bez omezování jejich možností. Grafické uživatelské rozhraní bylo také vytvářeno s ohledem na již zmíněné nezbytné návrhové principy. Na obr. 7.2 je vyobrazeno koncové grafické dotykové rozhraní této nadstavbové apky (tzv. *i-CT Framework*), které je implementací výše uvedeného návrhového principu (asistence a správy) na rovině GUI.

[30] V označení ***i-CT Framework*** je obsažen význam této práce. Písmeno ***i*** zde zastupuje je iPad a znaky ***CT*** jsou zkratky z anglického *Computer as Therapy*, které bylo motem naší práce. Slovo ***Framework*** je označením určité znovu použitelné části programu, která slouží více dalším spolupracujícím aplikacím.









Obr. 7.2: přehled obrazovek aplikační nadstavby (i-CT Framework) reflektující návrhový princip asistence a správy, 1. obrazovka: přihlášení asistenta pedagoga nebo správce, 2. obrazovka: přehled uživatelů a jejich rolí, 3. obrazovka: přehled cílových aplikacích postiženého uživatele propojených s touto nadstavbou, 4. obrazovka: nastavení profilu postiženého uživatele s volbou jeho cílových aplikací (vlastní studie).

7.4. Zhodnocení uvedeného inspirovaného přístupu

V úvodu této kapitoly jsme předpokládaly, že vývojem programu dle OOP s inspirovaností ve stvořeném dosahujeme nejen podobnosti se stvořením a potažmo pojmů ontologie, ale také nás tento přístup může dovést k lepším výsledkům, tj. zejména lepší použitelnost a přístupnost vyvíjených apek např. pro postižené uživatele. Jak jsme již uvedli v podkapitole 1.5. pojmy použitelnost a přístupnost jsou měřitelné metriky dle HCI. Výsledky námi vyvinutých apek je tedy možné porovnávat s jinými podobnými apkami a z toho vyvodit závěr.

Aplikace postavené (resp. spolupracující s nadstavbou uvedeného frameworku) na těchto návrhových principech byly proto testovány koncovými uživateli s mentálním postižením dle metrik UX designu (zejména z pohledu použitelnosti a přístupnosti) a zjištěné výsledky pak srovnány s ostatními podobnými aplikacemi.

Na obr. 7.3 je ukázka srovnání vyvíjené koncové apky (pro nácvik komunikace ANO/NE) spolupracující se svou aplikační nadstavbou *i-CT Frameworku* s jinými dostupnými aplikacemi tohoto druhu. Jako parametry srovnání jsou zde uvedeny ty parametry, které úzce souvisí právě s měřitelnými metrikami použitelnosti a přístupnosti.

Jak můžeme ze srovnání vidět, z naší strany vyvíjená aplikace ANO/NE společně se svou spolupracující nadstavbou dle výše uvedených návrhových principů vyhověla všem očekávaným parametrům z pohledu použitelnosti a přístupnosti. Tento výsledek však nenacházíme u jiných aplikací tohoto druhu.

Výsledky zde zmíněné dřívější naší studie nám tak potvrzují jeden z předpokladů této práce a sice, že vhodně uchopené OOP s vhodnou strategií, která nám při vývoji programů umožňuje naplňovat inspirovanost ve stvořeném, se tak podobá pojmům ontologie a ve výsledku je i koncový SW bližší a přínosnější pro koncového uživatele.

	Yes/No (I Can Do Apps)	YesNoCommunication (Počítačová terapie)	Aplikace ANO/NE (produkt této práce)
Multiplatformita	✗jen iOS	✓Android a iOS[1]	✓Android a iOS
Otevřenost	✗proprietární	✓otevřené	✓otevřené
Rozšiřitelnost	nelze posoudit	✗dvě samost. impl.	✓jediná impl.
Stanovený cíl	✓stanoven	✓stanoven	✓stanoven
Použitelnost	✓offline	✓offline	✓offline (+ online)
Konfigurovatelnost	✗nelze tvořit páry	✓lze tvořit	✓lze tvořit
Bezp. – dat	✓není co chránit	✓přihlášení	✓přihlášení
Bezp. – opuštění	✓Asist. režim iOS	✗na Android ne	✓opouštěcí gesto
Přístupnost	✓zdarma	✓zdarma	✓zdarma

Obr. 7.3: přehled srovnání aplikací zaměřených na základní komunikaci typu ANO/NE pro mentálně postižené uživatele. Parametry srovnání vychází z požadavků přístupnosti a použitelnosti dle metrik HCI. Aplikace vyvíjená na naší straně dle uvedených 14 návrhových principů zde vyhověla více parametrům než jiné srovnatelné aplikace (převzato z vedené diplomové práce[31] na téma aplikační nadstavby i-CT Frameworku).

V námi identifikovaných výše zmíněných čtrnácti návrhových principech můžeme také podobně jako u evolučních algoritmů spatřovat inspirovanost ve stvoření (uživatelé s postižením), kde je stále patrný otisk Boha, a to i přes přítomnost mentálního postižení. Pokud se tedy rozhodujeme pro inspiraci z těchto božích otisků vhodnou metodou (jako je např. popsáno v této práci skrze OOP a vhodnou strategii vývoje) i v otázce softwaru, můžeme vždy jen získat ve srovnání s neinspirovaným vývojovým procesem pro daný SW.

[31] KALINA, Jan. *Vývoj i-CT frameworku a jeho aplikace pro komunikaci typu ANO/NE*, diplomová práce (vedoucí práce Fiala Jiří). Brno: VUT FIT, 2016.

8. Zhodnocení přínosu z pohledu teologie

Teologií obecně rozumíme rozumovou reflexi naší víry. Právě díky daru víry, kterou teologie předpokládá, tak můžeme ve světle víry odkrývat inteligentní principy jako otisky boží péče o své stvoření, tam kde by nemusely být bez světla víry zjevné. Při tomto odkrývání můžeme pracovat s rozumovou reflexí, čímž se rozšiřuje naše rozumové poznání, které pak recipročně dává opět růst i naší víře.

Díky rozumové reflexi, které se věnujeme i v této práci, tak můžeme rozumovými argumenty podpořit důkaz boží existence, který vychází z pěti aposteriorní důkazů Tomáše Akvinského. V 5. z těchto důkazů (tzv. důkaz z účelnosti přírody) Tomáš doslova říká:

> *„Pátá cesta se béře z řízení věcí. Vidíme totiž, že některé věci, které postrádají poznání, totiž přírodní tělesa, jsou činná pro cíl. Což je zřejmé z toho, že vždycky nebo velmi často jsou činná týmže způsobem a dosahují toho, co jest nejlepší. Z toho jest patrné, že ne náhodou, nýbrž z úmyslu docházejí k cíli. Co však nemá poznání, nesměřuje k cíli, leč řízeno někým poznávajícím a rozumějícím, jako šíp lučištníkem. Tedy jest něco rozumějící, jímž všechny přírodní věci jsou řízeny k cíli, a to nazýváme Bohem.“*[32]

Předem také nutno podotknout, že s ohledem na cíl a rozsah této práce současně pomíjíme známé subtility tohoto Tomášova důkazu, které jsou diskutovány v odborných zdrojích.[33] Tomáš zde říká, že účelné uspořádání přírody nemůže být nahodilé. Jinými slovy, vyjádřeno soudobou češtinou, zde tak volně současně říká:

> *„Musí existovat inteligentní princip, který vytváří ve světě řád. Z toho plyne, že musí existovat Ten, který mu dal vzniknout.“*

[32] AKVINSKÝ, Tomáš: *Summa theologiae*. Olomouc: Krystal, 1937. 1. část, 2. otázka, 3. článek.

[33] CARDAL, Roman. Moderní problematizace dokazování Boží existence. *Distance – Revue pro kritické myšlení* 2002, roč. 2002, č. 1, s. 28–48.

Tento Tomášův důkaz vychází z logického předpokladu, že nic není samo o sobě uspořádané. Dnes bychom mohli také odborně hovořit o míře neurčitosti systému neboli o entropii. Tomáš zde úsudkem rozumu dochází k závěru, že ve světě nevznikají uspořádanosti sami od sebe (tj. systémy s nízkou entropií nevznikají náhodou), ale musí nutně existovat Ten, kdo zde bude uspořádanost vytvářet, tj. snižovat entropii cíleně, která je jinak sama od sebe vysoká.

Podobně se chováme i my lidé, když chceme, aby v našich životech nebyl chaos, plánujeme si např. naše události, vytváříme některá pravidla jako např. pravidla silničního provozu, čímž vnášíme do stvoření (systému) řád, který nám umožnuje žít v tomto světě, aby nevládnul v daných oblastech našeho života nekontrolovaný chaos (cíleně musíme snižovat entropii našich systémů, tam kde je to zapotřebí).

Přestože jako lidé jsme schopni vnášet určité řády do našich životů, pozorujeme současně, a to vedle teologického a filozofického poznání také i díky dalším oborům jako např. i díky matematice, lékařství a přírodně vědním oborům, že jistý řád už je přítomen ve stvoření, a to i bez našeho přičinění.

Dle Tomášovo úvahy musel zde tedy logicky existovat prvotní hybatel (resp. uspořadatel), který vtisknul do stvoření prvotní řád, čímž umožnil vzniku i nás samotných (dal vzniknout naší vlastní uspořádanosti), a tedy i naší schopnosti vnímat a vytvářet řády dle našich lidských možností. V důsledku toho můžeme také říct, že ani naše vlastní schopnost vnímat potřeby uspořádanosti a tuto uspořádanost vytvářet není v nás sama od sebe přítomná.

Pokud se jako lidé rozhodneme vytvořit řád pro určitou oblast našeho života a budeme spatřovat takový řád v dané oblasti jako přínosný, a pokud se současně ukáže, že tento řád je jistou analogií toho, což již zde dříve bez našeho přičinění bylo, z toho následně pro nás vyplývá, že takový řád nepřímo v konkrétním případě ukazuje na platnost 5. důkazu boží existence dle Tomáše. Výše uvedené můžeme jinými slovy vyjádřit také následovně:

> *Hleďme, zde je inteligentní princip, který je pro danou oblast našeho života přínosný, ale pamatujme, že zde není sám od sebe, ale je inspirován principem, který zde byl už i bez našeho přičinění. Musel tedy nutně existovat Ten, který mu dal vzniknout.*

Pokud bychom se podívali na tuto situaci v konkrétním kontextu naší práce můžeme uvést následující vztahy:

Teologie

Inteligentní princip = uspořádání světa dle božího řádu

Popis = metafyzika Tomáše Akvinského

Informatika

Inspirovanost inteligentním principem = OOP

Popis = OOP a strategie v jazyce UML

Vyjádřenou volným jazykem tedy můžeme říct, že OOP je tím lidským řádem (přínosným v rovině vývoje SW), který je inspirován řádem, vzniklým bez lidského přičinění dle popisu Tomáše, což v tomto konkrétním případě znamená řádem božím.

Z pohledu přirozené teologie tato skutečnost podporuje výše uvedený 5. Tomášův důkaz o existenci Boha, neboť bez prvotního hybatele (rozuměj uspořadatele), který vnáší první řád, by nebyla Tomášova metafyzika, od níž by nemohla vzniknout lidská inspirace v podobě OOP jako řád pro vývoj SW, a nemohl by tak nastat ani přínos, kterého díky právě existenci OOP při vývoji SW pro uživatele dosahujeme.

Přínos této práce lze tedy spatřovat právě v tom, že podobnost OOP a Tomášovo metafyziky podporuje v konkrétním případě 5. Tomášův důkaz, neboť ukazuje na to, že ve chvíli, kdy virtuální svět začneme vytvářet právě dle vhodně uchopeného OOP, snižuje se entropie tohoto virtuálního světa (viz předchozí kapitola o přínosu v oblasti SW), přičemž je podstatné, že je zde zřejmá podobnost mezi OOP a Tomášovo metafyzikou, která se vztahuje ke stvořenému.

A právě k existenci této podobnosti lze říct, že jako je OOP přínosné pro zvýšení řádu virtuálního světa, podobně je Boží princip popsaný Tomášem Akvinským v jeho metafyzice přínosný pro zvýšení uspořádanosti našeho světa. Nepřímo tak naším konkrétním případem ukazujeme na to, že zde přebíráme inteligentní princip, který musel nutně vzniknout bez našeho přičinění, a který zjevně nemohl vzniknout sám od sebe, čímž ukazujeme opět na legitimitu 5. Tomášova důkazu o boží existenci. V neposlední řadě se můžeme na teologický přínos této práce dívat jako na odkrývání sebesdělení Boha o sobě samém (jak jsme již naznačili také v úvodu této práce), který nám do stvořeného ukryl inteligentní principy, abychom tím, že je budeme objevovat a inspirovat se jimi, mohli v důsledku poznávat jeho moudrost a starost i péči o své stvoření.

Závěr

V této práci jsme si dali za cíl přehledově prezentovat oblasti informatických inspirací se zaměřením na objektově orientované paradigma (OOP). Cílem tohoto zaměření bylo také demonstrovat souvislosti mezi pojmy metafyziky Tomáše Akvinského a OOP. To jsme se pokusili ukázat skrze vhodné příklady (viz 6. kapitola o srovnání a 7. kapitola s případovou studií). Po jednotlivých pojmech jsme postupně ukazovali na vztahy mezi pojmy OOP a ontologií. Ze zde prezentovaných příkladů je zřejmé, že existují nenáhodné souvislosti mezi pojmy OOP a metafyzikou.

Jak jsme již uvedli v úvodu této práce, nejedná se o zcela nové zjištění, ale protože je tato problematika v odborné literatuře souvisejících disciplín (zejména teologie, filozofie a informatiky) málo diskutována, rozhodli jsme se na tyto souvislosti poukázat vlastní způsobem v této práci. Proto jsme do této práce také začlenili případovou studii, která při vývoji reálného programu ukazuje současně na vhodnost použití OOP jakožto určité paralely metafyziky, která umožní vhodně abstraktně uchopovat řád stvořeného a ve výsledku tak proto dosáhne i lepších vlastností ve srovnání s jinými SW téhož druhu.

V této případové studii byla představena práce na vývoji programu (aplikační nadstavby) pro aplikace mentálně postižených uživatelů, která vyžadovala nemalé nároky na přístupnost a praktickou použitelnost těmito uživateli. Vycházeli jsme zde také z vlastní víceleté studie, během které jsme mohli poznat způsob života těchto uživatelů a jejich potřeb a následně na základě této zkušenosti pak odvodit (skrze využití k tomu vhodných prostředků z OOP) požadavky na jejich aplikace a tyto následně převést dle vývoje OOP do hotového programu. Jak bylo také ukázáno, takto vytvořený program byl těmto uživatelům blíže, v čemž můžeme také spatřovat výhody inspirovaného přístupu, který prostředky OOP při vývoji programu mohou nabídnout.

Nakonec tak můžeme vidět přínos této práce zejména ve skutečnosti, že i v na první pohled technicko-informatické činnosti můžeme nacházet odkaz inteligence stvořitele skrze otisk do stvořeného, v němž se inspirujeme, a tak docházet k podpoře rozumovému poznání existence Boha. To nás pak může skrze vhodné inspirace, jakou je např. OOP, vést nejen k lepším technických výsledkům, ale také nás může učit vnímat tuto inspirovanost jako způsob poznávání Boha jako stvořitele skrze jeho stvoření, respektive poznávání jeho inteligentních záměrů ve stvořeném. Toto poznání nás tak může přivádět i k tichému úžasu, kdy zůstáváme v údivu nad boží velikostí a moudrostí.

Ačkoliv při rozkrývání inteligentních záměrů Boha jako stvořitele otisknutého do svého stvoření můžeme někdy také nabývat dojmu objevu velkých věcí, je třeba mít stále na paměti, že při tomto inspirování v principech stvořitele, poznáváme jen nepatrný zlomek z Jeho moudrosti. I kdybychom snad nějakým zázrakem dosáhli ještě vyšších stavů poznání, nikdy přitom nezapomínejme, že toto poznání samo sobě nic neznamená.

Seznam použitých zkratek

- DS = design systém
- GUI = grafické uživatelské rozhraní (*interface*)
- HCI = *human computer interaction*
- HW = hardware
- OOP = objektově orientované paradigma
- SW = software
- UX = *user experience*
- UML = *unified modelling language*
- USD = *use case diagram*

Použitá literatura

1. PRAMENY

1.1 Dokumenty magistéria

JAN PAVEL II. *Fides et ratio*, encyklika ze dne 14. 9. 1998 – český překlad. Praha: Zvon, 1999.

JAN PAVEL II. *Poselství Jana Pavla II. adresované grémiu Papežské akademii věd na téma evoluce*, proneseno dne 22. října 1996. In: EV 17. Bologna: EDB, 1999, č. 1346–1354.

1.2 Monografie

AKVINSKÝ, Tomáš. *O bytí a bytnosti*, český překlad. Praha: Nákladem dědictví sv. Prokopa, 1887.

AKVINSKÝ, Tomáš: *Summa theologiae*. Olomouc: Krystal, 1937.

HARTMANN, Nicolai. *Nové cesty ontologie*. Bratislava: Pravda, 1976.

KANT, Immanuel: *Kritika čistého rozumu*. Praha: Oikoymenh, 2001.

NORMAN, D. A. *The design of everyday things*, revised and expanded edition. Cambridge: MIT Press, 2013.

THOMPSON, Arcy. *On Growth and Form*, the complete revised edition. New York: Dover Publications, 1992.

OESTEREICH, Bernd. *Developing Software with UML: Object-oriented Analysis and Design in Practice*. Amsterdam: Addison-Wesley, 2002.

1.2 Elektronický zdroj

FRANTIŠEK. *Evolution ... is not inconsistent with the notion of creation* (27. 10. 2014) [2022-1-1]. <https://religionnews.com/2014/10/27/pope-francis-evolution-inconsistent-notion-creation/>.

2. SEKUNDÁRNÍ LITERATURA

2.1 Monografie

NIELSEN, Jakob. – BUDIU, Raluca. *Mobile Usability*. Berkley: New Riders, 2013.

POSPÍŠIL, Ctirad V. *Zápolení o pravdu, naději a lidskou důstojnost: Česká katolická teologie 1850–1950 a výzvy přírodních věd*. Praha: Karolinum, 2017.

SOUSEDÍK, Prokop – SVOBODA, David. *Je matematika věda: Mezi formalismem a strukturalismem.* Praha: Triton, 2017.

SOUSEDÍK, Stanislav. *Jsoucno a bytí: Úvod do četby sv. Tomáše Akvinského.* Praha: Křesťanská akademie, 1992.

ŠVARCOVÁ, Iva. *Mentální retardace: vzdělání výchova, sociální péče*. Praha: Portál, 2011.

WHITEHEAD, North, Alfred: *Process and Reality*. Londýn: Free Press, 2010.

2.2 Článek v časopisu

CARDAL, Roman. Moderní problematizace dokazování Boží existence. *Distance - Revue pro kritické myšlení* 2002, roč. 2002, č. 1, s. 28–48.

FIALA, Jiří. – ZENDULKA, Jaroslav. Mentally challenged as design principles and models for their applications. *Applied Computer Science* 2016, roč. 12, č. 4, s. 28–48.

FIALA, Jiří. – KOČÍ, Radek. Počítačová terapie jako koncept nové formy terapie pro osoby s mentálním postižením: teorie i praxe. *Journal of Technology and Information Education* 2014, roč. 6, č. 1, s. 89–103.

2.2 Článek ve sborníku

FIALA, Jiří. Multigramatiky a jejich aplikace. In: *PROCEEDINGS OF THE 13TH CONFERENCE STUDENT EEICT 2007*. Brno: Fakulta elektrotechniky a komunikačních technologií VUT v Brně, 2007, s. 207-209.

2.3 Elektronický zdroj

OUKOPEC, Jindřich. *ČVUT otevírá nový obor, studenty naučí interakčnímu designu* (28. 9. 2016) [2022-1-1]. <https://www.czechdesign.cz/temata-a-rubriky/cvut-otevira-novy-obor-studenty-nauci-interakcnimu-designu>.

PETRŽELKA, Josef. *Aristotelské motivy v Tomášově metafyzice* [2022-1-1]. <https://is.muni.cz/do/rect/el/estud/ff/ps10/phil/web/ta7.html>.

SAMOKHIN, Vadim. *OOP Meets Metaphysics* (1. 4. 2020) [2022-1-1]. <https://wrong-about-everything.github.io/OOP-Meets-Metaphysics/>.

TARKO, Vlad. *The Metaphysics of Object Oriented Programming* (28. 5. 2006) [2022-1-1]. <https://news.softpedia.com/news/The-Metaphysics-of-Object-Oriented-Programming-24906.shtml>.

VAŠÍČEK, Zdeněk. *Biologií inspirované počítače – kartézské genetické programování* [2022-1-1]. < https://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/>.

3. NEPUBLIKOVANÉ PRÁCE

KALINA, Jan. *Vývoj i-CT frameworku a jeho aplikace pro komunikaci typu ANO/NE*, diplomová práce (vedoucí práce Fiala Jiří). Brno: FIT VUT v Brně, 2016.