Zkušenosti s objektově orientovanou analýzou a návrhem

Vojtěch Merunka

ČZU Praha, Provozně ekonomická fakulta, Katedra informatiky, Česká Republika
ČVUT Praha, Fakulta elektrotechnická, Katedra počítačů, Česká Republika

Úvod

V dnešní době jsme svědky toho, jak se vyplnily předpovědi průkopníků z přelomu 80. a 90 let o budoucnosti objektově orientovaného přístupu. První oblastí, kde OOP prokázalo svoje přednosti, bylo programování. Objevily se objektově orientované programovací jazyky (*např. první Smalltalk v roce 1972 a Simula dokonce již v roce 1967*). Na širší rozšíření objektově orientovaných aplikací jsme si však museli ještě 20 let počkat. Nejbližší budoucnost nám zřejmě přinese objektově orientované operační systémy a databáze - pokud nás ale obratní prodejci těch současných nepřesvědčí, že jsou vlastně již teď všechny jejich produkty 100% objektově orientované (*nabízí se tu analogie s používáním přívlastku „Hi-Fi"*).

Ještě přibližně do roku 1993 se na odborných konferencích a v odborných časopisech zcela vážně diskutovalo o tom, jaké a zda vůbec má OOP výhody v oblasti projektování informačních systémů. Tato otázka byla již dnes zodpovězena, neboť dnes až na výjimky žádný CASE prostředek již nepodporuje jinou metodu, než objektově orientovanou. Stejně tak se samotný pojem „objektový" nebo „objektově-orientovaný" stal díky podařenému marketingu největších softwarových dodavatelů běžnou součástí slovníku všech uživatelů výpočetní techniky. OOP však nepřináší jen určité vesměs pozitivní změny do oblasti rychlého vývoje softwarových aplikací a využívání grafických uživatelských rozhraní. To, že aplikace obsahuje barevné grafické rozhraní s tlačítky, menu a ikonami, ještě neznamená, že je objektově orientovaná. Být „objektově orientovaný" totiž vždy znamená tři velmi často na sobě nezávislé vlastnosti:

1. **Jak aplikace funguje.** Mít grafické uživatelské rozhraní ještě vůbec nemusí znamenat, že celý software je objektově orientovaný. je samozřejmě pravda, že OOP se výhodně používá pro sestavení tohoto rozhraní a že většina knihoven v dnešních programovacích jazycích má rysy OOP, ale rozhraní ještě nic neříká o vazbách a vlastnostech entit uvnitř samotného programu v paměti počítače. Tak jako existují neobjektové programy (*které mimochodem nemusejí být špatné*) s grafickým rozhraním, tak existují i programy objektové, které mají jednoduché textové rozhraní bez dnes tolik oblíbené grafiky.
2. **V čem byla aplikace naprogramována.** Pro objektovou aplikaci ve smyslu 1) lze totiž použít (*samozřejmě s různou výhodností*) jakýkoliv programovací jazyk. Například servery objektových databází jsou velmi často programovány v jazyce C (*ne C++*) a nebo přímo ve strojovém kódu a naopak většina aplikací psaných v C++, které je považováno za objektově orientované, obsahuje klasické datové typy a způsob algoritmizace.
3. **Jakou technikou analýzy a návrhu byla aplikace navrhována.** Samotné použití objektového CASE nástroje ještě nemusí znamenat, že výsledek bude aplikace objektová podle bodu 1). Nástroje OOA a OOD totiž dovolují modelovat v duchu

zásad klasické datové a funkční analýzy. Jinými slovy to znamená, že použití objektového diagramu jako nástroje ještě nemusí nutně znamenat, že technika, se kterou je pomocí tohoto nástroje model aplikace vytvářen, je automaticky také objektově orientovaná.

Změny v profesi informatika

Nemělo by se zapomínat na to, proč se vlastně na počátku 70. let vůbec objektová orientace objevila, a proč je dnes jedním z hlavních trendů rozvoje informatiky. Nejde pouze o skvělou budoucnost projevující se v nabídce stále „barevnějších" a lepších (*větších a dražších!*) verzí programů sloužících k uspokojení potřeb zákazníka, jak tvrdí prodejci všeho druhu, ale především o **snahu řešit** již delší dobu trvající krizi programování, která se projevuje rostoucí složitostí programů, nebezpečně nízkou spolehlivostí softwaru a neúměrnými náklady **na** pracovní sílu pro programování a údržbu provozovaného softwaru. Tato ohrožení jsou částečně kompenzována zvyšujícím se výkonem hardware (*kapacita paměti, rychlost*), ne však jeho dokonalejší konstrukcí.

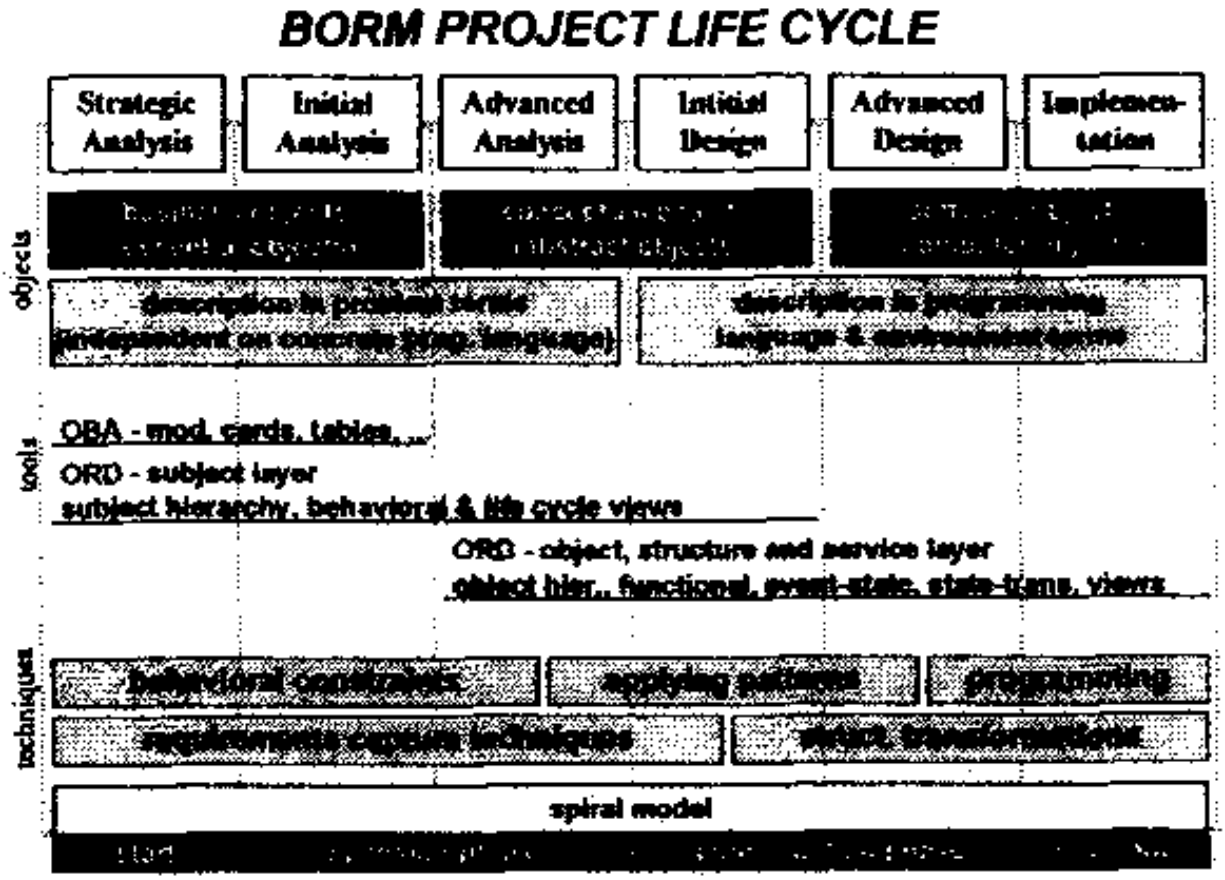
Všechny výše zmíněné problémy s počítači mají základ v původní **von Neumannovské architektuře počítače** (vNA), která dnes neodpovídá abstraktním vyjadřovacím schopnostem moderních programovacích jazyků a požadavkům současných uživatelů na abstraktní komunikaci s počítačem. Bezchybná implementace současných výpočetních systému na klasické von Neumannově architektuře je vzhledem ke své programové složitosti téměř neřešitelný problém. Objektově orientovaný přístup, je právě zatím jedinou úspěšnou variantou vyřešení tohoto problému. Jeho úspěšnost spočívá v tom, že objektový model výpočtu předpokládá **jinou architekturu stroje** než je klasická vNA. Dnes, v polovině 90 let, se výhody OOP již neprojevují jen na speciálně konstruovaných počítačích (*několik jich bylo skutečně za posledních 20 let vyrobeno*), ale začínají se výrazně prosazovat i na klasické architektuře stroje.

V této perspektivě je velmi žádoucí, aby informatik ovládal kromě klasických znalostí (*např. nauka o hardware či algoritmizace*) ještě následují okruhy dovedností [Meyer]:

- schopnost znovuvyužívat v nových projektech již hotové softwarové komponenty a ne se snažit vytvářet všechny části systému znovu (*což je v rozporu se stávajícím pojetím výuky programování a projektování*),
- schopnost kombinovat znalosti v oblasti programovacích jazyků se znalostmi metod analýzy a návrhu IS,
- neorientovat se při tvorbě softwarových aplikací pouze na zvládnutí moderních grafických rozhraní, ale být schopen navrhovat a realizovat i algoritmy „uvnitř aplikace" v duchu zásad OOP (*Domnívat se, že algoritmizaci lze nahradit vizuálním programováním v rozhraní - jak to dělá značná část vývojářů - je politováníhodné*)
- před „strojově orientovanými" programátorskými dovednostmi preferovat abstraktnější úrovně myšlení, jako např. zvládnutí základů matematické logiky, teorie automatů, formálního kalkulu, algeber příslušných datových modelů apod. (*což je vlastně odpovědí na zjednodušenou tezi o „překonanosti programování" - programování je totiž stále třeba, ale „ruční" práce v něm bude stále méně a celý proces se stává stále méně „strojovým" a více „matematičtějším"*).

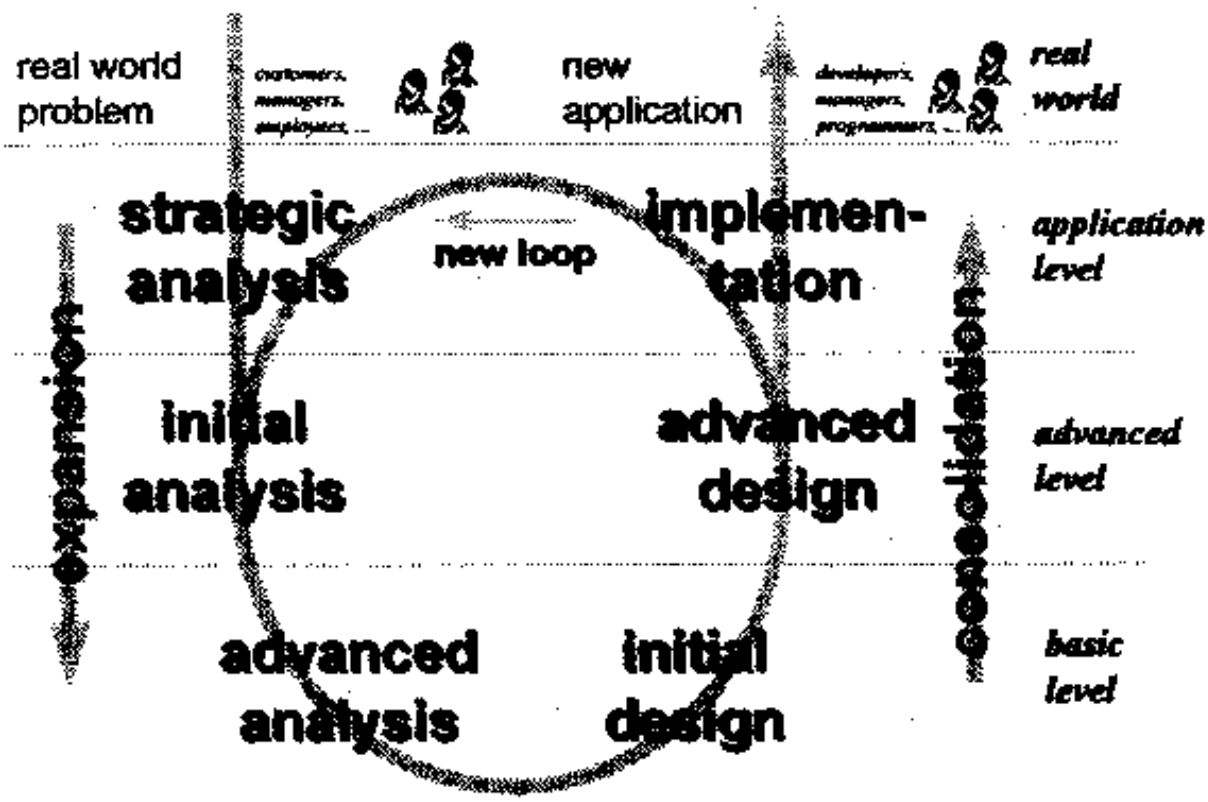
Jak objektově modelovat

Pokud chceme/musíme používat OOP ve všech fázích vývoje životního cyklu IS, jak ukazuje následující obrázek, tak je třeba po celou dobu nějakým způsobem pracovat s pojmem objekt.



Samotný pojem objektu včetně jeho vlastností se však v průběhu vývoje IS mění. Jinak chápe objekt programátor při implementaci v nějakém konkrétním programovacím jazyce a jinak chápe objekt zadavatel, protože pro něj je objekt zobrazením nějaké entity reálného světa, která je v okruhu jeho zájmu při formulaci zadání.

Problém tvorby software lze zjednodušit na komunikaci mezi zadavatelem a implementátorem a celý proces lze znázornit pomocí následujícího schématu:

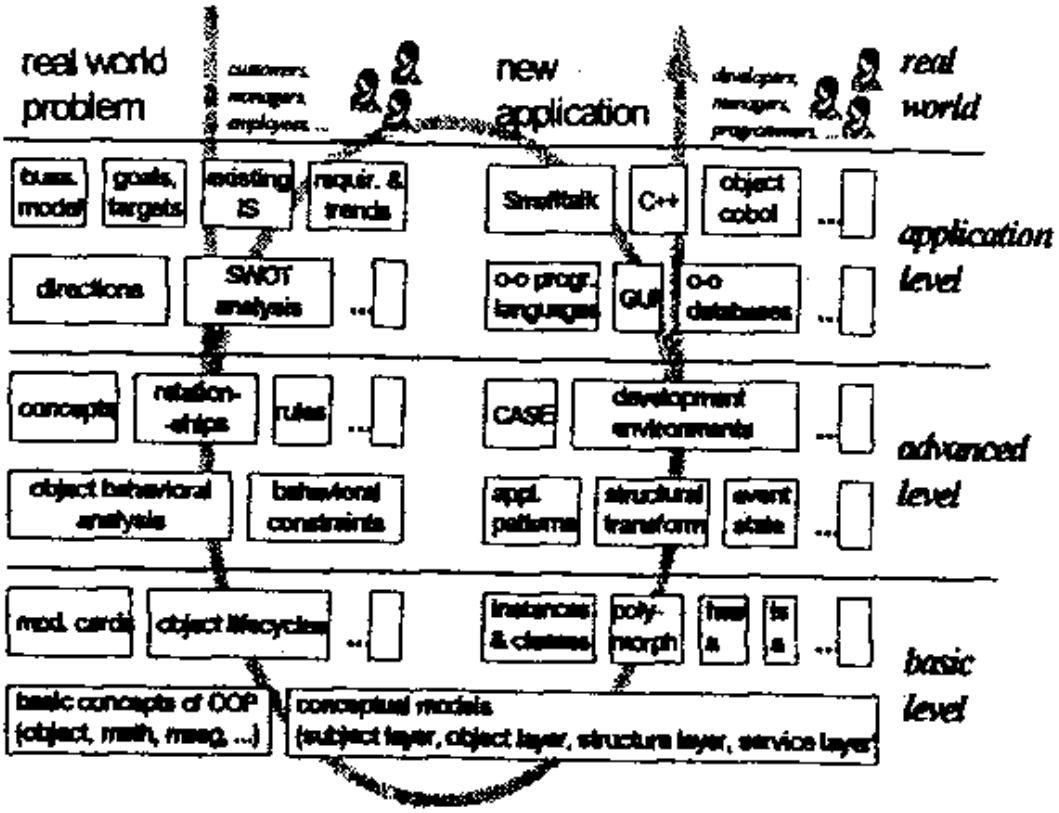


Jeden cyklus vývoje IS může mít různé fáze, ale vždy lze rozpoznat **dvě hlavní stadia**. Prvním je **stadium expanze,** protože při něm dochází ke hromadění informací potřebných pro vytvoření aplikace. Stadium expanze končí s dokončením analytického konceptuálného modelu, který na logické úrovni reprezentuje požadované zadání a formálně popisuje řešený problém.

Zbývající fáze od návrhu přes implementaci k testování a provozu se označují jako **stadium konsolidace**. Je tomu tak proto, že v těchto etapách se model, který je produktem předchozí expanze, postupně stává fungujícím programem, což znamená, že na nějakou myšlenkovou "expanzi" zadání zde již není prostor ani čas. V tomto stadiu je také počítáno s tím, že od některých idejí z expanzního stadia bude třeba upustit vzhledem k časovým, kapacitním, implementačním nebo i intelektuálním schopnostem.

Z pohledu jednotlivých nástrojů a technik je celý proces v detailu následující

BORM - detail



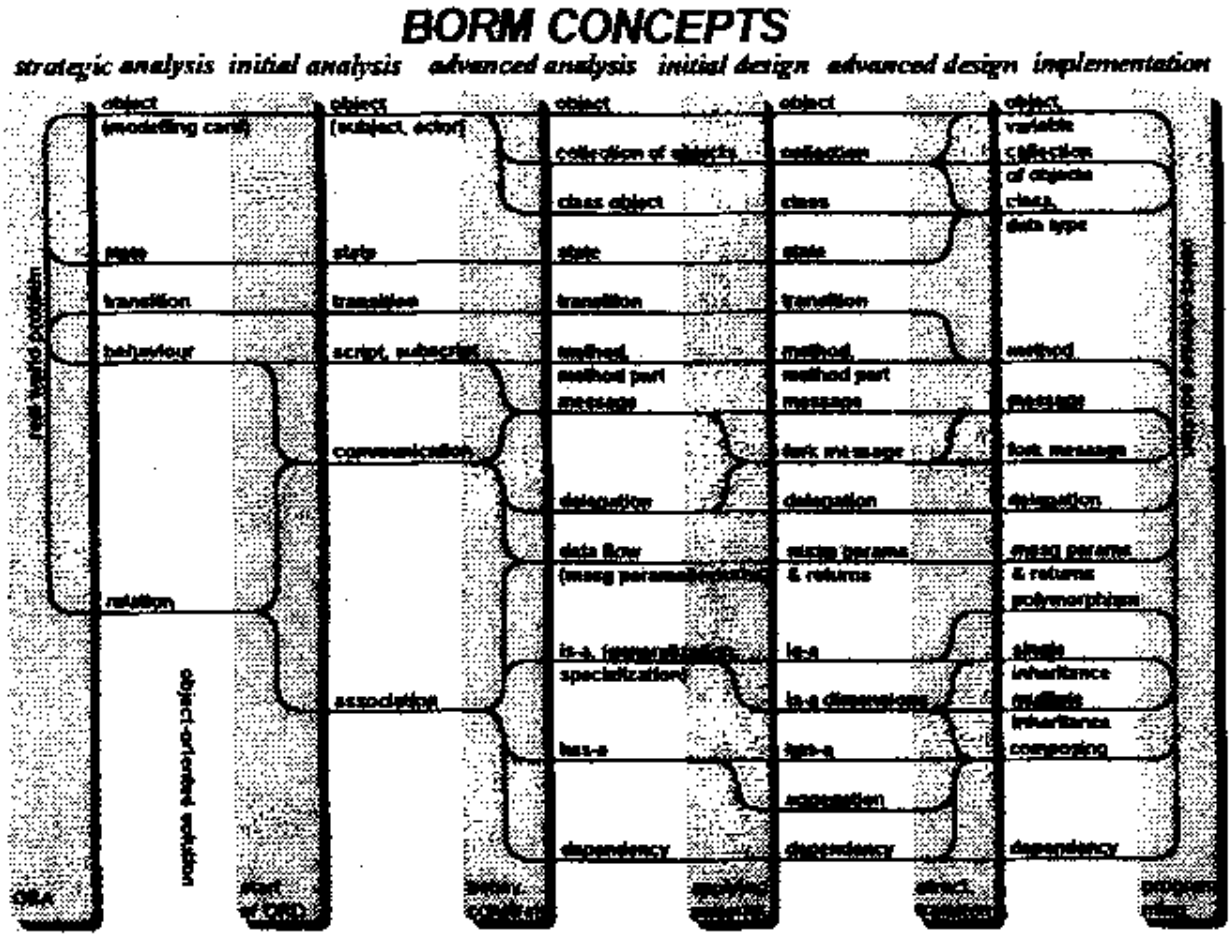
U každé z obou zainteresovaných skupin při vývoji IS lze rozlišit ještě tři různé úrovně chápání zadání a realizace softwarové aplikace [Martin].

- **Aplikační úroveň** představuje okruh znalostí a dovedností, se kterými přichází jedna nebo druhá skupina do běžného pracovního styku. Pojmy z aplikační oblasti jsou tedy pro příslušníka skupiny známé a srozumitelné. Bohužel jsou však nejvíce vzdálené pojmům aplikační úrovně skupiny druhé. (*Pokud se tvorba IS odehrává jen na aplikační úrovni, tak se hovoří o naivní technologii tvorby software*)
- **Základní úroveň** je určité komunikační optimum mezi velmi od sebe vzdálenými aplikačními úrovněmi. Největší roli zde hraje konceptuální modelování, které dovoluje dostatečně srozumitelné a formální diagramové nástroje, které jsou ještě sledovatelné „neprogramátory" z první skupiny a zároveň poskytují dostatek informací pro příslušníky druhé skupiny.
- **Pokročilá úroveň** je tvořena sadou dalších nástrojů a technik, které v první skupině dovolují najít správný konceptuální model a druhé skupině tento model umožňují transformovat do softwarové podoby.

Z výše uvedených informací vyplývá, že **není možné pracovat ve všech etapách tvorby IS se stejným pojmem objektu**. Lze očekávat, že jednotlivé atributy a vazby mezi objekty se budou v průběhu vývoje IS měnit, a že každý následující pojem bude mít zřejmě svého abstraktnějšího předchůdce, ze kterého byl odvozen. Tyto tzv. transformace jednotlivých pojmů mezi sebou jsou obsahem jednotlivých technik v různých fázích tvorby informačního systému. Každý pojem používaný v OOA a OOD tedy může mít

1. **okruh nadřazených pojmů**, ze kterých může být na základě nějaké techniky odvozen,

2. **okruh podřízených pojmů**, které z něj mohou být pomocí nějaké techniky odvozeny,
3. **okruh platnosti**, neboť v jiných fázích vývoje IS, než které mu přísluší, je místo pro jemu nadřazené nebo podřízené pojmy a
4. **sadu technik či pravidel**, pomocí kterých je transformován na z něj odvozené pojmy.



Protože se v různých fázích vývoje IS obsah pojmu objektu mění, tak je výhodné objekty v průběhu projektování rozdělit na

1. **Softwarové objekty**, se kterými se pracuje v závěrečných fázích vývoje IS. Tyto objekty obsahují pojmy přímo odpovídající konstrukcím z objektových programovacích jazyků.
2. **Konceptuální objekty**, se kterými se pracuje v prostředních fázích vývoje IS. Tyto objekty obsahují základní pojmy objektově orientovaného paradigmatu, jako například polymorfismus objektů, zapouzdření, skládání, delegování, klasifikace objektů podle různých dimenzí, závislost objektů, třídy a množiny objektů atd. Je pravda, že některé z konceptuálních pojmů jsou shodné se softwarovými pojmy, ale značnou část je třeba při přechodu na softwarové objekty transformovat, protože současné používané programovací jazyky (*např. C++*) podporují pojmy OOP pouze omezeným způsobem. V rozsahu a náročnosti transformace mezi konceptuálním a softwarovým modelem také spočívá jedna z možností měření kvality použitých programovacích nástrojů (*např. u jazyka Smalltalk je tento přechod snadnější než u jazyka C++*) [Goldberg].
3. **Esenciální objekty**, pro které se v zahraničí prosazuje také označení „**Business objects**". Tyto objekty vyplňují mezeru mezi zadáním - tj. chápáním na aplikační úrovni zadavatele a mezi konceptuálním objektovým modelem.

Proč „Business" objekty

Používání business objektů ve vývoji IS je poměrně novou záležitostí [Ould] a většina současných CASE nástrojů je zatím nepodporuje a přímo začíná vývoj IS od budování objektově-třídního diagramu - tedy z našeho pohledu jakoby od prostředka. V poslední době se objevila určitá doplnění CASE nástrojů [UML] ve směru k esenciálním objektům nejčastěji v podobě přidání tzv. use-case analýzy [Jacobson] ke stávajícím metodám OOA a OOD. Do okruhu práce s business objekty také patří námi používaná metoda **Object-Behavioral Analysis** [OBA], která pomocí práce s modelujícími kartami a s různými tabulkami dovoluje **získávat a ověřovat** esenciální objektový model podle zadání. OBA byla původně navržena jako metodologický nástroj pro porozumění zadání z interview a slovní specifikace a nalezení požadovaných vlastností budovaného systému.

Konceptuální objektový model je totiž nevhodné stavět na „zelené louce", neboť obsahuje již dost složité pojmy (*v OMT nebo UML to jsou např. kvalifikátor, vazební třída, ternární relace, agregace apod.*), **které jsou podřízeny budoucí softwarové implementaci**.

Důležitost objektového esenciálního modelování tedy spočívá v nevhodnosti objektových konceptuálních nástrojů pro zachycení prvotní informace. Porovnáme-li si např. „object-class diagram" z OMT nebo UML a klasický ERD ze strukturovaných metod, tak zjistíme, že ERD používá tři základní pojmy, ale objektový diagram okolo dvaceti. Mezi konceptuálními objektovými nástroji neexistuje žádný jednoduchý nástroj, který by byl snadno srozumitelný pro „neprogramátory", protože většina konceptuálních objektových pojmů souvisí nějakým způsobem s počítačovou implementací.

Jaké by tedy měl mít esenciální objekt (*a tedy i esenciální diagram*) vlastnosti?

- měl by rozlišovat základní a odvozené vlastnosti OOP, což znamená, že by měl podporovat pouze základní pojmy OOP a ty odvozené by měl ponechat na pozdějších transformacích do konceptuálního modelu,
- neměl by zahlcovat uživatele množstvím složitých grafických symbolů a s nimi souvisejících pojmů na různé úrovni abstrakce a
- měl by být použitelný nejen pro zahájení tvorby softwarové aplikace, ale i pro podporu práce např. manažerů pro tvorbu modelů při rozhodování, aniž by model vždy nutně končil softwarovou implementací.

Námi používaný esenciální (business) model, který byl vyvinut v rámci projektu VAPPIENS, má následující vlastnosti:

- Používá pouze pojmy **objekt, chování objektu, asociace (datové vazby) mezi objekty a komunikace (zprávy) mezi objekty**.
- Na každý objekt je nahlíženo jako na **automat se stavy a přechody**, přičemž stavy a přechody jednoho objektu jsou závislé na chování druhých objektů.
- **polymorfismus objektů** lze rozdělit na
  1. různé chování různých objektů na stejnou zprávu (*klasický polymorfismus*),
  2. různé chování jednoho objektu v různých stavech na stejnou zprávu (*tzv. vnitřní polymorfismus*) a
  3. různé chování jednoho objektu v jednou stavu na stejnou zprávu od různých objektů (*tzv. subjektivita chování objektů*).

V příloze tohoto článku je ukázka analýzy marketingového informačního systému softwarové firmy na úrovni business objektů. První stránka obsahuje příklad modelujících karet a jedné z tabulek vztahů používaných v OBA. Na dalších stránce je celkový model nejprve z pohledu asociací mezi objekty a poté z pohledu chování objektů. Poslední stránka ukazuje stavy a přechody dvou business objektů, přičemž diagram ukazuje, jak jsou jednotlivé přechody a stavy těchto objektů ovlivňovány chováním okolních objektů. Tyto okolní objekty samozřejmě mohou mít také svoje vlastní stavy a přechody, takže je možné a z naší zkušenosti někdy i žádoucí na celý esenciální model nahlížet jako na systém vzájemně se ovlivňujících objektů s na sobě závislými stavy a přechody.

Pokud se podaří systém vzájemně komunikujících esenciálních objektů najít, tak máme dostatečnou informaci pro přechod do konceptuálního modelu, kde se doposud namodelovaná informace výhodně zhodnotí například v návrhu např. struktury tříd nebo jednotlivých metod, zpráv a parametrů.

Závěr

S nástupem stále dokonalejších nástrojů vizuálního programování a podpory rychlého vývoje aplikací se může zdát, že celý proces tvorby software se díky používání objektově orientovaného přístupu usnadňuje. Toto tvrzení by však platilo pouze v případě, že se nemění složitost zadání a požadavky uživatelů na software. Protože tento předpoklad neplatí, tak je třeba ze strany tvůrců software neustále **posilovat úvodní fáze vývoje**.
Ještě v poměrně nedávné době platilo konceptuální modelování za optimální střed pomyslné osy od zadání k implementaci. Dnes se však vlivem vývojových změn poloha konceptuálního modelování posouvá směrem k jejímu implementačnímu konci a na jeho místo se postupně dostávají nově vytvářené techniky a nástroje modelování „business" objektů.

Literatura

[1]Rumbaugh James, Blaha Michael, Premerlani William, Eddy Frederic, Lorensen William, Object-Oriented Modelling and Design, Prentice Hall 1991, ISBN 0-13-630054-5

[2]Goldberg Adele, Kenneth Rubin S, Succeeding with Objects - Decision Frameworks for Project Management, Addison Wesley 1995, ISBN 0-201-62878-3

[3]Rubin K.S., Goldberg A. Object Behavioural Analysis, pp 48-62 Communications of the ACM 35(9) 1992

[4]Meyer Bertrand: Towards an Object-Oriented Curriculum, Object Currents, SIGS Publications, Inc., New York, NY, USA
http://www.sigs.com/publications/docs/oc/9602/oc9602.c.meyer.html

[5]Knott R P, Polák J, Merunka V, Business Object Relation Modelling Methodology, závěrečná zpráva k řešení projektu VAPPIENS, British Council - Know-How Fund, Dept. of Computer Studies, Loughborough University.

[6]Unified Modelling Language, Rational Inc. http:/www.rational.com

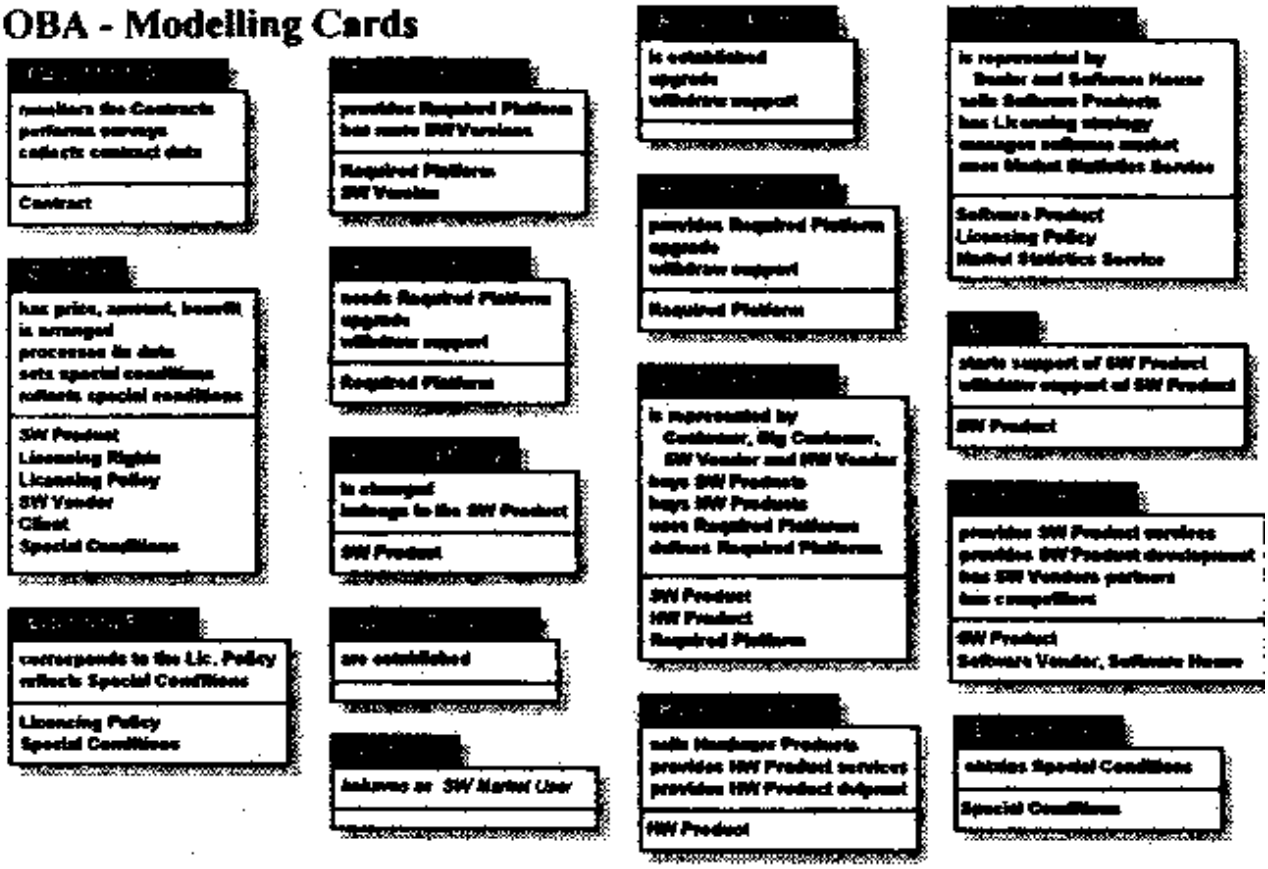
[7]Martin James, Odell James J, Object-Oriented Methods - A Foundation, Prentice Hall 1995, ISBN 0-13-63856-2

[8]Ould Martyn A, Business Processes - Modelling and Analysis for Reengineering and Improvement, John Willey 1995 ISBN 0-471-95352-0

[9]Jacobson Ivar, Object-Oriented Software Engineering - A Use Case Driven Approach, Addison Wesley 1992, ISBN 0-201-54435-0

Příloha - esenciální (business) objekty v analýze informačního systému pro softwarovou firmu

OBA - Modelling Cards

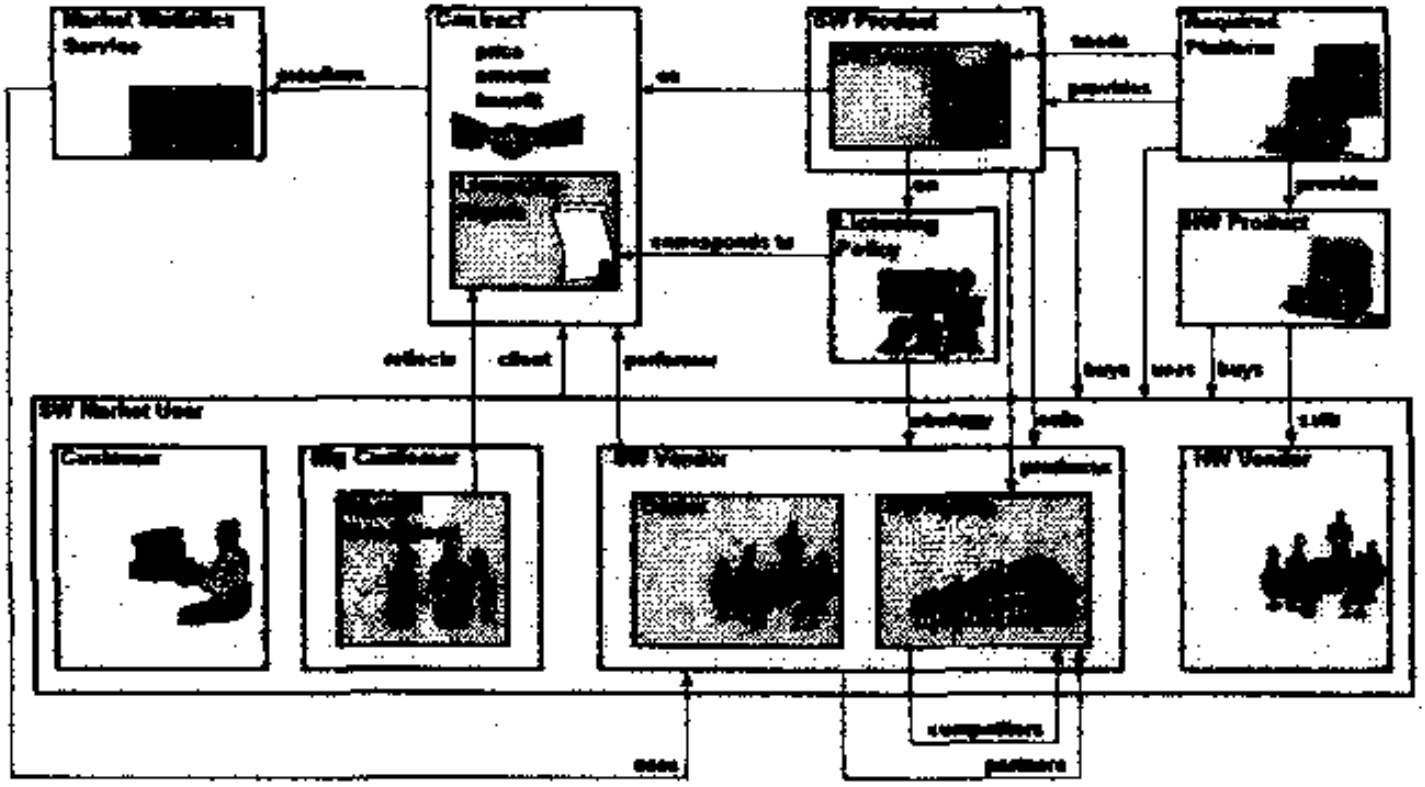


OBA - Object Relationships

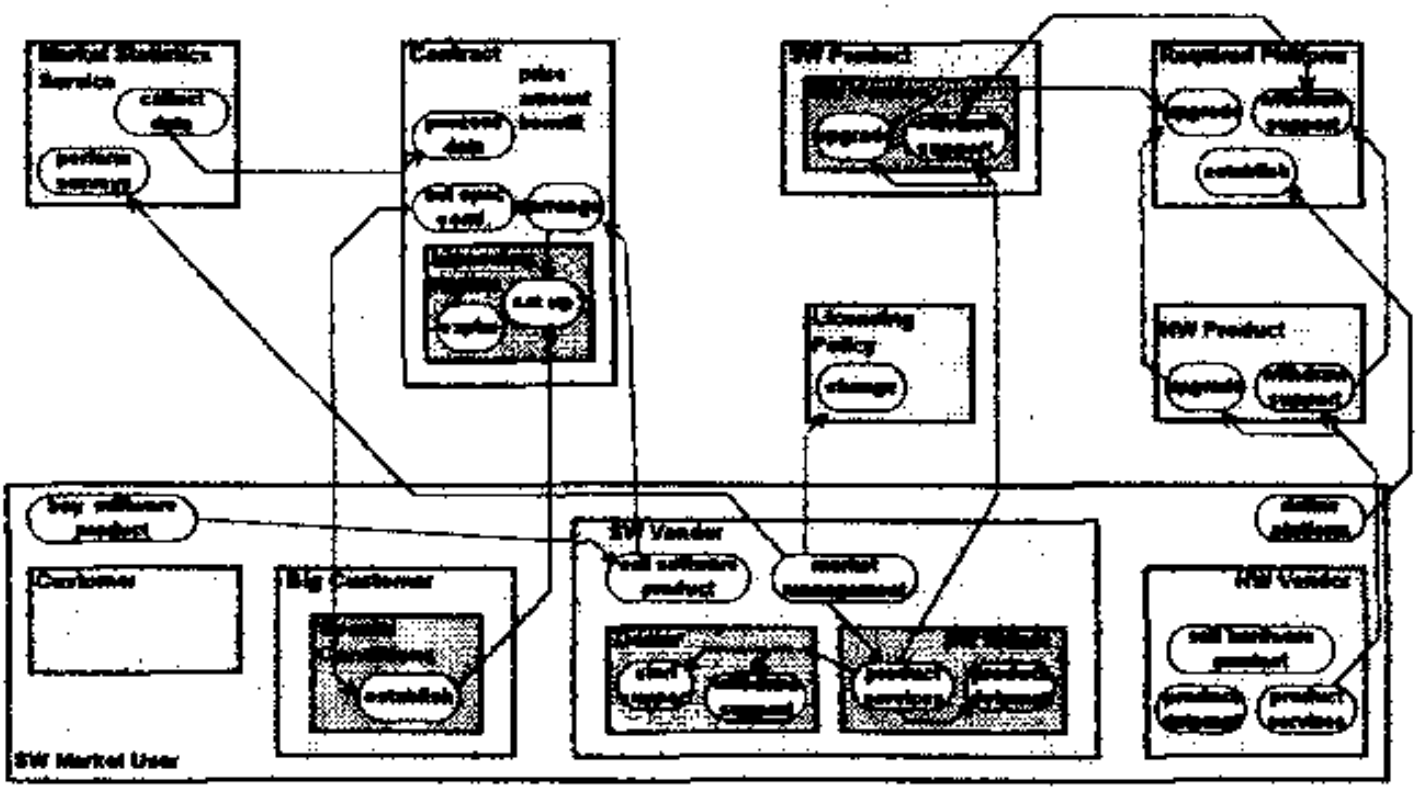
Market Statistics Service	monitors get data from	Contract Contract
Contract	is made on has associated sets up is performed by has client as sets special cond.	Software Product Licensing Rights Licensing Rights Software Vendor SW Market User Special Conditions
Licencing Rights	reflect correspond to	Special Conditions Licencing Policy
Licensing Policy	is made on	Software Product
Software Product	has creates	Software Version Required Platform
Software Version	needs upgrades withdraws support of	Required Platform Required Platform Required Platform
Special Conditions	set up	Licencing Rights
Customer	behaves as	SW Market User
Big Customer	has associated	Special Conditions
Required Platform	is created by is created by	Software Version Hardware Product

Hardware Product	creates upgrades withdraws support of	Required Platform Required Platform Required Platform
SW Market User	is represented by is represented by is represented by is represented by buys buys from buys uses defines	Customer Big Customer Software Vendor Hardware Vendor Software Product Software Vendor Hardware Product Required Platform Required Platform
Software Vendor	is represented by is represented by has strategy of sells performs changes uses	Dealer Software House Licensing Policy Software Product Contract Licencing Policy Market Stat. Service
Dealer	behaves as	Software Vendor
Software House	produces upgrades withdraws support of has partners of has competitors of	Software Product Software Product Software Product Software Vendor Software House
Hardware Vendor	sells upgrades withdraws support of	Hardware Product Hardware Product Hardware Product

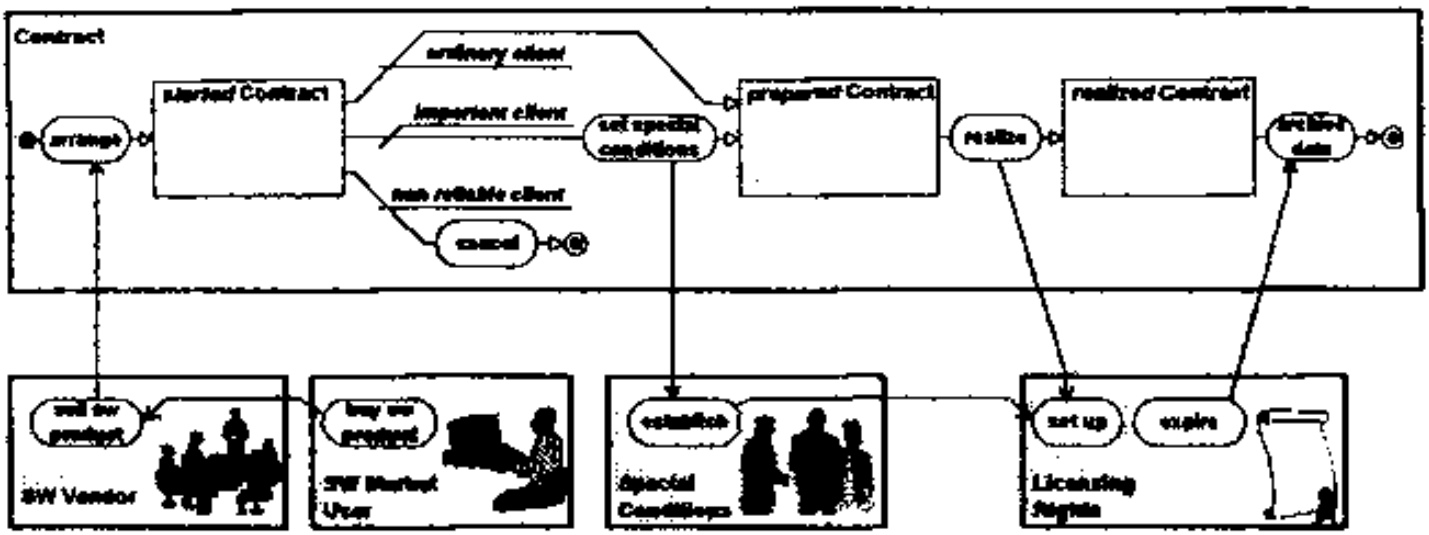
ORD - Initial Analysis - subjects, associations



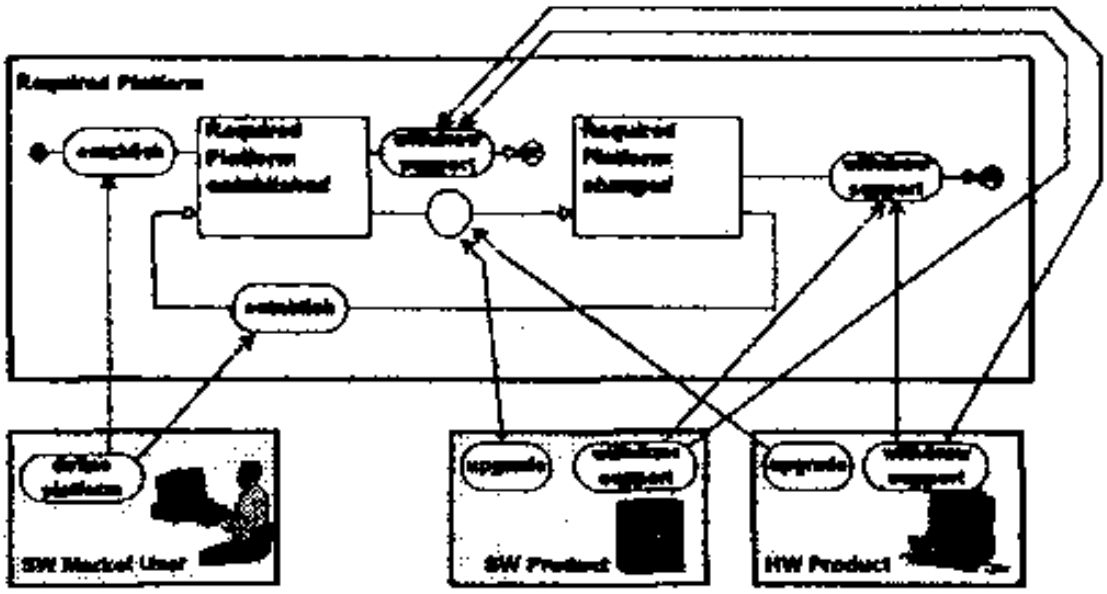
ORD - Initial Analysis - subjects, behaviours and communications



ORD - Initial Analysis, Contract Life-Cycle



ORD - Initial Analysis, Required Platform Life-Cycle



Realizace víceuživatelského systému pomocí OOMT

Pavel Drbal, Zdenek Kadlec

VŠE Praha, W. Churchilla 4, 130 67 Praha, Česká Republika

Úvod

V současné době jsou módními termíny pojmy objekt, objektový přístup, objektové metodiky. Mnozí nadšenci v tom vidí spásu, jiní mnoho povyku pro věci dávno známé. Chtěl jsem si prakticky vyzkoušet, nakolik jsou či nejsou objektové postupy užitečné. Měl jsem zkušenosti s vývojem menších programů za pomoci strukturovaných postupů, ale nebyly to projekty ani velké, ani složité. Než jsem stačil nějaký větší projekt vytvořit, uchvátily mne právě objektové techniky.

Předkládaný projekt (dále EDIK – EviDence studentů včetně Identifikačních Karet) je v současné době (březen 97) v provozu na Vysoké škole ekonomické, v oddělení provozu počítačových učeben a studoven. Systém řeší kontrolu oprávněnosti přístupu studentů do počítačové sítě VŠE. Systém pracuje v reálném čase a má především restrikční funkci (pokud někdo nesplňuje zadané podmínky, není mu umožněna práce v počítačové síti VŠE).

Tímto článkem bych chtěl seznámit čtenáře s mým pohledem na objektové techniky, především na OOMT, s nímž jsem se v rámci grantu (pod vedením pana Pavla Drbala) setkal. Uvádím zde základní filosofii systému EDIK a jeho architekturu. Cílem je ukázat rozsah navrhovaného systému, a umožnit tak čtenáři seznámit se s prostředím, z něhož mé názory na OOMT vychází.

Popis původního stavu

Provoz učeben (k 1.3. 1995)

Počítačové učebny jsou umístěny ve 3 podlažích jedné budovy. V přízemí jsou studovny, do nichž mají studenti přístup celý den (7:00 - 20:45) a mohou zde individuálně pracovat na cca 90 počítačích. V prvním a druhém patře jsou počítače rozmístěny vždy v 5 učebnách po 18 počítačích (tj. 90 počítačů v každém patře). V těchto učebnách probíhá výuka. V každém patře je vstup do prostoru počítačových učeben či studoven možný jen jedním vchodem, u kterého sedí pracovník obsluhy.

Do studoven mají přístup studenti, vyučující a ostatní zaměstnanci školy. Do učeben (tj. do 1. a 2. patra) mají povolen vstup pouze ti studenti a vyučující, kteří zde mají výuku. V případě, že některá učebna není využita pro výuku, mohou ji využít i ostatní studenti pro svou individuální práci. Obsluha (služba) u dveří musí zajistit uvolnění učebny před začátkem další výuky.

Při vstupu do studoven (resp. učeben) musí každý příchozí odevzdat svou identifikační kartu (plastiková karta s fotografií a čárovým kódem) pracovníkovi obsluhy. Při odchodu si kartu u obsluhy vyzvedne. Od povinnosti odevzdávat identifikační kartu jsou osvobozeni studenti, kteří jdou na výuku.