Univerzita Karlova v Praze
Filozofická fakulta
Ústav informačních studií a knihovnictví

Dizertační práce

PhDr. Ivan Bartoš

Metodologie a problémy při transformaci dat a určení jejich významu v rámci integrace heterogenních informačních zdrojů

Praha, 2012

školitel: PhDr. Richard Papík, Ph.D

Prohlašuji, že jsem dizertační práci vypracoval samostatně, že jsem řádně citoval všechny použité prameny a literaturu a že práce nebyla využita v rámci jiného vysokoškolského studia či k získání jiného nebo stejného titulu.

V Praze dne 15. 3. 2012 ……..…..……...

PhDr. Ivan Bartoš

Identifikační záznam

BARTOŠ, Ivan, PhDr. Metodologie a problémy při transformaci dat a určení jejich významu v rámci integrace heterogenních informačních zdrojů. Praha, 2012. 109 s. Dizertační práce. Univerzita Karlova v Praze, Filozofická fakulta, Ústav informačních studií a knihovnictví.

BARTOŠ, Ivan, PhDr. Methodology and issues of data transformation and its information value estimation during the integration of the heterogenous information sources. Prague, 2012. 109 p. Dissertation. Charles University in Prague, Faculty of Arts, Institute of Information Studies and Librarianship.

Abstrakt

Tato práce řeší problematiku transformace dat a s nimi souvisejících informací, která je aktuálním problémem v řadě vědních, ale i komerčních oblastí. Výpovědní hodnota informace, její kvalita a kvalita dat, ze kterých vychází, se v různých systémech liší. Toto se děje nejen z důvodu odlišné typologie určitého zdroje informací, ale často i díky samotnému způsobu chápání či zachycení informace o popisované entitě skutečného světa. Informační systémy, v případě této práce jsou to konkrétně databázové systémy, mohou bezchybně fungovat jako samostatné celky. Problém nastává až v momentě potřeby integrace dvou takových heterogenních systémů a následné migraci informací mezi nimi. Na základě této potřeby lze práci rozdělit do čtyř hlavních částí.

V první části je popsán způsob, jakým je možné klasifikovat kvalitu dat ve zdroji určeném k integraci, ze kterých lze informace získávat. Vzhledem k obecně známému problému nedostatečné projektové a systémové dokumentace (STOLOVITSKY, 2010) jsou zde popsány takové metody, které lze využít i za předpokladu minimální spolupráce s tvůrcem či správcem zdrojového systému. Prvním krokem je získání čistě statistických hodnot o jednotlivých popisovaných entitách, které jsou samotným systémem automaticky uchovávány (velikost popisované entity, množství jejích vlastností-atributů, četnost (kardinalita) entity atp.). Další metody pak zkoumají samotnou kvalitu dat, jejich význam v kontextu dalších informací, které systém reprezentuje, jejich stabilitu v závislosti na čase a v neposlední řadě konzistenci v reprezentaci popisované skutečnosti. Tyto metody se souhrnně dají označit jako datová profilace. Popsána je i metodika dokumentace takových měření a rozšířená validace za využití externích pravidel (pro ilustraci např. regulárních výrazů).

Druhá část práce se zabývá výběrem a přenosem již klasifikované informace ze zdrojového systému do systému cílového, tedy převodem dat na informaci a jejím následném uložení opět ve formě dat. V tuto chvíli je již známé spojení mezi atributy zdrojového systému a odpovídajícími atributy v systému cílovém. Jednotlivé popisované metodologie lze označit zjednodušeně jako extrakci (Extraction), čištění (Cleansing),

opravu (Correction), konverzi (Conversion) a transformaci (Transformation). Techniky využité v rámci těchto metodologií jsou ilustrovány názornými příklady.

Třetí část, která předchází vzniku modelů a tvorbě mapování mezi modely různých systémů, se zabývá analýzou entit skutečného světa, jejich vztahů a jejich omezení. Výsledkem této analýzy je buď ontologie, nebo model entit v úrovni metadat spolu s definicí jednotlivých atributů těchto entit, typů atributů a opět omezujících pravidel. Tento tzv. konceptuální model je základem pro vznik datového slovníku. Konkretizace tohoto modelu, a tedy i slovníku, do logické a následně fyzické roviny se již zcela odvíjí od platformy, na které se nachází konkrétní instance resp. od prostředí, ve kterém jsou data popisující skutečnost uchovávána.

Ve čtvrté části je blíže popsán způsob, kterým je možné sémanticky mapovat entity jednotlivých zdrojů. Výsledkem této činnosti je vznik terciálních informací o samotném zdroji, entitách a pravidlech jejich integrace, které lze také označit za jakýsi metadatový model metadat. V této části jsou popsány a ilustrovány základní metody a způsoby takovéhoto mapování v jeho člověkem čitelné reprezentaci. Vzhledem k vysoké ceně již existujících softwarových nástrojů, kterými lze tento problém řešit, je zde kladen větší důraz na teoretické a metodologické prvky, raději než na popis práce s určitým komerčním softwarem.

Tato práce si klade za cíl nabídnout v praxi využitelné metody integrace různých zdrojů dat a nastínit možná úskalí, kterých by se měl tým zodpovědný za realizaci takovéto integrace vyvarovat.

V závěru práce je uveden seznam excerpovaných pramenů, původních dokumentů a dalších relevantních zdrojů.

Abstract

This study focuses mainly on the data and information transformation issue. This topic is currently critical in several scientific and commercial areas. Information value, information quality and the quality of the source data differs between the various systems. This is not only due to the different topologies of the information sources but also because of its different understanding and a manner of storing the information describing the entity of the enterprise. Such information systems, respectively database systems in the scope of the thesis, could perform well as the stand alone systems. The issue appears in the moment when such heterogeneous systems are required to be integrated and the information shall be migrated between each other. The thesis is logically divided into four major parts based on these issues.

The first part describes the methods that can be used to classify the data quality of the source system (the one to be integrated) from which the information can be extracted. Based on assumption of the common lack of project and system documentation (STOLOVITSKY, 2010) hereby introduced methods can be used for such qualification even when the creator or administrator of the source system is not willing to cooperate. The first step is usually to obtain clear statistic values about the described entities that are automatically gathered and stored by the system itself (size of described entity, amount of its features-attributes, cardinality of the entity etc.) Each method then exercises the very quality of the data, its importance among the other information that system represents, its stability related to the time and the consistency in representation of the described state of the enterprise. The methodology of the documentation of such measuring will be described as well together with the extended validation of the data usign external rules (ilustrated using Regular Expressisons).

The second part of the study mainly focuses on the selection and transformation of already classified information from the source to the target system. It is basically the concept of the ‘data - information – data’ transformation, extraction, transformation and final storage into the target. The connection between the source system and the target

system attributes is already known. Individual methodologies can be described as Extraction, Cleansing, Correction, Conversion and Transformation. Each step is illustrated by practical examples.

The third part which precedes the creation of the models and further mapping between models of different systems describes how the real world entities as well as its relations and constraints are analyzed. The result of such analysis is either an ontology or an entity model on the metadata level with its attributes types and constraints rules. This conceptual model is an input to the creation of the Data Dictionary. Its concretization to the logical and physical layer is fully dependent on the platform of the database instance – on the environment where the data about the real world is stored.

The fourth part describes in detail the way how the entities from different sources can be mapped on the semantic level. The result of such activity is a creation of the tertiary information about the sources, its entities and rules of the integration. The product of this activity is sometimes referred as the metametadata model. This study will explain the basic concepts of such mapping in human readable form of its representation. Since the existing commercial software tools are easy to use but expensive to buy, the more importance will be given to the theoretical and methodological aspects rather than to the description of usage of the particular proprietary software.

The main goal of this study is to provide methods for the heterogeneous data sources integration that can be easily put into the practice and meanwhile to point out some issues and risks that the team realizing such type of project should be aware of.

There is a list of excerpted materials, original documents and relevant sources at the very end of the paper.

Klíčová slova:

Profilace dat, transformace dat, transformace informací, extrakce dat, extrakce informací, čištění dat, oprava dat, konverze dat, systémová integrace, informační hodnota, ontologie, sémantické mapování, sémantický metadatový model, derivace podobných objektů a schémat, UML, model entit, regulární výrazy

Key Words:

Data Profiling, Data Transforamtion, Information Transformation, Information Extraction, Data Cleansing, Information Cleansing, Data Correction, Data Conversion, System Integration, Information Value, Information Weight, Ontology, Semantic Mapping, Semantic Metadata Model, Deriving Similarities Of Objects And Subschemes, UML, Entity Model, Regular Expressions

Obsah

Úvod

Není potřeba zdůrazňovat roli a důležitost informace ve společnosti. Pokud se oprostíme od pojetí informace jako základní lidské potřeby, která umožňuje jedinci svobodně se rozhodovat a pomineme-li touhu jedince po vědění, kdy člověk informace přímá a transformuje je ve znalosti, je dnes informace chápána často jako obchodovatelná komodita. Ať se jedná o průmyslové informace, které ovlivňují samotné výrobní procesy, informace o trzích, které v rámci competitive inteligence umožňují konkurenční boj mezi firmami, anebo takové informace, které jsou po důkladné analýze a vložení do obchodního modelu používány marketingovými týmy při rozhodování o další strategii firmy (analýzy odbytu, chování zákazníka/uživatele, atp.) (PAPÍK, 2001).

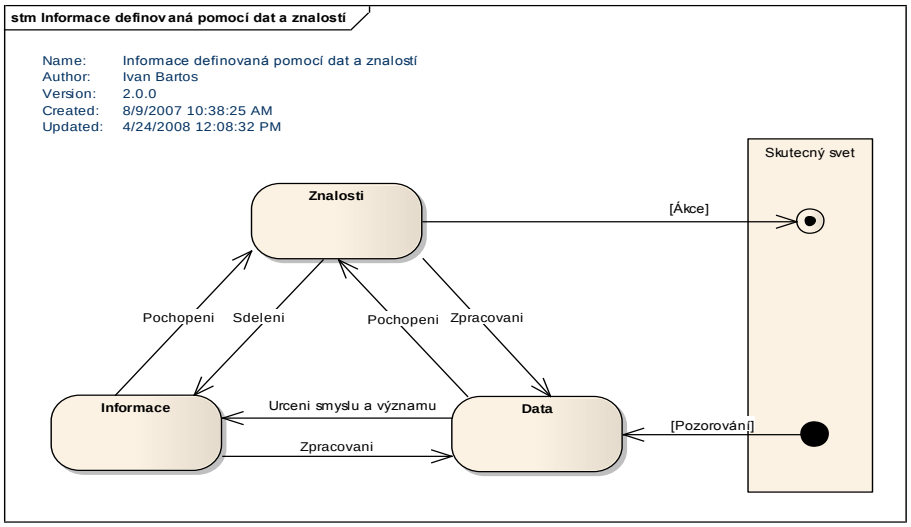
I když lze postupy popisované v této práci určitým způsobem aplikovat na prakticky jakýkoliv typ takové "komerční" informace, uvedené příklady metodologií se budou týkat převážně skupiny třetí, tj. dat využívaných oddělením marketingu jako zdroj pro **data-mining**[1] a **BI**[2] systémy resp. informace, jejichž způsob uložení ve formě dat v databázích a jejichž následné pochopení (transformace na znalost) generuje v konečné fázi ekonomicky měřitelný zisk.

V práci budou často používány termíny data, informace a znalosti. Filozofické rozbory těchto pojmů nechme však stranou. Ačkoliv může toto označení v běžném hovoru, v akademické diskuzi, nebo z pohledu informatiky splývat, nebo být značně diskutabilní, je toto rozdělení a přechod mezi jednotlivými rovinami důležité, převážně z důvodů jednotlivých úrovní abstrakce, která je pro transformace a jejich modelování důležitá.

- Data jsou v práci referována jako záznam v nějaké struktuře, například číslo 35 v určitém sloupci v tabulce.

[1] **Data Mining** - analytická metodologie získávání netriviálních skrytých a potenciálně užitečných informací z dat.
[2] **Business Inteligence** - proces, jak získat neznámé a hodnotné informace z konsolidovaných databází za účelem jejich využití pro následující strategická a taktická rozhodnutí.

- Informace je v kontextu správnosti záznamu a dalších souvislostí chápána následujícím způsobem: číslo 35 je věk uživatele, muže, registrovaného do systému v určitý den, měsíc, rok.
- Znalostí pak budou označeny například jeho preference, které lze použít pro cílené oslovení v kampani, nebo model jeho chování či statistiky, které lze využít v predikování vývoje celého systému (znalost je zde chápána jako entita komplexního charakteru a je přímo aplikovatelná v reálném světě). (SKLENÁK, 2001)



Obrázek 1: Teorie informace : informace – data – znalosti

Každý subjekt shromažďující a skladující informace používá určité (často vlastní) metodologie zpracování znalostí, způsob jejich uložení v relačních nebo jiných strukturách a způsob interpretace uložených informací pro své potřeby nebo pro potřeby jejich obchodování. Tyto metodologie, pokud v daném případě existují, mohou být více či méně korektní, konzistentní a zdokumentované. Míra kvality dat se může negativně odrážet na funkci samotného "mateřského" subjektu a tím na ziscích, které je schopen generovat (Aberdeen Group, 2007). Výpovědní hodnota informace, její kvalita a kvalita dat, ze kterých vychází, se v různých systémech liší. Toto se děje nejen z důvodu odlišné typologie určitého zdroje, ale často i díky samotném způsobu chápání či zachycení informace o popisované entitě skutečného světa. Podobné systémy, v dizertační práci konkrétně databázové systémy, mohou bezchybně fungovat jako samostatné celky (systém je nějak nastaven a případné chyby, pokud je jejich výskyt zdokumentován, se stávají standardem

systému). Zásadní problém nastává až v momentě potřeby integrace dvou takových heterogenních systémů a následné migraci informací mezi nimi.

Je třeba si uvědomit, že se nejedná o nějaké výjimečné případy. Hýbe-li světem fenomén nazývaný globalizace, pak je integrace informačních systémů ve světě informačních technologií tím samým fenoménem. Oba tyto fenomény jsou na sobě přímo závislé, navzájem se evokují a naopak, každý z nich řeší důsledky projevů toho druhého (HAY, ROSAMOND, 2002). Proto je popisovaná problematika nejen mimořádně aktuální, ale lze přepokládat, že poptávka po kvalitním zvládnutí metodologie integrace a transformace a jejím praktickém využití bude v budoucnu dále narůstat. Na trhu IT služeb existují již delší dobu subjekty, které se zabývají téměř výhradně teoretickou i technologickou realizací systémové integrace (např. Wipro, Informatica, a další).

Následující kapitoly budou popisovat právě problematiku integrace systému, jehož navržení, správa, údržba a dokumentace zaostává, jehož kvalita dat a možnost následné interpretace informací je problematická, a jehož tvůrci či administrátoři jsou vzhledem k častému fenoménu nucené integrace (jež je důsledkem například akvizice firmy) značně laxní v jakékoliv spolupráci. Podobná situace, jež nemusí být samozřejmě situací nejčastější, je ideální výzvou a zároveň příkladem, na kterém lze ilustrovat jednotlivé kroky, které jsou potřebné k dosažení integrace dat zdrojového systému do cílového systému takovým způsobem, aby cílová homogenita a kvalita zůstala zachována. Dizertační práce si klade za cíl nabídnout v praxi využitelné konkrétní metody integrace různých informačních zdrojů a nastínit možná úskalí, kterých by se měl tým nebo jednotlivec zodpovědný za realizaci této integrace vyvarovat. Jednotlivé kapitoly popisují kroky, které je nezbytné pro splnění zadaného úkolu učinit. Kromě analýzy systému, jeho struktury a dat v něm uložených, se zaměříme také na metody modelování, tvorbu ontologií a mapování vzniklých modelů za účelem maximálně standardizované a v ideálním případě automatizované integrace systémů a transformace dat resp. informací v nich obsažených.

I přesto, že se práce zabývá příklady z komerčního prostředí, jsou popisované metodologie univerzální i pro aplikace v akademickém či státním sektoru. Entity skutečného světa figurují ve všech prostředích podobným způsobem.

Autor práce se problematikou integrace a transformace dat zabýval v podstatě celý svůj dosavadní akademický i profesní život. První setkání s problematikou integrace dat proběhlo v rámci aktivity spojené se sdílenou katalogizací při spolupráci na tvorbě a udržování souboru národních autorit pod patronátem Národní knihovny v Praze. Zde se skutečně většina finální deduplikace, standardizace a unifikace dat (v tomto případě záznamů o autoritách), která vznikají se značným podílem lidského faktoru v průběhu katalogizace nových zdrojů v jednotlivých knihovnách, zpracovávala manuálními prolinky mezi jednotlivými záznamy strukturovanými ve formátu UNIMARC/Autority. V rámci integrace heterogenních zdrojů, jimiž i co do formátu dat částečně standardizované aplikace bezpochyby jsou, pomáhal zavádět nyní již prakticky historický protokol Z39.50 do knihoven v České republice. Zde působil převážně v oblasti standardizace a analýzy praktické aplikovatelnosti ve stávajících prostředích. V tomto období se společně s ing. Šmilauerem ze Státní technické knihovny v Praze podílel na výkladu průvodní normy protokolu (BARTOŠ, ŠMILAUER, 2002) a následně publikoval i studii možností aplikace protokolu Z39.50 v knihovnickém prostředí včetně predikce budoucího vývoje za využití otevřených standardů (BARTOŠ, 2003). V rámci svého doktorandského studia vedl semináře o databázových systémech a tvorbě datových modelů. Této problematice se věnoval i v rámci půlročního studijního pobytu na Computer Science faculty - University of New Orleans ve státě Louisiana, USA. V praxi působil v několika firmách, jejichž obchodní modely a aktivity souvisely primárně právě s kvalitou dat a možnostmi jejich následné interpretace, a to jak na úrovni programátorské, kde se zabýval konkrétní realizací transformačních procesů převážně v souvislosti s datovými sklady, tak na úrovni návrhů komplexních řešení, ve kterých je zvažována integrace až desítek systémů, aplikací či jejich jednotlivých modulů. V současnosti pracuje jako funkční architekt v oblasti telekomunikací.

Vzorový projekt

Marketingové oddělení firmy Alfa co., která podniká v oblasti skladování a distribuce publikací nakladatelství CheapBooks, shromažďuje informace o registrovaných uživatelích systému (registrovaný zákazník má zřízený účet na webových stránkách společnosti). Uživatel o sobě v rámci registrace zadává informace skrze .NET formulář v rozhraní WWW. Tento formulář je navržen způsobem, který se snaží v maximální míře eliminovat zadávání nesmyslných údajů. Výsledná data uložená v **OLTP**[3] databázi jsou v nočních hodinách přesouvána pomocí **ETL**[4] procesů do jednoduché **OLAP**[5] struktury Alfa co. **datamartu**[6]. Na jeho základě vytváří marketingové oddělení emailové kampaně s aktuální nabídkou titulů, které periodicky rozesílá specifickým cílovým skupinám uživatelů např. pouze uživatelům s vysokoškolským vzděláním žijícím na území USA atp.

Firma Alfa co. provedla akvizici společnosti Beta&sons, která působí na poli distribuce periodik, a jejíž zákazníci (uživatelé) souhlasili s poskytnutím svých informací pro marketingové účely. Firma Beta&sons bude stále provozovat svoji činnost pod zavedenou značkou, pouze její marketing bude nyní spravován novou "mateřskou" firmou. Stávající zákazníci budou taktéž začleněni do cílové skupiny oslovovaných uživatelů. Firma Alfa co. bude těmto "novým" uživatelům nadále zasílat informace z oblasti trhu periodik tak, jak tomu bylo dříve pod hlavičkou Beta&sons, zároveň však i nabídky knižních titulů podle stejného klíče, jakým byly selektovány cílové skupiny jejich původních uživatelů. OLTP databáze firmy Beta&sons zůstává nadále plně funkční. Cílem týmu je synchronizovat (integrovat) data společnosti Beta&sons do marketingového datamartu Alfa co.

---

[3] **Online Transaction Processing** - technologie uložení dat v databázi, umožňující jejich nejsnadnější a nejbezpečnější modifikaci v mnoha-uživatelském prostředí.

[4] **Extraction, Transforamtion, Load** - extrahování, transformace, nahrání. Též **datová pumpa** - technologie/koncept umožňující přenos a transformaci dat ze zdrojových systémů do databáze datového skladu.

[5] **Online Analytical Processing** - technologie uložení dat v rozsáhlých databázích určená k analytickým účelům.

[6] **Datamart** - subsystém datového skladu (data warehouse) zaměřený na určitou oblast. Může existovat i samostatně jako jednoduchý datový sklad.

Samotný mechanismus přesunu dat ze struktury společnosti Beta&sons (datové pumpy, bulk operace (jednorázová načtení), exporty souborů (dumps) atp.) není předmětem projektu vašeho týmu. Proveďte potřebné kroky k integraci tohoto systému do OLAP struktury databáze firmy Alfa co. včetně vytvoření potřebné dokumentace k jednotlivým etapám.

Doplňující informace k projektu

Firma Alfa co.: Způsob uložení dat v OLAP systému a jejich význam je vám znám. Schéma uložení dat nebude z důvodu integrace zákaznických dat ze systému Beta&sons nijak upravováno ani rozšiřováno, a to z důvodů již namapovaného nástroje, který využívá oddělení marketingu pro tvorbu kampaní.

Firma Beta&sons: Tým firmy Alfa co. má od firmy Beta&sons k dispozici databázové spojení (database link), přiděleny přihlašovací údaje a práva pro čtení z tabulek, které by měly obsahovat informace relevantní pro rozhodování o zasílání emailových kampaní i z tabulek, ve kterých jsou uchovávána systémová metadata. Vzhledem ke špatným vztahům mezi firmami (předpokládaný zánik oddělení marketingu Beta&sons), lze ve vzájemné komunikaci s tvůrci a správci systému Beta&sons očekávat nanejvýše potvrzení či vyvrácení vašich předpokladů.

1. Analýza a klasifikace kvality dat – Data profiling

Řada společností si teprve nyní uvědomuje, jak málo pozornosti věnovala kvalitě dat v průběhu mnohaletého vývoje svých systémů. Zatímco se lhůty dodání řešení výrazně zkracovaly, velikost a komplexita jednotlivých projektů naopak narůstaly. Společnosti byly nuceny implementovat aplikace v časovém horizontu, který by byl akceptovatelný oddělením prodeje služeb. Vzhledem k časové tísni bylo nutné vzdát se některých nároků na výsledné aplikace. Díky přístupu řídícího se principem "**Time to market**[7]" vzniká řada dočasných, často nekonzistentních řešení, která mají za následek nekvalitní data ve zdrojových systémech a s tím i problematickou interpretaci informací v nich uložených. (ADELMAN, 2005).

Automatizovaná profilace dat je velmi efektivní metoda pro určení kvality dat. Důkladnou a správně navrženou analýzou lze objevit pro uživatele či marketingového pracovníka často neviditelné datové nepřesnosti, které mohou narušovat funkčnost celého systému nebo negativně ovlivňovat jeho budoucí rozvoj či související obchodní taktiku.

Podívejme se nyní na nejčastější příklady chyb, které lze identifikovat v relačních či jiných databázích, a které lze díky navrhované analýze odhalit.

• **Nekorektní data -** Aby mohla být data označena za korektní resp. validní, musí všechny jejich hodnoty pocházet z domény, jež je definuje. Například měsíc v roce musí být v rozmezí 1-12, věk osoby by neměl přesáhnout 130 let, apod. Korektnost dat lze často programově ošetřit například pomocí lookup tabulek (číselníků) a souvisejících omezení (constrains) např. cizích klíčů.

• **Nepřesná data** - Ačkoliv je hodnota v datech korektní, přesto může být nepřesná. Například pro kód amerického státu "CA" a název města "Boston" jsou obě hodnoty z

[7] **Time to market (TTM) -** je čas, který uplyne od doby, kdy byl produkt navržen do momentu jeho komerčního spuštění či uvolnění do prodeje. Jedná se o dobu od nápadu a rozhodnutí o jeho realizaci do momentu úplné realizace. V rámci konkurenčního boje je na tento ukazatel kladen veliký důraz.

pohledu domény korektní. Jsou-li použity společně, je již tato informace nepřesná (Boston leží ve státě Massachusetts). Přesný kód státu by měl být tedy "MA". Přesnost, a to obzvláště u závisejících datových hodnot, se bohužel velmi těžce programově zajišťuje a ani využití číselníků nemusí vézt k odstranění těchto problémů. I když je někdy možné zkontrolovat kontextuální přesnost dat vůči jiným polím nebo celým souborům, často nezbývá jiná možnost, než její manuální ověření (rozumějme pomocí ad-hoc skriptů atp.) nebo jejich postoupení další osobě například analytikovi s patřičným know-how, zákazníkovi, který potvrdí pravdivost o něm uchovávané informace či vendorovi systému, který má potřebné informace a znalosti pro verifikaci takových dat

• **Data, která nesplňují obchodní pravidla –** Dalším problémem kvality dat je jejich nekonzistentnost vzhledem k obchodním pravidlům tzn. s jejich logickou podstatou. Jednoduchým příkladem takové chyby může být záznam o uživateli, který má datum vytvoření objednávky dřívější než datum své první registrace do systému. Tato chyba vzniká nejčastěji nesprávnou synchronizací aplikací či jejích modulů, které zajišťují dva různé business procesy např.: a) registrace uživatele v nějakém **CRM**[8] systému b) vytvoření objednávky v elektronickém obchodě. Nový uživatel vstupuje do systému (registruje se) za účelem vytvoření své první objednávky. Tyto dva kroky nemají větší časovou dilataci. (Standardní souslednost kroků při nákupu v e-shopu bývá taková, že zákazník nejdříve plní "nákupní košík" a až poté, při odeslání hotové objednávky vyplňuje údaje o sobě, jako jsou nacionále, email, či adresa). Při integraci výstupů těchto dvou procesů musí být brána v potaz jejich logická časová souslednost.

• **Nekonzistentní data -** Nekonzistence dat pochází ze zásadního problému, který je ovšem přítomný ve velké většině organizací. Jedná se o tzv. datovou redundanci (přílišná kumulace a nadbytek podobných dat). Toto je velmi obvyklý jev právě u informací o zákazníkovi. Ten například figuruje v bázi objednávek jako "Jason Clark" v databázi zákazníků pak jako "Jason Louise Clark" a v marketingovém datamartu jako "Jason L. Clark". Stejná entita resp. její atributy tak nabývají v různých instancích jiné hodnoty.

[8] **Customer relationship management** - (řízení vztahů se zákazníky) je databázovou technologií podporovaný proces shromažďování, zpracování a využití informací o zákaznících firmy.

Ačkoliv mohou být jednotlivé instance efektně propojeny například pomocí souvisejícího rodného čísla, je takováto nekonzistence nežádoucí.

• **Nekompletní data** - V průběhu definice požadavků na daný systém jsou často opomíjeni jeho sekundární uživatelé, kteří jsou dále po "proudu" toku informací. To může být již dříve zmiňované oddělení marketingu. Vytváříme-li například systém pro finanční instituci, který má spravovat výhradně evidence půjček zákazníků, zajímají nás převážně údaje o výši půjčky, výši měsíční splátky, výši úroku a další, pro daný účel kritické informace. Pro oddělení marketingu této instituce jsou ovšem důležitější údaje typu: věk zákazníka, pohlaví zákazníka či rodinné poměry, protože chtějí zákazníka oslovovat například s nabídkou nových na míru upravených produktů. Tyto údaje nebyly z důvodu primárního zaměření aplikace v prvotním designu požadovány ani zohledněny. To je důvod, proč řada datových prvků ve fungujících systémech zcela chybí, anebo používá přednastavené (defaultní) hodnoty.

• **Neintegrovaná data** - Většina organizací uchovává data redundantně a nekonzistentně v celé řadě svých systémů, které nikdy nebyly navrženy za účelem možné integrace. Záznamy nelze jednoznačně identifikovat, primární klíče se mezi systémy liší, nejsou unikátní ani v rámci jediného systému (zde nehovoříme o primárním klíči tabulky, který, pokud je implementován, samozřejmě jedinečný být musí), nebo vůbec neexistují. Navíc je stále častěji vývoj a správa systémů svěřována osobám neznajícím fungovaní celé "mateřské" organizace, nebo je kompletně předána do správy nějaké třetí straně (DAVENPORT, COHEN, JACOBSON, 2005). V tuto chvíli je kvalita a konzistence dat hlavním rizikem. Vezměme si příklad, kdy stejný zákazník existuje ve dvou či více systémech daných do správy různým firmám. V každém systému figuruje pod jiným identifikačním číslem, jeho jméno je napsáno rozdílně a jeho adresa nebo telefon se také liší. Vzájemná integrace takových dat se pak stává velice problematickou.

Z námi uvedených příkladů vyplývají i nejčastější problémy související s kvalitou dat. Ty lze konkretizovat následovně:

- Datové prvky jsou použity k jiným než původně zamýšleným účelům
- Existence prázdných sloupců (zcela bez dat)
- Chybné hodnoty ve sloupcích
- Nekonzistence v reprezentaci stejné hodnoty
- Chybějící hodnoty
- Porušení strukturálních závislostí
- Porušení předpokládaných vztahů v rámci sloupce (např. pořadí datumů)
- Porušení obchodních pravidel
- Nerealistická procenta specifických hodnot v daném sloupci

Tyto a jiné problémy, pokud jsou v systému identifikovány, jsou předkládány business analytikovi, který se pokusí určit jejich příčinu a navrhne jejich následnou opravu, je-li to možné (OLSON, 2002).

Zde uvedený výčet možných chyb nemusí být samozřejmě konečný. Chyby mohou být generovány prakticky v každém kroku počínaje prvotním ukládáním dat do databází, jejich přenosem či špatnou archivací. Studie "A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing" (SINGH, SINGH, 2010) jich pouze na úrovni zdroje dat, tedy chyb vznikajících na rozhraní aplikace-databáze, identifikuje celkem 52.

Analytické techniky aplikované na data vypovídají o správnosti jejich struktury, obsahu a kvalitě. Tyto techniky jsou zcela odlišné od těch, které se používají pro účely obchodních rozhodnutí. Business analýza dat je založena na podpoře BI systémů a na dataminingu resp. na hledání souvislostí a vytvoření obchodovatelné informace z dat, zatímco zde popisovaná profilace dat vytváří informaci o datech samotných, tedy

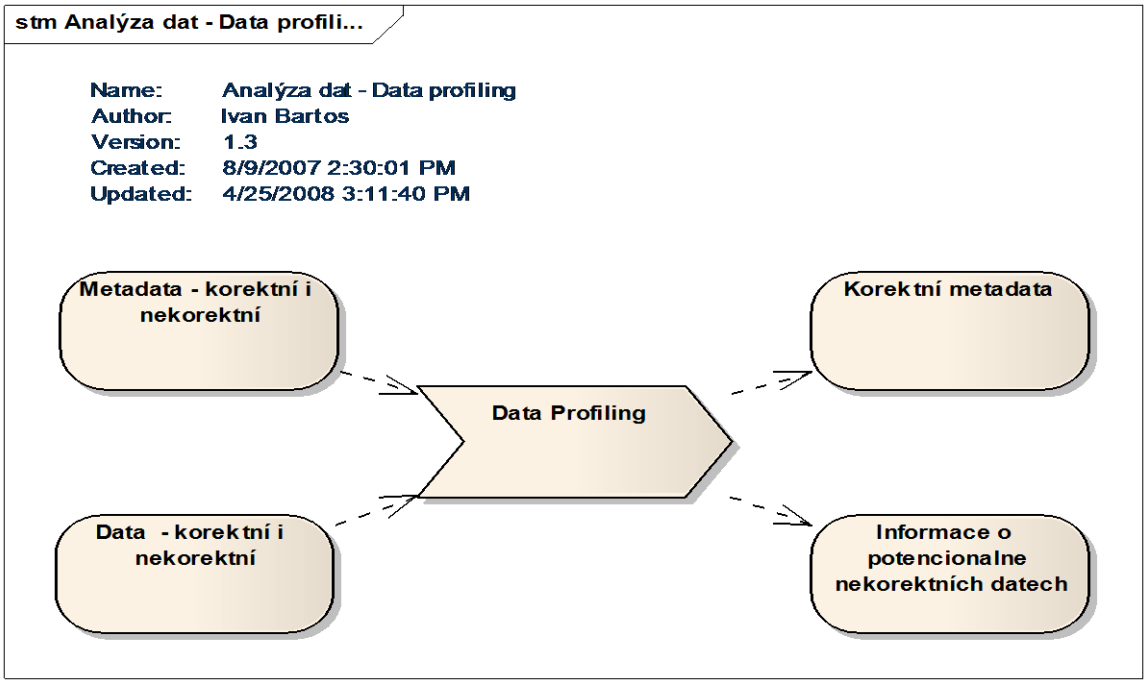
metadata[9]. Její pravidla určují, která data lze označit za akceptovatelná (korektní), resp. validní, a která nikoliv.

Při datové profilaci je třeba nejdříve shromáždit skupinu metadat reprezentujících pravidla, která jsou specifická pro danou aplikaci či systém. Následně jsou data obsažená v systému porovnávána vzhledem těmto pravidlům. Tak je určena jejich korektnost či nekorektnost.

Nelze předpokládat, že je kompletní soubor metadat (pravidel) k dispozici již na počátku analýzy. Nová metadata jsou objevována při samotném procesu profilace, a to zejména v případě, kdy na počátku buď žádná metadata neexistují, nebo je jejich kvalita velmi nízká. Ačkoliv se metody automatické profilace neustále zdokonalují, tento proces nakonec vždy vyžaduje lidský zásah. Získaná metadata hrají pak v budoucnu důležitou roli při monitorování a zaručení kvality dat v systému.

I přesto, že existuje řada komerčních nástrojů od společností, jako jsou například Ascential Software, DataFlux, Evoke Software, Firstlogic, Informatica nebo Trillium Software, které dokáží v různé míře pokrýt jednotlivé kroky, poměrně vysoké procento společností (57%) využívá k analýze dat vlastní **SQL**[10] dotazy nebo uživatelské funkce a reporty (ECKERSON, 2004). Důvodů lze najít hned několik. Hlavní příčinou bývá neochota investovat do nákupu drahé aplikace, jejíž funkcionalita může být substituována z vlastních zdrojů, nebo jejíž cena neodpovídá náročnosti a hodnotě řešených projektů. Dalšími faktory mohou být například hardwarová náročnost (analýza obsáhlých datových struktur může trvat i na výkonných strojích celé dny a navíc ji nelze aplikovat na živých-realtime fungujících systémech), přílišná komplexita takové aplikace, nebo nedostatečná kvalifikace pro její používání. Pro potřeby této práce předpokládejme zařazení právě do té skupiny firem, která se rozhodne kvalitu dat a informací řešit vlastními silami.

[9] **Metadata** – název pochází z řeckého meta = mezi, za + latinského data = to, co je dáno. Metadat jsou strukturovaná data o datech.
[10] **Structured Query Language** - je standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích.



Obrázek 2: Model profilace dat

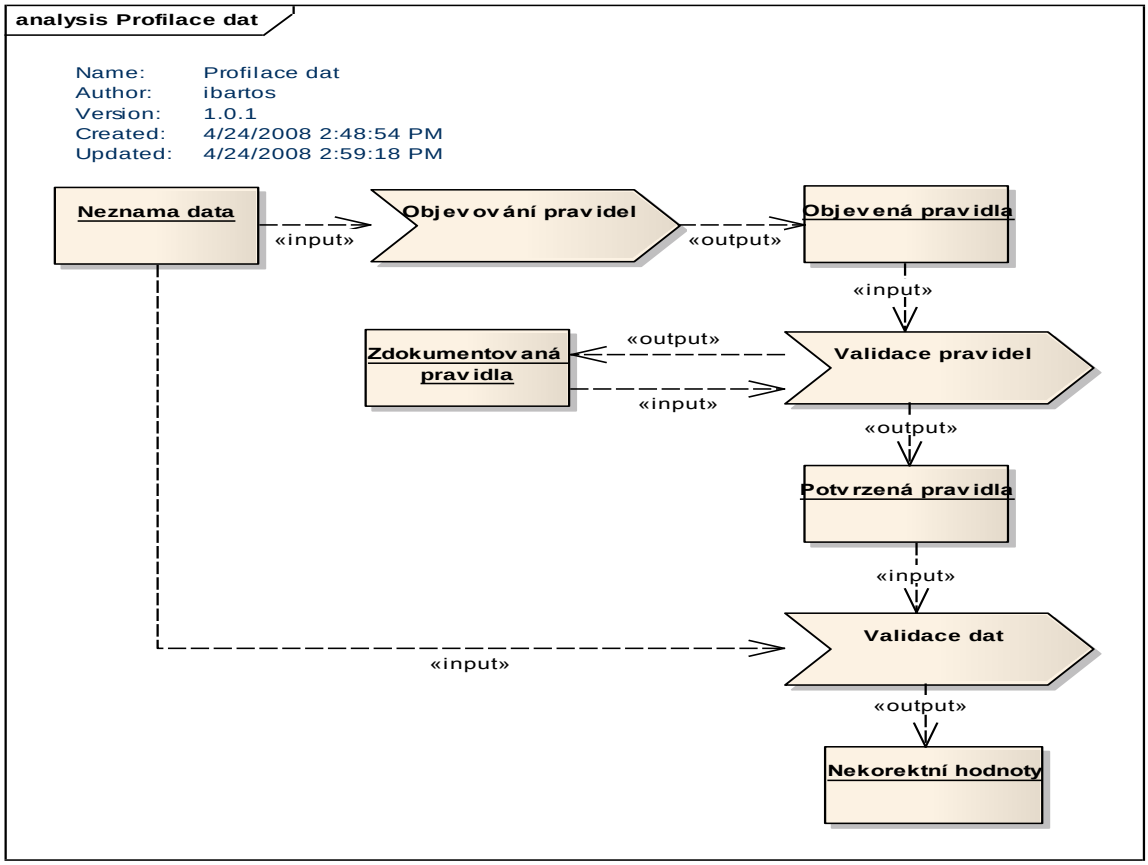
1.1 Proces analýzy a profilace dat (teorie)

Samotný proces analýzy (prvotní pohled na data založený na obecných znalostech datových prvků a jejich sémantiky) a profilace (aplikace technik určených k získání metadat) dat vyžaduje v každé fázi určitou množinu pravidel. Tato pravidla jsou na počátku procesu často neznámá či vágně definovaná. Proto je třeba pravidla průběžně vytvářet v iterativním (v cyklech se opakujícím) procesu, který se skládá z následujících kroků: objevování pravidel, validace pravidel (vyžadující lidský zásah) a validace samotných dat, tedy testování dat oproti těmto pravidlům.

Získávání (objevování) pravidel předpokládá existenci mechanismu, který dokáže odhadnout pravidlo, aniž by bylo předem přesně definováno, tedy pouze na základě vyhodnocení profilovaných dat. Tyto mechanismy mohou objevit například vlastnosti sloupců, funkční závislosti, primární klíče nebo duplicitní datové sloupce.

Validace zjištěných pravidel je založena na porovnání objevených vlastností s metadaty a případné spolupráci s business analytikem, který potvrdí správnost takového pravidla.

Posledním krokem je validace dat, při které jsou data porovnávána s potvrzenými platnými pravidly za účelem objevení takových, která pravidlo nesplňují. Výstupem je dokument obsahující informace o potenciálně nekorektních datech, který je předáván dále, nejčastěji osobám zodpovědným za kvalitu dat, aby patřičně upravily stávající systém, nebo je prezentován oddělení marketingu, aby přehodnotilo své požadavky, kterým kvalita profilovaných dat neodpovídá (např. kritérium věku v určité kampani nelze zohlednit).



Obrázek 3: Profilace dat

Jak již bylo zmíněno v úvodu, uvažujme, že máme k dispozici pouze data o nám blíže nespecifikovaných entitách (jejich atributy nám nejsou známy), která jsou uložena v libovolných relacích. Aniž bychom se nyní opírali o jakékoliv sémantické prvky (názvy tabulek, názvy sloupců, aliasy atp.), můžeme již takto neklasifikovaná data analyzovat.

V tuto chvíli nepracujeme s možnou interpretací těchto dat ani s extrakcí informace v nich uložené, ale analyzujeme převážně jejich typ, kvalitu a kvantitu.

A) V analyzovaném systému má entita **Uživatel** přiřazen atribut **Pohlaví.** Atribut je typu **řetězec o délce jednoho znaku**. Tento atribut nemusí být vyplněn (není zde omezení NOT NULL), přesto je v 80% zadán. Je v něm uloženo **18 rozdílných znakových hodnot**. Na základě datové analýzy je tento atribut **vhodný pro převzetí z hlediska četnosti jeho využití.**

B) Pokud ovšem zvážíme sémantický význam názvu atributu Pohlaví, dospějeme k závěru, **že může nabývat dvou, respektive tří hodnot: (M)už, (Ž)ena, NULL (nespecifikováno).** Pokud **je distribuce 18 hodnot dle A) rovnoměrná**, tedy nepřevládá výrazně M, Ž a NULL, je tento atribut **pro integraci nevhodný**. Atribut **může být přesto integrován** po transformaci, ale již **s menší informační hodnotou** vzhledem ke zdrojovému systému, kdy u většiny uživatelů nemůžeme určit pohlaví, ačkoliv oni nějaké zadali.

Příklad 1: Kvalita dat – kvalita informace

Objevená pravidla je třeba uchovávat pro budoucí použití (viz. již dříve zmíněný iterativní proces profilace). Standardním způsobem uchování zjištěných pravidel a omezení, mají-li být použita pro automatickou nebo poloautomatickou profilaci, je formát **XML**[11] (s ním pracují i dříve zmiňované komerční nástroje). Formát XML je snadno komunikovatelný a je zároveň univerzální pro implementaci do různých řešení, která profilaci dat realizují.

Způsob uložení metadat a pravidel pro validaci
Úroveň 1:
Tabulky **<tables>**
Lze rozvinout o

Úroveň 2:
Název tabulky **<table_name> [Table Name] </table_name>**
Informace o tabulce **<table_info> </table_info>**
Lze rozvinout o

Úroveň 3:
Název sloupce **<column_name> [Column Name] </column_name>**
Informace o sloupci **<column_info> </column_info>**
Lze rozvinout o
Datová pravidla **<sdata_rules> </sdata_rules>**
Úroveň 4

- V rámci **Column_Info** tagu v Úrovně 3 je li datový typ číslo. Je třeba specifikovat:
  Datový typ **<data_type>number</data_type>**
  Minimální hodnota **<min_value>100</min_value>**
  Minimální hodnota **<max_value>999999</max_value>**
  Možnost nulové hodnoty **<null>not null</null>**
  Unikátní **<unique>no</unique>**
- V rámci **Column_Info** tagu v Úrovně 3 je li datový typ řetězec Je třeba specifikovat:
  Datový typ **<data_type>string</data_type>**
  Délka **<length>20</length>**
  Možnost nulové hodnoty **<null>null</null>**
  Unikátní **<unique>no</unique>**
- V rámci **sdata_rules** tagu v Úrovně3. Je možné specifikovat další tagy:
  Pravidlo **<srule>Výplata je menší nebo rovna 35000 </srule>**
  (V **<srule>** tagu lze specifikovat pravidlo buď v přirozeném jazyce, nebo matematickou formulí)

Příklad 2: Návrh struktury XML metadat pro validaci dat (pravidla)

Vzniklý dokument je pak odkazován programovou jednotkou, která analýzu a validaci provádí (viz. kapitola 1.5). V této práci není v příkladu návrhu aplikace reference na XML dokument s pravidly použita. Pro větší názornost jsou pravidla implementována v jednotlivých částech kódu, které řeší vždy jedno definované pravidlo. Způsob uložení těchto pravidel však opět záleží na potřebách organizace. Pro komunikaci objevených pravidel je v práci popsána výstupní tabulka, která je následně převedena do reportu v nějakém zvoleném formátu (ODT, DOC, RTF).

[11] **Extensible Markup Language** - obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů, u kterých popisuje strukturu z hlediska věcného obsahu jednotlivých částí,

1.2 Extrakce metadat uložených v systému DBMS

Databázové systémy, ve kterých jsou data uložena, uchovávají informace o způsobu jejich uložení v jednotlivých tabulkách ve svém tzv. **data dictionary**[12], což je v podstatě skupina pohledů na systémové tabulky, tedy opět tabulky-relace. Pro každou tabulku s daty lze automaticky či ad-hoc vytvářet statistiky, které se do data dictionary ukládají, a to ihned po jejím vytvoření, nebo po změně (přidání či odebrání) dat. Jsou to metadata, která o svých datech uchovává samotný **DBMS**[13].

```
BEGIN
     DBMS_STATS.GATHER_TABLE_STATS('<VLASTNÍK>','<NÁZEV TABULKY>');
END;
```

Příklad 3: Ad-hoc shromáždění metadat o dané tabulce (Oracle)

```
SELECT COLUMN_NAME, DATA_TYPE, DATA_LENGTH, DATA_PRECISION,DATA_SCALE, NULLABLE,
NUM DISTINCT, LOW VALUE, HIGH VALUE, NUM NULLS, AVG_COL_LEN, CHAR_LENGTH FROM
ALL TAB COLS WHERE TABLE NAME = <název tabulky>
```

Příklad 4: Získání zvolených informací z data-dictionary (Oracle)

Výsledkem extrakce těchto metadat pak může být například následující tabulka:

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE
ID_CODE_TXT	VARCHAR2	36		
DATE_CREATED	DATE	7		
DATE_MODIFIED	DATE	7		
LAST_MAILBOX_CHECK_DATE	DATE	7		
FIRST_NAME_TXT	VARCHAR2	60		
AGE_CURRENT	NUMBER	22		
RACE_HISPANIC_TXT	VARCHAR2	3		

NULLABLE	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE
N	63517	30303034464E4B524	5A5A5A49
Y	62683	77C4091A012A03	786B080A052636
Y	21519	786B0809060514	786B080A053A1D
Y	0		
Y	2804	27544F4249	5A555249
Y	150	6331	C1415F64
Y	1	59	59

[12] **Data dictionary –** centrální úložiště (repository) informací o datech jako je jejich význam, vztah k jiným datům, původ, způsob užití a formát. Označováno také jako metadat repository (úložiště metadat).

[13] **Database management system** – komplexní množina softwarových programů, která řídí organizaci, uložení, správu a získávání dat z databáze. DBMS obsahuje modelovací jazyk pro schéma databáze, datové struktury (pole, záznamy, soubory, objekty), dotazovací jazyk a transakční mechanismy zajišťující integritu dat.

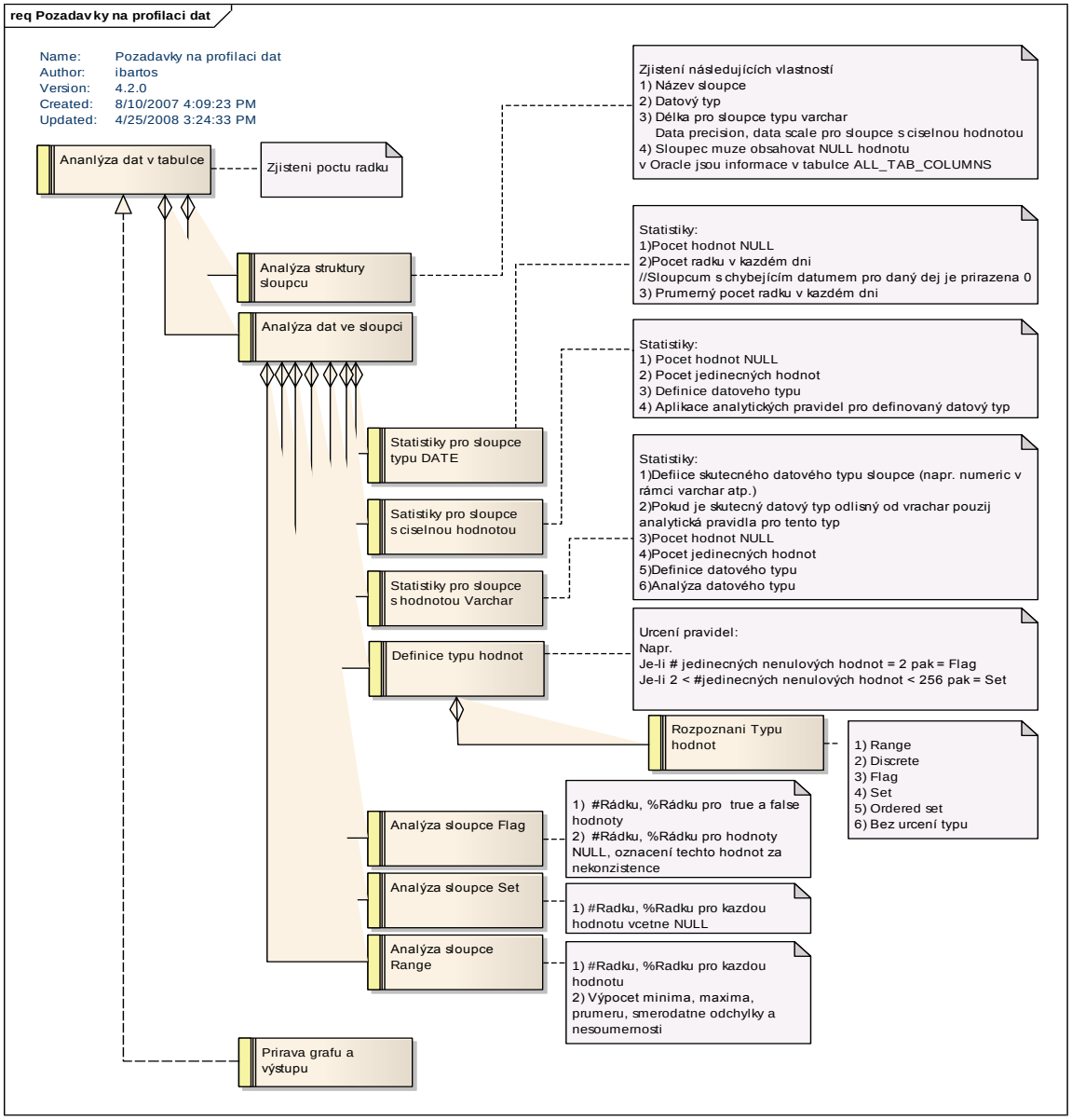
NUM_NULLS	AVG_COL_LEN	CHAR_LENGTH
0	13	36
0	8	0
0	8	0
63517	1	0
3882	7	60
7750	5	0
55971	2	3

Příklad 5: Metadata z pohledu ALL_TAB_COLS (Oracle)

1.3 Rozšíření metadat pomocí uživatelských skriptů

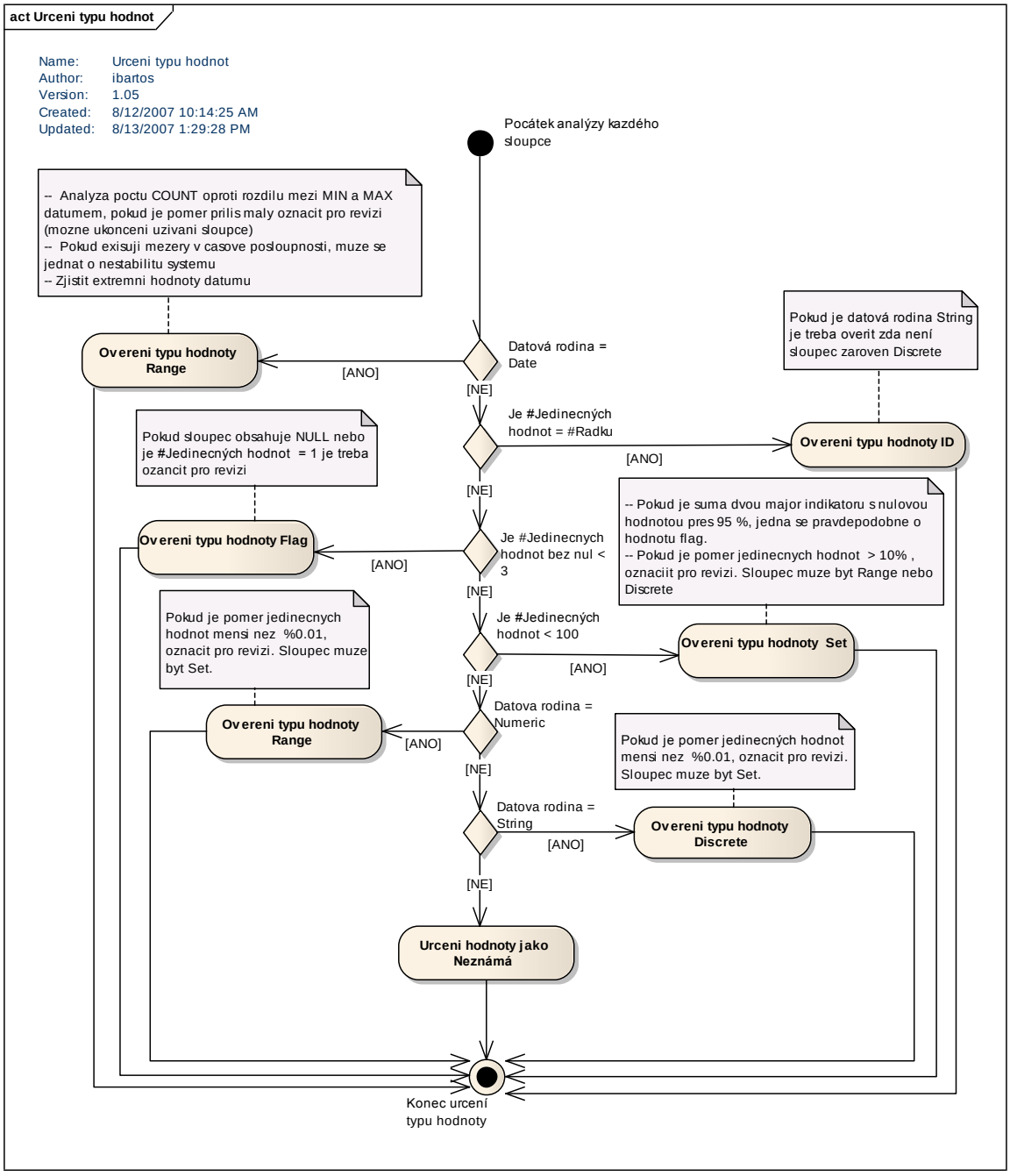
Základní analýza popsaná v předchozí kapitole postihuje pouze elementární vlastnosti uložených dat. Pro sofistikovanější analýzu je třeba použít vhodný pokročilejší způsob profilace dat. K tomu mohou sloužit již existující aplikace jmenované dříve v této studii, nebo jejich integrované služby, jako například Data Profiler služba v Oracle Warehouse Builder – 10g2. Podobné nástroje nabízí velmi propracované a detailní způsoby analýzy dat s množstvím definovatelných grafických výstupů. Zároveň umožňují analyzovat změny dat v čase, monitorovat systém, a tak předcházet případné nekonzistenci dat. Problémem těchto aplikací je ovšem jejich determinovanost (nabízí omezenou, a tudíž konečnou sadu nástrojů) a hlavně jejich provozní náročnost (celkové zpomalení profilovaného systému).

Druhou možností je vytvořit vlastní automatizovaný způsob profilace dat. Ten ovšem předpokládá hlubší znalost programovacích a selekčních jazyků a orientaci v prostředí a problematice databázových systémů. Napsání aplikace, která profilaci provádí, je velmi komplexní a časově náročný úkol, kterému by měla předcházet důkladná analýza potřeb. Proto se nejdříve zaměříme na požadavky, které jsou na aplikaci resp. uživatelský skript, který bude pravděpodobným výstupem procesu jejího vytváření, kladeny (viz. následující obrázek).



Obrázek 4: Požadavky na aplikaci, která umožní automatickou profilaci dat

Jednotlivé požadavky odpovídají určitým funkčním celkům výsledné aplikace. Hloubka analýzy zcela závisí na potřebách řešeného projektu. Uveďme si pro příklad detailnější návrh algoritmu pro analýzu typu hodnoty v rámci určitého atributu (sloupce).



Obrázek 5: Aktivity potřebné pro určení typu hodnoty v určitém atributu (sloupci)

Výsledkem podrobného rozboru požadavků a následně pravidel specifických pro data uložená v databázi je, po převedení do kódu, aplikace resp. programová jednotka, která dokáže, pokud možno automaticky, provést profilaci dat na požadované úrovni, tj. vyprodukovat potřebná metadata, která popisují pravidla platná v profilovaném systému, popřípadě generovat vhodné reporty pro oddělení, která s těmito výsledky dále pracují.

Toto je konkrétní programový balíček (package) v jazyce PL/SQL pro prostředí Oracle včetně komentářů v kódu.

```
create or replace package PKG_TABLE_ANALYZE is
  -- Created: 5/25/2010 3:10:20 PM
  -- Created by: IVAN Bartos
  -- Purpose: Provede automatickou analyzu dat v tabulce
  -- Parameters: p_owner: Vlastnik tabulky; p_table_name: Jmeno zdrojove tabulky
  -- Results: Vysledky budou ulozeny v ('DAS_' || p_owner || '_' || p_table_name)
tabulce
  -- Pokud bude vysledny nazev tabulky delsi nez 30 znaku, bude oriznut na prvnich 30
znaku.

PROCEDURE ANALYZE(p_owner VARCHAR2, p_table_name VARCHAR2);

end  PKG_TABLE_ANALYZE;
```

```
create or replace package body PKG_TABLE_ANALYZE IS
-- Definice konstant
-- Rodina dat
c_data_family_date CONSTANT VARCHAR2(10) := 'DATE';
c_data_family_string CONSTANT VARCHAR2(10) := 'STRING';
c_data_family_numeric CONSTANT VARCHAR2(10) := 'NUMERIC';
c_data_family_unkown CONSTANT VARCHAR2(10) := 'OTHER';
-- Typy hodnot
c value type id CONSTANT VARCHAR2(10) := 'ID';
c_value_type_flag CONSTANT VARCHAR2(10) := 'Flag';
c value type set CONSTANT VARCHAR2(10) := 'Set';
c_value_type_range CONSTANT VARCHAR2(10) := 'Range';
c_value_type_discrete CONSTANT VARCHAR2(10) := 'Discrete';
c_value_type_unknown CONSTANT VARCHAR2(10) := 'Unknown';
-- Privatni funkce vytvarejici jmeno DAS tabulky pro analyzu
FUNCTION GET DAS TABLE_NAME(p_owner VARCHAR2, p_table_name VARCHAR2)
  RETURN VARCHAR2 IS
BEGIN
  RETURN SUBSTR('DAS_' || p_table_name, 1, 30);
END;
-- Privatni procedura vytvarejici DAS tabulku pro analyzu
PROCEDURE CREATE TABLE FOR ANALYSIS(p owner VARCHAR2, p table name VARCHAR2) IS
a_table_name VARCHAR2(30) := GET_DAS_TABLE_NAME(p_owner, p_table_name);
a_count NUMBER;
a_sql_string VARCHAR2(4000);
BEGIN
  SELECT COUNT (*) INTO a_count
    FROM USER OBJECTS
    WHERE OBJECT_NAME = a_table_name;
    IF a_count > 0 THEN
      a sql string := 'DROP TABLE ' || a_table_name;
        EXECUTE IMMEDIATE a_sql_string;
    END IF;
    a_sql_string := 'CREATE TABLE ' || a_table_name || '(
          COLUMN_ID NUMBER(4) NOT NULL,
                    COLUMN_NAME VARCHAR2(30) NOT NULL,
                    DATA TYPE VARCHAR2(30) NOT NULL,
                    DATA_LENGTH NUMBER(10),
                    DATA PRECISION NUMBER(10),
                    DATA_SCALE NUMBER(10),
                    DATA_NULLABLE VARCHAR2(1) NOT NULL,
                    ROW_COUNT NUMBER(10),
                    DISTINCT COUNT NUMBER(10),
                    DISTINCT RATIO NUMBER(5,2),
                    NULL_COUNT NUMBER(10),
                    NULL_RATIO NUMBER(5,2),
                    MIN_VALUE VARCHAR2(50),
                    MAX VALUE VARCHAR2(50),
                    AVG_VALUE NUMBER(12,2),
                    STD VALUE NUMBER(12,2),
                    GRAPH_DATA CLOB,
                    VALUE_TYPE VARCHAR2(10),
                    VALUE_REVIEW NUMBER(4),
```

```
                    ANALYZE_DATE DATE DEFAULT SYSDATE NOT NULL)';
  EXECUTE IMMEDIATE a_sql_string;
  a_sql_string := 'INSERT INTO ' || a_table_name ||
          '(COLUMN_ID, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, DATA_PRECISION,
DATA_SCALE, DATA_NULLABLE)
                    SELECT COLUMN_ID, COLUMN_NAME, DATA_TYPE, DATA_LENGTH,
DATA_PRECISION, DATA_SCALE, NULLABLE
                    FROM ALL_TAB_COLUMNS WHERE OWNER = ''' || p_owner || ''' AND ' ||
'TABLE_NAME = ''' || p_table_name
                    || ''' ORDER BY COLUMN_ID';
    EXECUTE IMMEDIATE a_sql_string;
    COMMIT;
END;
-- Privatni funkce urcujici datovou rodinu
FUNCTION GET_DATA_FAMILY(p_data_type VARCHAR2) RETURN VARCHAR2 IS
BEGIN
  IF p data type = 'DATE' THEN
      RETURN c_data_family_date;
    ELSIF p_data_type IN ('FLOAT', 'LONG', 'NUMBER') THEN
      RETURN c_data_family_numeric;
    ELSIF  p_data_type IN ('CHAR', 'NCHAR', 'NVARCHAR', 'NVARCHAR2', 'VARCHAR',
'VARCHAR2') THEN
      RETURN c_data_family_string;
    ELSE
      RETURN c_data_family_unkown;
    END IF;
END;
-- Privatni procedura analyzujici sloupec typu date
PROCEDURE ANALYZE_DATE_COLUMN(p_owner VARCHAR2, p_table_name VARCHAR2, p_column_name
VARCHAR2) IS
a_table_name VARCHAR2(30) := GET_DAS_TABLE_NAME(p_owner, p_table_name);
a_sql_string VARCHAR2(8000);
a_row_count NUMBER(10);
a distinct count NUMBER(10);
a distinct ratio NUMBER(5,2);
a_null_count NUMBER(10);
a_null_ratio NUMBER(5,2);
a_min_value VARCHAR2(50);
a max value VARCHAR2(50);
a_cur INTEGER;
a exec_result INTEGER;
BEGIN
  a_sql_string := 'SELECT
                      ROW_COUNT,
                        DISTINCT COUNT,
                        DISTINCT_RATIO,
                        ROW COUNT - NOTNULL COUNT AS NULL COUNT,
                        ROUND((ROW_COUNT - NOTNULL_COUNT) * 100 / ROW_COUNT, 2) AS
NULL_RATIO,
                        MIN_DATA,
                        MAX_DATA
                    FROM
                      (
                        SELECT
                          COUNT(*) AS ROW_COUNT,
                            COUNT(' || p column name || ') AS NOTNULL COUNT,
                            COUNT(DISTINCT TRUNC(' || p_column_name || ')) AS
DISTINCT COUNT,
                            ROUND(COUNT(DISTINCT TRUNC(' || p_column_name || ')) * 100
/ CASE WHEN COUNT(TRUNC(' || p_column_name || ')) = 0 THEN 1 ELSE COUNT(TRUNC(' ||
p_column_name || ')) END, 2) AS DISTINCT_RATIO,
                            MIN(' || p column name || ') AS MIN DATA,
                            MAX(' || p_column_name || ') AS MAX_DATA
                        FROM ' || p_table_name || ')';
-- Iniciace kurzoru
    a_cur := DBMS_SQL.OPEN_CURSOR;
    DBMS SQL.PARSE(a cur, a sql_string, DBMS_SQL.NATIVE);
-- Definice sloupcu pro kurzor
    DBMS SQL.DEFINE COLUMN(a cur, 1, a row count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 2, a_distinct_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 3, a_distinct_ratio);
```

```
    DBMS_SQL.DEFINE_COLUMN(a_cur, 4, a_null_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 5, a_null_ratio);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 6, a_min_value, 50);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 7, a_max_value, 50);
-- Spusteni SQL
  a_exec_result := DBMS_SQL.EXECUTE(a_cur);
    LOOP
-- Nacteni dalsi radky
      IF DBMS_SQL.FETCH_ROWS(a_cur) > 0 THEN
-- Ziskani hodnot pro danou radku
      DBMS SQL.COLUMN VALUE(a cur, 1, a row count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 2, a_distinct_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 3, a_distinct_ratio);
            DBMS_SQL.COLUMN_VALUE(a_cur, 4, a_null_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 5, a_null_ratio);
            DBMS_SQL.COLUMN_VALUE(a_cur, 6, a_min_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 7, a_max_value);
      ELSE
          EXIT;
        END IF;
    END LOOP;
  DBMS_SQL.CLOSE_CURSOR(a_cur);

    a_sql_string := 'UPDATE ' || a_table_name || ' SET
            ROW_COUNT = ' || a_row_count || ',
                    DISTINCT_COUNT = ' || a_distinct_count || ',
                    DISTINCT_RATIO = ' || a_distinct_ratio || ',
                    NULL COUNT = ' || a null count || ',
                    NULL_RATIO = ' || a_null_ratio || ',
                    MIN VALUE = ''' || a min value || ''',
                    MAX_VALUE = ''' || a_max_value || '''
                    WHERE COLUMN_NAME = ''' || p_column_name || '''';
    EXECUTE IMMEDIATE a_sql_string;
END;
-- Privatni procedura analyzujici sloupec typu numeric
PROCEDURE ANALYZE_NUMERIC_COLUMN(p_owner VARCHAR2, p_table_name VARCHAR2, p_column_name
VARCHAR2) IS
a_table_name VARCHAR2(30) := GET_DAS_TABLE_NAME(p_owner, p_table_name);
a sql string VARCHAR2(8000);
a_row_count NUMBER(10);
a distinct count NUMBER(10);
a_distinct_ratio NUMBER(5,2);
a_null_count NUMBER(10);
a_null_ratio NUMBER(5,2);
a min value VARCHAR2(50);
a_max_value VARCHAR2(50);
a avg value NUMBER(12,2);
a_std_value NUMBER(12,2);
a_cur INTEGER;
a_tmp INTEGER;
BEGIN
  a sql string := 'SELECT
                      ROW_COUNT,
                        DISTINCT_COUNT,
                        DISTINCT_RATIO,
                        ROW COUNT - NOTNULL COUNT AS NULL COUNT,
                        ROUND((ROW_COUNT - NOTNULL_COUNT) * 100 / ROW_COUNT, 2) AS
NULL RATIO,
                        MIN_DATA,
                        MAX_DATA,
                        AVG_DATA,
                        STD_DEV_DATA
                    FROM
                      (
                        SELECT
                          COUNT(*) AS ROW_COUNT,
                            COUNT(' || p column name || ') AS NOTNULL COUNT,
                            COUNT(DISTINCT ' || p column name || ') AS DISTINCT COUNT,
                            ROUND(COUNT(DISTINCT ' || p column name || ') * 100 / CASE
WHEN COUNT(' || p_column_name || ') = 0 THEN 1 ELSE COUNT(' || p_column_name || ') END,
2) AS DISTINCT_RATIO,
```

```
                            MIN(' || p_column_name || ') AS MIN_DATA,
                            MAX(' || p_column_name || ') AS MAX_DATA,
                            AVG(' || p_column_name || ') AS AVG_DATA,
                            STDDEV(' || p_column_name || ') AS STD_DEV_DATA
                        FROM ' || p_table_name || ')';
-- Iniciace kurzoru
    a_cur := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(a_cur, a_sql_string, DBMS_SQL.NATIVE);
-- Definice sloupcu pro kurzor
    DBMS_SQL.DEFINE_COLUMN(a_cur, 1, a_row_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 2, a_distinct_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 3, a_distinct_ratio);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 4, a_null_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 5, a_null_ratio);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 6, a_min_value, 50);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 7, a_max_value, 50);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 8, a_avg_value);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 9, a_std_value);
-- Spusteni SQL
  a_tmp := DBMS_SQL.EXECUTE(a_cur);

    LOOP
-- Nacteni dalsi radky
      IF DBMS_SQL.FETCH_ROWS(a_cur) > 0 THEN
-- Ziskani hodnot pro danou radku
      DBMS_SQL.COLUMN_VALUE(a_cur, 1, a_row_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 2, a_distinct_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 3, a_distinct_ratio);
            DBMS_SQL.COLUMN_VALUE(a_cur, 4, a_null_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 5, a_null_ratio);
            DBMS_SQL.COLUMN_VALUE(a_cur, 6, a_min_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 7, a_max_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 8, a_avg_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 9, a_std_value);
      ELSE
          EXIT;
        END IF;
    END LOOP;
  DBMS_SQL.CLOSE_CURSOR(a_cur);

    a_sql_string := 'UPDATE ' || a_table_name || ' SET
            ROW_COUNT = ' || a_row_count || ',
                    DISTINCT_COUNT = ' || a_distinct_count || ',
                    DISTINCT_RATIO = ' || a_distinct_ratio || ',
                    NULL_COUNT = ' || a_null_count || ',
                    NULL_RATIO = ' || a_null_ratio || ',
                    MIN_VALUE = ''' || a_min_value || ''',
                    MAX_VALUE = ''' || a_max_value || '''';
    IF a_avg_value IS NOT NULL THEN
        a_sql_string := a_sql_string || ', AVG_VALUE = ' || a_avg_value ;
    END IF;
    IF a_std_value IS NOT NULL THEN
        a_sql_string := a_sql_string || ', STD_VALUE = ' || a_std_value ;
    END IF;
    a_sql_string := a_sql_string || ' WHERE COLUMN_NAME = ''' || p_column_name || '''';
    EXECUTE IMMEDIATE a_sql_string;
END;
-- Privatni procedura analyzujici sloupec typu retezce
PROCEDURE ANALYZE_STRING_COLUMN(p_owner VARCHAR2, p_table_name VARCHAR2, p_column_name
VARCHAR2) IS
a_table_name VARCHAR2(30) := GET_DAS_TABLE_NAME(p_owner, p_table_name);
a_sql_string VARCHAR2(8000);
a_row_count NUMBER(10);
a_distinct_count NUMBER(10);
a_distinct_ratio NUMBER(5,2);
a_null_count NUMBER(10);
a_null_ratio NUMBER(5,2);
a_min_value VARCHAR2(50);
a_max_value VARCHAR2(50);
a_avg_value NUMBER(12,2);
a_std_value NUMBER(12,2);
```

```
a_cur INTEGER;
a_tmp INTEGER;
BEGIN
  a_sql_string := 'SELECT
                      ROW_COUNT,
                        DISTINCT_COUNT,
                        DISTINCT_RATIO,
                        ROW_COUNT - NOTNULL_COUNT AS NULL_COUNT,
                        ROUND((ROW_COUNT - NOTNULL_COUNT) * 100 / ROW_COUNT, 2) AS
NULL_RATIO,
                        MIN DATA,
                        MAX_DATA,
                        AVG_DATA,
                        STD_DEV_DATA
                    FROM
                      (
                        SELECT
                          COUNT(*) AS ROW_COUNT,
                            COUNT(' || p_column_name || ') AS NOTNULL_COUNT,
                            COUNT(DISTINCT ' || p_column_name || ') AS DISTINCT_COUNT,
                            ROUND(COUNT(DISTINCT ' || p_column_name || ') * 100 / CASE
WHEN COUNT(' || p column_name || ') = 0 THEN 1 ELSE COUNT(' || p_column_name || ') END,
2) AS DISTINCT_RATIO,
                            MIN(LENGTH(' || p_column_name || ')) AS MIN_DATA,
                            MAX(LENGTH(' || p_column_name || ')) AS MAX_DATA,
                            AVG(LENGTH(' || p_column_name || ')) AS AVG_DATA,
                            STDDEV(LENGTH(' || p_column_name || ')) AS STD_DEV_DATA
                        FROM ' || p_table_name || ')';
-- Iniciace kurzoru
    a cur := DBMS SQL.OPEN CURSOR;
    DBMS_SQL.PARSE(a_cur, a_sql_string, DBMS_SQL.NATIVE);
-- Definice sloupcu pro kurzor
    DBMS_SQL.DEFINE_COLUMN(a_cur, 1, a_row_count);
    DBMS SQL.DEFINE COLUMN(a cur, 2, a distinct count);
    DBMS SQL.DEFINE COLUMN(a cur, 3, a distinct ratio);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 4, a_null_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 5, a_null_ratio);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 6, a_min_value, 50);
    DBMS SQL.DEFINE COLUMN(a cur, 7, a max value, 50);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 8, a_avg_value);
    DBMS SQL.DEFINE_COLUMN(a_cur, 9, a_std_value);
-- Spusteni SQL
  a_tmp := DBMS_SQL.EXECUTE(a_cur);
    LOOP
-- Nacteni dalsi radky
      IF DBMS_SQL.FETCH_ROWS(a_cur) > 0 THEN
-- Ziskani hodnot pro danou radku
      DBMS_SQL.COLUMN_VALUE(a_cur, 1, a_row_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 2, a_distinct_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 3, a_distinct_ratio);
            DBMS SQL.COLUMN VALUE(a cur, 4, a null count);
            DBMS SQL.COLUMN VALUE(a cur, 5, a null ratio);
            DBMS_SQL.COLUMN_VALUE(a_cur, 6, a_min_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 7, a_max_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 8, a_avg_value);
            DBMS_SQL.COLUMN_VALUE(a_cur, 9, a_std_value);
      ELSE
          EXIT;
      END IF;
    END LOOP;
  DBMS_SQL.CLOSE_CURSOR(a_cur);

    a_sql_string := 'UPDATE ' || a_table_name || ' SET
            ROW COUNT = ' || a row count || ',
                    DISTINCT_COUNT = ' || a_distinct_count || ',
                    DISTINCT_RATIO = ' || a_distinct_ratio || ',
                    NULL COUNT = ' || a null count || ',
                    NULL RATIO = ' || a null ratio || ',
                    MIN VALUE = ''' || a min value || ''',
                    MAX_VALUE = ''' || a_max_value || '''';
    IF a_avg_value IS NOT NULL THEN
```

```
        a_sql_string := a_sql_string || ', AVG_VALUE = ' || a_avg_value ;
    END IF;
    IF a_std_value IS NOT NULL THEN
        a_sql_string := a_sql_string || ', STD_VALUE = ' || a_std_value ;
    END IF;
    a_sql_string := a_sql_string || ' WHERE COLUMN_NAME = ''' || p_column_name || '''';

    EXECUTE IMMEDIATE a_sql_string;
END;

PROCEDURE ANALYZE_COLUMNS(p_owner VARCHAR2, p_table_name VARCHAR2) IS
a_family VARCHAR2(30);
a_table_name VARCHAR2(30) := GET_DAS_TABLE_NAME(p_owner , p_table_name);
a_sql_string VARCHAR2(4000);
CURSOR a_cur IS
        SELECT COLUMN_NAME, DATA_TYPE
                FROM ALL_TAB_COLUMNS WHERE OWNER = p_owner AND TABLE_NAME =
p_table_name
                ORDER BY COLUMN_ID;
BEGIN
  FOR a_rec IN a_cur
  LOOP
       a_family := GET_DATA_FAMILY(a_rec.DATA_TYPE);
        IF a_family = c_data_family_date THEN
          ANALYZE_DATE_COLUMN(p_owner, p_table_name, a_rec.COLUMN_NAME);
        ELSIF a_family = c_data_family_numeric THEN
          ANALYZE_NUMERIC_COLUMN(p_owner, p_table_name, a_rec.COLUMN_NAME);
        ELSIF a family = c data family string THEN
          ANALYZE_STRING_COLUMN(p_owner, p_table_name, a_rec.COLUMN_NAME);
        ELSE
          a_sql_string := 'UPDATE TABLE ' || a_table_name || 'SET
                    VALUE_TYPE = ''' || c_data_family_unkown || ''', REVIEW = 1
                              WHERE COLUMN_NAME = ''' || a_rec.COLUMN_NAME || '''';
            EXECUTE IMMEDIATE a_sql_string;
            COMMIT;
        END IF;
  END LOOP;
END;
-- Privatni procedura analyzujici typ hodnoty
PROCEDURE ANALYZE_VALUE_TYPE(p_owner VARCHAR2, p_table_name VARCHAR2) IS
a table name VARCHAR2(30) := GET DAS TABLE NAME(p owner , p table name);
a_sql_string VARCHAR2(8000) := 'SELECT COLUMN_NAME, DATA_TYPE, ROW_COUNT,
DISTINCT_COUNT, DISTINCT_RATIO, NULL_COUNT FROM ' || a_table_name;
a_cur INTEGER;
a tmp INTEGER;
a_column_name VARCHAR2(30);
a data type VARCHAR2(30);
a_row_count NUMBER;
a_distinct_count NUMBER;
a_distinct_ratio NUMBER;
a null count NUMBER;
a with null NUMBER;
a_value_type VARCHAR2(10);
a_review NUMBER;
BEGIN
-- Iniciace kurzoru
    a_cur := DBMS_SQL.OPEN_CURSOR;
    DBMS SQL.PARSE(a_cur, a_sql_string, DBMS_SQL.NATIVE);
-- Definice sloupcu
    DBMS_SQL.DEFINE_COLUMN(a_cur, 1, a_column_name, 30);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 2, a_data_type, 30);
    DBMS SQL.DEFINE COLUMN(a cur, 3, a row count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 4, a_distinct_count);
    DBMS SQL.DEFINE COLUMN(a cur, 5, a distinct ratio);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 6, a_null_count);
-- Spusteni SQL
  a tmp := DBMS_SQL.EXECUTE(a_cur);
    LOOP
-- Nacteni dalsi radky
      IF DBMS_SQL.FETCH_ROWS(a_cur) > 0 THEN
-- Ziskani hodnot pro danou radku
```

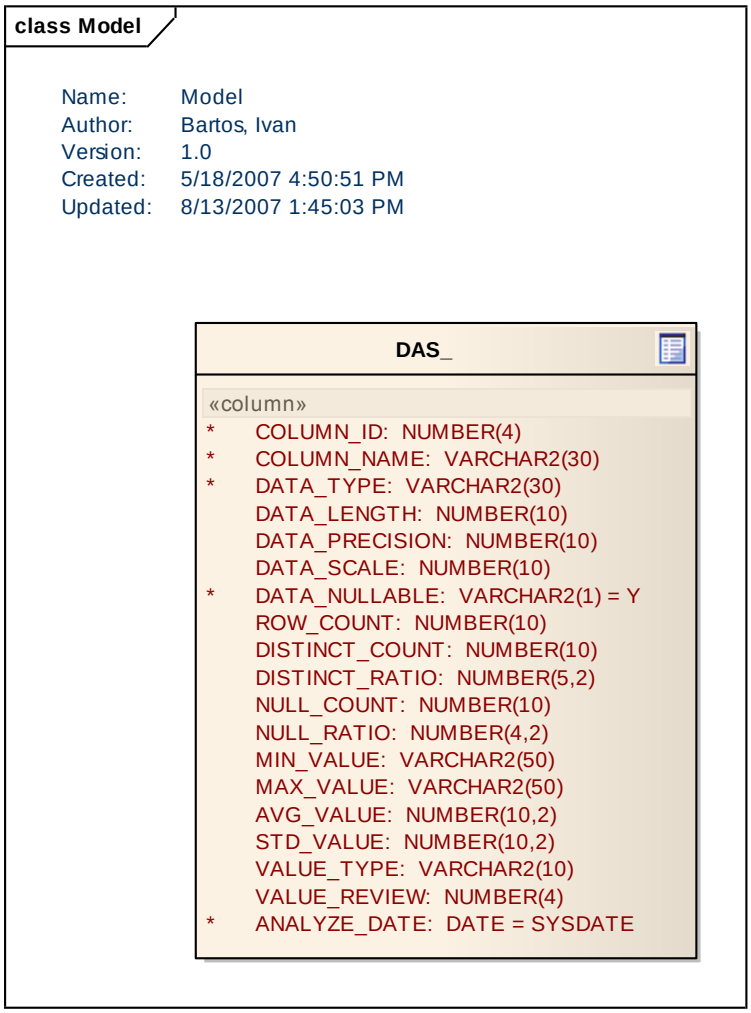
```
      DBMS_SQL.COLUMN_VALUE(a_cur, 1, a_column_name);
            DBMS_SQL.COLUMN_VALUE(a_cur, 2, a_data_type);
            DBMS_SQL.COLUMN_VALUE(a_cur, 3, a_row_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 4, a_distinct_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 5, a_distinct_ratio);
            DBMS_SQL.COLUMN_VALUE(a_cur, 6, a_null_count);
-- Definice typu hodnoty
      a_review := 0;
            a_with_null := 0;
            IF a_null_count > 0 THEN
              a with_null := 1;
            END IF;

      IF GET_DATA_FAMILY(a_data_type) = c_data_family_date THEN
              a_value_type := c_value_type_range;
            ELSIF (a_distinct_count - a_with_null) = a_row_count THEN
              a value type := c value type id;
                IF a_data_type = c_data_family_string THEN
                  a_review := 1;
                END IF;
            ELSIF (a_distinct_count - a_with_null) < 3 THEN
              a value type := c value type flag;
                IF a_distinct_count = 1 OR a_with_null = 1 THEN
                  a_review := 1;
                END IF;
            ELSIF a_distinct_count < 100 THEN
              a_value_type := c_value_type_set;
                IF a distinct ratio > 10 THEN
                  a_review := 1;
                END IF;
            ELSIF GET_DATA_FAMILY(a_data_type) = c_data_family_numeric THEN
              a_value_type := c_value_type_range;
                IF a_distinct_ratio < 0.01 THEN
                  a review := 1;
                END IF;
            ELSIF GET_DATA_FAMILY(a_data_type) = c_data_family_string THEN
              a_value_type := c_value_type_discrete;
                IF a_distinct_ratio < 0.01 THEN
                  a review := 1;
                END IF;
            ELSE
              a_value_type := c_value_type_unknown;
            END IF;
            a_sql_string := 'UPDATE ' || a_table_name || ' SET VALUE_TYPE = ''' ||
a value type || ''', VALUE_REVIEW = ' || a_review || ' WHERE COLUMN_NAME = ''' ||
a_column_name || '''';
            EXECUTE IMMEDIATE a_sql_string;
      ELSE
          EXIT;
        END IF;
    END LOOP;
  DBMS SQL.CLOSE_CURSOR(a_cur);
    COMMIT;
END;

FUNCTION PREPARE GRAPH XML(p_querry VARCHAR2) RETURN CLOB IS
a_ctx DBMS_XMLGEN.CTXHANDLE;
a graph data CLOB;
a_graph_xml CLOB;
a_erease INTEGER := 21;
BEGIN
  a ctx := DBMS XMLGEN.NEWCONTEXT(p querry);
  DBMS_XMLGEN.SETNULLHANDLING(a_ctx, 2);
  a graph data := '<values>';
    DBMS_LOB.APPEND(a_graph_data, DBMS_XMLGEN.GETXML(a_ctx));
    DBMS_XMLGEN.CLOSECONTEXT(a_ctx);
  DBMS LOB.APPEND(a graph data,'</values>');
    DBMS LOB.ERASE(a graph_data, a_erease, 9);
    RETURN a_graph_data;
END;
```

```
PROCEDURE PREPARE_GRAPH_DATA(p_owner VARCHAR2, p_table_name VARCHAR2) IS
a_das_table_name VARCHAR2(30) := GET_DAS_TABLE_NAME(p_owner , p_table_name);
a_cur INTEGER;
a_execution_result INTEGER;
a_sql_string VARCHAR2(4000) := 'SELECT COLUMN_NAME, DATA_TYPE, ROW_COUNT, VALUE_TYPE
FROM ' || a_das_table_name;
a_column_name VARCHAR2(30);
a_data_type VARCHAR2(30);
a_row_count NUMBER(10);
a_value_type VARCHAR2(10);
a_graph_data CLOB;
a_update_sql CLOB;
a_update_cur INTEGER;
BEGIN
  a_cur := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(a_cur, a_sql_string, DBMS_SQL.NATIVE);
-- Definice sloupcu
    DBMS_SQL.DEFINE_COLUMN(a_cur, 1, a_column_name, 30);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 2, a_data_type, 30);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 3, a_row_count);
    DBMS_SQL.DEFINE_COLUMN(a_cur, 4, a_value_type, 10);
-- Spusteni SQL
  a_execution_result := DBMS_SQL.EXECUTE(a_cur);
    LOOP
-- Nacteni dalsi radky
      IF DBMS_SQL.FETCH_ROWS(a_cur) > 0 THEN
-- Ziskani hodnot pro danou radku
      DBMS_SQL.COLUMN_VALUE(a_cur, 1, a_column_name);
            DBMS_SQL.COLUMN_VALUE(a_cur, 2, a_data_type);
            DBMS_SQL.COLUMN_VALUE(a_cur, 3, a_row_count);
            DBMS_SQL.COLUMN_VALUE(a_cur, 4, a_value_type);
-- Priprava SQL
      IF GET_DATA_FAMILY(a_data_type) = c_data_family_date THEN
              a_sql_string := 'SELECT TRUNC(' || a_column_name || ') AS NAME, COUNT(*)
AS VALUE, ROUND(COUNT(*)*100/' || a_row_count || ',4) AS PERCENTGE FROM ' ||
p_table_name || ' GROUP BY TRUNC(' || a_column_name || ')';
              DBMS_OUTPUT.PUT_LINE(a_sql_string);
            ELSIF GET_DATA_FAMILY(a_data_type) IN (c_data_family_string,
c_data_family_numeric) THEN
              a_sql_string := 'SELECT ' || a_column_name || ' AS NAME, COUNT(*) AS
VALUE, ROUND(COUNT(*) * 100/' || a_row_count || ', 4) AS PERCENTGE FROM ' ||
p_table_name || ' GROUP BY ' || a_column_name;
            END IF;
      IF a_value_type IN (c_value_type_flag, c_value_type_set, c_value_type_range) THEN
              a_graph_data := PREPARE_GRAPH_XML(a_sql_string);
                a_update_cur := DBMS_SQL.OPEN_CURSOR;
                a_update_sql := 'UPDATE ' || a_das_table_name || ' SET GRAPH_DATA =
:my_graph_data WHERE COLUMN_NAME = ''' || a_column_name || '''';
        DBMS_SQL.PARSE(a_update_cur, a_update_sql, DBMS_SQL.NATIVE);
                DBMS_SQL.BIND_VARIABLE(a_update_cur, ':my_graph_data', a_graph_data);
                 a_execution_result := DBMS_SQL.EXECUTE(a_update_cur);
                DBMS_SQL.CLOSE_CURSOR(a_update_cur);
                COMMIT;
            END IF;
      ELSE
          EXIT;
        END IF;
    END LOOP;
  DBMS_SQL.CLOSE_CURSOR(a_cur);
END;
-- Procedura ridici celkovou analyzu tabulky (vola vnitrni procedury a fce)
PROCEDURE ANALYZE_BASIC(p_owner VARCHAR2, p_table_name VARCHAR2) IS
BEGIN
    CREATE_TABLE_FOR_ANALYSIS(p_owner, p_table_name);
    ANALYZE_COLUMNS(p_owner, p_table_name);
    ANALYZE_VALUE_TYPE(p_owner , p_table_name);
END;
END PKG_TABLE_ANALYZE;
```

Příklad 6: Funkční Oracle package automatizace analýzy dat v tabulce dle Obr. 4 Požadavku na aplikaci provádějící automatickou profilaci dat

1.4 Způsoby uložení a výstupy analýzy dat

Standardním výstupem profilace dat je tabulka (nebo XML dokument), kterou aplikace vytváří, nebo ji plní výslednými metadaty o entitách a jejich atributech. Cílem našeho měření s využitím kombinace extrakce již existujících metadat a získání metadat vlastních, tak jak předchozí kapitoly popisují, je vytvoření a naplnění následující tabulky.



Obrázek 6: Schéma tabulky s výsledky profilace dat získaných kombinací extrakce data dictionary a profilací pomocí vlastní aplikace (Oracle custom package)

Podobné výstupy jsou vhodné pro další strojové zpracování. Pokud jsou navíc rozšířeny o dimenzi času, kdy jsou jednotlivé statistiky v podstatě verzovány a trvale uchovávány, stávají se vhodným nástrojem pro monitorování kvality dat v systému v dlouhodobém časovém horizontu (RUSSOM, 2007). Jako výstup, který je možný

předložit jinému oddělení (obchodní oddělení, oddělení **quality assurance**[14] atp.), jsou však zcela nevhodné (i případný XML dokument je třeba interpretovat v rámci nějaké další nadstavby) a nelze očekávat, že by další strana souhlasila s jejich užíváním. Proto je namístě vybudovat nad těmito zdroji metadat další, pro člověka srozumitelnější rozhraní (BOHUSLAV, 2006). Nejschůdnějším řešením je, v případě implementace vlastní datové profilace, vytvoření uživatelské šablony (template) na tvorbu dokumentů (reportů) a profilaci dat tak standardizovat, popřípadě automatizovat například pomocí maker či jiných scriptů. Nabízí se i využití nějakého komerčního nástroje pro reporting, nikoliv však klasické business inteligence aplikace. Typologie těchto "data quality" reportů charakteristice a účelu obchodních reportů neodpovídá. Jejich zaměření je interního charakteru.

[14] **Quality assurance (QA) -** se týká obecně všeho od návrhu, vývoje, nasazení, údržby až po dokumentaci produktu. Cílem této aktivity je dohlédnout, že výstupy z jednotlivých částí budou mít odpovídající kvalitu, která byla určena.

B&S_USERS

Analýza tabulky

Statistiky:

Počet řádků: 35,200,183
Průměrný počet vložených řádků: 8,802
Průměrný počet aktualizovaných řádků: 40,213

Sloupce:

1. ID_CODE_TEXT VARCHAR (36) NOT NULL

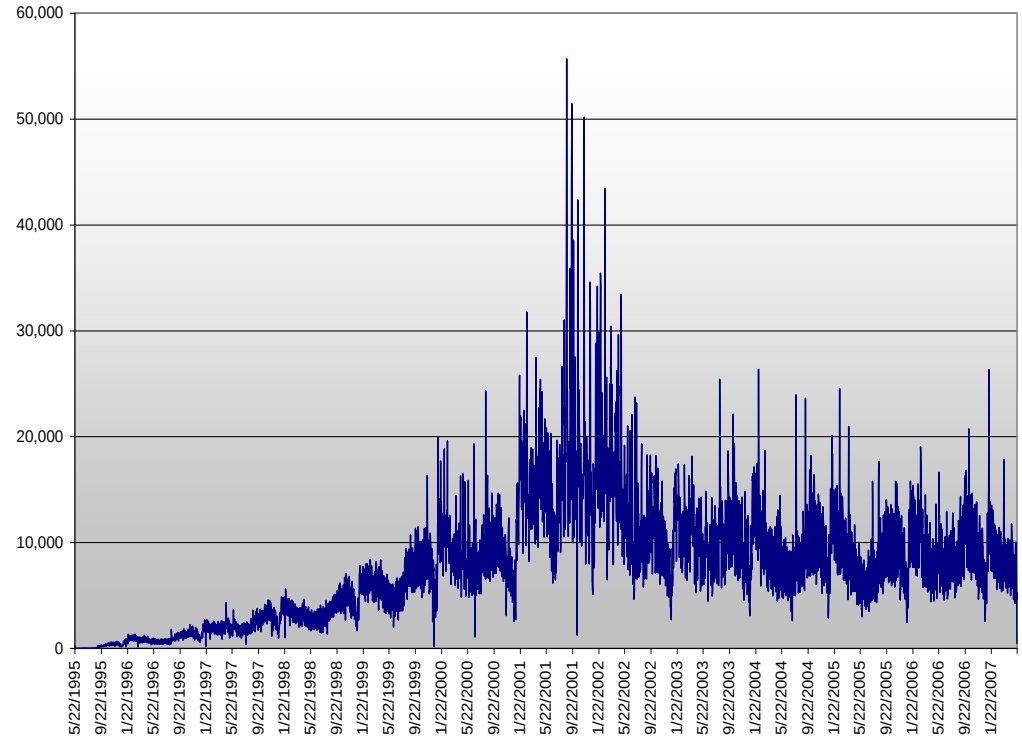
Popis
Primární klíč

Data
Typ: ID
Primární klíč

2. ENTER_DATE DATE

Popis
Datum vytvoření záznamu

Data
Typ: Date
Nulové hodnoty: 0 (0.00%)
Min hodnota: 5/22/1995 12:00:00 AM
Max hodnota: 5/21/2007 4:20:41 AM



Nekonzistence
Bulk operace.

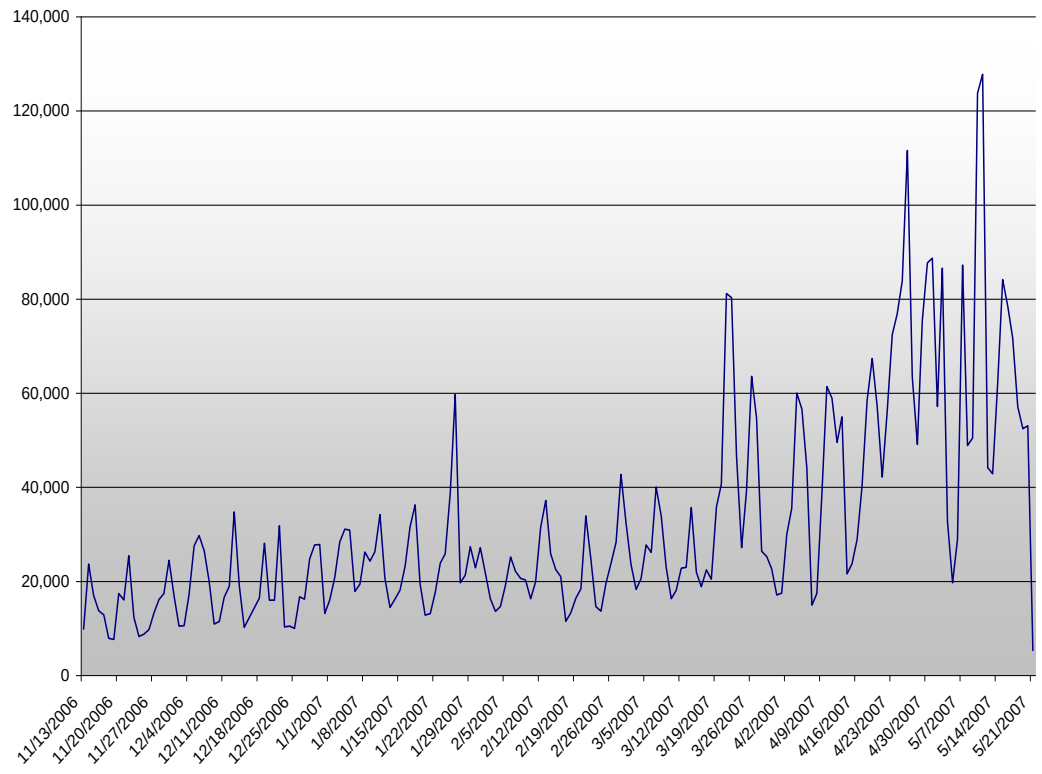
Komentář

Sloupec by měl být změněn na NOT NULL.

3. LAST_MODIFICATION_DATE DATE

Popis
Datum poslední změny řádku.
Tento sloupec je kalkulován z dalších datumových sloupců.
//TBD Marketing by měl specifikovat logiku výpočtu.

Data
Typ: Date
Nulové hodnoty: 29,149,794 (82.81%)
Min hodnota: 11/13/2006 2:10:27 PM
Max hodnota: 5/21/2007 4:29:52 AM



Nekonzistence
Nulové hodnoty.
Bulk operace.

Komentář
Nulové hodnoty se většinou vyskytují pouze pro starší záznamy. Předpokládáme, že logika výpočtu posledního data změny funguje pouze pro nejnovější data.

4. LAST_MAILBOX_CHECK_DATE DATE

Popis
Datum, kdy si uživatel otevřel naposledy svoji emailovou schránku.

Data
Typ: Flag
Nulové hodnoty: 35,200,183 (100%)
Min hodnota: N/A
Max hodnota: N/A

Nekonzistence

Pouze nulové hodnoty.
Komentář Atribut nebyl nikdy využit.

Příklad 7: Ukázka standardizovaného výstupu profilace dat určeného pro obchodní oddělení

1.5 Možnosti automatické validace dat pomocí REGEX[15]

Předchozí kapitoly popisují validaci dat na základě pravidel identifikovaných ve zdrojovém systému. Z hlediska analýzy dat nebo statistické analýzy je takový postup postačující. V rámci validace dat lze ovšem aplikovat i jiné přístupy. Jedním z možných řešení může být použití již definovaných pravidel, tj. takových, která vznikla v nějakém předchozím procesu profilace, anebo jiných vnějších validačních pravidel.

Není jednoduché zvolit vhodný nástroj, kterým lze vstupní data validovat. Nejvhodnějším řešením je využití funkcionality, která je podporována cílovým systémem. Ačkoliv je v definici pravidel preferováno spíše řešení obecné, tedy takové, které není vázáno na konkrétní platformu, existují metody, jejichž zápis je standardizovaný, jednoznačně interpretovatelný a zároveň podporovaný většinou současných systémů. Jednou z hojně využívaných možností je využití regulárních výrazů - Regular Expressions (dále RegEx). Tyto jsou přístupné ve velké škále aplikací a jsou podporovány jak operačními systémy, tak databázovými nástroji např. při validaci vstupních dat formulářů HTML, v systému UNIX, či v databázích Oracle, MS SQL, apod.

```
^((?>[a-zA-Z\d!#$%&'*+\-/=?^_`{|}~]+\x20*|"((?=[\x01-\x7f])[^"\\]|\\[\x01-
\x7f])*"\x20*)*(?<angle><))?((?!\.)(?>\.?[a-zA-Z\d!#$%&'*+\-/=?^_`{|}~]+)+|"((?=[\x01-
\x7f])[^"\\]|\\[\x01-\x7f])*")@(((?!-)[a-zA-Z\d\-]+(?<!-)\.)+[a-zA-
Z]{2,}|\[(((?(?<!\[)\.)(25[0-5]|2[0-4]\d|[01]?\d?\d)){4}|[a-zA-Z\d\-]*[a-zA-
Z\d]:((?=[\x01-\x7f])[^\\\[\]]|\\[\x01-\x7f])+)\])(?(angle)>)$
```

Příklad 8: RegEx pro validaci emailové adresy (RFC 2822 mailbox). (Regular Expression Library, 2008)

```
^(((((((((jan(uary)?)|(mar(ch)?)|(may)|(july?)|(aug(ust)?)|(oct(ober)?)|(dec(ember)?))
((3[01])|29))|(((apr(il)?)|(june?)|(sep(tember)?)|(nov(ember)?))
((30)|(29)))|(((jan(uary)?)|(feb(ruary)?|(mar(ch)?)|(apr(il)?)|(may)|(june?)|(july?)|(a
ug(ust)?)|(sep(tember)?)|(oct(ober)?)|(nov(ember)?)|(dec(ember)?))) (2[0-
8]|(1\d)|(0?[1-9])))),?|(((((1[02])|(0?[13578]))[\.\-
/]((3[01])|29))|(((11)|(0?[469]))[\.\-/]((30)|(29)))|(((1[0-2])|(0?[1-9]))[\.\-/](2[0-
8]|(1\d)|(0?[1-9])))[\.\-/])|(((((3[01])|29)[\-
\./]((jan(uary)?)|(mar(ch)?)|(may)|(july?)|(aug(ust)?)|(oct(ober)?)|(dec(ember)?)))|(((
30)|(29))[\.\-/]((apr(il)?)|(june?)|(sep(tember)?)|(nov(ember)?)))|((2[0-
8]|(1\d)|(0?[1-9]))[\.\-
/]((jan(uary)?)|(feb(ruary)?|(mar(ch)?)|(apr(il)?)|(may)|(june?)|(july?)|(aug(ust)?)|(s
ep(tember)?)|(oct(ober)?)|(nov(ember)?)|(dec(ember)?))))))[\-
\./])|((((3[01])|29)((jan)|(mar)|(may)|(jul)|(aug)|(oct)|(dec)))|(((30)|(29))((apr)|(ju
n)|(sep)|(nov)))|((2[0-8]|(1\d)|(0[1-
9]))((jan)|(feb)|(mar)|(apr)|(may)|(jun)|(jul)|(aug)|(sep)|(oct)|(nov)|(dec)))))(((175[
3-9])|(17[6-9]\d)|(1[89]\d{2})|[2-9]\d{3})|\d{2}))|((((175[3-9])|(17[6-
9]\d)|(1[89]\d{2})|[2-
```

[15] **Regular Expressions** - speciální řetězce znaků, které představují určitý vzor (masku) pro textové řetězce. Užívané v C#, Java, Visual Basic .NET, Perl, PHP, Javascript, Unix/Linux, SQL, atd.

```
9]\d{3})|\d{2})((((1[02])|(0[13578]))((3[01])|29))|(((11)|(0[469]))((30)|(29)))|(((1[0-
2])|(0[1-9]))(2[0-8]|(1\d)|(0[1-9])))))|(((29feb)|(29[ \.\-/]feb(ruary)?[ \.\-
/])|(feb(ruary)?29,??)|(0?2[\.\-/]29[\.\-
/]))((((([2468][048])|([3579][26]))00)|(17((56)|([68][048])|([79][26]))))|(((1[89])|([2-
9]\d))(([2468][048])|([13579][26])|(0[48])))))|(((02468][048])|([13579][26])))))|(((((([2
468][048])|([3579][26]))00)|(17((56)|([68][048])|([79][26]))))|(((1[89])|([2-
9]\d))(([2468][048])|([13579][26])|(0[48])))))|(((02468][048])|([13579][26])))(0229)))$
```

Příklad 9: RegEx pro validaci datumů pro MS SQL Server 2005 validující i přestupný rok (Regular Expression Library, 2008)

Konečná implementace validace vstupních dat pomocí ReGex je většinou založena na následujícím algoritmu.

Vstupní data –> funkce obsahující regulární výraz (parametr Vstupní data) –> hodnota TRUE či FALSE

Validace dat může být ovšem daleko sofistikovanější. Při validaci lze například automaticky identifikovat typ chyby, která se ve vstupních datech objevuje. Na jejím základě je pak možné určitou skupinu nevalidních dat například dávkově upravit. Podívejme se na možný příklad validace emailové adresy uživatele. Na počátku identifikujme pravidla pro platné emailové adresy. Tato pravidla se ovšem týkají pouze správnosti syntaxe emailové adresy (v našem případě to znamená, že emailová adresa odpovídá normě RFC 822) a správnosti sémantiky (např. doména adresy patří mezi platné existující domény). Její fyzickou neplatnost tzn. neexistenci lze zjistit pouze zasláním emailové zprávy a následným vyhodnocení úspěšnosti jejího přijetí.

Pro zjednodušení lze uvažovat následující pravidla vycházející z definice RFC 822, která předpokládá sémantiku text@text.text, což jsou:

- Adresa je vyplněná
- Adresa má odpovídající délku (6 bitů až 256 bitů)
- Adresa obsahuje právě jeden znak @
- Adresa neobsahuje neplatné znaky
- Adresa náleží platné doméně (toto může být vzhledem k uvolnění nových obecných domén budoucnu značně problematické: @text.jakýolivzaplacenýtext)

Jednotlivá pravidla lze shrnout do následujících výstupních hodnot zamýšlené funkce:

- Adresa je validní
- Adresa je prázdná
- Formát adresy je nevalidní
- Doména adresy je nevalidní

Výsledná funkce, resp. Oracle balíček obsahující validační funkci, může vypadat například takto:

```
CREATE OR REPLACE PACKAGE PKG_EMAIL_ADDRESS IS
/******************************************************************************
PURPOSE: Package pro manipulaci s emailovou adresou
REVISIONS:
      Ver        Datum        Autor            Popis
      ---------  ----------  ---------------   ------------------------------------
      1.0        16.8.2007   Ivan Bartos       1. Created
/******************************************************************************/
FUNCTION VALIDATE_EMAIL_ADDRESS (p_email_address in VARCHAR2) RETURN NUMBER;
END PKG_EMAIL_ADDRESS;
```

```
CREATE OR REPLACE PACKAGE BODY PKG_EMAIL_ADDRESS_VALIDATE IS
FUNCTION VALIDATE_EMAIL_ADDRESS (p_email_address IN VARCHAR2) RETURN NUMBER IS
/******************************************************************************
  NAME:     VALIDATE_EMAIL_ADDRESS
  PURPOSE: Validuje emailovou adresu
           Pouziva kombinaci RegEx a String funkci Oracle
******************************************************************************/
-- Definice atributu
  a_global_domains constant VARCHAR2(95) :=
'cat|com|edu|gov|jobs|mil|mobi|net|org|int|biz|info|name|pro|aero|coop|museum|arpa|job
s|cat|mobi';
  a_country_domains constant VARCHAR2(743) :=
'ac|ad|ae|af|ag|ai|al|am|an|ao|aq|ar|as|at|au|aw|az|ba|bb|bd|be|bf|bg|bh|bi|bj|bm
|bn|bo|br|bs|bt|bv|bw|by|bz|ca|cc|cd|cf|cg|ch|ci|ck|cl|cm|cn|co|cr|cu|cv|cx|cy|cz|de|dj
|dk|dm|do|dz|ec|ee|eg|eh|er|es|et|fi|fj|fk|fm|fo|fr|ga|gd|ge
|gf|gg|gh|gi|gl|gm|gn|gp|gq|gr|gs|gt|gu|gw|gy|hk|hm|hn|hr|ht|hu|id|ie|il|im|in|io|iq|ir
|is|it|je|jm|jo|jp|ke|kg|kh|ki|km|kn|kp|kr|kw|ky|kz|la|lb|lc
|li|lk|lr|ls|lt|lu|lv|ly|ma|mc|md|mg|mh|mk|ml|mm|mn|mo|mp|mq|mr|ms|mt|mu|mv|mw|mx|my|mz
|na|nc|ne|nf|ng|ni|nl|no|np|nr|nu|nz|om|pa|pe|pf|pg|ph|pk|pl|pm|pn|pr|ps|pt|pw|py|qa|re
|ro|ru|rw|sa|sb|sc|sd|se|sg|sh|si|sj|sk|sl|sm|sn|so|sr|st|sv|sy|sz|tc|td|tf|tg|th|tj|tk
|tm|tn|to|tp|tr|tt|tv|tw|tz|ua
|ug|uk|um|us|uy|uz|va|vc|ve|vg|vi|vn|vu|wf|ws|ye|yt|yu|za|zm|zw|ax|cs|eu|gb|tl';
  a_email_address VARCHAR2(255);
  a_address_part VARCHAR2(255);
  a_domain_part VARCHAR2(255);
  a_tld VARCHAR2(255);
  a_chr_position INT;

  a_id_valid constant INT := 1; -- validni adresa
  a_id_empty constant INT := 41; -- prazdna adresa
  a_id_invalid_address_format INT := 42; -- chybny format adresy
  a_id_invalid_domain_name INT := 43; -- neplatna, neexistujici domena

BEGIN
-- Extrakce adresne a domenove casti emailove adresy
  a_address_part := SUBSTR(p_email_address, 1, INSTR(p_email_address, '@') - 1);
  a_domain_part := SUBSTR(p_email_address, INSTR(p_email_address, '@') + 1);
-- Veskera validace bude probihat nad adresou psanou malym pismem
-- (nasledne cisteni pak adresu prevede do formatu lower)
```

```
  a_email_address := LOWER(TRIM(p_email_address));
-- RFC822 text@text.text validace -- fce vrati 42
  IF REGEXP_INSTR(a_email_address, '<[[:graph:]]+@[[:graph:]]+>') > 0 THEN
     RETURN a_id_invalid_address_format;
  END IF;
-- Pokud je adresa prazdna -- fce vrati 41
  IF nvl(LENGTH(a_email_address),0) = 0 THEN
    RETURN a_id_empty;
  END IF;
-- Validace delky adresy, pokud je adresa prilis kratka ci dlouha -- fce vrati 42
  IF nvl(LENGTHB(a email address),0) > 255 OR nvl(LENGTH(a_email_address),0) < 6 THEN
    RETURN a_id_invalid_address_format;
  END IF;
--  Pokud adresa neobsahuje zadny znak @ nebo obsahuje 2@ - fce vrati 42
  IF INSTR(a_email_address, '@', 1, 1) = 0 OR INSTR(a_email_address, '@', 1, 2) > 0
THEN
    RETURN a_id_invalid_address_format;
  END IF;
-- Pokud je cast adresy null nebo cast domeny mensi nez 4 znaky - fce vrati 42
-- Obsahuje zaroven kontrolu na vyskzy povolenzch znaku
  IF NOT REGEXP_LIKE(NVL(a_address_part,'X'),'^[a-z0-9!#$%*/?|^{}`~&''+=_.-]+$') THEN
    RETURN a_id_invalid_address_format;
  END IF;
-- Kontrola domenove casti
-- Kontorluje zda  domena obsahuje pouze validni znaly a ma nejmene 2 domenove urovne
(tld + 1 sub level domenu)
  IF NOT REGEXP_LIKE('.'||a_domain_part,'^(\.[a-z0-9]([-a-z0-9]*[a-z0-9]|[a-z0-
9]*)){2,}$') THEN
    RETURN a_id_invalid_domain_name;
  END IF;
-- Overi platnost TLD
  a_tld := SUBSTR(a_domain_part, INSTR(a_domain_part, '.', -1 , 1) + 1);
  a_chr_position := 0;
  a chr position := SIGN(INSTR(a global domains,'|'||a_tld||'|') +
INSTR(a country domains,'|'||a_tld||'|'));
  IF a_chr_position != 1 THEN
    RETURN a_id_invalid_domain_name;
  END IF;
-- V jakemkoliv jinem pripade predpokladejme adresu jako validni
    RETURN a_id_valid;
END VALIDATE EMAIL ADDRESS;
END PKG_EMAIL_ADDRESS_VALIDATE;
```

Příklad 10: Funkční Oracle package automatizace validace vstupní hodnoty Emailová Adresa

Pokud bychom chtěli být ještě preciznější, lze zvažovat rozšíření původních pravidel například o tato:

- Adresa není ohraničena dalšími znaky typu (), <>,‘‘,“‘‘
- Adresa nezačíná ani nekončí neplatným znakem (.)
- Adresa neobsahuje dva zakázané znaky v sérii (..)

Zajímavým příkladem by pak mohla být validace IP typu domény.

```
…
IF NOT REGEXP_LIKE(a domain part, '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$') THEN
RETURN a_id_invalid_domain_name;
END IF;
…
```

Příklad 11: Rozšíření validace o typ domény

Na základě validace dat lze zvolit následující postupy:

1. Data odpovídají pravidlům – **přebrat**
2. Data neodpovídají pravidlům a nelze je automaticky čistit - **nepřebírat**
3. Data neodpovídají pravidlům, a lze je opravit - **aplikovat jejich čištění**

Z hlediska výkonu automatizovaného systému může být žádoucí provádět validaci a případné čištění (cleansing) v rámci jednoho úkonu a to následujícím způsobem:

1. Nejdříve jsou určena pravidla a identifikovány možné chyby.
2. Je definován způsob opravy možných chyb.
3. Data jsou čištěna (opravena).

```
--  Pokud je emailova adresa ohranicena zavorkami, odstran je
IF SUBSTR(a email address, 1, 1) = '(' AND SUBSTR(a_email_address,
LENGTH(a_email_address), 1) = ')' THEN
       a_email_address := SUBSTR(a_email_address, 2, LENGTH(a_email_address) - 2);
END IF;
```

4. Data jsou ověřována podle platných pravidel.

```
-- Kontroluje, zda  domena obsahuje pouze validni znaky a ma nejmene 2 domenove urovne
(tld + 1 sub level domenu)
a domain part := SUBSTR(a email address, INSTR(p email address, '@') + 1);
IF NOT regexp like('.'||a_domain_part,'^(\.[a-z0-9]([-a-z0-9]*[a-z0-9]|[a-z0-
9]*)){2,}$') THEN
       RETURN empty_email_address; -- Vynuluje emailovou adresu
END IF;
```

5. Výstupem tohoto procesu může být původní hodnota atributu Emailová Adresa, nová hodnota a dodatečná informace o její validitě.

EMAIL_ADDRESS_OLD	EMAIL_ADDRESS	VALIDITY
(manager@alfaco.com)	manager@alfaco.com	1
(clerkr@alfaco.c.m)	-	42
	-	41

6. Finální report, obohacený o výsledky validace, pak umožňuje hlouběji porozumět datům uložených v analyzovaném systému.

6. EmailAddress VARCHAR (100)	
Popis Emailová adresa uživatele	
Data Typ: Discrete Min délka: 1 Max délka: 100 Unikátní hodnoty: 93,376,469 (95.34%) Nulové hodnoty: 6,829,415 (6.52%)	**Hodnoty** Validní: 71,500,234 (76.57%) Prázdná: 6,829,415 (6.52%) Nevalidní formát: 7,720,256 (8.26%) Nevalidní doména: 7,326,564 (7.84%)

Shrnutí

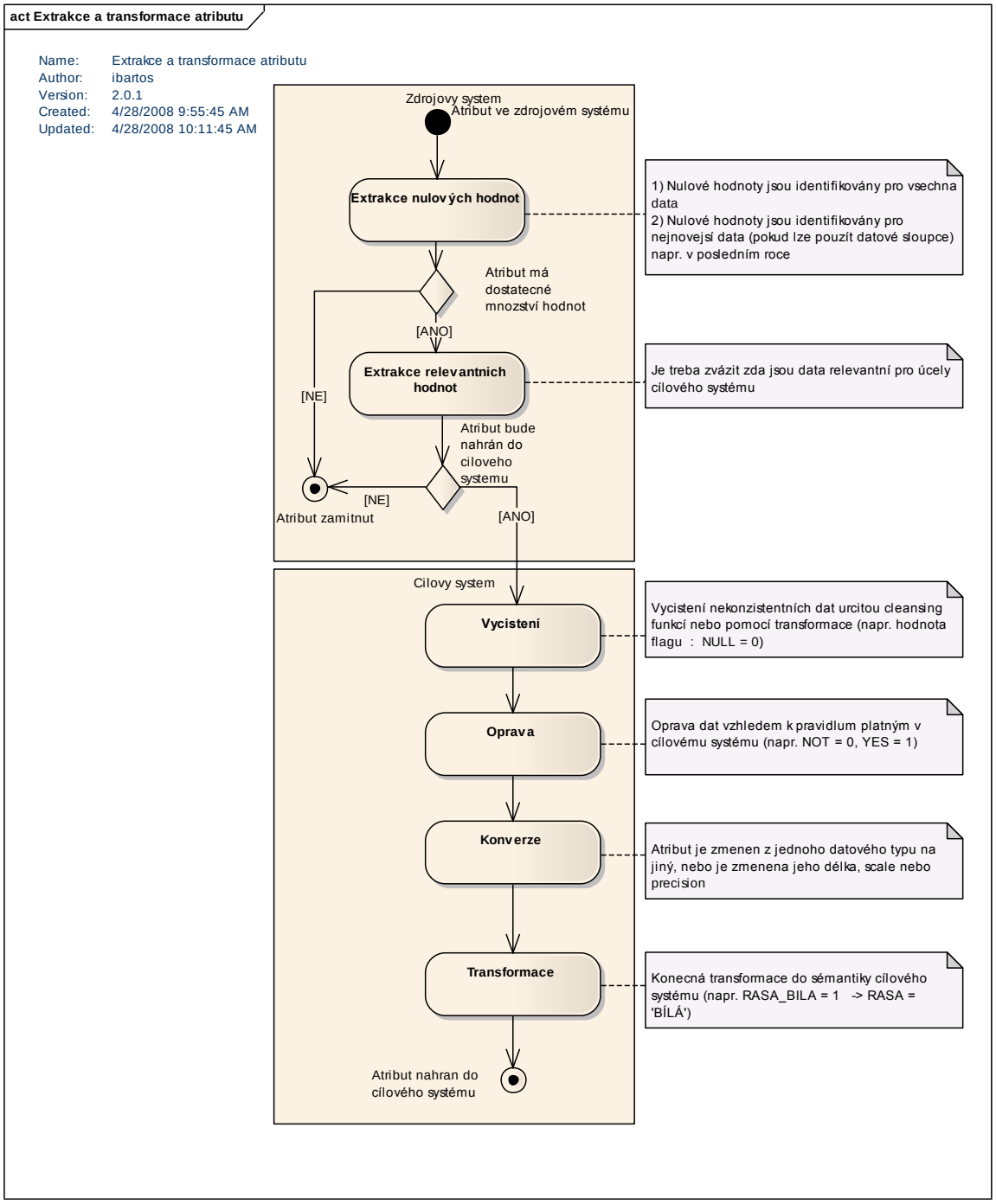
Jak ukazují průzkumy (ABERDEEN GROUP, 2005; ADELMAN, 2005), je pravdou, že se požadavky na rychlý vývoj systémů negativně odráží na výsledné kvalitě dat, se kterými pracují. Právě proto, by měl být na kvalitu dat kladen větší důraz. Pouze standardizované a iterativní procesy umožňují tuto kvalitu zajistit a udržet. Pomocí důkladné analýzy dat a následného zásahu jak do dat samotných, tak i do procesů, která je vytvářejí, lze i nekonzistentní a nečitelný systém postupně vyčistit a případně integrovat do systému jiného.

Ačkoliv se zdá být nelepším řešením koupě či převzetí již existujícího nástroje pro datovou profilaci, naráží tento postup na řadu, zejména finančních problémů. I přes časovou náročnost, která se však zcela odvíjí od požadované hloubky analýzy dat a složitosti řešeného problému (integrace, synchronizace, atp.), je vytvoření vlastního efektivního nástroje vhodným a spolehlivým řešením, zejména z důvodu možnosti jeho integrace do již existujících aplikací a procesů domovské organizace a možností jeho přizpůsobitelnosti aktuálním potřebám.

V této kapitole byla popsána jednoduchá metodologie profilace a validace dat, kterou lze ve většině případů velmi lehce aplikovat. Pokud budeme vycházet z předchozí analýzy požadavků na profilovaný nástroj a zvážíme-li všechny situace, které mohou v rámci ukládání informací o entitách nastat (viz. příklady aktivity diagramů v Obr. 4 a detailněji v Obr. 5), zbývá vyřešit pouze hloubku (detail) takové analýzy a konkrétní jazyk, ve kterém bude vytvořena aplikace pro zvolené prostředí. Tato expertní znalost je v případě volby tvorby vlastního nástroje přepokládána. Následnou formu prezentace zjištěných skutečností bohužel nelze explicitně standardizovat, protože je zcela odvislá od zvyklostí v určité firmě/instituci, nebo musí být výsledkem konsenzu jednotlivých oddělení, které s problematikou kvality dat přicházejí do styku.

2. Transformace informací ze zdrojového do cílového systému

Předpokládejme nyní, že již máme dostatečné povědomí o datech, která jsou uložena ve zdrojovém systému. Na základě profilace dat v kombinaci se sémantickou analýzou názvu sloupců a potvrzení významu atributů je možné určit, co která data reprezentují. Data jsou klasifikována a mění se v informaci. Na takto klasifikovaná data se aplikují další metodologie resp. techniky, které je "přetransformují" do informačního konceptu a datové struktury cílového systému. Jednotlivé metody mohou být v rámci různých pojetí charakterizovány a děleny jinak, než je uvedeno v této práci, nebo se mohou vzájemně překrývat. Přesto jsou popisované kroky pro transformaci dat mezi dvěma systémy v různých implementacích víceméně totožné (tvoří ucelený blok).



Obrázek 7: Metodologie přenosu dat mezi zdrojovým a cílovým systémem

Ačkoliv v tuto chvíli ještě nelze hovořit o sémantickém modelu mapování entit a jejich atributů, můžeme ze znalostí obou systémů určit, které entity resp. jejich atributy uchovávají totožnou informaci. Toto mapování lze zobrazit pomocí obyčejné tabulky.

Atribut p přebrán beze změny
Neexistuje odpovídající atribut
Více atributů odpovídá jednomu

Zdrojová tabulka Beta&Sons	Zdrojový sloupec	Cílová tabulka Alfa co.	Cílový sloupec
B&S_USERS	FIRST_NAME_TXT	DM_CUSTOMERS	FIRST_NAME
B&S_USERS	LAST_NAME_TXT	DM_CUSTOMERS	LAST_NAME
B&S_USERS	USER_EMAIL	DM_CUSTOMERS	EMAIL_ADDRESS
B&S_USERS	US_CITIZEN_FLAG	DM_CUSTOMERS	CITIZENSHIP
B&S_USERS	RACE_WHITE_TXT	DM_CUSTOMERS	RACE
B&S_USERS	RACE_BLACK_TXT	DM_CUSTOMERS	RACE
B&S_USERS	RACE_INDIAN_TXT	DM_CUSTOMERS	RACE
B&S_USERS	RACE_HISPANIC_TXT	DM_CUSTOMERS	RACE
B&S_USERS	RACE_ASIAN_TXT	DM_CUSTOMERS	RACE
NOT USED		DM_CUSTOMERS	STEPS_COMPLETED_CS
NOT USED		DM_CUSTOMERS	STATUS

Příklad 11 : Lineární zobrazení odpovídajících si entit a jejich atributů

2.1 Extrakce (Extraction)

První krok, respektive první techniku, která je aplikována na data, lze označit jako extrakci. Tuto extrakci lze rozložit do dvou kroků.

Prvním z nich je extrakce dat na základě četnosti nulových hodnot jednotlivých atributů. Atributy s vysokým obsahem nulových hodnot (hodnota pro ně není specifikována), nebo atributy obsahující pouze nulové hodnoty (pokud tato hodnota není později identifikována jako zápor), nemají prakticky žádnou informační hodnotu, a jako takové nejsou pro přenos do cílového systému žádoucí. Zde je důležité uvědomit si závislost dat na změnách v čase. Jednotlivé systémy se neustále vyvíjí na základě měnících se business požadavků (**BR**[16]) a pravidel a souvisejících změn na úrovni samotné aplikace či databáze. Atributy entit mohou v rámci systému zastarávat, mohou být vytvořeny za účelem budoucího možného využití, nebo může být od jejich využívání dočasně upuštěno a záhy mohou být znovu iniciovány. Z tohoto důvodu je potřebné provádět měření nulových

[16] **Business Requirements -** Požadavky na realizaci nějakého nového procesu formulované netechnickými odděleními firmy směrem k realizačnímu týmu (podobně existují též požadavky funkční (tzv. Functional Requirements) či technické (Technical Requirements). Tyto vždy specifikují nějakou potřebu, která je následně v projektu realizována např. nákupem nějakého hardware, jeho konfigurací, či vývojem nebo úpravou aplikací).

hodnot jak na kompletní množině dat, tak i v dílčích časových úsecích. Nejvhodnějšími dalšími vzorky jsou aktuální data např. za poslední rok, nebo četnost obsahu nulových hodnot v poměru k počtu záznamů v sekvencích jednotlivých měsíců.

Atribut bude použit
Atribut nebude použit

B&S_USERS	Celkový počet nulových hodnot	Počet nulových hodnot- rok 2011 (904,277)
FIRST_NAME_TXT	0.00%	0.00%
LAST_NAME_TXT	0.00%	0.00%
EMAIL	2.21%	0.00%
LAST_MAILBOX_CHECK_DATE	13.67%	5.15%
LAST_UPDATE_PROFILE_DATE	16.47%	0.00%
LAST_UPDATE_OTHER_DATE	98.30%	100.00%
MINIREG_FLAG	100.00%	100.00%
MINIREG_ENTER_DATE	100.00%	100.00%
STEPS_COMPLETED_TXT	0.00%	0.00%
TOTAL_MAILBOX_CHECKS_NUM	9.22%	5.15%
COUNTED_FLAG	82.55%	100.00%
LCOUNTED_FLAG	82.55%	100.00%

Příklad 12: Extrakce atributů na základě nulových hodnot v celkové množině dat a ve vzorku aktuálních dat z posledního roku

Druhá úroveň extrakce je označována jako extrakce hodnot. Zde je třeba určit hodnotu relevance extrahovaných dat resp. informací v nich uložených vzhledem k cílovému systému.

Atribut bude použit
Atribut nebude použit

B&S_USERS	Obsah atributů
LAST_MODIFICATION_DATE	Poslední datum změny
SEARCH_COMPLETE_FLAG	Interní flag zdrojového systému
SEARCH_STATUS_FLAG	Interní flag zdrojového systému
PREVIOUS_MAILBOX_CHECK_DATE	Datum, kdy byla naposledy zkontrolována emailová schránka
TOTAL_MAILBOX_CHECKS_NUM	Celkový počet navštívení emailové schránky
COUNTED_FLAG	Interní flag zdrojového systému
LCOUNTED_FLAG	Interní flag zdrojového systému
FIRST_ OFFER_CHECK_DATE	Datum, kdy byla poprvé prohledána nabídka periodik
CITY_PERM	Nemá uplatnění v cílovém systému
ADDR_TYPE_PERM	Nemá uplatnění v cílovém systému
FOREIGN_COUNTRY_CURRENT	Země původu
FOREIGN_COUNTRY_PERM	Nemá uplatnění v cílovém systému
PHONE_HOME	Nemá uplatnění v cílovém systému (V cílovém systému je vyžadován pouze mobilní telefon)

PHONE_MOBILE_TXT	Mobilní telefon uživatele

Příklad 13: Extrakce atributů na základě hodnot a jejich uplatnění v cílovém systému

2.2 Čištění (Cleansing)

Extrahovaná data musí projít procesem čištění. Toto čištění je specifické vzhledem ke standardům cílového systému. Obecně lze hovořit o následujících algoritmech:

- **rozpoznání** (parsing) obsahu datových položek, opravy datových položek (odstranění překlepů, nesprávných zápisů, formátů apod.),
- **standardizace** – převod datových položek na jednotný formát, který je pak možno použít pro porovnání s rejstříky a číselníky a pro porovnání hodnot datových položek spravovaných různými systémy,
- **obohacení** – doplnění chybějících položek, pokud je to možné (např. chybějících částí adresy),
- **unifikace** – určení všech záznamů, které představují jeden konkrétní subjekt (např. nalezení a jednoznačné označení všech evidovaných záznamů o konkrétní osobě, adrese),
- **deduplikace** – výběr nejlepšího záznamu, který bude nadále reprezentovat konkrétní subjekt,
- **identifikace –** pro nové datové záznamy – určení konkrétního subjektu (například osoby), ke kterému záznam patří (KYJONKA, 2006).

Algoritmy čištění mohou být součástí existující aplikace nebo, a to je vzhledem k různým standardům systémů častější příklad, jsou specifikovány v rámci tzv. cleansing funkcí, jejichž vstupním parametrem je zvolený atribut a výstupem pak již vyčištěný prvek. Tyto funkce jsou následně odkazovány v kódu aplikace. Jejich výhodou je možnost opakovaného použití například při integraci více různých modulů jednoho systému obsahujících stejnou či podobnou informaci, nebo, jsou-li navrženy dostatečně univerzálně, i při možné integraci dalšího zdroje.

Atribut k čištění
Nevyžaduje čištění

B&S_USERS	CLEANSING
FIRST_NAME_TXT	funkce CLEAN_FIRST_NAME
LAST_NAME_TXT	funkce CLEAN_LAST_NAME
EMAIL	funkce CLEAN_EMAIL_ADDRESS
DIM_DOMAIN_ID	Jedná se o vnitřní klíč dimenze domén
PHONE_MOBILE_TXT	funkce CLEAN_PHONE_NUMBER
ADDRESS1_CURRENT	funkce CLEAN _ADDRESS 1
ADDRESS2_CURRENT	funkce CLEAN _ADDRESS 2
ADDR_TYPE_CURRENT	Přebíráno tak, jak je. Neexistuje specifická funkce pro čištění
CITY_CURRENT	funkce CLEAN _CITY
FOREIGN_COUNTRY_CURRENT	Porovnání vůči číselníku existujících zemí
STATE_CURRENT	Neexistuje specifická funkce pro čištění
ZIP_CURRENT	funkce CLEAN_POSTAL_CODE
US_CITIZEN_FLAG	Pouze hodnoty 'Y' a 'N'
WANT_TO_RECIEVE_EMAIL	NULL = N

Příklad 14: Čištění atributů

```
FUNCTION CLEAN_PHONE_NUMBER(PHONE_NUMBER VARCHAR2)
    RETURN VARCHAR2 IS
g_null_str VARCHAR2(1) := '';
a_strtemp VARCHAR2(256);
BEGIN
a_strtemp := PHONE_NUMBER;
-- Odstrani mezeru
a_strtemp := REPLACE(a_strtemp, ' ', '');
-- Pokud telefonni cislo zacina "+", zmeni znak na "00"
IF SUBSTR(a_strtemp, 1, 1) = '+' THEN
a_strtemp := CONCAT('00', SUBSTR(a_strtemp, 2));
END IF;
-- Odstrani veskere nenumericke znaky
a_strtemp := TRANSLATE(a_strtemp, CHR(1) || TRANSLATE(a_strtemp, CHR(1) ||
'1234567890', CHR(1)), CHR(1));
-- Pokud je delka PHONE_NUMBER mensi nez 7, vynuluj
-- Pokud je delka PHONE_NUMBER větsi nez 15, vynuluj
IF LENGTH(a strtemp) < 7 OR LENGTH(a_strtemp) > 15 THEN
  a_strtemp := g_null_str;
END IF;
-- Pokud jsou prvni 3 nebo vice znaku 000, vynuluj
IF SUBSTR(a strtemp, 1, 3) = '000' THEN
  a_strtemp := g_null_str;
END IF;
-- Pokud jsou prvni 2 znaky "00", nahrad "+"
IF SUBSTR(a_strtemp, 1, 2) = '00' THEN
  a strtemp := CONCAT('+', SUBSTR(a_strtemp, 3));
END IF;
RETURN SUBSTR(a_strtemp, 1, 50);
END;
```

Příklad 15: PL SQL funkce pro čištění telefonního čísla (Oracle)

2.3 Oprava (Correction)

Opravou chápeme úpravu atributu dle pravidel platných v cílovém systému. Jedná se například o to, aby byly hodnoty stejného významu označovány konzistentně. Například hodnota atributu typu flag je v cílovém systému vždy vyjádřena jako 0 a 1, nikoliv nekonzistentně, někdy 1, 0, jindy "T", "F" a podobně.

Atribut vyžaduje opravu
Nevyžaduje opravy

B&S_USERS	Předchozí CLEANSING	Oprava
WANT_TO_RECIEVE_EMAIL	NULL = N	Y=1, N=0
DIM_DOMAIN_ID		
ENTRY_LEVEL_JOB_FLAG	NULL=N	Y=1, N=0
INTERNSHIP_FLAG	NULL=N	Y=1, N=0
PART_TIME_JOB_FLAG	NULL=N	Y=1, N=0
WHO_REGISTERED_TYPE_TXT		
ALLOW_MARKETING_FLAG	NULL=F	F=0 ; T = 1
HOLD_NOTIFICATION_EMAIL_FLAG	NULL=0	
HOLD_NEWSLETTER_EMAIL_FLAG	NULL=0	
HOLD_OTHER_EMAIL_FLAG	NULL=0	

Příklad 16: Opravy atributů

2.4 Konverze (Conversion)

Dalším krokem je konverze na úrovni jednotlivých atributů. Tím se rozumí převážně konverze datových typů, případně jejich preciznosti či délky. Nejčastějším nástrojem jsou zde, pro daný systém nativní, konverzní funkce (např. TO_DATE, TO_CHAR v prostředí Oracle:). Pokud tvůrce systému nepostupuje podle standardů (např. standardizovaný datový typ emailové adresy – VARCHAR2 (256)), nebo se význam atributu v průběhu existence systému změní, datový typ atributu nemusí odpovídat typu informace v něm uložené (např. numerický identifikátor uložený v textovém poli). Hlavním důvodem konverze je transparentnost a validita v rámci jednotlivých systémů a použitých ETL procesů, kterými jsou data jednorázově nebo průběžně přenášena.

2.5 Transformace (Transformation)

Transformací se rozumí finální převod dat do sémantiky cílového systému. Ta může například řešit opačné chápání významu hodnot TRUE (1), FALSE (0) mezi dvěma systémy, sloučení významu dvou zdrojových atributů do jednoho cílového, nebo další transformace založené například na podmínkách typu IF-THEN-ELSE (když-tak-jinak) nebo CASE (v případě, že).

Sloučení významu dvou atributů

B&S_USERS	**Transformace**	**Cílový sloupec**
HOLD_NOTIFICATION_EMAIL_FLAG	0 -> 1; 1 -> 0	OPTIN_NOTIFICATION
HOLD_NEWSLETTER_EMAIL_FLAG	0 -> 1; 1 -> 0	OPTIN_NEWSLETTER
HOLD_OTHER_EMAIL_FLAG	0 -> 1; 1 -> 0	OPTIN_OTHER_EMAIL
CURRENT_SCHOOL_YR_CODE_TXT	Not NULL -> 0	SCHOOL_ENROLLED
CURRENTLY_ENROLLED_FLAG	Not NULL -> 1, orhers NULL	
US_CITIZEN_FLAG	Y -> 'US', others -> '-'	CITIZENSHIP

Příklad 17: Vybrané transformace atributů

Shrnutí

Následující příklad ilustruje veškeré techniky, které by bylo třeba aplikovat při transformaci informace o rase uživatele ze systému Beta&sons do systému Alfa co.

Rasa uživatele je ve zdrojovém systému zastoupena více atributy, zatímco v cílovém systému reprezentuje rasu pouze jeden atribut. Mapování je tedy následující.

Zdrojový atribut	**Datový typ**	**Délka**	**NULL**
ŘACE_WHITE_TXT	VARCHAR2	3	Y
RACE_BLACK_TXT	VARCHAR2	3	Y
RACE_ASIAN_TXT	VARCHAR2	3	Y
RACE_INDIAN_TXT	VARCHAR2	3	Y
RACE_HISPANIC_TXT	VARCHAR2	3	Y

Cílový atribut	**Datový typ**	**Délka**	**NULL**
RACE	VARCHAR2	50	N

Dalším krokem je extrakce nulových hodnot. Jak je vidět na následujícím příkladě, poměr zastoupení nulových hodnot v celé množině dat a vzorku z roku 2011 je víceméně konzistentní.

Zdrojový atribut	Celkový počet nulových hodnot	Počet nulových hodnot- rok 2011 (904,277)
RACE_WHITE_TXT	48.16%	51.92%
RACE_BLACK_TXT	83.39%	83.03%
RACE_ASIAN_TXT	92.26%	93.24%
RACE_INDIAN_TXT	97.33%	97.45%
RACE_HISPANIC_TXT	90.81%	88.92%

Atributy rasy obsahují ve zdrojovém systému hodnoty “Y“, “N“ nebo NULL. Je tedy třeba aplikovat následující čištění.

Zdrojový atribut	Čištění
RACE_WHITE_TXT	NULL -> “N“
RACE_BLACK_TXT	NULL -> “N“
RACE_ASIAN_TXT	NULL -> “N“
RACE_INDIAN_TXT	NULL -> “N“
RACE_HISPANIC_TXT	NULL -> “N“

Hodnoty typu flag jsou v cílovém systému specifikovány jako 1 (TRUE) a 0 (FALSE). Oprava je tedy následující.

Zdrojový atribut	Oprava
RACE_WHITE_TXT	“N“ -> 0, “Y“´ -> 1
RACE_BLACK_TXT	“N“ -> 0, “Y“´- > 1
RACE_ASIAN_TXT	“N“ -> 0, “Y“´ -> 1
RACE_INDIAN_TXT	“N“ -> 0, “Y“´- > 1
RACE_HISPANIC_TXT	“N“ -> 0, “Y“´ -> 1

Poznámka: V takto jednoduchém příkladě lze samozřejmě předchozí kroky čistění a opravy sloučit, což platí i pro další kroky v rámci celé transformace. Pokud však pracujeme s větším množstvím dat, je žádoucí provádět jednotlivé operace separátně vzhledem k jejich náročnosti na systémové prostředky (složitost výpočtů je přímo úměrná nárokům na paměť). Data jsou pak po každém kroku ukládána ve speciálních tzv. “stage“ tabulkách, odkud jsou v následujícím kroku přelévána do dalších tabulek až do momentu nahrání do cílové struktury. Dobrou konvencí je tyto tabulky označovat prefixem S_<název tabulky> a postfixem, který odpovídá předchozí operaci např. S_ B_AND_S_USERS_CLNS (vyčištěná uživatelská data) či S_USERS_CONSOLIDATED (konsolidovaná uživatelská data z více zdrojů/tabulek).

Vzhledem k tomu, že cílový atribut není typu hodnoty flag, proběhne konverze a transformace současně.

Zdrojový atribut	Transformace	Cílový atribut
ŘACE_WHITE_TXT :VARCHAR2(3), NULL,	1 -> 'WHITE'	RACE: VARCHAR2(50), NOT NULL,
RACE_BLACK_TXT :VARCHAR2(3), NULL	1 -> 'BLACK'	
RACE_ASIAN_TXT:VARCHAR2(3) NULL	1-> 'INDIAN'	
RACE_INDIAN_TXT :VARCHAR2(3) NULL	1 -> 'HISPANIC'	
RACE_HISPANIC_TXT :VARCHAR2(3) NULL	1 -> 'ASIAN',	
	Others -> '-'	

Popsané kroky by tedy v ideálním případě vedly k úspěšnému namapování atributu rasy mezi dvěma systémy a následnému převodu potřebné informace tj. atributu rasy entity člověk (uživatel).

Rizika

V popisu vzorového projektu není ovšem specifikováno, jakým způsobem firma Beta&sons pracuje s daty v momentě jejich zadávání, resp. jakým způsobem brání možnosti vyplňování nesmyslných údajů při registraci uživatele.

Nelze tedy předpokládat, že jsou původní flag hodnoty RACE_<spec> navzájem výlučné, tj. uživateli nemusí být zabráněno ve volbě více různých ras. S touto nekonzistencí je třeba počítat a učinit patřičné kroky k jejímu odstranění. (Nezvažujme nyní variantu empirického ověření, kdy se jednoduše analytik zaregistruje do systému Beta&sons a jako nový uživatel funkcionalitu příslušného formuláře ověří.) V takovýchto případech se často určují priority na základě konzultace s marketingovým oddělením. V případě určení rasy uživatele však logicky nelze žádnou hodnotu upřednostňovat. Konečná transformace je tedy složitější a bude při ní nutné ověřit i vzájemné kombinace hodnot atributů RACE_<spec>. V případě výskytu hodnoty 1 ve více atributech jednoho uživatele bude výstup výsledné transformace rasy nabývat hodnotu Others (Jiná) -> '-'.

3. Model entit a vznik datového slovníku

Z důvodu komplexity aplikací a struktur (databází), které jsou pro tyto aplikace zdrojem dat, je v reálu nemožné (nebo minimálně nerozumné) uchovávat metadata pouze v jakési ploché textové formě. Ideální kombinací je udržovat tyto informace v různých úrovních abstrakce modelů a souvisejících slovníků. Od ontologického modelu, který popisuje zvolenou oblast "tak, jak je", přes obecný model entit, konceptuální model, až po konkrétnější reprezentaci, kterou je logický či fyzický model. Související dokumentace popisující jednotlivé entity a jejich atributy se v každé této úrovni také liší. V nevyšších úrovních pak vůbec nefiguruje. Existence modelu i datového slovníku minimálně na straně cílového systému, v našem případě tedy datamartu firmy Alfa co., je předpokladem pro zvládnutí úkolu integrace nových dat z jiného prostředí. Ontologický model může být na druhou stranu prvním krokem pro pochopení fungování procesů na straně firmy Beta&sons. Následující kapitola popisuje jednotlivá východiska pro tvorbu takových modelů a souvisejících datových slovníků.

Analýza entit a jejich vztahů se skládá ze tří hlavních prvků abstrakce, které popisují modelované skutečnosti. Hovoříme zde o entitách, jejich relacích a atributech. Entity chápejme jako navzájem odlišné (distinct) prvky skutečného světa, relace jsou pak vzájemné vztahy mezi nimi a atributy rozumíme povahu nebo vlastnosti jednotlivých entit.

V případě softwarového designu se podobné objekty sdružují do skupin tedy entit. Aby bylo možné modelovat interakce mezi jednotlivými objekty množiny entit, použijeme relace či množinu relací. Modelování relací může být někdy dosti obtížné. Zatímco entity dokážeme poměrně dobře identifikovat (např. uživatel, objednávka, kniha), vztahy mezi nimi již nemusí být vůbec patrné. Občas je tedy nutné postupně vytvářet model něčeho, co ve skutečnosti fyzicky jako samostatný objekt neexistuje. Z tohoto důvodu lze identifikovat dvě (popřípadě i více) vrstev modelování entit a jejich vztahů.

Právě k popisu procesu vývoje modelů nám může pomoci tzv. princip tří architektur (P3A). V souvislosti s architekturou P3A se také často hovoří o třech úrovních návrhu informačního systému.

Architektura P3A definuje způsob použití abstrakce, což znamená, že nám umožňuje rozčlenit právě zkoumanou problematiku návrhu datové základny na mentálně zvládnutelné části. Jak již název vypovídá, skládá se P3A ze třech vrstev (úrovní) - konceptuální, technologické (logické) a implementační (fyzické). Z toho vyplývá i rozdělení typologie vzniklých modelů. Jedná se o konceptuální datové modely, modely logické a modely fyzické. V případě budování datových skladů však běžně konceptuální a logický model splývají.

Z pohledu implementace entit a jejich atributů do systému lze navíc identifikovat další, jakési meta úrovně (2, 3, 4.), které sice popisují již fyzické objekty, přesto nejsou vázány na jejich implementaci v daném systému. Tyto možné vrstvy ilustruje následující příklad reprezentace atributu Křestní Jméno entity Zákazník.

1)

ENTITY	ENTITY_ATTRIBUTE	DATA_TYPE	CONSTRAINTS	VALUE
Customer	FirstName	Text	MaxLength(100), UTF8, not ("";,:*()~!@$%^*_[]{}<>?|)	Discrete

Příklad 18: Konceptuální/logická vrstva

2)

ENTITY	ENTITY_ATTRIBUTE	SYNONYM	DATA_TYPE	NULLABLE	DEFAULT
Customer	FirstName	FIRST_NAME	VARCHAR2(100)	No	

Příklad 19: Vrstva implementace atributu v rámci databáze Oracle (nezávislá na konkrétním modulu systému : OLTP, stage oblast, datamart či reportingové formuláře)

3)

ENTITY_ATTRIBUTE	SYNONYM	DATA_TYPE	NULLABLE	DEFAULT	USED_IN
FirstName	FIRST_NAME	VARCHAR2(100)	No	'-'	DM_CUSTOMERS
FirstName	FIRST_NAME	VARCHAR2(100)	Yes		OFFERS_TO_SEND

Příklad 20: Fyzická vrstva popisující uložení atributu v datamartu Alfa co.

3.1 Ontologie a konceptuální model

Konceptuální model, jak vyplývá z jeho názvu, pojednává o pojmech. Jedná se o model reálného světa. Konceptuálních modelů může existovat celá řada. Nejstarší generace těchto modelů vznikala v době nástupu relačních databází a v souladu s jejími schopnostmi obsahovala pouze jediný typ vztahů a to vztah prostý v angličtině označovaný většinou jako "relationship". I samotný konceptuální model se však vyvíjel a postupně se obohatil o další typy vztahů, tak jak je dnes známe z oblasti objektového paradigmatu. Jedná se o vztah **specializace-generalizace** (lze označit i jako **dědičnost)** a vztah **celek-část**, také nazývaný **agregace** či **kompozice**. Konceptuální model je zcela obecný základní pojmový model, absolutně nezávislý na jakékoliv potenciální implementaci jeho obsahu (MOHANEC, 2006).

Existuje několik přístupů, jak vybudovat teoretické základy konceptuálního modelu. Problém nastává v okamžiku, kdy potřebujeme přesněji stanovit některé pojmy, například: co je to objekt, co je to třída, jak se liší vztah agregace od prostého vztahu 1:M, atp. Tvrzení, že výklad jednotlivých pojmů závisí na zkušenosti analytika, je sice do určité míry pravdivé a většina odborníků intuitivně výše uvedené pojmy správně používá, ale pro vybudování formálních základů empirický přístup nestačí. Na straně druhé je nutné zdůraznit, že empirická zkušenost je jediným platným potvrzením těchto teoretických hypotéz (MOHANEC, 2006).

Jaké tedy existují možnosti pro vybudování formálního základu konceptuálního modelování? Uveďme si dva základní přístupy, a to:

Matematicko-logický přístup, který vychází z teorie množin, relací a matematické logiky,

a pro naši potřebu zajímavější

Ontologický přístup, což je přístup novější, vycházející z paradigmatu ontologií.

3.1.1 Ontologie

Ontologie je poměrně populární pojem, zvláště v poslední době, kdy vzniká celá řada prací na téma využití ontologií v prostředí Internetu. My se však budeme zabývat původnějším užitím tohoto pojmu, a to jako označení teoretického východiska pro vybudování konceptuálního modelu, který je odrazem reálného světa. Ontologický přístup zkoumá význam jednotlivých pojmů co je to objekt, co je vlastnost, co je to vztah. Na rozdíl od matematicko-logických přístupů je pro ontologický přístup prvotní význam sám o sobě, a teprve na druhém místě matematicko-logický formalizmus. Pochopitelně je tomu i obráceně. Matematicko-logické přístupy zkoumají význam jednotlivých kategorií reálného světa, a proto pracují i s ontologiemi. (MOHANEC, 2006).

Pokud bychom chtěli ukotvit ontologii z pohledu filozofie, jako vhodný poslouží například citát definice z publikace Úvod do ontologie: "Pojem ontologie (slovo odvozeno z řečtiny; on, ontos – jsoucí, logos – výklad) se objevuje až v 17. století – 1613 Goclein, 1656 Clauberg – a jeho použití ve filosofickém významu je spojeno se jménem Christiana Wolffa, který jím chce nahradit pojem metafyziky ve smyslu učení o bytí vůbec. Ontologie jako filosofická disciplína se zajímá o bytí ve smyslu jeho nejobecnějších určení, vlastností a projevů. V tomto smyslu se tedy příliš nevzdaluje původnímu (Aristotelovskému) významu pojmu metafyzika (ve smyslu první filosofie)." (ŠMAJS, KROB, 1994)

Nás však více zajímá ontologie spíše jako inženýrská disciplína vhodná coby východisko pro formální teorii konceptuálního modelování. Ontologie jako předmět praktického ontologického inženýrství, tj. jako informační artefakt, je univerzální soustava znalostí popisující objekty, jevy a zákonitosti světa "tak, jak je", tj. maximálně nezávisle na lidském usuzování o něm (SVÁTEK, 2002).

Ontologie specifikuje použitý způsobu konceptualizace skutečnosti. Nejčastěji ji představuje skupina pravidel, která lze aplikovat v procesu získávání a reprezentaci znalostí. Vzhledem k jiným reprezentačním prostředkům se zde zdůrazňuje vyjadřování sémantických a pragmatických stránek využívaných a předávaných informací. (GUARION, GIARETTA, 1995).

Uveďme si nyní některé základní specifikace ontologie:

- Je řečeno proč a k jakému účelu se ontologie tvoří. (Ontologie pro distribuci emailové korespondence bude jiná, než ontologie pro konstruování automobilů, i když obě ontologie budou používat společný průnik terminologie.)

- Je určen způsob práce s ontologií, forma počítačové podpory a uživatel ontologie (přímý i nepřímý).

- Je řečeno, které objekty, relace, operace a další formální objekty budou použity ke konceptualizaci odborné oblasti, pro kterou se ontologie zpracovává.

- Je stanovena terminologie dané odborné oblasti (zohledňující různé školy, tradice, apod.)

- Je určeno reprezentační prostředí, ve kterém se ontologie zapíše i prostředí, ve kterém se bude implementovat.

- Podle předmětu formalizace se dnes rozlišují ontologie: **doménové,** úlohové, aplikační a generické.

- Základními strukturálními prvky ontologií jsou: Třídy, relace a funkce

(BÍLA, TLAPÁK, 2004).

Pro popis a reprezentaci ontologií existuje několik jazyků. Jedním z nich je jazyk OWL (Web Ontology Language). Tento jazyk umožňuje publikování a sdílení ontologií v prostředí WWW (MCGUINESS, 2004). Ontologii lze reprezentovat logicky i v jazyce UML. O tom však později v této práci.

Ačkoliv se využití jazyka OWL může zdát v rámci námi řešeného úkolu zavádějící, OWL dobře reprezentuje možnosti ontologií a zároveň lze vysledovat jistou paralelu mezi

zápisem ontologie (resp. vlastností objektů) jednoho systému a mapováním mezi atributy (resp. jejich objektovými reprezentacemi) tak, jak budou popisovány v kapitolách 4.4 – 4.7 této práce.

Pro jazyk OWL existuje několik nástrojů, kterými je možné ontologie vytvářet např. Altova SemanticWorks, TopBraid Composer, nebo Protege-OWL. Poslední zmíněný patří do kategorie open source, a jako takový se jeví vhodný pro seznámení se s touto problematikou, zvláště když jsme již na počátku zvolili řešení, které předpokládá minimální investice do nákupu existujících nástrojů. Podrobnější popis sémantiky jazyka OWL lze nalézt v použitých pramenech. (HORRIDGE, RECTOR, STEVENS, WROE, 2004; KLEMPA, 2006/2007), (MCGUINESS, 2004). Uveďme si pro ilustraci příklad, který předchází, ale zároveň zpětně doplňuje, následný konceptuální model části datamartu Alfa co. v jazyce **UML**[17] (kapitola 3.1.2 Konceptuální model a datový slovník).

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY Ontology11704153788632
"http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#" >
    <!ENTITY Ontology11704153788633
"http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#1+" >
    <!ENTITY Ontology11704153788635
"http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#5+" >
    <!ENTITY Ontology1170415378863
"http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#7+" >
    <!ENTITY Ontology11704153788634
"http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#10+" >
]>
<rdf:RDF
xmlns="http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#"
    xml:base="http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:Ontology11704153788635="&Ontology11704153788632;5+"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:Ontology11704153788633="&Ontology11704153788632;1+"
     xmlns:Ontology11704153788634="&Ontology11704153788632;10+"
     xmlns:Ontology1170415378863="&Ontology11704153788632;7+"
xmlns:Ontology11704153788632="http://www.semanticweb.org/ontologies/2007/1/2/Ontology11
70415378863.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

[17] **Unified Modeling Language** – standardizovaný jazyk vizuální specifikace určený pro modelování objektů. Tento obecný jazyk umožňuje graficky znázornit abstraktní model systému, v našem případě. UML model tříd.

```
    <owl:Ontology rdf:about="">
        <dc:creator>Ivan BARTOS</dc:creator>
        <dc:language>English</dc:language>
        <owl:versionInfo xml:lang="en"
            >v1.0.0 Basic etities, clases and object properties
created</owl:versionInfo>
    </owl:Ontology>
    <!--
////////////////////////////////////////////////////////////////////////////////////////
    //
    // Annotation properties
    //
////////////////////////////////////////////////////////////////////////////////////////
     -->
    <owl:AnnotationProperty rdf:about="&dc;language"/>
    <owl:AnnotationProperty rdf:about="&dc;creator"/>
    <!--

////////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
////////////////////////////////////////////////////////////////////////////////////////
     -->
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#belongsTo -->
    <owl:ObjectProperty rdf:about="#belongsTo">
        <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
        <rdf:type rdf:resource="&owl;ReflexiveProperty"/>
        <rdfs:domain rdf:resource="#GeoRegion"/>
        <rdfs:range rdf:resource="#GeoRegion"/>
    </owl:ObjectProperty>
    <!-- http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#has -
->
    <owl:ObjectProperty rdf:about="#has">
        <rdfs:range rdf:resource="#Career"/>
        <rdfs:range rdf:resource="#Education"/>
        <rdfs:range rdf:resource="#Experience"/>
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#has"/>
                <owl:someValuesFrom rdf:resource="#EmailAddress"/>
            </owl:Restriction>
        </rdfs:range>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#has"/>
                <owl:someValuesFrom rdf:resource="#EducationLevel"/>
            </owl:Restriction>
        </rdfs:range>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#has"/>
                <owl:someValuesFrom rdf:resource="#Occupation"/>
            </owl:Restriction>
        </rdfs:range>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#has"/>
                <owl:someValuesFrom rdf:resource="#ExperienceLevel"/>
            </owl:Restriction>
        </rdfs:range>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#has"/>
                <owl:someValuesFrom rdf:resource="#CreerLevel"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
    <!--
```

```
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#hasGeoCoordina
tes -->
    <owl:ObjectProperty rdf:about="#hasGeoCoordinates">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="#GeoRegion"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasGeoCoordinates"/>
                <owl:onClass rdf:resource="#GeoCoordinates"/>
                <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#isPartOf -->
    <owl:ObjectProperty rdf:about="#isPartOf">
        <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
        <rdf:type rdf:resource="&owl;ReflexiveProperty"/>
        <rdfs:range rdf:resource="#GeoRegion"/>
        <rdfs:domain rdf:resource="#GeoRegion"/>
    </owl:ObjectProperty>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#livesIn -->
    <owl:ObjectProperty rdf:about="#livesIn">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:subPropertyOf rdf:resource="#specifiesGeoRegion"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#livesIn"/>
                <owl:allValuesFrom rdf:resource="#GeoRegion"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#speaks -->
    <owl:ObjectProperty rdf:about="#speaks">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#speaks"/>
                <owl:onClass rdf:resource="#Language"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
            </owl:Restriction>
        </rdfs:range>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#speaks"/>
                <owl:someValuesFrom rdf:resource="#LanguageLevel"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#specifiesGeoRe
gion -->
    <owl:ObjectProperty rdf:about="#specifiesGeoRegion">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#specifiesGeoRegion"/>
                <owl:someValuesFrom rdf:resource="#GeoRegion"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#subscribes -->

    <owl:ObjectProperty rdf:about="#subscribes">
        <rdfs:range rdf:resource="#Mailing"/>
        <rdfs:domain rdf:resource="#User"/>
```

```
    </owl:ObjectProperty>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#subscribesIn -
->
    <owl:ObjectProperty rdf:about="#subscribesIn">
        <rdfs:range rdf:resource="#Mailing"/>
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:subPropertyOf rdf:resource="#subscribes"/>
    </owl:ObjectProperty>
      <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#unsubcribesGlo
bally -->
    <owl:ObjectProperty rdf:about="#unsubcribesGlobally">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:subPropertyOf rdf:resource="#subscribes"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#unsubcribesGlobally"/>
                <owl:allValuesFrom rdf:resource="#Mailing"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#wantsWorkIn --
>
    <owl:ObjectProperty rdf:about="#wantsWorkIn">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:subPropertyOf rdf:resource="#specifiesGeoRegion"/>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#wantsWorkIn"/>
                <owl:someValuesFrom rdf:resource="#GeoRegion"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#worksFor -->
    <owl:ObjectProperty rdf:about="#worksFor">
        <rdfs:range rdf:resource="#Company"/>
        <rdfs:domain rdf:resource="#User"/>
    </owl:ObjectProperty>
    <!--
///////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
///////////////////////////////////////////////////////////////////////////////////////
     -->
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Address -->
    <owl:Class rdf:about="#Address">
        <owl:equivalentClass rdf:resource="#City"/>
        <owl:equivalentClass rdf:resource="#Country"/>
        <owl:equivalentClass rdf:resource="#PostalCode"/>
        <owl:equivalentClass rdf:resource="#State"/>
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Career -->
    <owl:Class rdf:about="#Career">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!-- http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#City
-->
    <owl:Class rdf:about="#City">
        <rdfs:subClassOf rdf:resource="#GeoRegion"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Company -->
    <owl:Class rdf:about="#Company">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
```

```
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#CompanySize --
>
    <owl:Class rdf:about="#CompanySize">
        <rdfs:subClassOf rdf:resource="#Company"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Continent -->
    <owl:Class rdf:about="#Continent">
        <rdfs:subClassOf rdf:resource="#GeoRegion"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Country -->
    <owl:Class rdf:about="#Country">
        <rdfs:subClassOf rdf:resource="#GeoRegion"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#County -->
    <owl:Class rdf:about="#County">
        <rdfs:subClassOf rdf:resource="#GeoRegion"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#CreerLevel -->
    <owl:Class rdf:about="#CreerLevel">
        <rdfs:subClassOf rdf:resource="#Career"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Domain -->
    <owl:Class rdf:about="#Domain">
        <rdfs:subClassOf rdf:resource="#EmailAddress"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Education -->
    <owl:Class rdf:about="#Education">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#EducationLevel
-->
    <owl:Class rdf:about="#EducationLevel">
        <rdfs:subClassOf rdf:resource="#Education"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#EmailAddress -
->
    <owl:Class rdf:about="#EmailAddress">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Experience -->
    <owl:Class rdf:about="#Experience">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#ExperienceLeve
l -->
    <owl:Class rdf:about="#ExperienceLevel">
        <rdfs:subClassOf rdf:resource="#Experience"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#GeoCoordinates
-->
    <owl:Class rdf:about="#GeoCoordinates">
        <rdfs:subClassOf rdf:resource="#GeoInformation"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#GeoInformation
-->
    <owl:Class rdf:about="#GeoInformation">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
```

```
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#GeoRegion -->
    <owl:Class rdf:about="#GeoRegion">
        <owl:equivalentClass rdf:resource="#Loaction"/>
        <rdfs:subClassOf rdf:resource="#GeoInformation"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Industry -->
    <owl:Class rdf:about="#Industry">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Language -->
    <owl:Class rdf:about="#Language">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#LanguageLevel
-->
    <owl:Class rdf:about="#LanguageLevel">
        <rdfs:subClassOf rdf:resource="#Language"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Latitude -->
    <owl:Class rdf:about="#Latitude">
        <rdfs:subClassOf rdf:resource="#GeoCoordinates"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Loaction -->
    <owl:Class rdf:about="#Loaction">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Longitude -->
    <owl:Class rdf:about="#Longitude">
        <rdfs:subClassOf rdf:resource="#GeoCoordinates"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Mailing -->
    <owl:Class rdf:about="#Mailing">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Occupation -->
    <owl:Class rdf:about="#Occupation">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#PostalCode -->
    <owl:Class rdf:about="#PostalCode">
        <rdfs:subClassOf rdf:resource="#GeoRegion"/>
    </owl:Class>
    <!-- http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#State
-->
    <owl:Class rdf:about="#State">
        <rdfs:subClassOf rdf:resource="#GeoRegion"/>
    </owl:Class>
    <!-- http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#User
-->
    <owl:Class rdf:about="#User">
        <rdfs:subClassOf rdf:resource="&owl;SMAS_Enterprize"/>
    </owl:Class>
    <!-- http://www.w3.org/2002/07/owl#SMAS Enterprize -->
    <owl:Class rdf:about="&owl;SMAS_Enterprize"/>
    <!--
///////////////////////////////////////////////////////////////////////////////////////
    //
    // Individuals
    //
///////////////////////////////////////////////////////////////////////////////////////
```

```
     -->
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#1+_to_2_Years
-->
    <ExperienceLevel rdf:about="#1+_to_2_Years"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#10+_to_15_Year
s -->
    <ExperienceLevel rdf:about="#10+_to_15_Years"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#5+_to_7_Years
-->
    <ExperienceLevel rdf:about="#5+_to_7_Years"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#7+_to_10_Years
-->
    <ExperienceLevel rdf:about="#7+_to_10_Years"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Associate_Degr
ee -->
    <EducationLevel rdf:about="#Associate_Degree"/>
      <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Bachelors_Degr
ee -->
    <EducationLevel rdf:about="#Bachelors_Degree"/>
      <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Certification
-->
    <EducationLevel rdf:about="#Certification"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Doctorate -->
    <EducationLevel rdf:about="#Doctorate"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Entry_Level --
>
    <CreerLevel rdf:about="#Entry_Level"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Executive_(SVP
,_VP,_Department_Head,_etc) -->
    <CreerLevel rdf:about="#Executive_(SVP,_VP,_Department_Head,_etc)"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Experienced_(N
on-Manager) -->
    <CreerLevel rdf:about="#Experienced_(Non-Manager)"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#High_School_or
 equivalent -->
    <EducationLevel rdf:about="#High_School_or_equivalent"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Large_(1000+)
-->
    <CompanySize rdf:about="#Large_(1000+)"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Less_than_1_Ye
ar -->
    <ExperienceLevel rdf:about="#Less_than_1_Year"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Manager_(Manag
er/Supervisor_of_Staff) -->
    <CreerLevel rdf:about="#Manager_(Manager/Supervisor_of_Staff)"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Masters_Degree
-->
    <EducationLevel rdf:about="#Masters_Degree"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Medium_(100_-
_999) -->
    <CompanySize rdf:about="#Medium_(100_-_999)"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#More_than_15_Y
ears -->
```

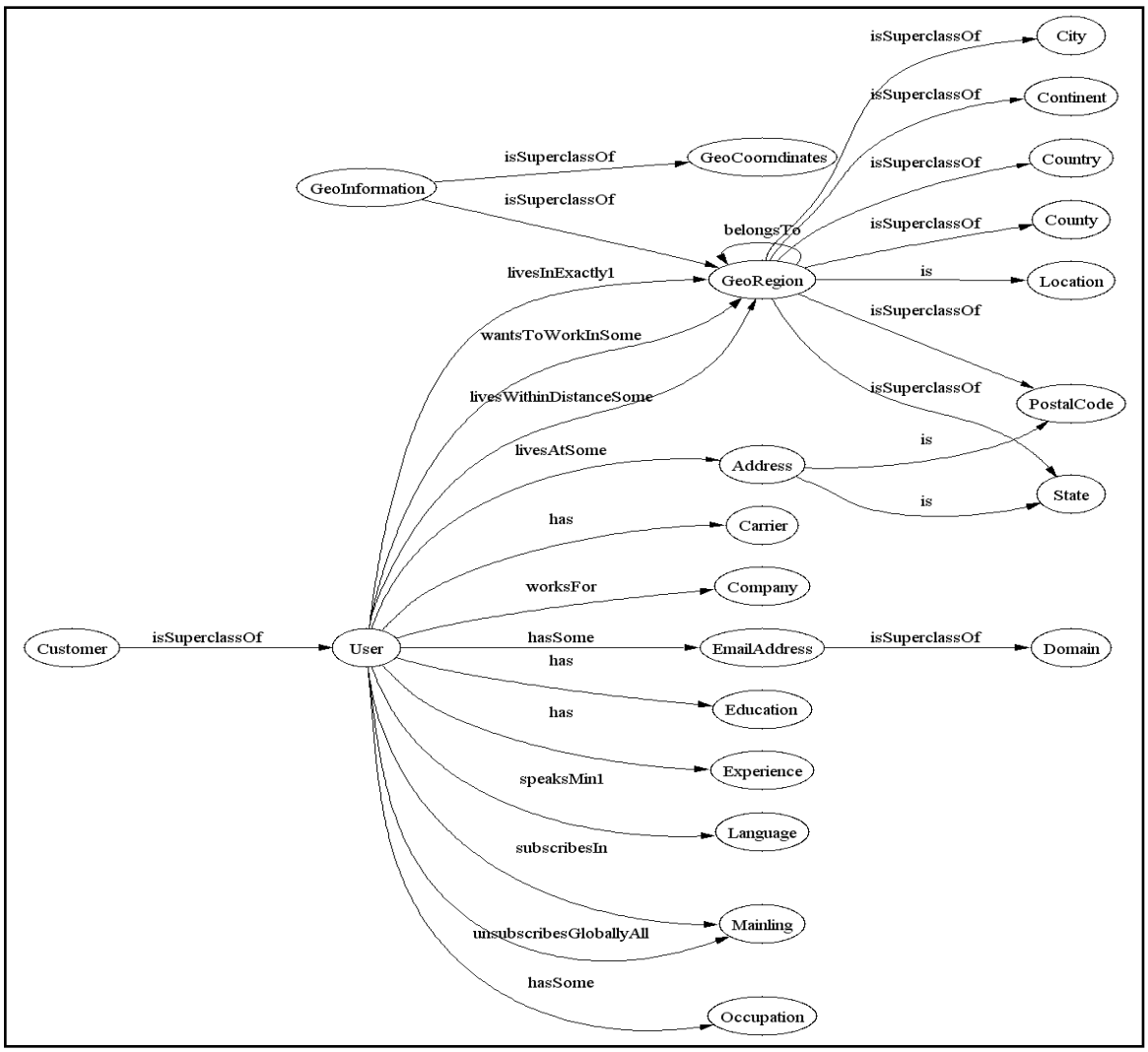
```
    <ExperienceLevel rdf:about="#More_than_15_Years"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#No_Preference
-->
    <CompanySize rdf:about="#No_Preference"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Not_Specified
-->
    <CreerLevel rdf:about="#Not_Specified">
        <rdf:type rdf:resource="#CompanySize"/>
        <rdf:type rdf:resource="#EducationLevel"/>
        <rdf:type rdf:resource="#ExperienceLevel"/>
    </CreerLevel>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#OptIn_1st -->
    <Mailing rdf:about="#OptIn_1st"/>
     <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#OptIn_3rd -->
    <Mailing rdf:about="#OptIn_3rd"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#OptIn_Newslett
er -->
    <Mailing rdf:about="#OptIn_Newsletter"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Professional -
->
    <EducationLevel rdf:about="#Professional"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Senior_Executi
ve (President, CEO, etc) -->
    <CreerLevel rdf:about="#Senior_Executive_(President,_CEO,_etc)"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Small_(1_-_99)
-->
    <CompanySize rdf:about="#Small_(1_-_99)"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Some_College_C
oursework_Completed -->
    <EducationLevel rdf:about="#Some_College_Coursework_Completed"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Some_High_Scho
ol_Coursework -->
    <EducationLevel rdf:about="#Some_High_School_Coursework"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Student_(High_
School) -->
    <CreerLevel rdf:about="#Student_(High_School)"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Student_(under
graduate/graduate) -->
    <CreerLevel rdf:about="#Student_(undergraduate/graduate)"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Vocational -->
    <EducationLevel rdf:about="#Vocational"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Vocational_-
_Degree -->
    <EducationLevel rdf:about="#Vocational_-_Degree"/>
    <!--
http://www.semanticweb.org/ontologies/2007/1/2/Ontology1170415378863.owl#Vocational_-
_High_School -->
    <EducationLevel rdf:about="#Vocational_-_High_School"/>
    <!--
///////////////////////////////////////////////////////////////////////////////////////
    //
    // General axioms
    //

///////////////////////////////////////////////////////////////////////////////////////
     -->
    <rdf:Description>
```

```
        <rdf:type rdf:resource="&owl;AllDifferent"/>
        <owl:distinctMembers rdf:parseType="Collection">
            <rdf:Description rdf:about="#OptIn_Newsletter"/>
            <rdf:Description rdf:about="#OptIn_1st"/>
            <rdf:Description rdf:about="#OptIn_3rd"/>
        </owl:distinctMembers>
    </rdf:Description>
</rdf:RDF>
<!-- Generated by the OWL API (version 2.2.1.962) http://owlapi.sourceforge.net -->
```

Příklad 21: Ontologie Alfa co. v jazyce OWL – open source Protege – OWL dle (MCGUINESS, 2004)

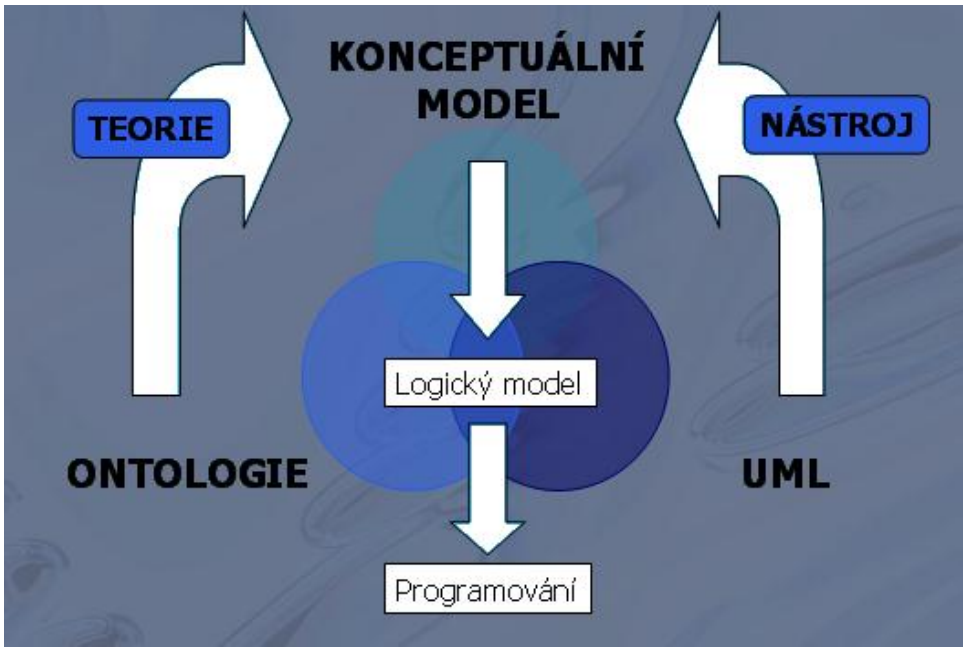
```
digraph G
{rankdir="LR";
User->Address[label="livesAtSome"];
User->Carrier[label="has"];
GeoRegion->City[label="isSuperclassOf"];
User->Company[label="worksFor"];
GeoRegion->Continent[label="isSuperclassOf"];
GeoRegion->Country[label="isSuperclassOf"];
GeoRegion->County[label="isSuperclassOf"];
EmailAddress->Domain[label="isSuperclassOf"];
User->Education[label="has"];
User->EmailAddress[label="hasSome"];
User->Experience[label="has"];
GeoInformation->GeoCoorndinates[label="isSuperclassOf"];
GeoRegion->GeoRegion[label="belongsTo"];
GeoInformation->GeoRegion[label="isSuperclassOf"];
User->GeoRegion[label="livesInExactly1"];
User->GeoRegion[label="wantsToWorkInSome"];
User->GeoRegion[label="livesWithinDistanceSome"];
User->Language[label="speaksMin1"];
GeoRegion->Location[label="is"];
User->Mainling[label="subscribesIn"];
User->Mainling[label="unsubscribesGloballyAll"];
User->Occupation[label="hasSome"];
GeoRegion->PostalCode[label="isSuperclassOf"];
Address->PostalCode[label="is"];
GeoRegion->State[label="isSuperclassOf"];
Address->State[label="is"];
Customer->User[label="isSuperclassOf"];
}
```

Příklad 22: Export ontologie Alfa co. jako vstup pro tvorbu DOT diagramu



Obrázek 8: Rozšířený DOT diagram ontologie Alfa co.

Jak tedy přirozeně spojit ontologii s problémem modelování architektury P3A a následné tvorbě datového slovníku? Je třeba si uvědomit základní rozdíl a zároveň provázanost obou konceptů.



Obrázek 9: Ontologie – konceptuální model – UML (MOHANEC, 2005)

- Ontologie dává přesný význam konceptuálnímu modelu.
  - Definuje přesně jeho jednotlivé pojmy (díky pojmenování vazeb tak, jak v reálném světě fungují).
- Konceptuální model je možné vyjádřit v UML.
  - Používá UML přesně definovaným způsobem.
  - Upřesňuje jeho sémantiku s ohledem na použitou ontologii.

(BÍLA, TLAPÁK, 2004)

Jaký je tedy skutečný význam ontologie pro konceptuální modelování?

- Ontologie poskytuje vědecký (filozofický) základ pro výklad konstruktů konceptuálního modelu.

- Společně s logikou poskytuje základ pro formální popis konceptuálního modelu. (GUARINO, 1995)

3.1.2 Konceptuální model a datový slovník

Jak již bylo řečeno, konceptuální model reprezentuje nejvyšší vrstvu logické struktury, která je zcela nezávislá na své implementaci (použitý software, způsob a

struktura uložení dat). Konceptuální model často obsahuje objekty, které ve skutečnosti neexistují, nebo zatím nejsou implementovány ve fyzické databázi. Takový model umožňuje formální reprezentaci dat, která je obrazem reálného světa, a na které je možné postavit potřebnou obchodní aktivitu.

Mezi hlavní vlastnosti konceptuálního modelu můžeme zahrnout následující:

- Obsahuje důležité entity a vztahy mezi nimi.
- Může (pokud není žádoucí sloučení konceptuálního a logického modelu), ale nemusí obsahovat specifikaci atributů.
- Neobsahuje žádné primární ani sekundární klíče.

Mohanec ve své práci vymezuje konceptuální model například takto:

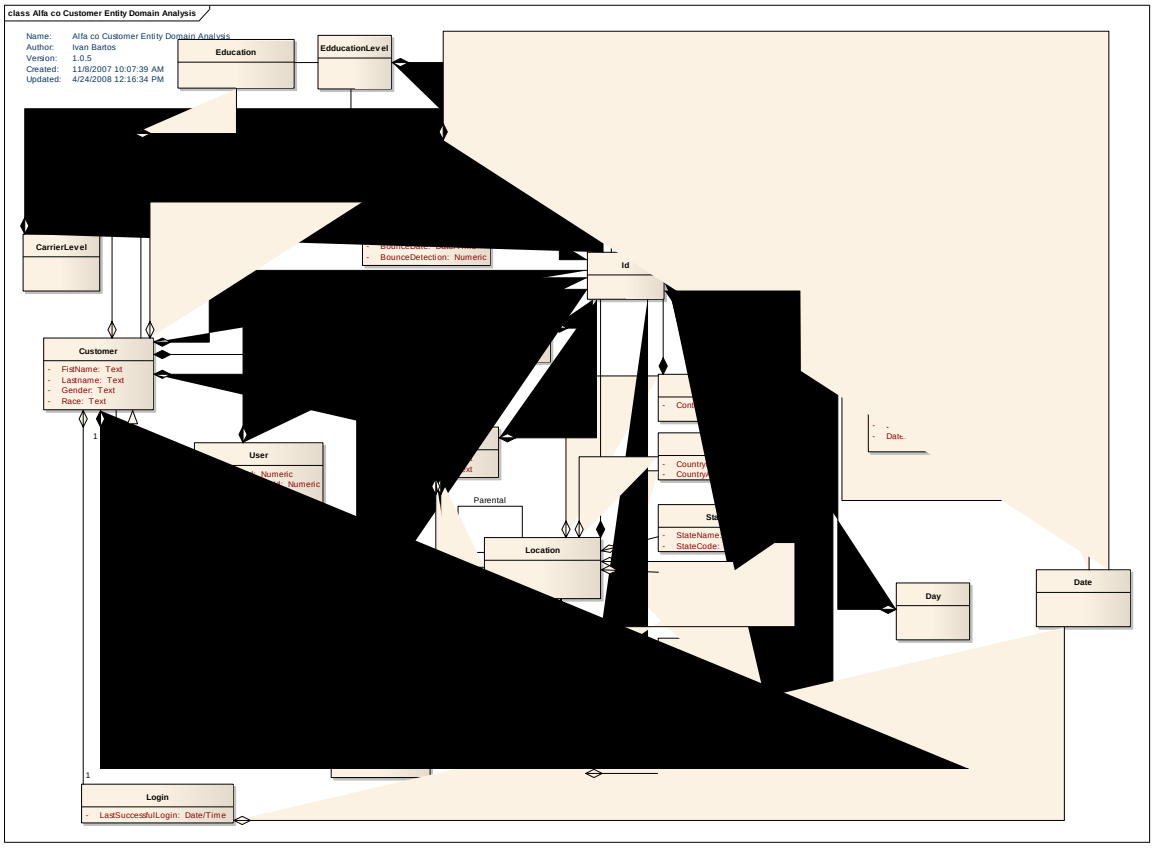
1. Objektem jeho zájmu jsou objekty v reálném nebo abstraktním světě (objekty).
2. Zajímá se o vlastnosti těchto objektů (atributy a vztahy).
3. Zajímá se o identifikaci těchto objektů v reálném světě (jedinečnost).
4. Jeho zájmem je klasifikace těchto objektu dle různých společných vlastností (třídy).
5. Jeho zájmem jsou vztahy mezi těmito klasifikačními třídami (generalizace-specializace).
6. Jeho zájmem jsou struktury tvořené objekty (skládání).
7. Zajímá se o další vzájemné vztahy mezi objekty a o klasifikaci těchto vztahů (vztahy prosté).

(MOHANEC, 2006)

Na konceptuální úrovni se tedy zaměřujeme pouze na identifikaci nejdůležitějších vztahů mezi různými entitami.

Tento model může být obecný až do takové úrovně, že jeho následná implementace nemusí záviset ani na moderních informačních technologiích. V konkrétním případě vzorového projektu Alfa co. lze model uplatnit, resp. následně realizovat, například formou klasické papírové kartotéky s kartami členů klubu čtenářů, kterým firma zašle na danou adresu pro ně specifickou nabídku titulů. To znamená takovou nabídku titulů, která

odpovídá skupině čtenářů vybrané marketingovým oddělením firmy Alfa co. na základě jimi poskytnutých údajů (věk, vzdělání, pohlaví, atp.), kde tyto informace budou pouhými vyplněnými kolonkami na registrační kartě.



Obrázek 10: Konceptuální/Logický model - UML diagram tříd

Vztahy mezi jednotlivými entitami lze znázornit dvěma způsoby, a to na základě dvou odlišných přístupů.

První přístup inklinuje více k logickému a fyzickému chápání modelu. Je to způsob známý z klasických **ER**[18] modelů (diagramů). Tímto přístupem lze vyjádřit **kardinalitu vzájemných vztahů** např. 1:N, M:N atp.

Druhý přístup vychází z objektově orientovaného pojetí. Tento přístup rozpoznává další typy vztahů: **dědičnost** a **celek-část**, podobně jako je to obvyklé v objektově

[18] **Entity Relationship** –Vztahy mezi jednotlivými entitami.

orientovaném modelování. V tzv. fyzickém modelu se však objevují jen konstrukty realizovatelné v relační databázi, tj. entita, atribut a vztah 1:1 a 1:M. Vztahy, které se vyskytují v konceptuálním modelu, se musí transformovat na vztahy povolené v logickém a fyzickém modelu. Pro grafické znázornění fyzického modelu se však používá podobná notace jako pro konceptuální model, což může být pro začátečníky opět matoucí (MOHANEC, 2004).

Na základě konceptuálního modelu vzniká obecný datový slovník, který bývá označován jako Enterprise Data Dictionary, z pohledu dat pak jako "meta" metadatový slovník (dále jen EDD). EDD by měl postihovat několik hlavních kriterií popisované skutečnosti:

- Skutečný název datového prvku v kontextu entit
- Název asociovaný ke každému datovému prvku (synonymum)
- Popis každého datového prvku v přirozeném jazyce (relevantní technické informace atp.)
- Informaci o registrační autoritě datového prvku (správce prvku, data steward)
- Detaily o každém datovém prvku týkající se způsobu jeho zpracování systémem (délka datového prvku ve znacích, zda je numerický, alfa-numerický či jiného datového typu, popřípadě, který systém ho v reálně používá)
- Validační pravidla pro každý datový prvek (např. povolené hodnoty, rozsah atp.)

3.1.2.1 EDD Metadata (ISO/IEC 11179 Doporučení)

Norma ISO/IEC 11179 standardizuje a nabízí možný způsob uchovávání informace o určitém datovém prvku za účelem vytvoření datového slovníku. Informace zachycené v následující tabulce postihují základní pravidla ISO/IEC 11179 pro tvorbu EDD.

Datový prvek	Popis
Název datového prvku (Data Element Name)	Jednotka dat, pro která je formou množiny atributů specifikována její definice, identifikace, reprezentace a povolené hodnoty.
Identifikátor (Identifier)	Unikátní identifikátor datového prvku v rámci registrační autority (nezávislý na jazyce).
Verze (Version)	Identifikace verze specifikace datového prvku v rámci vývojové řady spravované registrační autoritou.
Registrační autorita (Registration Authority)	Organizace nebo osoba autorizovaná k potvrzení vložení datového prvku do EDD.
Synonymum (Synonymous Name)	Jednoslovné či víceslovné označení, které se může lišit od jeho skutečného názvu, ale reprezentuje stejný koncept datového prvku,
Kontext (Context)	Určení nebo popis aplikačního prostředí nebo disciplíny, ve které je datový prvek uplatněn nebo odkud pochází.
Definice (Definition)	Výrok, který vyjadřuje základní povahu datového prvku a jeho odlišnost od všech dalších datových prvků.
Datový typ (DataType)	Množina distinktních hodnot charakterizována vlastnostmi těchto hodnot a operacemi nad těmito hodnotami.
Maximální velikost (Maximum Size)	Maximální velikost datového prvku.
Minimální velikost (Minimum Size)	Minimální velikost datového prvku.
Povolené hodnoty (Permissible Values)	Množina reprezentací povolené instance datového prvku rozšiřující již definované formě vyjádření, layoutu, datovému typu, maximální a minimální velikosti prvku. Množina může být specifikována názvem, odkazem na zdroj, výčtem reprezentací dané instance nebo pravidly pro generování instance jako například seznamem hodnot (List of Values/Set) nebo rozsahem (Range).
Poznámky (comments)	Jakákoliv další vysvětlující poznámka k datovému prvku.

(ISO/IEC 11179 - 1, 1999)

Množina těchto informací může být dále rozšířena, resp. datový slovník může poskytovat komplexnější a detailnější informaci o každém datovém prvku, a to za hranice stanovené ISO/IEC 11179 normou. Tyto informace lze klasifikovat následujícím způsobem:

- **Identifikace a popis:** Obsahuje popis datového prvku a jeho definici. Množina těchto polí je aplikována na všechny datové prvky (např. identifikátor, definice)

- **Konfigurace:** Obsahuje podstatné informace o konfiguraci a správě datového prvku. Tyto informace jsou poskytovány autoritou spravující daný prvek. Tato množina je rovněž aplikována na všechny datové prvky (verze, komentář, použití v jiných modelech atp.)

- **Vlastnosti:** Obsahují informaci o atributu či sloupci (zdroj dat, délku dat, typ hodnoty, zabezpečení atp.)

- **Asociace:** Obsahuje detaily o atributech/sloupcích napříč logickými a fyzickými modely, které se vážou k danému datovému prvku (např. hierarchii atp.)

Míra popisu jednotlivých datových prvků je zcela závislá na potřebách specifické organizace a na složitosti modelovaných/popisovaných struktur. V rámci jedné organizace by však měl být přijat určitý konsensus, který by hloubku a způsob popisu stanovil a unifikoval, aby mohla být zaručena interoperabilita dílčích aplikací a systémů a usnadněna jejich případná integrace.

3.2 Logický model

Pro logický model je specifické, že zahrnuje:

- veškeré entity a vztahy mezi nimi
- veškeré atributy všech specifikovaných entit
- primární klíč každé entity
- specifikaci cizích klíčů (klíče, které určují vztahy mezi rozdílnými entitami)
- normalizaci

Na této úrovni jsou již data popsána do maximálně možného detailu, bez ohledu na to, jakým způsobem budou fyzicky implementovány v databázi. V jeho popisu vlastností tak nenalezneme ani informace o způsobu uložení dat, ani náhled SQL pro vytvoření jednotlivých objektů (**DDL**[19]).

3.3 Fyzický model

Fyzický model je již úzce spjat s předpokládanou cílovou platformou, v našem případě s určitým databázovým serverem. Kvalitní nástroje **CASE**[20] podporují až několik desítek druhů cílových databázových serverů. Ve fyzickém modelu se může analytik zabývat zvláštnostmi a odlišnostmi jednotlivých platforem, týkajícími se např. návrhu indexů, atd. Nástroje CASE umožní "ušít" implementaci datového modelu „na míru" konkrétní databázové platformě. Vytvořený fyzický model již lze převést do posloupnosti

[19] **Data Definition Language –** jazyk pro definici dat v SQL. Pomocí DDL se vytvářejí databázové objekty, jako jsou tabulky, klíče, omezení, apod.

[20] **Computer Aided System Engineering** - v různých etapách analýzy, návrhu a částečně implementace systémových řešení hrají důležitou roli programové nástroje označované jako CASE. Nabízejí podporu tvorby jednotlivých modelů (v současné době hlavně na základě UML). Některé CASE také zahrnují možnost forward i reverse engineeringu kódu i databáze.

SQL příkazů (v této souvislosti se můžeme setkat také s označením **database kit**[21]), které, pokud jsou spuštěny nad databázovým serverem, vytvoří odpovídající tabulky (a případně indexy) a jejich vzájemné relace. Samozřejmostí je jednoduchý přímočarý (a často automatický) přechod z konceptuálního modelu na fyzický. Nástroj CASE sám provede překlady domén, relací M:N (které je možné používat i na úrovni konceptuálního modelu), klíčů a dalších prvků pro zvolenou cílovou platformu. Možný je samozřejmě i zpětný přechod od fyzického ke konceptuálnímu modelu, stejně jako generování zdrojového kódu aplikace na základě fyzického modelu a zároveň vymodelování fyzického modelu pomocí reverzního inženýrství (reverse engineering), který většina CASE nástrojů nativně umožňuje.

Mezi vlastnosti fyzického modelu patří:

- Specifikace všech tabulek a jejich sloupců
- Cizí klíče určující vztahy mezi tabulkami
- V případě požadavku uživatele se zde může objevit prvek denormalizace
- Zohlednění fyzické implementace může zapříčinit značné rozdíly oproti modelu logickému

Na této úrovni je již zcela specifikováno, jakým způsobem bude původní logický model realizován v rámci databázového schématu. Toto se děje v následujících krocích.

1. Entity jsou zkonvertovány na tabulky
2. Relace mezi entitami jsou zkonvertovány na cizí klíče
3. Z atributů se stávají sloupce
4. Jsou aplikovány další (pro zvolenou platformu specifická) omezení

[21] **Database Kit** – Označení databázový kit je často používáno pro ukončenou sekvenci SQL kódu, která může obsahovat jak kód DDL tak i kód DML. Z tohoto pohledu lze hovořit o kitu schématu – vytvoří objekty databáze a o datovém kytu – naplní tyto objekty konkrétními daty. Spuštění jedné nebo obou sekvencí má pak za následek úplné vytvoření databáze včetně její naplnění daty, a/nebo inkrementální vytvoření nové funkcionality k již existujícímu systému.

ID	ENTITY	ATTRIBUTE	SYNONYM	TYPE	CONSTRAINTS	VALUETYPE	DESCRIPTION	AUTHORITY	VER.	APP.
DE01	Address	AddressLine1	ADDRESS_LINE_1	Text	MaxLength(150), UTF8, not ("";,:"*()~!@$%^*_[]{}<>?|)	DISCRETE	Customer Street address, P.O. box, company name.	Alfa co.DBA	1.0.0	WebForm
DE02	Address	AddressLine2	ADDRESS_LINE_2	Text	MaxLength(150), UTF8, not ("";,:"*()~!@$%^*_[]{}<>?|)	DISCRETE	Customer Apartment, suite, unit, building, floor, etc.	Alfa co.DBA	1.0.0	WebForm
DE05	CarrierLevel	Id	CARRIER_LEVEL_ID	Id	^[1-9]+[0-9]*	ID	ID refering to Carrier level lookup.	Alfa co.DBA	1.0.0	AlfaCore
DE07	City	Id	CITY_ID	Id	^[1-9]+[0-9]*	ID	ID refering to City.	Alfa co.DBA	1.0.0	WebForm
DE08	City	CityName	CITY	Text	MaxLength(60), not ("";,:"*()~!@$%^*_[]{}<>?|)	DISCRETE	Standardized city name.	Alfa co.DBA	1.0.0	WebForm
DE09	Continent	Id	CONTINENT_ID	Id	^[1-9]+[0-9]*	ID	ID refering to Continent.	Alfa co.DBA	1.0.0	AlfaCore
DE10	Continent	ContinentName	CONTINENT	Text	MaxLength(100), UTF8, not ("";,:"*()~!@$%^*_[]{}<>?|)	DISCRETE	Standardized continent name.	Alfa co.DBA	1.0.0	WebForm
DE11	Country	Id	COUNTRY_ID	Id	^[1-9]+[0-9]*	ID	ID refering to Country.	Alfa co.DBA	1.0.0	AlfaCore
DE13	Country	CountryName	COUNTRY	Text	MaxLength(100), UTF8, not ("";,:"*()~!@$%^*_[]{}<>?|)	DISCRETE	Standardized country name.	Alfa co.DBA	1.0.0	WebForm
DE14	Country	CountryAbbreviation	COUNTRY_ABBREV	Text	MinLength (2) MaxLength(2) Capital ASCI	DISCRETE	A2 of ISO 3166 Codes(Countries).	Alfa co.DBA	1.0.0	WebForm
DE17	Customer	Id	CUSTOMER_ID	Id	^[1-9]+[0-9]*	ID	ALFA_CO_DMART unique ID identifying customer.	Alfa co.DBA	1.0.0	AlfaCore
DE18	Customer	LastLogin	LAST_LOGIN	Date/Time	between 1/1/1963 and Now()	DATE	Date when user logged into his account for the last time.	Alfa co.DBA	1.0.0	AlfaCore
DE19	Customer	Race	RACE	Text	ASCI ,MaxLength (20)	SET	Name of the valid race.	Alfa co.DBA	1.0.0	WebForm
DE20	Customer	FirstName	FIRST_NAME	Text	MaxLength(100), UTF8, not ("";,:*()~!@$%^*_[]{}<>?|)	DISCRETE	Standardized personal first name.	Alfa co.DBA	1.0.0	WebForm
DE21	Customer	Gender	GENDER	Text	Male/Female/Unknown/NotSpecified	SET	Specifies gender of the person. Male: M; Female: F; not specified '-'.	Alfa co.DBA	1.0.0	WebForm
DE22	Customer	LastName	LAST_NAME	Text	MaxLength(100), UTF8, not ("";,:*()~!@$%^*_[]{}<>?|)	DISCRETE	Standardized personal last name.	Alfa co.DBA	1.0.0	WebForm
DE32	EmailAddress	EmailAddress	EMAIL_ADDRESS	Text	ASCII,MaxLength 255	DISCRETE	Standardized email address used to contact customers within campaigns.	Alfa co.DBA	1.0.0	WebForm
DE33	EmailAddress	Id	EMAIL_ADDRESS_ID	Id	^[1-9]+[0-9]*	ID	ALFA_CO_DMART unique ID refering to the email address dimension.	Alfa co.DBA	1.0.0	AlfaCore

Příklad 23: Datový slovník popisující konceptuální model entit

ID	SYNONYM	DATA_TYPE	NULLABLE	DEFAULT	USED_IN	IMPLEMENTED	AUTHORITY	VERSION
DE01	ADDRESS_LINE_1	VARCHAR2(150 CHAR)	Yes	"-"	DM_CUSTOMERS, DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE02	ADDRESS_LINE_2	VARCHAR2(150 CHAR)	Yes	"-"	DM_CUSTOMERS, DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE05	CARRIER_LEVEL_ID	NUMBER(10,0)	Yes		DM_CUSTOMERS CARRIER_LEVELS	Yes	Alfa co.DBA	1.0.0
DE07	CITY_ID	NUMBER(10,0)	No		DM_CUSTOMERS, DM_ADDRESSES CITIES	Yes	Alfa co.DBA	1.0.0
DE08	CITY	VARCHAR2(100 CHAR)	No		DM_CUSTOMERS, DM_ADDRESSES CITIES	Yes	Alfa co.DBA	1.0.0
DE09	CONTINENT_ID	NUMBER(10,0)	No		DM_CUSTOMERS, DM_ADDRESSES CONTINENTS	Yes	Alfa co.DBA	1.0.0
DE10	CONTINENT	VARCHAR2(100 CHAR)	No		DM_CUSTOMERS, DM_ADDRESSES CONTINENTS	Yes	Alfa co.DBA	1.0.0
DE11	COUNTRY_ID	NUMBER(10,0)	No		DM_CUSTOMERS, DM_ADDRESSES COUNTRIES	Yes	Alfa co.DBA	1.0.0
DE13	COUNTRY	VARCHAR2(100 CHAR)	No		DM_CUSTOMERS, DM_ADDRESSES COUNTRIES	Yes	Alfa co.DBA	1.0.0
DE14	COUNTRY_CODE	VARCHAR2(2)	No		DM_CUSTOMERS, DM_ADDRESSES COUNTRIES	Yes	Alfa co.DBA	1.0.0
DE17	CUSTOMER_ID	NUMBER(10,0)	No	ID	DM_CUSTOMERS, OFFERS_TO_SEND	Yes	Alfa co.DBA	1.0.0
DE18	LAST_LOGIN	DATE	No	Sysdate	DM_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE19	RACE	VARCHAR2(50)	No	"-"	DM_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE20	FIRST_NAME	VARCHAR2(100CHAR)	Yes		DM_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE21	GENDER	VARCHAR2(1)	Yes	"-"	DM_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE22	LAST_NAME	VARCHAR2(100CHAR)	No	"Customer"	DM_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE32	EMAIL_ADDRESS	VARCHAR2(256)	Not	"-"	DM_CUSTOMERS, OFFERS_TO_SEND EMAIL_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE33	EMAIL_ADDRESS_ID	NUMBER(10,0)	No	Id of default email address	DM_CUSTOMERS, OFFERS_TO_SEND EMAIL_ADDRESSES	Yes	Alfa co.DBA	1.0.0

Příklad 24: Datový slovník popisující obecnou implementaci atributů v systému Alfa co. datamartu

ID	SYNONYM	DATA_TYPE	NULLABLE	DEFAULT	USED_IN	IMPLEMENTED	AUTHORITY	VERSION
DE01	ADDRESS_LINE_1	VARCHAR2(150 CHAR)	Yes		DM_CUSTOMERS,	Yes	Alfa co.DBA	1.0.0
DE01	ADDRESS_LINE_1	VARCHAR2(150 CHAR)	No	"-"	DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE02	ADDRESS_LINE_2	VARCHAR2(150 CHAR)	Yes		DM_CUSTOMERS,	Yes	Alfa co.DBA	1.0.0
DE02	ADDRESS_LINE_2	VARCHAR2(150 CHAR)	No	"-"	DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE05	CARRIER_LEVEL_ID	NUMBER(10,0)	Yes		DM_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE05	CARRIER_LEVEL_ID	NUMBER(10,0)	No	SEQ_CARRIER_ LEVELS	CARRIER_LEVELS	Yes	Alfa co.DBA	1.0.0
DE07	CITY_ID	NUMBER(10,0)	Yes		DM_CUSTOMERS,	Yes	Alfa co.DBA	1.0.0
DE07	CITY_ID	NUMBER(10,0)	No		DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE07	CITY_ID	NUMBER(10,0)	No	SEQ_CITIES	CITIES	Yes	Alfa co.DBA	1.0.0
DE08	CITY	VARCHAR2(100 CHAR)	Yes		DM_CUSTOMERS,	Yes	Alfa co.DBA	1.0.0
DE08	CITY	VARCHAR2(100 CHAR)	No		DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE08	CITY	VARCHAR2(100 CHAR)	No		CITIES	Yes	Alfa co.DBA	1.0.0
DE09	CONTINENT_ID	NUMBER(10,0)	Yes		DM_CUSTOMERS,	Yes	Alfa co.DBA	1.0.0
DE09	CONTINENT_ID	NUMBER(10,0)	No		DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE09	CONTINENT_ID	NUMBER(10,0)	No	SEQ_CONTINEN TS	CONTINENTS	Yes	Alfa co.DBA	1.0.0
DE10	CONTINENT	VARCHAR2(100 CHAR)	Yes		DM_CUSTOMERS,	Yes	Alfa co.DBA	1.0.0
DE10	CONTINENT	VARCHAR2(100 CHAR)	No		DM_ADDRESSES	Yes	Alfa co.DBA	1.0.0
DE10	CONTINENT	VARCHAR2(100 CHAR)	No		CONTINENTS	Yes	Alfa co.DBA	1.0.0

Příklad 25: Finální (fyzická) implementace atributů v systému Alfa co. datamartu (způsob A)

		COUNTRIES					DM_CUSTOMERS				
ID	SYNONYM	NULL.	DEFAULT	IMPL.	AUTH.	VER.	NULL.	DEFAULT	IMPL.	AUTH.	VER.
DE11	COUNTRY_ID	No	SEQ_COUNTRIES	Yes	Alfa co.DBA	1.0.0	Yes		Yes	Alfa co.DBA	1.0.0
DE13	COUNTRY	No		Yes	Alfa co.DBA	1.0.0	Yes		Yes	Alfa co.DBA	1.0.0
DE14	COUNTRY_CODE	No		Yes	Alfa co.DBA	1.0.0	Yes		Yes	Alfa co.DBA	1.0.0
DE17	CUSTOMER_ID						No	SEQ_CUSTOMERS	Yes	Alfa co.DBA	1.0.0
DE18	LAST_LOGIN						No	Sysdate	Yes	Alfa co.DBA	1.0.0
DE19	RACE						No	"-"	Yes	Alfa co.DBA	1.0.0
DE20	FIRST_NAME						Yes		Yes	Alfa co.DBA	1.0.0
DE21	GENDER						Yes	"-"	Yes	Alfa co.DBA	1.0.0
DE22	LAST_NAME						No	"Customer"	Yes	Alfa co.DBA	1.0.0

Příklad 26: Finální (fyzická) implementace atributů v systému Alfa co. datamartu (způsob B)

Shrnutí

Pro potřeby návrhu a rozvoje informačního systému, pochopení informace v něm obsažené a jeho procesů a následné uchování či sdělení této znalosti jsou klasická metadata zcela nevyhovující. Téměř vždy je nutné pracovat s nějakou vyšší úrovní abstrakce, raději než přepisovat a vysvětlovat části konkrétního kódu. V předchozí kapitole byly popsány některé základní metody reprezentace systému jako celku (jeho prvků a vzájemných vazeb). Ideální systém by měl být téměř ve všech úrovních takto zdokumentovaný. V úvodu zmíněný přístup "Time to market", kterým se bohužel řídí vývoj aplikací v komerční sféře, patrně takovou systematičnost a preciznost neumožní.

Pokud již pracujeme s nějakým systémem, je však žádoucí alespoň některé základní koncepty doplnit, chybí-li. Nejjednodušším krokem je získání fyzického modelu systému (v případě přidělení patřičných přístupových práv, lze toto aplikovat i na zcela neznámou strukturu systému Beta&sons) pomocí reverzního inženýrství. Postup, při kterém skutečně vycházíme z původní ontologie přes konceptuální model až po fyzickou implementaci, lze aplikovat spíše u projektů (systémů), které začínají tzv. "na zelené louce" (vzniká zcela nový systém, jehož komerční spuštění je teprve plánováno, je zde prostor pro patřičnou rozvahu, analýzu entit, apod.) Vznik datového slovníku a jeho udržování by však mělo být samozřejmostí ve všech případech. Ten je nezbytný již jen kvůli tomu, abychom dokázali říci, které entity a jejich atributy v systému uchováváme a jakým způsobem jsou použity.

Pokud zvažujeme následnou integraci systémů, kterou hodláme v maximální míře standardizovat a následně automatizovat, je existence modelů a datového slovníku, minimálně na straně cílového systému, nezbytností, pokud se ovšem nehodláme spokojit pouze s tabulkami, které popisují jednotlivé transformační kroky, jak bylo představeno v kapitole "Transformace informací ze zdrojového do cílového systému" (2).

4. Sémantické mapování modelů různých zdrojů

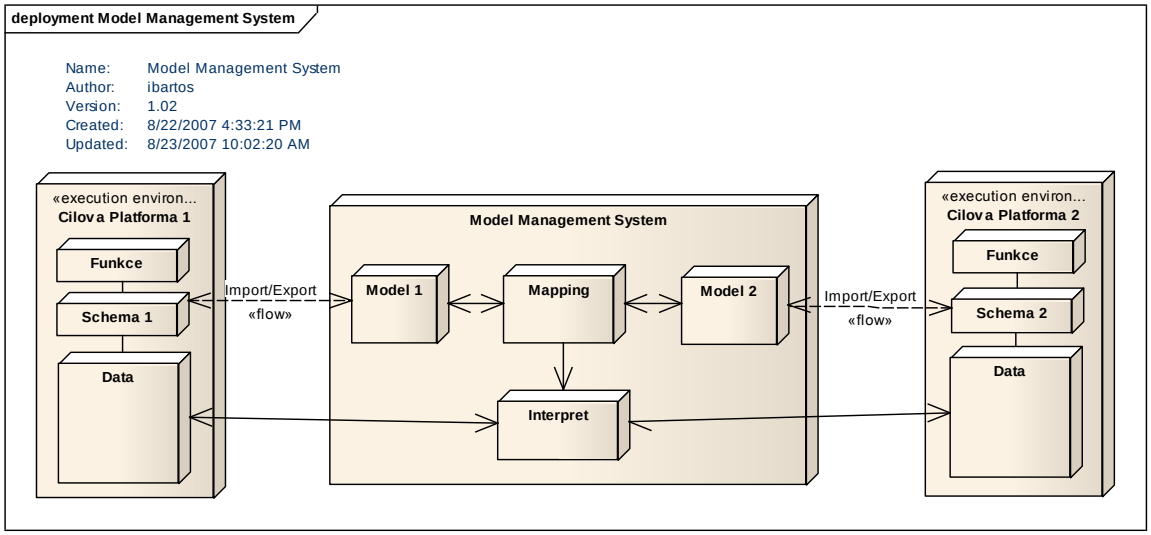
Informační systémy obecně vyžadují design, integraci a údržbu velmi komplexních artefaktů. Jedním z nich jsou právě databáze. Aby s nimi bylo možné efektivně provádět výše vyjmenované aktivity, je třeba využít nástroj, který umožní manipulaci s jejich formálním popisem, tedy *model* (viz. předchozí kapitola). Tato manipulace často zahrnuje design transformací mezi jednotlivými modely, která vyžaduje nějakou formu explicitního vyjádření resp. mapování (*mapping)*. Mapování popisuje souvislosti a vzájemné vazby, ve kterých figurují jednotlivé modely. Příklady mapování mohou být následující:

- Mapování mezi definicí třídy a relačním schématem.
- Mapování mezi XML schématy pro řízení překladu zpráv.
- **Mapování mezi zdroji dat a zprostředkovatelským schématem pro řízení integrací heterogenních dat.**
- Mapování mezi databázovým schématem a plánem jeho budoucí implementace (Releasem) za účelem řízení migrace dat.
- Mapování mezi entity relationship (ER) modelem a SQL schématem (logickým a fyzickým) pro navigaci v databázi (BERNSTEIN, 2003).

Nejvhodnějším způsobem, kterým lze takový nástroj vytvořit je přetransformovat jednotlivé komponenty do objektově orientovaného vyjádření a následně manipulovat s modely a jejich mapováními pouze na úrovni této reprezentace. Onou manipulací rozumějme návrh mapování mezi modely, generování modelu z jiného modelu včetně mapování mezi nimi, úpravy jednotlivých modelů a mapování, jejich interpretace a následné generování kódu.

Modely a mapování jsou vyjádřeny jako abstrakce (viz. předchozí kapitola), které jsou implementovány a následně spravovány v systému správy modelu (**Model**

Management System[22]). Tento přístup je definován obecně a jeho jednotlivé koncepty mohou být používány napříč všemi případnými modelovými prostředími jako je UML, ER (EER) nebo XML. Jejich následná implementace, která vychází z reprezentace modelů založených na těchto konceptech, je taktéž platformově nezávislá.

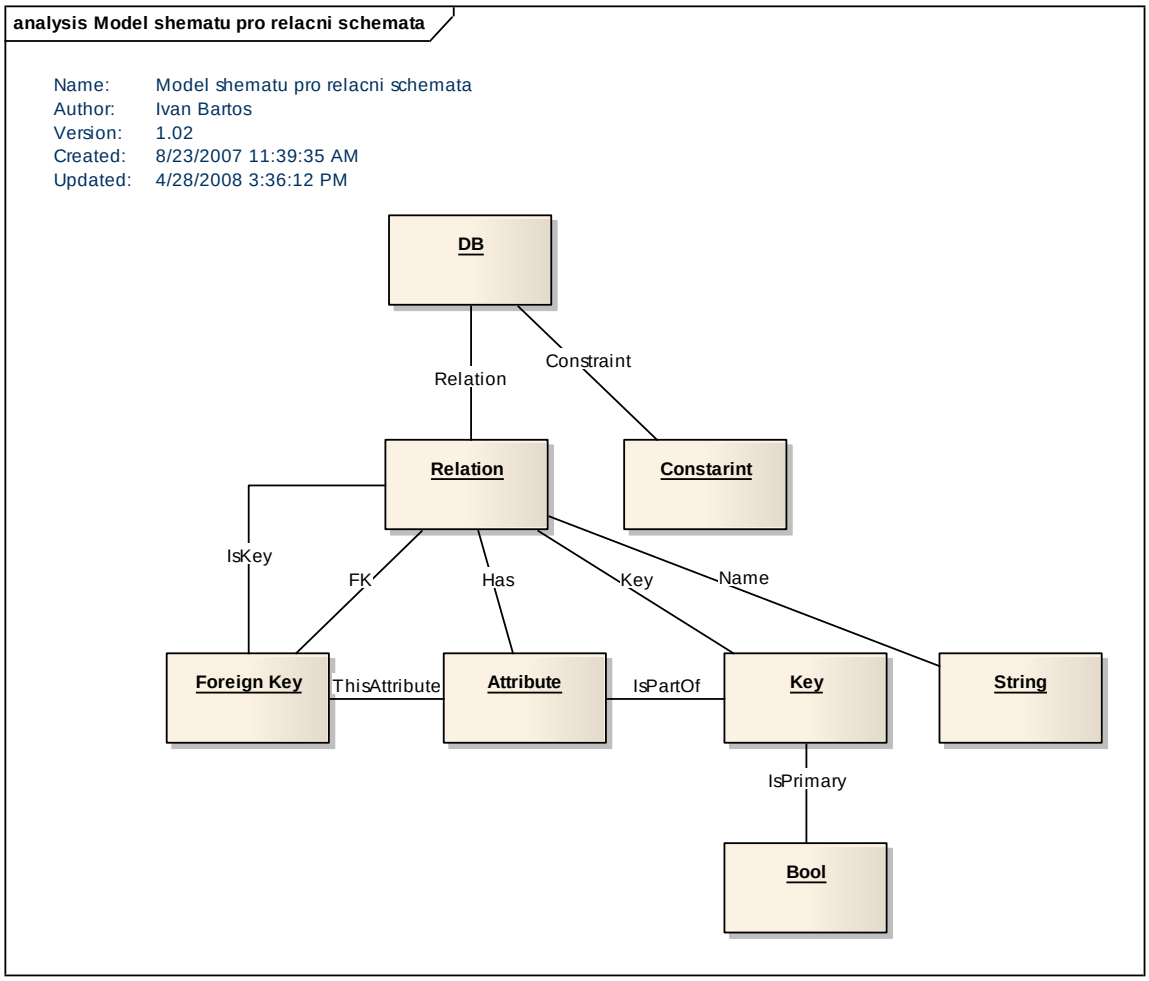


Obrázek 11: Obecná (high level) architektura správy modelů

4.1 Modely

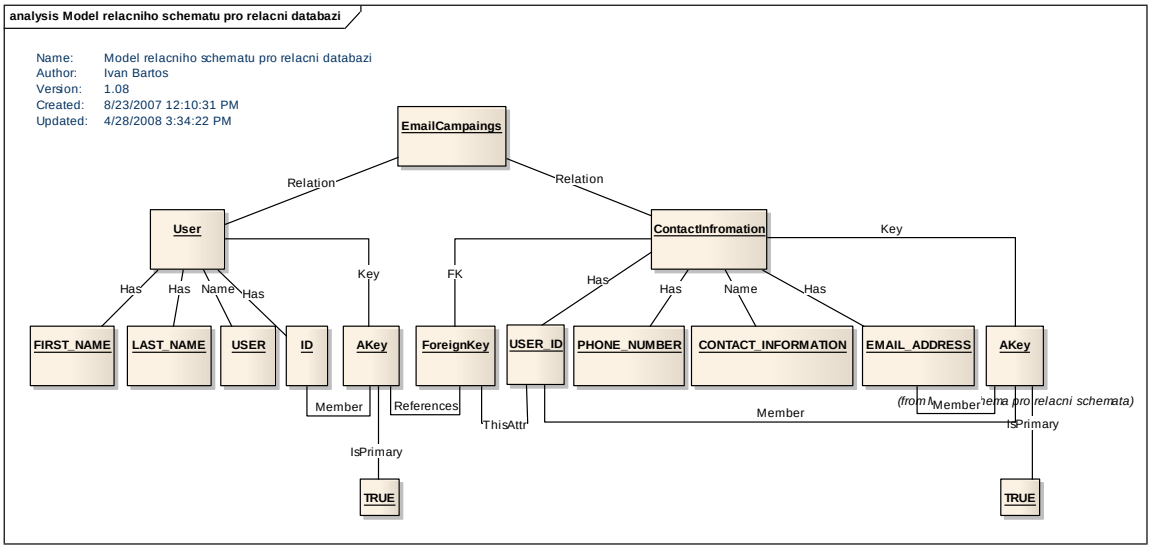
Vhledem k tomu, že se tato práce zabývá právě databázemi, budou i následující příklady popisovat "high level" modelování a mapování relačních schémat (entit, atributů). Fragment obecného vyjádření relačního schématu může vypadat například takto.

[22] **Model Management System (MMS)** – Systém pro správu modelů. Jedná se specifický softwarový systém (nástroj), který podporuje operace pro práci s modely, správu databází, umožňuje kooperativní práci s modely ve více úrovních abstrakce.



Obrázek 12: Model schématu pro relační schémata

Jednotlivé uzly reprezentují třídy a vzájemné vztahy pak typy relací. Model popisuje, které relace, atributy, integritní omezení, typy atributů a cizí klíče v relačním schématu existují. Následující obrázek ilustruje instanci schématu z obrázku 12 tak, jak by mohla být aplikována pro vzorovou firmu Alfa co. z našeho příkladu.



Obrázek 13: Model instance schématu relační databáze

4.2 Manipulace s modely

S modelem nebo jeho objekty lze provádět celou řadu operací. Tyto obecné algebraické operace jsou použity v jednotlivých dotazech za účelem manipulace s celými modely nebo jejich mapováními. Jejich základním předpokladem je možnost jejich vnořování (skládání) tedy, že každá operace vrátí opět model, a jejich obecnost, která umožní aplikaci v jakémkoliv modelu schématu, tj. lze ji použít v jakékoliv aplikaci určené ke správě modelu. Vzhledem k tomu, že vzorový projekt předpokládá nulovou možnost zásahu do cílového schématu databáze Alfa co., věnujeme těmto operacím jen minimální pozornost. Podrobnější informace lze najít v odkazovaných zdrojích.

Operace s modely:

- *Create* (Vytvoř) - Vytvoří model tak, že vytvoří jeho kořen (*root*). *CreateModel* pak vytvoří z dané šablony (*template*) kompletní strukturu celého modelu (ten je dále třeba parametrizovat za účelem vytvoření příslušné instance k takovému modelu.

- *Update* (Aktualizuj) - pracuje s nejmenšími prvky modelu, tj. s jednotlivými objekty, jejich vlastnostmi a vzájemnými vztahy.

- *Delete* (Vymaž) - Maže kořen (*root*) a propaguje se dlále na jeho objekty. Lze použít možnost kaskádovitého mazání (propagovat i na obsažené objekty), nebo naopak restrikce (nemazat objekty, které jsou stále v relaci s jinými objekty).

- *Select* (Vyber) - Aplikujeme-li ho na model, získáme množinu submodelů, které odpovídají kvalifikaci selectu. Lze jej použít i při porovnávání struktur a obsahu jednotlivých submodelů, pokud je kvalifikován například množstvím submodelů.

- *Project* (Přenes) - Projektuje model do submodelu jednotlivých schémat. To může zapříčinit neplatnost některých tříd a vztahů, pokud v daném schématu neexistují. Ty se nepřenesou a model není již dále spojitý.

- *SetDifference* (UrčiRozdíl) - Vrátí objekty z jednoho modelu, které se nevyskytují v modelu druhém.

- *ApplyFunction* (PoužijFunkci) - Použije funkci na veškeré objekty modelu (např. vložit prefix před veškeré názvy, prosadit mezi objekty určitá omezení atp.).

- *Copy* (Kopíruj) – Zkopíruje veškeré objekty obsažené v modelu (ve vztazích v rámci modelu lze specifikovat, zda daný objekt skutečně zkopírovat, či pouze kopírovat propojení (link) na tento objekt).

- *(Model)Enumeration* (Výčet) – Umožňuje manipulaci s jednotlivými objekty modelu, nebo jejich skupinami individuálně

(BERNSTEIN, HALEVY, POTTINGER, 2000).

4.3 Mapování Modelů

Hlavními cíli správy modelů je zajistit vhodnou podporu pro řízení změn v rámci modelu a, což je pro naši potřebu důležitější, zajistit mapování objektů mezi dvěma odlišnými modely (např. modely jednotlivých modulů jednoho systému nebo zcela odlišné systémy). Manipulace s mapováním probíhá podobně jako manipulace s prvky samotných modelů (BERNSTEIN, HALEVY, POTTINGER, 2000), což může být například zkopírování mapování, selekce z mapování, jeho smazání atp. Jednotlivá mapování se skládají ze spojení mezi instancemi dvou modelů, které mohou být zároveň instance zcela různých schémat (např. relační a XML DTD). Jedno mapování může pomocí jazyka pro budování komplexnějších výrazů spojovat skupiny objektů v jednom modelu se skupinou objektů v modelu druhém. Například mapování mezi relačním schématem Alfa co. AM1 a Beta&sons BM2 může specifikovat, že pohled (*view*) AV1 na model AM1 odpovídá pohledu BV1 na BM2. Definice jednotlivých pohledů jsou součástí mapování. Jak již bylo naznačeno, k modelu se může vázat jazyk, umožňující vytvářet komplexní výrazy nad jednotlivými jeho prvky, jako je třeba SQL, nebo různé matematické operace. Takový jazyk popisuje vnitřní logiku mapování. Posledním předpokladem pro mapování je možnost jejich vnořování (skládání). To umožňuje jejich recyklaci, tj. mapování z jednoho modelu může být použito jako součást mapování v modelu druhém.

Na základě předchozích specifikací si lze pro naše potřeby obecné mapování modelů definovat například takto:

- Mapování modelu, které spojuje model M1 s modelem M2, charakterizuje společný kořenový prvek (tzv. *rootElement*). Ten má dvě jednoznačné vlastnosti vztahu, které ukazují na kořenové objekty M1 a M2. V praxi se tyto vlastnosti označují jako *domainRoot* a *rangeRoot* a identifikují modely, které se váží k mapování.
- Každý objekt mapování má vlastnost, označovanou jako *Expr* (výraz), která vyjadřuje potřebnou transformaci mezi objekty v M1 a M2. Informace obsažená v Expr, není předem definovaná. Může to být třeba CQL popis transformace (tedy string), SQL, nebo funkce.
- Každé mapování v obecném modelu mapování má dvě vlastnosti vztahu, které se nazývají *domain* a *range.* Ty zahrnují všechny objekty z M1 a M2, tj. ty, které jsou odkazovány v rámci *Expr* (BERNSTEIN, HALEVY, POTTINGER, 2000).

Existuje celá množina dalších vlastností mapování, které lze vyjmenovat. Pro zachycení potřebné informace o transformaci dat/informace mezi dvěma systémy jsou však tyto specifikace postačující.

4.4 Interpretace mapování

Mapování lze specifikovat na různých úrovních. Jedním extrémem může být specifikace úplného sémantického vztahu mezi dvěma modely, na straně druhé pak mapování čistě strukturální, které specifikuje pouze *domain* a *range***,** tzn. objekty ze dvou modelů jsou v relaci a neexistuje zde žádná sémantika mapování (*Expr* je prázdná).

Interpretace sémantiky mapování přináší řadu výhod. Pokud je mapování plně interpretováno (vlastnost *Expr* obsahuje formuli určitého matematického systému – logiku, algebru, nebo gramatiku), může být následně přetransformováno do programu,

který provádí transformaci dat z instance jednoho modelu do instance modelu druhého, a to i automaticky. Na sémantice mapování velmi závisí i případná inverze a kompozice mapingu nebo porovnávání modelů. Na druhou stranu je třeba si uvědomit, že využití sémantiky mapování vnáší do datového modelu již konkrétní definice operací a tím i značnou komplexitu na úkor obecnosti. Míra uplatnění sémantiky tedy závisí na požadavcích, které jsou na model kladeny, resp. na konečný účel takovéhoto modelu.

4.5 Metodika mapování – operace Match

Prvním krokem mapování je vyhledání odpovídajících elementů v obou systémech resp. jejich modelech a vytvoření jejich spojení tedy mapování. Tuto operaci lze označit právě jako *Match*. Jakmile je toto mapování vytvořeno, je třeba důkladně analyzovat sémantiku každého elementu ve zdroji (analýza a profilace dat v systému Beta&sons) a vytvořit transformaci (*Expr*), která prvek synchronizuje do sémantiky cíle (datový slovník datamartu Alfa co.) (RAHM, BERNSTEIN, 2001). Pokud toto aplikujeme na náš příklad z kapitoly 4.3, lze *Match* definovat jako operaci, která na vstupu zpracovává modely AM1 a BM2 a jako výstup vrací mapování mezi nimi. Každé mapování, které je výsledkem *Match* specifikuje určité prvky modelu AM1 a logicky korespondující prvky modelu BM2, kde je sémantika této korespondence vyjádřena ve vlastnosti mapování *Expr*.

Zjednodušeně lze *Match* přirovnat k *Join* operaci používané v relačních databázích. Podobně jako *Match,* i *Join* je binární operace, která určuje odpovídající si entity/atributy na základě jejich vstupních operandů. Na druhou stranu lze objevit i řadu rozdílů. Zatímco *Match* pracuje s metadaty modelu (schématem), *Join* pracuje s daty konkrétní instance. Na rozdíl od *Join,* umožňuje *Match* spojovat více prvků z obou modelů (nejedná se o paralelu s funkci concatenate, kterou podporuje například SQL). Poslední odlišností, kterou je třeba si uvědomit, je schopnost *Match* produkovat pro stejné prvky různé mapovací Expr, zatímco *Join* produkuje jedinou *Expr* závaznou pro veškeré vstupní prvky.

4.6 Různé přístupy při operaci Match

Implementace *Match* může využívat různé algoritmy. Jejich volba se odvíjí od domény dané aplikace a typu modelovaného schématu. Tyto algoritmy lze používat jednotlivě, mapování je vypočítáno na základě jednoho kritéria, nebo je lze kombinovat, a to buď jako hybridní algoritmus spojující dvě kritéria v jedno (např. jméno a datový typ), nebo kombinovat více výsledků vyprodukovaných různými algoritmy. Jednotlivé algoritmy lze zevrubně rozdělit například následovně:

- *Instance X Schéma* - *Match* může zvažovat data instance (obsah), nebo pouze informaci na úrovni schématu (modelu).
- *Element X Struktura* - *Match* může zvažovat jednotlivé prvky schématu (atributy atp.), nebo kombinaci prvků (komplexní struktury schématu).
- *Jazyk X Omezení* - *Match* může použít lingvistické přístupy (jména a popisky elementů modelu), nebo přístupy zvažující omezení (klíče a vztahy).
- *Kardinalita* – Každý prvek ve výsledném mapování může odpovídat jednomu nebo více elementům (1:1,1:n, n:m).
- *Přidané informace* – Společně se vstupními modely AM1 a BM2 lze využít i další externí informace, jako jsou slovníky, globální schémata, knihovny předchozích rozhodnutí, nebo zásah uživatele (RAHM, BERNSTEIN, 2001).

Předmětem této práce však není podrobný rozbor těchto jednotlivých přístupů. Následující příklad proto ilustruje různé výsledky dosažené pomocí různých hybridních operací *Match*.

AM1 model. B&S_USERS	**AM2 model.DM_CUSTOMERS**
USER_ID	CUSTOMER_ID
FIRST_NAME_TXT	CUSTOMER_NAME
LAST_NAME_TXT	EMAIL_ADDRESS
EMAIL	RACE
RACE_WHITE_TXT	
RACE_BLACK_TXT	

Příklad 27 : Vstupní a cílové prvky modelu pro operaci *Match*

Úplná specifikace výsledků operace Match na úrovni modelu schématu/ /Element/Jazyk/Kardinalita. Kvalifikované mapování by tedy mohlo být:

```
"B&S_USERS.USER_ID = DM_CUSTOMERS.CUSTOMER_ID" and
"concatenate(B&S_USERS.FIRST_NAME_TXT, B&S_USERS.LAST_NAME_TXT) =
DM_CUSTOMERS.CUSTOMER_NAME", and " B&S_USERS.EMAIL = B&S_USERS.EMAIL_ADDRESS " and "
concatenate (B&S_USERS.RACE_WHITE_TXT, B&S_USERS.RACE_BLACK_TXT) end = DM_CUSTOMERS
RACE "
```

Příklad 28 : Mapování získané operací *Match* v kombinaci Schema/Element/Jazyk/Kardinalita (SQL)

Jak ukazuje předchozí příklad, použití této kombinace přístupů není pro "spolehlivé" namapování entit Customer a User postačující (spojení dvou dle názvu souvisejících atributů, které platí pro JménoUživatele pro namapováni Atributu Rasy nefunguje). O řád komplexnější a z pohledu transformace informace mezi systémy správnější je kombinace následující tj. Instance/Element/Kardinalita/PřidanáInformace (v tomto případě slovník obsahující názvy pro jednotlivé rasy).

```
"B&S_USERS.USER_ID = DM_CUSTOMERS.CUSTOMER_ID" and
"concatenate(B&S_USERS.FIRST_NAME_TXT, B&S_USERS.LAST_NAME_TXT) =
DM_CUSTOMERS.CUSTOMER_NAME", and " B&S_USERS.EMAIL = DM_CUSTOMER.SEMAIL_ADDRESS" and
"case when B&S_USERS.RACE_WHITE_TXT='Y' then 'WHITE' when B&S_USERS.RACE_BLACK_TXT ='Y'
then 'BLACK' else '-' end = DM_CUSTOMER RACE "
```

Příklad 29 : Mapování získané operací *Match* v kombinaci Instance/Element/Jazyk/Kardinalita/Přidaná Informace (SQL)

Poznámka: V posledním případě lze uvažovat i Match pomocí omezení, pokud jsou ID atributy zároveň primárními klíči.

4.7 Vizualizace mapování v modelu a následné generování pravidel pro mapování

Aktuální problém všech modelovacích technik a nástrojů je těžké určení míry jejich preciznosti. Na jedné straně by měl být model abstraktní, lehce uchopitelný a srozumitelný. Na straně druhé stojí model, který obsahuje všechny podrobnosti, může být použit pro generování kódu. Ten již není tak přehledný a uchopitelný. (RICHTA, 2010)

Nejčastěji je mapování definováno pomocí přirozeného jazyka, což inklinuje k jisté vágnosti, která je často vyvažována příklady, které definici vysvětlují. Pokud je takové mapování prezentováno, bývá definice zobrazena například pomocí řady barevných šipek a podobně. Potřebné komplexity a jednoznačnosti nelze pomocí přirozeného jazyka dosáhnout. Zároveň je tato definice nevhodná pro spravování většího množství modelů v rámci procesu vývoje aplikací (HAUSMANN, KENT, 2003).

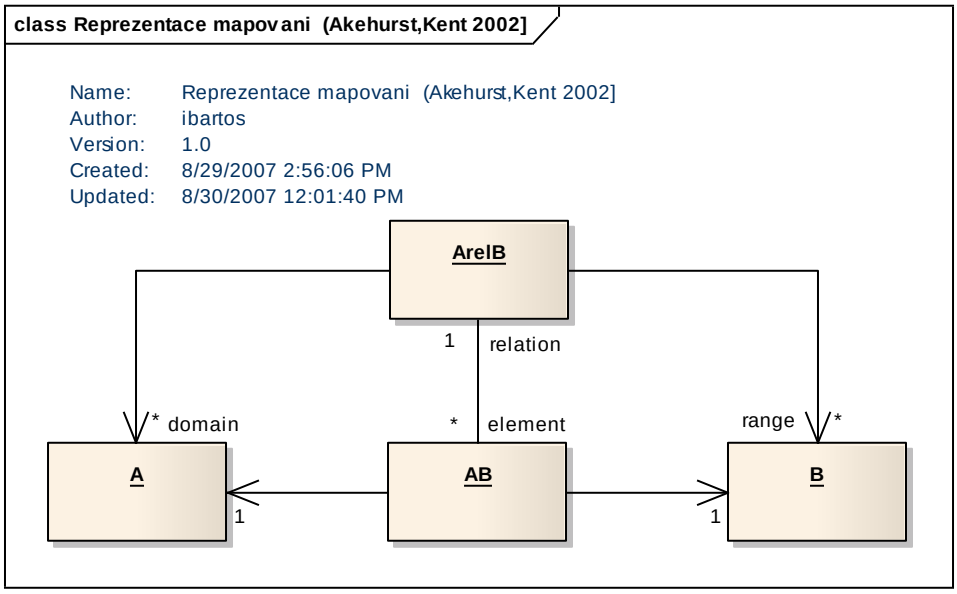
Dalším způsobem modelování a zobrazení mapování mohou být transformační algoritmy určené pro generování kódu. Ty určují, jakým způsobem jsou relace mezi jednotlivými elementy vypočítávány. Tyto metody jsou na jednu stranu vyhovující pro strojové zpracování, přesto je k jejich pochopení opět potřeba příkladů, nebo hlubšího zkoumání daného algoritmu. Jejich další nevýhodou je značná zaujatost jedním směrem, čímž neumožňují zpětné generování zdrojového elementu ani možnou rekonciliaci(synchronizaci) modelu.

Jazyk UML obsahuje prvek Abstrakce, která nativně obsahuje mapování. Tento prvek zachycuje myšlenku mapování mezi objekty modelů, ale neumožňuje dostatečně zachytit syntaxi (snad jen přerušovaná čára pro Závislost (Dependency)), ani sémantiku takového mapování. Jednotlivé stereotypy Abstrakce (Trace, Derive, Realize a Refine) mohou nahradit dodatečnou informaci o sémantice, nikoliv však o jejím použití.

Pro účely vyjádření vztahu, resp. mapování mezi objekty dvou databází je nejvhodnější modelování v jazyce UML. Zde navrhované řešení představují ve své studii Ankehurts a Kent (AKENHURST, KENT, 2002). Řešení předpokládá použití vztahu ArelB a dvojtřídu (AB) pro každé zamýšlené mapování elementů a následně jazyk

OCL[23] pro specifikaci k definici domain, range a dalších omezení daného vztahu (všimněme si zde analogie k objectProperties v jazyce OWL v kapitole 3.1).

Vztah je zde vyjádřen objektem, který je asociován s množinami párů. Relace má opět domain a range. Veškeré objekty, které jsou asociovány s touto dvojicí představující relaci, musí být voleny z domain a range. Tento koncept následně vede k zobrazení mapování v diagramu tříd. Asociace lze řídit pouze ze vztahu ArelB tak, že A i B zůstávají definicí vztahu nedotečeny.



Obrázek 14: Reprezentace mapování modelem tříd (AKENHURST, KENT, 2002)

Pro nejčastější případy pak existují předpřipravená OCL omezení. (Pro důkladné pochopení syntaxe a funkce jazyka OCL je podmínkou nastudování příslušné specifikace OMG: Object Constraint Language (OMG, 2010)) To umožňuje poměrně přesnou definici mapování (včetně vnořených mapování). Vyjádření vztahů tímto způsoben, resp. využití tohoto přístupu pro modelování mapování má však také své limitace. Uživatel musí vytvořit vzorové třídy a asociace pro veškeré existující relace, zvolit jim odpovídající názvy, převézt předpřipravená OCL omezení a modifikovat je tak, aby odpovídala novým jménům. Tato práce je poměrně náročná, což zvyšuje riziko chyb, ale

[23] **Object Constraint Language (OCL) -** jazyk pro specifikaci vstupních a výstupních podmínek a invariantů v jednotlivých diagramech. Doplňuje UML přesnými popisy různých integritních omezení.

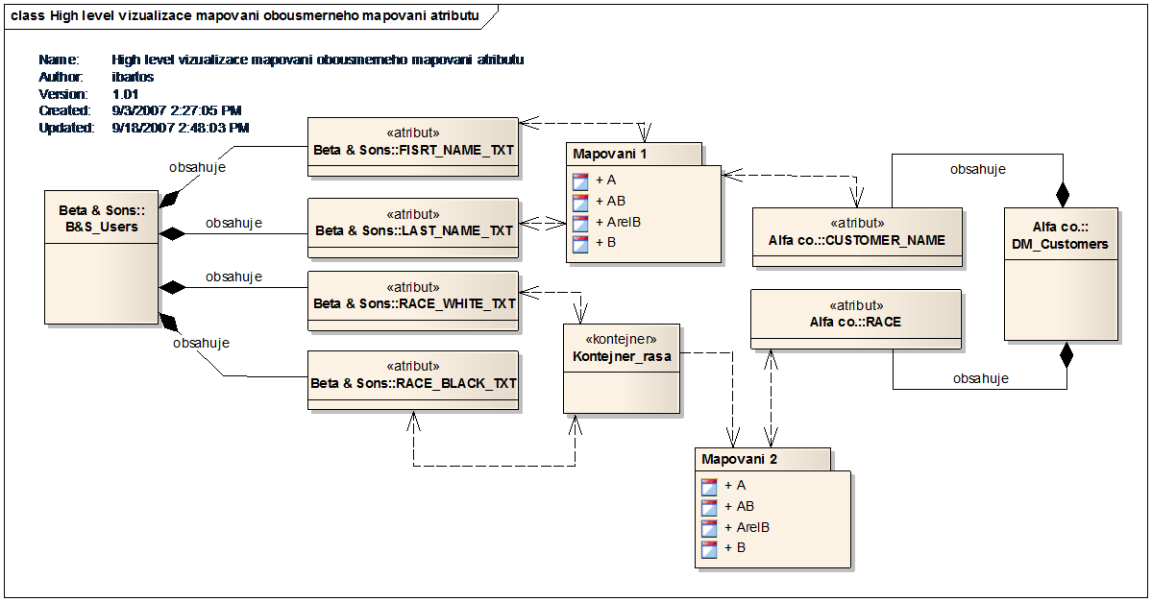
výsledkem bude komplexní model vztahu (mapování) vhodný pro automatické zpracování (generování kódu) s poměrně dobrou čitelností pro případného uživatele. OCL není samozřejmě jediným způsobem formální specifikace integritních omezení. V současné době se ale zdá být nejperspektivnější (RICHTA, 2010 (2)). Nevýhodou tohoto zobrazení zůstává pouze skutečnost, že nelze pracovat s podmnožinami z domain a range, jelikož třídy A a B zahrnují všechny jejich prvky. V modelu relačního schématu však předpokládáme A a B jako množiny všech hodnot daného atributu (AKENHURST, KENT, 2002), proto není tato "nedokonalost" kritická.

context ArelB **inv**:
domain->includesAll(elements.a->asSet) and
range->includesAll(elements.b->asSet)

Je třeba specifikovat omezení, aby žádná z dvojic vztahu neodpovídala dvěma stejným prvkům z domain a range. Tím je vyloučena redundance a značně to zjednodušuje konstrukci a definici SQL dotazů, které budou výsledkem tohoto mapování.

context ArelB **inv**:
elements->forAll(a, b |
(e.a = f.a and e.b = f.b) implies e = b)

Popsaný OCL koncept lze zabalit do balíčku s definicí mapování a jednotlivé, již konkrétní mapování na úrovni atributů vizualizovat následujícím způsobem (LUJÁN-MORA, VASSILIADIS, TRUJILLO, 2004).



Obrázek 15: Vizualizace mapování na úrovni atributů, umožňující následný strojový překlad

Související generování mapování například do jazyka SQL z této UML/OCL notace je prakticky bezproblémový, stejně tak vytvoření OCL pravidla pomocí reverzního inženýrství. Podobný přechod funguje i mezi platformově nezávislým konceptuálním modelem v notaci UML/OCL do SQL a pro zpětný převod logického modelu z SQL do OCL. Tyto přechody jsou nativní a nejsou programově náročné (RICHTA, 2010 (2)).

Shrnutí

V řešení vzorového projektu se může takto komplexní sémantické mapování a jeho modelování jevit jako práce navíc. Přesto přináší do celého konceptu integrace zásadní prvky a je tak jakýmsi završením celého procesu. Vlastnosti těchto modelů a možnosti manipulace s nimi umožňují jejich znovupoužití při řešení podobných úloh. Vzniklé definice mapování je možné dále modifikovat, protože již jednou definované *Range* náleží "mateřskému" systému (datamart Alfa co.), jehož změny jsme v projektu neočekávali. Minimálně jedna strana relace je tedy v určitém časovém horizontu neměnná. To usnadní práci při podobných integračních projektech v budoucnosti. Popis mapování na námi zvolené úrovni je jednak srozumitelný i pro komunikaci (není závislí na znalosti implementační platformy) a naopak umožňuje pohodlný přechod do

konkrétního jazyka (různé varianty jazyka SQL jsou podporovány nejčastěji využívanými databázovými řešeními).

Závěry

V této práci byly rozebrány jednotlivé kroky, které je vhodné provést při realizaci požadavku na integraci dat ze zdrojového systému do systému cílového. Zde popisované metodologie se v praxi osvědčily a byly úspěšně použity například při integraci OLTP zdrojů trak.com (nyní monstertrak.com) v roce 2006, fastweb.com v roce 2007 a jednotlivých komunit uživatelů serverů military.com v roce 2008-2009 a affinitylabs.com v roce 2009 do OLAP struktury analytického datamartu firmy Monster Worldwide pro účely řízení masivních uživatelských kampaní **EMM**[24] nástrojem UNICA Affinium Suite.

Výčet a specifikace popisovaných metod nejsou jistě kompletní a pro zvolenou implementaci do procesů firmy či informační instituce ani závazné. V práci je popsána jedna z možných cest, nikoliv však jediná. Konkrétní projekty vyžadují často specifické přístupy. Pro některé projekty je hloubka profilace dat, jejich následná transformace i způsob uložení v modelech jistě práce navíc. Pro komplexnější projekty může být tento přístup nepostačující, analýza se může jevit jako povrchní a ukázkové transformace jako příliš jednoduché (je možné, že požadované transformace lze popsat pouze pomocí celých stránek kódu atp.). Jednotlivé metody by ovšem měly být minimálně zváženy. Neznalost vlastních dat či struktur, ve kterých jsou uložena, je kritickým problémem nejen pro rozvoj systému, ale i pro jeho každodenní fungováni. Vybrané modely, existence alespoň nějakého datového slovníku a způsobu průběžné kontroly kvality dat v systému jsou skutečně nezbytným minimem. Bez těchto znalostí je jakákoliv budoucí integrace prakticky nesplnitelným úkolem.

V modelovém případě cíleného oslovování uživatelů může vézt špatné pochopení, mylná interpretace a následná chybná integrace dat nejen k celkovému neúspěchu při propagaci nabízených titulů či periodik, ale i, a to je pro růst jakékoliv společnosti

[24] **EMM (Enterprise Marketing Management) –** Kategorie softwaru pro marketingové operace, ve kterém jsou jednotlivé jejich procesy kompletně spravovány (end to end řešení). To zahrnuje procesy výběru a škálování cílových skupin, designem jednotlivých kampaní, jejich plánováním a spuštěním, monitoringem, měřením reakcí na tyto kampaně a případným reportováním úspěšnosti.

kritické, k následnému úbytku zákazníků (uživatelů), kteří se budou cítit obtěžováni nabídkami, které mohou být chybně personalizovány, nebo "pouze" vůbec neodpovídají jejich čtenářskému profilu.

Je třeba si uvědomit fakt, který byl zmíněn již v samotném úvodu. Kvalita dat, jejich čitelnost, platné souvislosti a správná interpretace je v prostředí marketingu klíčová. Kvalitní data, která vypovídají (i historicky) o stavech trhu, chování uživatele nebo jiných skutečnostech reálného světa, jsou ceněnou komoditou a v konkurenčním boji pak nebezpečnou zbraní. V oblasti vzdělávání nebo vědy mohou mít chyby, které vznikají na základě opomenutí těchto aspektů, zásadní dopady zejména v oblasti sdílení informací či jejich efektivního vyhledávání. S tímto vědomím je třeba k problematice datové integrace a následné migrace informací přistupovat.

Použité prameny

1. ABERDEEN GROUP. 2007. *Customer Data Quality:Roadmap for Growth and Profitability* [online]. A White Paper of A Hartle-Hanks Company. June, 2007. [cit. 2012-01-22]. Dostupný z WWW: <http://research.ittoolbox.com/white-papers/pdfViewer.asp?r=http://hosteddocs.ittoolbox.com/Aberdeen_CDQ.PDF>.

2. ADELMAN, Sid; MOSS, Larisa; ABAI, Majid. 2005. Data Strategy. Addison-Wesley Professional IN, June 25, 2005. ISBN 978-0321240996.

3. AKENHURST, D. H.; KENT, S. 2002. A relational approach to defining transformations in a metamodel. In UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings [online]. Springer, J. M. Jezequel, H. Hussmann, and S. Cook, Eds., vol. 2460 of LNCS, 243-258. [cit. 2011-08-10]. Dostupný z WWW: <http://www.cs.kent.ac.uk/projects/kmf/Documents/uml02transf.pdf>.

4. BARTOŠ, Ivan; ŠMILAUER, Bohdan Ing. Získávání dat z informačních systémů (Z39.50): Definice aplikačních služeb a specifikace protokolu - volný výklad původní normy. ZIG - CR. 2002, Praha. (200 s.) (Information Retrieval (Z39.50): Application Service Definition and Protocol Specification) Dostupný z WWW < http://www.stk.cz/ZIG/Z39.50.zip>.

5. BARTOŠ, Ivan. Aplikace protokolu Z39.50 a perspektivy dalšího rozvoje. 2003, Praha. 150 s. (white paper).

6. BENYOVSZKY, Štěpán, Ing. 2003. eProvisioning: Synchronizace obsahu informací mezi nesourodými systémy. In ISSS2003 - Konference Internet ve státní správě a samosprávě. Hradec Králové, 23. 3. 2003. [online]. [cit. 2012-02-20]. Dostupný z WWW:

<http://www.isss.cz/archiv/2003/download/prezentace/BENYOVSZKY_clarionet.ppt>.

7. BERNSTEIN, Philip A.; HALEVY, Alon Y.; POTTINGER, Rachel, A. 2000. A vision of management of complex models [online]. SIGMOD Record 29(4):55-63 (2000). [cit. 2011-11-10]. Dostupný z FTP: <ftp://ftp.research.microsoft.com/pub/tr/tr-2000-53.pdf>.

8. BERNSTEIN, Philip, A. 2003. Applying Model Management to Classical Meta Data Problems. In 2003 CIDR Conference [online]. (Microsoft Research, One Microsoft Way) [cit. 2009-08-10]. Dostupný z WWW: <http://research.microsoft.com/~philbe/PBERNSTEINCIDR12ext.pdf>.

9. BÍLA, J.; TLAPÁK, M. 2004. Inženýrské ontologie pro reprezentaci funkcí v konceptuálním navrhování [online]. In Jemná mechanika a optika, No. 5, 2004, p. 134-137. ISSN: 0447-6441

10. BOHUSLAV, Jiří. 2006. Metody a procesy čištění dat. IT Systems [online]. Příloha Business Intelligence.7-8/2006. Str. 14. [cit. 2011-12-10]. Dostupný z WWW: <http://www.systemonline.cz/business-intelligence/metody-a-procesy-cisteni-dat.htm>. ISSN 1802-615X.

11. DAVENPORT, Thomas, H.; COHEN, Don; JACOBSON, Al. 2005. Competing on Analytics. [online]. Babson Executive Education - Working Knowledge Research Report. May, 2005. 4-5 s. [cit. 2012-02-10]. Dostupný z WWW: <http://www.babsonknowledge.org/analytics.pdf>

12. ECKERSON, Wayne. 2004. Data Profiling: A Tool Worth Buying (Really!). DM Review Magazine [online]. June, 2004 Issue [cit. 2011-08-03]. Dostupný z WWW: <http://www.dmreview.com/article_sub.cfm?articleId=1003990>.

13. GUARINO, Nicola. 1995. Formal Ontology, Conceptual Analysis and Knowledge Representation [online]. In International Journal of Human-Computer Studies - Special issue: the role of formal ontology in the information technology archive. Volume 43 Issue 5-6, Nov./Dec. 1995. Academic Press, Inc. Duluth, MN, USA. p.19 [cit. 2012-02-10]. Dostupný z WWW: <http://www.loa.istc.cnr.it/Papers/FormOntKR.pdf>.

14. GUARION, N.; GIARETTA, P. 1995. Ontologies and knowledge bases, towards a terminological clarification [online]. 1995. [cit. 2012-02-10]. Dostupný z WWW: <http//www.ladseb.pd.cnr.it/infor/Ontology/Papers/KBKS95.pdf >.

15. HAUSMANN, Hendrik, Jan; KENT, Stuart. 2003 Visualizing Model Mappings in UML In Proceedings of the 2003 ACM symposium on Software visualization 2003, San Diego, California., June 11 - 13, 2003. SESSION: All things UML [online]. Strana: 169-178. [cit. 2011-11-11]. Dostupný z WWW: < http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/Visualizing_Model_Mappings_in_UML.pdf >. ISBN:1-58113-642-0

16. HAY, Colin; ROSAMOND. Ben. In Journal of European Public Policy, Volume 9, Issue 2, 2002 [online]., 147-157. [cit. 2011-08-10]. Dostupný z WWW: <http://users.ox.ac.uk/~ssfc0041/globalisation.pdf>.

17. HORRIDGE, Matthew; RECTOR, Allan; STEVENS, Robert; WROE, Chris. 2004. A Practical Guide To Building OWL Ontologies Using The Proétgé-OWL Plugin and CO-ODE Tools [online]. Edition 1.0. The University Of Manchester. August 27, 2004. [cit. 2009-10-01]. Dostupný z WWW: <http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>.

18. ISO/IEC 11179 - 1:1999. Information technology — Specification and standardization of data elements — Part 1:Framework for the specification and

standardization of data elements [online]., [cit. 2011-10-22]. Dostupný z WWW: <http://metadata-standards.org/11179-1/ISO-IEC_11179-1_1999_IS_E.pdf>.

19. KLEMPA, Tomáš. 2006/2007. Opis jazyka OWL pre reprezentáciu ontológií [online]. Slovenská technická univerzita, Fakulta informatiky a informačných technológií, Ústav informatiky a softvérového inžinierstva. Prispevok k prednaske Znalostné systémy. [cit. 2012-02-02]. Dostupný z WWW: <http://www2.fiit.stuba.sk/~kapustik/ZS/Clanky0607/klempa/index.html>.

20. KYJONKA, Vladimír. 2006. Datová kvalita pod lupou. IT Systems [online]. Příloha Business Intelligence.7-8/2006. Str 16. [cit. 2012-02-11]. Dostupný z WWW: <http://www.systemonline.cz/business-intelligence/datova-kvalita-pod-lupou-1.htm>. ISSN 1802-615X.

21. LUJÁN-MORA, Sergio; VASSILIADIS, Panos; TRUJILLO, J. 2004. Data Mapping Diagrams for Data Warehouse Design with UML. In Congrès ER 2004: conceptual modeling (Shanghai, 8-12 November 2004) [online]. [cit. 2012-01-2]. Dostupný z WWW: <http://www.cs.uoi.gr/~pvassil/publications/2004_ER/ER_2004.pdf>

22. MCGUINESS, Doborah; VAN HARMELEN, Frank. 2004. OWL Web Ontology Language Overview. W3C 2004. Dostupný z WWW: <http://www.w3.org/TR/owl-features/>

23. MOHANEC, Martin. 2004. Několik poznámek k porozumění objektového paradigmatu. In Objekty 2004. Ostrava, VSB-TUO, 2004, s. 189-197.

24. MOHANEC, Martin. 2006. KONCEPTUÁLNÍ MODELOVÁNÍ, FORMÁLNÍ ZÁKLADY A ONTOLOGIE [online]. České vysoké učení technické – FEL. Česká republika. 2006. [cit. 2012-01-20]. Dostupný z WWW: <http://formular-ekf.vsb.cz/formulare/F01/tsw/getfile.php?prispevekid=873>.

25. MOHANEC, Martin. 2005. Ontologie a konceptuální modelování (stručný úvod) [online]. České vysoké učení technické – FEL. Česká republika. [cit. 2012-01-20] Prezentace v MS PowerPoint. Dostupný z WWW: < www.gisaci.upol.cz/file/637/ontologie-4-omo.html>.

26. OLSON, Jack. 2002. Data Profiling: The Data Quality Assurance Analyst's Best Tool. DM Direct Newsletter [online]. December 13, 2002 Issue [cit. 2011-08-09]. Dostupný z WWW: < http://www.information-management.com/infodirect/20021213/6156-1.html>.

27. OMG. 2010. Object Constraint Language, Version 2.2.[online].February 2010. [cit. 2011-08-10]. Dostupný z WWW: <http://www.omg.org/spec/OCL/2.2/.>.

28. PAPÍK, Richard. 2001. Competitive Intelligence, informační služby, Internet a informační profese. Ikaros [online]. 2005. Roč. 5, č. 4 [cit. 2011-07-18]. Dostupný z WWW: <http://www.ikaros.cz/node/739>. URN-NBN:cz-ik739. ISSN 1212-5075.

29. RAHM, Erhard; BERNSTEIN, Philip, A. 2001. On Matching Schemas Automatically [online]. Microsoft Research Technical Report MSR-TR-2001-17. February, 2001. [cit. 2011-08-10]. Dostupný z FTP: <ftp://ftp.research.microsoft.com/pub/tr/tr-2001-17.pdf>.

30. Regular Expression Library[online]. 2008. [cit. 2011-02-28]. Dostupný z WWW: <http://regexlib.com/>.

31. RFC 2822. Internet Message Format [online]. 2001 Resnick, P. April 2001 [cit. 2011-02-28]. 51 s. Dostupný z FTP: <ftp://ftp.rfc-editor.org/in-notes/rfc2822.txt>.

32. RICHTA, Karel. 2010. Jazyk OCL a modelem řízený vývoj. In: Moderní database 2010. Nesuchyně, Komix. Praha 2010 [cit. 2011-02-28]. Dostupný z WWW: <https://www.ksi.mff.cuni.cz/~richta/publications/Richta-MD-2010.pdf>.

33. RICHTA, Karel. 2010 (2). Rekonstrukce OCL z SQL. [online]. In DATAKON 2010. Ostrava: Ostravská univerzita, 2010, s. 1-10. [cit. 2011-02-28]. Dostupný z WWW: <https://www.ksi.mff.cuni.cz/~richta/publications/Datakon-2010-Richta.pdf>. ISBN 978-80-7368-424-2.

34. RUSSOM, Philip. 2007. Unifying the Practices of Data Profiling, Integration, and Quality (dPIQ) [online]. TDWI Monograph Series. October, 2007. [cit. 2011-12-10]. Dostupný z WWW: <http://download.101com.com/pub/tdwi/Files/TDWI_Monograph_DataFlux_Oct2007.pdf>.

35. SATRAPA, Pavel. 2000. Seriál Regulární výrazy. Root.cz [online]. 2000. [cit. 2012-02-25]. Dostupný z WWW: <http://www.root.cz/serialy/regularni-vyrazy/>. ISSN 1212-8309.

36. SINGH, Ranjit; SINGH, Kawaljeet, Dr. et al. 2010. A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing. In IJCSI International Journal of Computer Science Issue, Vol. 7, Issue 3, No. 2, May 2010 [cit. 2012-02-25]. Dostupný z WWW: <http://www.ijcsi.org/papers/7-3-2-41-50.pdf>.ISSN (Online): 1694-0784.

37. SKLENÁK, V. 2001 Data, informace, znalosti a Internet. Praha. C.H.Beck, 2001. s. 3-4. ISBN 80-7179-409-0.

38. STOLOVITSKY, Neil. 2010. Managing the Project Document: The importace of an effective dokument management stratem for project success [online]. Project

Smart 2000-2010. [cit. 2012-02-15] Dostupný z WWW: <http://www.projectsmart.co.uk/pdf/managing-the-project-document.pdf>.

39. SVÁTEK, Vojtěch; LABSKÝ, Martin. 2003. Objektové modely a ontologie - podobnosti a rozdíly [online]. Katedra informačního a znalostního inženýrství, Vysoká škola ekonomická v Praze, nám. W. Churchilla 4, 130 67, Praha 3. [cit. 2012-02-15]. Dostupný z WWW: <http://nb.vse.cz/~svatek/obj03fi.pdf>.

40. SVÁTEK, Vojtěch. 2002. „Ontologie a WWW“ in DATAKON 2002, Brno, 19. – 22. 10. 2002, p. 1–35, ISBN 80-210-2958-7.

41. ŠMAJS, Josef, Doc. PhDr. CSc.; KROB, Josef, PhDr., CSc. 1994. Úvod do ontologie [online]. Masarykova univerzita, Brno, 1991, 1994. Kapitola Co je ontologie? [cit. 2012-02-15]. Dostupný z WWW: <http://www.phil.muni.cz/fil/eo/skripta/index.html>. ISBN 80-210-0879-2.