Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Diplomant	: Martin Šlouf
Vedoucí diplomové práce	: doc. RNDr. Pavel Drbal, CSc.
Recenzent	: Ing. Martin Labský

TÉMA DIPLOMOVÉ PRÁCE

Analýza systému TUTOS

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu,

V Praze dne 13.12. 2004 .....

podpis

Abstrakt

Tento dokument předkládá analýzu informačního webového systému TUTOS. TUTOS je projekt SourceForge.net šířený pod licencí GPL, který lze využít pro řízení projektů a související činnosti. Cílem je pomocí metodiky OMT popsat návrh systému, vymezit chyby v návrhu a implementaci a doporučit konkrétní řešení vedoucí k nápravě. První kapitoly práce jsou věnovány především popisu systému, následující části diskutují doporučené úpravy. Při popisu systému se vychází z jeho jednotlivých vrstev (prezentační, aplikační a databázové). V databázové a aplikační vrstvě se vyskytuje několik málo nedostatků, jež mohou být relativně snadno odstraněny. Největší problémy jsou s prezentační vrstvou, která již nevyhovuje současným potřebám systému — nedostatečným způsobem podporuje alternativní výstupní formáty, zejména XML. Podstatná část práce se věnuje návrhu nové prezentační vrstvy, jež by měla nabídnout lepší řešení při respektování současných principů tvorby stránek.

Klíčová slova: TUTOS, OMT, objektová analýza, objektový design, open source

Abstract

This thesis presents analysis of TUTOS, the web information system. TUTOS is SourceForge.net project licensed under GPL. It can be used as a project management tool, while providing also other functionality. The main objective is to provide description of the system design using OMT method as well as present detailed view on problematic parts of the system and suggest an appropriate solution. First part of the thesis describes the system. Second part suggests and discusses the solutions. Description of the system is based on system layers — presentation, application (business) and database layer. Database and application layers have very few failures and all of them could be safely fixed. The most complicated solution is suggested for presentation layer, which lacks support for alternative outputs formats, such as XML. Significant part of the thesis is dedicated to the new presentation layer design.

Keywords: TUTOS, OMT, object analysis, object design, open source

Předmluva

Tato práce popisuje informační systém TUTOS [2] z pohledu vývojáře. Od tohoto dokumentu může očekávat užitek zejména komunita vývojářů systému TUTOS, mezi které patří i autor této práce.

TUTOS je systém s volně dostupným zdrojovým kódem (open source) šířený pod licencí GPL [1] (free software). Jedná se o program pro řízení projektů a podobné činnosti. Nabízí příjemné prostředí, dobrou funkcionalitu, nezávislost na platformě, lokalizaci a stabilitu. O použitelnosti (a užitečnosti) svědčí aktivita ve fórech, elektronických konferencích a na webových stránkách ([3], [4]). Systém je využíván i v českém prostředí.

Snahou a motivací autora je, mimo jiné, nahradit jistý nedostatek řady volně šiřitelných programů — totiž nedostatek kvalitní dokumentace.

Poděkování

Na tomto místě bych rád poděkoval svému školiteli, panu doc. RNDr. Pavlu Drbalovi, CSc., za jeho toleranci a přizpůsobivost, se kterou vedl moji diplomovou práci, a svým blízkým, kteří projevili značné pochopení po dobu, kdy tento dokument vznikal.

Samozřejmostí je též vyjádření obdivu a podpory komunitě lidí propagujících volně šiřitelný software, bez nichž by tato práce nikdy nevznikla.

Typografické konvence

Z důvodu snazší čitelnosti jsou na některých místech cizí termíny skloňovány, místo aby bylo použito zdlouhavého opisného tvaru.

Označení prvků a popisky v modelech jsou anglicky — tak mohou modely posloužit většímu okruhu čtenářů.

Jména tříd jsou převzaty ze systému TUTOS.

`Strojem jsou psány URL odkazy, názvy programů a kód.`

Skloněné písmo je použito pro termíny z rejstříku.

Tento dokument byl vytvořen v systému LaTeX 2$_\varepsilon$ za použití textového editoru GNU Emacs, modely byly nakresleny v programu Dia. Je volně dostupný na adrese `http://www.justlogin.cz/martin/projekty/diplomka.tgz`.

Obsah

Předmluva **3**

Poděkování **3**

Typografické a stylistické konvence **3**

Obsah **4**

Seznam obrázků **6**

1 Úvod **7**
1.1 Krátké představení systému TUTOS 7
1.2 Cíl a smysl práce 7
1.2.1 Historie projektu 8
1.2.2 Současný stav 8
1.2.3 Přínosy práce 10
1.2.4 Způsob dosažení cíle 10
1.3 Struktura dokumentu 12

2 Základní popis systému **13**
2.1 Vysvětlení pojmů 13
2.2 Vrstvy systému a vazby mezi nimi 14
2.2.1 Prezentační vrstva 14
2.2.2 Aplikační vrstva 16
2.2.3 Databázová vrstva 17
2.3 Shrnutí 17

3 Stávající struktura systému **18**
3.1 Prezentační vrstva 18
3.1.1 XML 20
3.2 Aplikační vrstva 21
3.2.1 Dědičnost 21
3.2.2 Asociace 24
3.3 Databázová vrstva 27
3.4 Shrnutí 30

4 Úpravy databázové vrstvy **31**
4.1 Nová databázová knihovna 31

4.2 Speciální požadavky 32
4.3 Dopady na systém 34
4.4 Shrnutí 34

5 Úpravy aplikační vrstvy **35**
5.1 Třída `Tutos_Base` 35
5.2 Třída `Tutos_Module` 36
5.2.1 Tvorba vlastního modulu 36
5.2.2 Odstranění dědičnosti 39
5.3 Další diskuze 39
5.3.1 Zpracování vstupu uživatele 39
5.3.2 Rozdělení třídy `Tutos_Base` 41
5.4 Dopady na systém 41
5.5 Shrnutí 43

6 Úpravy prezentační vrstvy **44**
6.1 Problémy prezentační vrstvy 44
6.2 Požadavky na prezentační vrstvu 45
6.3 Analýza nové prezentační vrstvy 48
6.3.1 Výchozí situace 48
6.3.2 Základní koncept 50
6.4 Podrobný model 52
6.4.1 Celkový přehled 54
6.4.2 Export/Import 55
6.4.3 Zamýšlené použití 56
6.5 Dopady na systém 57
6.6 Shrnutí 59

7 Závěr **60**
7.1 Problémové části a doporučení 60
7.1.1 Shrnutí postupu analýzy 60
7.1.2 Dosažené výsledky 60
7.2 Dokumentace 61
7.3 Diskuse nad přínosy práce 62
7.4 Další postup 63
7.4.1 `php.MVC` 63
7.4.2 Shrnutí 64

Reference **65**

Rejstřík **67**

Seznam obrázků

1 Pojmový model 13
2 Vrstvy systému TUTOS a jejich vzájemné vazby 15
3 Prezentační vrstva 18
4 Prezentační vrstva — inspirace Javou 19
5 Dědičnost 22
6 Asociace 24
7 Asociace — `Projects` ↔ `Contacts` 25
8 Databázová vrstva 28
9 Třída `Tutos_Base` 35
10 Třída `Tutos_Module` 37
11 Prezentační vrstva — model jednání 45
12 Prezentační vrstva — zjednodušený model 51
13 Prezentační vrstva — objektový model 53

1 Úvod

V úvodu naleznete základní přehled systému, stanovení konkrétního cíle a způsoby jeho dosažení. Je představena struktura práce.

Jak již bylo řečeno, práce se zabývá systémem TUTOS, jeho návrhem a implementací.

1.1 Krátké představení systému TUTOS

TUTOS, [2] a [3], je informační systém vyvíjený a šířený Gero Kohnertem a několika dalšími vývojáři pod licencí GPL, [1]. *TUTOS*

Jedná se o silný nástroj, vhodný zejména pro menší až střední firmy v oblasti služeb, který vedle jiného poskytuje:

1. *Řízení projektů* s možností utvářet různé vztahy mezi projekty (hierarchie a časové návaznosti).
2. Kompletní *správu kontaktů* a *schůzek*.
3. *Sledování chyb* u implementovaných projektů u zákazníka, včetně ohlášení chyby zákazníkem a sledování jejího stavu.
4. Úplnou *správu dokumentů* s verzováním a uzamykáním.
5. Nabízí *evidenci odpracovaných hodin*.
6. Díky vhodnému návrhu databázových struktur umožňuje navzájem velmi dobře provázat všechny *objekty* systému — nejste omezeni téměř ničím.

Můžete například navázat dokument na projekt, schůzku ke kontaktu apod.

TUTOS je implementován jako klient/server aplikace, která jako klienta používá internetový prohlížeč a ke komunikaci se serverem používá protokol HTTP. Na straně serveru je využit buďto (1) webový server s podporou PHP a SQL databáze nebo (2) webový kontejner (Java) a SQL databáze.

1.2 Cíl a smysl práce

Hlavní přínosy této práce by měly být:

- Identifikace problémových částí TUTOSu, resp. jeho návrhu.
- Konkrétní doporučení, jak danou část zlepšit.
- Dokumentace, na které lze dále stavět.

S těmito body je ztotožněn cíl práce a její smysl. Než bude cíl podrobněji rozebrán, je vhodné říci pár slov o historii projektu TUTOS.

1.2.1 Historie projektu

TUTOS byl na SourceForge.net zaregistrován v červenci 2000 jako groupwarová aplikace pro správu týmů a projektů, napsaná v PHP. V červnu 2003 byl registrován *Java TUTOS* — snaha re–implementovat TUTOS do Javy. Zatímco původní PHP verze se dočkala řady aktualizací, implementace v Javě ustrnula.

Java TUTOS

Aktuální stabilní verze TUTOSu je 1.2, jež byla uvolněna 18. října 2004. Aktuální vývojová verze je 1.3, která bude uvolněna někdy během příštího roku. Vedle toho existuje ještě verze 2.0, kde se testují některé zásadní nové přístupy.

V roce 2000 bylo aktuální PHP verze 3 a verze 4 byla žhavá novinka. Obě verze nabízejí základní podporu objektového programování. V současné době PHP verze 5 je zdrojový kód aplikace připraven k přechodu na tuto verzi, ale rozhodně nevyužívá všech jejích možností.

Již v prvních fázích vývoje systému byl dobře rozmyšlen systém rozšíření (plug-in) skrze moduly, nezávislost na databázi a časem byl dobře propracován systém oprávnění.

Bohužel si TUTOS vedle těchto vhodně navržených částí nese i známky toho, že řada vlastností byla implementována bez vážnějšího rozmyšlení — prostě takový způsob „*chci tuto vlastnost a chci ji hned*“ — způsob, který je pro některé open source projekty typický (blíže viz [7]).

1.2.2 Současný stav

Současný stav (konec roku 2004) je, bohužel, úzce spjat s implementačním prostředím. To, že je systém primárně implementován v PHP je jeho požehnáním a prokletím zároveň.

Použití komponentních knihoven Na mnoha místech TUTOSu je použito komponentních knihoven. Začlenění některých z nich do kódu aplikace by se však mělo věnovat více pozornosti.

Při použití komponentní knihovny je nutné zamyslet se nad:

1. *Použít tuto knihovnu*, čímž sice bude ušetřen čas implementací (a analýzou), ale může dojít ke ztrátě rychlosti běhu aplikace a musí být dobře rozmyšleno její začlenění do aplikace.

2. *Použít vlastní kód*, čímž bude vytvořen kód přesně na míru, ale neušetří se čas analýzou a implementací.

Obecně se lze konstatovat, že téměř ve všech případech se vyplatí použít komponentní knihovnu, což potvrzuje například i [9][1] a [6].

[1] Cituji: (str. 19) *„Pro běžnou praxi zpravidla bývá mnohem významnější, zda program*

Použití PHP PHP je mladý programovací jazyk, který se lze velmi rychle naučit, snadno v něm lze udělat relativně složitější úlohy díky množství *knihoven* a právě díky tomu se stal velmi populární pro dynamické webové aplikace.

Řada knihoven PHP je navržena po vzoru Javy a poskytují řadu užitečných komponent.

Jeho podpora objektového programování však není ideální a v každé verzi se navíc tato mění[2].

Na rozdíl například od Javy záleží na programátorovi, zda a jak bude dodržovat zásady *správného stylu*[3].

Snadno dostupné komponentní knihovny ve spojení se snadno použitelným PHP znamenají rychlý vývoj, ale pokud je prováděn bez rozmyslu a návrhář/programátor nemá sám dost disciplíny, snadno se sklouzává ke špatnému návrhu/implementaci.

Rozsáhlost systému Pokud aplikace doroste určitých rozměrů, prokážou se výhody aplikačního serveru, který může poskytovat řadu vhodných služeb, které mohou být ihned využity (adresářová služba, repository pro globální objekty a zdroje aj.). Avšak PHP kód TUTOSu není pro žádný takový server uzpůsoben[4].

Zároveň se v systému takového rozsahu (a bez dokumentace) může provést určité provizorní řešení („jen protentokrát, pak se to upraví“), které se však postupem doby stane nedílnou součástí kódu. Postupně se tak mohou vytratit dobré zásady původního návrhu, pokud nejsou vynuceny jazykem či nějakou autoritou.

Kritická hranice Systém dosáhl podle mnohých vývojářů určité kritické hranice, kdy je vhodné, zamyslet se nad dalším vývojem. Objevila se řada nových vlastností (podpora XML, zlepšení perzistence, řada nových modulů) a ukazuje se, že by bylo záhodno, aby v určitých oblastech byl systém koncepčněji navržen.

Panuje všeobecný konsensus mezi vývojáři, že nyní je ten správný okamžik probrat a zpracovat všechny možné návrhy změn, aby verze 2.0 byla po stránce designu pokud možno bezchybná.

Shrnutí Samozřejmě je mylné se domnívat na základě těchto faktů, že TUTOS je „slepenina“ neumělého kódu s řadou špatně integrovaných knihoven — přesvědčit se o tom koneckonců může kdokoli díky otevřenému zdrojovému kódu, jež je dostupný na [3] — spíše lze říci:

vyvíjíte dva dny nebo týden, než zda jeho běh trvá 5 nebo 15 sekund.“

[2] Ačkoli je dodržována zpětná kompatibilita se staršími verzemi mnohem lépe než například u Visual Basicu.

[3] (1) přidělení zdrojů, (2) použití a ošetření chyb (3) uvolnění zdrojů.

[4] V současné době se objevují první implementace PHP aplikačních serverů, např. [10] a [11].

TUTOS je v hlavních aspektech dobře navržen, používá řadu vhodných komponentních knihoven a díky PHP je snadno rozšiřitelný, přístupný pro každého (vývojáře) a o jeho užitečnosti svědčí i jeho rozšíření mezi uživateli. Avšak na několika místech se projevila jistá živelnost[5] *a nevhodná implementace, ačkoli návrh a zamýšlené použití jsou v pořádku.*[6]

1.2.3 Přínosy práce

Výše definovaný cíl a smysl práce zahrnuje tyto body:

- Identifikace problémových částí TUTOSu, resp. jeho návrhu.
- Konkrétní doporučení, jak danou část zlepšit.
- Dokumentace, na které lze dále stavět.

Po přečtení předchozích odstavců je zřejmé, že není bezúčelný.

Přínosy práce jsou evidentní — pokud se podaří zde navrhované změny prodiskutovat a prosadit v komunitě vývojářů (byť modifikované), povede to ke zkvalitnění návrhu systému, jeho větší modularitě a zároveň k vypracování kvalitní dokumentace, což může mít ten příjemný vedlejší efekt, že se do vývoje ochotně zapojí další lidé.

1.2.4 Způsob dosažení cíle

Jedná se o velmi prakticky zaměřenou práci (i proto bylo toto téma zvoleno), jež přináší analýzu (chápanou nyní v širším slova smyslu jako *popis a diskusi*) k již existujícímu systému. Této skutečnosti a osobním preferencím autora je uzpůsoben výběr metodiky, pomocí které bude systém představen — jedná se o *OMT*, viz [5].

OMT — Objektově orientované metodiky a technologie

S metodikou OMT má autor největší zkušenosti a je třeba na ní ocenit její praktický přístup. Sama metodika neklade žádný konkrétní striktní předpis, ale nabízí sadu několika nástrojů a do jisté míry ponechává na samotných návrhářích, které nástroje zvolí — ti by samozřejmě měli svoji volbu odůvodnit.

Jak je naznačeno na straně 11, systém je vcelku rozsáhlý a podat jeho vyčerpávající popis by si vyžádalo mnoho prostoru. Není cílem práce do všech podrobností popsat celý systém — spíše by tento dokument měl poskytnout ucelený náhled bez toho, aniž by se utápěl v nepodstatných detailech.

[5] Typickým představitelem této kategorie je snaha, aby jistá služba či vlastnost byla rychle dostupná.

[6] PHP umožňuje porušit řadu *správných zásad*, protože si je neumí vynutit na úrovni jazyka.

Systém je představen v mnoha modelech s různou úrovní podrobnosti a abstrakce. Některé modely cíleně zanedbávají některé prvky systému tak, aby do popředí vystoupily jiné a čtenář nebyl zavalen množstvím nepodstatných detailů. Cílem popisu je postup od nepodrobného a obecného k podrobnému a konkrétnímu. Snahou autora je vhodným způsobem abstrahovat od detailů, jež nejsou v daný okamžik podstatné (proto postupné zpodrobňování modelů).

Některé prvky systému budou rozebírány několikrát na různých úrovních podrobnosti, některé prvky budou pouze zmíněny a dále jim nebude věnována pozornost. Upřednostnění některých částí před jinými se bude dít na základě (jistě nijak překvapivě):

1. *Významu vzhledem k mému řešení* — čím podstatnější, tím větší prostor (význam by měl vyplynout z předkládaných faktů).
2. *Složitosti* — čím složitější, tím větší prostor.

Základní charakteristiky systému V pojmech OMT lze systém popsat jako *transakční manažer* (databázová aplikace) s *interakčním rozhraním* (formuláře) s občasnými prvky *dávkového zpracování* (importy a exporty dat).

Jedná se o *otevřenou architekturu*.

Paralelnost je řešena na úrovni databáze pomocí transakcí databázového stroje.

Aktuální stabilní PHP verze systému má přes 30 tříd aplikační logiky, více jak 40 databázových tabulek a přes úctyhodných 160 000 SLOC[7]. Nyní by se mohlo zdát, že (hrubým odhadem) pouhých 10 tabulek připadá na vazby a vztahy `m:n`, avšak jak bude patrno dále (3.2.2, 3.3), vůbec tomu tak není.[8]

Důležité modely Protože jde o *transakční manažer*, vyplývá z [5], že hlavním modelem je objektový model mapovaný přímo na databázi. Dynamické modely nejsou důležité — operace jsou známy předem[9].

transakční manažer — systém zpřístupňující informace pro více uživatelů

Hlavními modely tedy budou objektové modely. U vybraných problémových oblastí bude postupováno podle charakteru problému a výběr modelů tomu bude uzpůsoben.

Omezení Existuje několik omezujících předpokladů, jež je třeba respektovat:

[7] SLOC = Source Line Of Code = Řádek zdrojového kódu

[8] Budiž toto příklad, jak tyto jistě užitečné charakteristiky mohou mýlit.

[9] Jedná se především o různé aktualizace a dotazování.

1. *Existující systém.* Je třeba respektovat množství hotového kódu, kde vůle dělat změny není veliká — vždy je třeba pečlivě zvážit, kolik práce se spotřebuje na implementaci doporučujících návrhů.

2. *Implementace v PHP a Javě.* Ačkoli je hlavní větví TUTOSu jeho PHP implementace, je vhodné změny navrhovat i s ohledem na Javu. Myslím, že toho může být dosaženo bez většího úsilí — čeho lze dosáhnout v PHP, stejně tak jde udělat i v Javě (a spíše snáze v Javě než v PHP).

1.3 Struktura dokumentu

Dokument lze rozdělit na dvě samostatnější části (vedle úvodu a závěru):

1. *Popis systému.*

   Kapitola 2 představuje čtenáři základní funkčnost a části systému.

   Kapitola 3 si podrobněji všímá částí vymezených v předchozím textu a blíže popisuje problémy, které budou předmětem řešení.

2. *Doporučené úpravy.*

   Objeveným nedostatkům jsou postupně věnovány kapitoly 4, 5 a 6. Dále zpodrobňují pohled na systém a do větší hloubky analyzují vybrané oblasti. Snaží se konkretizovat postupy vedoucí k odstranění nalezených problémů.

Všechny kapitoly, zejména pak ty, jež popisují úpravy systému, lze číst samostatně. Každá kapitola je uvedena krátkým výčtem toho, co od ní čtenář může očekávat a je zakončena stručným shrnutím. U kapitol popisujících úpravy systému předchází shrnutí diskuze dopadů na systém.

2 Základní popis systému

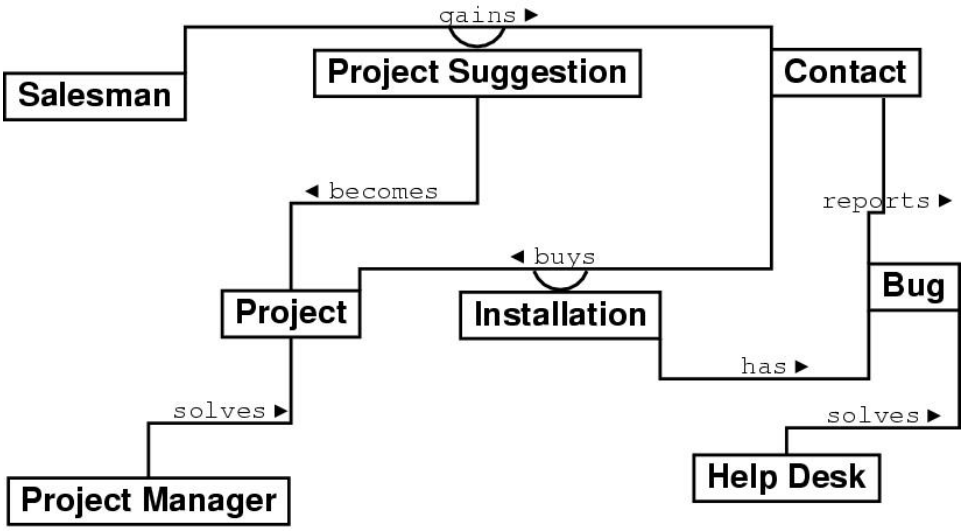
Tato kapitola popisuje základní pojmy systému (2.1) a podává úvodní přehled návrhu systému — přináší přehled jeho vrstev (vertikální členění, 2.2), jež je dále rozebíráno v navazujících kapitolách.

Pro připomenutí základní funkcionality systému viz oddíl 1.1, prvotní technické vymezení systému viz oddíl 1.2.4

2.1 Vysvětlení pojmů

Systém TUTOS je primárně určen pro řízení projektů. Na obrázku 1 je zobrazen *pojmový model*, ze kterého jsou nejlépe patrné hlavní pojmy a vazby mezi nimi.

pojmový model



Obrázek 1: Pojmový model

Základním pojmem TUTOSu je pochopitelně *projekt* (*Project*). Tento pojem je nutné chápat velmi obecně — projektem může být v prvním přiblížení cokoli, o co má potenciální zákazník (*Contact*) zájem[10].

projekt

Model má několik samostatnějších částí:

1. Obchodník (*Salesman*) získává potenciální zákazníky (*Contact*) a, pokud jde o vážného zájemce, vzniká projekt.

2. Projektový manažer řeší zadaný projekt a ten je v konečném důsledku koupen zákazníkem. Zákazník však nekupuje projekt, ale jeho jeden

[10] (1) informační systém, (2) průzkum trhu, (3) ...

exemplář (*Installation*). Vztah mezi projektem a exemplářem je analogický vztahu *titul* → *konkrétní kniha*.

3. Kontakt se stává zákazníkem v okamžiku, kdy vlastní jedinečný exemplář projektu, neboli tzv. instalaci. V tomto exempláři se samozřejmě mohou vyskytovat různé chyby (*Bugs*) a proto má klient možnost zpětné vazby.

Pokud se nad tímto modelem zamyslíme, je patrné, že se ve skutečnosti jedná o obecný model *konzultantské firmy*[11]! Toto zjištění je velmi uspokojující, protože je z něho minimálně patrné, že pro nasazení v konzultantské společnosti je TUTOS vhodným kandidátem, což odpovídá jeho zamýšlenému použití (viz 1.1 a [12])[12].

TUTOS samozřejmě podporuje řadu dalších agend (dokumenty, schůzky, odpracované hodiny, ...) a různé vazby mezi nimi, ale všechny tyto doplňky jsou v rámci systému chápány jako rozšíření funkcionality v oblasti řízení projektů, ačkoli mohou být využity i samostatně.

2.2 Vrstvy systému a vazby mezi nimi

vrstvy systému

Nyní je vhodný okamžik podívat se blíže na jednotlivé *vrstvy systému*. Přehled přináší obrázek 2. Záměrně došlo k opomenutí řady tříd (zejména z oblasti aplikační vrstvy), avšak díky tomu mohou vyniknout vzájemné důležité vazby.

2.2.1 Prezentační vrstva

prezentační vrstva

Prezentační vrstva je na první pohled prostá. Existují dvě hlavní základové abstraktní třídy (podle konvence UML znázorněné *kurzívou*): (1) Třída vzhled (`Theme`) a (2) třída rozvržení (`Layout`).

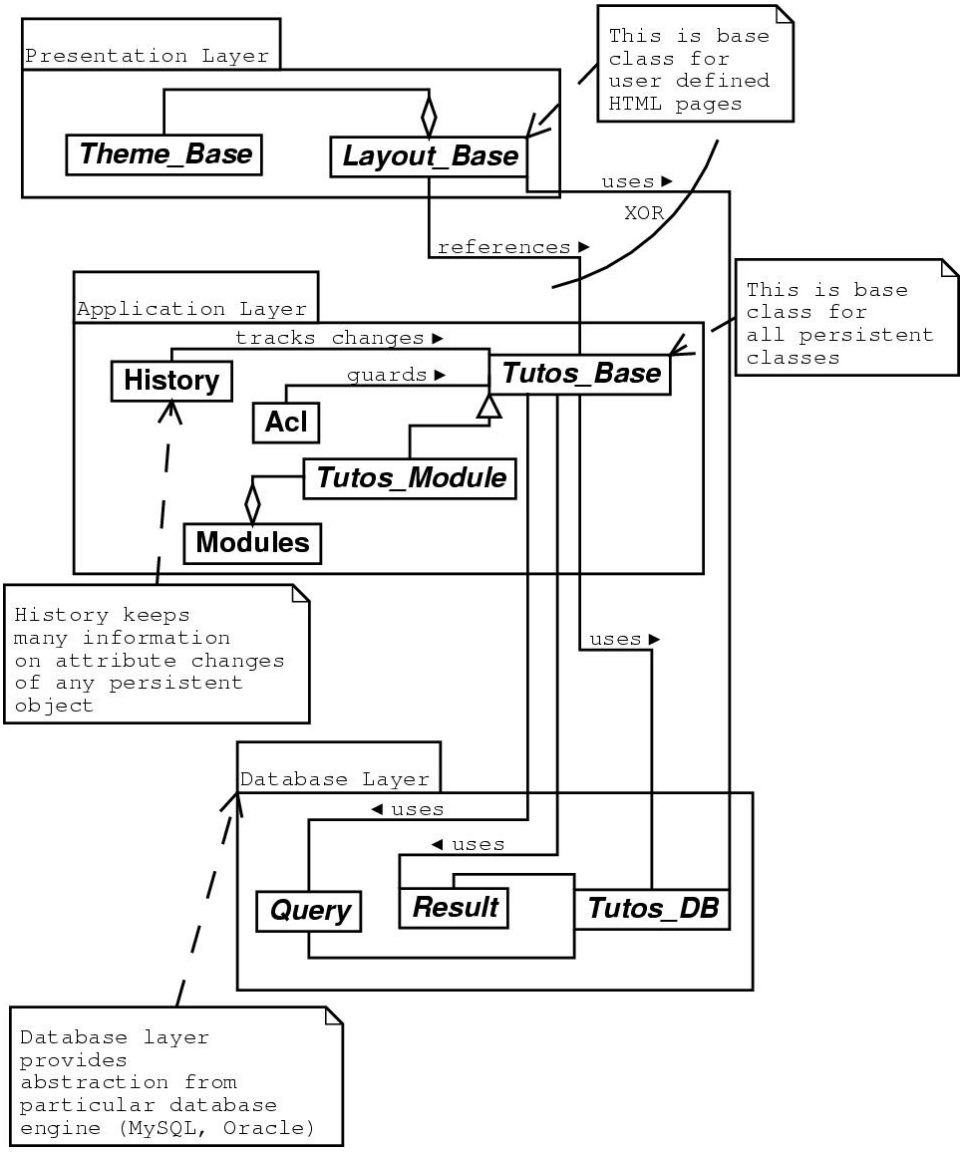
Třída vzhledu `Theme` příliš zajímavá není. Jejím hlavním úkolem je definovat grafické prvky (obrázky) a styly písma, což v dnešní době CSS stylů většinou znamená pouze postarat se o připojení vhodného souboru stylů, dále má na starosti zavedení javascriptu a několik málo dalších drobností.

Třída rozvržení `Layout` je mnohem důležitější. Definuje základní rozvržení na stránce — řídí umístění nabídky, pracovní plochy a realizuje její konkrétní obsah. Pro jednu (běžnou) třídu aplikační logiky se většinou definuje několik `Layout` podtříd, které zajišťují:

1. Prosté zobrazení jednoho konkrétního objektu aplikační logiky.

[11] Což je podtrženo i hlavními rolemi zaměstnanců — (1) obchodník, (2) projektový manažer a (3) help–desk.

[12] Potvrzují to i instalace v České republice, viz http://www.finapp.cz



Obrázek 2: Vrstvy systému TUTOS a jejich vzájemné vazby

2. Editaci atributů jednoho konkrétního objektu. Většinou lze beze změn použít i pro vložení nového objektu.
3. Zobrazení seznamu objektů daného typu.
4. Vyhledávání objektů daného typu.

Existují i výjimky, kdy třída typu `Layout` není svázána s žádným objektem aplikační logiky a přistupuje přímo k databázi, jak je ostatně patrno z modelu[13].

2.2.2 Aplikační vrstva

aplikační vrstva

Aplikace typu *transakční manažer* bývají často realizovány tak, že třídy *aplikační vrstvy* jsou odvozeny od společné základové třídy, jejímž úkolem je zajistit perzistenci objektům (viz [5] a [13]).

třída `Tutos_Base`

TUTOS v tomto není výjimkou. Služby perzistence zajišťuje právě základová *třída* `Tutos_Base`. Jako jediná spolupracuje s třídami databázové vrstvy a poskytuje virtuální metody, které řeší aktualizace a dotazování do databáze, jež jsou vždy v příslušné aplikační třídě přetíženy a doplněny.

Každý perzistentní objekt má v rámci systému unikátní identifikační číslo. To umožňuje složité provázání objektů, i s menším počtem vazebních tabulek (viz strana 11 a oddíl 3.2.2).

Velmi zajímá je třída `History`. Její instance sledují u perzistentních objektů změny v datových atributech objektu po celou dobu jeho existence, včetně informací o tom, kdo, kdy a samozřejmě jak objekt změnil.

ACL = Access Control List

Objekty třídy `Acl` ke každému objektu drží systém oprávnění, neboli *ACL*, který jasně definuje kdo má k objektu přístup a jaké operace nad ním může vykonávat. Existují 4 druhy oprávnění:

1. Zobrazit — právo prohlížet si objekt.
2. Použít — právo použít objekt — právo operovat nad vazbami objektu.
3. Upravit — právo editovat datové atributy objektu.
4. Odstranit — právo odstranit objekt z databáze.

třída `Tutos_Module`

Třída `Tutos_Module` je základní třídou pro tvorbu nových rozšíření TUTOSu — každé rozšíření by ji mělo zdědit a poskytnout tak jádru TUTOSu potřebné informace. To, proč dědí od třídy `Tutos_Base` zatím nechme stranou (podrobněji viz 3.2.1).

[13] Příkladem mohou být například různé administrativní stránky, reporty aj.

2.2.3 Databázová vrstva

databázová vrstva

Hlavním úkolem *databázové vrstvy* je dobře odstínit zbytek aplikace od konkrétní použité databáze. Podporovány jsou tyto databáze: (1) MySQL, (2) PostgreSQL, (3) Interbase, (4) Oracle a (5) MSSQL, ve svých stabilních verzích.

Rozumnou snahou vývojářů TUTOSu je používat určitou podmnožinu ANSI SQL92, takže by nemělo být složité přenést TUTOS i na jiné databáze. Nejdiskutovanější může být z tohoto hlediska podpora MySQL, která dlouhou dobu přes své nesporné kvality v jiných oblastech zaostávala a zaostává v podpoře užitečných funkcí (transakce, datové pohledy a generátory), ale protože jde o velmi rozšířenou a populární databázi, byla podporována od samého začátku a některé nedostatky (generátory) byly rozumným způsobem nahrazeny v samotném kódu aplikace.

Tři uvedené třídy (stále obrázek 2) jsou běžnými obdobami modernějších databázových rozhraní, jak je známo například z JDBC (viz [13]).

třída `Tutos_DB`

Třída `Tutos_DB` představuje spojení na databázi a přes své metody zpřístupňuje třídy s obecným rozhraním podle konkrétní databáze `Query` (určené ke konstrukci dotazů[14]) a `Result` (určené ke zpracování výsledků). Řízení transakcí je možné přímo přes metody třídy `Tutos_DB`.

Pokud bych měl připodobnit tyto třídy například ke třídám a rozhraním balíku `java.sql.*`, viz například [13], vypadalo by to přibližně takto:

```
Tutos_DB : Driver, Connection, DatabaseMetaData, Statement
Query    : Statement, PreparedStatement
Result   : ResultSet, ResultSetMetaData
```

Je vidět, že tyto 3 třídy přebírají řadu odpovědností a někde se i překrývají, blíže k tomuto problému viz dále.

2.3 Shrnutí

Lze konstatovat, že na této úrovni abstrakce systém splňuje všechny obecné požadavky (až na nevysvětlenou dědičnost `Tutos_Base` → `Tutos_Module`).

Vztahy mezi základními pojmy stejně jako pojmy samotné jsou v pořádku. Vazby mezi jednotlivými vrstvami (prezentační, aplikační a databázovou) jsou také v pořádku.

Jak bude vidět dále, při dalším zpodrobnění vystoupí již do popředí některé nedostatky, kterým bude věnován zbytek této práce.

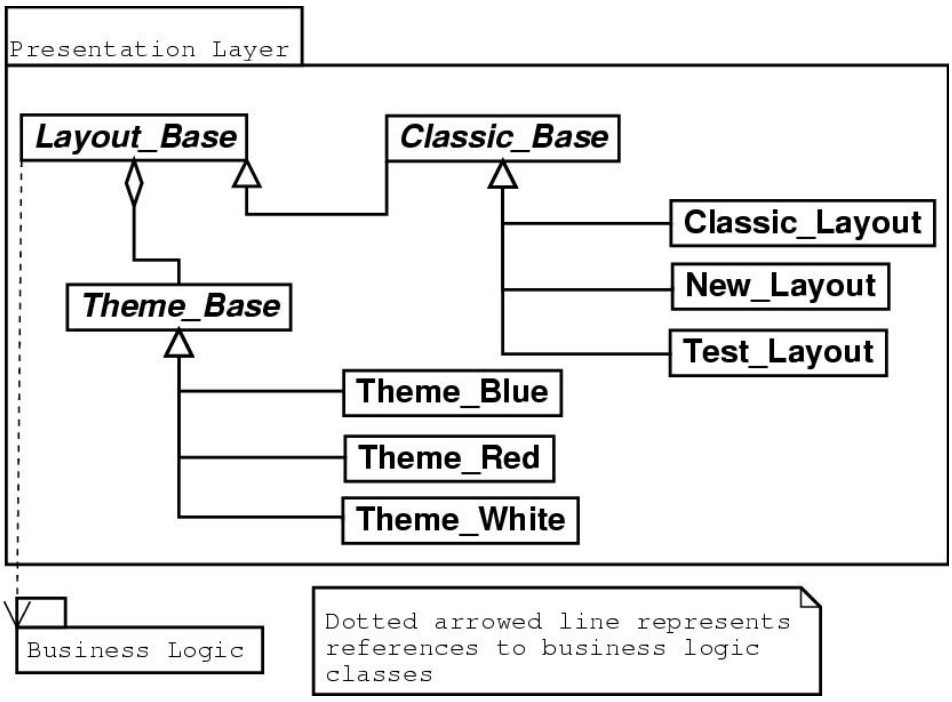
[14] Jak DML tak i DDL dotazů.

3 Stávající struktura systému

Tato část nabízí podrobný přehled jednotlivých vrstev uvedených výše v oddílu 2.2. Všímá si samotného návrhu a implementace jednotlivých vrstev (3.1, 3.2, 3.3) a odkrývá problémy, jež budou předmětem řešení práce (3.4).

3.1 Prezentační vrstva

Model na obrázku 3 přináší podrobný pohled na *prezentační vrstvu*.



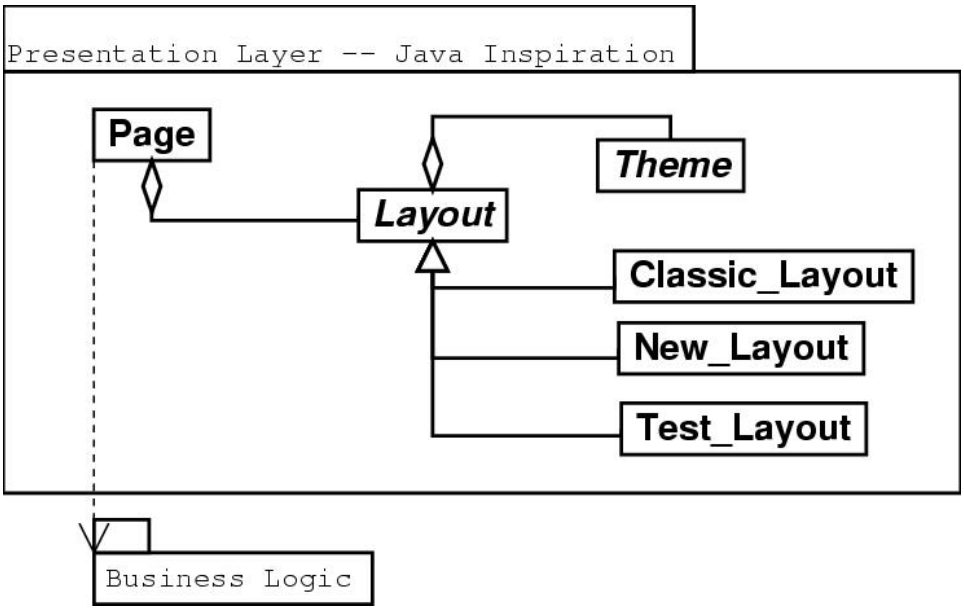
Obrázek 3: Prezentační vrstva

Třídy odvozené z `Theme_Base` představují konkrétní implementace různých témat, jež definují grafický vzhled stránek (ikonky, písmo). Třídy odvozené z `Layout_Base` představují jednotlivé typy rozvržení hlavních oblastí stránky (nabídka, pracovní plocha).

Model ve skutečnosti bohužel není úplně přesný. Třída `Classic_Base` má v reálné implementaci vždy pouze jediného potomka (třídu `Layout`), jež má tři implementace ve třech různých souborech, avšak díky vlastnostem skriptovacích jazyků (PHP) nedochází ke kolizi jmen — je vždy zaveden totiž jen jeden soubor (podle uložených preferencí uživatele). To umožňuje aplikačnímu programátorovi dědit vždy od třídy `Layout`, aniž by se staral o to, jaký definiční soubor je ve skutečnosti zaveden pro její implementaci.

To není ideální řešení. Vystoupí do popředí v okamžiku, kdy se toto chování navrhne bez použití dynamického zavádění definic tříd (což je vlastnost, jež by neměla být obecně používána).

Pokud se nechám inspirovat Javou mohl bych dojít například k modelu na obrázku 4.



Obrázek 4: Prezentační vrstva — inspirace Javou

Odlišnost mezi modely je zřejmá, jak je vidět, *třída* `Layout_Base` a její odvozeniny ve skutečnosti kombinují dvě rozdílné funkčnosti (odpovědnosti) v jednu, přičemž v tomto případě je vhodnější, aby byly oddělené:

třída `Layout_Base`

1. Funkčnost *co* má být zobrazeno.
2. Funkčnost *jak* to má být zobrazeno.

Jinými slovy, `Layout_Base` slouží jako kontejner pro prvky (*co*) a zároveň rozhoduje o umístění prvků (*jak*). Pokud však chceme transparentně rozhodovat o rozmístění prvků, je mnohem lepší oddělit tyto dvě odpovědnosti od sebe.

Na obrázku 4 je vidět kontejner `Page`, který (1) má vazbu na aplikační objekty a (2) podle potřeby agreguje příslušný `Layout`, jež řídí rozmístění prvků v kontejneru. Vzniká tak značně pochopitelnější model z těchto důvodů:

1. Je přehledný, logicky správnější a obecnější — jde o uplatnění design patternu *model–view–controller* (např. [13], [16] a [17]), blíže viz dále.

model–view–controller

2. Je analogií k modelu, jež používá řada grafických knihoven na různých platformách, viz například [15] či [13], takže vývojáři jsou na něj zvyklí.

Ještě zpět k obrázku 3. Zaujmout možná může i dědičnost `Layout_Base` → `Classic_Base`, přičemž ze třídy `Layout_Base` již není odvozena žádná jiná třída. Jedná se o pozůstatek nejranější implementace TUTOSu, který již nemá žádné opodstatnění.

3.1.1 XML

Dalším problémem prezentační vrstvy je neuspokojivé řešení XML exportů.

Nedávno byla v TUTOSu implementována možnost XML výstupu a vstupu. Než přistoupím k popisu problému s XML výstupem, je na místě uvést, jak vůbec probíhá generování HTML stránky (což byl donedávna jediný způsob presentace dat pro klientskou aplikaci).

Generování stránky Jak vyplývá z předchozího výkladu, o generování HTML stránky se starají třídy odvozené z `Layout`.

Každá HTML stránka kterou klient zobrazí by měla být (ale nemusí) odvozena z třídy `Layout`. Díky tomu má vývojář zajištěny tyto *služby poskytované prezentační vrstvou*:

služby prezentační vrstvy

1. Natažení uživatelem vybraných stylů (a vůbec zohlednění jeho nastavení týkající se vzhledu).
2. Zobrazení hlavní nabídky systému podle uživatelových oprávnění.
3. Zobrazeni vývojářem definovaných formulářů nebo HTML kódu v oblasti pracovní plochy. Toho je docíleno většinou přetížením několika příbuzných metod: `Layout::prepare()`, `Layout::navigate()` a `Layout::info()`.
4. Přístup ke kontextové nabídce (a možnost její změny).
5. Přístup ke kontextové nápovědě (a možnost její změny).
6. Přístup k lokalizačním souborům (ať již globálním, či těch zavedených vývojářem).
7. Přístup k mechanismu informování uživatele pomocí zpráv (například o výsledku databázové operace).

A tyto *služby poskytované jádrem aplikace*:

služby aplikační vrstvy

1. Několik aktuálních objektů přístupných v globálním jmenném prostoru (aktuální uživatel, spojení na aktuální databázi).

2. Ověření uživatele.

3. Zavedení užitečných knihoven funkcí.

Pokud stránka není odvozena z `Layout` (jak naznačuji výše), musí (měl by) si vše zajistit vývojář sám. Při dobré znalosti systému k tomu lze využít určitých „neveřejných“ funkcí, ale rozhodně se nejedná o doporučovaný postup.

XML A nyní zpět k XML. Při presentaci dat v jiné než HTML podobě (XML, CSV, ...) nejsou prakticky služby poskytované prezentační vrstvou nutné — buď by prezentační vrstva měla tyto služby poskytovat v modifikované podobě anebo zcela jinak[15].

Zároveň však služby poskytované jádrem aplikace jsou užitečné téměř vždy (autorizace, přístup ke globálním objektům a ke knihovnám).

Řešení celého problému je prosté — oddělit tyto dvě oblasti služeb (prezentační a aplikační).

Bohužel, v současné implementaci to sice je možné, ale rozhodně to není pohodlné.

Současná praxe je taková, že například při generování XML výstupu se raději odvodí třída z `Layout` (a tím se pro každou XML stránku generuje například zhola nepotřebná HTML nabídka aj.), než aby vývojář ručně zaváděl potřebné knihovny.

3.2 Aplikační vrstva

Aplikační vrstva je podrobněji zachycena na obrázcích 5, 6 a 7.

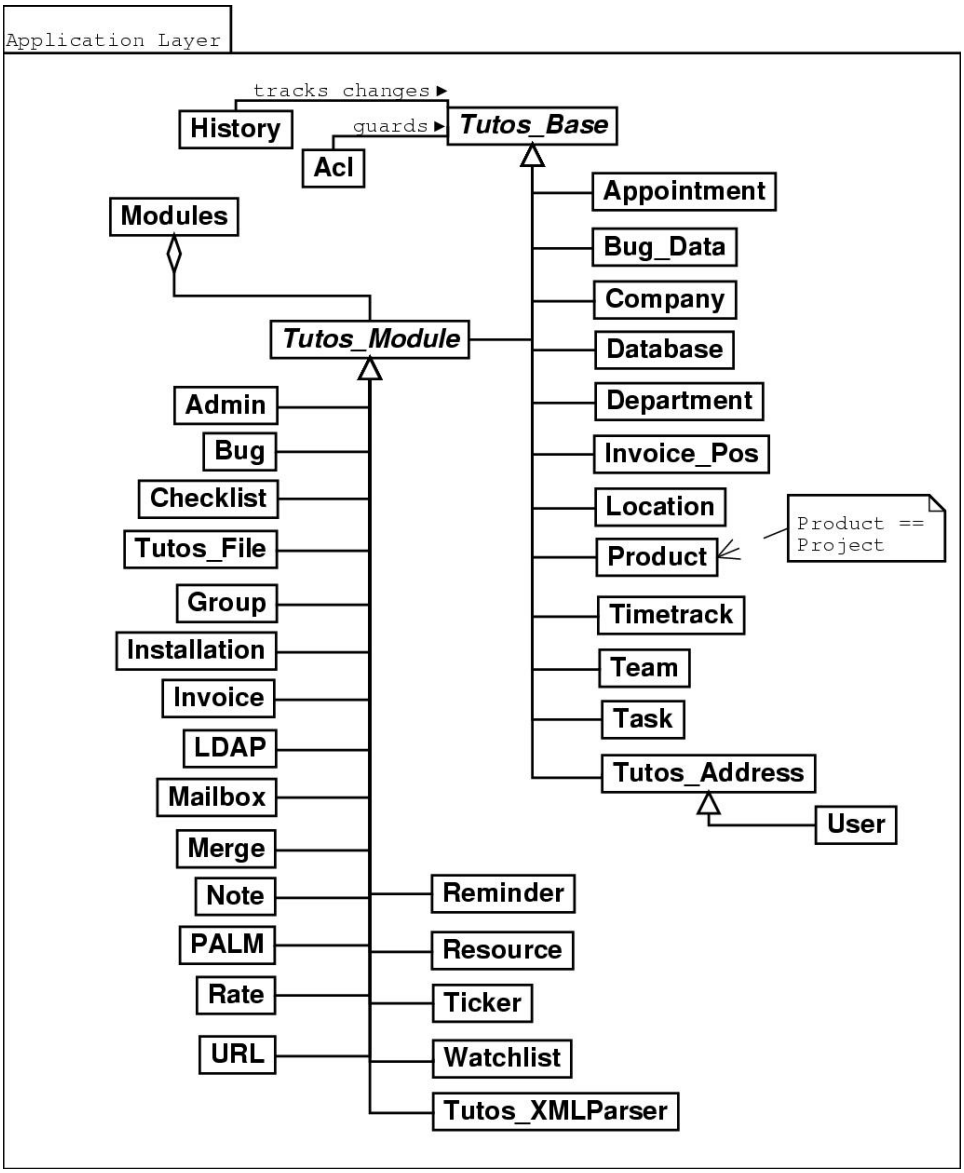
3.2.1 Dědičnost

Obrázek 5 je značně zjednodušený model, kde chybí (až na výjimky) asociace mezi třídami, kromě vazeb dědičnosti (což nahrazuji právě na modelech vyobrazených na obrázcích 6 a 7). Proč je zde vůbec model uveden v takovéto podobě? Slouží zde jako ilustrace toho, že:

1. TUTOS má skutečně mnoho aplikačních tříd.

2. Snaha zachytit je všechny na jednom modelu spolu se všemi jejich vazbami je nesmyslné.

3. Je z něho patrná jedna chyba v návrhu (právě ve vazbách dědičnosti), která by při jiném zobrazení nebyla tak zřejmá (viz níže).



Obrázek 5: Dědičnost

Jak si můžete na modelu s dědičností povšimnout (obrázek 5), množství tříd je odvozeno ze *třídy* `Tutos_Module` a ta sama je odvozena ze *třídy* `Tutos_Base`.

Pokud odpovědností `Tutos_Module` je definovat chování obecného rozšíření TUTOSu, jaký smysl pro něj může mít zajišťování služeb perzistence, které poskytuje třída `Tutos_Base`? K čemu ho ukládat do databáze?

Jde o chybu v návrhu — došlo ke sloučení dvou funkčností do jedné třídy z důvodu lenosti aplikačních programátorů, ač původní zamýšlené použití bylo jiné:

Každé rozšíření TUTOSu by mělo být implementováno skrze `Tutos_Module`. Díky tomu může jádro systému (representováno v tomto případě třídou `Modules`) snadno určit základní vazby na systém, položky nabídky, nápovědu, lokalizační soubory apod. Není naprosto žádný důvod ukládat tyto informace do databáze — tyto informace jsou příliš proměnlivé a pro každý modul jedinečné — proto jsou buď definovány jako interní datové struktury či uloženy v externích datových souborech.

Stejně tak i každá perzistentní třída by měla dědit od společné třídy `Tutos_Base`, díky čemuž získá databázové operace a přímý přístup k databázovým třídám, jež usnadní její práci s databází.

To, že došlo ke sjednocení těchto odlišných funkcionalit do jedné třídy, bylo dáno řadou jednoduchých rozšíření (například *správa poznámek*), povětšinou operujících nad jednou databázovou tabulkou, do které se ukládají datové atributy a jednou vazební databázovou tabulkou, v níž se uchovávají vazby na ostatní objekty systému[16]

Při tvorbě nového rozšíření bylo nutné:

1. Implementovat vlastní třídy, které by prováděli požadované operace (odvozené od `Tutos_Base`).
2. Definovat jednu třídu rozšíření (odvozenou) od `Tutos_Module` a v ní popsat závislosti na systému.

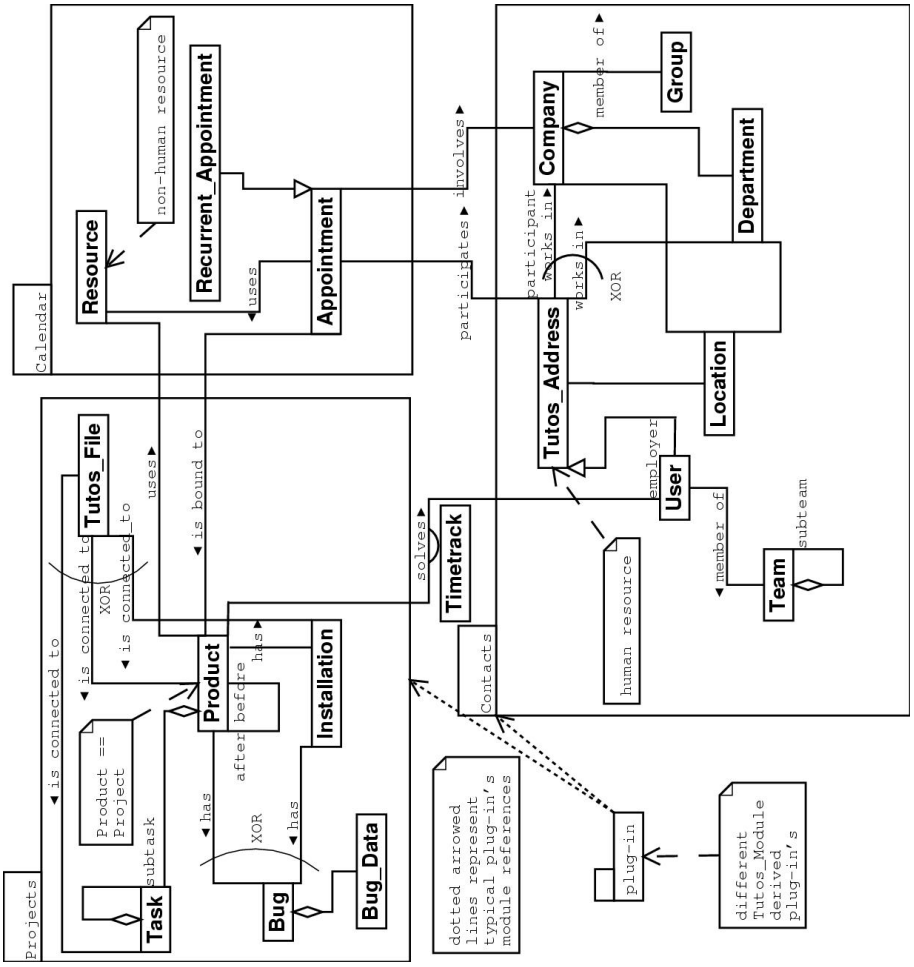
Pro nově vznikající rozšíření to byla, podle mnohých, zbytečná práce: Proč definovat dvě jednoduché třídy (a vytvářet dva soubory) když by bylo možné zkrátit a zpohodlnit dobu implementace dědičností?

Podle [18] jde o ukázkové špatné použití dědičnosti. Nevhodnost tohoto postupu je například patrná i z rozšíření `Invoice` či `Bug`, které pak definují ještě další třídy odvozené pouze z `Tutos_Base`.

[15] Proč generovat nabídku aplikace pro CSV formát? K čemu je mi nabídka v podobě HTML, když posílám PDF výstup?

[16] Systém generování a správa klíčů je v TUTOSu dost ojedinělý a umožňuje jednou vazební tabulkou propojit několik typů objektů. Viz 3.3.

3.2.2 Asociace



Obrázek 6: Asociace

objektový model asociací

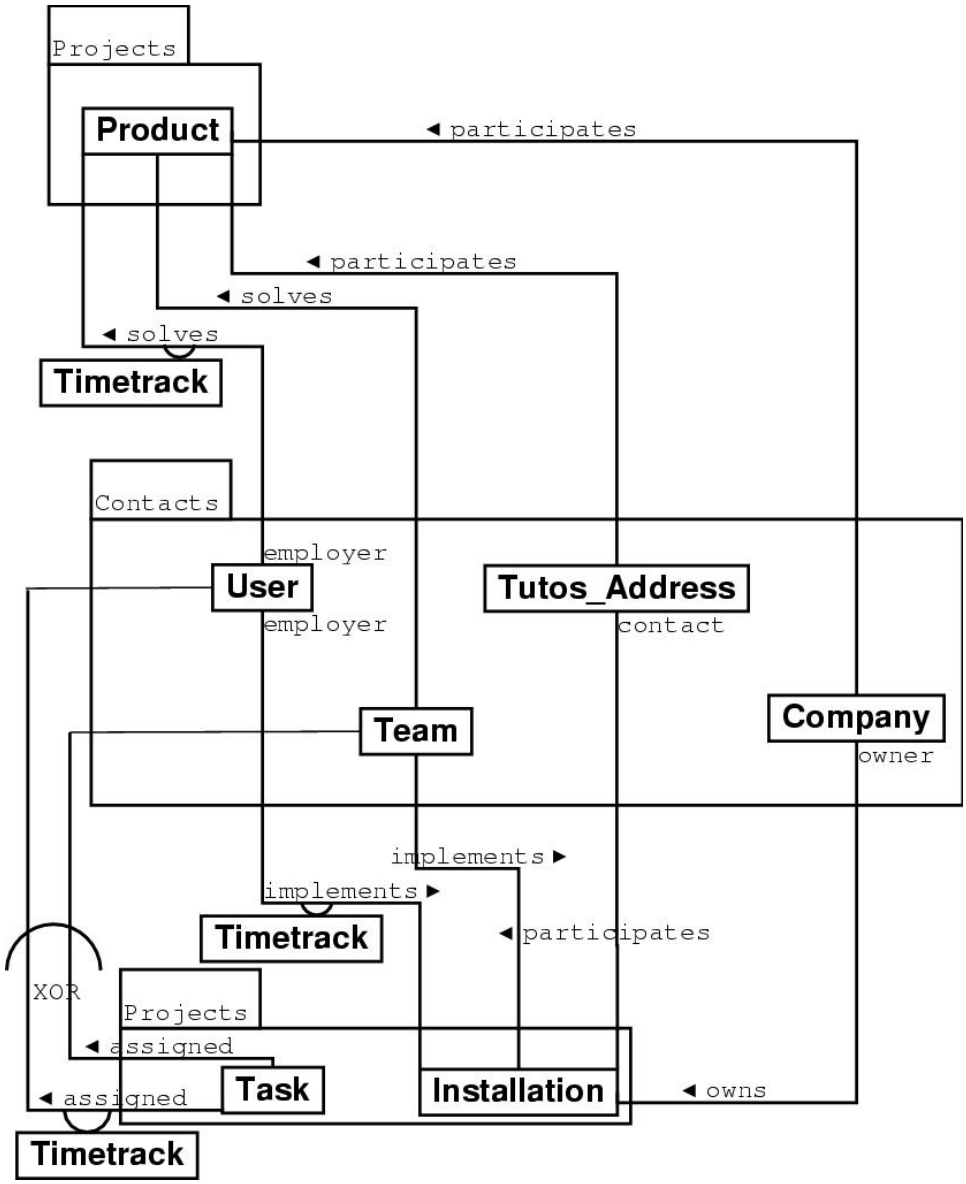
Model s asociacemi je zachycen na dvou obrázcích — 6 a 7. Obrázek 7 doplňuje obrázek 6 o vazby mezi podsystémy `Projects` ↔ `Contacts`.

Na obrázeku 6 jsou vidět hlavní uskupení tříd a jejich vztahy. Oproti obrázku 5 jsem si dovolil zanedbat některé třídy a rozčlenil jsem třídy z jádra TUTOSu do podsystémů[17].

Aby model byl přehledný, nezachytil jsem do něj tyto vztahy, jež je snadné si domyslet (platí i pro obrázek 7):

1. *Tvůrce* — každá perzistentní třída je asociována s třídou `User`. Instance této asociace (link) mezi perzistentním objektem a objektem

[17] Podotýkám, že *pravým* podsystémem (modulem) nemusí být vždy nutně to, co je odvozeno od třídy `Tutos_Module` a naopak.



Obrázek 7: Asociace — `Projects` ↔ `Contacts`

třídy `User` udává tvůrce objektu (tento link je neměnný).

2. *Vlastník* — každá perzistentní třída je asociována s třídou `User`. Link mezi perzistentním objektem a objektem třídy `User` udává aktuálního vlastníka objektu (tento link lze měnit), tj. lze předat vlastnictví a s ním související práva nad objektem.

3. Každá perzistentní třída je potomkem třídy `Tutos_Base`[18].

Implementace asociací *tvůrce* a *vlastník* je úzce spjata s generováním identifikátorů pro objekty — blíže viz 3.3 — na tomto místě pouze poznamenám, že každý objekt je vybaven jedinečným identifikátorem v rámci celého systému, takže teoreticky lze skutečně propojit linkem libovolný objekt s libovolným jiným objektem. V praxi však samozřejmě existují vazby, jež systém nepodporuje, tj. neumožňuje je zadat a zobrazit prostřednictvím uživatelského rozhraní a nelze je uložit do databáze.

Z tohoto tvrzení též nepřímo vyplývá jeden fakt. Vazby mezi třídami, jež jsou zachyceny na obrázku 6, jsou pouze ty „doporučené“, tak jak bylo původně zamýšleno, aby systém pracoval. Během vývoje TUTOSu došlo k různým rozšířením, které sice vzájemné reference v systému značně komplikují, ale uživatelům poskytují velký komfort a jsou známkou jisté jedinečnosti systému. Mezi vazby, jež na obrázku 6 zachyceny nejsou, patří například asociace, umožňující:

1. Propojit `Tutos_File` přímo ke třídě `User`.

2. Sledovat `Timetrack` (odpracované hodiny) pro `Team` (obrázek 7).

3. Přidat `Task` (úlohu) přímo k uživateli

Pokusit se zobrazit vše najednou by jistě nikam nevedlo, navíc, pro samotné úvahy, jak je systém navržen to není podstatné, stačí si uvědomit, že toto vzájemné odkazování je možné a vědět, jak ho implementovat, což je mimo jiné probíráno v oddílu 3.3.

Podsystémy Na obrázku 6 jsou zachyceny 3 hlavní podsystémy:

1. `Projects`. Tento podsystém obsahuje vše, co souvisí s projekty a jejich řízením.

   Projekt (`Product`) může být hierarchicky rozčleněn do úkolů (`Task`), které mohou být ještě dále děleny na podúkoly. Mezi projekty mohou být vazby časové následnosti.

[18] Některé navíc, poněkud nešťastně, i potomkem `Tutos_Module` viz výše.

Zavedením projektu u zákazníka vzniká instalace (`Installation`). Ta může (stejně jako projekt) obsahovat chyby (`Bug`). K projektům, úkolům či instalacím se mohou vázat doprovodné soubory (`Tutos_File`).

2. `Contacts`. Tento podsystém spravuje kontakty (fyzické (`Tutos_Address`) i právnické (`Company`) osoby).

   Právnické osoby mohou být rozděleny na jednotlivá oddělení (`Department`).

   Všechny kontakty mají nějakou adresu (sídlo, místo bydliště = `Location`).

   Fyzické osoby často pracují v nějaké společnosti (či jejím oddělení). Společnosti mohou být kategorizovány do skupin (`Group`).

3. `Calendar`. Součástí jádra systému je i plánování času. Schůzky mohou být napojeny na projekt a mohou zainteresovat nějaký kontakt.

   Speciálním případem schůzky je opakovaná (pravidelná) schůzka (`Recurrent_Appointment`).

Vazby `Projects` ↔ `Contacts` Modul `Projects` je velmi úzce napojen na modul `Contacts`. Vzniká mezi nimi mnoho vazeb, jež vesměs vyjadřují, kdo má za co odpovědnost, viz obrázek 7.

Typicky je kardinalita vazeb `m:n`. Pokud se odpovědnost týká uživatele systému (typicky pracovníka firmy), jsou s vazbou spojeny i odpracované hodiny (`Timetrack`).

3.3 Databázová vrstva

Jak je vidět z obrázku 8 a z návrhu `java.sql.*` (např. [13]), tři třídy *databázové vrstvy* TUTOSu skutečně kombinují několik funkčností, jež jsou v Javě implementovány v řadě tříd a rozhraní.

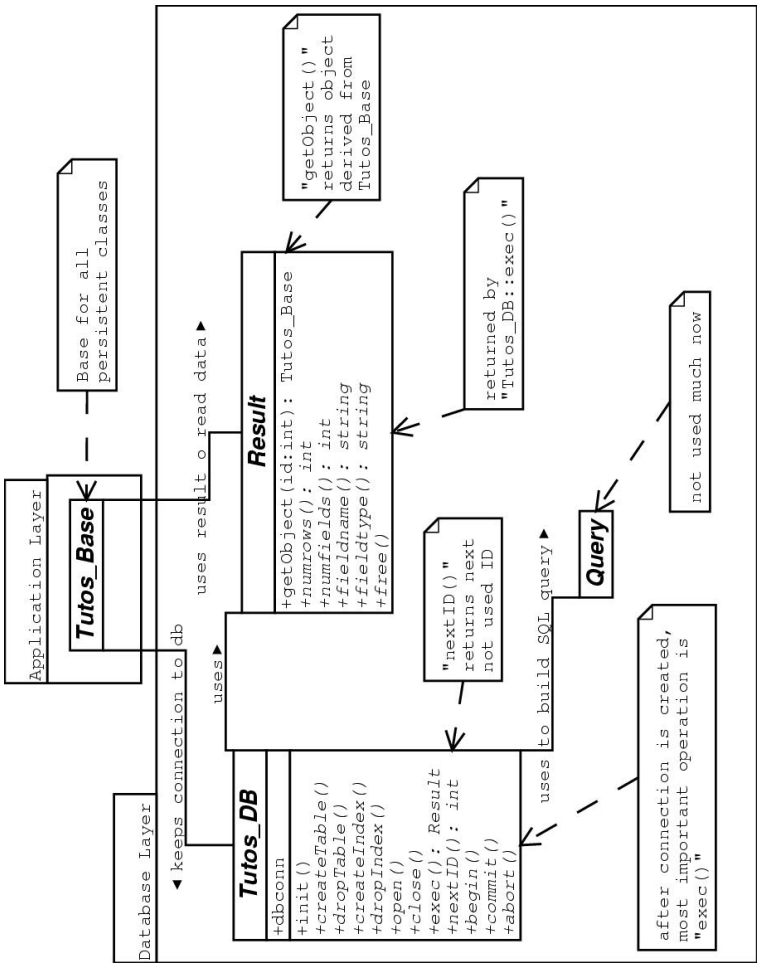
Následující tabulka (převzata z části 2.2.3) to elegantně shrnuje:

Tutos_DB	: Driver, Connection, DatabaseMetaData, Statement
Query	: Statement, PreparedStatement
Result	: ResultSet, ResultSetMetaData

Toto databázové rozhraní provází TUTOS již od jeho prvních verzí. Když si uvědomíme, kdy TUTOS vznikal (oddíl 1.2.1), je to na tehdejší dobu a PHP skutečně dobrá databázová abstrakce.

Databázová vrstva navíc poskytuje 2 ne zcela typické funkce:

1. `Tutos_DB::nextID()`, jež poskytuje nové volné ID, které může být použito pro nově vkládaný objekt.



Obrázek 8: Databázová vrstva

2. `Result::getObject()` — tato funkce vybere z databáze objekt na základě jeho jedinečného ID.

Nezávislost na databázi je zajištěna dědičností. Pro každou podporovanou databázi (viz 2.2.3) je z abstraktní třídy databázové vrstvy (`Tutos_DB` a `Result`) odvozena nová třída, jež příslušným způsobem implementuje příslušné metody.

Třída `Query` již není používána. Sloužila ke konstrukci DDL (zřídka i DML) SQL dotazů, které byly konstruovány i na základě metadat, jež si samotný systém udržuje o databázi (názvy tabulek, typy sloupců apod.). Nyní má význam pouze během instalace TUTOSu, kdy tyto informace nemohou být zjištěny z databáze, či dodány vývojářem v kódu.

Přestože je databázová vrstva navržena pro potřeby TUTOSu dobře, lze mít pár výhrad:

1. Z osobní zkušenosti vychází největší výtka — ve většině databázových abstrakcích se rozlišují (při běžné práci) tyto třídy (pro jejich označení použiji označení z Javy):

   - `java.sql.Driver` — přebírá parametry pro spojení a je schopna vytvořit spojení na databázi.
   - `java.sql.Connection` — spojení na databázi. Representuje databázi jako takovou, umožňuje nastavení řízení transakcí a skrze další třídy poskytuje pohodlný přístup k datům v ní uložených.
   - `java.sql.Statement` — představuje SQL dotaz položený databázi.
   - `java.sql.ResultSet` — představuje výsledek SQL dotazu.

   Jak je vidět, v TUTOSu sice jsou k dispozici všechny tyto možnosti, ale nemusí být k nalezení ve třídách, kde je na ně vývojář zvyklý.

2. S novými verzemi databází přicházejí nové vlastnosti těchto databází (zejména MySQL v nových verzích). Pokud chceme tyto nové vlastnosti využít, je nutno tyto vlastnosti implementovat (tj. je na vývojářích TUTOSu, aby zajistili dostupnost nové funkce) a to zabírá čas, jež může být využit lépe.

Oba tyto nedostatky mohou být vyřešeny velmi prostě — nahradit databázovou vrstvu nějakou běžně užívanou komponentní knihovnou, jako například [19] či [20].

3.4 Shrnutí

Po podrobnějším představení jednotlivých vrstev aplikace se již vynořily nějaké nedostatky v návrhu a implementaci.

Z celkového pohledu nemám zásadnější výtky. TUTOS je aplikace typu transakční manažer a jako celek je implementován podle doporučených postupů. Nicméně:

1. V prezentační vrstvě se objevuje problém ve špatném návrhu (dědičnost místo agregace), viz oddíl 3.1 a obrázek 4, a špatně řešený výstup do jiných formátů (generování výstupní stránky není ideální), blíže viz 3.1.1.

2. V aplikační vrstvě je nevhodně použitá dědičnost, z důvodu lenosti vývojářů, viz oddíl 3.2.1. Jinak není k řešení samotné aplikační logiky žádných výtek.

3. V databázové vrstvě nejsou žádné větší nedostatky, avšak použití jiné (standardní) knihovny namísto vlastně udržovaného kódu by mohlo zlepšit a zprůhlednit vlastnosti databázové vrstvy.

4 Úpravy databázové vrstvy

Tato část přináší podrobnější návrhy a diskuzi k úpravám databázové vrstvy, jak vyplývá z oddílu 3.3 a 3.4.

V části 3.4 bylo řečeno, že *databázová vrstva* netrpí většími nedostatky, avšak použití nějaké obecné knihovny pro přístup k databázi může vlastnosti této vrstvy zlepšit.

4.1 Nová databázová knihovna

Použitím nové databázové knihovny získá TUTOS především ve standardizaci databázového kódu a využití některých nových vlastností databázových strojů zcela bez námahy vývojářů.

Požadavkům systému budou vyhovovat obě v současné době rozšířené knihovny, jež jsou užívány v řadě projektů — `ADOdb` [19] a `PEAR::DB` [20].

Použitím takovéto knihovny bude dosaženo těchto přínosů:

1. *Standardizace databázového kódu.*

   Jedná se o značně rozšířené knihovny, jež čerpají inspiraci z řady dalších podobných projektů (například JDBC [23]).

   Vývojáři je umí používat, jsou rozsáhle testovány a poskytují velmi dobré odstínění od konkrétního databázového stroje.

2. *Odpadne nutnost udržovat databázový kód.*

   V současnosti by podpora transakčního zpracování či generátorů v nových verzích MySQL musela být komunitou vývojářů TUTOSu implementována, pokud bude zájem tyto služby využívat. Použitím dostupné knihovny tuto vlastnost bude možné ihned využít.

3. *Podpora více databází.*

4. *Alternativní přístup k datům.*

   Budou podporovány alternativní způsoby přístupu k databázím (Embedded SQL a jiné).

5. *Parametrizovatelné SQL dotazy.*

   Parametrizovatelné SQL dotazy znamenají automatickou konverzi datových typů a rychlejší vykonávání příkazů.

6. *Ošetřování chyb pomocí výjimek.*

Knihovna `ADOdb` navíc oproti běžným databázovým abstrakcím nabízí:

- Snadnější přenositelnost mezi databázemi — vlastní datové typy pro čísla, řetězce a časové typy, včetně jejich transparentní vnitřní konverze mezi PHP datovými typy na databázové typy.
- Podporu specifických vlastností daných databázových strojů (což v TUTOSu nebude využito — tam je hlavní přenositelnost).
- Množství alternativních způsobů, jak zpracovávat výsledky dotazů:
  1. Své vlastní metody pro přístup k datům, podobné těm z JDBC [23].
  2. `PEAR::DB` rozhraní
  3. Iterační rozhraní (*iterator* design pattern, viz [22]), jež je možné využít například v PHP verze 5. *iterator*
- Podporu relace (session) na úrovni databáze (opět není třeba používat v TUTOSu).

4.2 Speciální požadavky na databázovou vrstvu

V části 3.3 jsou jmenovány dvě speciální vlastnosti databázové vrstvy, implementované ve dvou metodách (převzato z části 3.3):

1. `Tutos_DB::nextID()`, jež poskytuje nové volné ID, které může být použito pro nově vkládaný objekt.
2. `Result::getObject()` — tato funkce vybere z databáze objekt na základě jeho jedinečného ID.

Implementace těchto dvou funkčností je samozřejmě možná i s použitím nové knihovny.

`Tutos_DB::nextID()` je implementován pomocí generátorů. Tam, kde nejsou generátory implementovány (MySQL) je použito speciální tabulky s jedním sloupcem typu celé číslo a chování generátoru je emulováno v kódu.

`Result::getObject()` je implementován jako *factory method* design pattern, viz [21]. Tento design pattern slouží k tvorbě objektů různého typu se stejným chováním, ale různou implementací tohoto chování (tj. objektů odvozených od stejné základové třídy), kdy typ vznikajícího objektu je určen nějakými vstupními parametry. Při tvorbě objektu na základě jeho jednoznačného identifikátoru (vstupní informace) je postupováno takto: *factory method*

1. Alternativní krok. *Načtení objektu z vyrovnávací paměti.* Během zpracování požadavku si TUTOS udržuje již jednou načtené objekty ve vyrovnávací paměti a programátor si může vyžádat získání objektu právě z této paměti. Pokud objekt není nalezen, je načten z databáze, jak uvádí další postup.

2. *Zjištění typu objektu.* TUTOS si udržuje databázovou tabulku, kde mapuje identifikátory objektů na jejich typy. Dotazem do této tabulky se získá typ objektu (representován jako celé číslo na úrovni databáze a jako konstanta na úrovni aplikace).

3. *Vytvoření objektu příslušného typu.* (Vytvoření probíhá v rozsáhlém příkazu `switch`.)

4. *Volání metody* `read()` *na objektu.* Tímto voláním jsou do objektu načtena data z databáze. Jde o přetíženou metodu zděděnou od třídy `Tutos_Base`, viz 5.1.

Jak je vidět, za těmito operacemi se neskrývá žádné tajemství a obě tyto metody lze bez potíží implementovat s libovolnou knihovnou.

Zůstává pouze otázka, zda tyto metody ponechat ve třídách, v nichž nyní jsou (rozuměj: v obdobných třídách nové knihovny), nebo je začlenit do jiného jmenného prostoru.

- `Tutos_DB::nextID()` by měla být ze svého charakteru dostupná vždy a měla by probíhat mimo transakční kontext (aby byla rychlá, viz [14]) s tím, že musí být zajištěna atomicita inkrementu čísla v databázi. Pokud databázový stroj podporuje generátory, je/může být toto zajištěno.

  Proto je její začlenění ve třídě reprezentující databázové spojení v pořádku (jakmile je dostupné databázové spojení, mohu začít bez ohledu na cokoli generovat nové identifikátory).

- `Result::getObject()` je implementován jako dotaz do databáze. Měl by probíhat v rámci aktuálního transakčního kontextu, který je řízen aplikačním programátorem (pouze on ví, jaké jsou jeho úmysly s načítaným objektem).

  Proč by mělo být nutné vytvářet objekt třídy `Result`, když mým jediným cílem je volat jeho metodu `getObject()`?

  Nazíráno z tohoto hlediska je začlenění do `Result` nevhodné. I tato metoda by měla být dostupná v okamžiku, kdy je dostupné spojení na databázi.

Závěrem lze shrnout: Buď tyto metody implementovat jako globální funkce, nebo je vnořit do jmenného prostoru třídy představující spojení na databázi[19].

Nejspíše je vhodné modifikovat kód databázové knihovny co možná nejméně, takže bude žádoucí definovat tyto metody jako globální funkce.

[19] V samotném TUTOSu je ve skutečnosti `Result::getObject()` implementován tak, že volá globální funkci `getObject()`, které je předáno spojení na databázi, které si interně objekt třídy `Result` udržuje.

4.3 Dopady na systém

Začlenění nové knihovny do systému si vyžádá tyto změny v implementaci:

- Přepsání všech databázových metod zděděných od `Tutos_Base`, viz 5.1.

  To v řadě případů bude znamenat pouze změnu v deklaracích proměnných, v některých (zejména u metod `Tutos_Base::delete()`, `Tutos_Base::readHistory()`, či `Tutos_Base::deleteHistory()`)

  Avšak u metod jako `Tutos_Base::read()` a `Tutos_Base::save()`, jež jsou ve většině tříd přetěžovány, to bude znamenat pozměnit většinu kódu. Jak je patrno z obrázku 5, tříd k přepsání je přes 30.

- Změnu všech stránek, které k databázové vrstvě přistupují přímo (systém má otevřenou architekturu, 1.2.4). V sekci 2.2.1 je uvedeno, že typicky se k jednomu objektu tvoří 4 stránky.

  Stránky zobrazující (1) seznam objektů a (2) vyhledání objektu v databázi patří do této kategorie. To je přibližně 60 stránek k přepsání (2×30).

- Drobné úpravy v inicializačním kódu.

- Implementace dvou specifických funkčností TUTOSu za využití nové knihovny.

Je vidět, že by to znamenalo množství práce. Avšak zejména při použití knihovny `ADOdb` a jejího iteračního rozhraní by se mohl kód značně zpřehlednit a zjednodušit. Pokud by TUTOS dále využil vlastnosti této knihovny (definice časových typů) mohlo by se dostavit užitek i v jiných oblastech (přenositelnost dat mezi databázemi[20]).

4.4 Shrnutí

Databázová vrstva tak, jak je navržena, implementována a používána nyní, neobsahuje žádnou zásadní chybu, spíše drobné nedostatky a nutnost udržovat kód aktuální.

Její změna by přinesla zejména (1) standardizaci de–facto (tj. snazší použití pro vývojáře zvyklé na jiné projekty) a (2) odpadla by nutnost udržovat databázové třídy. Nepříjemnou skutečností by však byla nutnost přepsat značnou část kódu.

[20] TUTOS například podporuje funkci *zrcadlení databáze*, kdy jsou data z jedné databáze přenesena do druhé. Lze tak řešit například zálohování.

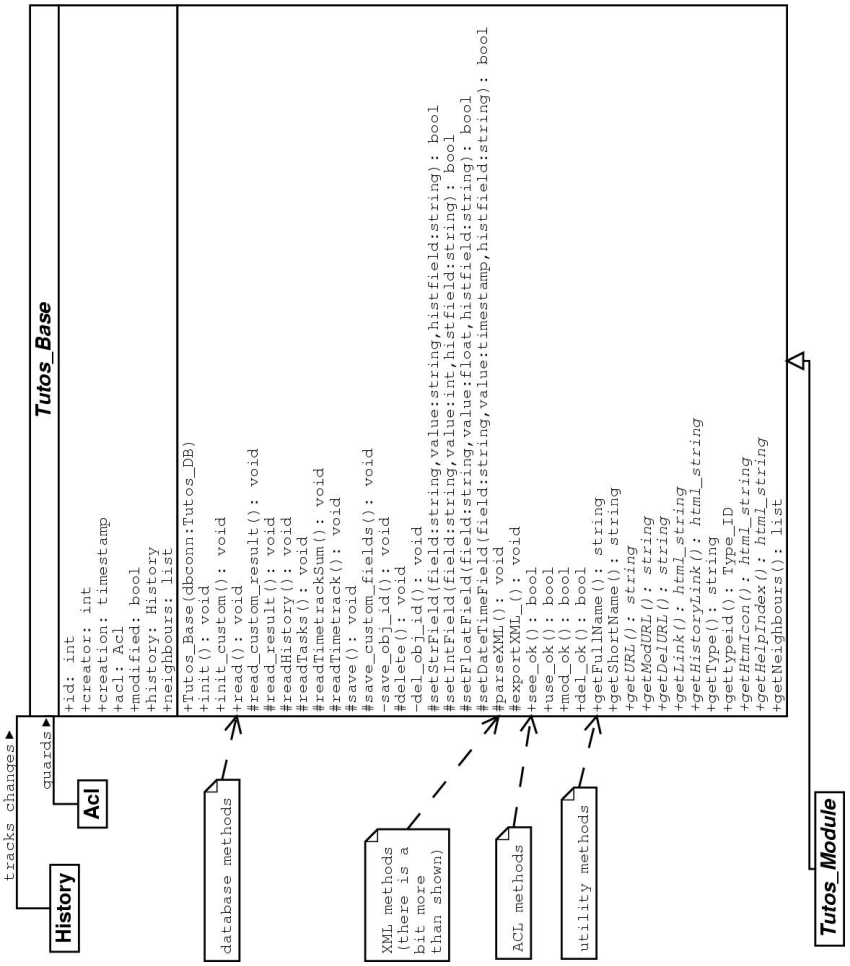
5 Úpravy aplikační vrstvy

Tato část se věnuje změnám na úrovni aplikační vrstvy, jež byly představeny v oddíle 3.2. Dále nabízí několik dalších postřehů k implementaci základových tříd.

Za největší problém *aplikační vrstvy* bylo označeno nevhodné použití dědičnosti (3.2.1), tj. odvození třídy `Tutos_Module` ze třídy `Tutos_Base`.

Modely na obrázcích 9 a 10 poskytují detailnější pohled na tyto třídy.

5.1 Třída `Tutos_Base`



Obrázek 9: Třída `Tutos_Base`

Na obrázku 9 je možné nalézt tyto skupiny metod ve *třídě* `Tutos_Base`:

- *Databázové metody.* To jsou metody, jež by určitým způsobem měly

být přetíženy v odvozených třídách — každá třída ví sama nejlépe, které atributy a jak uložit do databáze.

Některé se však nepřetěžují: `Tutos_Base::save_obj_id()` a `Tutos_Base::del_obj_id()`, nebo je to ve většině případů není nezbytné (např. `Tutos_Base::readHistory()`)

- *XML metody.* Těchto metod je použito pro podporu XML exportu (a importu).
- *ACL metody. ACL* metody se používají pro kontrolu prováděných operací nad objektem.
- *Užitečné funkce.* Jedná se o skupinu „šikovných“ operací, jež je dobré mít stále po ruce, jako například získání textové reprezentace objektu.

Všechny tyto metody se jistě vztahují právě k hierarchii tříd odvozených z `Tutos_Base` a ač můžeme diskutovat o zlepšení návrhu (možná by stálo za to uvažovat o rozdělení třídy kvůli jejímu rozsahu, viz dále), nelze označit její předpis za nevhodný.

5.2 Třída `Tutos_Module`

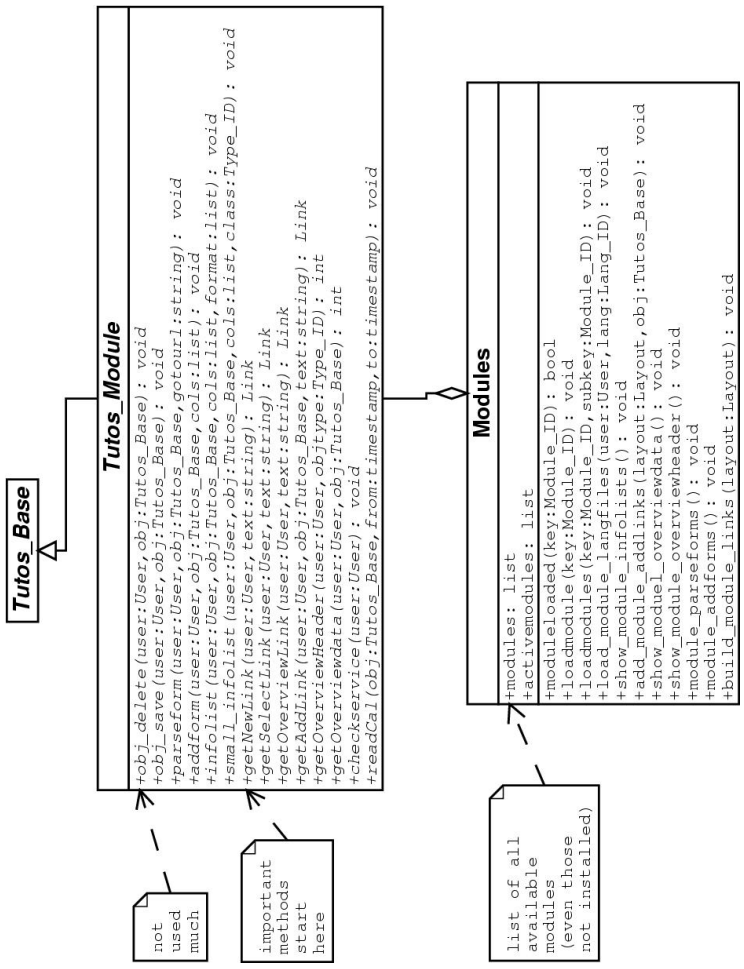
Na obrázku 10 je společně se *třídou* `Tutos_Module` zobrazena ještě třída `Modules`, která je s touto třídou velmi úzce spjata. Při pohledu na operace obou tříd je možné říci, že až právě třída `Modules` a její operace dává třídě `Tutos_Module` její skutečný smysl.

Význam a funkce těchto tříd bude nejviditelnější z příkladu.

5.2.1 Tvorba vlastního modulu

Pokud chceme rozšířit chování TUTOSu, je nejvhodnější udělat to skrze definici vlastního modulu, než bezhlavou úpravou existujícího kódu. Vývojář tak získá řadu funkčnosti automaticky díky využití aplikačního rámce (application frame), jež poskytují třídy `Tutos_Module` a `Modules`. Konkrétní přínosy jsou:

- Začlenění do ACL modulů — jednotlivým uživatelům lze povolit/zakázat využití modulu.
- Automatické využití lokalizačních souborů poskytnutých společně s modulem.
- Jednoduchá instalace modulu do libovolné distribuce TUTOSu.
- Podporu při tvorbě GUI, pokud vývojář užívá vlastních tříd odvozených ze základové třídy `Layout` (doporučeno).



Obrázek 10: Třída `Tutos_Module`

Při tvorbě vlastního modulu je možné postupovat podle několika málo jednoduchých kroků. Za předpokladu hotové aplikační logiky lze tvorbu nového modulu v současnosti shrnout takto:

1. *Vytvoření konfiguračního souboru modulu.* Úkoly konfiguračního souboru (podle konvence označovaného jako `mconfig`) jsou dva:

   (a) Přidat modul do globálních datových struktur TUTOSu — konkrétně do několikrát zmiňované třídy `Modules`.

   (b) Definovat všechna globální nastavení nutná pro tento modul (globální proměnné, jimiž může uživatel měnit chování modulu, odkazy na knihovny třetích stran).

   Konfigurační soubor má pro jednoduché moduly typicky několik málo řádek zdrojového kódu (cca 30).

2. *Definovat vlastní třídu odvozenou z* `Tutos_Module`*.* Vůbec není nutné definovat všechny metody rozhraní třídy `Tutos_Module` (ty nepřepsané nevykonávají žádný kód). V praxi se u jednoduchých modulů vyplatí:

   (a) Přetížit `Tutos_Module::getOverviewLink()`. Tím může TUTOS generovat odkaz na hlavní HTML stránku našeho modulu.

   (b) Přetížit další metody podle potřeby, například `Tutos_Module::getNewLink()`, jež napomáhá při generování kontextové nabídky TUTOSu.

   (c) Definovat soukromé metody pro využití v rámci modulu, vhodným kandidátem může být například metoda pro sestavení kontextové nabídky modulu (pokud se nemění) apod.

   Typicky jde opět o několik málo řádek zdrojového kódu (cca 50–100).

3. *Vytvořit GUI našeho modulu.* V naprosté většině to znamená vytvořit HTML stránky, při jejichž tvorbě může značně pomoci odzvození ze společných základových tříd (`Layout`).

   Zde pochopitelně množství nutného kódu záleží na propracovanosti rozhraní. Může být značné (rozsáhlé formuláře, složitá kontrola vstupních dat apod.).

4. *Instalace.* Doporučuje se ukládat všechny soubory nového modulu do vlastního adresáře. Pak stačí adresář překopírovat do kořenového adresáře instalace TUTOSu a zavést konfigurační soubor modulu (`mconfig`) do globálního konfiguračního souboru. Od tohoto okamžiku již lze nad modulem nastavovat oprávnění a používat jeho HTML stránky.

5.2.2 Odstranění dědičnosti

Z tohoto krátkého příkladu a části 5.1 je jistě patrné, že třída `Tutos_Module` a `Tutos_Base` mají velmi málo společného — jejich odpovědnosti jsou naprosto odlišné.

Zatímco instance `Tutos_Base` představuje perzistentní objekt systému a stará se o jeho ukládání a načítání, XML representaci a obsluhuje s ním spojené oprávnění, instance třídy `Tutos_Module` představuje rozšíření TUTOSu a její odpovědností je komunikovat s jádrem systému (objektem třídy `Modules`). Objekt třídy `Tutos_Module` nemá zájem o to být uložen v databázi, natož aby byl převoditelný do XML podoby.

Jediným společným prvkem je využití *ACL*, avšak v případě objektu `Tutos_Module` jde o kvalitativně zcela odlišnou formu oprávnění[21] — jde o oprávnění nad částí funkčnosti celé aplikace (například nad *správou kontaktů*), zatímco v případě `Tutos_Base` jde o oprávnění nad jedním perzistentním objektem (například jedním konkrétním kontaktem).

Lze konstatovat, že požadavek na oddělení těchto tříd (tj. zrušení třídy `Tutos_Base` jakožto předka třídy `Tutos_Module`), je oprávněný.

Když už se měla použít dědičnost, byla by asi vhodnější vícenásobná dědičnost (a považovat ji za implementaci rozhraní).

5.3 Další diskuze k aplikační vrstvě

5.3.1 Zpracování vstupu uživatele

TUTOS Při zpracování vstupu se v TUTOSu typicky provádí:

- Kontrola vstupních dat získaných z formulářových polí. Jde o běžné kontroly typu platné datum, rozsah čísla, ...
- Kontrola složitějších integritních omezení — jsou oprávnění uživatele dostatečná?, existuje objekt, na který je odkazováno? je vstup uživatele jednoznačný?, je služba (SMTP například) k dispozici?, ...

U některých metod `Tutos_Module` je uvedeno, že nejsou příliš užívány (viz obrázek 10). Jsou součástí třídy `Tutos_Module` od prvotních návrhů systému a jejich cílem bylo pomoci vývojáři zaregistrovat data získávaná z formulářů a pomoci s jejich automatickým zpracováním (kontrola typů, základní operace — získání referencí na objekty apod.).

Ukazuje se však, že každý formulář je příliš specifický a nelze (až na nejjednodušší případy) tuto funkčnost nějak generalizovat pro kontrolu více

[21] Potvrzením této skutečnosti je i to, že se uchovává ve zvláštních tabulkách a interpretuje se jinak.

formulářů najednou, resp. je nevhodné ji takto pojímat a umísťovat do třídy `Tutos_Module`, blíže k tomuto tématu viz kapitola 6.

Proto je většinou ponechána kontrola dat z formulářů buď (1) na úrovni prezentační vrstvy, která, než nastaví hodnoty objektů z formulářů, provede kontrolu ve vlastním kódu, anebo (2) přímo na aplikační třídě odvozené z `Tutos_Base`.

Ze složitějších kontrol je nejčastější:

1. *Ověření nějakého oprávnění.* K tomuto účelu velmi dobře slouží ACL metody třídy `Tutos_Base`.
2. *Ověření nějaké aktuální hodnoty.* Není určen žádný postup.
3. *Ověření nějaké hodnoty na odkazovaném objektu.* Je nutno získat odkazovaný objekt a pak na něm provést kontrolu jako výše. Odkazovaný objekt lze v TUTOSu nejsnáze získat pomocí funkce `Result::getObject()` anebo pomocí globální funkce `getObject()`.

Obecný postup Obecným vhodným postupem vstupních kontrol uplatňovaným například v [24] či [25] je tento:

1. Ponechat kontrolu vstupních dat na prezentační úrovni.
2. Definovat třídu pro každý formulář (tedy nikoli jednu třídu pro všechny formuláře, jak je to zamýšleno v `Tutos_Module`), u které bude:
   (a) Vhodným způsobem definována *invarianta*.
   (b) Jednotným způsobem zajištěno informování o vzniklých problémech.

Invarianta je vlastnost objektu, která ho udržuje v dobře definovaném stavu, viz např. [18].

Toto oddělení odpovědností kontroly vstupních dat od jejich použití na aplikační úrovni navíc přináší jednak (1) vhodný způsob, jak skrze relaci (session) předávat informace o zadaných hodnotách a zároveň jejich chybách u bezstavového protokolu HTTP (vše zůstává pohromadě a je přenášeno minimální množství informace), a jednak (2) dále snižuje závislost mezi vrstvami aplikace, viz též kapitola 6.

Při použití v jazyku jako PHP vystupují napovrch ještě další výhody:

- Bezproblémová typová konverze (to je velmi nepříjemné při zpracování HTML formulářů v Javě).
- Podpora datových typů jako je asociativní pole na úrovni jazyka. To v mnoha případech znamená, že není nutno definovat zvláštní třídu — hodnoty z formulářů HTTP lze velmi snadno a s veškerým komfortem uchovat právě v asociativním poli. Zbývá pouze stanoveni invarianty.

S novou verzí PHP (verze 5) přichází podpora výjimek a bylo by škoda jich nevyužít. Pokud se komunita vývojářů rozhodne k využití výjimek, bude stejně nutno přepsat nějaké části systému a není důvod významněji nezměnit i ošetření vstupních dat.

5.3.2 Rozdělení třídy `Tutos_Base`

Třída `Tutos_Base` je velmi rozsáhlá. Nabízí se otázka, zda ji nerozdělit do více tříd. její metody lze rozdělit takto (převzato z části 5.1):

- *Databázové metody.*
- *XML metody.*
- *ACL metody.*
- *Užitečné funkce.*

Databázové metody tvoří základ třídy.

XML metody jsou vhodným kandidátem na vyloučení ze dvou důvodů:

1. Ne všechny odvozené třídy je přetěžují (tj. používají). U některých je XML export delegován na jiné třídy — typicky tam, kde je vztah agregace. Příkladem může být faktura `Invoice` a položka faktury `Invoice_Pos`.
2. Jde o obecnou vlastnost, která nemusí být pouze doménou tříd odvozených od `Tutos_Base`.

Volání *ACL metod* je z hlediska designu delegováno na třídu `Acl`. V kódu je třída `Acl` implementována pomocí několika bitových masek a ACL metody operují přímo nad těmito maskami. Není vhodné vyloučit — není kam.

U *užitečných funkcí* by bylo potřeba postupovat funkci od funkce (například na získání textové representace objektu lze též nahlížet jako na obecnou vlastnost, již lze realizovat rozhraním), avšak přímo se nabízí otázka, jakou výhodu to přinese? Odpověď je prostá — žádnou.

Z předkládaných faktů je možné doporučit oddělení XML metod. Tato skutečnost získá na významu v kapitole 6.

5.4 Dopady na systém

Odstranění nevhodné dědičnosti ze systému bude znamenat:

- Jednodušší úpravu kódu tříd, kterých se týká — rozdělení definic do více souborů (z jednoho definičního souboru získám dva).

- Úpravu ostatních souborů systému, jež tyto třídy používají.

  Mělo by se jednat o minimální úpravy, protože díky automatickému zavádění modulů pomocí `Modules::loadmodules()`, je zajištěno nahrání i závislých modulů (pokud jsou správně definovány při tvorbě modulu).

Dále byly diskutovány drobné úpravy ostatních tříd:

- U třídy `Tutos_Base` bylo navrženo oddělení XML metod.

  Tam, kde si objekty odvozené z `Tutos_Base` budou muset ponechat schopnost generování XML výstupu, bude možné využít vícenásobného dědění (a pohlížet na něj jakožto na implementaci rozhraní).

  To se bude týkat minimální úpravy zdrojových souborů tříd odvozených od `Tutos_Base` (počet těchto tříd: přes 30).

- Zrušení nepodstatných metod třídy `Tutos_Module`.

  To nebude mít na systém nejmenší dopad.

A zároveň byl navržen jeden radikálnější zásah do kódu pro jednotné zpracování uživatelského vstupu (5.3.1). Ač by se jednalo o větší změnu, bylo by možné ji provádět postupně (třeba jen u nových modulů), protože se dotkne jen velmi specifických souborů, které nemají téměř žádné vazby na systém — to je možné díky povaze, jakým jsou zpracovávány HTML formuláře ve webových aplikacích obecně.

Formulář je zpravidla spojen pevně se stránkou, která je odpovědná za jeho zpracování (atribut `action` elementu `form`). Tato stránka většinou negeneruje žádný výstup určený uživateli. Ať již je kontrola vstupních údajů úspěšná či nikoli, klient je přesměrován na jinou HTML stránku s informacemi o proběhlé operaci, jež zároveň často plní i jinou funkci. Tento postup je nutný, aby se zabránilo obnovení těch stránek, které by mohlo způsobit problémy, například těch, kde je vkládán objekt do databáze.

- Zhruba platí: jedna třída = jeden formulář = jeden obslužný kód.

  To znamená změnu přibližně 30 souborů, které mohou být měněny postupně.

  Změny informačních stránek není nutné uvažovat — informování uživatele pomocí různých zpráv je již jednotně řešeno v bázových třídách uživatelského rozhraní (viz 3.1.1).

- Zároveň dojde k jednoznačnému přesunu odpovědnosti do prezentační vrstvy (a tím k vzájemnému zvýšení nezávislosti vrstev).

Obecně lze dále doporučit více využívat relaci (session). Tím dojde ke zmenšení množství předávaných informací mezi klientem a serverem (budou uchovávány přímo na serveru) a zároveň bude možné předávané informace lépe strukturovat (seznamy, asociativní pole, objekty), což je při výměně informací přes protokol HTTP nemožné.

5.5 Shrnutí

Z úprav aplikační vrstvy je nejpodstatnější oprava špatně použité dědičnosti (viz 3.2.1 a 5.2), která si vyžádá pouze malé změny v kódu.

Ostatní navrhované úpravy nejsou nijak kritické a není nutné je implementovat, popř. je možné je implementovat postupně. Jedná se především o drobné změny v bázových třídách a transparentnější ošetření chyb (oddíl 5.3).

6 Úpravy prezentační vrstvy

Tato kapitola blíže rozebírá úpravy prezentační vrstvy. Oddíl 6.1 předkládá aktuální problémy prezentační vrstvy. Část 6.2 definuje požadavky, jež by prezentační vrstva měla splňovat. Další části se podrobněji věnují navrhovaným změnám. Kapitolu opět zakončuje diskuze dopadů na systém (6.5) a shrnutí (6.6).

6.1 Problémy prezentační vrstvy

V oddíle 3.1 je rozebrána *prezentační vrstva* a je jí vytýkáno:

- *Celkový návrh struktury tříd.*

  V diskuzi k obrázku 3 je upozorňováno na nevhodné sloučení odpovědností[22] u třídy `Layout_Base` a je naznačeno možné řešení pomocí design patternu *model–view–controller*, viz například [17].

  Design pattern *model–view–controller* slouží k oddělení datových struktur aplikace (model), uživatelského rozhraní (view) a řídícího prvku (controller) do třech relativně samostatných částí. Cílem je zajistit, aby změny uživatelského rozhraní měly minimální dopad do datových struktur.

  Tento přístup lze ale zobecnit i na jiné oblasti, než pouze uživatelské rozhraní — ve skutečnosti jde o velmi flexibilní způsob, jak snížit závislost různých částí kódu, jak uvádí [13] a jak je vidět například z [29].

- *Nedostatečná podpora nových výstupních formátů, zejména pak XML.*

  V aktuálních verzích TUTOSu je výstup do XML vyřešen doplněním *XML metod* (viz 5.1) do bázové třídy perzistentních objektů `Tutos_Base` a jejich přetížením v odvozených třídách.

  Pro generování XML neexistují žádná doporučení či normy (*DTD* apod.), což jednak (1) činí implementaci metod složitou a jednak (2) je prakticky znemožněno zpracování XML dokumentu podle běžných zvyklostí[23].

  DTD = Document Type Definition

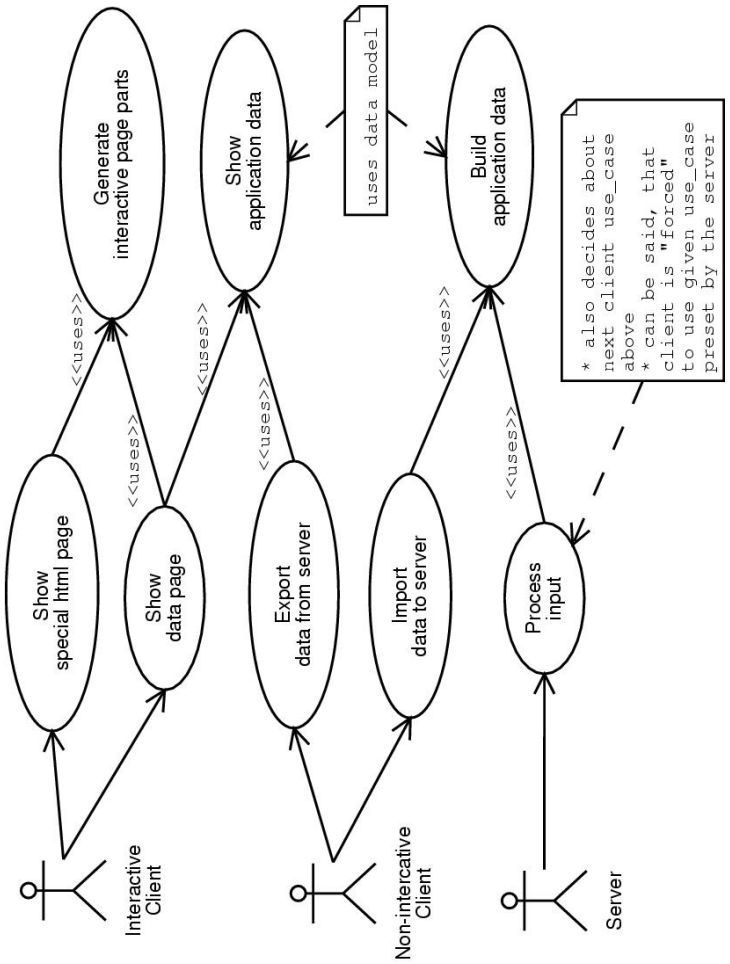
  V části 5.3.2 bylo též poukázáno na diskutabilní začlenění XML metod do bázové třídy `Tutos_Base` s ohledem na možné využití exportu i pro jiné objekty (neodvozené z `Tutos_Base`).

V některých částech TUTOSu se objevuje též export do jiných formátů, než XML. Jedná se například o proprietární formát programu Microsoft Excel, PDF či textový CSV. Většina těchto částí je již na první pohled implementována nekoncepčně a na jedno použití. To znamená:

[22] (1) *co* se zobrazuje a (2) *jak* se to zobrazuje.
[23] Tj. zpracování pomocí SAX a DOM rozhraní.

- Často je přistupováno přímo k databázi, jejíž struktura se může měnit.
- V řadě případů je generován zbytečný kód, který klientem není (a ani být nemůže) nijak využit (jak je poukazováno v části 3.1.1)
- A to vše v problematické struktuře tříd (viz 3.1).

6.2 Požadavky na prezentační vrstvu



Obrázek 11: Prezentační vrstva — model jednání

model jednání prezentační vrstvy

Na obrázku 11 je vyobrazen *model jednání prezentační vrstvy*. Ačkoli TUTOS v současné podobě umožňuje všechny možnosti (typy jednání) na něm zobrazené, žádný z nich není přímo podporován systémem (například vhodným aplikačním rámcem) a záleží na vývojáři jaký postup zvolí pro jeho implementaci, přičemž musí čelit v různé míře problémům shrnutým v části 6.1 (blíže viz oddíl 3.1).

Popis jednotlivých aktorů je shrnut v následující tabulce:

Interactive Client	Interaktivní klient představuje běžnou práci uživatele se systémem. Je očekáván HTML výstup (prezentace nebo formuláře) a vstup je zasílán serveru přes požadavky HTTP GET a HTTP POST.
Non-interactive Client	Neinteraktivní klient představuje importy/exporty dat do/ze serveru. Při importu je vstup typicky předán přes HTTP požadavek kódovaný jako `multipart/form-data` (atribut `enctype` elementu `form`). Exportní soubor může mít různou podobu, nejčastěji XML.
Server	Aktor server představuje serverovou interakci s prezentační vrstvou.

Smysl a stručný popis základních typů jednání předkládá tato tabulka:

Show special html page	V rámci TUTOSu existují určité stránky, jež slouží (především) k administrativním účelům a mají příliš specifické chování. Tyto stránky jsou dostupné pouze v interaktivním režimu (zobrazení v HTML). Většinou je nutné mít zcela pod kontrolou jejich výstup a jsou typické svým přímým přístupem k databázi. (Generování některých prvků stránky lze zobecnit — typ jednání *Generate interactive parts*.)
Show data page	Jedná se o stránky, které uvidí uživatel nejčastěji. Zobrazují jednak (1) aplikační objekty systému[24] (v různé podobě — graf, formulář) a jednak (2) seznam aplikačních objektů, typicky vše ve formátu HTML.
Export data from server	I u těchto stránek se jedná o zobrazení (1) aplikačních objektů či (2) jejich seznamu, avšak bez interaktivních prvků a v jiném formátu, typicky XML.
Import data to server	Hromadný import dat na server je představován uploadem nějakého souboru a jeho zpracováním. Ten může být realizován prostřednictvím přímého uploadu (HTTP PUT či jiný způsob) nebo prostřednictvím formuláře (HTTP POST). Tento typ jednání představuje oba způsoby — zpracování na serveru se pro ně liší minimálně[25].

[24] Tedy objekty, jež jsou většinou odvozeny ze třídy `Tutos_Base`.
[25] Tj. pouze ve způsobu, jakým se data na server dostanou.

Process input	Tento typ jednání představuje interakci serverové části s prezentační vrstvou. Úkolem je (1) zpracovat data a (2) přeposlat klientovi stránku, jež si vyžádal.

Z modelu jednání je patrné:

- Rozdělení práce klienta na interaktivní a neinteraktivní (dávkové) požadavky (aktoři `Interactive Client` a `Non-interactive Client`).
- Hlavní rozdíly a zároveň společné rysy při zpracování interaktivních a dávkových požadavků pomocí rozšířených typů jednání.
- Zachování naprosté kontroly nad speciálními případy, jejichž zpracování je velmi odlišné od běžných požadavků (typ jednání *Show special html page*) — to je vymezeno pouze pro interaktivní práci.
- Zpracování vstupu na úrovni prezentační vrstvy (typ jednání *Process Input*), jak bylo navrženo v části 5.3.1 .

Primárním požadavkem je co možná největší odstíněnost aplikačních objektů od jejich prezentace. Díky tomu bude možné měnit radikálně vzhled interaktivních stránek i formát výstupu (HTML, XML, ...), aniž by se to dotklo aplikační logiky — je evidentní, že to je právě role design patternu *model–view–controller*, viz [13], [17].

Dále je třeba zajistit:

- *Zachovat současný postup při tvorbě stránek.*

  TUTOS obsahuje množství napsaného kódu a je nutné do jisté míry respektovat současné postupy (diskuze k tomuto tématu viz 7.4.1). Znamená to:

  - Princip tvorby nové stránky se nemění — nová stránka bude znamenat odvození od nějaké společné základové třídy. Její zobrazení pak volání nějaké přetížené metody.
  - Zachování základních služeb poskytovaných *prezentační* a *aplikační vrstvou*, jak je popisováno v části 3.1.1. (Nyní však budou poskytovány odděleně.)

- *Dobrou podporu referencí na objekty.*

O provázání objektů bylo pojednáváno v předchozích částech (například část 3.2.2). Prezentační vrstva musí reference mezi objekty vhodným způsobem podporovat. Problémy s referencemi mohou nastat zejména během importu či exportu. Při importu jsou navíc tyto problémy násobené možností aktualizace stávajícího objektu systému[26].

- *Podpora oprávnění.*

  Při interaktivní práci je nutné respektovat složitý systém oprávnění, který může ponechat část objektu skrytu (typ jednání *Show data page*).

  Při neinteraktivní práci (import/export) by se mělo se systémem oprávnění zacházet jinak — import/export se musí provádět s kompletním objektem, nikoli jeho částí (tuto myšlenku lze samozřejmě rozšířit pro potřeby TUTOSu na import/export několika propojených objektů, kdy uživatel má právo přístupu jen k některým z nich[27]).

  1. Při *exportu* nejde o tak citlivou oblast — i kdyby uživatel, omezen systémem oprávnění, mohl exportovat objekt a získat tak vizuální přístup ke všem jeho částem, stále je nemůže měnit.

     V tomto specifickém případě to lze akceptovat, protože v drtivé většině případů stejně uživatel exportující objekt je jeho vlastníkem. Jiná situace není běžná.
  2. Naopak *import* by měl být omezen — aktualizovat objekt importem jiného by nemělo být možné, pokud nad ním importující nemá výhradní kontrolu (protože by mohlo dojít k přepisu částí objektu, jež nemá právo číst, natož modifikovat).

Je jistě patrné, že současný návrh a implementace prezentační vrstvy, která provází TUTOS od jeho počátků a byla původně určena jen pro formátovaný HTML výstup, již není vyhovující. Zatímco požadavky na prezentační vrstvu rostly, její design a implementace zůstával stále stejný. Proto je namístě uvažovat o novém návrhu prezentační vrstvy.

6.3 Analýza nové prezentační vrstvy

6.3.1 Výchozí situace

Inspirace Než bude podán a diskutován návrh nové prezentační vrstvy, je vhodné se zamyslet, jaké prvky by mohly pomoci při návrhu a realizaci, vycházeje ze zkušeností podobných projektů (například [28], [29], [30], [31]):

[26] Podle čeho rozpoznat objekty nově vkládané a objekty, jež mají nahradit jiné, stávající?

[27] V TUTOSu je nejmenší jednotkou nad kterou lze nastavit oprávnění právě jeden perzistentní objekt.

- *Využití reflexe při komunikaci s objekty aplikační vrstvy.* (V PHP je dobrá podpora reflexe obsažena až ve verzi 5, viz [32].)

  Zde jsou aktuální možnosti TUTOSu malé — aktuální stabilní verze se vyvíjejí pro PHP verze 4. Je snahou vývojářů, aby TUTOS 2.0 využíval plně vlastností PHP verze 5.

- *Využití informací o aplikaci a propojenosti jejích jednotlivých částí,* nejčastěji poskytovaných ve formě XML souborů (kde je snížena možnost syntaktické chyby při editaci). Může jít o informace typu: v jakých souborech jsou uloženy formuláře pro tento objekt, jaký styl použít při formátování, ...

  V této oblasti TUTOS poskytuje pouze služby tříd `Tutos_Module` a `Modules`.

- *Využití různých knihoven pro tvorbu grafických prvků* v interaktivním rozhraní (nabídka, kalendář, ...), tyto prvky často bývají označovány jako *widgets* či *controls*.

  Zde poskytuje TUTOS několik málo vlastních grafických komponent.

- U aplikací napojených na databázi je podstatná podpora vazeb mezi objekty (linků) na aplikační úrovni a řešení perzistence. Ve spojení s reflexí lze získat velmi mocné nástroje využitelné v rámci všech vrstev aplikace (např. [33] či [34]).

  V TUTOSu je perzistence i podpora vazeb řešena uspokojivě. Reflexe není využívána (jak bylo naznačeno, podpora reflexe v PHP verze 4 není dobrá).

Krátké zamyšlení Je dobré neztrácet ze zřetele celkový náhled a vidět za abstraktními pojmy jejich konkrétní naplnění. Při zobecnění části 6.2 lze konstatovat, že:

- TUTOS generuje tři formy výstupních informací (stránek):

  1. Interaktivní speciální stránky, jež nelze zaškatulkovat.
  2. Interaktivní stránky úzce svázané s objekty aplikační vrstvy.
  3. Neinteraktivní exportní stránky objektů aplikační vrstvy.

  Je na místě připomenout, že to zcela odpovídá tomu, co bylo probíráno v oddíle 2.2.1 a je patrno z obrázku 2.

  Potvrzuje se, že (1) systém je ve svých základech navržen dobře a zároveň to, že (2) předkládaný návrh nenaruší koncept celého systému (vazby mezi jednotlivými vrstvami), pokud se bude držet požadavků uvedených v 6.2, ač si může vyžádat řadu změn v prezentační vrstvě.

- TUTOS zpracovává vstup předaný pomocí protokolu HTTP. Vstup může být vcelku v libovolné podobě (soubor, několik málo proměnných).

  Zpracování vstupu probíhá jakoby „mimo aplikaci“[28], viz 5.4, což je dáno obecným charakterem webových aplikací[29]. V rámci tohoto zpracování není přímo oddělena prezentační a aplikační vrstva (viz 5.3.1).

  Mezi klientskou a serverovou prezentační vrstvou je zajištěn pouze mechanismus obecného předávání zpráv pomocí relace či přímo přes protokol HTTP, užívaných nejčastěji k informování uživatele o výsledku operace.

Prezentační vrstva by měla nabízet rozumnou podporu všem těmto činnostem. Některé jsou lépe strukturované (stránky svázané s aplikačními objekty) a v těchto případech může být podpora při tvorbě stránek ze strany prezentační vrstvy větší. U některých může být nabídnut pouze základní rámec zpracování vstupu/výstupu (speciální stránky a zpracování vstupu).

6.3.2 Základní koncept

Z rozšířených typů jednání (viz obrázek 11) jsou vidět některé společné prvky, a dále je z nich patrné, jaké činnosti musí být zajištěny.

Za návrhem i implementací stojí snaha o dodržení design patternu *model–view–controller*. Jako model figuruje třída odvozená z `Data_Model`. Role view připadá třídě `Layout` a role controller je rozdělena mezi ostatní třídy prezentační logiky `Output_Page`, `Input_Page`[30] a jádro systému `System Core`, viz obrázek 12, které spolupracují na tvorbě výstupu[31].

Následuje krátký popis jednotlivých tříd z obrázku 12:

- `Data_Model` *třída* `Data_Model`

  Tato třída slouží k prezentaci libovolných dat. Přetížením abstraktních metod by měla být zajištěna:

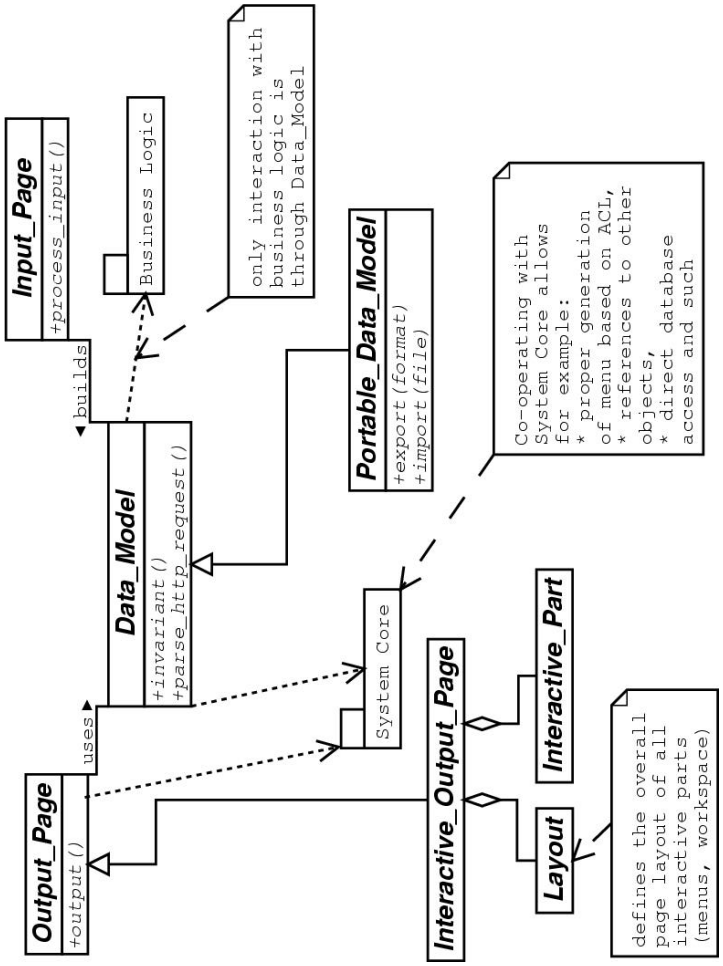
  1. Kontrola dat, metoda `invariant()`.
  2. Zpracování dat odeslaných z formuláře, metoda `Data_Model::parse_http_request()`.

[28] Tj. výkoný kód není nijak pevně svázán s GUI — propojení je pouze přes atributy `action` elemntů `form`, na rozdíl například od jasné vazby dané vyvoláním nějaké funkce.

[29] Některé uváděné zdroje, např. [29], to považují za nedostatek a snaží se zpracovávat vstup přes událostní model — snaží se tak o pevnou vazbu mezi GUI a výkonným kódem.

[30] Ano i třída `Input_Page`, protože již zde probíhá rozhodování o tom, jaká stránka bude zaslána klientovi. K předání tohoto rozhodnutí může být užito relace (session) či jiného mechanismu (po té co se zpracuje vstup).

[31] Úzké propojení těchto rolí (třeba i v rámci jedné třídy) není tak neobvyklé, viz [35]



Obrázek 12: Prezentační vrstva — zjednodušený model

Tato třída odstiňuje aplikační logiku od jakékoli znalosti prezentační vrstvy. Slouží jako prostředník, který přejímá vstup uživatele, kontroluje ho a podle potřeby nastavuje datové atributy aplikačních tříd (ať již těch odvozených od `Tutos_Base` či jiných).

Za konkrétní instancí této třídy je možné vidět libovolný HTML formulář a s ním svázaná data[32].

- `Portable_Data_Model` (odvozeno z `Data_Model`)

  Odpovědností této třídy je prezentace dat aplikační logiky. Ta mohou být navíc zobrazována v neinteraktivním režimu, což značí možnost jejich importu/exportu.

  Typicky by se mělo jednat o třídy odvozené z `Tutos_Base`, avšak není to podmínkou.

- `Output_Page`

  Tato třída je odpovědná za správné zobrazení celkové stránky. Její instance slouží jako kontejner pro objekty, jež mají být předány klientovi. Samotný výstup řídí třída `Layout`. Přetížením `Outpu_Page::output()` se ovlivní zobrazení objektu třídy `Data_Model`.

  V případě neinteraktivního výstupu je většinou pouze volána `Portable_Data_Model::export()`

- `Interactive_Output_Page` (odvozeno z `Output_Page`)

  Interaktivní stránka vedle objektů třídy `Data_Model` zobrazuje i interaktivní části stránky (objekty typu `Interactive_Part`). Celkové rozložení grafických prvků řídí třída `Layout`.

  V rámci této třídy jde o podobný model, jenž byl navržen na obrázku 4, tj. implementace design patternu model–view–controller[33].

- `Input_Page`

  Tato třída se stará o zpracování vstupu (`process_input()`).

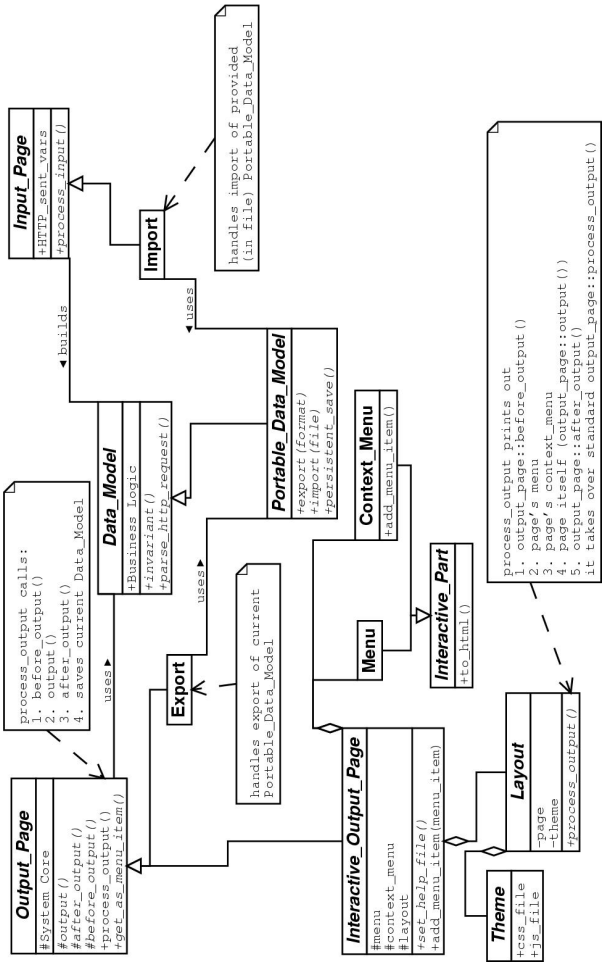
  V případě, že je možné očekávat na vstupu objekt třídy `Data_Model`, má situaci značně ulehčenu, protože tyto objekty umí zpracovat vstup samostatně (`parse_http_request()`, `import()`), v ostatních případech si musí pomoci sama.

6.4 Podrobný model

Při pohledu na *podrobný model* (obrázek 13) přibylo několik tříd a metod, ale základní koncept se nemění.

objektový model prezentační vrstvy

[32] Nicméně data nemusí být zobrazena ve formuláři — podobu zobrazení řídí až



Obrázek 13: Prezentační vrstva — objektový model

6.4.1 Celkový přehled

Tento návrh (obrázek 13) poskytuje lepší podporu pro tvorbu výstupních stránek než současná využívaná verze, zároveň však ponechává současné principy nezměněny.

V části 3.4 a 6.1 je podáno shrnutí nejpalčivějších problémů prezentační vrstvy, jež jsou v oddíle 3.1 (zejména pak v 3.1.1) probrány podrobněji. Stručně řečeno (převzato z 6.1), prezentační vrstva trpí:

- *Celkovým návrhem struktury tříd.*
- *Nedostatečnou podporou nových výstupních formátů, zejména pak XML.*

Jakým způsobem to řeší předkládaný model?

- *Celkový návrh struktury tříd.*

  Stručný popis tříd viz též 6.3.2.

  1. Zavedením *třídy* `Data_Model` bylo dosaženo striktního oddělení prezentační vrstvy od aplikační vrstvy v těch místech, kdy ji prezentační vrstva využívá[34].

     Bylo též dosaženo jednotné kontroly vstupních dat tak, jak bylo navrhováno v 5.3.1 (`Data_Model::invariant()` a `Data_Model::parse_http_request()`).
  2. Služby aplikační vrstvy (jádra systému) a služby vrstvy prezentační od sebe byly odděleny (část 3.1.1).

     Služby prezentační vrstvy jsou navíc nyní rozděleny na služby pro interaktivní zobrazení (`Interactive_Output_Page`) a služby pro neinteraktivní zobrazení (bázová `Output_Page` a třída `Export`).
  3. Byla odstraněna nevhodná dědičnost a nahrazena agregací jak je navrhováno v části 3.1. Kód je tak čistší a je možné měnit zobrazení kdykoliv, jak je běžné u jiných grafických komponentních knihoven.

- *Nedostatečná podpora nových výstupních formátů, zejména pak XML.*

  Exportovat a importovat je možné (z logiky věci) jen nějaké perzistentní objekty. Typicky půjde o potomky `Tutos_Base`, ale teoreticky

`Output_Page`.

[33] Modelem je `Data_Model`, role view připadá na třídu `Layout` (rozhoduje o konkrétním rozvržení na stránce) a controllerem je `Interactive_Output_Page`, která po svém zobrazení poskytuje (a řídí) způsob, jakým uživatel může provádět změny modelu, viz [35].

[34] Zároveň však prezentační vrstva neztrácí přístup ke službám aplikační vrstvy jak jsou definovány v 3.1.1 díky vazbám na jádro aplikace (v modelu pouze symbolicky jako `Output_Page::System Core`).

je možné exportovat třeba i nastavení modulu (perzistentní úložiště je soubor s nastavením), pokud bude vůle něco takového implementovat.

Jedná se o rozsáhlejší téma, a proto je mu věnována samostatná část níže (6.4.2).

6.4.2 Export/import a podpora ze strany prezentační vrstvy

Export potomka třídy `Portable_Data_Model` zajišťuje třída `Export`, jež vhodným způsobem přetěžuje metody báze `Output_Page`.

Primárně je export vždy proveden do XML prostřednictvím `Portable_Data_Model::export()` a přilinkováním příslušného XSLT stylu bude provedena konverze do jiného formátu[35].

Pro usnadnění komunikace mezi klientem a serverem při exportu je vždy při každém interaktivním zobrazení aktuální objekt `Portable_Data_Model` uložen na serveru (v metodě `Output_Page::process_output()`) a tak může být bez problémů použit třídou `Export` k exportu.

Podobně, jako je snahou exportu rozložit objekt `Portable_Data_Model` do XML, je cílem importu naopak převzít XML a vytvořit z něho nový objekt `Portable_Data_Model`. Jistě se při tom může uplatnit *factory method* design pattern. Importovaný objekt již zná své vazby na aplikační vrstvu a může provést uložení do příslušných perzistentních úložišť (metoda `Portable_Data_Model::persistent_save()`).

Import je přímo podporován pouze z XML formátu (třída `Import`), avšak vždy je možnost přetížit metody třídy `Import` a zajistit tak import z jiného formátu (kupříkladu CSV).

Pro bezproblémové zpracování XML je žádoucí definovat jeho strukturu prostřednictvím *DTD* (které může být vloženo přímo do XML souboru a tak lze příjemci usnadnit maximální možnou měrou jeho zpracování). *DTD*

Dobře vytvořené DTD může být použito i jinak:

1. Při tvorbě databáze v době instalace systému jako předpis pro tabulky a typy jejich sloupců (u popisu typů se lze inspirovat například specifikací XML–RPC [36]). Mohla by tak být zcela odstraněna nepotřebná třída `Query`, viz 3.3.

2. Může částečně nahradit reflexi pro datové atributy objektu.

Pro řešení problémů, jež jsou naznačeny v části 6.2 — totiž (1) podpora oprávnění ACL a (2) podpora referencí — jsou přijata tato doporučení:

[35] Teoreticky i například do HTML a tak tento způsob může být použit i pro zobrazení interaktivních dat. Na první pohled elegantní přístup však skrývá řadu úskalí a v obecnější míře by šel aplikovat jen na objekty s pevným rozhraním (např. potomky `Tutos_Base`) a nejlépe ještě s využitím reflexe.

1. *Podpora oprávnění.*

   Při exportu bude oprávnění ignorováno a vždy bude exportován kompletní objekt (včetně referencí). Nebude–li tento model vyhovovat, vždy lze pozměnit implementaci tak, aby export nebyl umožněn uživateli, jež nemá kompletní kontrolu nad exportovanými objekty[36].

   Při importu bude vždy vytvářen nový objekt (objekty) pod kontrolou uživatele, jenž provádí import. Aktualizace objektů nebude pro složitost implementace podporována.

2. *Podpora referencí.*

   DTD lze vytvořit tak, aby se jednotlivé exportované objekty mohly v rámci datového souboru XML vzájemně odkazovat (typ XML atributu `ID` a `IDREF`).

O vhodnou podporu těchto doporučení se musí postarat vývojář sám implementací v metodách `import()` a `export()`.

6.4.3 Zamýšlené použití

Základní rys předchozího postupu (jedna HTML stránka = jedna nově odvozená třída) zůstává zachován. Avšak díky lepší separaci odpovědnosti je možné uvážlivěji postupovat při exportu/importu a třídu `Data_Model` lze použít jako obecný prvek kontroly vstupu.

Tvorba interaktivní stránky

1. K vytvořené třídě aplikační logiky je nutno vytvořit třídu odvozenou z `Portable_Data_Model` (pokud je žádoucí třídu exportovat/importovat), nebo přímo z `Data_Model`.

   Toto je rozdíl oproti předchozímu postupu. Dříve nebyla hranice mezi prezentační vrstvou a aplikační takto vymezena.

   Výhodou je jednotné zpracování vstupních dat a již zmiňované oddělení vrstev zvyšující nezávislost aplikační vrstvy.

2. Pro každé zamýšlené zobrazení dat příslušného objektu odvozeného od třídy `Data_Model`[37], je nutné odvodit třídu od `Interactive_Output_Page` a přetížit její metody.

   Tento krok odpovídá původnímu postupu, kde se odvozovalo od původní třídy `Layout`.

[36] Může to být vzato v úvahu při generování nabídky například — pokud oprávnění nemá, nabídka pro export se nevygeneruje.

[37] Například (1) zobrazení a (2) editace

Výhodou oproti předchozímu kroku je snadnější přístup k jádru aplikace v rámci jmenného prostoru třídy (`Output_Page::System Core`) a tím i důslednějšímu oddělení služeb jádra a služeb prezentační vrstvy a lepší generování stránky zprostředkované novou třídou `Layout`.

Tvorba speciální HTML stránky (typ jednání *Show special html page*) se v ničem neliší. Pouze první uvedený krok je alternativní. Pokud se to nehodí, může se zcela vynechat. Přístup k aplikačním a prezentačním službám je nezměněn.

Tvorba neinteraktivní stránky U exportu by mělo pouze dojít k přetížení metody `Portable_Data_Model::export()` a vygenerování validního XML. Zbytek odpovědnosti za výstup byl přesunut zcela mimo aplikaci — do doprovodných XSLT stylů.

Při XML importu stačí přetížit `Portable_Data_Model::import()`.

Při importu z jiného formátu může být požadováno vedle přetížení metod třídy `Portable_Data_Model` i implementace stránky zpracující vstup (nejspíše odvozenou od obecnější `Input_Page`).

Lze si jistě představit i situaci, kdy vývojář bude postupně dodávat podporu pro XML import/export. Třídy `Export` a `Import` lze upravit tak, aby testovaly, zda je možné skutečně třídu importovat/exportovat, a aby byly schopné transparentně informovat uživatele v případech, kdy to nebude možné.

Tvorba stránky zpracovávající vstup Při vstupu lze výhodně použít metody třídy `Data_Model` pro zpracování vstupu. Jako objekt `Data_Model` může vystupovat díky obecnému navržení libovolný formulář a k němu navázaná data, která nemusí mít přímou reprezentaci v aplikační vrstvě (což lze využít zejména pro implementaci typu jednání *Show special html page*).

Z modelu je sice patrné, že by měla být instance stránky `Input_Page`, ale protože v běžných činnostech půjde nejčastěji o zpracovávání objektů odvozených od `Data_Model`, byla by její implementace pouhým dodržováním postupu.

Proto nejspíše není nutné ji vždy implementovat a uchýlení k běžnému „skriptování“ lze tolerovat.

6.5 Dopady na systém

Zatímco v předchozích kapitolách, byly navrhované změny spíše minimalistické, v této kapitole byla navržena kompletní změna celé prezentační vrstvy. Naštěstí jde o nejvyšší vrstvu, takže se tyto změny dotknou pouze využívání vrstev nižších.

- *Dopady na databázovou vrstvu.*

  Žádné.

- *Dopady na aplikační vrstvu.*

  Třídy aplikační vrstvy od tohoto okamžiku nejsou odpovědné za žádnou prezentaci svých dat. To kompletně přebírají odvozeniny `Data_Model` a `Output_Page`. V důsledku to znamená odstranění *XML metod* (viz část 5.1).

  Dále aplikační třídy ztrácí nutnost kontrolovat vstupní data (to se týká jen některých tříd, kde tato kontrola není doposud prováděna v prezentační vrstvě).

  XML metody bude možné prostě přesunout a změnit do nově vytvořených tříd odvozených z `Data_Model`. Tato úprava kódu v první fázi (přesun) bude snadná. Druhá fáze (úprava) bude vyžadovat vytvořit dobré a (bohužel) složité *DTD*, až potom bude možné třídy upravit.

  Kontrola vstupních dat může být převzata (jistě s malými úpravami) do metod nově vzniklých tříd `Data_Model`. Sem by zároveň měla být přesunuta i kontrola dat, jež probíhá nyní v současných třídách/souborech prezentační logiky.

- *Dopady na prezentační vrstvu.*

  Značné. Kompletní re–implementace.

  Jedná se o kompletní změnu struktury tříd, o přesuny odpovědností a o zavedení přísnějšího rámce pro tvorbu nových stránek. Z vnějšího pohledu se sice vše jeví jako dříve, ale vnitřní struktura prezentační vrstvy je jiná.

  Prezentační třídy TUTOSu v současné době tvoří hrubým odhadem asi tak $\frac{2}{5}$ kódu (každá zobrazitelná HTML stránka + stránky zpracovávající vstup).

Rámec pro prezentační vrstvu navrhovaný v této kapitole je obecný a při jeho implementaci by bylo vhodné využít i jiné knihovny — jde zejména o knihovny, které podporují generování různých ML (Markup Language) — například `Smarty` [37], které by značně zjednodušily jak generování, tak i výslednou práci s ML[38].

Lze jen doporučit implementovat tyto změny až v PHP verze 5, tj. v TUTOSu 2.0, jež poskytuje mnohem lepší podporu objektového programování (reflexe, výjimky).

[38] Například podporou šablon, parsrovacích mechanismů aj.

6.6 Shrnutí

Tato kapitola podává zcela nový návrh prezentační vrstvy při respektování řady požadavků a podmínek (6.2).

Nutnost nového návrhu vychází ze současné neuspokojivé situace (viz 6.1).

Předkládaný návrh odstraňuje vytýkané nedostatky (část 6.4.1), ale při jeho implementaci se naráží zejména na (1) relativně větší vnitřní složitost a (2) obsáhlost změn, které přináší.

7 Závěr

Práce se zabývala návrhem a implementací systému TUTOS [2]. Cíl práce byl stanoven takto (převzato z 1.2):

- Identifikace problémových částí TUTOSu, resp. jeho návrhu.
- Konkrétní doporučení, jak danou část zlepšit.
- Dokumentace, na které lze dále stavět.

První dva body je vhodné diskutovat najednou (7.1), poslednímu je věnován oddíl 7.2.

7.1 Problémové části a doporučení

7.1.1 Shrnutí postupu analýzy

V kapitolách 2 a 3 byl postupně představován systém TUTOS. Systém byl popisován s ohledem na jednotlivé vrstvy (prezentační, aplikační a databázovou). Popis jednotlivých vrstev byl postupně zpodrobňován.

Každá vrstva byla představena v interakci se svým okolím a zároveň podrobněji rozebrána v samostatných částech práce (kapitoly 2 a 3).

Doporučující úpravy byly probrány pro každou vrstvu samostatně od jednodušších úprav ke složitějším. Jsou jim věnovány tyto kapitoly: 4 (databázová vrstva), 5 (aplikační vrstva) a 6 (prezentační vrstva). Každou z těchto kapitol uzavírá diskuze dopadu změn na systém a krátké shrnutí.

7.1.2 Dosažené výsledky

Z celkového hlediska je systém navržen dobře (2.2). V rámci jednotlivých vrstev se však objevují některé nedostatky.

Databázová vrstva V databázové vrstvě (viz 3.3) nejsou žádné zásadní problémy. Je doporučeno využít jiných knihoven pro databázovou abstrakci, protože v současné době poskytují (1) podporu pro více databází, (2) trochu lepší odstínění od databáze a (3) podporu nových vlastností databázových strojů.

TUTOS byl nucen využívat vlastní knihovnu, protože v době jeho vzniku nebyly tak dobré databázové abstrakce ještě k dispozici. Dnes již ale je situace jiná. Využitím takové knihovny zároveň odpadne nutnost udržovat databázový kód.

Aplikační vrstva Aplikační vrstva (3.2) nebyla popsána v celé své šíři. Byl vybrán úsek klíčových modulů systému (3.2.2) a v něm byly odhaleny drobnější nedostatky v implementaci, viz 5.3, a jeden v návrhu při smíchání odpovědností u tříd při špatně užité dědičnosti, viz 3.2.1.

Bylo navrženo, jakým způsobem situaci napravit (odstranit dědičnost) a zároveň byly diskutovány další drobné změny v aplikační vrstvě s cílem zpřehlednit implementaci a jasně vymezit odpovědnosti u tříd (5.3).

Především šlo o to odstranit z aplikační vrstvy nevhodné odpovědnosti, jež by bylo, s ohledem na úpravy prezentační vrstvy, vhodné řešit jinde a jinak — týkalo se to (1) kontroly vstupních dat, 5.3.1, a (2) odstranění XML metod z bázové třídy `Tutos_Base`, 5.3.2.

Prezentační vrstva Největších navrhovaných změn (a i největší diskuze) se dočkala prezentační vrstva.

Nutnost nové implementace prezentační vrstvy byla odůvodněna v částech 6.1 a 6.2. Bylo poukázáno na nevhodnost současného návrhu s ohledem na výstup do jiných výstupních formátů (XML).

Nový návrh plně podporuje nové požadavky (6.2) a zároveň ponechá tvorbu výstupních stránek téměř beze změn (6.4.3).

Vnitřní struktura tříd nově navržené prezentační vrstvy je složitější (byť ucelenější) a tudíž implementačně náročnější. Výměnou prezentační vrstvy se bude muset upravit mnoho kódu.

7.2 Dokumentace

TUTOS je projektem na SourceForge.net, [3], a je k němu dostupný zdrojový kód; avšak kromě toho téměř žádná dokumentace. O modelech popisující fungování systému z trochu větší perspektivy si může nechat jen zdát.

Na tomto poli tudíž není s čím srovnávat. Tato práce je prvním veřejným autorovi známým zdrojem, jež se snaží popsat fungování systému pomocí názorných modelů a doprovodných komentářů.

Nevýhodou jistě je to, že práce není psána anglicky. Tento nedostatek a z něho povstávající oprávněná výtka byl způsoben nedostatkem času. Alespoň diagramy byly vypracovány důsledně v anglickém jazyce a názvy tříd v nich uvedené odpovídají těm ze skutečného kódu v TUTOSu.

Bylo představeno celkem 13 modelů. Zachytit systém v celé jeho šíři na jednom modelu je nesmyslné. Některé modely tudíž volí hrubší měřítko a popisují základní stavební prvky systému. Některé si naopak vybírají jednotlivé výseky a ty podrobně zpracovávají.

Za významné mohou být považovány tyto modely:

Pojmový model (obrázek 1)	celkový náhled
Vrstvy systému (obrázek 2)	jednotlivé vrstvy systému
Asociace (obrázek 6)	podrobnější pohled na aplikační logiku
Prezentační vrstva — model jednání (obrázek 11)	požadavky na prezentační vrstvu

Z hlediska přínosů práce pak i model na obrázku 13 — podrobný objektový model nově navržené prezentační vrstvy.

Zkušenému vývojáři možná chybí datový model — rozumný pohled na data systému však může poskytnout vhodný nástroj pro práci s databází, kterých je celá řada, například [38].

S dokumentací souvisí i dokumentování zdrojového kódu — bylo by žádoucí doporučit (jednotně) nějaký systém pro dokumentování zdrojového kódu — například `phpDoc`, [40].

7.3 Diskuse nad přínosy práce

Při zpětném pohledu na cíl, viz 7, lze konstatovat, že cíl a smysl práce byl v rozumných mezích splněn.

V době vniku práce (konec roku 2004) se jedná o ojedinělý počin (s ohledem na komunitu uživatelů a vývojářů kolem TUTOSu), který se snaží zmapovat rozsáhlý systém využitelný v široké škále činností. Motivací autora bylo přispět hodnotným materiálem této komunitě.

Dokument předkládá řadu konkrétních doporučení a nevyhýbá se ani špatným stránkám systému, jež se snaží na základě provedené analýzy napravit.

Metodika OMT, [5], jež byla v práci využita, rozlišuje několik úrovní abstrakce rozboru systému: (1) rozbor zadání, (2) analýzu, (3) design a (4) implementaci.

Tento dokument osciluje mezi všemi těmito úrovněmi a pouze nepřesně vymezuje jednotlivé úrovně abstrakce během prováděných analytických postupů. To mu může být vytýkáno.

Je však třeba si uvědomit:

- Byl popisován rozsáhlý systém, k němuž neexistuje jiná dokumentace, než zdrojový kód. To mohlo vést někdy k přílišnému tíhnutí k podrobným a implementačně zaměřeným částem (jako například 4.2).
- Protože jde o existující systém, některé části rozboru systému postrádají smysl (rozbor zadání) a postup zpracování by to měl náležitě respektovat a volit jen ty vhodné nástroje k popisu (k čemuž metodika OMT přímo vybízí).

- Velikost systému. Nebylo též záměrem popsat celý systém. To by vzhledem ke komplexitě a rozsahu celého systému bylo spíše námětem disertační práce. Cílem bylo odhalit jednotlivé (nejpalčivější) nedostatky a pokusit se o jejich nápravu, ne popisovat ty části, kde buď (1) je vše v pořádku, anebo (2) nejsou příliš významné (rozšíření bez většího významu).

7.4 Další postup

Je možné předpokládat, že změny navrhované v databázové a aplikační vrstvě nevyvolají příliš připomínek, protože se jedná o vcelku dobře uchopitelné případy a doporučení jsou jasná a konkrétní.

Rozhodně to však nelze říci o prezentační vrstvě, která byla navržena zcela nově při respektování současných omezení systému. Právě té by měl být dán největší prostor při dalším (navazujícím) rozboru. Některé z možných připomínek se pokusí zodpovědět právě tento oddíl.

V současné době se objevuje řada velmi zajímavých projektů, řešící programování webových aplikací lehce jiným přístupem — přes událostní model, který plně umožňuje využít design pattern *model–view–controller*, [28], [29]

Současný model zpracování stránek ve webových aplikacích (a zejména v PHP) je příliš ovlivněn modelem zpracování, kdy jednomu formuláři odpovídá jeden skript obsluhující formulář.

Je to patrné i mém návrhu (`Output_Page` — formulář, `Input_Page` — obsluha formuláře). Sice se snažím o určité zobecnění (`Data_Model`), ale není to zobecnění dokonalé. Stále je zde nutnost odvozovat řadu stránek z `Output_Page` a z `Input_Page`[39], i když sama třída `Data_Model` zastane práci jak při tvorbě formuláře (poskytuje data pro zobrazení), tak i při jeho zpracování (kontrola vstupních dat).

Provázání stránek (formulář a obsluha formuláře) je příliš volné a příliš je spjato s implementací — propojení se děje skrze jméno souboru, které je uvedeno v atributu `action` elementu `form`. Přejmenování souboru najednou znamená nefungující aplikaci ve všech částech, kde je jméno souboru uvedeno jako stránka zpracovávající formulář[40].

7.4.1 `php.MVC`

Problémy naznačené v předchozím odstavci se snaží řešit například knihovna `php.MVC`. Inspiraci jejího návrhu lze hledat u jiných jazyků a jejich knihoven, například `mod_python`, viz [39].

[39] Přičemž v části 6.5 sám uvádím, že tvorba stránek `Input_Page` je mnohdy zbytečná.

[40] Pokud je kupříkladu generován odkaz na tento soubor v nabídce musí se změnit i ta. Samozřejmě je možné mít nějaké centrální repository souborů, kde by mohlo docházet k přemapování, ale to pouze posouvá problém jinam

Cílem použití této knihovny je odstínit GUI webové aplikace (webové stránky) od propojení na konkrétní zdrojové soubory zpracovávající tyto stránky.

Dosahuje se toho tím, že formuláře (webové stránky, view) se nepropojují s konkrétním zdrojovým souborem, ale s akcí (model), která má být vykonána (typicky volání nějaké funkce). Všechny formuláře jsou následně odeslány určité komponentě (controller), která na základě dostupných informací vyvolá příslušející obslužnou akci.

Začlenění takové knihovny do aplikace však bude mít dalekosáhlé dopady na její návrh a implementaci jak ve vrstvě prezentační, tak i aplikační. Dojde ke kompletní změně modelu tvorby nových stránek a bude třeba přijmout zcela nový přístup k implementaci aplikace. (Předkládaný návrh se snaží zachovat současné postupy.)

Při pokusu konkretizovat přínosy začlenění podobné knihovny lze dojít k následujícímu seznamu (při srovnání s navrhovaným řešením v kapitole 6):

1. Odpadne zcela tvorba stránek odvozených z `Input_Page`.
2. Mapování objektů třídy `Output_Page` na obslužné stránky by nebylo závislé na jménu souboru, ale bylo by spjato přímo s instancemi `Data_Model` a jejich metodami.
3. Přístup k řadě užitečných funkcí (systémy šablon, zpracování XML, integrace s databázovými knihovnami) — například `php.MVC` je schopna spolupráce s knihovnami `ADOdb`, [19] či `Smarty`, [37], na něž lze nalézt odvolávky i v této práci.

7.4.2 Shrnutí

Databázová a aplikační vrstva jsou v zásadě v pořádku a ať již navrhované změny budou zapracovány či ne, TUTOS stále bude funkčním a použitelným software. Je to prezentační vrstva, kde musí dojít k zásadnímu rozhodnutí, jak dál.

Proti předloženému návrhu nové prezentační vrstvy stojí nové slibné projekty, které staví vývoj aplikací v PHP na zcela jinou úroveň. Zvážit všechny důsledky začlenění tak rozsáhlé knihovny, jež je popisována v 7.4.1, do kódu TUTOS by jistě stálo za úvahu.

Reference

[1] http://www.gnu.org/copyleft/gpl.html 3, 7

[2] http://www.tutos.org/ 3, 7, 60

[3] http://sourceforge.net/projects/tutos/ 3, 7, 9, 61

[4] elektornická konference tutos-devel@lists.sourceforge.net 3

[5] Pavel Drbal: Objektově orientované metodiky a technologie — 1. a 2. díl, Vysoká škola ekonomická v Praze, Praha 1997, ISBN 80–247–0029–8 10, 11, 16, 62

[6] Jindřich Zelený, Josef Nožička: COM+, CORBA, EJB, BEN — technická literatura, Praha 2002, ISBN 80–7300–057–8 8

[7] Glyn Moody: Rebel Code, Perseus Publishing, Cambridge 2001, ISBN 0–7382–0670–9 8

[8] Marc J. Rochkind: Advanced UNIX programming, 2nd edition, Addison–Wesley, 2004, ISBN 0–13–141154–3

[9] Pavel Satrapa: Perl pro zelenáče, Neokortex, Praha 2000, ISBN 80–86330–02–8 8

[10] http://www.carthag.org/ 9

[11] http://www.phplens.com/ 9

[12] http://www.tutos.org/homepage/links.html 14

[13] Brett Spell, Professional Java Programming, Wrox Press, Inc., 2000, ISBN 1–861003–82–X 16, 17, 19, 20, 27, 44, 47

[14] Pavel Císař, Interbase/Firebird Tvorba, administrace a programování databází, Computer Press, Brno 2003, ISBN 80–7226–946–1 33

[15] http://www.gtk.org/ 20

[16] http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html 19

[17] http://en.wikipedia.org/wiki/Model-view-controller 19, 44, 47

[18] Bjarne Stroustrup, The C++ Programming Language, Addison–Wesley, 2000, ISBN 0-201-70073-5 23, 40

[19] http://adodb.sourceforge.net/ 29, 31, 64

[20] http://pear.php.net/package/DB 29, 31

[21] http://en.wikipedia.org/wiki/Factory_method_pattern 32

[22] http://en.wikipedia.org/wiki/Iterator_pattern 32

[23] http://java.sun.com/products/jdbc/download.html 31, 32

[24] http://twistedmatrix.com/ 40

[25] http://www.zope.org/ 40

[26] Joseph Schmuller: Myslíme v jazyku UML,
Grada Publishing, spol. s r. o., 2001, ISBN 80–247–0029–8

[27] Jim Arlow, Ila Neustad: UML a unifikovaný proces vývoje aplikací,
Computer Press, 2003, ISBN 80–7079–740–1

[28] http://rwfphp.multispan.com/ 48, 63

[29] http://phpmvc.net/ 44, 48, 50, 63

[30] http://jakarta.apache.org/velocity/index.html 48

[31] http://jakarta.apache.org/turbine/index.html 48

[32] http://www.php.net/manual/en/language.oop5.reflection.php 49

[33] http://www.hibernate.org/ 49

[34] http://skunkweb.sourceforge.net/PyDO/ 49

[35] http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html
50, 54

[36] http://www.xmlrpc.com/spec 55

[37] http://smarty.php.net/ 58, 64

[38] http://squirrel-sql.sourceforge.net/ 62

[39] http://www.modpython.org/ 63

[40] http://sourceforge.net/projects/phpdoc/ 62

Rejstřík

ACL, 14, 34, 37
aplikační vrstva, 14, 19, 33

databázová vrstva, 15, 25, 29
DTD, 53, 56

factory method, 30, 53

invarianta, 38
iterator, 30

Java TUTOS, 6

model jednání prezentační vrstvy, 43
model–view–controller, 17, 42, 45, 48, 61

objektový model asociaci, 22
objektový model prezentační vrstvy, 50

pojmový model, 11
prezentační vrstva, 12, 16, 42
projekt, 11

služby aplikační vrstvy, 18, 45
služby prezentační vrstvy, 18, 45

transakční manažer, 9, 14
TUTOS, 5, 6
třída `Data_Model`, 48, 52
třída `Layout_Base`, 17
třída `Tutos_Base`, 14, 21, 33
třída `Tutos_DB`, 15
třída `Tutos_Module`, 14, 21, 34

vrstvy systému, 12