České vysoké učení technické v Praze
Fakulta elektrotechnická

Diplomová práce

Návrh webového systému řízení malé společnosti

Vladimír Macek

Vedoucí práce: RNDr. Štěpán Kvapilík

Studijní program: Elektrotechnika a informatika, kombinovaný magisterský

Obor: Výpočetní technika

květen 2007

Poděkování

Chtěl bych vyjádřit svůj vděk vedoucímu práce RNDr. Štěpánu Kvapilíkovi za to, že mi i ve svém nabitém programu vychází vstříc. Dále děkuji mé partnerce MUDr. Janě Škopové a mým rodičům za dlouholetou podporu. Poslední dík patří mému příteli a kolegovi Marku Bradáčovi, bez kterého bych měl studium těžší.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze veřejně přístupné podklady, z nichž na některé odkazuji v části Použité zdroje.

Nemám žádný závažný důvod proti použití tohoto díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Liberci dne 24. května 2007 _____

Abstrakt

Práce představuje analýzu konkrétních nároků uživatelů počítačové sítě malé společnosti na provoz systému pro nakládání s provozními informacemi. Dále je zpracovávána problematika uživatelských rozhraní a použitelnosti, což je aplikováno na návrh nového systému. Jedním z výsledků je první implementovaný evidenční modul, který však používá univerzální jádro pro nakládání s daty typu tabulkový záznam umožňující funkci sledování a vracení změn. Jeho uživatelské rozhraní je vícejazyčné.

Summary

This document analyses the needs of local area network users in the small company for system to manage operative business data. Then the area of user interface design and usability is elaborated and results applied on the new system design. As one of the results comes the first implemented module that is using universal core for managing the table record data type and also allowing users to trace and revert data changes. The user interface is multilingual.

Obsah

Seznam obrázků

1 Úvod, motivace

Cíl práce. Stručný úvod do problematiky informačních systémů. Podstatné procesy ve firmě zadavatele.

1.1 Cíl práce

Cílem této práce je připravit půdu pro sjednocení několika různých informačních systémů provozovaných u zadavatele. Hmatatelným výsledkem této fáze projektu bude první z modulů evidujících konkrétní druh provozních údajů. Povede k němu cesta přes analýzu potřeb uživatelů a jejich zkušeností s dosud paralelně provozovanými programy s podobným účelem, ale zejména přes návrh univerzálního jádra informačního systému pro správu datových záznamů, jehož podstatnou složkou má být sledování podrobné historie změn a možnosti jejich vracení.

Důležitým rysem nového systému musí být snadnost a intuitivnost uživatelského rozhraní. Implementace má spojit výhody vysokoúrovňového (objektového) nahlížení na datové položky i z pohledu programátora s rigidně nízkoúrovňovým pohledem na data nabízeným relační databází. Použití tohoto typu datového úložiště je požadavkem zadavatele. Kombinace má vyústit ve výhody:

1. rychlé implementace budoucích modulů a udržování těch stávajících v moderním prostředí a
2. zajištění cenných dat tradičním způsobem s využitím spolehlivých kontrol jejich integrity, k čemuž se relační databáze za desítky let své existence dopracovaly.

1.2 Informační systém a jeho přínos

Efektivnost činnosti každé dnešní společnosti závisí na dobré organizaci práce, organizaci a dostupnosti informací i odlehčení od opakujících se činností, které je možné automatizovat. Za *informaci* lze považovat výsledek zpracování a organizace vědomostí či jednoduše údajů a má znamenat přínos pro toho, kdo ji obdrží.[1] Mluvíme-li o výdělečném podniku, pak jak organizace práce, tak moudré nakládání s informacemi má pozitivní vliv na zisk.

Praktickým prostředkem k dosahování právě popsaného se ukázaly být *informační systémy* (IS), což je jen moderní název pro starobylý výtvor člověka. IS lze považovat za soubory komponent účelově definované ke sběru, udržování, zpracování a poskytování informací. Jako každý *systém*, je i IS vymezen svou hranicí oddělující jej od okolí. Toto okolí jej ovlivňuje a formuje.

Za IS tedy můžeme považovat cokoli vyhovující definici, tedy i ručně udržovanou kartotéku papírových složek. Ruku v ruce s realizací prvních *automatických počítačů* však přišlo na svět i jejich použití jako podnikových IS. Tento text se zabývá výhradně určitým druhem automatizovaných IS – interaktivními počítačovými programy komunikujícími s lidským uživatelem.

Pro popis, analýzu i implementaci je vhodné IS rozdělit do funkčně vzájemně závislých *vrstev*. Ty však mohou být navrženy a implementovány nezávisle na sobě a jejich role lze popsat například takto:

- **Uživatelské rozhraní.** Hranice IS, interakce s uživatelem a prezentace dat.
- **Logika aplikace.** Zajišťuje zejména zpracování, ale i zabezpečení dat.
- **Datové úložiště.** Znamená zapouzdření paměťového média.

Každá z rolí má své definované rozhraní. Vhodné rozhraní umožňuje mimo jiné

- průběžné zdokonalování vrstev nezávisle na zbytku systému,
- odpojení vrstvy z popsaného kontextu a její nahrazení nějakou ekvivalentní,
- paralelní práci více instancí vrstev s jednou instancí vrstvy na nižší úrovni (například zálohovací software pracuje přímo s datovým úložištěm nezávisle na provozu IS z hlediska uživatele).

1 Je možné, že Sir Francis Bacon svým „Scientia potentia est“ („Vědění je síla“) parafrázoval jedno z již starozákonních Přísloví krále Šalamouna „Moudrý člověk je silný, člověk informovaný však stále mocnější.“

Základem podnikového IS je soubor rozumně propojených evidencí nejrůznějších položek. Hlavním přínosem IS pro podnik má být efektivní prezentace uložených informací, ještě lépe jejich vzájemná kombinace. Předpokládá se také filtrace nedůležitých a nedostupných údajů podle momentálních potřeba uživatele a té které uživatelské role. Nutností je vhodný a pohodlný způsob zadávání a úprav údajů.

1.3 Zadavatel práce

1.3.1 Zavedené postupy

Zadavatel této externí diplomové práce, společnost Hieronymus s.r.o., je malá firma specializující se na technické počeštění (tzv. *lokalizaci*) softwaru, návodů ap. Fungování zajišťují následující druhy pracovníků:

- *externí překladatelé* (obvykle smluvně vázané fyzické osoby),
- *interní překladatelé* (zaměstnanci),
- *korektoři* (zkušení pracovníci),
- *vedoucí projektů* (zkušení zaměstnanci),
- *majitel firmy* a *administrativa.*

Projekt je v zavedené terminologii jednotlivá zakázka zadávaná vnějšími společnostmi (zákazníky) požadujícími provést lokalizaci svého produktu. Má tyto vlastnosti:

- Objem (počet slov k překladu) je různý, pohybuje se od desítek slov do stovek tisíc slov.
- Obvyklá podoba jsou elektronicky přenášené počítačové soubory ve formátu standardních překladatelských nástrojů (např. Trados, IBM Translation Manager, Microsoft Helium) či běžných textových dokumentů (Microsoft Office).
- Ve většině případů je možné projekt rozdělit na *úkoly* za účelem souběžné práce několika překladatelů.

- Je obvykle udržován na hlavním síťovém sdíleném diskovém svazku přístupném uživatelům jako logická jednotka ve Windows z lokální sítě i vzdáleně přes zabezpečenou virtuální privátní síť VPN.
- Překladatelský software umožňuje buď pracovat na projektu přímo na tomto sdíleném disku nebo pracovník hotové dílčí práce synchronizuje ze svého počítače.
- Všechny důležité servery provozuje operační systém Linux[2], sdílené disky poskytuje software Samba[3].

Vedoucí projektů mají zejména následující úkoly:

- Komunikace se zákazníkem, převzetí zakázky (obojí obvykle e-mailem),
- vyjednání překladatelské síly,
- na základě objemu projektu a počtu dostupných pracovníků rozdělení projektu na práce a jejich předání překladatelům,
- průběžná komunikace s překladateli, například řešení odborných dotazů,
- přebírání hotových dílčích prací od překladatelů a podle potřeby jejich předání ke korektuře,
- postupná kompletace projektu, na konci jeho předání zákazníkovi.

Majitel společnosti a administrativní pracovníci mají na starosti mimo jiné tyto oblasti:

- Dohoda a realizace odměn pracovníkům,
- dohoda cen pro zákazníky a režie plateb,
- smluvní vztahy mezi firmou a pracovníky a mezi firmou a zákazníky.

Pracovníci jsou odměňováni těmito způsoby:

- Pevná zaměstnanecká mzda plus odměny,
- mzda za odvedenou práci měřenou počtem slov převedených na normo strany,
- dohodnutá hodinová mzda.

2 http://www.linux.org
3 http://www.samba.org

1.3.2 Správa dokumentů

S léty docházelo ve firmě k rozšiřování skupiny nejrůznějších statických dokumentů pracovního rázu, které by měly být organizovány a zpřístupněny. Ukázalo se, že chyběl jednotný *systém pro správu dokumentů* (CMS, Content Management System).

Za účelem správy dokumentů byl vybrán a zprovozněn portál Plone[4] pracující na platformě Zope[5]. Zope i Plone jsou vyvíjeny v programovacím jazyce Python[6]. Ve všech třech případech se jedná o svobodný software s otevřeným zdrojovým kódem a licencemi GNU GPL[7] nebo kompatibilními s GPL. Portál Plone nabízí potřebné možnosti pro správu dokumentů. Podrobnější popis obou systémů se nachází dále v kapitole 4 na straně 17.

4 http://www.plone.org
5 http://www.zope.org
6 http://www.python.org
7 http://www.gnu.org/licenses/gpl.html

2 Výchozí stav a zadání

Současný stav počítačového zpracování informací ve firmě zadavatele. Specifikace obecnějších cílů a konkrétního cíle této práce. Vysvětlení volby implementační platformy.

2.1 Stávající řešení

Výše popsaná firma provozuje více oddělených evidenčních aplikací pro:

- vedení projektů a prací,
- tvorbu výsledkových sestav,
- vedení obchodních kontaktů,
- evidenci výkonů odvedených pracovníky,
- provozní záležitosti jako řízení záloh sdíleného diskového prostoru, evidence paměťových médií a literatury, správu sítě a další.

Popsané aplikace jsou implementovány jako *webové*, viz kapitola 5.1 na straně 31. Tím ovšem kompatibilita končí. Negativa provozovaného řešení jsou tato:

- Starší část agend byla implementována ve skriptovacím jazyku dnes už ne příliš populární komerční databáze `mSQL`. I když vývoj a uvedení do provozu probíhaly

s dobrou systematičností, výsledný software není moderními programovacími postupy dobře rozšiřitelný,

- zbývající webové aplikace tvořili během doby různí zaměstnanci a někteří silně nemetodicky. Platformou byl jazyk PHP, který se ale ukazuje jako nevhodný (více viz kapitola 4.3.7 na straně 27),
- uživatelské rozhraní obou aplikací se zcela liší,
- autentizace heslem není jednotná.

2.2 Obecný cíl

Vedení firmy došlo k názoru, že různorodost používaných evidenčních nástrojů je na škodu produktivitě zaměstnanců. Bylo rozhodnuto, že je třeba sjednotit všechny používané uživatelské aplikace do jedné. Vzešlý software má splňovat tyto zásady:

- Metodický návrh celé struktury,
- webové, jednotné a efektivní uživatelské rozhraní,
- otevřené programátorské prostředí vysokoúrovňového programovacího jazyka,
- zpracovávání otevřených datových formátů bez vazby na komerčního producenta,
- zabudování do čerstvě implementovaného systému správy dokumentů Plone,
- vícejazyčnost uživatelského rozhraní (možnost používání cizinci),

Výběr trojice Python – Zope – Plone pro implementaci správy dokumentů CMS byl od začátku veden zřetelem k výše uvedeným zásadám pro snadné zakomponování informačního systému. Proto je implementační prostředí IS dáno tímto rozhodnutím. Spojení systémů Plone a IS znamená i sjednocení oprávnění a rolí uživatelů, stejně tak fulltextové vyhledávání bude jednotné a prohledá jak dokumenty v systému CMS, tak datové položky v kartotékách. Název nového IS zadavatele je *HIS* (Hieronymus Information System).

2.3 Zadání práce na systému HIS

Pro tuto diplomovou práci bylo zadání konkretizováno následovně:

- Analyzovat a popsat požadavky zaměstnanců zadavatele vzniklé na základě zkušeností s dosud provozovanými na zakázku vyvinutými informačními systémy.
- Podle nových poznatků provést návrh skupiny propojených webových modulů (např. evidence výkonů zaměstnanců, vedení projektů, evidence osob a obchodních partnerů či nástroje správy sítě) s obsluhou různých uživatelských rolí.
- Navrhnout jednotné uživatelské rozhraní s jednoduchým používáním a možností návrhu nových ovládacích prvků či postupů zvyšujících efektivitu.
- Implementovat pilotní verzi vybrané části navrženého systému.
- Pro implementaci vyjít z již fungujícího systému správy dokumentů (CMS) Plone používajícího aplikační platformu Zope. Důležitá data musejí být ukládána do RDBMS.

3 Analýza potřeb uživatelů

Rozpracované zadání a podrobnější popis nároků na funkčnost nového systému.

Pro analýzu nové aplikace vyjdeme z informačních systémů, které se z historických důvodů ve firmě nezávisle na sobě provozují. I když tyto IS rozhodně nejsou z dnešního pohledu optimální, lidé s nimi pracují a jsou na ně zvyklí. A je dobré mít na paměti, že i jejich návrh vycházel z tehdejších praktických potřeb, z nichž většina se nezměnila.

Z toho důvodu bude jednak potřeba postupovat při přechodu na nový IS opatrně a za druhé zohlednit fakt, že uživatelé neočekávají, že nový systém bude *zcela* nový. Někteří z nich zjistí změnu IS až v den výměny...

3.1 Struktura systému HIS

Následuje popis nejdůležitějších plánovaných modulů systému HIS. Při popisu jednotlivých evidenčních záznamů vynecháváme popis takřka všudypřítomné položky *Poznámka*. Ta bude sloužit k zadání dodatečné textové informace, která bude samozřejmě vyhledatelná pomocí funkce fulltextové hledání.

3.1.1 Evidence obchodních partnerů, kontaktů a uživatelů

Tato část má zahrnovat dvě tabulky:

- **Evidence firem.** Důležité údaje o firmách, se kterými je zadavatel v obchodním styku (ať už jako dodavatel nebo jako odběratel). K dispozici má být seznam projektů vázaných na tuto firmu (a taky obráceně).
- **Evidence kontaktů a uživatelů.** Tato personální kartotéka sdruží následující druhy osob, u kterých umožní vyznačit jejich vztah k zadavateli:
  - Konkrétní osoby pracující ve firmách, se kterými má zadavatel obchodní vztah, pochopitelně s odkazem do evidence firem (a zpět),
  - externí pracovníky zadavatele,
  - interní zaměstnance,
  - potenciální kolegy (adepty na spolupráci),
  - jakékoli jiné osoby.

  Uživatelé HIS. Evidence kontaktů umožní u osob nastavit příznak „Uživatel systému HIS“, přístupové heslo a role. Vyplněním těchto položek se osoba stává uživatelem systému HIS s možností plnit svou zadanou roli.

3.1.2 Evidence projektů

Toto je nejdůležitější kartotéka firmy. Eviduje projekty pro jednotlivé zákazníky. Nejdůležitější položky projektu jsou:

- **Název** a **kód** projektu. Krátký jedinečný kód se generuje automaticky.
- **Kategorie.** Výběr z přednastavených klíčových slov; určí druh použitého překladatelského softwaru, ale zároveň to, zda se jedná o projekt interní a jaký. Příklad množiny: *Trados*, *Helium*, *Správa firmy*, *Správa poč. sítě*, *Vývoj intranetu.*
- **Stav.** V současnosti tyto hodnoty: *Naplánovaný*, *otevřený*, *dokončený*, *zavřený*, *odložený*, *zrušený.* Je důležité mít možnost v budoucnu množinu stavů měnit za rozumné složitosti úpravy příslušných pracovních postupů (workflow).

- **Firma** vybraná z evidence firem, obvykle v roli zákazníka. Při zakládání projektu bude v zájmu rychlosti též možné ze stejného formuláře založit novou firmu a zároveň s ní nový projekt sdružit.
- **Kontaktní osoba** u přidružené firmy.
- **Vedoucí projektu.** Zaměstnanec pověřený řízením projektu.
- **Jazyky**, ve kterých je překladatelský projekt veden. Podle této položky má být možné provádět výběry projektů. Může zahrnovat i údaj o směru překladu.
- **Termíny dokončení**, **odeslání**, **fakturace**.
- **Cesta k adresáři** na sdíleném diskovém svazku, který obsahuje soubory nutné pro dokončení projektu.

V uživatelském rozhraní se požadují dva druhy formuláře *Nový* projekt, standardní a zjednodušený. To proto, aby zakládání velmi malých projektů o objemu třeba 30 slov netrvalo déle než jejich vyhotovení.

3.1.3 Evidence úkolů

Stávající IS obsahuje vedle evidence projektů též evidenci tzv. *prací*. Ty se vztahují k překladatelským projektům, v přehledu projektu se zobrazují jako tabulka, jsou přidělovány pracovníkům, mají stavy apod. V systému HIS bude pojem práce více zobecněn a má mít následující vlastnosti:

- Pojem je změněn na *úkol*. Důvody jsou:
  - lépe to odpovídá rozšířenému významu,
  - může být dlouhodobější,
  - vyřešení úkolu nemusí být viditelné jako odevzdaná práce,
  - snaha o změnu pohledu uživatelů.
- Úkol udržuje svůj *stav*. Množina podobná množině stavů projektu.
- Úkol se může a nemusí vztahovat k nějakému projektu.
- Projekt nemůže změnit stav na takový, který je v kolizi se stavem kteréhokoli úkolu k tomuto projektu přidělenému.

- Úkol je vždy určitého typu. Například *Překlad*, *Korektura*, *Vedení projektů*, *Účetnictví*, *Administrativa* a pod. Je nutná možnost jednoduché změny této množiny v budoucnu.
- Vedoucí projektů přidělují práci na projektech jakékoli kategorie libovolné osobě vedené v evidenci kontaktů (nejen uživatelům HIS).
- Každý úkol je přidělen nejvýše jednomu pracovníku, obvykle ten dále upravuje jeho obsah.
- Každý uživatel systému bude moci tuto evidenci využívat jako plánovač či „úkolník“ pro svou potřebu.
- Automatizované sledování termínů dokončení a vazba na náhled kalendáře (ten je součástí portálu Plone).
- Volitelné zřetelné rozlišení úkolů podle stavů, termínů či jiných kritérií.
- Volitelné upozornění e-mailem na přidělení úkolu a na blížící se termín dokončení.

3.1.4 Evidence výkonů a dostupnosti

V předchozím IS se tato část nazývala *docházka* a byla vázána na projekt. V systému HIS má být výkon vázán na úkol, tudy tedy povede vazba k projektu. Pracovníci zde vyplňují následující údaje:

- **Doba od – do**, kdy se věnovali pracovní činnosti. Záznam může být i neuzavřený s tím, že údaj *do* se doplní později. Systém musí zajistit, že každý uživatel má v každém okamžiku nejvýše jeden neuzavřený záznam a že se žádné časové intervaly nepřekrývají.
- **Úkol**, který se tímto výkonem řešil.

Přínos celé této evidence ve spojitosti s evidencí úkolů pro vedení firmy a vedoucí projektů lze popsat takto:

- Přehled a hodnocení práce podřízených,
- pomoc s vyjednáváním pracovní síly při rozdělování úkolů projektu,
- automatizovaná tvorba sestav udávajících úhrnné úsilí vynaložené na dokončení jednotlivých projektů,
- přehledy nedostupnosti pracovníků, například pro plánování dovolených a zastupování.

Systém HIS má mít možnost vyhodnotit, které dny v měsíci jsou svátky a tedy které dny nemají být považovány za pracovní.

3.1.5 Kontext evidenčních modulů

Vzájemný vztah výše popsaných evidencí lze ilustrovat triviálním lineárním schématem zasazeným do kontextu pracovník – zákazník:

3.2 Uživatelské role

Uživatelské role slouží zejména k určení toho, kterou oblast příslušnému uživateli zpřístupnit. Ve druhé řadě se později použijí k optimalizaci prezentace dat (např. zobrazení postranních přehledových tabulek jen relevantním uživatelům).

Pro práci se systémem HIS definujeme uživatelské role, které v případě potřeby určují oprávnění až na úroveň jednotlivých záznamů v tabulce. V takovém případě je atributem nejčastěji to, že uživatel položku sám vytvořil. Níže v dílčích odrážkách se popisují oprávnění k jednotlivým evidencím (modulům systému HIS), které byly popsány dříve. Popis začíná uživatelem s nejnižším oprávněním a každá další role popisuje pouze oprávnění přibývající.

- **Pracovník.** Informuje se na stav projektů, aktualizuje přidělené úkoly a eviduje své výkony:
  - *kontakty a firmy* zobrazení všechny,
  - *projekty* zobrazení všechny,
  - *úkoly* zobrazení a změny jen své a nové,
  - *výkony* zobrazení a změny jen své a nové.
- **Vedoucí projektu.** Komunikuje se zákazníky a pečuje o projekty:
  - *kontakty a firmy* zobrazení a změny všechny,
  - *projekty* zobrazení a změny všechny,
  - *úkoly* zobrazení a změny všechny,
  - *výkony* zobrazení a změny jen své a nové.
- **Vedení firmy.** Provádí mj. účetnictví na základě výkonů, potřebuje tedy navíc:
  - *výkony* zobrazení a změny všechny.
- **Správce.** Zajišťuje bezproblémový chod celého systému.
  - Bez omezení a má přístup i ke specializovaným funkcím správy.

3.3 Vícejazyčnost aplikace

Celý portál Plone je tvořen jako vícejazyčná aplikace s podporou přepínání jazyka uživatelského rozhraní za běhu a podporou udržování jednoho dokumentu ve více jazycích. Pak je možné, aby se systémem mohli pracovat také cizinci. Jejich webový prohlížeč zahrnuje požadavek na jazykovou verzi dokumentu v požadavku a tak není třeba ani nic ručně volit.

Požaduje se, aby vlastnost přepínání jazyka rozhraní za běhu měl také systém HIS.

4 Popis implementační platformy

Stručný úvod do objektového programování, použitého programovacího jazyka Python, systémů Zope a Plone.

4.1 Objektově orientované programování

V zásadě se jedná o proces rozlámání libovolně složitého problému na menší části, které se lépe spravují. Razí princip, že tyto kusy by měly pracovat tak nezávisle, jak jen to jde a komunikovat spolu pomocí daných *rozhraní* (*Application Programming Interface*, API).

Na počátku stojí myšlenka spojit data a kód, který s nimi pracuje, dohromady. Tento celek tvoří tzv. *objekt*. Data objektu (jeho vlastní proměnné) se nazývají *atributy* (v Zope pak také *vlastnosti – properties*, viz dále) a s nimi prováděné operace *metody*. Přistupujeme-li k objektu pouze pomocí jeho API, je pro nás tento objekt černou skříňkou a nezajímá nás ani to, zda na něm programátor stále pracuje a mění jeho implementaci. Program se tím stává vysoce modulární.

Třída je abstraktní popis objektu. Definuje, co by mohl dělat, kdyby „ožil“. Takové oživení třídy objektu se nazývá *instance*. Třídu si lze představit jako podrobný plán ke stavbě domu a instanci jako dům již stojící. Podle jediného plánu lze postavit nekonečně mnoho domů, které budou mít všechny zcela stejné atributy, ale na sobě budou nezávislé a budou různě využívané.

Často definujeme *potomky* již existujících tříd, využíváme tedy mechanismu *dědičnosti*. Máme-li např. další plán na stavbu přístěnku s poznámkou, že veškerá dokumentace k domu se nachází v původním plánu, vytvořili jsme odvozenou třídu. Tu však lze použít jako kompletní návod na stavbu domu s přístěnkem, tedy k vytvoření plnohodnotné instance.

V praxi se tato metodika při programování využívá tak, že nadřazená třída (superclass) zobecňuje a sdružuje co nejvíce funkčnosti tříd, které z ní budou odvozeny. Ty pak konkretizují své vlastnosti.

Skutečnost, že většina funkcionality zůstává definována a vylepšována pouze na jednom místě, poskytuje obdivuhodné možnosti opětovného využívání již hotové práce. Díky dělení problému do funkčních a pojmenovaných celků se i rozsáhlý projekt lépe udržuje a rozšiřuje.

4.2 Programovací jazyk Python

Jedná se o obecný, interpretovaný a dynamický jazyk s velice jednoduchou a intuitivní syntaxí. Je zde určitá syntaktická zvláštnost: Vnořené bloky a definice se vymezují větším odsazením od levého okraje a nikoli speciálním značením, jak je obvyklé z tradičních jazyků (v C `{ }`, v Pascalu `begin` a `end`). To vyžaduje jistý zvyk a přijetí, ale šetří psaní a vynucuje přehlednější zdrojový kód.

V jazyce Python je vše objekt a k dispozici jsou veškeré výhody objektového programování. Klasické procedurální programování je zde však stále možné. Významnou vlastností je jednoduché používání vysokoúrovňových datových typů (n-tice, pole, asociativní pole, komplexní čísla, ... až po možnost definice vlastních typů).

Programátor není obtěžován množstvím zbytečných podrobností, může se soustředit na jádro problému. Proto jazyk Python nabízí jednu z nejkratších cest od modelu k funkčnímu programu a je ideální pro první výuku programování. Dokumentační řetězce ke všemu, co uživatel definuje (třídy, metody, funkce...), jsou přímou součástí kódu. Je možné k nim přímo přistupovat z programu nebo interpreteru.

4.3 Systém Zope

Systém *Zope* (*Z Object Publishing Environment*) je integrovaný aplikační framework a webový server zaměřený na publikaci dynamických dat. Základem je soustava předdefinovaných i uživatelských objektů, které se mohou vzájemně *obsahovat* (a tvořit tím

hierarchii) a využívají své schopnosti podobnou adresářové. Aplikace v Zope se tvoří objektovým způsobem s vydatnou pomocí objektového jazyka Python, neboť i Zope samotné je v tomto jazyce implementováno.

4.3.1 Vlastnosti a výhody Zope

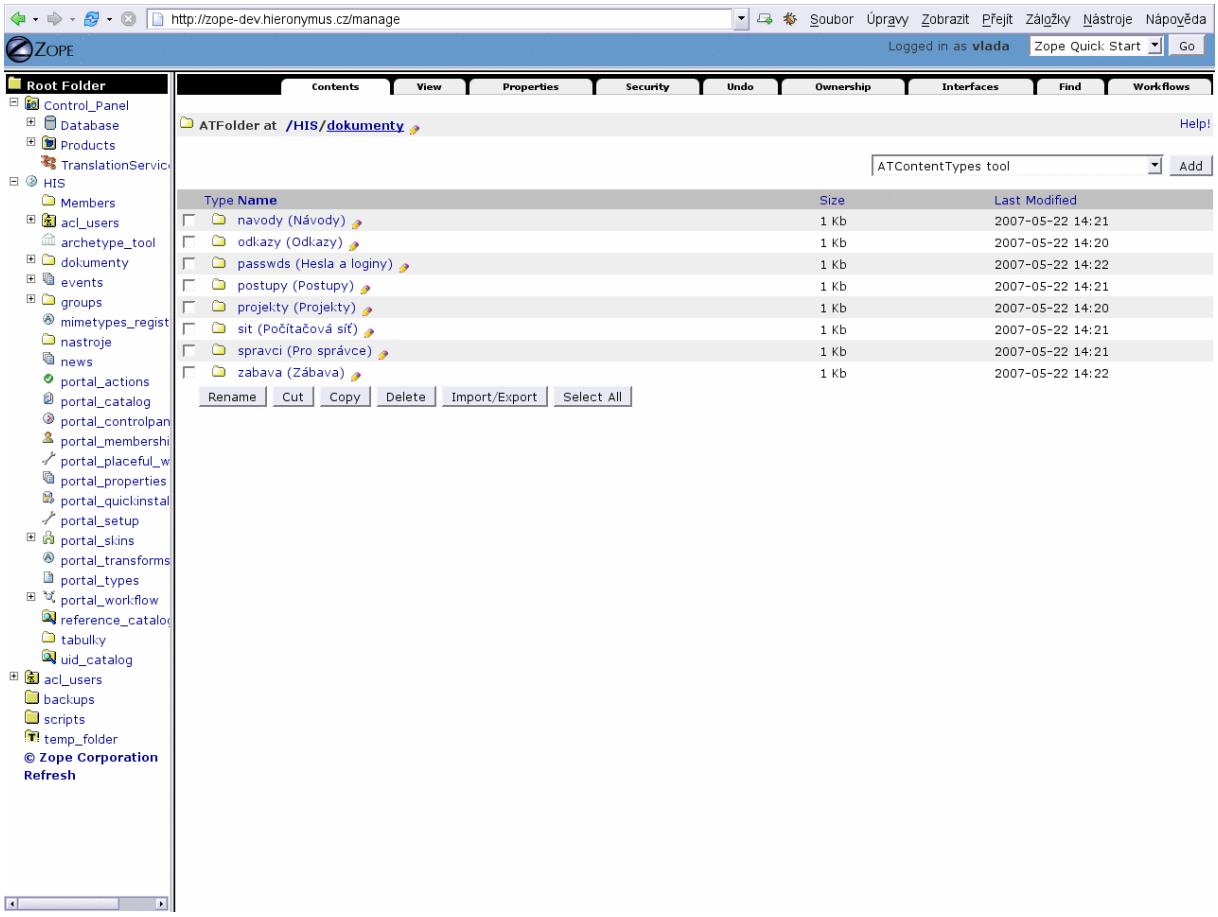
Systém je zcela otevřený, obklopený velkou a ochotnou komunitou. Nejdříve se jednalo o komerční produkt a jeho producent je se svým rozhodnutím o otevření stále spokojen a jeho zaměstnanci s vnější komunitou spolupracují. Naprosto stejné prostředí lze spustit jak ve Windows, tak i v Unixu. Pojem portování aplikace je zde tedy irelevantní.

Lze přímo připojit k množství vnějších zdrojů, programů či nástrojů: WYSIWYG editory, předřazený webový server, SQL databáze, LDAP, SMTP, IMAP, POP, NNTP a cokoli se doprogramuje, přičemž jazyk Python již má obrovskou softwarovou základnu.

Zope odstiňuje programátora od detailů přenosu dat pro a od uživatele, např. od protokolu http (přenos webových stránek). Aplikaci je kdykoli k dispozici asociativní pole `REQUEST` s daty webových formulářů, soubory cookies, daty HTTP požadavku, atd… Bez prezentační vrstvy orientované čistě na jazyk HTML se vlastně programuje obecná objektová aplikace využívající *persistenci* (viz dále). V metodě není nutné napsat ani řádek navíc, chceme-li přijmou data z formuláře a publikovat výsledky. Zope publikuje „ven“ obecné objekty (nikoli pouze stránky).

Téměř úplně lze spravovat a vyvíjet přes webové rozhraní ZMI (*Zope Management Interface*), příklad na obr. 4.1. Nabízí efektivní práci se záložkami, což jsou vlastně různé pohledy na aktuální objekt. V rozhraní ZMI lze funkčnost většiny i těch nejmenších stavebních kamenů aplikace ihned otestovat. Zope se automaticky dožaduje vstupních hodnot a prezentuje výsledky.

V ZMI je k dispozici vyčerpávající kontextová nápověda a programátorská reference včetně fulltextového vyhledávání. Každý produkt jednoduchým postupem registruje svou nápovědu do hlavního stromu. Často k dosažení cíle není potřeba žádné programování, úkoly jsou prováděny standardním předdefinovaným způsobem.



obr. 4.1 Webové rozhraní ZMI (systém Zope)

Jazyk Python poskytuje vynikající nástroj pro ladění – *traceback*. Dojde-li kdykoli při provádění aplikace k nečekané události a vygenerovaná výjimka není na své cestě ven z vnořených metod odchycena, dojde k zapsání podrobných informací o stavu aplikace do protokolu chyb Zope. Tyto informace zahrnují čas, uživatele, adresu URL, typ a popis výjimky, kompletní data požadavku HTTP, zejména však výpis vnořených metod a funkcí včetně čísel řádků od ZPublisheru (viz dále) až k místu výskytu problému.

Správci stačí kontrolovat protokol a má k dispozici ucelené ladicí informace o problému, na který narazili uživatelé aplikace.

4.3.2 Nevýhody Zope

Provozovatel rozvíjeného řešení na bázi Zope musí mít trvale k dispozici alespoň jednoho člověka, který celý komplexní a originální systém ovládá a připravuje tvůrčí prostředí ostatním. Těm stačí kratší nebo vůbec žádné školení (podle jejich zaměření).

V porovnání s jinými řešeními generování dynamického webu Zope v absolutní rychlosti vyřizování požadavků prohrává. Je to dáno robusností operace publikování objektů namísto prostého provedení skriptu a substituce dynamického textu do stránek. Pro informaci: Téměř celé samotné Zope je soustava zdrojových textů jazyka Python (včetně jeho internetového serveru!).

4.3.3 Součásti Zope

4.3.3.1 ZServer

Vícevláknový modul komunikující s vnějším světem například protokoly HTTP (publikace výsledků objektů), XML-RPC, FTP (přímé obousměrné zpřístupnění objektové hierarchie), WebDAV (pohodlná WYSIWYG editace objektů na dálku).

4.3.3.2 ZPublisher

Object Request Broker, který požadavky mapuje na průchod hierarchií trvale uložených objektů (URL traversal) a vrací výsledek transparentní interpretace/vyjádření objektu (*rendering*).

Příklad takové publikace objektu. Máme-li adresu URL:

```
http://myhost/object1/object2/method?count:int=1&name=foo
```

Je mapováno takto: Metoda `method` dílčího objektu `object2` dílčího objektu `object1` dostupného z kořenové složky bude volána s argumenty číselným `count=1` a textovým `name="foo"`.

Metoda `method` může být šablona HTML stránky nebo obecného XML dokumentu, Z SQL metoda, Python Script, Externí metoda, cokoli, co lze „renderovat“. Vše je zcela transparentní a většina objektů je tím vlastně jednoduše a automaticky publikována.

4.3.3.3 ZODB

Zope Object Data Base je přímou součástí Zope, i když je jakožto obecná objektová databáze k dispozici i nezávisle na Zope. Jedná se o úložiště všech objektů, kde se objekty aktivují a také čekají ve vyrovnávacích pamětech.

Objektová databáze nabízí trvalé (*persistentní*) instance objektů – programátor se nestará o žádné ukládání na disk. V Pythonu prostě napíše:

```
self.promenna_objektu = ['Petr', 'Pavel']
```

a tato proměnná („vlastnost“ v ZMI) natrvalo obsahuje tento seznam (řetězec, číslo, asociativní pole, odkaz nebo celou instanci třídy…).

ZODB dokáže automaticky předcházet konfliktům více klientů. Může být transparentně distribuovaná mezi více servery (systém ZEO), což ústí ve škálovatelnost, lepší dostupnost, rozložení zátěže nebo automatické replikace.

Transakce (skupiny změn dat) jsou typu `ACID` (Atomic – buď celá skupina nebo nic z ní, Consistency – stav celé databáze je v libovolném okamžiku konzistentní, Isolation – transakce se navzájem neovlivňují, Durable – jakmile se transakce ukončí příkazem `commit`, jí definované změny přetrvají i vypnutí systému) a zcela transparentní, bez jakékoli další péče. Změny provedené od příchodu požadavku až po odeslání výsledku prohlížeči jsou buď přijaty jako celek (`commit`) nebo jako celek odvolány (`rollback`).

Pokud relační databáze na pozadí podporuje transakce, Zope je zařadí do svého transakčního schématu. Opět žádné programování navíc.

ZODB udržuje historii změn všech objektů. Správce tedy může vyvolat operaci *Undo* k vrácení provedených změn, a to i nesouvisle. Databáze je uložena v jediném souboru `data.fs`, čas od času je vhodné ze ZMI provést „packing“ pro eliminaci nepotřebných starších verzí objektů.

V následujících částech této kapitoly probereme jen ty aspekty systému Zope, které jsou relevantní pro tento projekt.

4.3.4 Některé principy fungování Zope

4.3.4.1 Získávání (Acquisition)

Je to jeden ze základních principů fungování systému. Jde o dědění zdrojů na základě umístění objektu nebo místa volání v hierarchii databáze. Objekty Zope jsou obsaženy v jiných objektech (např. objektech typu Folder), to vytváří hierarchii. Objekty „získávají" vlastnosti (data) a chování (metody) z objektů typu Složka, které je obsahují. Je to podobné principu dědičnosti u tříd, ale místo vztahu předek–z něj definovaný následník je zde vztah obsahující–obsažený.

Důsledky. Kdekoli v hierarchii, ve webové šabloně, ve skriptu, atp. zavolám cokoli bez adresování. Toto něco je prostý identifikátor, může to být šablona, skript, atribut, vlastnost… Hierarchie bude prohledána od místa volání směrem ke kořeni a nalezenému objektu bude jako kontext předáno místo volání. On se tedy bude chovat tak, jakoby se nacházel ve volající složce. Výsledek volaného (vlastnost je prostě vrácena, metoda provedena, šablona renderována) je vrácen volajícímu. Volající může být i adresa URL ukazující někam hlouběji do struktury (elegantní sběr chybových hlášek a implicitních stránek – vše je definováno na jednom místě a chová se dynamicky).

Toto poskytuje obrovské možnosti opětovného využití veškeré práce, ať už se jedná o šablony, skripty… Nabízí to jednoduchou změnu chování celého podstromu (např. struktury webových stránek) pouhým „doklikáním" několika společných proměnných (vlastností složky). Překrytí nadřazeného pro daný podstrom.

4.3.4.2 Zabezpečení

Třídy uživatelů systému Zope jsou označeny tzv. *rolemi*. Rolím se přiřazují jednotlivá podrobná *oprávnění*. Oprávnění jsou textové řetězce a vztahují se k jednotlivým funkcím systému. Přiřazení oprávnění k funkcím softwaru je práce pro jeho vývojáře. Na druhé straně přiřazení oprávnění k rolím už může provádět správce v rozhraní ZMI.

4.3.4.3 Přístup k relačním databázím

Zope má podporu všech důležitých systémů RDBMS (včetně např. ODBC). Objekt spojení (database connection) v hierarchii poskytuje sdružování požadavků a jednoduchou mezipaměť (cache).

4.3.5 Prezentační vrstva

Page Template (ZPT). Jsou to šablony ve formátu XML, jde-li tedy o formát XHTML, je výsledkem webová stránka. Princip šablon je ten, že po vyvolání se interpretují a na označená místa dosadí hodnoty, které si samy zjistí voláním dostupných metod, ke kterým mají oprávnění. Značkovací jazyk má název TAL (*Template Attribute Language*). Prostřední slovo názvu vyjadřuje skutečnost, že jazyk má své místo výhradně v *atributech* elementů XML. Atributy mají zvláštní jmenný prostor (namespace) `tal`.

Obvyklé šablonové jazyky podporují „jednosměrný vývoj“: pouze od designéra stránek (grafika) k programátorovi. TAL však umožňuje trvalou spolupráci designéra a programátora na jediném zdrojovém textu. Designér je po krátkém zaškolení schopen pracovat a odstíněn od logiky aplikace. Má vždy k dispozici statickou webovou stránku, kterou může upravovat. WYSIWYG editory webů si neznámých atributů nevšímají, ty žádné zaškolení nepotřebují. Proto TAL používá atributy a ne elementy, které by mohly být náhodně eliminovány.

TAL pracuje na principu překrývání existujícího obsahu šablony obsahem dynamickým. Pro ilustraci jednoduchosti následuje seznam všech „příkazů“ (atributů) TAL: `tal:attributes`, `tal:define`, `tal:condition`, `tal:content`, `tal:omit-tag`, `tal:on-error`, `tal:repeat`, `tal:replace`.

Příklad: Iterace přes výsledky nějaké metody `files` ve složce, ve které se nachází i *tato* šablona (`container`) a zobrazení příslušných položek asociativního pole `item` (iterační proměnná).

```
…
<tr tal:repeat="item container/files">
  <td><a href="Sample.tgz" class="filename"
   tal:attributes="href item/getId"
   tal:content="item/getId">Sample.tgz</a></td>
  <td tal:content="item/getContentType">application/x-gzip-compressed</td>
  <td tal:content="item/getSize">22 K</td>
  <td tal:content="item/bobobase_modification_time">2006/09/17</td>
</tr>
…
```

Kdyby na prvním řádku bylo uvedeno `here/files`, byla by metoda `files` *získána* naopak *z místa volání*. Text nacházející se uvnitř elementů `<td>` je slepý a bude příkazem `content` dynamicky nahrazen. Tato technika pomáhá vizuálnímu návrhu stránek.

Jazyk *Standard TAL Expression Syntax* (TALES) popisuje syntaxi *výrazů*, které dodávají příkazům TAL a METAL data, ale není na ně vázána. Výraz se skládá z volitelné předpony ukončené dvojtečkou – určuje typ. Seznam klasifikátorů: `path`, `exists`, `nocall`, `not`, `string`,

python. Seznam vestavěných identifikátorů: nothing, default, options, repeat, attrs, CONTEXTS, root, here, container, template, request, user, modules.

Příklady:

```
request/URL
template/title
here/title
container/title
user/getUserName
python:1+2
string:Hello, ${user/getUserName}
request/cookies/verbose | nothing
```

Standard Macro Expansion Template Attribute Language (METAL) je nástroj pro předzpracování maker jazyka TAL. Makra jsou způsob, jak definovat a spojit dohromady kus nějaké stránky/šablony (např. kalendář, pár vstupních polí pro přihlášení, hlavičku stránky, postranice). Šablona pak svá makra sdílí s ostatními objekty, takže změna makra na jednom místě má okamžitý efekt.

Příklad – odkazovaná šablona s názvem master_page:

```
<p metal:define-macro="copyright">
  Copyright &copy; 2006, <em>Méďa Béďa</em> s.r.o.
</p>
```

Odkazující se šablona (složka templates může být *získána* z nadřazené struktury):

```
<hr>
<b metal:use-macro="here/templates/master_page/macros/copyright">
  Zde bude copyright
</b>
```

Odkaz na makro může být dynamický pomocí výrazu TALES. Makra lze dále dělit do podobným způsobem definovaných *slotů*. Jejich naplněním lze ve volající šabloně upravit obsah makra.

Příklad: V hlavní šabloně je definován hlavní navigační panel v jednom slotu a ponecháno místo na dodatečné, nevyplněné, informace:

```
<div metal:define-macro="sidebar">
  <div metal:define-slot="links">
    Links
    <ul>
<li><a href="/">Domů</a></li>
<li><a href="/products">Produkty</a></li>
<li><a href="/support">Podpora</a></li>
<li><a href="/contact">Kontakt</a></li>
    </ul>
  </div>
  <span metal:define-slot="additional_info"></span>
</div>
```

Odkazující se šablona si přeje standardní navigaci a dodá další informace. Chce-li, může předefinovat i slot s odkazy samotnými:

```
<p metal:use-macro="container/master_page/macros/sidebar">
  <b metal:fill-slot="additional_info">
    Nenechte si ujít naše <a href="/specials">speciality</a>.
  </b>
</p>
```

Sloty mohou být i do sebe vnořeny. V praxi se podřízená šablona odkáže na makro hlavní šablony webu rovnou v elementu `<html>` a pak už jen vyplňuje sloty. Celý web pak může vycházet z jediné šablony s jednotnou strukturou. Tak funguje nadstavba Plone a na tomto principu bude založen i systém HIS.

Seznam všech příkazů METAL: `metal:define-macro`, `metal:use-macro`, `metal:define-slot`, `metal:fill-slot`.

4.3.6 Produkty

Jde o nové objekty nebo jejich struktury, které se do Zope instalují „přikopírováním“ jako standardní „pythonský“ modul do speciálního adresáře na souborovém systému. Tímto způsobem se distribuují aplikace pro Zope, jako jsou diskuzní servery, databázové adaptéry a stovky dalších.

Po startu se systém Zope pokouší nainstalovat (integrovat do sebe) všechny moduly, které najde v adresáři `Products`. Třídy zde definované mohou být se Zope logicky propojené pevněji než např. třídy v externích metodách. Lze s nimi volněji pracovat.

4.3.7 Srovnání s jazykem PHP

Na tomto místě v několika bodech uvádíme důvody, proč byla pro projekt zvolena jiná implementační platforma než všudypřítomný jazyk PHP a také, proč byla vybrána ta, která byla vybrána.

- Jazyk PHP neodděluje prezentaci od logiky, naopak vše integruje. To je hlavní brzda tvorby větších projektů v něm. Programátor se jednoduše časem ztratí ve vlastním programu.
- PHP programátora nenavádí k modularitě a objektovosti. Nabádá spíš ke tvorbě stylem 'vlep a sleduj, co to udělá'. K nějaké koncepci se tvůrce musí donutit sám. Málokterý se dostane tak daleko.
- Zdrojový text PHP je velice nepřehledný. Každý programátor se totiž do jednoho zdrojového souboru snaží vložit jak logiku, tak prezentaci (vč. vlivu celkové koncepce webu) a design! Jen málo programátorů se může pochlubit dobrými schopnostmi ve všech třech oborech.
- Mění se a vyvíjí rychleji, než je na programovací jazyk zdrávo. Mnoho programátorů, kteří k němu přešli např. od Perlu, jej zase opustilo, protože se zde mezi majoritními verzemi najednou změnilo rozhraní a oni by svoje hotové programy museli přepisovat.
- Python je (podobně jako Perl) jako jazyk nesrovnatelně stabilnější a stejně jako Zope je dobře navržen. I po letech se mění pouze málo a nedochází ke zpětné nekompatibilitě.
- Jako doklad tohoto tvrzení: Zope je tvořeno několika mebibajty textových definic tříd. Když jeho tvůrce Jim Fulton chtěl přidat transparentní vyvažování zátěže a překlenutí selhání (failover), stálo ho to pouhé dva týdny práce a výsledek měl objem jen 100 KiB! To prostým přidáním k systému s dobrým návrhem, který předtím danou schopnost neměl. Bylo při tom využito mnoho kódu, který v Zope již byl.

4.4 Portál Plone

Portál Plone je produktem pro systém Zope (viz kapitola 4.3), tedy jeho aplikační nadstavbou, využívá jeho služeb. Jak bylo zmíněno v úvodu, účelem tohoto programu je jednotná správa dokumentů více uživateli. Následuje pouze stručný popis.

Jako většina aplikačních produktů pro Zope se i základní Plone skládá z těchto komponent:

- Webové šablony ZPT.
- Lokalizační soubory systému `gettext` (přeloženo do více než 35 jazyků).
- Aplikační a validační kód v jazyce Python. Validace kontroluje data zadaná uživatelem do webových formulářů.
- Self-testů korektnosti kódu.

Některé vlastnosti systému Plone:

- Striktní dodržování otevřených standardů.
- Dokumenty lze přímo z webového prohlížeče vytvářet a upravovat editorem typu WYSIWYG, který na pozadí vytváří platný kód HTML.
- Automatizovaná práce s obrázky vkládanými do dokumentů.
- Podpora firemních pracovních postupů (workflow). Má-li mít dokument určitý daný „oběh“, lze to definovat a plnit.
- Veškeré texty (včetně dokumentů Microsoft Office a PDF) jsou ihned po vložení indexované a je možné v nich vyhledávat.
- Vzhled lze upravovat nikoli úpravou šablon, ale přidáním do zásobníku témat, tzv. `skinů`.
- Každá nová vydaná verze obsahuje připravené migrační procedury, díky kterým mohou uživatelé upgradovat své instance bez ztráty dat.

Protože systém HIS vychází z portálu Plone, bude portál Plone graficky představen dále v popisu rozhraní celého systému HIS.

4.4.1 Správa dokumentů

Ve zbytku této části se stručně popisují funkce, které systém Plone nabízí pro správu dokumentů. Souhrnným názvem položka se zde nazývají všechny datové typy udržující a nějakým způsobem prezentující informace.

4.4.1.1 Základní typy položek

- **Dokumenty.** Obsahují text, který může být formátován jako tzv. Strukturovaný text nebo HTML nebo nemusejí mít formátování vůbec (tzv. „prostý text“).
- **Události.** Jsou vázány na časové období a mohou být zobrazovány v rámečku Kalendář.
- **Soubory.** Obsahují libovolné datové soubory ke stažení.
- **Složky.** Obsahují další položky včetně složek. Mohou mít definovaný vlastní způsob zobrazení (položka `index_html`), pokud tomu tak ale není, zobrazují se jako prostý seznam položek, které obsahují.
- **Obrázky.** Mohou být vkládány do dokumentů portálu nebo přímo odkazovány odjinud.
- **Odkazy.** Jsou okomentované adresy URL.
- **Novinky.** Obsahují krátký článek s nadpisem a nepovinným popisem. Mohou být zobrazovány v rámečku Novinky.
- **Rešerše.** Jsou skupiny přednastavených podmínek. Jejich zobrazení jsou množiny položek (např. dokumentů), které podmínkám v dané chvíli vyhovují. Jedná se o způsob organizace obsahu alternativní ke složkám.

4.4.1.2 Stavy položek

- **Nepřístupné.** Nikdo jiný než vlastník položky či její správce si položku nezobrazí ani nevyhledá.
- **Přístupné.** Položka se nevyskytuje v Navigaci uživatelů, kteří nejsou správci. Ti si ji tedy mohou zpřístupnit pouze následujícími způsoby: znají její přímou adresu (URL) nebo mají zobrazenu složku, která ji obsahuje, nebo vyhledají ji vyhledávačem.
- **Ke zveřejnění.** Jako Přístupné; položka se navíc zobrazuje správcům ve zvláštním levém rámečku *Žádosti o zveřejnění*, aby je upozornila, že ji mohou zveřejnit.
- **Zveřejněno.** Jako Přístupné; položka může být navíc nalezena pomocí Navigace (pokud jsou zveřejněné i všechny jí nadřízené složky). Dále se ve chvíli zveřejnění všem přidá do rámečku *Poslední položky.* Není-li její vlastní správce, už nemůže měnit její obsah, musel by jí nejdříve tzv. stáhnout (změna do stavu Přístupné).

5 Uživatelské rozhraní

Zásady, které by mělo ctít každé počítačové uživatelské rozhraní. Specifika webových aplikací. Kontext uživatelského rozhraní systému HIS.

5.1 Webová stránka jako uživatelské rozhraní

Webová aplikace (dále jen WA) je taková aplikace typu klient – server, která jako klientský program (software komunikující přímo s uživatelem) využívá webový prohlížeč.

Prohlížeč zde vystupuje v roli tzv. *tenkého klienta*, což znamená, že nezná logiku zpracovávaných dat, pouze předává příkazy a prezentuje. Znamená to vlastně určitý návrat k časům, kdy v podniku byl „na sále" jediný počítač a k němu byly připojené po budově rozmístěné „hloupé" znakové (terminálové) stanice.

WA jsou stále populárnější pro své výhody:

- Klientská aplikace (prohlížeč) je k dispozici prakticky všude s nulovou cenou za údržbu.
- Aplikace se spravuje a aktualizuje na jediném místě. Není třeba šířit a aktualizovat software všem uživatelům a řešit nekompatibilitu různých verzí.

- Nemusí se klást nároky na počítače uživatelů, a to ani na typ prohlížeče, ani na použitý operační systém. Aplikace tedy existuje v jediné verzi pro všechny.
- Veškeré výpočty (kromě některých triviálních kontrol vstupních hodnot) se obvykle provádějí v serveru.
- Server není zatížen „hloupou“ činností jako je rozmisťování grafických ovládacích a dekorativních prvků na obrazovce.

WA s uživatelem komunikuje zejména tak, že mu předkládá dynamicky generované stránky ve standardním formátu (X)HTML, jejichž sled evokuje práci s interaktivní lokální aplikací. Vstupní hodnoty očekává většinou z klasických webových formulářů.

Jsou zde však i některá nepříjemná *omezení*, které lokální aplikace neznají. WA obecně nemá moc nad konkrétním způsobem zobrazení objektů, které pošle prohlížeči a musí počítat s tím, že výsledná prezentace se může lišit. Chceme-li udržet kompatibilitu, nejsou k dispozici ani metody typu drag'n'drop (přetahování myší), které jsou jinak značně oblíbené.

Mnoha tvůrcům WA toto chybí natolik, že programují webový prohlížeč pomocí skriptovacího jazyka `JavaScript`. Tím riskují funkčnost WA, neboť nekompatibilita tohoto jazyka mezi různými prohlížeči je výrazně vyšší než v případě samotného jazyka HTML.

Patrně všechny WA mají v serveru určitou oblast svého programu společnou. Aby nebylo při tvorbě další WA nutné program kopírovat a potenciálně zatahovat chyby, existují ucelené programovací systémy, tzv. *frameworků*. Ty nabízejí hotovou společnou funkčnost a mnohdy ještě víc. Příkladem takového frameworku je systém `Zope` popsaný v kapitole 4.3, strana 18.

5.2 Zásady návrhu uživatelského rozhraní

Člověk není stroj a k věcem okolo sebe zaujímá pocitové postoje. Je-li *uživatelské rozhraní* (*User Interface*, UI) interaktivního počítačového systému nepřátelské, uživateli se jednoduše znelíbí, což ústí v to, že jej *opustí* a bude (vědomě či podvědomě) hledat jiné cesty k cíli. Je-li k používání takového rozhraní i nadále nucen, dostavuje se frustrace.

V případě nedobře navrženého rozhraní podnikového IS dochází přinejmenším ke ztrátě cenného času pracovníka. Času, který by mohl vydělávat. Funkčnost aplikace poskytované uživatelům je důležitá, ale způsob tohoto poskytování je stejně tak důležitý. Bez vhodného uživatelského rozhraní nezáleží na tom, jak geniální schopnosti aplikace má.

Abychom zajistili intuitivní, pohodlné a efektivní využívání, je třeba od první implementační chvíle dbát také na *zásady dobrého návrhu* (good design principles). Z dlouhé řady vodítek vyberme tyto, které se budeme snažit dodržet při návrhu nového systému:

1. **KONZISTENCE.** Je první a nejdůležitější zásada návrhu jakéhokoli systému. Funkčnost programu může být do určité míry nedokonalá, ale jeho prezentace musí být jednotná.

2. **Intuitivnost znamená sblížení.** Uživatel, kterému je umožněno, aby zkušenými pokusy ovládl neznámé funkce či prvky nového softwaru, jej podvědomě začne považovat za známý. I když pokusy nedopadnou dobře, výsledky mají být tak rozumné, že uživatel pochopí, co se stalo a poučí se.

   Lze vyslovit i obrácený výrok: Rozhraní je dobře navrženo, pokud se chová přesně tak, jak to od něj jeho uživatel očekává.

3. **Efektivní zprávy a návěští.** Je-li zobrazovaný text, který je hlavním zdrojem informací pro uživatele, nejasný či plný zkratek a kódů, budou mu lidé špatně rozumět, a to má vliv na vnímání celé aplikace. Zprávy mají být koncipovány jako pozitivní věty a evokovat *moc* uživatele nad programem. Také mají napovídat, jak lépe aplikaci používat. I zde je třeba být konzistentní ve výrazivu a pokud možno umisťovat zprávy na jednotném místě obrazovky.

4. **Změna velikosti písma a okna bez omezení.** Uživateli musí být umožněno zvětšit či zmenšit si písmo nebo okno aplikace bez toho, aby byla omezena funkčnost aplikace. Umístění prvků a velikosti je tedy třeba definovat relativně.

5. **S barvami opatrně.** Aplikace by neměla být příliš barevně kontrastní, vhodné jsou jemné odstíny. Funkčnost žádného prvku nesmí zcela záviset na barvě, neboť někteří uživatelé nemusejí barvy rozlišit. V takovém případě je třeba použít další indikaci.

6. **Barevné rozlišení stavů položek.** Je-li něco zobrazeno červeně, bude to zřejmě nepřístupné, již použité či jinak se lišící. Pozor ale na předchozí bod.

7. **Příliš zaplněná obrazovka zpomaluje práci.** Uživatel je pak zahlcen možnostmi, kterými se musí probírat.

8. **Skupiny ovládacích prvků.** Seskupení těch prvků na obrazovce, které k sobě mají logickou vazbu, například do viditelných nadepsaných rámců, pomáhá uživateli v prvotní orientaci.

9. **Co největší a zvýrazněné aktivní oblasti.** Je důležité, aby uživatel nemusel zdlouhavě zaměřovat kurzor myši na malé oblasti, jako je to ve webových aplikacích nechvalitebným zvykem. Aktivní oblast by se měla při přejetí kurzorem (tzv. hovering) tónovat. Velikost plochy, na kterou je třeba pro vyvolávání nějaké funkce zaměřit kurzor, by měla odpovídat významu této funkce.

10. **Posuvník zpomaluje práci.** Je-li třeba pro zobrazení nebo úpravy celého obsahu často posouvat oblast ve vertikálním směru, znamená to zdržení. Posun v horizontálním směru je ještě horší.

11. **Potlačení dopadu pomalosti načítání stránek.** Systém je na straně serveru implementován složitou strukturou kódu a dat prováděnou interpretačním programovacím jazykem. Z toho vyplývá, že i na rychlém serveru je doba načtení nové stránky v horším případě i v řádech sekund. Systém byl měl být s tímto vědomím navrhován tak, aby uživatelé byli touto skutečností obtěžováni co nejméně.

    V krajním případě je možné zvažovat i nasazení techniky AJAX (dynamické nahrazování částí stránek novou, na pozadí stahovanou, verzí – mnohem rychlejší než obnova celé stránky).

12. **Jazyk JavaScript použít zásadně jen jako přidanou hodnotu.** Nesmí na něm záviset dostupnost jakýchkoli funkcí aplikace a i na prohlížeči, kde není JavaScript (JS) možné dobře použít, musí být aplikace plně ovladatelná a intuitivní. JS nechvalně proslul svojí nekompatibilitou mezi prohlížeči. Univerzálně zprovoznit složitější aplikaci postavenou na JS je úkol velmi náročný.

    Dodržení tohoto bodu je přímém konfliktu s nasazením metody AJAX zmiňované v předchozím bodě, neboť ta na JS zcela závisí. V takovém případě by měl být systém navržen tak, aby vypuštění JS znamenalo přechod do klasického způsobu načítání celých stránek bez újmy na funkčnosti.

13. **Podpora rychlosti vnímání.** Uživatel by měl mít na stránce vždy možnost vidět co nejvíc relevantních údajů bez toho, aby se snížila přehlednost. Méně důležité údaje jsou zavřené v oblastech, ze kterých je zobrazen pouze titulek a ikonka evokující možnost rozevření. Po klepnutí na ní se oblast expanduje a po opětovném klepnutí zase zavře.

14. **Vnímání způsobů ovladatelnosti střídáním ukazatelů.** Webová aplikace musí napovídat prohlížeči, ve kterých oblastech stránek má zobrazovat který kurzor myši, což v době moderních stylů webových stránek je možné. Nad vstupními textovými poli je vhodná svislá čárka, nad oblastmi, na které je možné klepnout pro vyvolání nějaké

akce (včetně tónovaných částí), obvykle kurzor tvaru ruky (tzv. pointer). Pro zbývající oblasti je zde obvyklý kurzor tvaru šipky.

15. **Zaměření pozornosti na aktivní prvek.** Právě aktivní pole formuláře je zvýrazněno kontrastním okrajem (v portálu Plone implicitně oranžová).

16. **Klávesové zkratky.** Standardizované klávesové kombinace lze v dnešní době použít i na webu. Významné prvky rozhraní by mělo být možné vyvolat rychlou zkratkou (na platformách PC jde o `Alt+písmeno`).

17. **Ovladatelnost klávesnicí.** Musí být umožněn cyklický pohyb klávesou `Tab` mezi položkami formuláře.

18. **Minimalizace počtu operací k dosažení cíle.** Mnohé webové aplikace s oblibou používají k výběru z několika variant prvek roleta. Když je variant více, znamená ale výběr jedné z nich toto:

    - Zaměření úzkého proužku zavřené rolety, klepnutí,
    - případné zaměření jejího posuvníku a posun,
    - klepnutí na vybranou položku.

    Jedno z možných řešení této konkrétní neefektivnosti je ilustrováno na obr. 6.5 na str. 44. Existují však uživatelé, kterým na prvku roleta vyhovuje to, že mohou pomocí psaní písmen rychle vyhledat příslušnou položku, vědí-li předem, že se v seznamu nachází. Zde navrhovaný prvek UI by tedy měl být volitelný s možností návratu ke klasické roletě.

19. **Nejdůležitější pole formuláře jsou velká.** Hlavní políčko formuláře (příjmení při úpravě osoby, název projektu při úpravě projektu, časy začátku a konce práce při úpravě výkonu apod.) je znatelně větší, aby indikovalo svůj význam.

20. **Poznámky.** Ke všem databázovým položkám je možné napsat libovolně dlouhé poznámky. V přehledových sestavách je z každé poznámky zobrazováno prvních několik desítek znaků.

    Protože jak úvahy o návrhu, tak ani následné testování implementace neodhalí vše, je čas od času vhodné provést rešerši, zda lidé pole *Poznámky* nepoužívají k zadávání informací, pro které by bylo praktické zavést speciální položku příslušného formuláře.

21. **Přehledný a jednotný systém webových adres (URL).** Struktura URL má být průhledná a jednoduchá. Složitější formulářová data se programu předávají mimo

adresu URL (tzv. metoda `HTTP POST`). V adrese URL se naopak mají nacházet pouze nejdůležitější identifikační údaje odlišující například jeden datový záznam od druhého. Tím se také umožňuje ukládat funkční odkazy na důležité položky mimo systém.

Toto je další bod, který je konfliktní s případným nasazením techniky AJAX. Tam se adresa URL při přechodu z výše naznačených důvodů nemění.

22. **Nápověda.** V horní nebo dolní části obrazovek, které stojí za to vysvětlit, by měly být k dispozici rozevíratelné oblasti nadepsané např. *Nápověda* a obsahující popis toho, co uživatel před sebou vidí. Rozevíratelná nápověda neruší toho, kdo ji nepotřebuje a přenos několika stovek znaků textu ze serveru nečiní žádný problém. Součástí systému nápovědy jsou „bublinky“ krátkých vysvětlujících textů zobrazených při ponechání kurzoru myši na příslušné položce.

23. **Operace je možné vrátit.** Každý člověk chybuje a uživatel systému HIS nemusí mít obavy, že jeho činnost způsobí nevratnou škodu. Více viz kapitola 6.1.7, strana 45.

24. **Standardní chybová stránka s formulářem na zadání komentáře k chybě.** Jde o jednotný způsob komunikace s uživatelem v případě chyby a metoda včasného získání podrobnějších informací o chybě. Viz kapitola 6.1.8, strana 47.

5.3 Zasazení do existujícího rozhraní

Evidenční dílčí systém HIS bude integrální součástí portálu Plone, který má již zavedené webové uživatelské stereotypy. Nabízí své programátorské rozhraní (API), UI a ctí zásady jednotného a standardizovaného přístupu až na úroveň striktního protokolu XHTML. Jeho původní zaměření je poskytovat správu, navigaci a vyhledávání v dokumentech různých typů. Dokumenty jsou ukládány v objektové databázi systému Zope. Pro tyto objekty jsou zde propracované prezentační a manipulační stereotypy.

Naše zadání je v hlavní části jiné: Aplikace o více tabulkách s provázanými daty spravovanými externí databází, která je ovládaná jazykem SQL. Prezentační a manipulační stereotypy je zde tedy třeba vytvořit znovu. Cílem vývoje je, aby uživatel při používání nepoznal předěl mezi těmito dvěma oblastmi systému.

6 Realizace

Popis uživatelského rozhraní a fungování systému HIS.

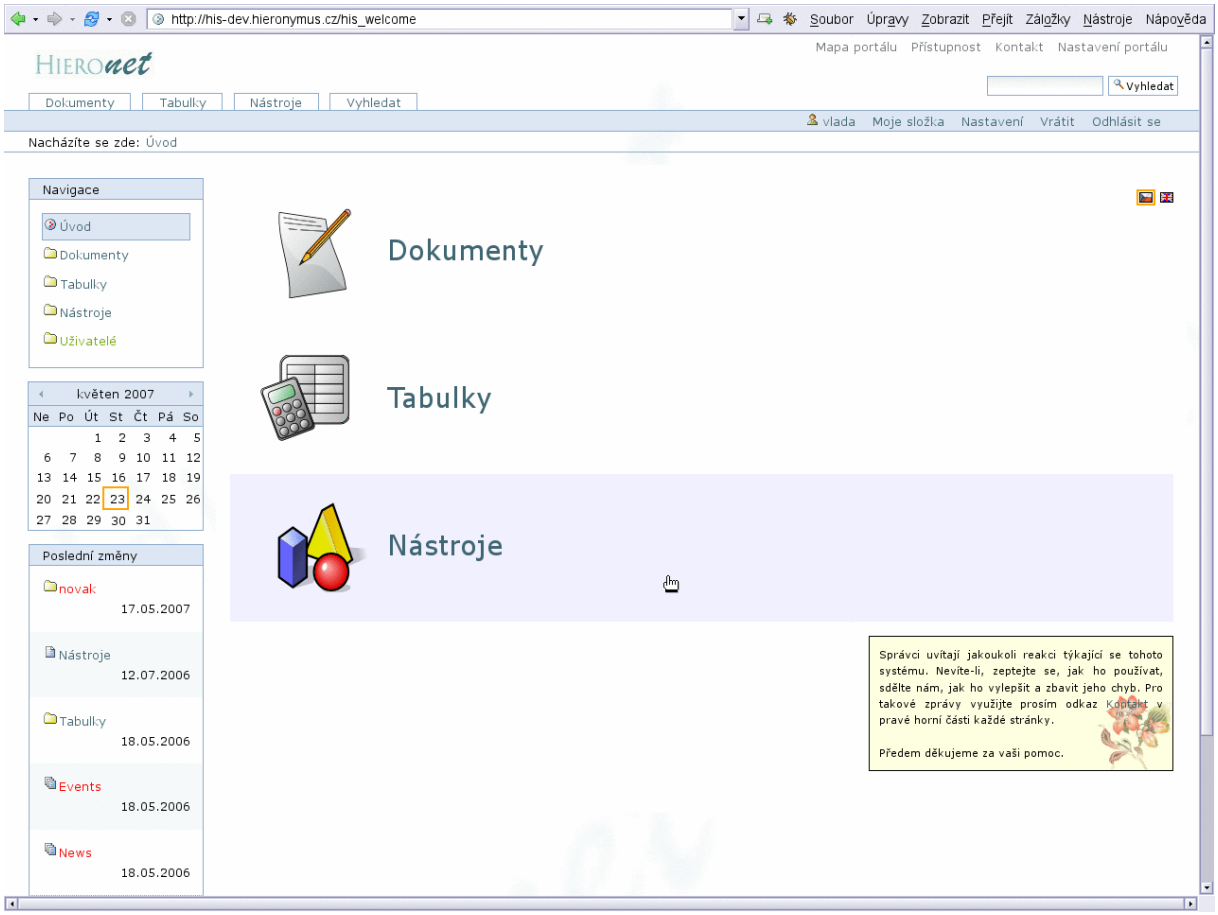
Z důvodu návaznosti na předchozí text začíná tato část popisem realizovaného uživatelského rozhraní systému HIS. Nižší vrstvy aplikace jsou popsány dále.

6.1 Uživatelské rozhraní systému HIS

Obrázek 6.1 ukazuje uvítání uživatele do systému a lze na něm také ilustrovat výchozí vzhled portálu Plone.

Horní a levá část obrazovky je stabilní a nabízí uživateli globální akce. Shora zprava: Odkazy na strukturu (mapu) portálu, údaje o přístupnosti pro postižené, kontaktní formulář na správce. Ten zde může rychle přejít ke globálnímu nastavení. Níže je vyhledávací pole, které kromě hledání v dokumentech bude nabízet i hledání v datech uložených v relační databázi tak, aby se uživatel dostal k výsledkům minimálním počtem úkonů. Klepnutí na logo vlevo nahoře znamená přechod na výchozí stránku.

Modrý pruh přes celou šíři nabízí individuálně závislé uživatelské akce (např. přechod do osobní složky, vlastní nastavení nebo odhlášení), modré záložky vlevo nabízejí přechod ke globálním kategoriím obsahu nebo hlavním funkcím. Řádek *Nacházíte se zde* v každé situaci zobrazuje cestu k zobrazovanému objektu. To umožní budoucí opakovanou navigaci po paměti na místo.



obr. 6.1 Úvodní obrazovka

Celý levý svislý pruh je vyhrazen pro graficky oddělené boxy zobrazované nebo skrývané v závislosti na kontextu. Příkladem je strom navigace, kalendář, seznam naposledy změněných položek, oblíbené položky. V systému HIS zde budou také individuální statistiky otevřených projektů nebo vlastních nevyřešených úkolů.

V největší hlavní části se zobrazují dokumenty, správy složek a veškerá kontextově závislá komunikace aplikace s uživatelem. Na obrázku je úvodní hlavní rozcestník do tří globálních kategorií: *Dokumenty*, *Tabulky* a *Nástroje*. První z nich odkazuje do stromu dokumentů, což je původní účel portálu Plone. Odkaz Nástroje vede na obrazovku s webovými systémovými nástroji ovládajícími a sledujícími různé aspekty počítačové sítě.

Nejdůležitější je nyní prostřední odkaz Tabulky, kde se nachází celá evidenční aplikace. Při pořizování snímku se v oblasti položky Nástroje nacházel kurzor myši a fialový pruh ilustruje jednu z důležitých zásad UI popsaných výše.

6.1.1 Jazyková lokalizace

Jak bylo zmíněno v čl. 3.3, systém Plone podporuje vícejazyčné dokumenty a rozhraní. Jazyk rozhraní lze přepínat za běhu pomocí malých ikonek vpravo, a to platí i pro systém HIS. Lokalizace do jazyků jiných než angličtina se provádí až ve webových šablonách prezentační vrstvy IS (nejbližší uživateli).

Vývoj softwaru probíhá v jednotném jazyce – angličtině. Při prezentaci stránky uživateli je konzultován takzvaný jazykový katalog systému `gettext` v adresáři `i18n` (zkratka slova *internationalization*). Ten obsahuje páry originál – překlad a lze ho nechat upravovat samostatně někým jiným.

Vzor katalogu je většinou vytvářen automaticky, speciální program prochází webové šablony a dokonce i zdrojový kód programu a hledá řetězce označené jako překladatelné. V další fázi synchronizuje nově nalezenou skupinu nových, změněných a odstraněných řetězců tak, aby nedošlo ke ztrátě již hotových překladů. Následuje příklad položky katalogu (první řádek odkazuje na místo v programu, ze kterého řetězec pochází):

```
#: ./his_zope.py:156
msgid "Add new effort"
msgstr "Přidat nový výkon"
```

6.1.2 Řetězce

Na více místech budoucího IS bude třeba mít k dispozici sadu krátkých řetězců, ze kterých bude moci uživatel jeden vybírat (příkladem je stav projektu) nebo i vybrat několik z nich zaškrtnutím (příklad: klíčová slova asociovaná s položkami). Aby nebylo nutné k přidání nového klíčového slova povolat programátora, byla obecná editace řetězců sdružena do jedné z tabulek evidenčního systému. Programátor definuje jen *druhy řetězců* a vazbu druhů na funkčnost softwaru.

V současnosti je funkční pouze jeden druh, a to **Klíčové slovo výkonu** (**Effort Keyword**): Je možné spravovat sadu klíčových slov, kterými pak kterýkoli uživatel může označit položku výkonu, aby například naznačil typ provedené práce.

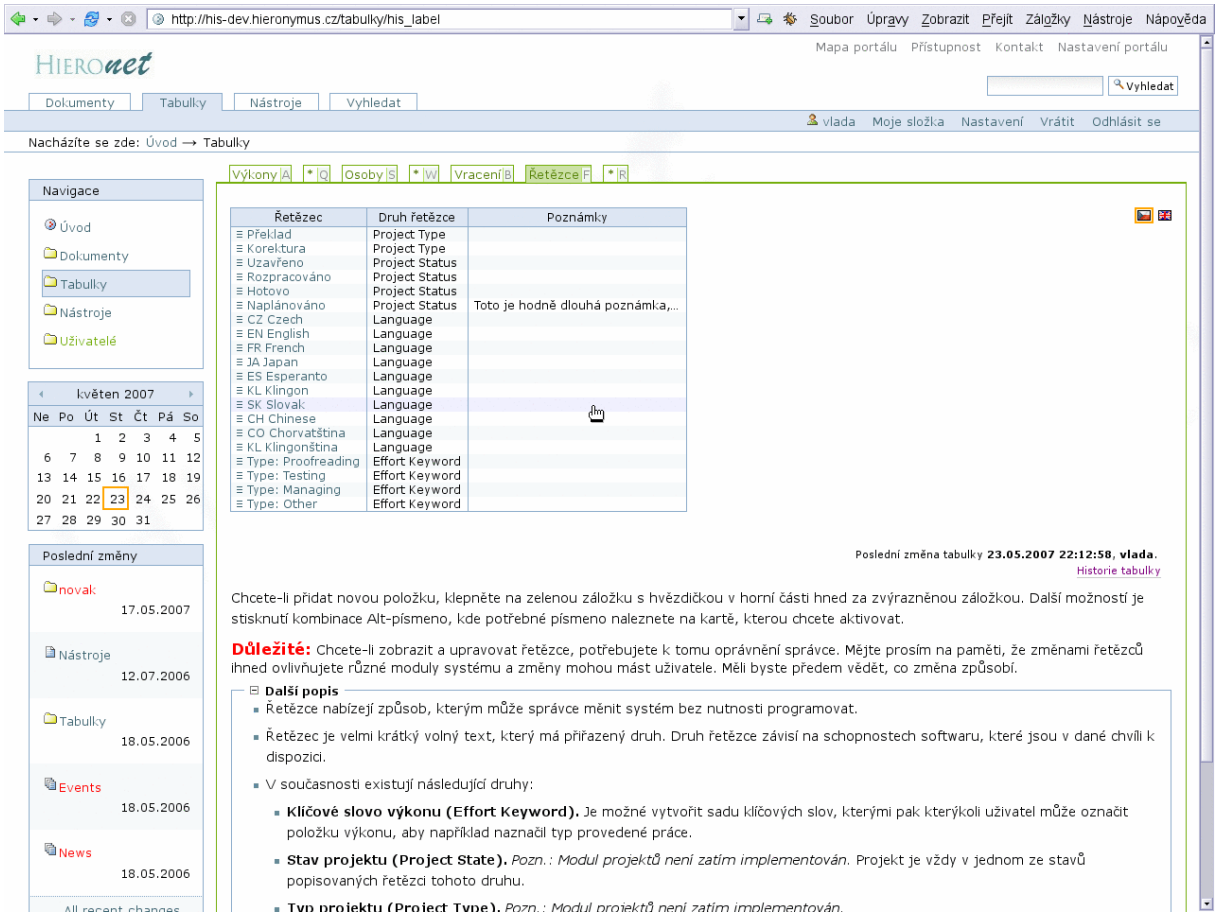
Zobrazení a úpravy tabulky Řetězce je umožněno pouze správcům. To proto, že okamžitě ovlivňuje různé jiné moduly a neuvážené změny by mohly zmást uživatele.

Grafická ukázka této jednoduché tabulky je v následující části, kde slouží pro ilustraci základního typu tabulkového přehledu.

6.1.3 Základní typ tabulkového přehledu

Všechny stránky evidenčního systému jsou ohraničeny rámečkem se záložkami, pomocí kterých je možné:

- přepínat pohled na různé tabulky systému, ke kterým má daný uživatel přístup,
- ihned přejít k založení nové položky v libovolné přístupné tabulce,
- identifikovat písmeno, které je třeba stisknout společně s klávesou `Alt` za účelem rychlého přechodu do příslušné záložky.



obr. 6.2 Základní typ tabulkového přehledu

Obrázek 6.2 ukazuje základní tabulku vytvořenou pomocí makra METAL (viz kapitolu 4.3.5, strana 24). Toto makro je v aplikaci použito na více místech a programátorovi pak stačí pouze vhodně připravit data pro zobrazení. Dalším efektem je pak jednotnost ovládání uživatelem. Celá aplikace využívá mnoho dalších maker, příkladem jsou jednotné údaje vpravo dole o posledních změnách tabulky.

Datové položky, pro jejichž zpracování je možné přejít do podrobnějších formulářů, jsou označeny symbolem ≡ evokujícím formulář. Pozadí celého řádku, který je v daném okamžiku pod kurzorem myši, je opět tónováno a pro jeho aktivaci není nutno zaměřovat se jen na klasický webový odkaz v prvním sloupci (existující pro méně schopné webové prohlížeče). Samozřejmostí je bublinková nápověda, která napoví, co se stane v případě klepnutí na danou oblast.

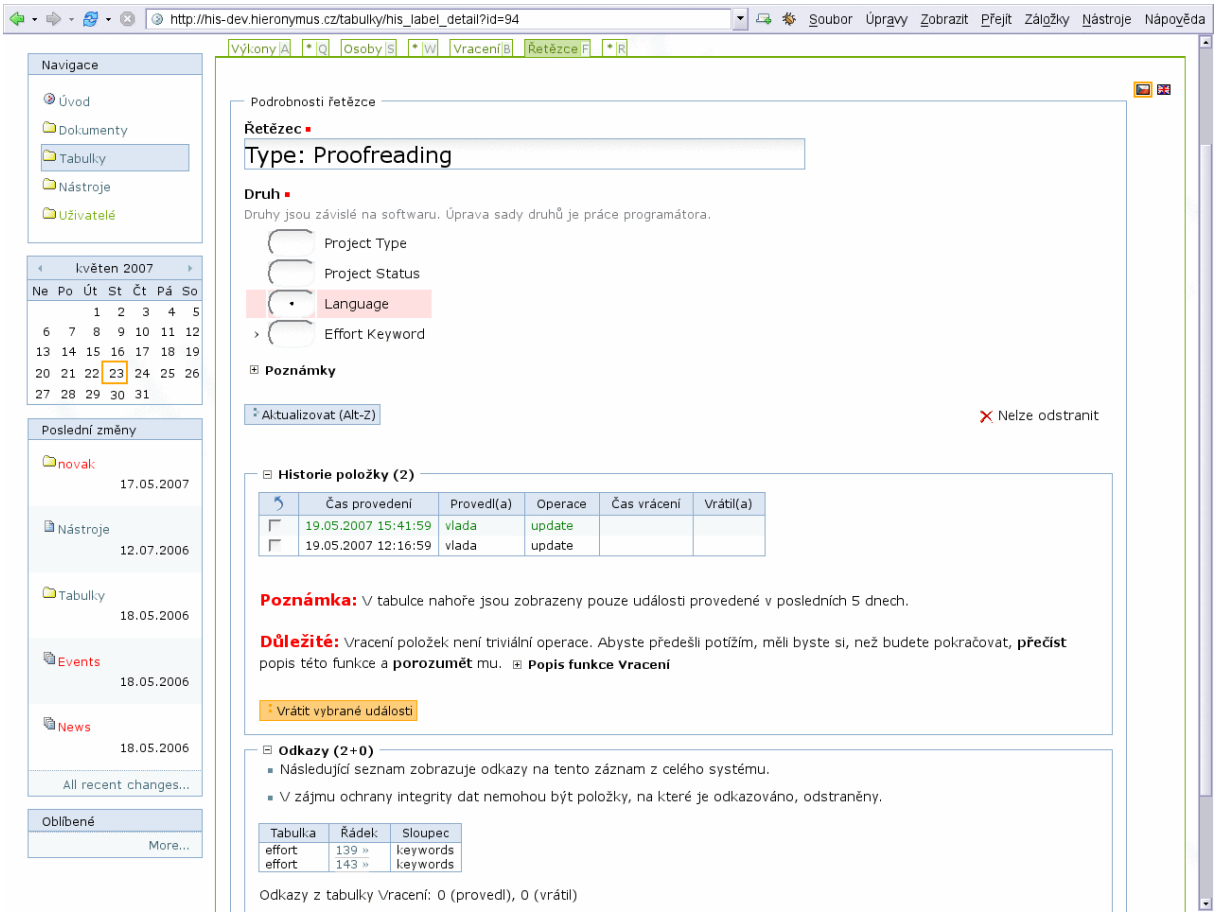
Je-li to pro daný typ tabulky potřeba, zobrazují se pod přehledem doplňující informace a odkazy, jak bude vidět u evidence výkonů.

6.1.4 Základní typ editace datové položky

Klepne-li uživatel v tabulkovém přehledu položek na některou z nich, přejde tím k formuláři zobrazujícímu všechna pole dané položky v ovládacích prvcích. Tak lze položku upravovat. Jako příklad slouží editace řetězce na obr. 6.3.

Dole pod položkou se nacházejí dva rozevřené bloky. V prvním je možné zobrazit makro *Historie změn* načtených z tabulky Vracení. Po klepnutí na některý z řádků se zobrazí přesný popis změny a změna se vybere. Výrazným tlačítkem níže lze vybrané změny přímo vrátit. Více viz kapitola 6.1.7. Blok *Odkazy* poskytuje přehled o reálných databázových závislostech (název odkazující tabulky, interní identifikace řádku a identifikátor pole v té tabulce neboli sloupec). Vzhledem k tomu, že systém nedovolí odstranit položku, na kterou odkazuje nějaká jiná existující položka, je toto dobré místo, ze kterého lze navštívit všechny existující odkazy a upravit je, případně odstranit.

V tomto formuláři je obvykle možné záznam také zcela odstranit, a to zvýrazněným tlačítkem vpravo (po klepnutí se zobrazí potvrzovací okno s informací, že operaci půjde v případě potřeby vrátit). Protože zjištěný seznam odkazů na tuto položku není prázdný, je v ukázce tlačítko *Odstranit tuto položku* nahrazeno textem *Nelze odstranit* a ikonkou symbolizující nemožnost.



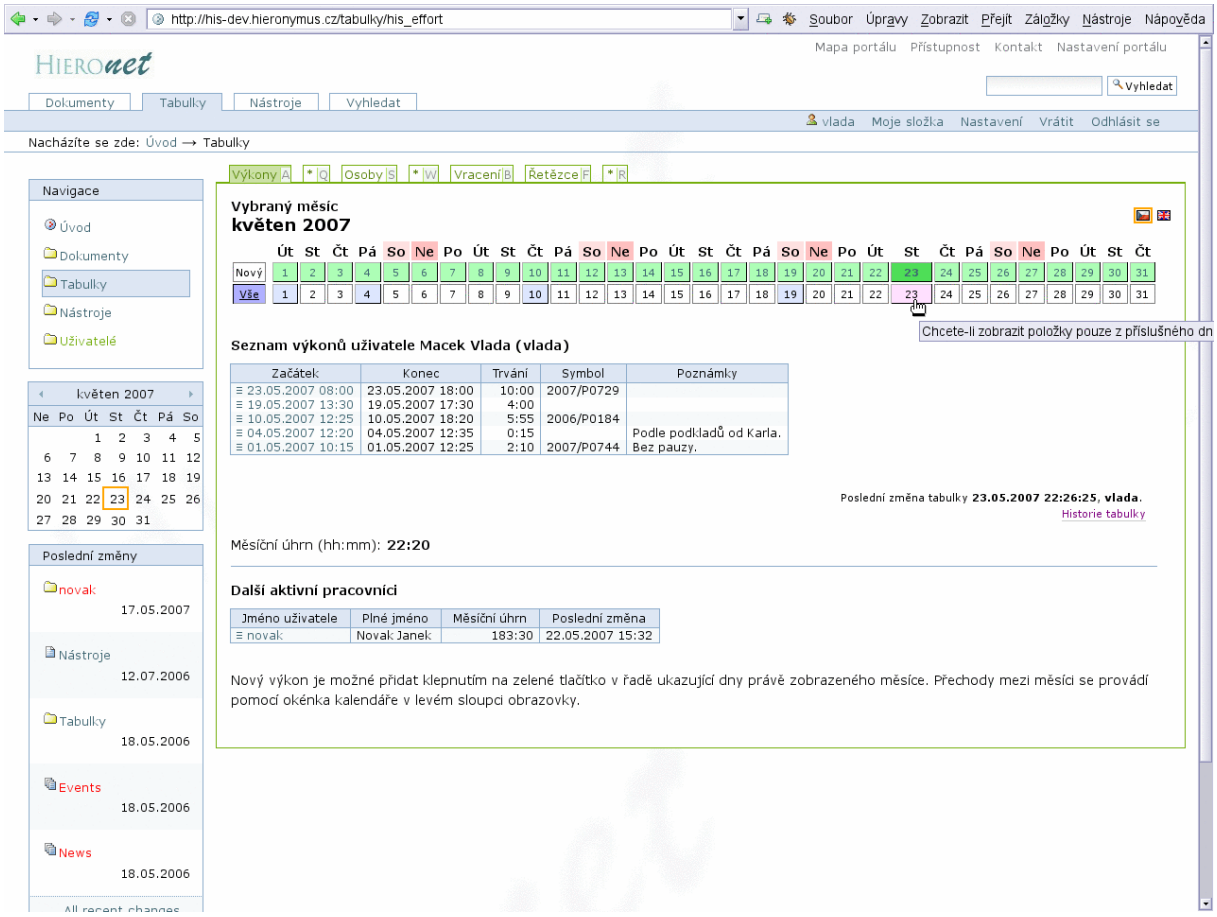
obr. 6.3 Základní typ editace datové položky

Formulář stejný jako pro editaci existující položky bude zobrazen v případě, rozhodne-li se uživatel vytvořit novou položku tabulky. Jde o stejnou šablonu, která automaticky vyloučí zobrazení bloků Historie a Odkazy.

6.1.5 Měsíční přehled výkonů

Primárním cílem je stále podpora rychlého zadávání a úprava dat. Zde se jedná o několik položek denně u každého uživatele (obvykle minimálně dvě: práce před obědem a práce po obědě). Zároveň je pro účtování práce potřeba rychle dostupný měsíční přehled.

Na obr. 6.4 je příklad řešení. Nahoře je výrazně označeno, pro který měsíc je přehled zobrazen. Pod tím je lineární kalendář daného měsíce s přehledně označenými víkendy a dvěma řadami tlačítek. Tyto řady budou pomocí stylů CSS vyloučeny z tiskového výstupu, protože nejsou na papíře potřebné.



obr. 6.4 Měsíční přehled výkonů

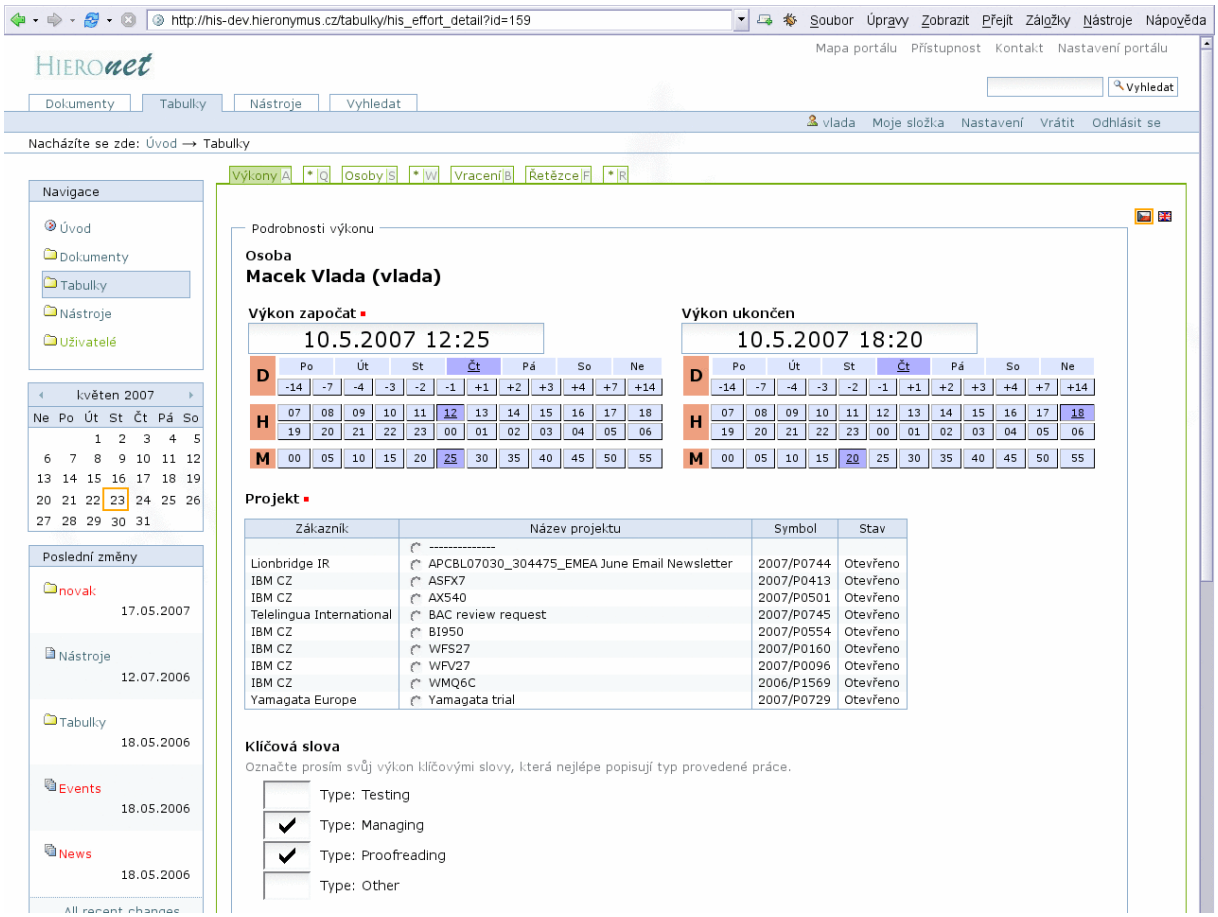
Použitím horní řady tlačítek lze ihned přejít k zadání nového výkonu pro konkrétní den. Ve spodní řadě existují modrá tlačítka pouze u těch dnů, pro které je zadán začátek nějakého výkonu. Klepnutím na některé z modrých tlačítek se pomocí procedury JavaScriptu rychle zúží celoměsíční zobrazení v tabulce pouze na daný den. Přepnutím tlačítkem Vše se opět zobrazí celý měsíc.

Kalendář vlevo lze v modulu Výkony použít na výběr aktivního měsíce. Oranžový rámeček označuje aktuální den (ten je zvýrazněn také širšími tlačítky v řadě). Je-li vybraný měsíc jiný než aktuální, zobrazí se pod kalendářem odkaz pro rychlý přechod zpět k dnešku.

V dolní části je trvale zobrazen hodinový souhrn za celý měsíc. Správce s rolí `HIS-EffortManager`, má k dispozici také spodní tabulku vypisující ostatní pracovníky s nenulovým počtem hodin v daném měsíci s možností přechodu na jejich měsíční přehled.

6.1.6 Editace výkonu

Výkon práce nemusí být časově ukončený. Systém je vázán na zabezpečovací zařízení budovy a pokud si zaměstnanec osobní kartou otevře vstupní dveře, resp. z budovy odchází, otevře se mu nový výkon na základě posledního, resp. se otevřený výkon práce zavře. Pro ruční úpravy je k dispozici rozhraní na obr. 6.5.



obr. 6.5 Editace výkonu

Modré tlačítkové oblasti umožňují na několik klepnutí myši nastavit potřebný čas s přesností na pět minut, případně přejít rychle na jiný den. Klepnutím na oranžové záhlaví tabulek D-H-M se nastaví aktuální datum a čas. Čas je samozřejmě možné upravit i ručním zápisem do polí a tlačítka podle toho reagují. Pokud je v poli nesmyslný údaj, je jeho pozadí vybarveno červeně.

Zde je ukázka nahrazení prvku roleta odkazovaná v dřívější kapitole: Možnosti, z nichž se má jedna vybrat, se nacházejí v tabulce, kde na každém jejím řádku je přepínač (radio button).

Vybraný řádek je navíc zvýrazněn. Další výhodou oproti roletě je více přehledněji zobrazených informací.

Zopakujme, že u zadavatele funguje skupina předchozích IS, která má dohromady podobnou funkčnost, jakou se snažíme docílit novým systémem HIS. Protože HIS bude implementován a nasazován u zadavatele postupně a pilotním modulem bude *evidence výkonů*, budou položky tohoto modulu vázány na projekty ve stávajících IS. Ke změně vazby na úkol, jak bylo popsáno v analýze, dojde později.

6.1.7 Historie a vracení zpět

Objektová databáze systému Zope nabízí možnost vracení všech dříve provedených operací, a to i nesouvisle. V aplikaci používající data z relační databáze byla tato vlastnost dodána také: Každá změna dat ve všech evidenčních modulech se popíše přidáním záznamu do tabulky *Vracení*, a to přibližně ve formátu na obrázku 6.6.

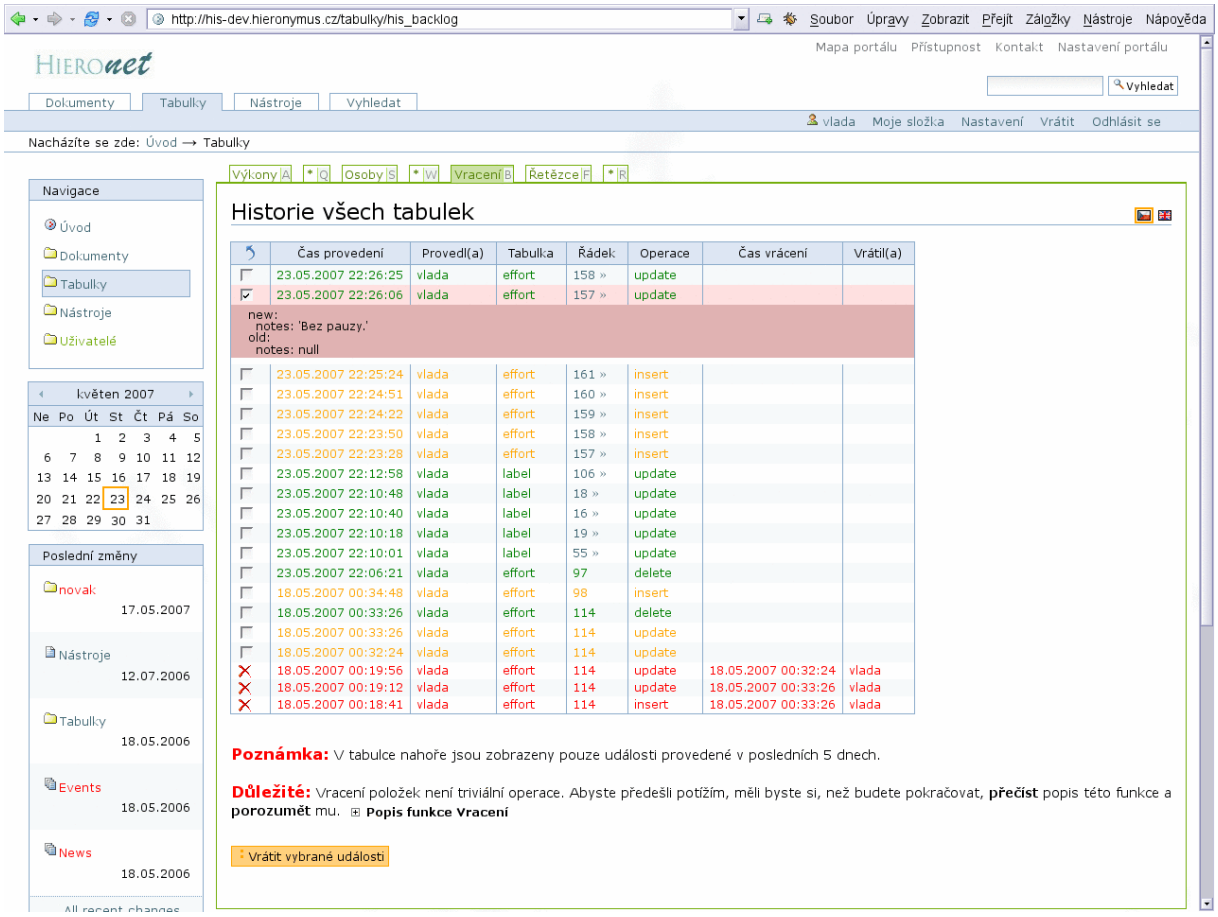
Každá událost zaznamenaná v tabulce Vracení nese informaci o jedné ze tří databázových modifikačních operací: vkládání (`INSERT`), změny (`UPDATE`) a odstranění (`DELETE`), které se vztahují ke konkrétnímu záznamu (řádku v tabulce) a také nese údaj o tom, který sloupec záznamu se změnil, jak a kdo změnu provedl.

Po klepnutí na řádek je zobrazen červený podřádek, kde je v jazyce `YAML`[8] podrobně popsána změna tak, aby jí systém mohl načíst. V ukázce je aktivní jen druhý řádek. Klepnutí na aktivovaný (a zaškrtnutý) řádek nebo jeho podřádek opět položku deaktivuje (a zruší zaškrtnutí). Po klepnutí na spodní tlačítko jsou vybrané operace vráceny. Vracení však také přidá běžný dopředný záznam (jak je vidět v posledních šesti řádcích), takže i vracení lze vrátit. Pokud byla jakákoli událost už vrácena zpět, je o tom učiněn záznam do posledních dvou sloupců a řádek již nelze použít.

Obyčejný uživatel má přístup k minimálně těm položkám tabulky Vracení, které sám vytvořil změnami dat v jiných modulech systému HIS, například Výkony. Tyto své změny může také vracet. Správce systému si může zobrazit a vrátit kterékoli změny.

Existuje důležité *omezení v zájmu konzistence dat*: Mohou být vráceny pouze události na takových položkách, kde cílový stav polí („sloupců“) je totožný s aktuálním stavem těchto měněných polí.

8 `http://www.yaml.org`



obr. 6.6 Historie a vracení zpět

Pro jednodušší práci s tabulkou Vracení jsou jednotlivé události obarveny nápovědným algoritmem. Barvy indikují stav položek a závislosti dat a mají následující významy:

- *Zelená* barva značí události, které lze přímo vrátit, neboť nebylo nalezeno žádné viditelné porušení závislostí. Tyto položky lze tedy bez potíží vybrat k vrácení, pokud tato operace nebude zastavena databázovým serverem (není časté).
- *Červená* znamená, že položka byla už vrácena (použita) a její opakované použití není možné.
- *Oranžová* událost znamená, že vyhodnocení jejího stavu se pomocnému algoritmu nepodařilo a závislosti dat nejsou známy. Operace vrácení může a nemusí být úspěšná.
- *Černá* je barva událostí, které není možné přímo vrátit kvůli neslučitelným změnám sloupců (viz popis omezení výše). Nicméně i u těchto položek jsou zaškrtávací políčka, protože tyto události lze učinit vratnými (zelenými) vrácením některých pozdějších

zelených změn, které tuto černou nyní blokují. Tak je zachována možnost vrátit několik vzájemně závislých změn najednou.

Je-li operace vrácení spuštěna, vrátí zpět změny provedené operací `UPDATE`, odstraní celý řádek, je-li operací `INSERT` a naopak vloží nový řádek (lépe řečeno ten starý a dokonce se stejným číselným `id`) pro vrácení operace `DELETE`. Konzistenční omezení popisované výše je pochopitelné v platnosti pouze pro operace `UPDATE` a `INSERT`, nicméně jakákoli operace vrácení může být zastavena samotnou databází, zjistí-li ta, že vrácení by mohlo narušit některé povinné vazby mezi daty.

Po úspěšné operaci vrácení bude vytvořena další položka tabulky Vracení, pomocí které je možné vrátit právě provedenou operaci. Jak je v tomto informačním systému obvyklé, každá změna dat se od první chvíle požadavku až do předání informací uživateli provádí v *transakcích*. Všechny požadované změny dat (i mnoho najednou) jsou buď provedeny najednou nebo v případě chyby najednou zapomenuty.

Makro této tabulky nabízí zúžení zobrazovaných dat také na konkrétního uživatele, tabulku (odkaz Historie tabulky, obr. 6.2 vpravo na straně 40) a na položku (obr. 6.3 na straně 42).

6.1.8 Když dojde k chybě…

Dojde-li v programu k jakékoli chybě, ze které se nelze zotavit za běhu, vyhlásí se *výjimka*, která putuje vzhůru v zásobníku volaných metod a může být *zachycena*. Pokud k zachycení nedojde, jsou rozpracované operace v databázích podporujících transakce zrušeny (tzv. *rollback*), nehrozí tedy ztráta integrity dat.

Dojde k zobrazení jednotné chybové zprávy s popisem a žádostí o napsání okamžitého komentáře obsahujícího relevantní údaje například pro reprodukci chyby vývojářem.

Ve chvíli zobrazení se na pozadí odesílá první e-mailová zpráva správci portálu, ve které je podrobný popis situace včetně tzv. *tracebacku* (zásobníku funkcí ve chvíli výjimky) a dalších ladicích informací. Pokud má aktuální uživatel oprávnění správce portálu, je traceback k dispozici ihned na chybové stránce po expandování bloku *Podrobnosti pro vývojáře*. Nejdůležitější část tracebacku, který může mít délku i dvou stránek, je dole. Proto expandování zároveň posune stránku dolů. Nachází se zde popis nejvnitřnějších dvou tzv. volacích rámců prováděného kódů a dokonce konkrétní zdrojový řádek, na kterém došlo k výjimce s dvouřádkovým okolím. Součástí tracebacku je i výpis všech lokálních proměnných posledních dvou rámců společně s jejich datovým type a obsahem. Vývojář má tedy ihned k dispozici kompletní údaje o chybě a většinou je již z tohoto zřejmá i příčina.

Vyhoví-li uživatel žádosti o zadání komentáře k chybě, je odeslána druhá e-mailová zpráva se stejným identifikátorem v Předmětu a záhlavím `References:` nastaveným tak, aby seřazení zpráv podle vláken v poštovním programu pro přehlednost umístilo tuto druhou zprávu jako podřízenou té první s tracebackem.

Důsledkem tedy je, že správce je vždy ihned o chybě informován bez ohledu na to, zda se uživatel rozhodne napsat k ní komentář.

Díky podrobnému tracebacku nemusejí chybové hlášky být podrobné a obsahovat například údaj o tom, která proměnná má nesprávný obsah. Z toho vyplývá, že je možné je jednoduše zařadit do systému lokalizace do jiných jazyků. Toto ještě není v prezentované verzi systému hotové.

6.2 Struktura programu

Logika a pozadí aplikace se skládá ze sady tříd uložených ve třech zdrojových souborech v jazyce Python: `his_backend.py`, `his_entries.py` a `his_zope.py` v pořadí od datového úložiště k uživateli. Třídy připravují data pro webové šablony a přejímají od nich pokyny. Podle potřeby používají pomocí textových příkazů jazyka SQL relační databázi PostgreSQL[9], kde jsou data udržována. Kooperují v následujících činnostech nad zobecněnými tabulkovými daty:

- zobrazení tabulkového přehledu uživateli,
- zobrazení detailu položky,
- vložení, úpravy a odstranění položky,
- vložení události z předchozího bodu do tabulky Vracení (`backlog`),
- použití události z tabulky Vracení k obnovení stavu před touto událostí,
- uplatňování omezení na základě oprávnění uživatele.

Systém je navržen tak, aby nebyl příliš závislý na systému Plone a Zope pro případ, že zadavatel tyto v budoucnu opustí. V jakémkoli prostředí, kde budou k dispozici systém webových šablon ZPT (které také nejsou závislé na Zope) a nějaký autentizační mechanismus, tam bude možné systém HIS zprovoznit. Z toho důvodu je na Zope zcela závislý jen malý modul `his_zope`, a ten v podstatě neobsahuje žádnou logiku aplikace a slouží jako adaptér.

9 `http://www.postgresql.org`

Další, ale už zcela minoritní závislostí systému HIS na Zope je třída pro práci s časem `DateTime`.

Cíl odpovídá principu objektového programování. Prezentační adaptér `his_zope` má pracovat pouze s tzv. *datovými objekty* nejvyšší úrovně, které reprezentují správu datových položek různých typů (řetězec, výkon, osoba, ...). Viz dílčí kapitoly o datových třídách níže. Tyto jednotlivé objekty vhodným způsobem rozšiřují obecnější schopnosti tříd, ze kterých vycházejí, takže každá schopnost je definovaná pouze jednou a je k dispozici pomocí jednotného rozhraní.

Zdrojový kód je bohatě komentován tak, aby popsal podrobnosti, které se nacházejí pod úrovní, které se může věnovat tento dokument.

6.2.1 Třídy

Diagram 6.7 ukazuje strukturu všech tříd. Tři naznačené oblasti lze považovat za vrstvy IS, kde uživatel komunikuje s oblastí ve spodní části a horní část je blízko databázovému serveru.

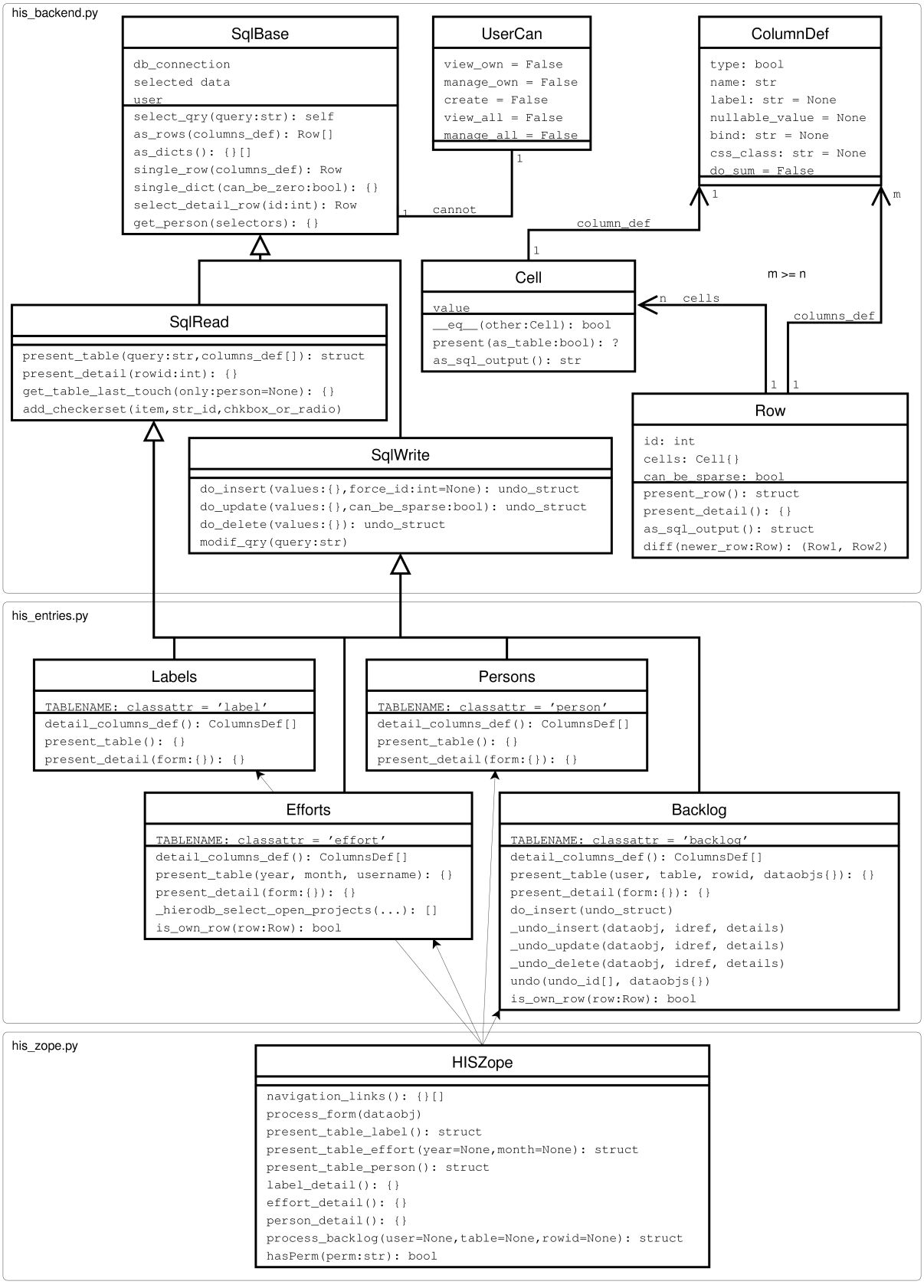
Použité symboly: `[]` je sekvence položek (pole), `{}` je asociativní pole (v terminologii Pythonu `dict`). Značka `{}[]` tedy znamená sekvenci asociativních polí. Je-li struktura dat složitější, je použito termínu `struct`. V jazyce Python není klasický model tajení atributů. Jakýkoli objekt může přistupovat k libovolnému atributu jiného objektu. Určité soukromí požívají atributy, jejichž názvy začínají podtržítkem. Tomu odpovídá i značení v diagramu.

6.2.1.1 Třída UserCan

Systém oprávnění uživatelů k práci s položkami jedné tabulky je na základě poznatků v kapitole 3.2 (strana 15) dělen takto: *Zobrazení vlastních položek* (`view_own`), *správa vlastních položek* (`manage_own`), *vytváření nových položek* (`create`), *zobrazení všech položek* (`view_all`) a *správa všech položek* (`manage_all`). Oprávnění „správa“ se vztahuje na operace úprav a odstraňování položek.

Třída při inicializaci rozhodne, která „vyšší“ oprávnění aktivují i oprávnění „nižší“. Tazatel na oprávnění tedy nemusí rozlišovat mezi tím, jestli má uživatel k funkci/položce přístup na základě vlastnictví nebo toho, že je správce. Jednoduše se dotáže na jedno oprávnění.

Modul `his_zope` vytváří instanci této třídy konverzí z oprávnění systému Zope (viz kapitola 4.3.4.2, strana 23) a dodá ji inicializacím datových objektů. (Instance datových objektů se vytvářejí pro každou operaci nad řádkem datové tabulky.)



obr. 6.7 Diagram tříd

6.2.1.2 Třídy ColumnDef, Cell

Systém HIS funguje jako prostředník mezi dvěma systémy, z nichž každý zpracovává informace jinak. Aby byla zajištěna bezpečná transformace hodnot a jejich spolehlivé porovnávání (to za účelem sledování změn), je nutné každou atomickou hodnotu (číslo, řetězec, čas, ...) uchovávat v interním tvaru. K přístupu k ní se pak používá specializovaných metod.

Tento postup je nutný z toho důvodu, že jazyk Python je dynamicky typovaný: podle potřeby převádí hodnoty z jednoho typu do druhého a také v současné verzi používá dva druhy textových řetězců. Proto je nutné udržovat nad typy hodnot kontrolu explicitně.

Objekt třídy `ColumnDef` definuje sloupec tabulky. Ve vyšších třídách se využívá jak k popisu jednoho sloupce relace databáze pro vstup i výstup (atributy `name`, `nullable_value` a `bind`), tak i jednoho sloupce tabulky přímo prezentované na webové stránce (atributy `label`, `css_class` a `do_sum`).

Objekt třídy `Cell` skutečně uchovává jedinou hodnotu v závislosti na předaném typu této hodnoty pomocí objektu `ColumnDef`. Atribut `type` určuje jeden z těchto typů hodnoty:

- **int**, celé číslo omezené na 32 bitů,
- **text**, řetězec interně uchovávaný v univerzálním objektu `unicode`. Pokud je inicializaci objektu Cell předán řetězec ve tvaru pole znaků, předpokládá se kódování UTF-8 a je provedena automatická konverze,
- **hhmm**, jde o interval v hodinách a minutách a při prezentaci hodnoty je takto zobrazen; jinak je s ním nakládáno jako s hodnotou typu `int`,
- **DateTime**, jde o třídu definovanou systémem Zope pro uchování a práci s časem,
- **inttuple**, n-tice čísel typu `int`; například odkazy na identifikátory zaškrtávacích polí.

Prezentace dat webovým šablonám a přejímání dat od nich jsou možné dokonce přímo v těchto typech. Systém šablon ZPT se postará o to, aby byl uživatelem odeslaný formulář přímo do těchto typů převeden.

Až na poslední popsaný typ je možné všechny hodnoty přímo ukládat do databáze pomocí databázového adaptéru, který respektuje i typ `DateTime`. Popis zpracování typu `inttuple` viz dále.

Každá hodnota může být v definici sloupce označena jako „nullable“. To znamená, že nemusí být zadána, atribut sloupce `nullable_value` určuje, jaká konstanta je považována za

prázdnou (v Pythonu `None`, v relačních databázích `NULL`). Pro řetězce je to obvykle prázdný řetězec, pro číselné identifikátory to může být nula.

Metoda `present()` prezentuje drženou hodnotu a podle parametru se určí, jestli pro tabulku nebo pro detailní formulář. Jediný rozdíl mezi tím je v současnosti v tom, že pro tabulkové zobrazení se oříznou řetězce na méně než 30 znaků. Pro nulové hodnoty je vrácena jim příslušná hodnota `nullable_value`.

Metoda `as_sql_output()` vrací drženou hodnotu jako řetězec, který je bezpečně možné připojit do textového modifikačního příkazu SQL tak, aby tato hodnota byla databází převzata tak jak má. Prázdné hodnoty přejdou v řetězec `'NULL'`, datum a čas je převeden do standardního formátu ISO 8601 a začátky a konce řetězců jsou speciálně označeny.

6.2.1.3 Třída Row

Objekt této třídy provádí abstrakci celého jednoho řádku tabulky, který má typicky několik sloupců a menší nebo stejný počet hodnot. Při inicializaci tedy obdrží sekvenci definic sloupců `ColumnDef` a asociativní pole samotných hodnot. Vytvoří pro hodnoty objekty typu `Cell` a je-li zadáno číselné `id` řádku, uloží ho zvlášť.

Metodou `present_row()` nabízí prezentaci řádku tabulky pro jednoduché zobrazení v tabulkovém přehledu. Medota `present_detail` vrací asociativní pole hodnot pro šablonu detailního formuláře jedné položky. Kromě toho třída obsahuje metody, díky kterým lze k její instanci přistupovat jako k asociativnímu poli.

Pokročilejší metoda `diff()` jako parametr přijímá jinou instanci třídy `Row`, jejíž hodnoty se považují za novější. Jako výsledek jsou vráceny dvě nové instance třídy `Row`. První obsahuje pouze změněné hodnoty v původním řádku a druhá pouze změněné hodnoty po změně. Metodu lze použít jak k bezpečnému porovnání dvou řádků, tak k vyhodnocení toho, co má být uloženo jako změnová položka do tabulky `backlog`.

6.2.1.3.1 Zpracování typu `inttuple`

Aby bylo možné využít silných kontrol integrity relační databáze, je typ `inttuple` při ukládání zpracován zvláštním způsobem. Ukážeme si to na příkladu odkazu z tabulky Výkony (`effort`) do tabulky Řetězce (`label`). Uživatel chce při úpravách položky Výkon „zaškrtnout“ libovolný počet řetězců konkrétního druhu „Effort Keyword“ (viz kapitola 6.1.2).

Pro práci s databází je definován sloupec `keywords`, jehož atribut `bind` má hodnotu `'label'`. Je-li vyvolána metoda pro předložení dat k uložení do databáze `as_sql_output()`, jsou

všechny sloupce s definovaným atributem `bind` vráceny zvlášť. Volající metody `do_insert()`, `do_update()` či `do_delete()` ve třídě `SqlWrite` (viz dále) aktualizují sadu hodnot `keywords` ve speciální relaci s automaticky konstruovaným názvem `bindarray_<effort>_<keywords>_<label>_id`. Závorky značí, že slova v nich jsou dynamicky dodaná. Tato relace má dva prvky `effort_id` a `label_id`, což jsou *cizí klíče* (foreign keys) do obou tabulek příslušných.

Malá skupina triggerů a vložených procedur zajišťuje, že druh odkazovaného řetězce je skutečně „Effort Keyword“ a druh nebude později změněn. Díky tomu je automaticky zajištěna referenční integrita dat za jejich současné abstrakce vyšším datovým typem.

Opačný směr, tedy prezentace typu `inttuple` z databáze do webové šablony se provádí přímo pomocí schopností příkazu `SELECT` jazyka SQL pro spojování tabulek.

6.2.1.4 Třída SqlBase

Třída `SqlBase` je základní třídou, ze které vycházejí všechny datové objekty. Její inicializace přijímá objekt databázového adaptéru, identifikaci uživatele a instanci objektu `UserCan` (kapitola 6.2.1.1, strana 49) definující oprávnění uživatele. Odvozené datové třídy definují třídní proměnnou `TABLENAME` obsahující název relace (tabulky) v databázi. Ten tedy není nadále potřeba předávat jako parametr.

Základem je metoda `select_qry()`, jejíž argument je dotaz SQL, který musí začínat slovem `SELECT`. Výsledky dotazu se v napůl zpracovaném tvaru uloží do instance a je možné je obdržet dalšími metodami buď převedené do objektů `Row`: `as_rows()` nebo do asociativních polí: `as_dicts()`. Jediný řádek či hodnotu lze vynutit sadou metod `single_row()`, `single_dict()` a `single_int()`. Vráti-li databáze jiný počet řádků než jeden, dojde k výjimce.

Podstatná je metoda `select_detail_row()`, která na zadanou číselnou identifikaci řádku vrátí daný řádek z databáze. Ale zjistí-li předtím, že některý sloupec má atribut `bind`, přidá pro každý takový sloupec příkazu `SELECT` vazbu na jeho tabulku `bindarray_`. Viz popis zpracování typu `inttuple` výše. Výsledkem je kompletní objekt `Row`.

Zbývající metodou je `get_person()`, která vrací asociativní pole s údaji z tabulky Osoby (`person`), a to podle různých kritérií. Tuto metodu používají ostatní funkce k převodům mezi různými způsoby identifikace uživatele.

6.2.1.5 Třída SqlRead, dědí SqlBase

Třída sdružuje metody nemodifikující databázi. První z nich je `present_table()`, která podle zadaného dotazu SQL a seznamu definic sloupců vrátí tabulku ve tvaru vhodném pro prezentaci v šabloně tabulkového přehledu položek. Zároveň provede součty hodnot ve sloupcích označených `do_sum`. Metoda `get_table_last_touch()` zjistí z tabulky `backlog` informace o posledních změnách v aktuální tabulce zobrazované v tabulce pod ní.

Metodu `present_detail()` nepřímo používají šablony detailních formulářů jedné položky, přijímá jako argument číselnou identifikaci řádku. Vrací asociativní pole obsahující objekt `Row` převedený do tvaru vhodného k prezentaci a také seznam odkazů na tuto položku z jiných relací databáze. Ke zjištění odkazů se používá tzv. introspekčních schopností serveru PostgreSQL.

Poslední metodou je `add_checkerset()`, která podle parametrů přidá pro šablonu informace potřebné k zobrazení skupiny zaškrtávacích polí nebo přepínačů (radio buttons).

6.2.1.6 Třída SqlWrite, dědí SqlBase

Zde jsou sdruženy metody provádějící změny v databázi: `do_insert()` pro přidání řádku do databáze, `do_update()` pro jeho změnu a `do_delete()` pro odstranění. Všechny tři metody přijímají webový formulář jako asociativní pole (při mazání se pro jistotu kontroluje, jestli jde o stejná data, která jsou v databázi) a převádějí ho na objekt `Row`. Vracejí strukturu dat nutných pro zapsání události do tabulky `backlog`. Také kontrolují pomocí objektu `UserCan`, jestli má daný uživatel k příslušné operaci oprávnění.

Vyjma metody `do_insert()` načítají předchozí verzi řádku z databáze do objektu `Row` a oba objekty se porovnávají. Metodě `do_insert()` je možné volitelně vnutit číselnou identifikaci nového řádku, aby při vracení operace `DELETE` došlo k obnovení i této identifikace a zachování integrity dat.

K samotné úpravě databáze používají metody výhradně metodu `modif_qry()`, která omezuje přijímaný argument tak, že může začínat pouze slovy `INSERT`, `DELETE` nebo `UPDATE`. Omezené této metody a metody `select_qry()` popsané výše slouží zabezpečení před chybami programátora.

6.2.1.7 Datové třídy Labels, Efforts, Persons

Všechny tyto třídy z modulu `his_entries` dědí obě podpůrné třídy `SqlRead` a `SqlWrite`. Je-li to pro jednotlivé typy položek potřeba, konkretizují schopnosti výše popsaných metod děděných tříd.

Každá ze tříd definuje dvě sady definic sloupců `ColumnDef`. Jednu, kterou vrací metoda `detail_columns_def()`, definuje, jak je záznam uložen v databázi (a pracují s ní metody v děděných třídách) a druhou, zapsanou v konkrétní metodě `present_table()`, se popisuje, jak má vypadat přehledová tabulka složená z těchto položek. Obě definice jsou si podobné a další verze softwaru by je ideálně měly sjednotit.

Metoda `present_table()` každé třídy zkonstruuje konkrétní příkaz `SELECT` pro databázi, do kterého musí zakomponovat své parametry a předá ho stejnojmenné metodě z děděné třídy. Výsledky jsou volitelně rozšířeny o další údaje (u tabulky Výkony například o tabulku sloužícím k přechodu k přehledům ostatních zaměstnanců) a vráceny šabloně.

Metoda `present_detail()` obvykle na začátku volá stejnojmennou metody z děděné třídy. Výsledky je třeba rozšířit o údaje potřebné k tomu, aby uživatel mohl upravovat položku ve formuláři. Tedy například načíst seznam klíčových slov, na které se položka odkazuje apod.

6.2.1.8 Datová třída Backlog

Třída `Backlog` je svým způsobem metadatová třída a svou funkcí je na rozhraní modulů `his_backend` a `his_entries`. Nachází se v modulu vyšší úrovně, protože její metody pomocí argumentů `dataobj` přejímají instance objektů `Labels`, `Efforts`, `Persons`, aby s nimi mohly pracovat.

Její metoda `present_table()` také rozšiřuje údaje vrácené děděnou metodou. Jednou z přidaných informací je i výstup nápovědného algoritmu v metodě `_is_item_undoable()`, jehož výsledkem je barvení položek (viz kapitolu 6.1.7, strana 45).

Metoda `do_insert()` této třídy přijímá jako parametr strukturu, která byla výstupem metod `do_insert()`, `do_update()` a `do_delete()` výše popsaných datových tříd. Její část popisující změny hodnot je převedena do značkování `YAML` (textový popis datových struktur). Pak je pomocí děděné metody `do_insert()` nová položka vložena do tabulky `backlog`.

Specifickou metodou této třídy je operace pro vracení událostí, `undo()`. Přijímá seznam číselných identifikátorů položek v tabulce `backlog`, který přímo přejme z makra šablony pro výběr událostí k vrácení. Tyto položky jsou načteny z databáze a je-li vrácení možné, pomocí metod `_undo_insert()`, `_undo_update()` a `_undo_delete()`. Provádějí operace původního

datového objektu: první `do_delete()`, druhá `do_update()` a třetí `do_insert()`. Výsledky těchto operací jsou opět vloženy vlastní metodou `do_insert()` do tabulky `backlog`. To zajistí, že i operace vracení je vratná. Metoda `_undo_update()` musí ještě explicitně kontrolovat dodržování restrikce popsané v kapitole 6.1.7, strana 45. Na konci ještě metoda `undo()` aktualizuje vrácené položky v tabulce `backlog` tak, aby nemohly být znovu použity.

Všechny relevantní metody popsaných datových tříd na začátku pomocí objektu `UserCan` kontrolují, jestli má uživatel k příslušným operacím oprávnění. U datových tříd, kde je pro kontrolu oprávnění potřeba rozlišit položku vlastněnou aktuálním uživatelem (v současnosti jen třídy `Backlog` a `Efforts`), je k dispozici metoda `is_own_row()`, která pro parametr typu `Row` určí, jestli ji aktuální uživatel vlastní.

6.2.1.9 Adaptační třída HISZope

Třída `HISZope` je definovaná v modulu `his_zope` a její objekt je jako jediný z evidenčního systému HIS vkládán do struktury objektů v databázi `ZODB` pomocí rozhraní `ZMI` (viz kapitola 4.3, strana 18). Pouze přes něj mohou webové šablony interpretované systémem Zope přistupovat k výše popsaným třídám.

Metody třídy jsou tzv. dekorované bezpečnostními prvky Zope určujícími oprávnění uživatele potřebné k přístupu k nim. Zde není podrobné omezení aplikováno a přístup má kdokoli s oprávněním `permHISAccess`, standardně jakýkoli přihlášený uživatel. Výše bylo popsáno, že přesná omezení mají na starosti až datové třídy.

Metoda `navigation_links()` vrací seznam asociativních polí popisujících horní zelené záložky společně s potřebným oprávněním. Šablony při zobrazování vyhodnotí, kterou záložku kterému uživateli zobrazí.

Zbývající metody jsou vcelku triviální adaptéry pro funkcionalitu popisovanou výše. Pro operaci nad určitým typem tabulky je vytvořena instance příslušného datového objektu a s parametry vyvolána jeho prezentační nebo manipulační funkce. Výstup manipulační funkce je předán metodě `do_insert` objektu třídy `Backlog`.

6.2.2 Relační schéma

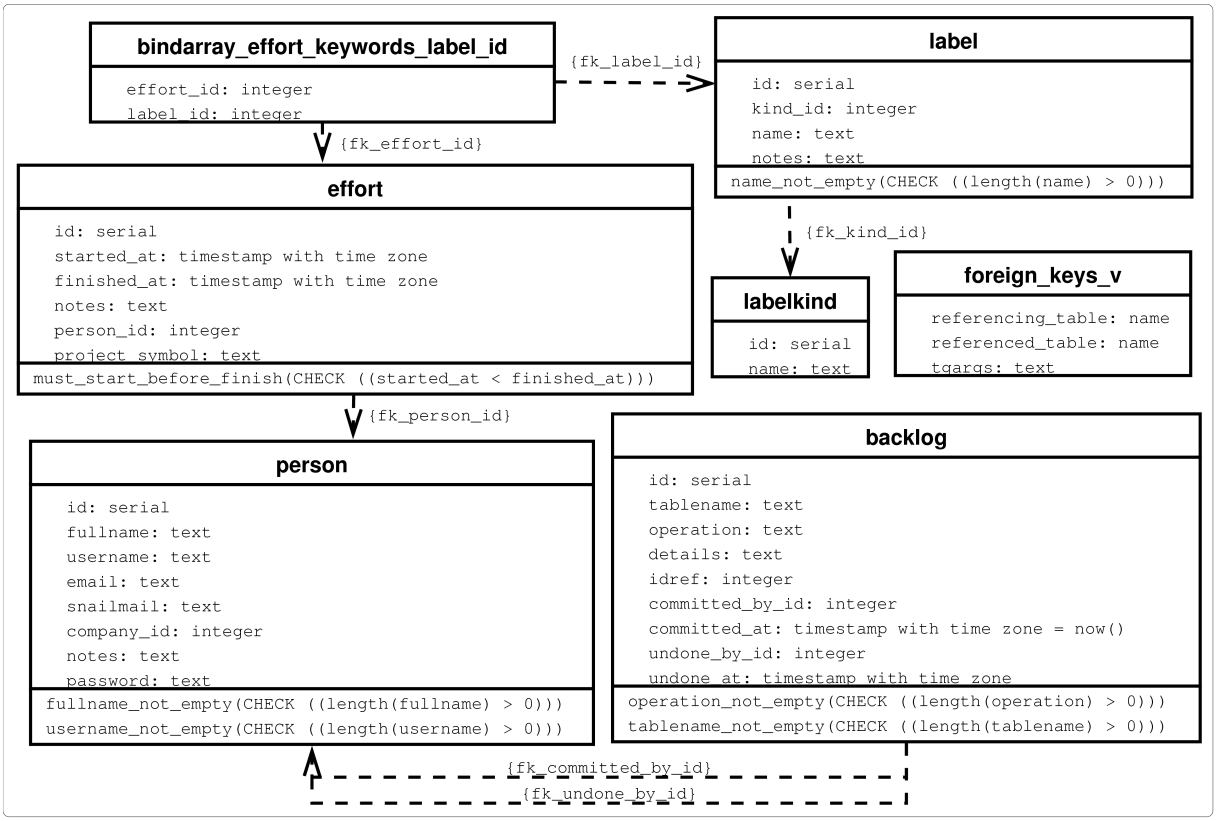
Aplikace HIS podle požadavků zadavatele využívá ukládání dat do relační databáze s přístupem pomocí jazyka SQL. Protože je aplikace implementována objektově, byl by patrně nejvhodnějším způsobem uložení dat objektový přístup. I podmínka relační databáze by se mohla s pohodlím splnit, pokud by mezi ní a aplikaci existoval adaptér. Ten by měl schéma

tabulek plně pod svou kontrolou. Nevýhodou je, že reprezentace dat nebyla příliš přehledná a tradiční. Relační databáze by zde hrála pouze roli jakéhosi „hloupého“ kontejneru.

Systém HIS je však postaven tak, aby co nejvíce využíval schopností datového úložiště (zejména kontrol referenční integrity) a zároveň navenek poskytoval přístup k datům na takové úrovní abstrakce, jaká je potřeba.

Díky tomu relační schéma ničím nepřekvapí a protože není stěžejním prvkem této práce, nebude zde ani podrobněji popisováno. Popis najde zájemce v adresáři `sql` na přiloženém CD. Jediný neobvyklý prvek, ukládání hodnoty typu pole za zachování referenční integrity, se popisuje v kapitole 6.2.1.3.1 na straně 52.

Diagram zjednodušeně zachycuje použité relace (tabulky), kontroly a cizí klíče (vztahy mezi tabulkami). Nepopisují se zde triggery ani vložené procedury. V textu výše zůstaly nezmíněny následující tabulky: `labelkind`, pomocí které se identifikují existující druhy řetězců a kterou upravuje programátor a databázový pohled (`VIEW`) `foreign_keys_v`. Ten se používá pro zjištění cizích klíčů za běhu za účelem zobrazení tabulky Odkazy (obr. 6.3 na str. 42).



obr. 6.8 Relační schéma

7 Testování

Plán testů použitelnosti rozhraní a testů funkcí programu.

7.1 Testování rozhraní aplikace

Výhoda testování použitelnosti této aplikace je ta, že jej budou denně používat osoby pracující v kontaktu s vývojářem. Moduly systému budou vznikat a nasazovat se postupně podle toho, kdy bude zadavatelem rozhodnuto o ukončení provozu starších aplikací. Zároveň bude probíhat doladění naživo u částí již nasazených. Produkt nebude zcela pálit mosty za koncepty aplikací, které se delší dobu ve firmě používají. Zvyklosti uživatelů pomůžou v adopci nového systému.

Z výše uvedených důvodů a z důvodů u zadavatele zaběhaných pořádků budou možnosti klasického monitorovacího testu použitelnosti (*testing room*) omezené. Po dokončení vývoje a odladění hlavních částí bude provedeno následující:

- Se systémem budou pracovat všechny tři skupiny uživatelů: řadoví zaměstnanci, vedoucí projektů i vedení společnosti. U těchto skupin jsou jednak nároky na různé funkce informačního systému a jednak různé úrovně lidského přístupu k užívání výpočetních systémů.
- Z těchto uživatelských skupin budou vybráni reprezentanti, budou navštíveni, nahrubo seznámeni s rozhraním a cílem systému. Na jejich počítačích dojde k testování částí systému, které se jich budou týkat. Rozhovor bude zaznamenán na diktafon, jsou očekávány prvotní dojmy a myšlenky bez rušivých vlivů.

- Hlavní zpětná vazba od uživatelů se očekává dlouhodobě. Značná výhoda systému je, že byl implementován ve vysokoúrovňovém prostředí. Z toho vyplývá, že jakákoli změna nebude vyžadovat velké úsilí.

Nyní máme zájem na zodpovězení například následujících otázek:

- Jaký vliv na výkon uživatele má vybrané webové prostředí?
- Jaký vliv na výkon uživatele má odezva webového serveru delší než se kterou se setkává u dosavadního systému.
- Bude způsob nabízený stránkou měsíčního přehledu výkonů dobře přijat a bude znamenat pokrok od systému současného?
- To samé platí o tlačítkových polích v editaci výkonu.
- Jaká je nejvhodnější velikost pole Poznámky (Notes) v každém formuláři. Nemělo by být příliš malé (aby se nezobrazovaly zbytečné posuvníky), ani příliš velké (aby nezaviňovalo časté posouvání celé stránky k ovládacím prvkům na spodní straně formuláře.
- Není možná jednotná cesta prohození pole Poznámky (Notes) a ovládacích prvků formuláře (tlačítka, bloky historie a odkazů, viz výše)?

Použitelnost aplikace se každopádně projeví až v reálném nasazení, nicméně pečlivým testováním předtím, než produkt do takové fáze projde, se předejde těm nejzásadnějším chybám a dopadům zavádějících myšlenek návrhu.

Náš uživatel musí být schopen rychle a efektivně používat aplikaci a její rozhraní. Nesmí pociťovat zdržení nezaviněné jím samým. Je třeba požadovat po mnohdy pasivních uživatelích, aby se během pravidelného používání systému domáhali zlepšení nejen funkčnosti, ale i uživatelského rozhraní.

7.2 Automatické testy

Každý programátor chybuje a i když každých napsaných několik řádků programu ihned testuje, nevyhne se u složitějších konstrukcí například tomu, že později prováděné změny zavlečou do programu nesprávné chování již dříve odladěné části.

Proto bude v budoucnu na dokončenou jednotku aplikován poloautomatický test typu *black box* (testování systému, jehož vnitřní fungování nemusí být známo). Systém bude napojen na

prázdnou testovací databázi se schématem, které má umět ovládat. Metody tříd budou na jednotném místě (co „nejblíž“ k uživateli) rozšířeny o protokolování svých vstupů a výstupů do souboru. Vývojář přímo prohlížečem otestuje všechny funkce systému včetně záměrných vyvolání chyb.

Díky protokolu bude moci následné sledování být automaticky prováděno z něj. Po každé změně programu se tedy bude moci ihned zjistit, jestli změna nepostihla oblast, která měla zůstat nedotčena.

8 Závěr

Vlastní zhodnocení práce a budoucnost systému.

8.1 Splněné cíle

- **Analyzovat a popsat požadavky zaměstnanců zadavatele na nový IS. Provést návrh skupiny propojených webových modulů s obsluhou různých uživatelských rolí.** Do části 3 včetně byly popisovány procesy ve firmě zadavatele a stávající řešení. Také byl z pohledu uživatelů podrobně popsán informační systém vycházející z dosavadních zkušeností a požadavků na změny u systémů stávajících, dnes již nevyhovujících.
- **Navrhnout jednotné uživatelské rozhraní s jednoduchým používáním a možností návrhu nových ovládacích prvků či postupů zvyšujících efektivitu.** Po teoretické části 5 popisující specifika webových aplikací a požadavky na dobré uživatelské rozhraní přišla kapitola 6.1 rozebírající nové rozhraní a ukazující mj. aplikaci zásad z předchozí kapitoly. Bylo implementováno i několik uživatelských stereotypů, které mají pomoci uživatelům s používáním systému.
- **Implementujte pilotní verzi vybrané části navrženého systému. Vyjděte z již fungujícího systému Plone. Důležitá data musejí být ukládána do RDBMS.** Částí 6 bylo popsáno jádro nového IS, které podporuje nakládání s obecnější datovou položkou a nabízí uživatelům komfortní funkce typu zobrazení křížových odkazů, historii položek či vracení operací, rychlé přechody mezi moduly.

Správce velice podrobně informuje e-mailem o všech chybách, ke kterým došlo, což zvyšuje rychlost a účinnost oprav.

Jako první byl podle požadavků vedoucího práce implementován modul *Evidence výkonů*. K tomu bylo zapotřebí zprovoznit i dva moduly, na kterých je tabulka výkonů závislá. Fungující program je založen na robusním databázovém systému PostgreSQL, který je využit nejen jako kontejner na data, ale uplatňuje i vlastnosti jako referenční integrita, triggery a vložené procedury. Cílem bylo, aby datové úložiště bylo v každém okamžiku v konzistentním stavu bez ohledu na to, že uživatel systém HIS a jeho kontroly obejde a jiným prostředkem přistoupí přímo k databázi.

Systém byl vytvořen v angličtině a jeho lokalizace do jiných jazyků (v současnosti pouze češtiny) se provádí až v prezentační vrstvě (webových šablonách). Je možné přepínání jazyka uživatelského rozhraní za běhu.

8.2 Co se zatím nepodařilo

Ač to nebylo v zadání přímo specifikováno, bylo očekáváno reálné nasazení prvního implementovaného modulu do reálné praxe. Toto bohužel oddálila skutečnost, že vývojáři systému Plone, na kterém program závisí v současné době přechází na jiný druh autentizačního mechanismu s názvem `PlonePAS`. Vyvíjený program měl rozpracovanou metodiku propojení své autentizace se systémem Plone, ale s příchodem nového mechanismu toto ztratilo smysl. Po adoptování `PlonePAS` bude moci být Evidence výkonů nasazena do reálného provozu.

Systém také dosud nebyl otestován větším počtem uživatelů. Problematika je však řešena kapitolou 7 a během nasazování do provozu a i po něm bude postupováno podle ní.

8.3 Zkušenosti s implementační platformou

Platforma Zope/Plone, která byla k implementaci systému HIS použita, není zcela obvyklá. Přesto se jedná o rozvinuté prostředí umožňující rychlé uvedení myšlenek do praxe (vyžaduje však předem získat nemalou míru orientace). Programátor se nemusí zaobírat podružnostmi a může se soustředit na podstatu problému. Uživatelská aplikace je přirozeným způsobem

rozdělena na vrstvy, což je v případě větších projektů naprostá nutnost. Skvěle v celém procesu pracuje i výkonný obecný programovací jazyk Python.

8.4 Budoucnost systému

Budoucnost implementačního jazyka Python, který se zdá být ve světě čím dál více populární, je neotřesitelná. Webová platforma Zope je také na poli webových serverů dosti používaná a má mnoho příznivců. Také nadstavba Plone prochází neustálým zkvalitňováním. Ztráta základů tedy novému systému nehrozí.

Interní procedury programu čeká další vývoj směrem ke zobecnění některých metod. Ideálně by se mělo podařit zobecnit databázový příkaz `SELECT` tak, aby nemusel být specifikován ručně, ale aby byl automaticky sestavován podle požadavku. Dobré by také bylo sjednotit dvojitou definici datových sloupců u každého typu tabulek do jedné. To souvisí s předchozí větou.

Co se týče rozvoje uživatelského rozhraní, bylo by například vhodné, aby dojde-li k chybě, byla tato zobrazena v jazyce uživatele. Zprávy o úspěšně provedených operacích zatím chybějí, mnozí uživatelé je ale vyžadují. Podrobnosti položek tabulky Vracení jsou příliš „programátorské“, další možností je zobrazovat je rozlámané do změněných sloupců v plovoucím rámci ve chvíli, kdy je kurzor na položce.

Podrobnější popis dosud neopravených chyb programu a plánů do budoucna se nachází v souboru `TODO` v kořenovém adresáři.

Otázkou je přijetí informačního systému u zadavatele, a to jak u zaměstnanců (jeho běžných uživatelů), tak i u vedení. Pokud byla tato diplomová práce zpracována dobře, jistě informace v ní obsažené pomůžou v tom, aby jak uživatelé, tak i budoucí vývojáři, systém lépe poznali.

9 Použité zdroje

[1] Dr. Jakob Nielsen on Usability and Web Design
http://www.useit.com/

[2] User Interface Design Tips, Techniques, and Principles
http://www.ambysoft.com/essays/userInterfaceDesign.html

[3] Intuitive equals familiar
http://www.asktog.com/papers/raskinintuit.html

[4] User Interface Design For Programmers by Joel Spolsky
http://www.joelonsoftware.com/uibook/fog0000000249.html

[5] Webová aplikace, článek na české Wikipedii
http://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace

[6] Python language website, český portál jazyka Python
http://www.python.org, http://www.py.cz

[8] Učebnice jazyka Python (aneb Létající cirkus)
http://sandbox.cz/~tuttle/python-tutorial-cz/tut

[9] První jazyk: Python
http://sandbox.cz/~tuttle/docs/prvni-jazyk-python.html

[10] Zope Book - oficiální dokumentace systému Zope
http://www.zope.org/Documentation/Books/ZopeBook

[11] Projekty Zope a Plone
http://www.zope.org, http://www.plone.org

10 Zdrojové soubory

10.1 Obsah přiloženého CD

`*.py` – zdrojové soubory aplikace,

`Extensions` – kód pro instalaci do systému Plone,

`skins` – webové šablony a podpůrné soubory pro web,

`i18n` – překladové soubory ve formátu `gettext`,

`sql` – výpis struktury databáze,

`doc` – text této práce v různých formátech včetně zdrojového.

10.2 Objem implementace

Pilotní verze systému HIS se skládá z

- cca 1 600 řádků obsahujících cca 68 000 znaků silně komentovaného kódu v jazyce Python,
- cca 2 150 řádků obsahujících cca 80 750 znaků webových šablon formátu XML, stylů CSS a programů v jazyce JavaScript,
- 141 lokalizačních řetězců jazykového katalogu `gettext`.

10.3 Postup instalace

V současnosti provozovaná verze IS používá verzi interpreteru jazyka Python 2.4.1, platformy Zope 2.9.6 a Plone 2.5.1. Ke správné funkci je třeba doinstalovat tyto moduly jazyka Python: `elementtree`, `mSQLmodule`, `PIL`, `psycopg2`, `reportlab`, `_xmlplus`, `yaml` a `zopyx`. Mezi produkty pro platformu Zope přidáme databázový adaptér ZpsycopgDA, pomocí kterého bude náš program komunikovat s databází. To vše se provádí standardními postupy a je zdokumentováno.

Systém HIS je plnohodnotný produkt pro systém Zope s názvem Hieronymus, instaluje se tedy nakopírováním do adresáře, kde Zope svoje produkty očekává a restartem Zope, pokud běží. Máme-li z rozhraní ZMI přidaný objekt `Plone Site`, přejdeme do jeho instalačního panelu sekce Instalace a vydáme příkaz k nainstalování produktu `Hieronymus`. Tím se registruje adresář webových šablon a kontaktní objekt `his_zope` se přidá do kořenové objektové složky portálu Plone.

Databázový systém PostgreSQL nainstalujeme standardním způsobem, vytvoříme specializovanou databázi a naplníme ji pomocí souboru SQL příkazů, který se nachází v adresáři `sql`. Z rozhraní ZMI přidáme objekt typu `Z Psycopg 2 Database Connection` do kořenové objektové složky portálu Plone. Nazveme ho `his_sql_connection` (povinné) a naplníme přihlašovacími údaji do databáze, kterou jsme vytvořili.

Je-li identifikátor objektu portálu Plone například `HIS`, přejdeme v prohlížeči na adresu `http://server/HIS/his_effort` a zobrazí se šablona tabulkového přehledu výkonů.