VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SADA PORTLETŮ PRO SPRÁVU ČASU

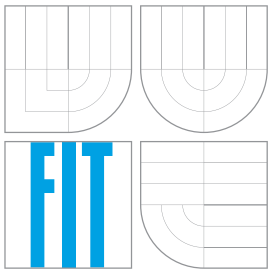
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN BASOVNÍK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SADA PORTLETŮ PRO SPRÁVU ČASU

PORTLETS BASED TIME MANAGEMENT TOOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTIN BASOVNÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK LETKO, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato diplomová práce se zabývá vývojem mini-aplikací tzv. portletů do prostředí portálů. Portály jsou alternativním typem webových informačních systémů. Portály umožňují dynamickou správu portletů, dokáží uživateli snadno přizpůsobit jejich obsah a jsou ideálním místem pro agregaci obsahu z různých datových zdrojů. Cílem této práce je vyzkoušet alternativní přístup kombinace několika technologií pro vytvoření sady portletů se zaměřením na správu času. Jedná se o kombinaci technologií portálů, portletů a tzv. portletových mostů. Tato práce je jedna z prvních demonstračních ukázek, jak se dají tyto technologie zkombinovat a bude použita firmou Red Hat jako výukový materiál pro vývoj portletů v jejich portále GateIn. Sada portletů pro správu času obsahuje správu uživatelských kalendářů, úkolů a kontaktů. V teoretické části práce se srovnávají existující aplikace pro správu času, zmiňují se standardy pro výměnu dat mezi nimi a následně se popisují všechny důležité technologie pro tuto práci. V praktické části je popisován návrh a implementace portletů a vyhodnocuje se vytvořený produkt. Vývoj popsaných portletů dokázal, že kombinace technologií portálů, portletů a portletových mostů je rychlý a efektivní způsob, jak vyvíjet informační systémy.

Abstract

This master's thesis focuses on development of mini-applications for portal environments called portlets. Portal is an alternative type of web information systems used to dynamic portlet management, content personalization and aggregation of content from different data sources. The aim of this thesis is to try an alternative combination of technologies of portals, portlets and portlet bridges. This thesis is one of the first demonstrations of the described combination of technologies and will be used as an educational material by Red Hat company for development of portlets on their portal GateIn. The set of portlets manages user calendars, tasks and contacts. The theoretical part of this thesis compares existing tools for time management and mentions standards for data exchange in this domain. Subsequently it describes all important technologies used in this thesis. The practical part describes the design and the implementation of portlets and evaluates results of this work. The development of the described set of portlets proved that combination of technologies of portals, portlets and portlet birdges is fast and appropriate way how to develop information systems.

Klíčová slova

Portál, portlet, portletový most, správa času, Getting Things Done, GateIn.

Keywords

Portal, portlet, portlet bridge, time management, Getting Things Done, GateIn.

Citace

Martin Basovník: Sada portletů pro správu času, diplomová práce, Brno, FIT VUT v Brně, 2013

Sada portletů pro správu času

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Letka, Ph.D.

.....
Martin Basovník
20. května 2013

Poděkování

Tímto děkuji Ing. Zdeňku Letkovi, Ph.D. za vedení a podporu při vzniku této diplomové práce, za poskytnutí konzultací, podnětných a cenných rad, bez nichž by projekt nemohl vzniknout. Dále bych chtěl poděkovat panu Mgr. Michalu Vančo z oddělení JBoss firmy Red Hat za podporu a důležité konzultace, které mi poskytl.

Obsah

Kapitola 1

Úvod

Ne nadarmo se říká, že čas jsou peníze. V dnešní uspěchané době, kdy jsme nuceni být někdy až na vteřiny dochvilní, si musíme efektivně řídit náš čas. Do domény řízení času bude víceméně patřit osobní i skupinová správa kalendářů, úkolů a kontaktů. Každý člověk by tuto problematiku měl řešit, pokud má zájem, aby nejenom stihl všechny svoje povinnosti, ale také aby mu zbyla trocha volného času na svoje záliby.

Tato práce se zabývá vývojem nástrojů (portletů) pro správu času v prostředí portálů. Portály jsou webové aplikace, které se skládají z miniaplikací tzv. portletů. Portály nám umožňují agregovat data z různých datových zdrojů, přizpůsobovat obsahy portletů pro jednotlivé uživatele a další věci, které z této technologie dělají velice zajímavou alternativu vývoje informačních systémů.

Cílem této práce je vyzkoušet alternativní přístup kombinace technologií portálů, portletů a portletových mostů. Kombinace těchto technologií je demonstrována na vývoji sady portletů podporujících správu času. Tato práce je jedna z prvních ukázek demonstrujících, jak se dají tyto technologie zkombinovat. Dokončená práce bude použita firmou Red Hat jako výukový materiál pro vývoj portletů v jejich portále GateIn.

Kapitole 2 popisuje problematiku správy času. Seznámí nás s pokročilými metodami, které se snaží tuto problematiku řešit a dozvíme se také něco o datových formátech, které se v této doméně využívají k výměně dat. Nakonec se podíváme na několik používaných nástrojů pro správu času a porovnáme si je z hlediska možnosti komunikace a několika dalších kritérií.

Kapitola 3 popisuje technologii portálů a portletů. Nejprve definuje důležité termíny a vysvětluje teorii vývoje aplikací v prostředí portálů. Následně navrhuje, jak je možné použít technologie v prostředí portálů, které portlety samy o sobě nepodporují. Může se jednat o některé technologie z platformy Java Enterprise Edition (dále jen Java EE).

Kapitola 4 definuje konkrétní požadavky na vyvíjenou sadu portletů pro správu času. Dále jsou tyto požadavky analyzovány a je vytvořen návrh architektury sady portletů včetně návrhu implementace důležitých případů užití.

V kapitole 5 se zaměříme na samotnou implementaci portletů. Důraz bude kladen zejména na řešení klíčových problémů a zajímavou funkcionalitu. Dále se zde podíváme na vytvořené grafické uživatelské rozhraní aplikace a popíšeme si ho.

Předposlední kapitola 6 porovnává vytvořenou sadu portletů s nástroji popsanými v kapitole 2. Následně vyhodnocuje uživatelské testování, které proběhlo, a nabízí možné rozšíření a úpravy implementovaných nástrojů pro efektivnější správu času.

Závěrečná kapitola shrnuje celou práci a rekapituluje důležité poznatky a přínosy práce.

Kapitola 2

Problematika správy času

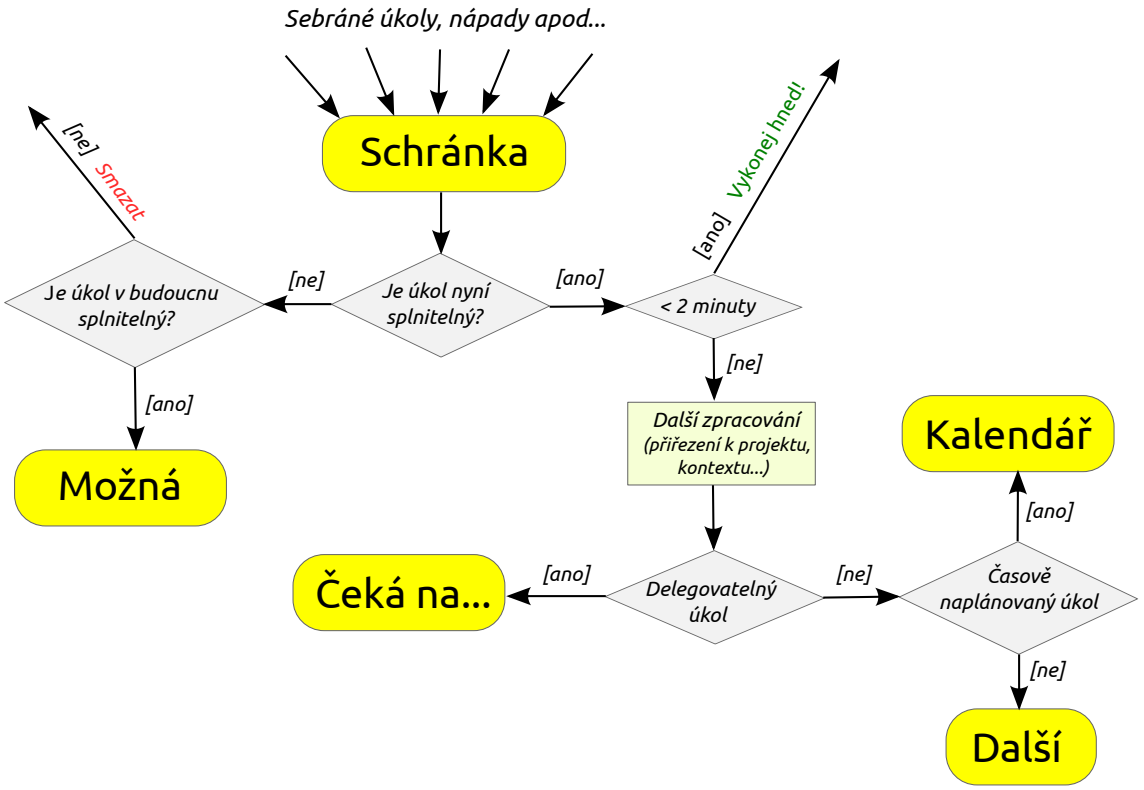
Efektivní správa času a úkolů není vůbec jednoduchá disciplína. Člověka jistě napadne, že by měl upřednostňovat důležité úkoly před nedůležitými a urgentní před těmi, co počkají. Pokud jsme si nahrubo naše úkoly subjektivně ohodnotili, tak jistě zjistíme, že některé aktivity se opakují v pravidelný čas a některé aktivity si můžeme časově naplánovat sami. Jak ale tyto aktivity efektivně vložit do našeho volného času, to je už větší problém. Nejdříve si musíme ujasnit, co do správy času vlastně patří. Určitě tam bude patřit správa osobního kalendáře, správa úkolů a správa kontaktů. Pro zmíněné potřeby existuje velké množství nástrojů. U nástrojů pro správu času bude velice důležitá přehlednost, jednoduchost a intuitivní grafické uživatelské rozhraní. V neposlední řadě budeme po našich nástrojích a aplikacích vyžadovat možnost exportu a importu uživatelských dat.

V této kapitole se nejprve podíváme na některé techniky správy času (sekce 2.1). Dále se zaměříme na nejpoužívanější nástroje pro správu času (sekce 2.3) a datové formáty, které tyto nástroje mohou používat k výměně dat (sekce 2.2).

2.1 Techniky správy času

Dnes již existuje velké množství technik pro správu času a v této sekci se podíváme na jednu z nich s názvem „Mít vše hotovo“. Čím dál více se také začínají kombinovat techniky správy času s různými filozofiemi, které člověka učí jak být efektivní a úspěšný ve všech aspektech jeho života. Tato sekce nám představí jednu takovou filozofii popsanou v knize „Sedm návyků skutečně efektivních lidí“ od Stephena R. Coveyho.

Mít vše hotovo (Getting Things Done) (GTD) [1, 23] je technika správy času, jejímž autorem je David Allen. Hlavní myšlenka spočívá v tom, že si vyčistíme hlavu od úkolů tím, že si je někam poznamenáme. Nyní se můžeme plně soustředit na náš úkol a nemusíme si nic pamatovat. Metoda vytváří několik různých seznamů pro zaznamenávání úkolů a proces správy úkolů se skládá z 5 kroků, a to *Sesbírej*, *Zpracuj*, *Zorganizuj*, *Zhodnoť* a *Udělej*. Na obrázku 2.1 je graficky znázorněné zpracování úkolů, které vysvětluje následující postup. První fáze obsahuje zaznamenávání všech úkolů, nápadů a poznámek do tzv. *Schránky*. Metoda předpokládá, že budeme schránku pravidelně vyprazdňovat. Ve druhé fázi zpracováváme úkoly ve schránce. Existují tři situace, které mohou nastat. První situace je taková, že rozhodneme, že daný úkol v tuto chvíli není splnitelný, nebo je už splněný. Úkol můžeme tedy buď vyhodit, nebo uložit do složky *Možná*, kam dáváme úkoly, které jsou určitým způsobem důležité nebo zajímavé. Mohou to být různé nápady apod. Pokud je úkol splnitelný do dvou minut, je vhodné ho vykonat hned. Smysl spočívá v tom, že pokud můžeme splnit



Obrázek 2.1: Zprácování úkolů technikou Mít vše hotovo (angl. Getting Things Done, GTD)

úkol tak rychle, tak by bylo časově nevýhodné ho dále zpracovávat. Delší úkoly nebudeme plnit hned, ale uložíme si záznam o něm efektivněji. Pokud je činnost úkolu rozložitelná na jednotlivé akce pro její splnění, může být vhodné z úkolu vytvořit projekt a z jednotlivých kroků úkoly. Můžeme také úkolům nastavit kontexty, které se týkají například místa, kde se dají splnit nebo nástroje, který ke splnění potřebujeme. Dále se u každého úkolu musíme rozhodnout, zda ho můžeme splnit pouze my, nebo zda ho můžeme na někoho delegovat. Pokud ho můžeme delegovat, je vhodné ho uložit do složky *Čeká na*. Pokud ho musíme vyřešit my, tak připadají v úvahu dvě úložiště. Do prvního úložiště s názvem *Další* ukládáme činnosti, které vykonáme, až budeme mít trochu času. V této složce se úkoly mohou řadit např. podle priority nebo projektu. Druhé úložiště je *Kalendář*. Sem umístíme úkol, pokud jeho vykonání lze časově naplánovat. David Allen také zdůrazňuje nutnost týdenní rekapitulace a zamyšlení nad svými úkoly. Jednou za týden by si člověk měl udělat v úkolech pořádek, zkontrolovat, zda pro každý projekt má nějakou činnost ve složce „Další“ a tím se vlastně ujistit, že na nic nemusí myslet a že systém funguje.

Sedm návyků skutečně efektivních lidí (The Seven Habits of Highly Effective People) [7] je velice známa kniha Stephena R. Coveyho, ve které filozoficky popisuje několik návyků, které nám pomohou být efektivní a úspěšní ve všech aspektech našeho života. Kniha klade důraz na to, abychom se oprostili od našich stávajících „paradigmat“, což mají být vlastnosti, které nejsou v souladu s férovostí, čestností, důstojností, trpělivostí, integritou apod. Bez těchto vlastností totiž podle Coveyho nemůže být člověk nikdy úspěšný ve všech oblastech života, na což právě Covey klade důraz. Dle metody bychom měli rozpoznat všechny naše životní role a snažit se být úspěšní ve všech. Covey definuje sedm návyků,

kterými bychom se měli řídit. První pravidlo „Buďte proaktivní“ nám říká, abychom si uvědomili, že za náš život neseme odpovědnost sami, abychom se k němu postavili čelem a nebáli se dělat správná rozhodnutí. Druhé pravidlo „Začínejte myšlenkou na konec“ zase říká, že bychom si měli srovnat naše priority, sny a cíle a podle toho se chovat. Význam třetího pravidla „To nejdůležitější dávejte na první místo“ je zřejmý. Čtvrté pravidlo „Myslete způsobem výhra – výhra“ nám chce říct, že bychom se měli snažit najít vždy řešení výhodné pro obě strany. Páté pravidlo „Nejdříve se snažte pochopit, potom být pochopeni“ také nepotřebuje další komentář. Šesté pravidlo „Vytvářejte synergii“ nám radí, abychom kombinovali sílu v našem týmu, díky níž budeme moci dosáhnout takových výsledků, jakých bychom při samostatné práci nikdy nedosáhli. Poslední pravidlo „Ošetřete pilu“ nám doporučuje, abychom si našli čas na regeneraci, doplnění sil, sebevzdělávání, četbu a další věci, které nám dodají energii a zvýší produktivitu v oblastech našeho zájmu.

Nyní už víme, že existují různé metody správy času, které můžeme lehce kombinovat i s filozofií. Bohužel ale není mnoho nástrojů, které by tyto metody uživateli vhodně nabízeli k použití. Podívejme se nyní na některé z nich a zaměřme se v tuto chvíli na lehce odlišnou vlastnost, a to vlastnost komunikace a sdílení dat. Tato vlastnost je velice důležitá, protože se v dnešní době klade stále větší důraz na různé standardizace firemních procesů a programovou kompatibilitu a s tím velice souvisí datové formáty. Podívejme se nyní na některé z nich.

2.2 Datové formáty pro reprezentaci událostí, úkolů a kontaktů

V úvodu kapitoly jsme zmínili, že jeden z požadavků na nástroje pro správu času je možnost komunikace a s ní spojená možnost exportu a importu dat. Právě touto problematikou se budeme zabývat v této sekci. Podíváme na formát *iCalendar* [51, 11, 14], který se standardně používá zejména pro textovou reprezentaci událostí a úkolů v kalendáři. Následně se podíváme na formát *vCard* [15, 40], který zase slouží k reprezentaci našich kontaktů. Lehce se u těchto formátů podíváme i na protokoly, které se využívají pro komunikaci klient – server.

Formát **iCalendar** je souborový formát, dle současného standardu RFC-5545 [11] schválený organizací Internet Engineering Task Force (IETF). Je to datový standard pro reprezentaci uživatelského kalendáře, který obsahuje zejména naplánované události a úkoly, ale i jiné typy objektů. Data se zpravidla ukládají do souboru s příponou `ical`, `ics`, `ifb` nebo `icalendar`. Formát používá ve standardu Multipurpose Internet Mail Extensions (MIME) označení typu obsahu `text/calendar` a kódování UTF-8 [56]. Každý řádek je ukončen znaky `CR+LF` (hex: 0D0A). Délka řádku je omezena na 75 bytů. Pokud potřebujeme řádek delší, je potřeba ho rozdělit a na novém řádku začít znakem mezera (hex:20) nebo tab (hex:09).

Na ukázce 2.1 je vidět, že prvním elementem kalendáře ve formátu iCalendar musí být element `BEGIN:VCALENDAR`. Tento element je párový a bude na konci dokumentu ukončen značkou `END:VCALENDAR`. Obsahu mezi těmito řádky se říká tělo (icalbody). Tělo může obsahovat vlastnosti a komponenty. Vlastnosti se vztahují k celému kalendáři a komponenty mohou reprezentovat jednotlivé události (`VEVENT`), úkoly (`VTODO`), žurnály (`VJOURNAL`), informace o volném a obsazeném čase (`VFREEBUSY`) apod. Žurnál dokáže připojit popis ke konkrétnímu datu. Může se použít k popisu denního výkazu činností, nebo k popisu postupu práce vzhledem k definovanému úkolu. Žurnál ovšem nezabírá oproti úkolům a udá-

```
1  BEGIN:VCALENDAR
2  VERSION:2.0
3  PRODID:-//xbasov00//calendar//CS
4  BEGIN:VEVENT
5  UID:20130501T080000Z-12345@dip-xbasov00.cz
6  DTSTAMP:20130501T080000Z
7  DTSTART:20130522T170000Z
8  DTEND:20130522T160000Z
9  SUMMARY:Termín odevzdání diplomové práce
10 END:VEVENT
11 END:VCALENDAR
```

Ukázka 2.1: Jednoduchý příklad kalendářového objektu událost ve formátu iCalendar.

lostem čas v kalendáři. Každá komponenta v kalendáři je tvořena několika kalendářovými vlastnostmi. Nyní si popíšeme strukturu formátu na naší ukázce. Na řádku 2 máme minimální potřebnou verzi formátu iCalendar. Řádek 3 určuje identifikaci produktu, řádek 4 a 10 obaluje objekt typu událost, který má na řádku 5 jeho globálně jedinečný identifikátor. K zajištění globální jedinečnosti rozdělujeme identifikátor na dvě části. V první části se zaznamenává datum a čas vytvoření objektu spolu s lokálním jednoznačným identifikátorem (např. identifikátor procesu). Tyto dvě položky bývají odděleny znakem '-'. Druhá část obsahuje identifikace domény, kde byl objekt vytvořen (může se použít i IP adresa). Části se zpravidla oddělují znakem '@'. Řádky 6, 7 a 8 vyjadřují postupně datum a čas, kdy byla instance události vytvořena a začátek a konec události. Použitý formát uvádí datum bez oddělovačů, dále čas bez oddělovačů uvozen znakem 'T' a poslední znak 'Z' značí, že se jedná o koordinovaný světový čas (Coordinated Universal Time, UTC), kterému se také někdy říká *Zulu time*. Řádek 9 pak značí popis události.

Omezením formátu iCalendar je neschopnost zakódovat informaci o způsobu nakládání s daty a neschopnost některých výrobců podporovat pokročilejší možnosti jako záznam žurnálů a úkolů. Formát je navíc kompatibilní pouze s gregoriánským kalendářem. Dnes již existují různá rozšíření formátu iCalendar (např. RFC-5546[1] [8]). Podpora těchto rozšíření ze strany výrobců je ovšem zatím velmi malá.

Export a import dat ve formátu iCalendar je podporovaný velkým množstvím aplikací, jako například Google kalendář, Apple kalendář, Microsoft Outlook, Yahoo kalendář, Lightning rozšíření pro Mozilla Thunderbird a další...

Datový formát **vCard** se používá jako standardní textový formát pro elektronické vizitky. V současné době se postupně opouští verze 2.1 kvůli novější verzi 3 podle RFC-2426 [15]. Existuje již verze 4 podle RFC-6350 [40], ale její podpora ze strany aplikací je zatím malá. Elektronické vizitky vCard mohou ve zmíněných verzích obsahovat jméno, adresu, telefonní čísla, emailové adresy, lokátory zdrojů (Uniform Resource Locator - URL [4]), loga, fotografie, kategorie (tagy), audio klipy, datum narození, geo polohu, poznámky a další atributy. Verze 3.0 se od verze 2.1 liší ve výskytu spíše méně používaných atributů jako třída (modifikátor pro zajištění přístupu) nebo prodid (identifikátor tvůrce kontaktu). Verze 4.0 ruší některé atributy z verze 3.0 jako třeba třída nebo mailer (typ emailového klienta), ale přidává několik dalších. Mezi nové atributy patří pohlaví, druh (např. jednotlivec nebo

[1] RFC-5546 specifikuje protokol, který používá objekt iCalendar k zajištění spolupráce mezi různými kalendářovými systémy.

```
BEGIN:VCARD
VERSION:3.0
N:Martin;Basovník
FN:Martin Basovník
ORG:Basovnik s.r.o.
TITLE:Manager
PHOTO;VALUE=URL;TYPE=GIF:http://www.basovnik.cz/photos/my_photo.gif
TEL;TYPE=WORK,VOICE:689 162 485
TEL;TYPE=HOME,VOICE:737 565 919
ADR;TYPE=WORK:293;;Padělek;Jimramov;;592 42;ČR
LABEL;TYPE=WORK:Padělek 293, 592 42 Jimramov, ČR
ADR;TYPE=HOME:40;;Purkyňova;Brno;;612 00;ČR
LABEL;TYPE=HOME:Purkyňova 40, 612 00 Brno, ČR
EMAIL;TYPE=PREF,INTERNET:xbasov00@stud.fit.vutbr.cz
REV:2013-05-16T22:04:00Z
END:VCARD
```

Ukázka 2.2: Jednoduchý příklad kontaktu ve formátu vCard.

organizace), jazyk, kterým kontakt hovoří, členství ve skupině (skupina je definovaná atributem druh), relace na jiné kontakty, XML data připojené k vizitce apod. Dále existuje spousta rozšiřujících atributů, které ale nemají své pevné místo ve specifikacích. Názvy těchto atributů jsou uvozeny znaky „X-“. Příkladem mohou být atributy `X-ICQ`, `X-JABBER`, `X-SKYPE` se zřejmým významem.

Na ukázce 2.2 můžeme vidět příklad vizitky uložené ve formátu vCard. Vizitka ve formátu vCard musí být obalena párovým elementem `VCARD`. Startovací značky párových elementů jsou uvozeny prefixem `BEGIN:` a ukončovací značky zase prefixem `END:`. Atributy na ukázce od shora dolů postupně znamenají verzi normy vCard, strukturovanou reprezentaci jména, formátovaný název (zobrazovaný), název organizace, funkční pozice v organizaci, fotografii, telefonní číslo do práce a domů, fyzické doručovací adresy do práce a domů a popisy, které se mají použít, následuje emailová adresa a datum poslední změny.

Protokol **CalDAV** [6] (Calendaring Extensions to WebDAV) dokáže vytvořit obálku pro naše data ve formátu iCalendar a slouží pro výměnu dat mezi klientem a serverem například pro potřeby synchronizace událostí v kalendářích. Protokol je v principu rozšířený *Hypertext Transfer Protokol* (HTTP) a umožňuje zasílat na server požadavky jako například vložení události, smazání události, získání událostí v určitém časovém období a podobně. Samotná data jsou pak již ve formátu iCalendar, jak už bylo zmíněno. Aby klient mohl tuto funkcionalitu na serveru použít, tak je potřeba, aby měl webový server podporu pro kalendářní funkce, což není úplně běžné.

Podobně jako protokol CalDAV pro komunikaci klient – serverem využívající datový formát iCalendar, existuje protokol **CardDAV** [9], který pracuje obdobně, ale s datovým formátem vCard. Podpora CardDav na webových serverech i na současných aplikacích pro správu kontaktů je ale momentálně velmi malá.

2.3 Softwarové nástroje pro správu času

V následujících pár odstavcích se zaměříme na softwarové nástroje pro správu času. Nástrojů pro správu času existuje celá řada. Definujme si tedy několik parametrů a kritérií, pomocí kterých bychom se mohli pokusit tyto nástroje porovnat. Jedno ze základních dělení těchto nástrojů může být na nástroje pro osobní použití a pro korporátní použití. Nástroje pro korporátní použití mají na rozdíl od osobních význam ve firmách, kde je existuje na sdílení určitých dat. Tyto nástroje pak slouží pro kolektivní práci na společných projektech a občas se tomuto typu softwaru říká *groupware* [54]. Pomocí nich jednotliví členové skupiny mohou komunikovat, sdílet různé dokumenty a organizovat svoji spolupráci. Do groupwaru patří elektronická pošta, diskusní fóra, chaty, kalendáře, úkoly, úložiště dokumentů apod. Nástroje mohou být dále buď ve formě tzv. *standalone* aplikace nebo *webové služby* [53]. Standalone aplikace se může být nainstalována na počítač uživatele a lze s ní pracovat bez připojení k internetu. Často bývá přítomna ještě serverová část pro zmíněné sdílení dat. Oproti tomu webová služba je aplikace přístupná pouze z Internetu, případně intranetu. Výhodou webových služeb může být to, že s nimi můžeme pracovat odkudkoliv, nevýhodou zase nutnost připojení k síti. Mezi další kritéria může patřit podpora konkrétních formátů pro export a import dat. Zaměříme se tedy dále na formáty, které jsme si popsali v předchozí kapitole. Toto kritérium považuji za důležité, protože kompatibilita dnešních nástrojů a využívání standardů je jistě základ kvalitních aplikací. Neméně významné kritérium bude jistě i cena nástroje. V následujících odstavcích se podíváme na nástroje Microsoft Outlook, IBM Lotus Notes, Mozilla Sunbird a Google Apps.

Microsoft Outlook [52] je standalone groupware od firmy Microsoft, který kromě služby emailového klienta nabízí také služby správy kalendářů, správy úkolů a kontaktů. Sdílení dat aplikaci zajišťuje *Microsoft Exchange Server*. Zaměřme se nyní na verzi 2007 tohoto programu, protože je v tuto chvíli stále ještě nepoužívanější. Kalendář podporuje import a export událostí mimo jiné do formátů iCalendar 2.0 a *vCalendar*[2] ve verzi 1.0 [46] a dalších různých CSV a textových souborů. Zajímavá vlastnost je, že umožňuje exportovat události jen v určitém časovém období a s určitou podrobností. Co se týče úkolů, tak Outlook nepodporuje export do formátu iCalendar. Outlook v dané verzi nepodporuje u formátu iCalendar ani typy objektů úkol a žurnál. Outlook dále umožňuje exportovat a importovat kontakty mimo jiné do formátu vCard verze 2.1. V souhrnu Outlook plně nepodporuje formáty iCalendar a vCard verze 3.0. Formát vCard verze 2.1 sice podporuje, ale neumožňuje uložit více vizitek do jednoho souboru. Outlook bývá dále kritizován za to, že si vytváří četné vlastní volitelné atributy pro zmíněné formáty v kapitole 2.2.

IBM Lotus Notes je také standalone groupware, nyní od firmy IBM, který slouží pro správu kalendářů, úkolů a kontaktů. Serverovou aplikací je zde *IBM Lotus Domino*. Aplikace má podobnou funkcionalitu jako Microsoft Outlook, ale má lepší podporu exportních a importních formátů. Nejenže podporuje formáty iCalendar a vCard, ale navíc podporuje i protokoly pro komunikaci nad těmito formáty, a to CalDAV a CardDAV.

Mozilla Sunbird je standalone groupware neziskové organizace Mozilla Foundation. Oproti výše zmíněným aplikacím nemá podporu pro správu kontaktů. Další nevýhodou může být absence vytvoření skupiny úkolů. Nicméně podporu formátů iCalendar i CalDAV nástroj má.

Google Apps je webová služba firmy Google, Inc., která obsahuje balíček nejrůznějších aplikací. Mezi tyto aplikace patří například aplikace *Google Calendar*, což je kalendář, který mimo standardní funkce nabízí možnost sdílení kalendářů a nejrůznější synchronizace s ji-

[2] Formát vCalendar byl použit jako základ pro vytvoření formátu iCalendar.

	Microsoft Outlook	**IBM Lotus Notes**	**Mozilla Sunbird**	**Google Apps**
Typ softwaru	standalone	standalone	standalone	webová služba
Verze	2007	8.5.3	1.0beta1	ke dni 18.12.2012
Zdarma	ne	ne	ano	ano
iCalendar	ano	ano	ano	ano
CalDAV	ne	ano	ano	ano
vCard	ano	ano	ne	ano
CardDAV	ne	ano	ne	ano

Tabulka 2.1: Porovnání softwarových nástrojů pro správu času

nými službami. Kalendář umožňuje v současné době export událostí do formátu iCalendar verze 2.0 a import událostí ve formátu iCal a formátu CSV od aplikací jako Microsoft Outlook, Yahoo Mail, Hotmail apod... Google Calendar mimo jiné také nabízí serverovou službu CalDAV, ovšem s omezenou funkcionalitou [20]. Další aplikací v balíčku Google Apps je aplikace *Gmail*, která kromě funkcionality emailového klienta nabízí také správu kontaktů a jednoduchou správu úkolů. Správa kontaktů podporuje export do formátu vCard 3.0 a CSV souboru (Gmail nebo Outlook). Podpora serverové služby CardDAV je zatím velmi omezena. Aplikace Gmail obsahuje také správu úkolů, ale má dle mého názoru velké slabiny a z hlediska funkcionality je prakticky nepoužitelná. Standardní funkci pro export a import úkolů momentálně nenabízí, ale lze použít externí aplikaci, například *Google Tasks Porter*, která pracuje s Google API a po přidělení oprávnění umí získat úkoly v několika formátech.

Přehledné shrnutí s výsledky porovnání se nalézá v tabulce 2.1. Všechny porovnávané nástroje můžeme považovat za groupware a tuto informaci tedy tabulka z důvodu přehlednosti neobsahuje. Můžeme konstatovat, že dle zvolených kritérii vypadá nejlépe nástroj IBM Lotus Notes hlavně díky dobré podpoře datových formátů. Google kalendář, jako webová služba, dopadl také velmi dobře. Naopak Microsoft Outlook má podporu formátu slabou a Mozilla Thunderbird má zase omezenou funkcionalitu (chybí správa kontaktů).

Kapitola 3

Technologie portálů a portletů

Portál [32, 33, 47, 27, 49] je webová aplikace, jejíž klíčovou vlastností je obsahová agregace z různých datových zdrojů. Tato aplikace se skládá z miniaplikací tzv. *portletů*, které můžeme do portálu dynamicky přidávat a zase odebírat. Technologie portálů se často využívá při vývoji informačních systémů, a to v situacích, kdy nám jde o integraci dat z více systémů a možnost personalizovat obsah pro každého uživatele.

Portlet [32, 33, 47, 27, 49] je miniaplikací, která zprostředkovává určitou část obsahu, která se má zobrazit na portálové stránce. Jednotlivé portlety se používají v portálech jako malé zásuvné moduly, které můžeme libovolně přidávat a zase odebírat. Kombinace portletů nám vytváří prezentační vrstvu portálu. Obsah generovaný portletem se nazývá *fragment* [32, 33], který je zpravidla tvořen značkovacím jazykem *eXtensible HyperText Markup Language* (XHTML) [30]. Uživatelé komunikují s portlety dle známého paradigmatu *požadavek/odpověď*, kde tvůrcem požadavků je výhradně uživatel. Životní cyklus a kompletní správu portletů včetně generování dynamického obsahu zajišťuje tzv. *portletový kontejner*.

Portletový kontejner [32, 33, 47] je aplikace, která běží na aplikačním serveru, spouští a spravuje portlety a zprostředkovává jim komunikaci s uživatelem. Portletový kontejner také samotné portlety obsahuje, řídí jejich životní cyklus a mimo jiné poskytuje úložiště pro uživatelská nastavení portletů. Portletový kontejner obdržuje uživatelské požadavky z portálu a příslušným způsobem je distribuuje dál portletům. Agregaci obsahových fragmentů, získaných od jetnotlivých portletů, zajišťuje právě portletový kontejner.

V případě, že se rozhodneme využít portálové řešení, můžeme řešit problém, že naše stávající webové aplikace na portál nepůjdou vložit a budeme je muset vytvořit znovu ve formě portletů. Dobrá zpráva je, že to nemusí být pravda. V tuto chvíli můžeme použít tzv. *portletový most* [18, 19, 47]. Portletové mosty dokáží mapovat naši aplikaci na portlet a díky tomu nemusíme naši aplikaci psát znovu. Příkladem takového portletového mostu může být most pro podporu technologie *Java Server Faces* [12, 13], která slouží k oddělení grafického uživatelského rozhraní od aplikační logiky. Portletový most se v vyplatí použít i když vytváříme aplikaci od začátku. Například technologie JSF nám totiž nabídne určitě lepší vyjadřovací prostředky než samotná technologie portletů.

Následující sekce 3.1 podrobněji rozebírá technologii portálů. Sekce 3.2 je zaměřena na technologii portletů a na ni navazuje sekce 3.3, kde se zaměříme na portletový životní cyklus. V dalších dvou sekcích se podíváme na alternativní možnosti vytváření portletů. V sekci 3.4 se podíváme na to, jak můžeme v našem portletu využít portletový most a v sekci 3.5 se podíváme na možnost vytváření portletů ve frameworku *Spring MVC Portlet*. Poslední sekce 3.6 bude definovat další nástroje a technologie důležité pro tuto práci.

3.1 Portály

Portály jsou webové aplikace, které se v dnešní době čím dál více začínají používat jako podnikové informační systémy. Jejich hlavní výhoda spočívá v možnosti agregace obsahu z různých datových zdrojů. Portály se nasazují na webový aplikační server a obsahují portletový kontejner, který spravuje všechny portlety, které se na portále vyskytují. Tyto portlety můžeme dynamicky přidávat a zase odebírat. Často využíváme portály v situacích, kdy chceme integrovat data z několika různých systémů na jednom místě. Klíčová data z jednotlivých systémů pak mohou být reprezentována jedním nebo více portlety. Důležitý rozdíl portálů a klasických webových aplikací je ten, že jeden HTTP požadavek může být v prostředí portálů obsloužen více aplikacemi (portlety), na rozdíl od klasických webových aplikací, kde jeden požadavek obslouží pouze jedna aplikace. O zpracování požadavků se více dozvíme v sekci 3.3.

Mezi další klíčové vlastnosti portálu patří autentizační mechanismus a personalizace obsahu. Přístup na portál zabezpečuje autentizační mechanismus a každý přistupující uživatel se musí prokázat platnými přístupovými údaji. Pokud má přihlášený uživatel do portálu dostatečné oprávnění, může si dále upravit vzhled portálových stránek, zejména množiny portletů, které budou nabízet. U jednotlivých portletů si pak uživatel může přizpůsobit vzhled nebo informace, které portlet zobrazuje. Podmínkou je samozřejmě podpora tohoto nastavení daným portletem.

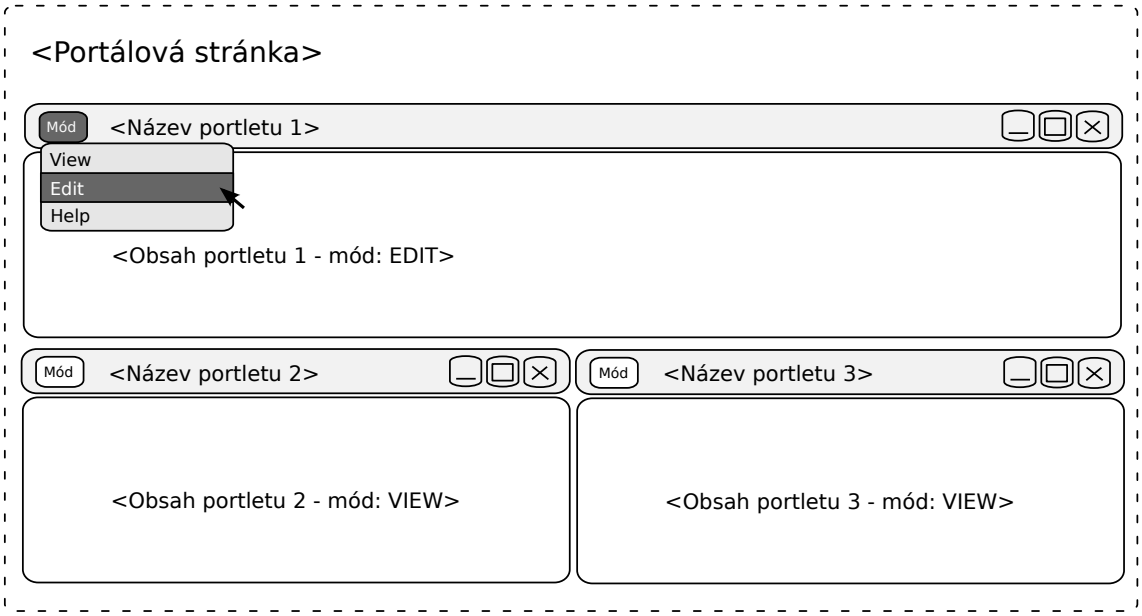
Kromě základních vlastností mívají konkrétní implementace portálů i různá svoje rozšíření. Všechny portály by ale měly implementovat rozhraní pro integraci portletů. Portlety jsou pak přenositelné mezi jednotlivými implementacemi portálů. Pokud programátor využije nadstandardní vlastnost některého portálu, musí tuto vlastnost ošetřit, aby nezpůsobovala problém při přenosu na jinou portálovou implementaci. Příkladem implementací portálů může být Liferay portál od společnosti Liferay, IBM WebSphere Portal od firmy IBM nebo GateIn portál od firmy Red Hat. GateIn portál bude výchozí implementace portálu použitá v této práci.

3.1.1 GateIn portál

GateIn portál [25] je volně dostupný portálový server od firmy Red Hat. GateIn je možné použít s libovolnou implementací aplikačního serveru jazyka Java. Implicitně se ale nabízí v kombinaci s aplikačním serverem *JBoss AS* od firmy Red Hat. Projekt GateIn vznikl spojením dvou projektů – *JBoss Portal* a *eXo Portal*. Nyní je dostupný jako samostatný produkt.

Základní vlastností tohoto portálu je samozřejmě podpora portletů verze 1.0 i 2.0, které budou popsány v následující sekci. Mezi zajímavé vlastnosti patří podpora portletového mostu pro technologii *Java Server Faces* (JSF), o které se dozvíme více v sekci 3.4.1. Tento portletový most navíc obsahuje i rozšíření technologie JSF o podporu frameworku *RichFaces*, o kterém si řekneme více v sekci 3.4.1 a 5.3. GateIn dále podporuje uživatelské privátní stránky tzv. nástěnky, podporuje přizpůsobení vzhledu portálu a portletů, má jednoduchou a intuitivní administraci a v kombinaci s dalšími neuvedenými vlastnosti vytváří velice kvalitní a použitelnou implementaci portálu.

Následující sekce bude pojednávat o zmíněných miniaplikacích, tedy portletech, které portály obsahují.



Obrázek 3.1: Příklad struktury portálové stránky.

3.2 Portlety

Momentálně existují dvě verze specifikace pro portlety. Tyto secifikace spravuje organizace *Community Process*. Starší verze portletů 1.0 definovaná normou *Java Specification Request 168* (JSR-168) [32] z roku 2003 byla rozšířena novější verzí 2.0 podle normy *JSR-286* [33] v roce 2008. Tato novější verze je zpětně kompatibilní se starší verzí. Portletové API verze 1.0 je založené na platformě *Java 2 Enterprise Edition* (J2EE) verze 1.3 [48] a verze 2.0 na pozdější verzi 1.4 [31]. V následujícím textu popíšu několik portletových vlastností. Zmíněné vlastnosti platí pro obě verze normy, pokud nebude upřesněno jinak.

Portlety generují obsah (fragment) nejčastěji ve značkovacím jazyce HTML. Portál standardně přidává k fragmentům generovaným portlety název portletu, ovládací tlačítka a další dekorátory, které vytváří dojem určitého okna, které vzhledově připomíná okna v grafických uživatelských rozhraních moderních operačních systémů.

Portletové módy umožňují definovat různé možnosti chování portletu. Módy říkají portletu, jaké úkoly má vykonávat a jaký obsah má generovat. Portletové módy se mohou samozřejmě programově měnit. Norma definuje tři základní módy. Mód `VIEW` zobrazuje standardní obsah, který je pro uživatelé důležitý. Mód `EDIT` slouží k přizpůsobení obsahu a často nabízí různé formuláře, kde uživatel nastavuje vzhled a přizpůsobuje si obsah, který se mu zobrazí v módu `VIEW`. Mód `HELP` nabízí informace o daném portletu a vysvětluje jeho chování. Všechny tyto módy jsou definované ve třídě `javax.portlet.PortletMode` a je možné si nadefinovat i svoje vlastní módy.

Stavy oken definují prostor, který bude portlet na dané portálové stránce zabírat. Tato informace souvisí i s množstvím informací a způsobem zobrazení informací na stránce. Základním stavem je stav `NORMAL`. Tento stav zabírá průměrnou velikost na portálové stránce vzhledem k ostatním portletům. Pokud je pro nás daný portlet momentálně důležitý, můžeme mu přiřadit stav `MAXIMIZED`. Nyní bude portlet zabírat buď celou portálovou stránku nebo její většinový obsah. Portletu, který momentálně nebudeme potřebovat,

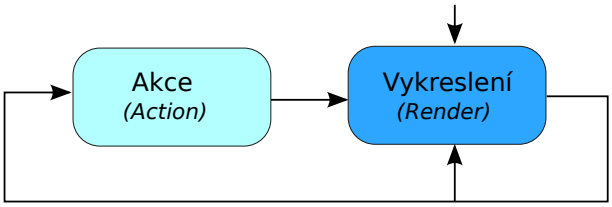
můžeme nastavit stav `MINIMIZED`. To způsobí že, se bude vykreslovat pouze minimální obsah nebo se nevykreslí vůbec. Stavy oken se definují ve třídě `javax.portlet.WindowState` a podobně jako u módů si můžeme nadefinovat svoje vlastní stavy.

Na obrázku 3.1 je uveden příklad, jak může vypadat portálová stránka. Jednotlivé portlety mají v hlavním panelu název, menu, kde je možné měnit mód zobrazení portletu a tlačítka pro změnu stavu okna. V aktuálním stavu se všechny portlety nachází ve stavu okna *NORMAL*. Tyto portlety může na portál dynamicky vkládat uživatel v roli administrátora portálu, pokud byly vloženy do příslušné složky na aplikačním serveru. Archívy reprezentující celou portletovou aplikaci mají příponu `war`. Na jedné portálové stránce se může vyskytovat stejný portlet i vícekrát. Každý bude pak pracovat nezávisle na ostatních.

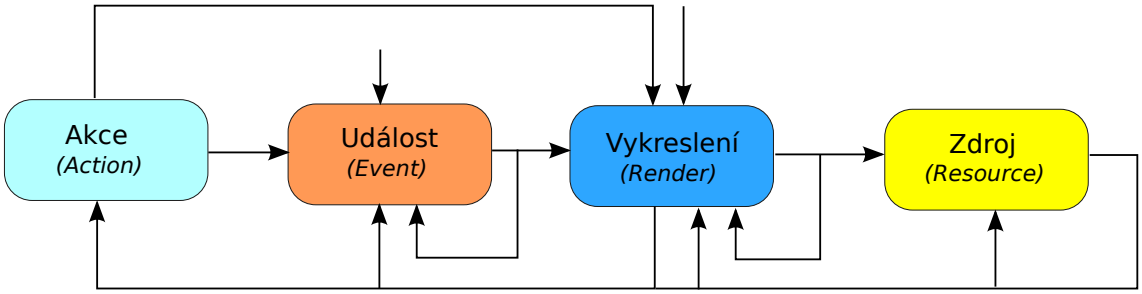
Komunikace s portletem probíhá na bázi požadavků. Existuje několik **typů požadavků** na portlet. Základní dva typy požadavků jsou požadavek na *vykreslení* (render) a požadavek *akce*. Od verze 2.0 obsahuje norma ještě požadavky na *zdroj* (resource) a *událost* (event). Při požadavku na vykresení portlet generuje obsah na základě svého interního současného stavu. Dalším typem požadavku je akce. Tento požadavek má za následek změnu stavu portletu, kam může patřit změna módu a stavu okna portletu nebo třeba operace s databází. Požadavek na zdroj umožňuje vyžádání dynamických zdrojů s rozšířenou funkcionalitou. Může to být požadavek na vrácení dynamicky vytvořeného obrázku nebo PDF souboru. Díky tomu, že požadavek prochází přes portletový kontejner, tak může využívat celého kontextové prostředí (informace o portálovém serveru jako např. název, podporované módy a stavy oken a další informace, které jsou například nastaveny v konfiguračním souboru portálu) kontejneru a můžeme lépe zajistit zabezpečení zdroje. Změny portletu tímto požadavkem nemohou ovlivnit vykreslení jiných portletů narozdíl třeba od požadavku na akci. U portletů verze 1.0 bylo možné tento požadavek na zdroj obejít pouze pomocí *servletů* [39]. Servlet je jednoduše řečeno komponenta spuštěná na webovém serveru, spravuje ji webový kontejner a umí obsluhovat klientské HTTP požadavky. Pro přijaté požadavky umí servlety generovat HTTP odpovědi. Pokud použijeme pro zpracování požadavku servlet místo portletu, ztratíme tak informace dostupné z portletového kontejneru a vlastně veškerý portletový kontext. Posledním typem požadavku je *událost*. Tento typ požadavku je novým typem komunikace mezi portlety od verze 2.0. Komunikace na bázi zasílání událostí bude popsána později.

Základní teoretický přehled o portletech máme probraný a můžeme se blíže podívat na implementační detaily. Tyto implementační detaily nám spolu s teorií pomohou lépe pochopit danou technologii. Nejprve se podíváme na rozhraní, které portlety musí implementovat a dále na nejpoužívanější třídu, která již zmíněné rozhraní implementuje a používá se jako rodičovská třída pro vyvíjené portlety.

Portlety implementují rozhraní `javax.portlet.Portlet`. Toto rozhraní deklaruje čtyři základní metody. Metoda `init` je inicializační metoda portletu, která slouží k připravení časově náročnějších operací a případnému nastavení počátečního stavu portletu. Metoda obsahuje jediný parametr typu `PortletConfig`, který je zprostředkován portletovým kontejnerem a mimo jiné obsahuje informace z hlavního konfiguračního souboru portletu, který má vždy název *portlet.xml*. Metoda `proccessAction` zpravovává požadavky typu akce. První parametr třídy `ActionRequest` obsahuje vstupní parametry pro akci a druhý parametr typu `ActionResponse` obsahuje odpověď na akci, která může obsahovat změnu velikosti okna, módu porletu, přesměrování na jinou URL nebo požadavek na perzistentní uložení vstupních parametrů apod... Metoda `render` obsluhuje požadavek na vykreslení portletu. V prvním parametru typu `RenderRequest` je obsažena instance portletu a případně některá další vstupní data. Metoda `destroy` slouží k uložení stavu portletu a k uvolnění instance



Obrázek 3.2: Automat znázorňuje možnou sekvenci jednotlivých portletových požadavků a příslušných fází v portletech verze **1.0**.



Obrázek 3.3: Automat znázorňuje možnou sekvenci jednotlivých portletových požadavků a příslušných fází v portletech verze **2.0**.

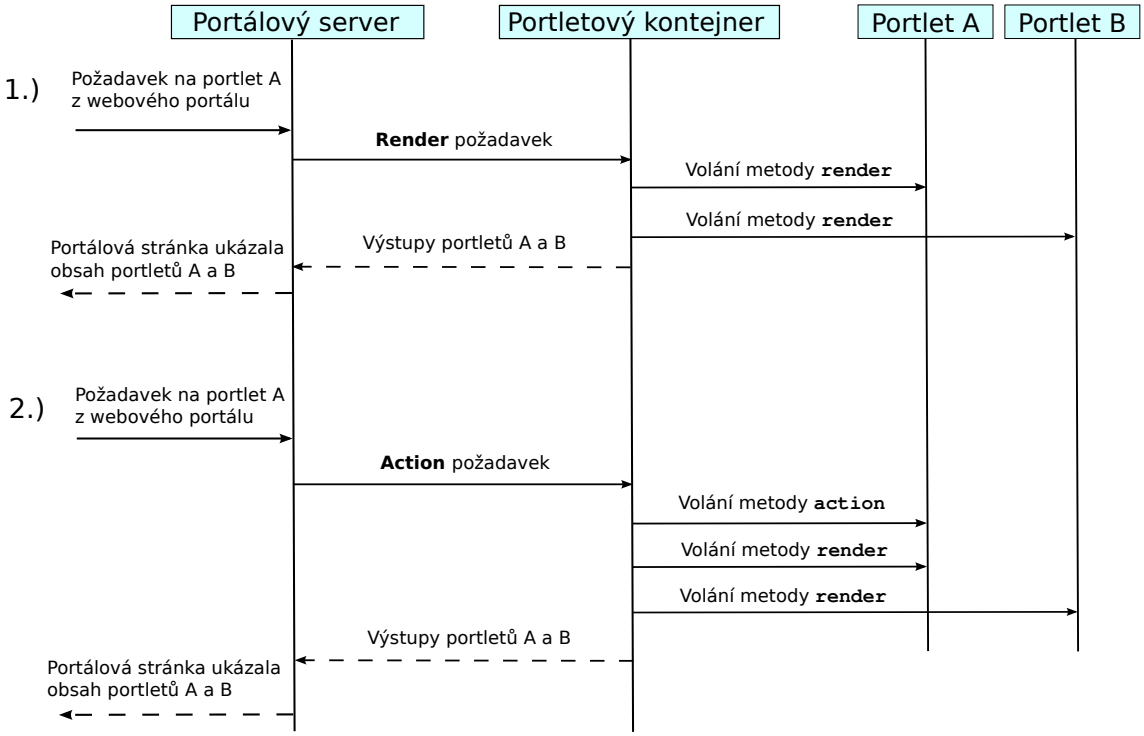
z paměti portletového kontejneru.

Abychom nemuseli u každého portletu implementovat vše na nejnižší úrovni, existuje základní implementace zmíněných metod v abstraktní třídě `GenericPortlet`, která zajišťuje pohodlnější práci s portletem. Tato třída již umí lépe pracovat s módy a mimo jiné implementuje další dvě důležitá rozhraní, které jsou v portletu verze 2.0 volitelné. První rozhraní se nazývá `EventPortlet`, která obsluhuje požadavky na události a druhé rozhraní je `ResourceServingPortlet`, které obsluhuje požadavky na dynamické zdroje. Díky anotacím `@ProcessAction`, `@ProcessEvent` a `@RenderMode`, můžeme používat vlastní názvy výše definovaných portletových metod. Tyto anotace nám budou naše názvy mapovat na názvy původní a dokonce podporují jemnější členění na metody daného typu pro určitý mód.

3.3 Portletový kontejner a životní cyklus portletu

Na začátku této kapitoly jsme řekli, že portletový kontejner zajišťuje kompletní správu portletů včetně řízení jejich životního cyklu. Každý portlet, který je na dané portálové stránce, portletový kontejner nejprve načte a vytvoří instanci, kterou si uloží do paměti. Následně inicializuje portlet jeho metodou `init`. Poté portletový kontejner obsluhuje všechny typy požadavků na portlety a nakonec ukončí portlet metodou `destroy`.

Nyní se podíváme na automat na obrázku 3.2, který má reprezentovat možnou sekvenci požadavků, které zasílá danému portletu portletový kontejner ve verzi 1.0. Uzly na obrázku charakterizují typy požadavků a orientované hrany reprezentují časovou závislost. Šipka vždy směřuje od předchozího typu požadavků k možnému následnému typu požadavku. Po inicializaci portletu může portlet nejdříve obdržet pouze požadavek na vykreslení. Následu-



Obrázek 3.4: Sekvenční diagram zpracování portletových požadavků

jící požadavek, který portlet obdrží může být další požadavek na vykreslení v případě, že se aktualizoval jiný portlet na stejné portletové stránce požadavkem na akci. Druhou možnosti je, že portlet sám obdrží požadavek na akci, následkem něhož se pak znovu překreslí opět požadavkem na vykreslení.

Sekvence možných požadavků v portletu verze 2.0 je už složitější, což můžeme vidět na obrázku 3.3. Hlavní rozdíl je ten, že zde již existují 4 typy požadavků. Určitě si všimnete, že zde může nastat jako počáteční požadavek i požadavek na obsluhu události. Funguje to totiž tak, že portletový kontejner zachytává všechny události, které se portletu týkají a těsně před následujícím požadavkem na vykreslení je portletu zašle. Portletový kontejner zachytává události pro portlet i pokud není na aktuální portálové stránce. Navíc je důležité, že události se mohou řetězit a jedna událost připravená pro portlet může vyvolat několik dalších. V momentě, kdy nastal požadavek na vykreslení může dále nastat několik scénářů. První scénář je takový, že portlet dostane opětovně požadavek na vykreslení kvůli akci na jiném portletu. Při druhé možností si klient vyžádá nějaký zdroj přes zdrojovou URL (netypický princip fungování portletových URL bude popsán o pár odstavců níže) dostupnou z aktuálně zobrazeného portletu (např. stažení dynamicky vytvořeného souboru). Požadavek na zdroj se může opakovat. Třetí možností je vykonání nějaká akce. Akce může být následována řetězcem událostí, které portletový kontejner pro portlet nastřádal a nakonec proběhne opět vykreslení. Poslední možnosti se před překreslením portletu obslouží nejprve všechny nashromážděné události.

Na obrázku 3.4 je znázorněn příklad komunikace *požadavek – odpověď*. V prvním případě vidíme příchozí požadavek na portlet A, který obdržel portál. Portál po rozpoznání požadovaného typu zprávy zašle požadavek na vykreslení na portletový kontejner. Portletový

kontejner na to reaguje tak, že zavolá metodu `render` u portletu A. Portlet A pak vrátí vygenerovaný obsah portletovému kontejneru. I přesto, že se volal požadavek nad portletem A, provádějí se render operace i nad ostatními portlety. Portletový kontejner následně zašle obsahy výstupů obou portletů portálovému serveru, který zkombinuje příslušným způsobem obsah, přidá nejrůznější dekorátory oken a zašle stránku prohlížeči. Ve druhém případe je zaslán požadavek na akci také portletu A. Po vykonání metody `proccessAction` portletem A zavolá portletový kontejner metodu `render` u všech portletů. Konec probíhá analogicky viz. první příklad. U požadavků na vykreslení nebo na akci volá nakonec portletový kontejner metodu `render` u všech portletů na dané stránce, protože obsahy mohou být provázány a zpravidla i jsou. V obou případech předpokládáme, že nenastala žádná událost, kterou by musel portlet obsloužit.

V jednom z předchozím odstavců jsme se zmínili o existenci speciálních URL reprezentujících požadavky na vyžádání zdroje. Jak tuto URL vytvořit? Nejprve je důležité si říct, že hypertextové odkazy na portletových stránkách se vytvářejí odlišným způsobem než jsme zvyklí. Portlety mají speciální portletové URL k zasílání požadavků určitého typu daným portletovým instancím. Na rozdíl od servletů nemáme možnost jak explicitně pomocí URL mapovat konkrétní portlet. Portletový kontejner disponuje objekty (př. `RenderResponse`), pomoci nichž si portlety vytvoří sebe-odkazující URL. Spolu s vygenerovanými daty zasílá portlet i příslušnou URL portletu a ta se nastaví u případných odkazů referencujících portlet např. v atributu `action` u formuláře. URL portletu se získá již při jeho prvním vygenerování. Z toho ovšem vyplývá i malý zádrhel – jednotlivé portlety nemohou vygenerovat URL na portlety jiné. Portletové požadavky popsané v kapitole 3.2 lze mapovat na konkrétní URL. Požadavky na akci obsahují v URL název akce a případně například požadavky na změnu módu nebo stavu okna portletu. Požadavky na vykreslení zase obsahují v URL informace důležité pro výběr obsahu a podobně vypadají i URL požadavků na zdroj. Jediný požadavek, který nemá odpovídající URL je požadavek na událost. To vyplývá už ze samé podstaty událostí, které se zasílají pouze mezi portlety. Uživatel tento požadavek iniciovat nemůže.

Již jsme se zmínili o jistém portletovém kontextu, který portlet využívá. Tento kontext obsahuje spoustu informací včetně nastavení z konfiguračního souboru portletu. Konfigurační soubor mimo jiné může obsahovat i nastavení, které ovlivňuje chování portletového kontejneru. Pojďme se na konfigurační soubor podívat detailněji.

Konfigurační soubor portletu umožňuje nastavit název portletu, třídu, která portlet implementuje, zobrazený název portletu na portálové stránce, povolené módy a stavy okna, úložiště statických textů tzv. *resource bundle*, nastavení portletového kontejneru a další věci... Portletová specifikace 2.0 definuje čtyři základní nastavení portletového kontejneru, které musí být podporovány. V různých implementacích kontejneru ale můžou být podporovány další nastavení. Jedním ze základních nastavení je například volba *actionScopedRequestAttributes*, která zajistí sdílení parametrů mezi metodou `processAction` a následující případnou sekvencí metod `render`.

Nyní už víme, jak funguje životní cyklus portletu, jaké fáze portletu mohou kdy nastat. Může ovšem nastat případ, kdy chceme klasické vykonání požadavku poupravit bez toho, abychom museli zasahovat do samotného portletu. K tomuto účelu slouží *portletové filtry*, na které se zaměříme v další sekci.

3.3.1 Portletové filtry

Portletové filtry [33, 47, 49] umožňuji upravit chování portletu na daný portletový požadavek. Filtry implementují návrhový vzor dekorátor a umožňují přidat rutinu před samotným zpracováním požadavku, po zpracování nebo před i po. Filtry můžeme řetězit a díky tomu je můžeme vyvíjet nezávisle. Musí pouze implementovat určité rozhraní. Filtry se využívají v případě, kdy chceme například doplnit dodatečné parametry do portletového požadavku nebo pokud chceme kontrolovat přístup k portletu a zvýšit tím bezpečnost přístupu k datům nebo například pokud chceme uchovávat nějaká statistická data apod. Pomocí filtrů se také implementují tzv. *portletové mosty*, na které se ale podíváme až v sekci 3.4.

Abychom vytvořili portletový filtr musíme vytvořit třídu, která bude implementovat rozhraní `PortletFilter` nebo některé rozhraní, které toto rozhraní rozšiřuje. Další rozhraní jsou `RenderFilter`, `ActionFilter`, `EventFilter` a `ResourceFilter`. Třída bude pak vždy obsahovat metody `init` a `destroy` a podle implementovaného rozhraní i metodu `doFilter` s příslušnými parametry. Pro `RenderFilter` to budou parametry typu `RenderRequest`, `RenderResponse` a `FilterChain`. První dva atributy mají pochopitelný význam a poslední parametr slouží k tomu, abychom mohli v těle filtru předat řízení dalším filtrům v řetězci. Posledním článkem řetězce je samotný portlet.

Filtr připojíme ke konkrétnímu portletu v konfiguračním sobouru `portlet.xml`. Nejdříve musíme filtr nadeklarovat a namapovat na typ požadavku, který bude filtr obalovat. Pro deklaraci filtru se používá tag `<filter>` a pro mapování tag `<filter-mapping>`.

3.3.2 Uživatelské nastavení portletů

U portletů bývá běžné, že si můžeme přizpůsobit některé jejich chování nebo vzhled. K nastavení slouží zpravidla mód `EDIT`. Portletový požadavek (objekt typu `PortletRequest`) obsahuje metodu `getPreferences()`, která vrací objekt typu `PortletPreferences`. Ten pak bude obsahovat dvě základní metody. Metoda `setValue(klic, hodnota)` umožňuje uložit hodnotu pod daným klíčem a metoda `getValue(klic, preddefinovanaHodnota)` vrací hodnotu s daným klíčem. Pokud klíč neexistuje, tak se vrátí `preddefinovanaHodnota`. Omezení uživatelských nastavení je takové, že umí ukládat pouze objekty typu řetězec. Použití je tedy značně omezené.

Portletové nastavení může standardně měnit kdokoliv, kdo má na to oprávnění. Oprávnění lze upravovat v nastavení portálových stránek a samotných portletů v administračním rozhraní portálu. Například portlet přidaný na skupinovou stránku bude moci nastavovat každý člen této skupiny a změny portletu se projeví všem. Pokud chceme mít soukromý portlet, který budeme moci upravovat pouze my, je důležité ho vložit do sekce *nástěnka* (angl. dashboard). Tuto sekci má každý uživatel vlastní.

3.3.3 Mezi-portletová komunikace

V rámci portálu lze provozovat několik portletových aplikací, přičemž každá portletová aplikace se může skládat z více portletů. Mezi-portletovou komunikaci využíváme k provázání jednotlivých portletů buď v rámci jedné portletové aplikace, nebo v rámci samotného portálu. Příkladem může být portlet, který vytváří nové události do kalendáře a jiný portlet, který zobrazuje několik nejbližších událostí. Určitě by se hodilo mít mechanismus k předání informace o vytvoření události z prvního portletu do druhého, který dle toho může v případě potřeby upravit svůj obsah. V portletu 2.0 existují tři základní druhy mezi-portletové komunikace, a to tzv. portletové sezení (angl. *Portlet session*), veřejné parametry (angl. *Public render parameters*) a portletové události.

Portletové sezení bylo jedinou možností jak komunikovat v portletrech verze 1.0. To umožňovalo komunikaci portletů pouze v rámci jedné portletové aplikace a pokud byla potřeba komunikovat s portletem jiným, musel se do aplikace přibalit. Použití více portletů v rámci jedné portletové aplikace vyžaduje definici konkrétních portletů v elementu `<portlet-app>` v souboru portlet.xml. K ukládání a získávání informací z portletového sezení slouží objekt typu `PortletSession`. Mezi výhody této komunikace patří možnost zasílat objekty libovolného typu, k implementaci není potřeba nic složitého díky tomu, že jsou portlety součástí jedné portletové aplikace a portlety spolu mohou takto komunikovat, i když jsou součástí různých portálových stránek. Mezi nevýhody patří již zmíněná nutnost zabalení všech portletů v rámci jedné aplikace do stejného balíčku [47].

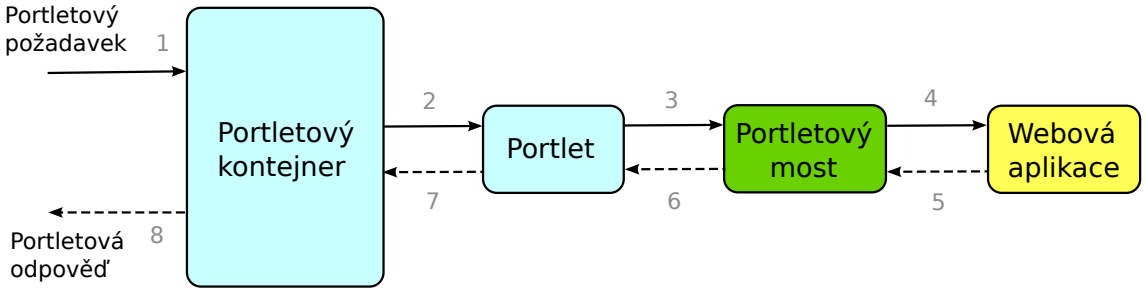
Veřejné parametry jsou jedním typem parametrů, které dostávájí *render* metody a jsou viditelné i ostatními portlety. Tento typ komunikace je dostupný ve verzi 2.0. Druhým typem jsou soukromé parametry. Veřejné atributy se označují v konfiguračním souboru v daném portletu elementem `<supported-public-render-parameter>`. Zviditelnění pro ostatní portlety se provede deklarací daného parametru v elementu `<public-render-parameter>`. Tento přístup je nejjednodušší z hlediska absence potřeby zásahu do kódu. Vše se nastavuje v konfiguračním souboru. Další výhodou je možnost komunikovat napříč portletovými aplikacemi uvnitř portálu. Kontejner tyto parametry ukládá do URL, takže je jednoduché si je v záložce prohlížeče uložit a při příštím spuštění stránek znovu hned inicializovat. Hlavní nevýhodou je podpora pouze datových typu řetězec (`String`) a pole řetězců (`String[]`). Další nevýhodou je nemožnost sdílet veřejné parametry mezi různými portálovými stránkami (kvůli odlišné URL) nebo opožděné obdržování aktualizovaných parametrů [47].

Portletové události jsou dalším typem portletové komunikace od verze 2.0. Jednotlivé portlety si zasílají zprávy skrze portletový kontejner. Zasílání konkrétních událostí se musí deklarovat v konfiguračním souboru pomocí elementů `<supported-publishing-event>` (pro vysílající portlet), `<supported-processing-event>` (pro přijímající portlet) a definice událostí v elementu `<event-definition>`. K práci s událostmi se potom používá objekt typu `EventResponse`. Mezi hlavní výhody patří možnost zasílat v události libovolné objekty, správa kešování portletovým kontejnerem a možnost vytváření řetězu událostí. K hlavním nevýhodám patří větší zatížení portletového kontejneru a obtížnější implementace [47].

3.3.4 Srování portletů se servlety

V této práci jsem již zmínil podobnost portletů a servletů z platformy Java EE. Servlety byly například využívány v portletech verze 1.0 při obsluze požadavků na nějaký zdroj, kvůli absenci požadavku typu Zdroj. Pomocí servletů se jednoduše daly obcházet nedostatky verze 1.0. Zkusím nyní ve zktrace popsat principiální rozdíly obou technologií, aby pojmy nemohly nijak splývat. Portlety jsou stejně jako servlety webové komponenty založené na Java EE technologii, které mají vlastní kontejner, který řídí jejich životní cyklus a celkově je spravuje. Portletový kontejner je zodpovědný za načítání portletů, vytváření instancí, inicializace, směrování požadavků a také odstraňování instancí podobně jako servletový kontejner. Portlety i servelety generují dynamický obsah.

Mezi hlavní rozdíly patří například rozdílná obsluha požadavků. Obsluhy jednoho HTTP požadavku se může nepřímo účastnit více portletů, ale v případě servletů pouze jeden servlet. Je to způsobeno tím, že při zaslání požadavku typu Akce nějakému portletu zašle následně kontejner požadavky na překreslení všem portletům na dané portálové stránce. Navíc se mohou vykonat před překreslením nějaké události, což se řadí také mezi typy požadavků. Další rozdíl je nevhodnost servletů pro obsahovou agregaci. Pokud chceme



Obrázek 3.5: Zpracování portletového požadavku při využití portletového mostu.

použít servlet pro agregování a prezentování obsahu v jednom okně, zodpovědnost zůstává na samotném servletu bez jakékoliv podpory kontejneru. Servlet dále vyžaduje více úsilí na personalizaci obsahu. Servlet musí sám vytvořit vhodné datové struktury pro ukládání a načítání personalizovaných dat a celý mechanismus pro ukládání a načítání dat. U portletů toto vše zajišťuje portletový kontejner. Další stěžejní rozdíl, který stojí za zmínění je ten, že servlety nemají podporu pro meziservletovou komunikaci založenou na událostech a celá komunikace se pak implementuje pouze přes Servlet API. Rozdílů exituje celá řada a ostatní je možné nalézt přímo v specifikaci JSR-286 [47].

3.4 Portletové mosty

Portletový most je komponenta, která stojí mezi portletovým prostředím a webovou aplikací a zajišťuje transformaci požadavků. Portletový most nám umožňuje implementovat naši webovou aplikaci bez nutnosti znalosti portletové technologie. Naše aplikace bude potom nezávislá na samotném portálu. Díky tomu můžeme naši webovou aplikaci vytvořit a až potom ji nasadit na portál s minimálními úpravami. Musíme si ale dát pozor na to, aby náš portletový most podporoval všechny technologie, které v naší aplikaci používáme.

Na obrázku 3.5 vidíme sekvenci reakcí na portletový požadavek při využití portletového mostu. Příchozí požadavek je zachycen portletovým kontejnerem, který ho zpracuje a přiřadí konkrétnímu portletu. Portlet zde funguje jako obálka nad portletovým mostem a ten zase jako obálka nad samotnou webovou aplikací. Portletový most obecně rozšiřuje třídu `GenericPortlet` a transformuje portletový požadavek na požadavek, kterému rozumí naše aplikace. Po zpracování operace se odpověď vrátí zpět [47].

Nejjednodušší možnost, jak přidat naši aplikaci do portletu je za použití tzv. *iFrame*. Tato technika nám dovoluje vnořit externí stránku do našeho portletu, ale ztrácí se nám tam veškeré výhody portletů. Jedná se například o ztrátu podpory personalizace obsahu a dochází zde k celkovému odtržení aplikace od portletového kontextu. Aplikace tedy dále nezjistí stav portletového okna, jeho mód a další informace. Mezi výhody by snad mohla patřit možnost vzdálené autentizace.

Další možností je použít tzv. *JSF Portlet*. Tato technologie nám umožňuje použít technologii *Java Server Faces* z balíčku Java EE. Portlet bridge je specifikován v JSR-301 [18] pro JSF 1.2 v kombinaci s portletem verze 1.0 a v JSR-329 [19] pro JSF 1.2 s portletem 2.0. Většina portálových serverů ale zajišťuje i podporu pro JSF 2.0, mezi které patří i portlet bridge od firmy RedHat s názvem *JBoss Portlet Bridge* [47]. JBoss Portlet Bridge podporuje mimo specifikaci technologie JSF i jiné technologie jako například framework *RichFa-*

ces [5]. RichFaces framework je bohatá knihovna komponent pro technologii JSF. RichFaces rozšiřuje JSF o nadstandardní komponenty s podporou asynchronních JavaScriptových serverových požadavků a XML (*AJAX*) [26], které usnadňují vývoj enterprise webových aplikací a vylepšují přívětivost jejich uživatelského rozhraní.

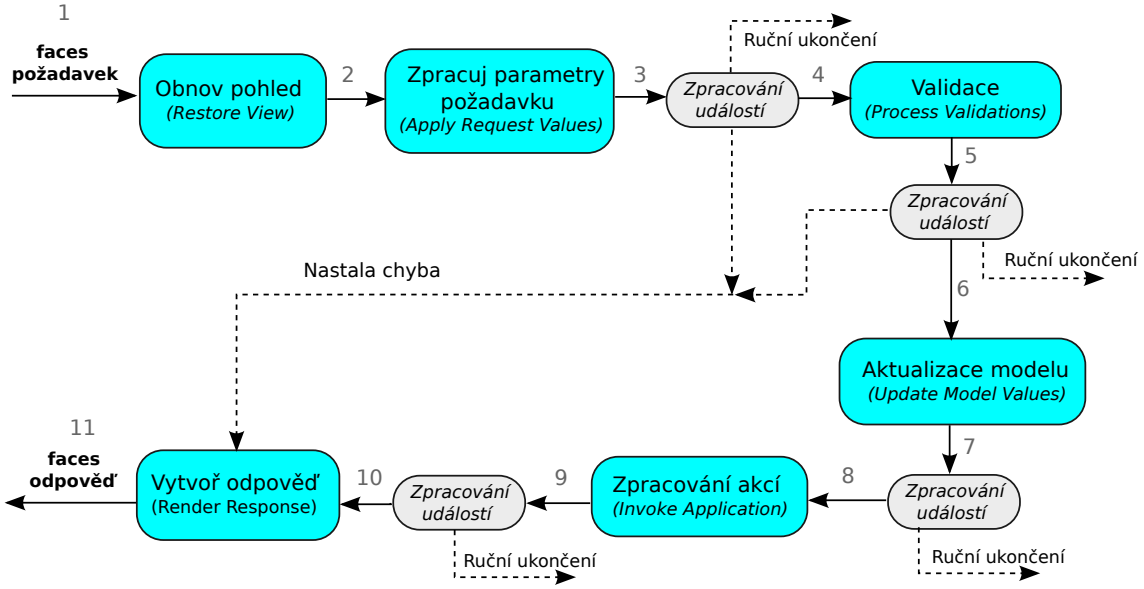
3.4.1 Java Server Faces portlety

V této sekci se zaměříme na to jak vytvořit portlet využívající technologii JSF. Nejdříve se ale podíváme jak JSF funguje a následně rozebereme, jak z JSF webové aplikace vytvořit JSF portlet a jak do ní přidat portletové rysy, které samotná JSF aplikace neumí. Jedná se například o možnost personalizace zobrazovaných dat apod. Detailněji si také popíšeme, jak se vypořádat s rozdílným životním cyklem JSF aplikací a portletů.

Java Server Faces je framework pro tvorbu uživatelského rozhraní ve webových aplikacích. Významným způsobem zjednodušuje psaní a údržbu webových aplikací běžících na Java aplikačních serverech, které generují klientský obsah webové stránky. JSF zjednodušuje používání webový formulářů a znovupoužitelných komponent. Architektura JSF frameworku ctí návrhový vzor *model-pohled-kontrolér* (angl. model-view-controller, MVC) a díky tomu výrazně zjednodušuje pohyb dat mezi klientskou webovou stránkou a datovým modelem aplikace. Prostředník mezi datovým modelem a uživatelským rozhraním je právě zmíněný kontrolér. JSF framework navíc pomáhá udržovat stav uživatelských komponent napříč klientskými požadavky, zjednodušuje psaní klientem generovaných událostí pro serverovou část aplikace a umožňuje snadné vytváření vlastních znovupoužitelných komponent pro uživatelské rozhraní. V této sekci budeme často využívat slovo *pohled*. Pohledem v kontextu technologie JSF budeme rozumět nějaký XHTML soubor, který obsahuje zpravidla zobrazitelné JSF komponenty uživatelského rozhraní. Pohled tedy vytváří reprezentaci stránky (nebo její části), kterou uživatel vidí v internetovém prohlížeči. Pohled je velice abstraktní slovo a občas se v kombinaci s technologií JSF mluví o tzv. *faceletech*. Oba pojmy budou mít v této sekci shodný význam.

JSF framework řeší řadu praktických webových problémů vznikajících psaním aplikací pro HTML klienty využívajících HTTP protokol. Jak už bylo zmíněno, JSF framework dokáže udržet stav komponent napříč klientskými požadavky. Dále zapouzdřuje značkovací jazyk pro tvorbu HTML kódu kompatibilního s různými webovými prohlížeči, umožňuje jednoduše validovat data odesílaná na server a umí elegantně obsluhovat případné chyby. Dále dovoluje konvertovat řetězce na objekty a naopak, což je pro potřeby textového HTTP protokolu velice příjemné. Za zmínku jistě také stojí schopnost jednoduché navigace mezi klientskými stránkami. Navigaci je možno definovat v konfiguračním souboru nebo přímo v Java kódu pomocí návratových řetězců klientských akcí.

Komunikace s naší JSF aplikací probíhá pomocí HTTP protokolu na bázi požadavek – odpověď. Existují dva typy požadavků a dva typy odpovědí. Pro každou kombinaci požadavku a odpovědi proběhne jiné zpracování. Prvním typem odpovědi je tzv. *faces odpověď*. Tato odpověď vznikla na základě vykonání fáze *Vykresli* (angl. Render Response) v JSF životním cyklu, který si probereme níže. Tato odpověď zpravidla obsahuje akční komponenty, které jsou namapovány na náš JSF kontrolér a často mohou následně přesměrovávat na jiný pohled naší aplikace. Pokud vykonáme nějakou akci na stránce vykreslené z faces odpovědi, tak zašleme tzv. *faces požadavek*. Cílem tohoto požadavku ale může být například i obrázek nebo kaskádový styl apod. Druhým typem požadavku je tzv. *non-faces požadavek*, který je směrován na nějaký servlet nebo *java server pages* (JSP) stránku. Tento požadavek může být také směrován na obrázek nebo kaskádový styl. Odpovědí na tento požadavek je tzv.

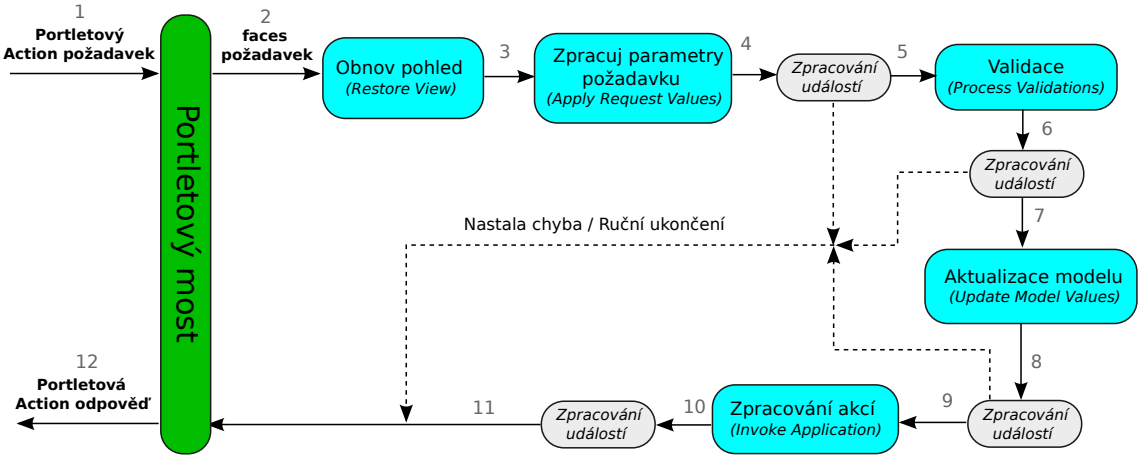


Obrázek 3.6: Životní cyklus faces požadavku.

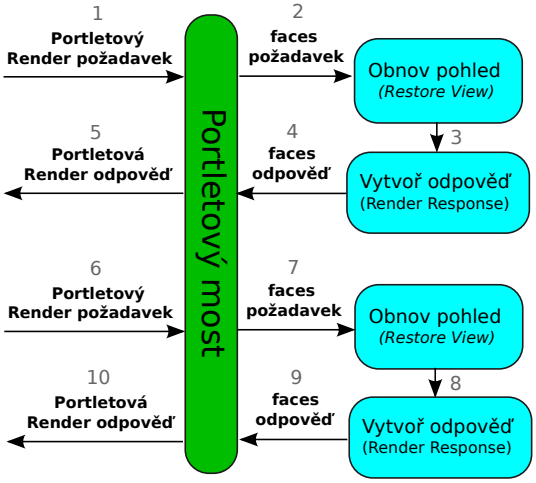
non-faces odpověď generovaná servletem. Dále se budeme zabývat jen faces požadavkem, který generuje faces odpověď.

Na obrázku 3.6 vidíme klasický životní cyklus faces požadavku s faces odpovědí. Cyklus obsahuje šest fází, nemusí ovšem proběhnout všechny. První fází *Obnov pohled* (angl. Restore View). V této fázi se zpracovává pohled požadované stránky. Pokud už byl pohled v minulosti jednou vytvořen, tak se jen načte a uloží do kontextu aplikace. Pokud vytvořen nebyl, musí se vytvořit, a následně se přejde do *Vytvoř odpověď* (angl. Render Response), kde se prázdný pohled naplní komponentami. Toto je speciální případ a na obrázku není zachycen. V běžném případě následuje fáze *Zpracování akcí* (angl. Apply Request Values). V této fázi se aktualizují lokální komponenty pohledu na základě informací v požadavku. Dochází také k případné konverzi řetězcových hodnot na objekty. Další fází je fáze *Validace* (angl. Process Validations), která jednoduše zkontroluje hodnoty lokálních komponent zaregistrovanými validátory. Následuje fáze *Aktualizace modelu* (angl. Update Model Values). Aktualizované a validované hodnoty ze strany lokálních komponent se uloží do modelu. I zde ale může dojít k validačním chybám. Nemusí například sedět datový typ apod. Předposlední fáze se nazývá *Zpracování akcí* (angl. Invoke Application). V této fázi se zpracují akce vzniklé například kliknutím na tlačítko odeslat ve formuláři. Na základě návratové hodnoty nebo konfiguračního souboru (`faces-config.xml`) se dále určí, na jakou stránku se aplikace přesměruje. Poslední fází je již zmíněná fáze *Vytvoř odpověď*, ve které se vytvoří faces odpověď a zašle se zpět klientovi. Tato fáze proběhne vždycky i pokud došlo k nějaké chybě. Vzniklé chyby se pak uživateli zobrazí v příslušné chybové komponentě, pokud ji vývojář na pohled přidal. Vysvětlený životní cyklus se ale dá i předčasně programátorsky ukončit, pokud je výsledkem například přesměrování na jinou URL nebo stahování souboru. To znázorňují na obrázku šipky s popiskem „Ruční ukončení“.

Hlavní rozdíl mezi životním cyklem JSF požadavku a portletového požadavku je ten, že všechny fáze životního cyklu JSF požadavku se vykonají během tohoto požadavku, ale u portletového požadavku se každá fáze vykonává samostatným požadavkem viz. sekce 3.3.



(a) Action požadavek



(b) Render požadavek

Obrázek 3.7: Životní cyklus portletového požadavku s využitím portletových mostů.

JSF požadavky spoléhají na to, že platnost kontextu požadavku zůstává mezi vykonáním akce a vytvářením odpovědi stejná, ale u portletových požadavků, to tak obecně není. Tento problém řeší právě portletový most. Na obrázku 3.7 můžeme vidět diagram životního cyklu Action požadavku, který využívá portletový most. Můžeme vidět, že narozdíl od diagramu na obrázku 3.6 se zde negeneruje faces odpověď. Do portletové Action odpovědi se ale zakóduje informace, odkud se získá faces odpověď pro následné portletové Render požadavky. Dalším rozdílem je to, že při ručním ukončení JSF životního cyklu se obsluha předá portletovému mostu.

Na obrázku 3.7 můžeme také vidět, jak bude probíhat obsluha portletového Render požadavku. Ve fázi *Obnov pohled* se již načte předpřipravený pohled a rovnou se z něj vytvoří faces odpověď. Pohled přetrvává i pro následné Render požadavky. Podpora portletových událostí a požadavků na zdroj je o něco složitější a pro bližší informace odkážeme čtenáře na odbornou literaturu [19].

Nyní se zaměřme na to, jak lze z JSF webové aplikace udělat JSF portlet. V prvé řadě je potřeba odstranit z faceletů (JSF pohledů) tagy, které jsou nadbytečné. Portletem vygenerovaný XHTML kód by totiž měl být validní. Jedná se například o tag `<html>`. Z hlediska validního kódu je nežádoucí, aby na naše stránka obsahovala čtyři tyto tagy, když může správně obsahovat pouze jeden. Tag `<html>` se nahrazuje tagem `<f:view>`. Dále je potřeba z kódu odstranit metody `redirect` objektu typu `ExternalContext`, protože portletový most přesměrování implicitně nepovoluje. Klíčovou roli hraje přidání konfiguračního souboru `portlet.xml`, díky němuž transformujeme naši webovou aplikaci na portlet. Již jsme si vysvětlili že v tomto souboru se definuje třída, která dědí od portletového rozhraní. My pro tento účel použijeme třídu `javax.portlet.faces.GenericFacesPortlet`. Nyní je dobré nastavit určité parametry důležité pro portletový most, abychom si mohli přizpůsobit chování portletového mostu. V portletovém mostu verze 2.0 existuje implicitně sedm typů inicializačním parametrů. Začneme tím, že si namapujeme naše podporované módy portletu na konkrétní facelet (JSF XHTML šablona). K tomu slouží inicializační parametr `javax.portlet.faces.defaultViewId.[mód]`. Dalším důležitým parametrem je `javax.portlet.faces.preserveActionParams`, který zajistí, že portletový most si zapamatuje parametry portletového požadavku na akci pro budoucí požadavky na vykreslení. Ostatní parametry není v tuto chvíli nutné dál rozebírat.

V praxi může nastat situace, že budeme chtít v naší JSF webové aplikaci přistoupit ke skutečným objektům typu `PortletRequest` nebo `PortletResponse`. K těmto objektům se dostaneme pomocí aktuálního kontextu aplikace, přes objekt typu `FacesContext`. Můžeme se například dostat k objektu typu `Request`, který stačí přetypovat na objekt typu `PortletRequest`. Přetypování ovšem nebude fungovat v prostředí klasické webové aplikace. Měli bychom tedy nejdříve testovat typ požadavku a přetypovávat až v podmíněném bloku. Naše aplikace by měla být co možná nejvíce univerzální, abychom ji mohli s minimálními úpravami použít jak v prostředí portletů, tak jako klasickou webovou aplikaci. Přes portletový požadavek se dostaneme i k uživatelským nastavením, kde můžeme uživateli nastavit některé jeho preference pro daný portlet. Jedná se o objekt typu `PortletPreferences`. Každý portál toto úložiště dat automaticky zajišťuje a měli bychom si zjistit, kam se data ukládají fyzicky. Portál GateIn v kombinaci s aplikačním serverem *JBoss AS*, který bude v této práci použit, má datové úložiště konfigurovatelné v souboru `$JBOSS_HOME/standalone/configuration/standalone.xml`[1].

V tuto chvíli byl položen teoretický základ pro vytvoření jednoduchého JSF portletu. Případné nejasnosti a podrobnější informace najde čtenář v aktuální specifikaci JSF portletového mostu [19].

3.5 Spring portletový MVC framework

Spring Portletový MVC framework [45], je framework pro tvorbu portletů. Tato technologie bude popsána jen okrajově, protože nebude použita, ale bude zajímavá pro srovnání možných technik tvorby portletů v sekci 4.2.

Framework slouží k jednodušší a efektivnější tvorbě portletů oproti využití základního portletového rozhraní. Tento framework je nadstavba nad základním portletovým API využívající Java anotace, což velice usnadňuje vývoj portletů. Framework zajišťuje tzv. inverzi řízení (angl. Inversion of Control, IoC), díky němuž může kontrolér načítat závislosti z konfiguračního souboru a následně je vkládat do požadované instance třídy (angl. Dependency

[1] https://docs.jboss.org/author/display/GTNPORTAL36/Database+Configuration

Injection, DI). Další výhodou oproti základnímu přístupu tvorby portletů je možnost mapovat kontrolér na požadavek jak podle typu požadavku a jeho módu, tak i podle parametrů požadavku.

Tato technologie je jistě zajímavá, ale více se jí zabývat nebudeme. Čtenáře tedy odkazuji na literaturu [45, 47].

3.6 Další použité technologie

Tato sekce má za cíl principiálně vysvětlit a seskupit technologie a nástroje, které byly použity v rámci této diplomové práce. Vzhledem k prostorovému omezení budou prezentovány jen klíčové prvky těchto technologií a nástrojů. Sekce bude nejdříve popisovat nástroj *Maven*, který jsem použil pro správu celého projektu. Dále budou popsány čtyři technologie z platformy Java EE, a to technologie *Java Persistence API* (JPA), *Enterprise Java Beans* (EJB), *Java API for RESTful Web Services* (JAX-RS) a *Contexts and Dependency Injection* (CDI). Technologii JSF jsem již popsal v sekci 3.4.1, takže ji v tomto přehledu neuvádím. Na konci sekce se ještě podíváme na knihovnu *Lombok*, která umožnila velice zpřehlednit kód díky svým užitečným anotacím.

Nástroj **Apache Maven** [17] umožňuje správu projektů v jazyce Java. Tento nástroj podporuje vytváření projektů dle nadefinovaných vzorů tzv. archetypů. Maven stáhne daný archetyp z internetu a připraví ho pro okamžité použití. Dále umožňuje jednoduchou a efektivní správu knihovních závislostí pomocí definovaných repozitářů. Z těchto repozitářů umí stahovat požadované knihovny, pokud ještě v systému nejsou. Maven dokáže spravovat současně několik projektů, je rozšiřitelný pomocí vlastních pluginů, umí kompilovat projekty do archívů a dále řídit jejich životní cyklus. Maven také podporuje dědičnost projektů, což je vlastnost dědění projektových nastavení od rodičovských projektů. Když dědíme z nějakého projektu, přebíráme jeho vlastnosti a zbývá nám pak už jen definovat název a verzi projektu. Samozřejmě ale lze zděděné vlastnosti přenastavit.

Maven projekt má předdefinovanou adresářovou strukturu, ale zejména obsahuje důležitý soubor `pom.xml`. V tomto souboru je definováno veškeré nastavení nástroje maven pro daný projekt. Celý konfigurační soubor je obalen tagem `<project>` a musí vždy mimo jiné obsahovat tagy `<groudId>`, `<artifactId>` a `<version>`, které jednoznačně a globálně identifikují náš projekt. Důležitými tagy jsou potom `<dependencies>`, kde jsou obsaženy všechny nezděděné závislosti projektu, a `<build>`, kde jsou nadefinovány jednotlivé cíle pro zpracování projektu. Tyto cíle mohou být *compile*, *test*, *paskage*, *install*, *deploy* a další. Cíl package nám třeba zkompiluje celý projekt a uloží ho do definovaného typu archívu.

Nyní se podíváme na několik technologií z platformy Java EE, které byly v práci použity. Knihovny těchto technologií musely být samozřejmě nadefinovány v konfiguračním souboru projektu pro nástroj Maven. Maven již pak zajistil stažení těchto knihoven a začlenění do projektu.

Technologie **Java Persistence API** (JPA) [10, 36] slouží k objektově – relačnímu mapování (ORM) entit. Abychom mohli mapovat instance třídy na záznamy v nějakém datovém úložišti, musíme využít třídní anotaci `@Entity`. Třídní anotací myslím takovou anotaci, která se uvádí před deklaraci třídy. Typicky se třída potom mapuje na tabulku v databázi. Třída může obsahovat pouze povolené typy atributů a pokud nám tyto nestačí, můžeme si vytvořit vlastní. Pro vytvoření asociace mezi entitami se využívají anotace *@OneToOne*, *@OneToMany*, *@ManyToOne* a *@ManyToMany*. Anotace *@OneToMany* se například využívá pokud chceme aby naše entita obsahovala kolekci jiných entit. Inverzní vazba v prvku této kolekce bude obsahovat atribut na rodiče s anotací *@ManyToOne*. Další dva typy asoci-

ací mají význam již zřejmý. Základní operace s entitami pak umožňuje správce entit (objekt typu `EntityManager`), který disponuje operacemi pro uložení nové entity do databáze, její aktualizací, mazáním a dalšími operacemi. Úložiště se kterými správce entit může pracovat jsou nadefinovány v souboru `persistence.xml`. Pro namapování konkrétního úložiště na správce entit se využívá anotace `@PersistenceContext` obsahující parametr s názvem úložiště. Správce entit dokáže také pracovat s transakcemi. Technologie JPA je pouze specifikací technologie. Specifikace má několik implementací, kde asi nejznámější je *Hibernate*, kterou momentálně vlastní a spravuje firma Red Hat.

Enterprise Java Beans (EJB) [29, 37] je technologie, která se také řadí k platformě Java EE. EJB definuje řízené komponenty enterprise aplikací na straně serveru. Zaměřuje se na *business* vrstvu, což je vrstva, ke které nemáme zpravidla přímo přístup z prezentační vrstvy. V business vrstvě se implementují řešení pro firemní procesy a často se opakující úkony. Programátor může tuto vrstvu vyvíjet skoro nezávisle. Komponentám v této vrstvě se někdy říká „back-end“ komponenty. Mezi business vrstvou a prezentační vrstvou se nalézá ještě jedna vrstva, která obsahuje rozhraní dostupné z prezentační vrstvy. Tuto prostřední vrstvu v našem případě zajišťuje technologie Java Server Faces a jejím komponentám se občas říká „front-end“ komponenty. Ale zpátky k EJB. EJB komponenty jsou spravovány tzv. EJB kontejnerem, který spravuje jejich životní cyklus, načítá závislosti (vlastnost *Dependency Injection*), zajišťuje automatické transakce na úrovni metod a řadu dalších funkcí... Existuje několik základních typů EJB komponent. Prvním typem jsou bezestavové EJB komponenty, které jsou kontejnerem vytvořeny pro každý požadavek znova. Označují se anotací `@Stateless`. Druhým typem jsou stavové komponenty (`@Statefull`), které v rámci jednoho sezení udržují stav komponenty. Třetím typem jsou tzv. *singleton* komponenty (`@Singleton`), které mají v rámci kontejneru pouze jednu instanci a jsou sdílené. Jednotlivé požadavky jsou ovšem synchronizovaně obsloužené. Singletony se například užívají pro globálně sdílené datové úložiště, jež chceme načíst dopředu pro šetření systémových prostředků (např. různé číselníky apod.). Posledním typem jsou EJB komponenty řízené zprávami (angl. *Message Driven Beans*), které v rámci tohoto textu nebudeme dále rozebírat.

Technologie **Java API for RESTful Web Services** (JAX-RS) [42, 34] je programové rozhraní zajišťující podporu pro tvorbu webových služeb podle architektury s anglickým názvem *Representational State Transfer* (REST). Rozhraní REST se využívá pro jednotný a snadný přístup ke zdrojům v distribuovaném prostředí. Zdrojem mohou být chápana jak data, tak i stav aplikace. Všechny zdroje musí mít jednoznačný identifikátor v podobě URL. RESTová služba obvykle umožňuje čtyři základní operace se zdrojem, a to čtení, vytváření, editaci a mazání. Tyto operace se obvykle popořadě mapují na HTTP typy požadavků `GET`, `PUT`, `POST` a `DELETE`. Architektura REST v podobě JAX-RS podporuje několik důležitých anotací. Anotace `@Path` umožňuje mapovat relativní zdrojovou cestu na třídu nebo metodu. Anotace `@GET`, `@PUT`, `@POST` a `@DELETE` nad metodou mapují volaní metody na konkrétní HTTP typ požadavku. Anotace `@PathParam` umožňuje mapovat část zdrojové cesty na parametr metody Další používanou anotací je anotace `@Produces`, která nastavuje v hlavičce odpovědi typ obsahu odpovědi (MIME typ [3]) např. „text/html“, „text/calendar“ nebo „image/jpeg“. Anotací dále existuje ještě celá řada, ale více se jimi zabývat nyní nebudeme.

Další užitečná Java EE technologie s názvem **Contexts and Dependency Injection** (CDI) [28, 35] slouží ke správě závislostí mezi jednotlivými komponentami. Technologie má jistý průnik schopností s technologií EJB, která umí spravovat závislosti také. Mně osobně se ovšem s touto technologií CDI pracuje lépe kvůli intuitivnějším názvům anotací,

a proto jsem ji použil. CDI nám dovoluje například nastavit závislost komponent z business vrstvy do komponent webové vrstvy. Pro načtení (injektování) dané instance komponenty je potřeba užít anotaci `@Inject`. Dále můžeme použít anotaci `@Produces`, která z atributu globální komponenty nebo její metody vytvoří připojitelnou komponentu. Podobně jako se dala nastavit délka životnosti EJB komponenty, tak se dá nastavit délka životnosti komponenty, která využívá technologii CDI. Životnost může mít pouze na jeden požadavek (`@RequestScoped`), v rámci sezení (`@SessionScoped`), v rámci celé aplikace sdíleně (`@ApplicationScoped`) a další. Pro pojmenování komponenty se používá anotace `@Named`, která má implicitně název shodný s názvem třídy. Název ovšem začíná malým písmenem.

Nakonec se podíváme na užitečnou knihovnu, kterou v projektu použiji. Jedná se o knihovnu projektu **Lombok** [24] pro efektivnější práci s nastavováním (*settery*) a získáváním (*gettery*) hodnot atributů objektů. Používání getterů a setterů zabírá spoustu místa v kódu a ten se pak stává velice nepřehledným. Proto jsem použil tuto knihovnu, která disponuje anotacemi `@Setter` a `@Getter`. Anotace jsou použity tam, kde má setter a getter implicitní funkci a neprovádí žádnou operaci navíc. Tento krok pomohl výrazně zpřehlednit kód. Projekt Lombok obsahuje i plugin do známých vývojových prostředí pro podporu těchto anotací. Více informací o použití instalaci pluginu naleznete v příloze v souboru `README.txt`.

Tato kapitola měla za cíl seznámit čtenáře se všemi důležitými technologiemi, které budou v této práci použity. Nyní se můžeme posunout dále k samotné analýze a návrhu sady portletů pro správu času.

Kapitola 4

Analýza a návrh

Tato práce má za cíl vytvořit sadu portletů pro správu času za použití moderních technologií vyvíjených pod záštitou firmy Red Hat. Výsledný produkt by měl být součástí GateIn portálu a JBoss Enterprise Portal Platformy jako příklad, jak je možné portlety navrhnout a implementovat.

Následující sekce obsahuje neformální specifikaci sady portletů, ze které bude vycházet celý návrh a implementace.

4.1 Neformální specifikace

Cílem projektu je vytvoření sady portletů pro správu času v portálovém prostředí. Bude se jednat o nástroje pro správu kontaktů, správu úkolů a správu uživatelského kalendáře.

Správa kontaktů musí uživatelům umožnit vytváření, editaci, mazání a prohlížení si svých kontaktů. U kontaktu by měla být možnost zadat standardní atributy jako jméno, druhé jméno, příjmení, tituly před jménem, tituly za jménem, zobrazované jméno, pohlaví, datum narození, datum výročí, jazyky, kterými kontakt hovoří, emailové adresy, adresu bydliště (případně další adresy), telefonní čísla, webové adresy, komentář a případně další vhodné atributy. Portlet by měl umět vyexportovat vizitku nebo skupinu vizitek ve formátu vCard. Zároveň musí portlet umět importovat vizitky z vCard souboru a zařadit je do nové skupiny. Jedna vizitka může být přiřazena k libovolnému množství skupin vizitek. Portlet musí umožnit i správu těchto skupin. Důležitá bude možnost rychlého vyhledávání kontaktů.

Správa úkolů musí uživatelům umožnit vytváření, editaci, mazání a prohlížení si svých úkolů. Úkol by měl obsahovat atributy předmět, datum a čas vytvoření úkolu, datum a čas splnění úkolu, datum a čas pro termínovaný úkol, prioritu úkolu, popis úkolu a další vhodné atributy. Úkoly se mohou seskupovat do projektů a jeden úkol může patřit zároveň nejvýše do jednoho projektu. Každému úkolu bychom měli mít možnost přiřadit libovolné množství kontextů. Kontextem může být například klíčové slovo, které určuje, kde můžeme úkol vykonat. Kvůli efektivní správě času je vhodné, aby logika správy úkolů a uživatelské rozhraní bylo inspirováno technikou Mít vše hotovo (angl. Getting Things Done) viz. sekce 2.1. V portletu by tedy měly existovat složky úkolů, které jsou definovány v této technice. První složkou bude složka „Schránka“, která bude obsahovat všechny nově vytvořené úkoly před zpracováním. Časově naplánované úkoly budou rozděleny do tří složek, a to „Dnes“, „Zítra“, „Naplánováno“. Složka dnes bude obsahovat mimo naplánované úkoly na dnešek také úkoly, které byly naplánované na některý předchozí den a nestihly se vykonat. Tyto úkoly musí být zvýrazněné. Složka „Zítra“ bude mít logický význam a složka „Naplánováno“ bude ob-

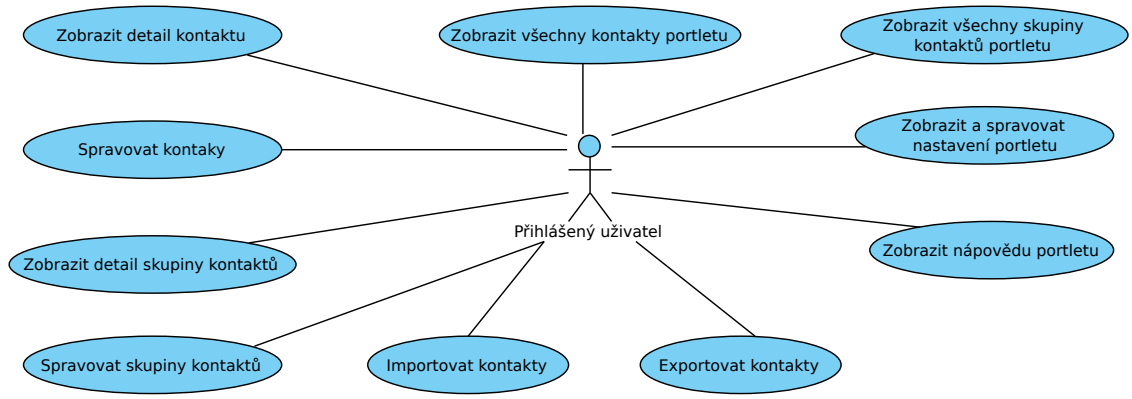
sahovat všechny naplánované úkoly od pozítří dál. Složka „Další“ musí obsahovat všechny úkoly, které je třeba udělat tak rychle jak to jen půjde, ale nemají přesně nadefinovaný termín. Další složkou bude složka „Někdy“, kde budou uvedeny všechny úkoly, na které není aktuálně čas, ale v budoucnu je zajímavé je vyřešit. Poslední složka „Čeká na“ bude obsahovat úkoly za jejichž vyřešení zodpovídá někdo jiný. Aplikace by dále měla umožnit listovat si již splněnými úkoly, které by měly být ve zvláštní sekci. Vhodná bude i možnost zobrazovat úkoly dle projektů, ke kterým patří. Nedílnou součástí portletu musí být export a import úkolů ve formátu iCalendar.

Posledním portletem bude uživatelský kalendář. Ten musí uživateli nabídnout následující úkony: vytvářet, editovat, mazat a prohlížet si svoje události. Událost by měla obsahovat atributy název, datum začátku a konce události, popis, místo, a případně další atributy. V definici termínu události je důležité, aby šlo nastavit, že je událost celodenní. Pokud tomu tak je a začátek akce není ve stejný den jako konec akce, automaticky se stává akce celodenní na všechny dny v tomto intervalu. U události je nutné, aby šlo nastavit její pravidelné opakovaní dle běžně používaných pravidel. Opakovaní tedy musí umět denní, týdenní, měsíční i roční opakování. Konec opakování půjde nastavit datem nebo počtem provedených opakování. U denních opakování musí jít nastavit opakování například každý třetí den, u týdenních opakování musí jít nastavit dny opakování, u měsíčních datum nebo například první pondělí v měsíci apod. Každá událost musí patřit právě do jednoho kalendáře. Kalendář musí obsahovat alespoň název a popis a musí existovat možnost jeho exportu do formátu iCalendar. Portlet by měl umožnit importovat události ze souboru do existujícího kalendáře. Pro možnost sdílení kalendářů je důležité, aby portlet uměl číst vzdálený kalendař v textové formě ve formátu iCalendar z dodané URL. Tento typ kalendáře půjde pouze číst. Pro potřeby možnosti sdílení kalendářů musí portlet umět pracovat se vzdálenými kalendáři pomocí CalDAV protokolu. Díky této funkcionalitě by měla být možná plná synchronizace například se službou Google kalendář, Yahoo kalendář a jinými. Oba typy vzdálených kalendářů musí disponovat manuální možností synchronizace a případně i pravidelnou synchronizací v nadefinovaných intervalech. V neposlední řádě bude kladen důraz na přehledné zobrazení událostí.

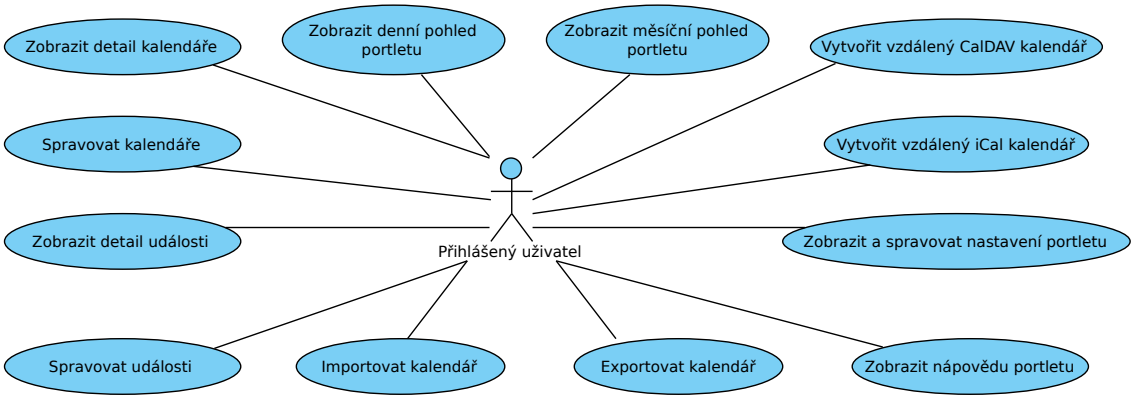
Aby byla dodržena filosofie portletů, tak je důležité, aby nově vytvořený portlet neobsahoval na začátku žádná data a jeden uživatel si tak mohl vytvořit na portálu klidně několik instancí portletů s různými daty. Díky možnosti importu a exportu dat si bude moci uživatel v případě potřeby tyto data přesouvat nebo zálohovat. Instance portletů s osobními daty se budou vkládat na soukromou portletovou nástěnku a nikdo je neuvidí. Naopak skupinové instance portletů se zase budou přidávat na skupinovou stránku apod. Další požadavek na všechny tři portlety bude podmíněné zobrazení obsahu portletů jen přihlášeným uživatelům na portále. Pro možnost sdílení by bylo vhodné, aby byla data pro export přístupná přes webové rozhraní a fungovalo to tedy jako jednoduchá webová služba. Identifikátor zdroje bude vygenerovaný soukromý řetězec, který musí jít v případě potřeby změnit.

4.2 Analýza požadavků

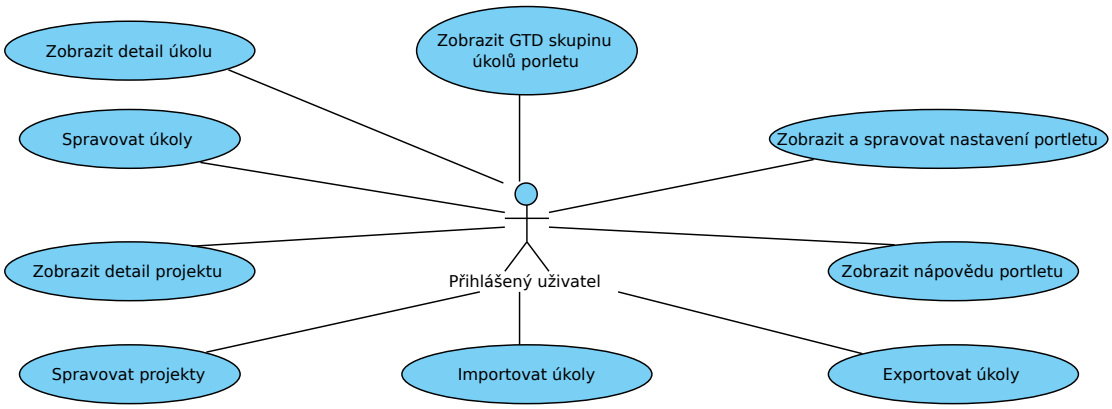
Byly identifikovány tři základní portlety, a to správa kontaktů, správa úkolů a správa kalendáře, které budou implementovány v prostředí portálu. Nyní se postupně podíváme na každý z nich. V následujicím textu budu používat pojem *entita*. Entitou budu nazývat všechny důležité prvky systému, které budou reprezentovány třídou, obsahují nějaké atributy a tato třída se mapuje na tabulku v databázi. Jedná se vlastně o persistentní objekty.



(a) Správa kontaktů



(b) Správa kalendáře



(c) Správa úkolů

Obrázek 4.1: Diagram případů užití

Na obrázku 4.1 je vidět diagram případů užití vytvořený v programu *Visual Paradigm for UML* [38], který byl odvozen z neformální specifikace. Diagram je rozdělen na tři menší diagramy, kde každý řeší funkcionalitu jednoho klíčového portletu (viz. popisek). Všechny diagramy obsahují možnost správy a zobrazení seznamu klíčových entit. Na diagramech je dále uvedena možnost importu a exportu dat a možnost přepnutí do portletových módů nastavení a nápověda. Podívejme se nyní na jednotlivé portlety zvlášť.

Portlet pro správu kontaktů bude obsahovat entity kontakt a skupina kontaktů. Vzhledem k tomu, že některé atributy mohou být ve formě kolekce, tak přibudou entity telefonní číslo, email, adresa, URL, webová adresa a jazyk. Uživatel portletu bude tedy moci přidávat, editovat a mazat jak kontakty, tak skupiny kontaktů. Dále bude mít možnost přiřazovat kontakt ke skupinám kontaktů. Dalším případem užití bude možnost exportu a importu kontaktů. Protože se nacházíme v portletovém prostředí, tak uživatel bude moci editovat nastavení svého portletu a zobrazit nápovědu.

Správa kontaktů bude obsahovat entity úkol a projekt. Uživatel portletu bude moci přidávat, editovat a mazat úkoly a projekty. V rámci editace úkolu bude moci přiřazovat úkoly do GTD složek a úkoly vykonat. Uživatel bude moci také importovat a exportovat úkoly. Stejně jako u předchozího portletu bude moci uživatel editovat nastavení portletu a zobrazovat nápovědu.

Portlet Kalendář bude určitě obsahovat entity událost a kalendář. Vzhledem k tomu, že kalendář může být lokální a vzdálený a vzdálený může být přístupný přes iCalendar soubor nebo přes CalDAV protokol, tak budeme muset entitu kalendář trochu specializovat pro dané případy. Dalším požadavkem je možnost nastavit opakování jednotlivých událostí. I přes to, že konkurenční aplikace, jako například Google kalendář, umožňuje nastavit pouze jeden typ opakování, bude lepší, když opakování budeme modelovat také jako entitu, abychom jich pro danou událost mohli vytvořit případně více. V neformální specifikaci tento požadavek není, ale lepší bude datový model připravit pro případné rozšíření. Uživatel portletu bude moci kompletně spravovat události z editovatelných kalendářů (všech kromě vzdálených pro čtení) a dále samotné kalendáře. Vzdálené kalendáře bude moci uživatel synchronizovat. Dalším případem užití bude import událostí ze souboru do existujícího kalendáře a jeho export do formátu iCalendar. Stejně jako u předchozích portletů bude moci uživatel editovat nastavení portletu a zobrazit nápovědu portletu.

Výběr technologie pro tvorbu portletů Pro samotnou implementaci portletů je možné použít několik přístupů. Základní přístup by byl vyjít pouze z portletové specifikace, která je popsaná v sekci 3.2. Další možností by bylo využít Spring Portletový MVC framework, jehož klíčové vlastnosti byly popsány v sekci 3.5. Poslední přístup, který připadá v úvahu, bude vytvářet portlety pomocí portletového mostu s podporou Java Server Faces, kterou jsme popsali v sekci 3.4.1. Tato technologie nám dovoluje vyvíjet portlet jako každou jinou webovou Java EE aplikaci a přitom nám neupře využít specifických vlastností portálového prostředí. Díky tomuto přístupu můžeme využít technologii JSF, o které se dá říct, že svou použitelností již překonala starší technologii Java Server Pages, která je standardně použita v předchozích dvou variantách vývoje portletů. Díky podpoře JSF budeme moci použít další rozšíření JSF pro podporu AJAXu. V zadání stojí, že uživatelské prostředí má být příjemné a má využívat moderní webové technologie jako AJAX. Ostatní dvě varianty vývoje portletů nemají tak kvalitní nástroj pro podporu AJAX. Na základě porovnání těchto tří možností jsem se rozhodl, že pro návrh a implementaci využiji JSF portletový most od firmy Red Hat, který se nazývá *JBoss Portlet Bridge*. Lehce jsem se o něm již zmínil v sekci 3.4. Pro AJAXovou podporu použiji knihovnu *JBoss Richfaces*, která obsahuje řadu užitečných komponent s podporou AJAXu. Některé z nich popíšu v sekci 5.3.

Výběr knihoven pro podporu iCalendar, CalDAV a vCard Vzhledem k tomu, že existuje několik volně dostupných a prověřených knihoven pro podporu práce s formáty iCalendar a vCard, tak je zbytečné si psát vlastní knihovnu a použiji některou existující. Pro podporu formátu iCalendar jsem se rozhodl použít knihovnu *iCal4j* [16] psanou v jazyce Java. Tato knihovna podporuje sice jen specifikace iCalendar dle RFC 2445 a novější RFC 5545 plně ne, nicméně je ověřená a používaná, proto bude vyhovovat jak pro reprezentaci událostí, tak pro reprezentaci úkolů. Knihovna vyžaduje verzi Javy alespoň 1.4. Knihovna také obsahuje rozšíření pro podporu CalDAV protokolu. Pro podporu formátu vCard jsem se rozhodl použít Java knihovnu EZ-vCard [2]. Existuje i rozšíření knihovny iCal4j pro podporu vCard, ale tato knihovna aktuálně nepodporuje čtení vizitek dle formátu vCard verze 4.0. Pro verzi 4.0 podporuje pouze zápis. Vybraná knihovna EZ-vCard tento nedostatek nemá.

4.3 Návrh portletů

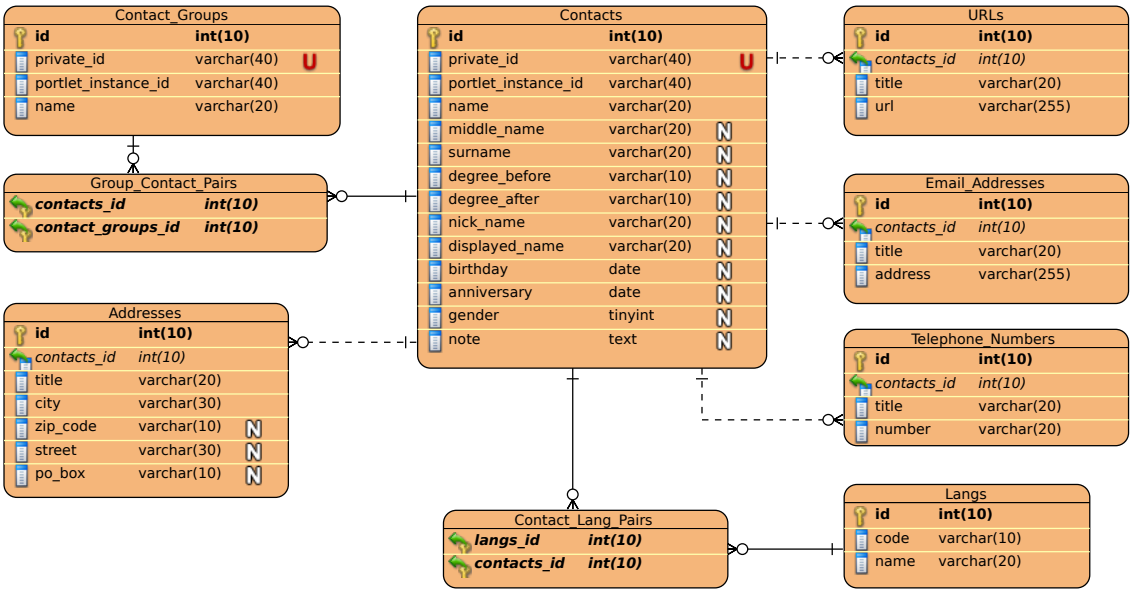
Z analýzy požadavků a identifikování případů užití portletů se nyní dostáváme k návrhu portletů. V prvé řadě se zaměříme na datový model. Ten by měl být totiž na zbytku aplikace nezávislý. Dále si popíšeme navrhnutou strukturu aplikace a v závěrečné části popíšu můj návrh klíčového případu užití, a to synchronizace kalendáře.

4.3.1 Datový model

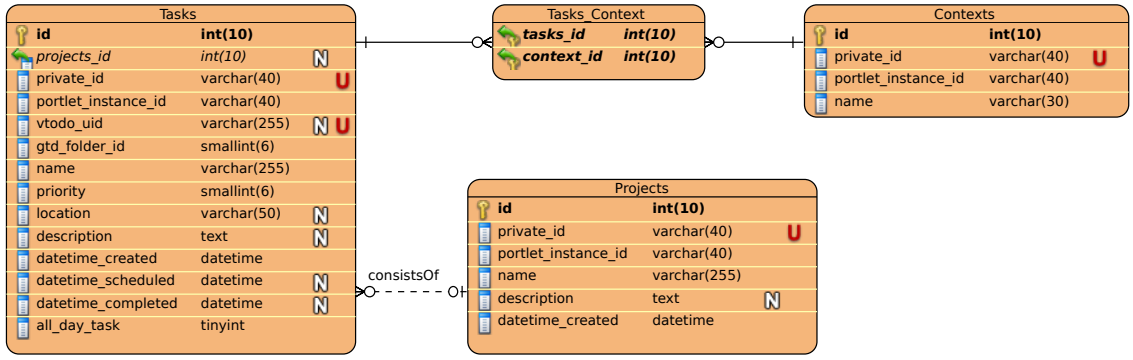
V předchozí sekci jsme si určili ze specifikace, jaké entity se v naší aplikaci objeví. Nyní tedy můžeme přistoupit k datovému modelu naší aplikace. Datový model budeme reprezentovat *entity-relationship diagramem* (ER diagramem, ERD).

Na obrázku 4.2 můžeme vidět tři ER diagramy vytvořené v programu Visual Paradigm for UML[1]. Diagram (a) obsahuje datový model portletu pro **správu kontaktů**. Zaměřme se nyní na dvě ústřední entity – kontakty a skupiny kontaktů. Entity jsou reprezentovány tabulkami `Contacts` a `Contact_Groups` a propojeny vazbou $M : N$. Protože chceme, aby byly všechny data portletově závislé, musíme u těchto entit přidat atribut, který reprezentuje identifikátor portletu. Pojmenoval jsem ho `portlet_instance_id`. Tuto informaci lze zjistit z portletového požadavku metodou `getWindowId()`, která vrací řetězec. Dále si můžeme všimnout, že obě entity obsahují atribut soukromý identifikátor (`private_id`). Tento identifikátor bude hrát roli v přístupu k datům pro export přes webové rozhraní (export dat probereme níže). Z bezpečnostních důvodů totiž není vhodné, aby kdokoli znal skutečný primární klíč záznamu v databázi. Soukromý identifikátor musí být v rámci entity unikátní a v aplikaci by měla být možnost ho změnit. Kontakt bude obsahovat také kolekci telefonních čísel, emailových adres, webových adres a adres bydliště. Tyto vazby budu modelovat jako $1 : N$. Zmíněné entity už nemusí obsahovat atribut `portlet_instance_id`, protože jsou přímo závislé na kontaktu, který tento atribut má. Nakonec se podíváme na entitu jazyky (`Langs`). Entita jazyky je ve vztahu ke kontaktům také ve vztahu $M : N$ podobně jako skupiny kontaktů. Jazyky ale nebudeme modelovat jako portletově závislou entitu. Když přidáme jazyky, budeme totiž chtít, aby se objevily v nabídkách ve všech instancích portletu. Běžný uživatel s entitou jazyky nebude moci manipulovat. Bude pouze moci přidat vazbu svého kontaktu na existující jazyk.

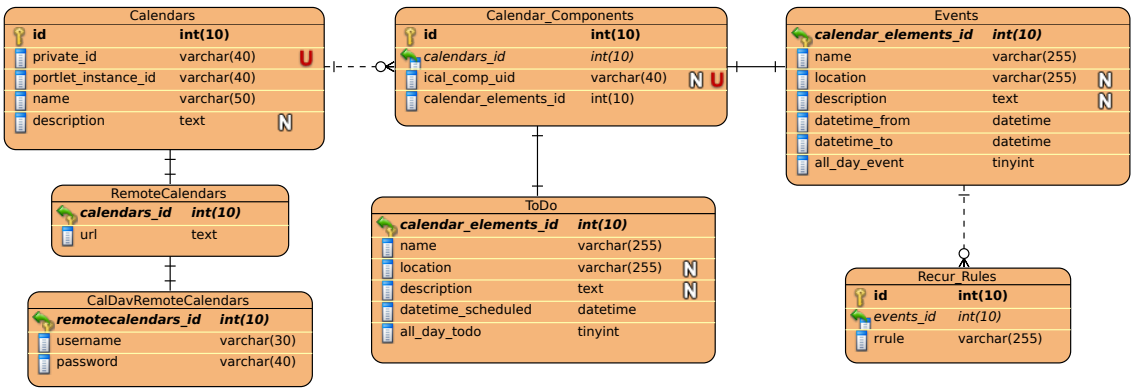
[1] Program Visual Paradigm for UML přidává v ER diagramech cizí klíče jako atributy tabulek, což se v používanější notaci nedělá při grafickém znázornění asociací a jejich kardinalit.



(a) ER diagram pro kontakty



(b) ER diagram pro úkoly



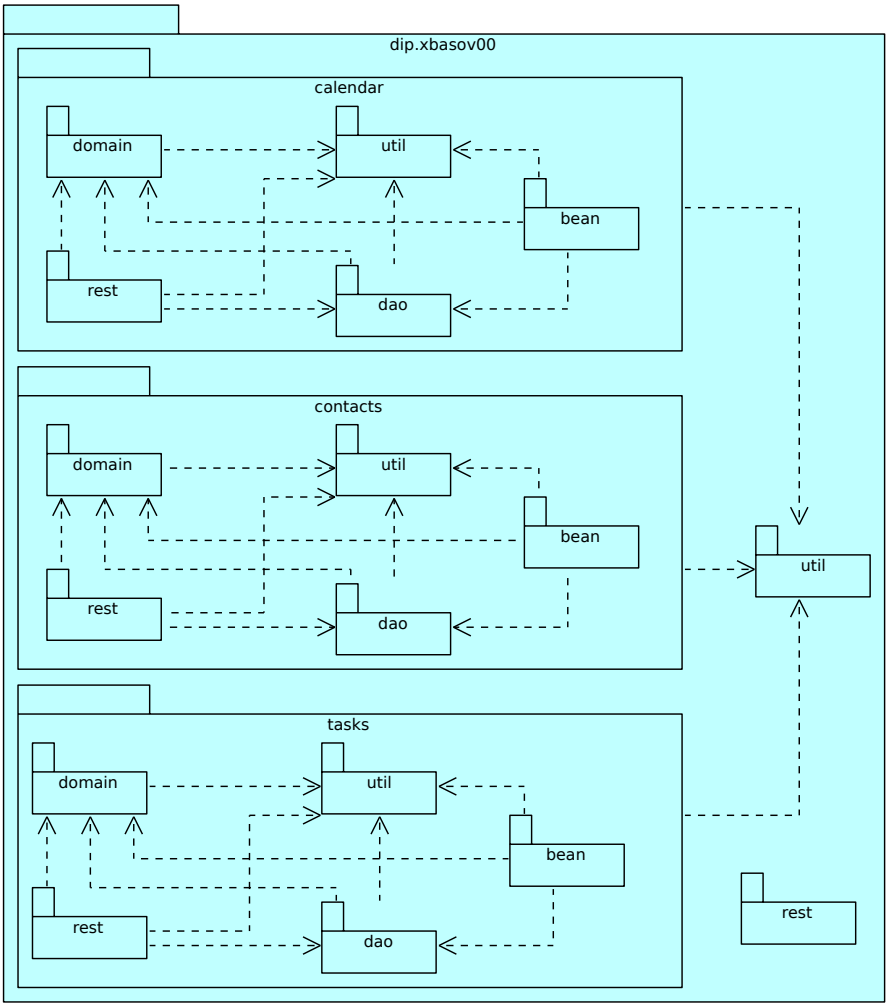
(c) ER diagram pro události

Obrázek 4.2: Entity-relationship diagramy.

Dalším diagramem na obrázku 4.2 je datový model portletu pro **správu úkolů**. I zde se zaměřme nejdříve na entity úkoly (`Tasks`) a projekty (`Projects`). Podobně jako kontakty a skupiny kontaktů budou entity obsahovat atributy `portlet_instance_id` a `private_id`. Mohlo by se zdát, že identifikace instance portletu u úkolu je zbytečná, když se úkol váže na projekt, který tuto identifikaci již má. My ale můžeme vytvářet i úkoly, které k žádnému projektu nepatří, a proto je nutné i úkoly mapovat na instanci portletu. Čtenáře by mohlo napadnout, že můžeme vytvořit pro každý portlet jeden předdefinovaný projekt, kam budou patřit všechny nezařazené úkoly. Tato varianta je také možná, ale dynamické vytváření by nebylo moc průhledné a koncept by byl zbytečně komplikovaný. Entita úkoly bude dále obsahovat povinný atribut `gtd_folder_id`, který bude úkol přiřazovat do některé složky z techniky správy času GTD (viz. sekce 2.1). Hodnota atributu se bude mapovat na výčtový typ v aplikační části programu. Dále můžeme v entitě vidět tři různé datumy. První datum `datetime_created` bude zaznamenávat datum a čas vytvoření úkolu. Druhý datum `datetime_scheduled` bude obsahovat datum a čas naplánovaného začátku plnění úkolu, pokud úkol spadá do GTD složky Kalendář (úkol lze nastavit jako celodenní příznakem `all_day_task`). A nakonec třetí datum `datetime_completed` nám určuje datum a čas splnění úkolu. Slouží i ke zjištění, zda byl úkol již splněn nebo ne. Co se týče entity projekty, tak ostatní její atributy jsou již pochopitelné. Pomocí entity kontexty (`Contexts`) můžeme jednotlivé úkoly řadit do vlastních kategorií. Můžeme třeba vytvořit kontext „Brno“, který pro nás bude znamenat, že úkol můžeme vykonat pouze v Brně. Entita kontexty bude samozřejmě také portletově závislá.

Posledním diagramem na obrázku je datový model portletu pro **správu kalendářů**. Kalendář může obsahovat obecné kalendářové komponenty (inspirace viz. iCalendar). V naší aplikaci budeme modelovat komponenty událost a úkol (`ToDo`). Oba tyto typy dědí od kalendářové komponenty. Komponenta bude obsahovat kromě identifikátoru ještě globální identifikátor, který bude společný pro všechny potomky entity. Mohlo by se zdát, že některé další atributy potomků jsou stejné a můžeme je přidat do rodiče, nicméně z důvodu rozšiřitelnosti jsem nechal tyto atributy duplicitní. Mohla by totiž existovat komponenta žurnál (viz. iCalendar), která třeba atribut lokalitu obsahovat nebude. V aplikaci nemůže existovat kalendářová komponenta, která by nepatřila některému kalendáři, proto budeme mapovat na danou instanci portletu pouze entitu kalendář. Kalendář bude opět obsahovat soukromý identifikátor a ostatní jeho atributy mají význam zřejmý.

Důležitou kolekcí, kterou obsahuje entita událost je kolekce pravidel opakování události. Vzhledem k tomu, že budeme v definici opakování vycházet ze specifikace opakování tzv. RRULE obsažené ve formátu iCalendar, tak se musíme vypořádat s tím, že existuje celá gramatika, jak opakování definovat. Atributy položek výrazu opakování mohou nabývat logicky různých datových typů, což také komplikuje možnost elegantní datové reprezentace v databázi. Proto jsem se rozhodl, že jako hodnotu opakování ve svém datovém modelu budu používat přímo výraz reprezentující opakování (viz. formát iCalendar) a následně si udělám speciální obslužný most, který mi dovolí pracovat objektově s jednotlivými částmi výrazu a následně to opět uloží jako jednu řetězcovou hodnotu. Datový model tak zůstane přehledný a obsluha díky mostu bude také pohodlná. Entita úkol bude mít v této aplikaci spíše sekundární úlohu. Portlet by měl podporovat pouze zobrazení úkolů, které mají nastaven datum vykonání. Portlet tedy nebude obsahovat správu této entity kromě operace smazání. Úkoly se mohou do portletu dostat buď importem ze souboru nebo přes URL vzdáleného kalendáře, který může odkazovat i na portlet Úkoly. Poslední důležitá věc v ER diagramu je datová reprezentace dědičnosti mezi různými typy kalendářů. Existuje entita kalendář a od ní bude dědit entita vzdálený kalendář, která díky svému atributu URL bude moci

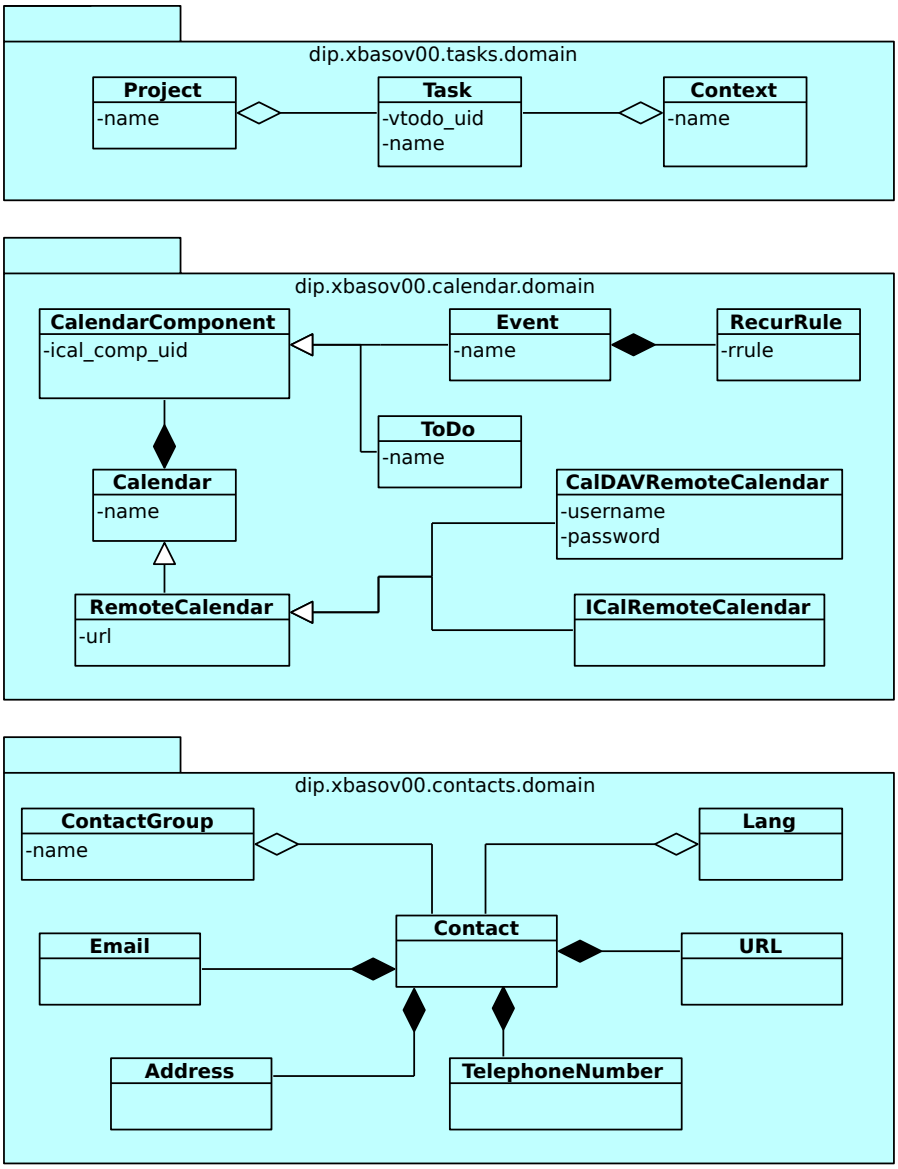


Obrázek 4.3: Diagram balíčků hlavní aplikace

aktualizovat svůj obsah. Nakonec se podívejme ještě na entitu CalDAV vzdálený kalendář, která dědí URL a přidává uživatelské jméno a heslo pro podporu synchronizace přes CalDAV protokol.

4.3.2 Struktura aplikace

Nyní se podíváme na to, do jakých logických celků dekomponujeme naši aplikaci. Na obrázku 4.3 můžeme vidět, že kořenový balík se nazývá `dip.xbasov00`. V tomto balíku se nacházejí tři důležité balíky s jednotlivými portlety. Balíky se postupně jmenují `calendar`, `contacts` a `tasks`. Všechny tři balíky se dále skládají z balíků `domain`, `dao`, `bean`, `util` a `rest`. Balík `domain` bude obsahovat všechny třídy reprezentující identifikované entity zaznamenané také v konceptuálním doménovém diagramu tříd 4.4. Balík `dao`, pojmenován dle návrhového vzoru *Data Access Object*, bude obsahovat rozhraní pro manipulaci s entitami a třídy, které toto rozhraní implementují. Význam je takový, že každá implementující třída může používat jiný mechanismus persistence dat nebo jinou techniku pro vytváření



Obrázek 4.4: Konceptuální diagram tříd

transakcí. V balíku budou také obsaženy JSF konvertory, které zajistí mapování řetězců na objekty a zpět. Řetězcem myslíme například hodnotu HTML reprezentace nějaké grafické výběrové komponenty (např. výběr kalendáře, ke kterému bude patřit událost). Balík `bean` bude obsahovat všechny kontroléry daného JSF portletu. Balík `rest` bude obsahovat třídy implementující RESTovou službu dle JSR-311 [42] (více o této službě naleznete v kapitole 5). Tyto třídy budou zajišťovat mechanismus pro export dat. Poslední balík se jmenuje `util` a ten bude obsahovat další užitečné třídy využívané v balíku. V balíku `dip.xbasov00` se ještě nachází balíky `rest` a `util`, které budou celou aplikací sdíleny. Jejich význam už byl popsán. Mezi jednotlivými balíčky jsou znázorněny vazby závislosti (`<<use>>`), které pro nás budou důležité z hlediska testování aplikace. Změny v konkrétním balíčku mohou

způsobit změny ve všech balíčcích na nich závislých.

V praxi bude aplikace fungovat tak, že prezentační vrstva bude uložena v JSF šablonách (faceletech). Při klasickém nebo AJAXovém požadavku bude požadavek zpracován příslušným kontrolérem. Kontrolér buď obslouží požadavek sám, nebo bude muset pracovat s modelem. K tomu použije příslušnou DAO komponentu. Pokud komponenta vrátí například nějakou entitu pro zobrazení, uložíme si ji do portletového sezení pro další lokální manipulaci a v případě potřeby převedeme entitu na řetězec pomocí již zmíněného konvertoru.

Na následujícím diagramu 4.4 můžeme vidět konceptuální diagram tříd neboli také diagram domény. Na diagramu jsou znázorněny jen důležité doménové třídy a jejich vazby. Zaměřil jsem se zde na tři doménové balíčky z diagramu balíčků 4.3. Diagram má objasnit, v jakém vztahu budou jednotlivé entity z objektově orientovaného pohledu. Význam jednotlivých vztahů je pochopitelný a čtenář si případně může dostudovat jazyk *Unified Modeling Language* (UML) [22], který byl v tomto diagramu použit. Upozorním pouze na rozdíl vazby *agregace* a *kompozice*. Objekty ve vztahu kompozice k celku nemohou existovat samostatně oproti objektům ve vztahu agregace. Kompozice je patrná mezi entitami kalendář a úkol. Naopak agregaci můžeme vidět mezi entitami skupina kontaktů a kontakt. V praxi budeme modelovat jako portletově nezávislé právě ty entity, které nejsou komponentou žádné jiné entity.

4.3.3 Návrh synchronizace kalendáře

Synchronizaci můžeme rozdělit na dva typy. Prvním typem bude synchronizace se vzdáleným kalendářem reprezentovaným souborem ve formátu iCalendar a druhým typem bude synchronizace kalendáře spravovaného kalendářovým CalDAV serverem. První typ kalendáře půjde pouze číst a druhý typ půjde i upravovat, tedy synchronizovat i opačným směrem. Podívejme se na první typ synchronizace. Nejdříve se z URL vzdáleného kalendáře stáhneme daný kalendář ve formátu iCalendar. Dále použijeme knihovnu iCal4j pro převod na objektovou reprezentaci. Objektovou reprezentaci následně musíme převést do na naši vlastní kalendářové reprezentace, se kterou umí pracovat zbytek aplikace. Našim cílem totiž je, aby případné změny knihovny neovlivnily naši aplikaci více než jen v místě převodu cizí objektové reprezentace kalendáře na naši vlastní. Dále budeme procházet komponenty našeho lokálního nesynchronizovaného kalendáře a pro každou komponentu zjistíme, zda existuje komponenta se stejným globálním identifikátorem (viz. atribut UID formátu iCalendar) v právě staženém kalendáři. Pokud ano, tak aktualizujeme atributy komponenty, ale lokální identifikátor musíme zachovat, abychom nepřišli o případné jiné důležité vazby. Pokud jsme globální identifikátor nenašli, tak byla komponenta zřejmě smazána a i my tuto komponentu smažeme. Ve druhé fázi projdeme komponenty staženého kalendáře a komponenty s neznámým globální identifikátor přidáme do naší lokální reprezentace kalendáře. Nyní jsou komponenty našeho kalendáře synchronizovány.

Druhý typ synchronizace bude probíhat obdobně s tím rozdílem, že se nejdříve musíme připojit na vzdálený kalendářový server. K tomu potřebujeme URL, jméno a heslo. Po úspěšném připojení stáhneme vzdálený kalendář a synchronizace bude probíhat obdobně jako v předchozím typu. CalDAV synchronizaci lze ale provádět i opačným směrem. Pokud vytvoříme, smažeme nebo upravíme nějakou kalendářovou komponentu, tak musíme zajistit, že se případné změny projeví i na vzdáleném serveru. K tomu využijeme *iCa4j Connector*, což je rozšíření knihovny iCal4j o podporu standardu CalDAV. Aby naše aplikace byla imunní vůči případný změnám knihovny, tak bude nutné vytvořit nějakého prostřed-

níka, který bude s knihovnou komunikovat. K tomu se výtečně hodí návrhový vzor *adaptér*. Návrhový vzor adaptér vytváří vlastní rozhraní o požadané funkcionalitě. Naše třída, která rozhraní implementuje obvykle obsahuje cizí objekt o cizím rozhraní jako soukromý atribut a volá jednotlivé požadované metody. Zbytek naší aplikace bude pracovat s naším vytvořeným rozhraním, ve kterém bude uložena instance našeho adaptéru.

Vzhledem k tomu, že nechceme po uživateli, aby při každé synchronizaci musel zadávat heslo k jeho vzdálenému kalendáři, musíme si toto heslo uložit. Heslo budeme ukládat do databáze v čisté nešifrované podobě. Z hlediska bezpečnosti je to samozřejmě obrovský problém a pokud by se aplikace měla nasadit pro reálné využití, je nutné implementovat bezpečné uložení hesla v šifrované podobě. Nabízí se použít nějakou symetrickou šifru. Jaký ale použít klíč? Navrhuji použít uživatelské heslo pro přihlášení do portálu jako klíč. K tomuto heslu se lze dostat přes následující programovou konstrukci:

```
ConversationState.getCurrent().getAttribute(Credentials.CREDENTIALS)
```

Pro použití zmíněné konstrukce je nutné přidat do projektu závislost na knihovnu `exo.core.component.security.core` do souboru `pom.xml`. Pro šifrování by se potom dala použít symetrická šifra *Advanced Encryption Standard* (AES)[2], která je považovaná za bezpečnou. Zmíněný návrh by fungoval pro porlety, ke kterým může přistupovat pouze jeden uživatel, protože on jediný by svým portálovým heslem mohl dešifrovat svá hesla na CalDAV servery. Jakým heslem ale šifrovat u portletů na skupinových stránkách, kde má každý uživatel jiné heslo? Tuto otázku zanechám otevřenou. Můžeme samozřejmě zvolit jako heslo například jméno skupiny, ale algoritmus by byl pomocí reverzního inženýrství[3] dohledatelný a jméno je známé. Pro potřeby naší práce nebudeme tento problém dále řešit, ale je nutné si uvědomit, že je to zranitelné místo aplikace.

[2] http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[3] http://en.wikipedia.org/wiki/Reverse_engineering

Kapitola 5

Implementace

V této kapitole se zaměřím na implementační detaily, které byly důležité při vývoji portletů. Vzhledem k rozsahu práce není možné probírat všechny části implementace podrobně, proto musíme zvolit alternativní přístup popisu. Sekce 5.1 bude obsahovat princip implementace portletů. Zaměřím se zejména na význam použití jednotlivých probraných technologií a princip jejich provázání. Sekce 5.2 bude dále popisovat implementační detaily těch nejzajímavějších částí projektu jako například implementaci exportu a importu dat nebo synchronizaci kalendáře. Poslední sekce 5.3 rozebere implementované grafické uživatelské rozhraní.

5.1 Implementační detaily

Hlavním implementačním jazykem této práce byla Java Enterprise Edition verze 6 [44]. Pro následující text předpokládám, že si čtenář podrobně nastudoval sekci 3.6. Celý projekt byl spravován nástrojem Maven verze 3.0.4 (použít minimálně 3.0.0) a použitá verze Javy byla 1.6.0_18 (použít minimálně 1.6.0). Projekt má všechna důležitá nastavení v souboru `pom.xml`. Projekt dědí důležitá nastavení od projektu `org.jboss.portletbridge:examples:3.2.0.Final`[1], což je projekt zaštiťující vzorové portlety aktuální pojmenované verze portletového mostu 3.2.0.*Final*. Další důležitá portletová nastavení naleznete v konfiguračním souboru `portlet.xml`. Upozorním v tomto souboru na nastavený portletový filtr s názvem `PortletCDIFilter`[2], který zajišťuje podporu technologie CDI v prostředí portletového mostu. Filtr musí být namapován na všechny tři implementované portlety.

Nyní zkusím krátce popsat, jaké Java EE technologie jsem použil, a k čemu. Nejdůležitější roli zde sehrála technologie Java Server Faces verze 2.1, která tvoří kostru celé aplikace. Jednotlivé JSF facelety jsou tvořeny `XHTML` soubory. Nastavení pro podporu `XHTML` souborů naleznete v konfiguračním souboru `web.xml`. Spolu s JSF byl použit i framework RichFaces verze 4.3.1.*Final* pro podporu AJAXových komponent. Další důležitá nastavení jako například podporu vícejazyčnosti lze dohledat v konfiguračním souboru `faces-config.xml`. Sada portletů plně podporuje češtinu a angličtinu a jazyk lze dynamicky přepínat v administraci portálu. Pro objektově relační mapování se použila technologie Java Persistence API verze 2.0, která v tuto chvíli používá HSQL databázi [21]. Data jsou ukládána do souboru `dip-xbasov00.h2.db`, který je uložen na portále GateIn a uživatel tedy před použitím ne-

[1] https://github.com/portletbridge/portletbridge/blob/3.2.0.Final/examples/pom.xml
[2] https://docs.jboss.org/author/display/PBR/CDI+Portlet+Development

musí nic nastavovat. Datový zdroj je definován v konfiguračním souboru `persistence.xml`, který odkazuje na soubor `dip-xbasov00-ds.xml`, v němž už jsou uloženy konkrétní detaily datového zdroje. Pro podporu transakcí se použila technologie Enterprise Java Beans 3.1. Třídě využívající tuto technologii se říká často *java bean*. Každá metoda takových tříd se vykonává jako jedna transakce. Vzhledem k tomu, že všechny java beany jsou bezestavové, mohl jsem u každé z nich použít anotaci `@Stateless`. Správu závislostí mezi jednotlivými třídami zajišťuje technologie CDI a její klíčové anotace `@Named` a `@Inject`. K podpoře technologie jsme museli využít již zmíněný portletový filtr. Poslední použitá technologie z balíku Java EE je JAX-RS verze 1.1. Byla využita pro podporu RESTového rozhraní exportu dat.

Zdrojové soubory následně nástroj Maven překládá do archívu `dip-xbasov00.war`, který lze přímo nahrát na GateIn portál. Portlety byly testovány na nejnovější verzi GateIn portálu $3.6.0.Beta01$[3], ale měly by fungovat i na starších verzích $3.5.0.x$.

Zdrojové soubory byly podrobeny statistickým výpočtům programu `sloccount` [50]. Projekt obsahuje celkem 5771 zdrojových řádků v jazyce Java. Zajímavou statistikou by bylo spočítat počet řádků v `XHTML` souborech, ale to nástroj bohužel neumí, protože `XHTML` soubory nezapočítává do svých `XML` statistik.

5.2 Důležité případy užití a funkcionalita

V této sekci bych chtěl popsat některé důležité případy užití a zajímavou funkcionalitu. Nejprve popíšu způsob, jakým řeším agregaci časových údajů ze všech portletů v kalendáři a možnost opakování událostí. Následuje důležitá část o exportu a importu dat. S tím souvisí i synchronizace, na kterou se podíváme vzápětí. Dále poukáži na využití portletového filtru ke kontrole přístupu přihlášených uživatelů. Důležitou částí této sekce bude i řešení problému využití jednoho portletu vícekrát jedním uživatelem na jedné portálové stránce, pokud si potřebuje ukládat data do sezení (angl. *session*) aplikace. Nakonec popíšu, jak jsem ve svých portletech využil lokalizaci a portletovou personalizaci.

5.2.1 Agregace časových údajů v portletu kalendář

Tato sekce jenom krátce vysvětlí jednoduchý princip sdílení časových údajů z různých portletů v Kalendáři. Základní požadavek je takový, že portlet Kalendář musí importovat data ze souboru s formátem iCalendar. Zobrazení úkolů je snadné. Formát iCalendar obsahuje komponentu `VTODO` a portlet Úkoly právě tyto komponenty umí produkovat. Stačilo tedy vytvořit reprezentaci úkolu v portletu Kalendář a dále přidat předka (typ `CalendarComponent`) k typu `Event` a `ToDo`. Objekt kalendář potom obsahuje kolekci těchto komponent a umí s nimi manipulovat jednotně. Vzhledem k tomu, že kalendář není primárně určen ke správě úkolů, tak umí úkoly pouze zobrazovat a v případě potřeby mazat.

Dále jsem řešil problém, jak můžeme do spolupráce portletů zařadit i portlet Kontakty. Napadlo mne řešení, že Kontakty budou umět exportovat data narození a výročí uživatelových kontaktů jako události s ročním opakováním ve formátu iCalendar. Při sdílení těchto dat kalendářem se zajistí, že změnou nebo přidáním nějakého kalendářového údaje do kontaktů se data zobrazí i v Kalendáři.

[3] https://github.com/gatein/gatein-portal/tree/3.6.0.Beta01

```
1 RecurRule rrule = new RecurRule();
2 rrule.setFrequency("WEEKLY");
3 rrule.setInterval(2);
4 rrule.setCount(10);
5 rrule.setDayList(new HashSet<String>(Arrays.asList("TU", "TH")));
```

Ukázka 5.1: Příklad reprezentuje opakování, které se dá dle formátu iCalendar zapsat následovně „`RRULE:FREQ=WEEKLY;INTERVAL=2;COUNT=10;BYDAY=TU,TH`“.

5.2.2 Opakování událostí

V kalendáři byla implementována možnost zadání událostí s opakováním. Nastavení opakování je téměř plně kompatibilní s aplikací Google Kalendář 2.3. Moje aplikace ovšem neumožňuje individuální změnu výskytu události v sérii opakování. V návrhu byl objasněn důvod, proč ukládám nastavení opakování ve formě řetězce, který reprezentuje větu, dle gramatiky opakování formátu iCalendar. Pro snadnou manipulaci s řetězcem byl vytvořen speciální most – třída `RecurRule`. Třída `RecurRule` obsahuje atribut `rrule`, který reprezentuje dané opakování události. Při každé editaci rekurence se vytvoří objekt typu `net.fortuna.ical4j.model.Recur`, který jako atribut umí vzít daný řetězec. Pokud proběhne interpretace řetězce v pořádku, tak se daný objekt naplní a třída `RecurRule` s ním může začít manipulovat. Před každou úpravou je potřeba vytvořit si tuto dočasnou objektovou reprezentaci a na konci metody se musí vygenerovat zpět řetězec, který se uloží do atributu `rrule`. Třída `RecurRule` má funkcí návrhového vzoru *adaptér* ke knihovně *iCal4j*, abychom s ní nemuseli pracovat přímo.

V ukázce 5.1 lze vidět, jak snadno se vytvoří opakování události (řádek 1), která proběhne celkem desetkrát (řádek 2) ve dnech úterý a čtvrtek (řádek 2 a 5) každý druhý týden (řádek 3). Dle formátu iCalendar by se podobných pravidel pro jednu událost mohlo zapsat libovolně mnoho. Žádný ze zkoumaných nástrojů pro správu času ale tuto vlastnost neumí, proto jsem z hlediska synchronizace nadstandardní možnost nepřidával. Rozšíření by ale bylo velice snadné. Stačilo by změnit JPA anotaci z `@OneToOne` na `@ManyToOne` a lehce upravit API. Pro uživatele by pak ale už mohlo být nastavení složité a nepřehledné.

5.2.3 Export a import dat

Nyní se podíváme na způsob, kterým jsem implementoval v portletech export a import dat. Entity, které je možno exportovat jsou kalendáře, úkoly a kontakty včetně skupin kontaktů. Dané entity obsahují vždy metodu, která převede mojí reprezentaci objektů na objekty použitých knihoven podporujících převod na řetězec ve formátu iCalendar nebo vCard. V sekci 4.3 bylo uvedeno, že entity pro export budou obsahovat atribut `private_id`. Tento atribut bude sloužit jako identifikátor pro přístup k datům přes RESTové rozhraní. Atribut jsem vygeneroval pomocí třídy `java.util.UUID`, která umožňuje generovat globálně jednoznačné identifikátory založené na časovém razítku podle RFC 4122 [41].

Moje aplikace obsahuje dále třídu `dip.xbasov00.rest.JaxRsActivator` s třídní anotací `@ApplicationPath`, která dědí od třídy `javax.ws.rs.core.Application`, což reprezentuje „ne-XML“ přístup jak aktivovat technologii JAX-RS s podporou RESTového rozhraní. Zdroje jsou pak dostupné relativně k cestě servletu a definované v řetězcovém parametru anotace `@ApplicationPath`. Nyní už jsem mohl vytvořit konkrétní RESTové třídy pro každý portlet. Pro portlet kalendář

```
1  @GET
2  @Path("{privateId}")
3  @Produces("text/Calendar")
4  public Response getICalendar(@PathParam("privateId") String privateId)
5      throws CalendarException {
6
7      Calendar calendar = calendarDao.getCalendarFromPrivateId(privateId);
8      if (calendar == null) {
9          return Response.status(Status.NOT_FOUND)
10             .type(MediaType.TEXT_HTML).entity("Not found!").build();
11     } else {
12         net.fortuna.ical4j.model.Calendar vcalendar =
13             calendar.getVCalendar();
14         return Response.ok(vcalendar.toString())
15             .header("Content-Disposition",
16                 "attachment; filename=calendar.ics").build();
17     }
18 }
```

Ukázka 5.2: Příklad zkrácené metody umožňující export dat přes RESTové rozhraní.

```
1  <rich:fileUpload
2      id="upload"
3      fileUploadListener="#{calendarController.filesUploadlistener}"
4      acceptedTypes="ical, ics, ifb, icalendar"
5      ontyperejected="alert('#{text.SupportedICalFilesDesc}');"
6      maxFilesQuantity="3">
7      <a4j:ajax event="uploadcomplete" render="calendarView"/>
8  </rich:fileUpload>
```

Ukázka 5.3: Příklad definice tagu `<rich:fileUpload>` z knihovny RichFaces pro ajaxový import souborů.

existuje třída `dip.xbasov00.calendar.rest.CalendarREST` obsahující třídní anotaci `@Path("/calendars")`. Ta nám udává, že po první částí relativní URL s definovaným jménem `rest` bude další část `calendars`. Třída potom obsahuje jako atribut objekt implementující rozhraní `dip.xbasov00.calendar.dao.CalendarDAO` a dále metodu `getICalendar()`, na které si vysvětlíme princip, který funguje i u ostatních RESTových tříd.

Na ukázce 5.2 vidíme zkrácenou metodu pro export kalendáře. Metoda obsahuje několik anotací. Anotace `@GET` (řádek 1) umožňuje přístup k metodě přes HTTP GET požadavek. Anotace `@Path("privateId")` (řádek 2) obsahuje další část relativní URL, přes kterou přistoupíme ke zdroji. Části ve složených závorkách definují proměnné. V tomto případě zde je soukromý identifikátor, který se dále namapuje na atribut metody anotací `@PathParam("privateId")` (řádek 4). Na řádku 7 získáme kalendář dle jeho identifikátoru. Pokud kalendář nebyl nalezen kvůli chybnému identifikátoru, tak vrátíme HTML odpověď s kódem 404, který znamená, že zdroj nebyl nalezen. Tuto odpověď vygenerujeme díky na-

stavení `Status.NOT_FOUND` (řádek 9). Třída `Response` implementuje návrhový vzor builder a nakonec tedy musíme zavolat metodu `build()`. Pokud se kalendář podařilo najít, tak jeho metodou `getVCalendar()` vrátíme objekt, který umí vytvořit výstup dle formátu iCalendar. Nyní vracíme HTTP odpověď s kódem 200 (zajisti metoda `ok()`) (řádek 14), která značí, že vše proběhlo v pořádku a jako přílohu připojíme náš vyexportovaný kalendář. Příloha se nastavuje v hlavičce odpovědi, což můžeme vidět na řádcích 15 a 16. Význam anotace `@Produces("text/Calendar")` (řádek 3) nám zajistí patřičný typ obsahu *angl. content type* dle standardu MIME. Tvar URL pro získání kalendáře s daným soukromým identifikátorem může být následující:
`<protocol>//<host>:<port>/<app-name>/rest/calendars/<private-id>`

Pro import entit používám komponentu `<rich:fileUpload>` [43] z knihovny RichFaces. Komponenta umožňuje AJAXový upload souborů na server. Import vysvětlím na ukázce 5.3. Element obsahuje atribut `fileUploadListener` (řádek 3), kde je odkaz na metodu kontroléru, která zpracuje načtení souboru (*upload*). Atribut `acceptedTypes` (řádek 4) obsahuje povolené typy přípon a atribut `ontyperejected` (řádek 5) JavaScriptovou akci, která se provede při nepovoleném formátu. V mém případě vyskočí varovné okno s lokalizovanou zprávou. Dále můžeme v elementu nastavit maximální počet souborů pro nahrání. Element pak obsahuje vnořený element `<a4j:ajax>`, který zajistí překreslení panelu s identifikátorem „calendarView“ po dokončení nahrávání.

5.2.4 Synchronizace událostí

V sekci 4.3.3 byl vysvětlen princip, jakým bude probíhat synchronizace. Rozhodl jsme se, že pro potřeby CalDAV synchronizace použiji návrhový vzor adaptér. Aplikace tedy obsahuje rozhraní `CalDAVConnector` a implementuje ho naše třída `ICal4JConnector`. Podívejme se na to, jak funguje připojení na vzdálený sever. Každý kalendářový server může přijímat URL požadavky s lehce jinou strukturou. To je ale obecně problém, protože knihovna zajišťující synchronizaci pak neví, jak přesně požadovanou URL vytvořit (bavíme se i o URL s funkci vytváření, editace a mazání komponent na vzdáleném serveru). Knihovna iCal4j konektor vytváří množinu typů objektů implementující rozhraní `PathResolver`. Nazývejme tedy pro potřeby této práce dané podtypy jako „procházeče“. Procházeče mají za úkol se s problémem vypořádat. Pro synchronizaci s *Google* kalendářovým serverem se musí použít procházeč typu `GCalPathResolver` naopak pro synchronizaci s *Yahoo* kalendářový server se využívá procházeč typu `CalendarServerPathResolver`. Celkem obsahuje knihovna aktuální verze 0.9.3 jedenáct typů procházečů a jeden obecný, který si může uživatel vytvořit na míru. Vzhledem k tomu, že knihovna neumí vybrat z podporovaných procházečů ten správný, tak jsem se rozhodl všechny postupně projít a zkoušet. V momentě kdy narazím na špatný procházeč, který neumí transformovat požadavek pro potřeby serveru, tak zachytím výjimku s oznámením chyby připojování a přecházím na další procházeč. Pokud nastane situace, že nefunguje ani jeden a bylo by případně potřeba vytvořit typ na míru, musím uživateli oznámit chybu synchronizace.

Při implementaci jsem v knihovně iCal4j konektor verze 0.9.3 nalezl chybu ve třídě `CalDavCalendarCollection.java` a její metodě `removeCalendar(String uid)` (řádek 455). Chyba spočívá v tom, že vytváří požadavek na smazání chybným způsobem (viz. následující ukázka). Nicméně metoda `getPath()` může vracet řetězec končící znakem '/' a URL pak bude obsahovat dva znaky lomeno vedle sebe, což není v tomto místě URL povoleno.

```
DeleteMethod deleteMethod = new DeleteMethod(getPath() + "/" + uid + ".ics");
```

Naštěstí se tato metoda nevolala nikde uvnitř třídy, ale je veřejná a volá se pouze z

venku. Mohl jsem si tedy vytvořit tělo metody s opravenou chybou. Opravenou část metody můžete vidět níže. Záznam nalezené chyby jsem odeslal na *bugtracker* této knihovny[4].

```
String path = calCollection.getPath();
if (!path.endsWith("/")) {
    path = path.concat("/");
}
String str = path + uid + ".ics";
DeleteMethod deleteMethod = new DeleteMethod(str);
```

5.2.5 Kontrola přihlášení

Portály mohou obsahovat stránky, které jsou přístupné bez nutnosti přihlášení. I na těchto stránkách ale můžeme mít portlet, který chceme mít přístupný jenom pro přihlášeného uživatele. Tento problém jsem se snažil vyřešit. Jako nejlepší místo kontroly přihlášení jsem zvolil portletový filtr. Portletové filtry jsme si už probrali v kapitole o technologiích portálů a portletů v sekci 3.3.1. Každý požadavek prochází definovanými portletovými filtry ještě před tím, než přijdou na řadu klasické životní fáze portletu. Náš filtr tedy zamezí přístupu nepřihlášeným uživatelům ještě před tím, než získá kontrolu nad portletem portletový most. Třída `UserLoggedFilter` v balíčku `dip.xbasov00.util` reprezentuje právě tento filtr.

Na ukázce 5.4 můžeme vidět, že filtr implementuje metody rozhraní `RenderFilter`. Zaměříme se pouze na metodu `doFilter`, protože ostatní implementované metody mohou mít prázdný obsah. Na ukázce jsou reprezentovány třemi tečkami. Metoda `doFilter` obsahuje větvení na základě přihlášenosti uživatele (řádek 7). Pro nepřihlášeného uživatele vrací výraz hodnotu `null` a my vypíšeme chybu do těla portletu a nepředáme řízení dál. K výpisu můžeme použít objekt typu `PrintWriter`, který je obsažen v parametru `response` (řádek 8). Pokud je uživatel přihlášen, tak předáme řízení na další filtry v řetězci filtrů (řádek 12), kde na konci bude samotný portlet, který v našem případě předá kontrolu portletovému mostu. Po provedení se metoda ukončí.

5.2.6 Použití více instancí jednoho portletu jedním uživatelem

Instance implementovaných portletů jsou na sobě úplně nezávislé. Aby tento princip mohl v prostředí technologie JSF fungovat, musel jsem vyřešit několik problémů. První problém představovaly vazby klíčových entit na danou instanci portletu. Klíčovou entitou myslím takovou entitu, která není ve vazbě kompozice na jinou entitu (viz. digaram 4.4). Tyto entity budou obsahovat atribut `portlet_instance_id`. Na základě tohoto atributu budu potom entity filtrovat v požadavcích na databázi.Následně bylo potřeba vyřešit, aby každý uživatel měl vlastní prostředí a tedy vlastní instance požadovaných komponent. To se zajistí jednoduše přidáním anotace `@SessionScoped` ke každé takové komponentě. Poslední problém spočíval ve vyřešení otázky, jak zajistit, aby mohl mít uživatel více instancí jednoho portletu na portále a komponenty daného sezení byly portletově závislé. Jednoduše chceme, aby každá komponenta byla jednoznačně definovaná identifikátorem sezení a identifikátorem portletu. Bohužel neexistuje žádná anotace s názvem `@PortletScoped`, takže jsem si musel tuto funkcionalitu dodělat ručně. Princip spočívá v tom, že portlety mají dva typy sezení. Prvním typem je aplikační sezení, což je klasické sezení ve webových aplikacích, a druhým typem je portletové sezení, které má každý portlet vlastní. Úkolem

[4] http://sourceforge.net/tracker/?func=detail&atid=646395&aid=3613111&group_id=107024

```
1  public class UserLoggedFilter implements javax.portlet.filter.
       RenderFilter {
2      ...
3      @Override
4      public void doFilter(RenderRequest request, RenderResponse response,
5              FilterChain chain) throws IOException, PortletException {
6
7          if (request.getRemoteUser() == null) {
8              PrintWriter out = response.getWriter();
9              out.println("Log in first, please!");
10         }
11         else {
12             chain.doFilter(request, response);
13         }
14     }
15 }
```

Ukázka 5.4: Portletový filtr pro kontrolu přihlášení uživatele.

tedy bylo zajistit namapování všech potřebných atributů komponent s anotací `@SessionScoped` na sezení portletové. Odstranil jsem tedy samotné atributy těchto komponent a přidal jenom ukládací metody (*settery*) a načítací metody (*gettery*) pro práci s portletovým sezením. Data musejí mít jednoznačný identifikátor. Identifikátor jsem zvolil ve tvaru „`<nazev-třídy>:<název-proměnné>`“.

Na ukázce 5.5 můžeme vidět dvě důležité metody, která zajišťují zmíněné mapování proměnných do portletového sezení. První metoda zajišťuje načítání proměnných. Na řádku 3 nejdříve získáme sezení portletu. Atribut metody `getSession` určuje, zda se má nové sezení vytvořit, pokud neexistuje. Na řádku 7 si ověřím, že sezení implementuje portletové rozhraní a následně ho přetypuji. Dále již načítám atribut dle jména s dodatečným atributem `PortletSession.PORTLET_SCOPE`, který zajišťuje načítání právě z portletového sezení. Prototyp metody navic obsahoval typ, kterého by měl objekt být. Na řádku 11 tento typ kontroluji a následně objekt přetypovávám a výsledek vracím. Pokud nastal nějaký problém, tak metoda vrací hodnotu `null`. Ukládací metoda je jednodušší. Načte si dané portletové sezení (řádek 21) a následně do něj uloží objekt s daným klíčem (řádek 26).

5.2.7 Lokalizace a personalizace

Implementovaná sada portletů podporuje vícejazyčné prostředí. Plně implementované jazyky jsou angličtina a čeština. Jazyky lze přepínat v nastavení použitého portálu. Podporované jazyky musí být deklarovány v konfiguračním souboru portletu (`portlet.xml`) i v konfiguračním souboru technologie Java Server Faces (`faces-config.xml`). V portletovém konfiguračním souboru jsou použity dvě značky `<supported-locale>` uvnitř tagu `<portlet>` s hodnotami „en“ a „cs“. V konfiguračním souboru JSF je kromě deklarace podporovaných jazyků navíc uvedena cesta ke zdroji překladů. V mém případě hodnota „`dip.xbasov00.text`“ znamená, že názvy souborů s překlady budou ve tvaru „`text_<lang>.properties`“ a budou umístěny ve složce s cestou „`src/main/resources/dip/xbasov00`“. Ve *faceletech* (JSF šablonách) už stačí jen použít konstrukci ve tvaru „`#{text.<klíč>}`“, kde klíčem se myslí pojmenovaný klíč pro překlad. Pro přístup k

```
// Načítání objektů z portletové session
public static Object getPortletSessionAttribute(String attrName,
    Class<?> className) {
    Object objSession = FacesContext.getCurrentInstance()
        .getExternalContext().getSession(false);

    if (objSession instanceof PortletSession) {
        PortletSession portalSession = (PortletSession) objSession;
        Object o = portalSession.getAttribute(attrName,
            PortletSession.PORTLET_SCOPE);
        if (className.isInstance(o)) {
            return className.cast(o);
        }
    }
    return null;
}sec:impl:gui

// Ukládání objektů do portletové session
public static void setPortletSessionAttribute(String attrName,
    Object object) {
    Object objSession = FacesContext.getCurrentInstance()
        .getExternalContext().getSession(false);

    if (objSession instanceof PortletSession) {
        PortletSession portalSession = (PortletSession) objSession;
        portalSession.setAttribute(attrName, object,
            PortletSession.PORTLET_SCOPE);
    }
}
```

Ukázka 5.5: Mapování objektů na portletovou session.

```
public static ResourceBundle getBundle() {
    String basename = "dip/xbasov00/text";
    Locale locale = FacesContext.getCurrentInstance()
        .getViewRoot().getLocale();
    return ResourceBundle.getBundle(basename, locale);
}
```

Ukázka 5.6: Přístup k překladům přímo z aplikace

překladům ze samotné aplikace jsem použil speciální metodu ve třídě `Resources.java`, kterou můžeme vidět na ukázce 5.6. Na řádku 2 si nejdříve definujeme relativní cestu ze složky `resources` k našemu zdroji překladů. Na řádku 3 zjistíme aktuální lokalizaci naší aplikace a na řádku 5 již vracíme objekt typu `ResourceBundle`, ke kterému lze přistoupit odkudkoliv, protože se jedná o statickou metodu. Použitím metody `containsKey(key)` a `getString(key)` pak získáme lokalizovaný překlad.

V portletu pro správu kalendářů byla dále použita personalizace (viz. sekce 3.3.2). Vzhledem k tomu, že v našich podmínkách je běžné, že týden v kalendáři začíná pondělím, ale v anglicky mluvicích zemích nedělí, použil jsem nastavení, kde se každý uživatel bude moci nastavit variantu, která mu vyhovuje. Individuální nastavení předpokládá, že portlet bude umístěn v sekci nástěnka (angl. Dashboard). V ostatních portletech mne nenapadlo vhodné a hlavně smysluplné nastavení, proto jsou tyto pohledy nechal prázdné.

5.3 Grafické uživatelské rozhraní

V této sekci bude blíže představeno navržené a implementované grafické uživatelské rozhraní (angl. Graphical User Interface, GUI). Mým cílem bylo vytvořit takové grafické uživatelské rozhraní, které by bylo velice jednoduché, ale přitom přehledné a aby vzhled všech portletů byl uniformní. Vzhledem k tomu, že portlety jsou tak trochu předurčeny k tomu, že jich bude na jedné portletové stránce víc, je důležité, aby rozvržením svých komponent a barvami nebily uživatele do očí. Dále jsem si stanovil požadavek, aby nebylo rozvržení šířek jednotlivých komponent příliš fixní. Portletová stránka může být rozložena na sloupce. Sloupce lze dynamicky přidávat a odebírat v editaci portálové stránky. Je důležité, aby portlet vypadal dobře pokud jeho šířka zabírá polovinu portálové stránky i celou šířku. Pokud stránku rozdělíme na čtyři sloupce, tak už se pochopitelně portlet stane nepřehledný, protože na tak malém prostoru jej nelze efektivně ovládat. Dva sloupce při standardním rozlišení ale musí zvládat s přehledem.

Většina portletových nastavení probíhá ve vyskakovacích (angl. *pop-up*) oknech. Věřím, že je potom ovládání pro uživatele přehlednější a pohodlnější zvlášť v prostředí portálů. Efekt „vyskočení“ modálního okna se zastíněním zbytku stránky drží uživatelovu pozornost na daném formuláři. Navíc každý formulář může být jinak velký a fixní šířka portletu nevypadá potom nejlépe. Vhodnější je vyskakovací okno s proměnlivou šířkou.

Pro potřeby vyskakovacích oken se zastíněním stránky jsem použil ajaxovou komponentu z knihovny RichFaces[5]. Komponenta se zapisuje tagem `<rich:popupPanel>`. Jmenný prostor „rich“ je dostupný na adrese `http://richfaces.org/rich`. Komponenta obsahuje atribut `modal`, jehož boolovská hodnota `true` zajistí vytvoření modálního okna. Další důležitý atribut je `autosized`, jehož boolovská hodnota `true` zase zajistí, že velikost vyskakovacího okna se přizpůsobí obsahu. Abychom docílili dynamického ajaxového přepínaní mezi vyskakovacími okny, vytvořil jsem komponentu viditelnou na příkladu 5.7. Tuto komponentu nyní popíšu, protože z ní vychází celý koncept návrhu mého uživatelského rozhraní. Většinu portletových pohledů totiž zobrazuji právě ve vyskakovacím okně.

Na řádku 12 vidíme definici vyskakovacího okna s již popsanými atributy. Hlavička (řádek 13 – 17) obsahuje titulkový pruh okna, jehož text bereme z kontroléru `popupManagerCalendarBean`. Komponenta pro výpis textu je obalena komponentou `<a4j:outputPanel>` z knihovny RichFaces, která slouží k AJAXovému překreslování obsahu. Stejnou konstrukcí je i obaleno tělo okna (řádek 12-16), které obsahuje komponentu `<ui:include>`, pomocí níž budeme natahovat do šablony potřebné obsahy JSF faceletů. Tělo ještě obaluje komponenta reprezentující formulář pro správné fungování překreslení. Kdykoliv budeme chtít akčním tlačítkem otevřít AJAXově vygenerované okno, použijeme například akční komponentu `<a4j:commandButton>` (opět z knihovny RichFaces), která zajistí, že požadavek bude AJAXový. V rámci akce musíme nastavit příslušný pohled do vyskakovacího okna a jeho titulkový pruh (řádek 5). Dále musíme překreslit vyskakovací okno (řádek 6) a nakonec okno

[5] http://docs.jboss.org/richfaces/latest_4_X/vdldoc/

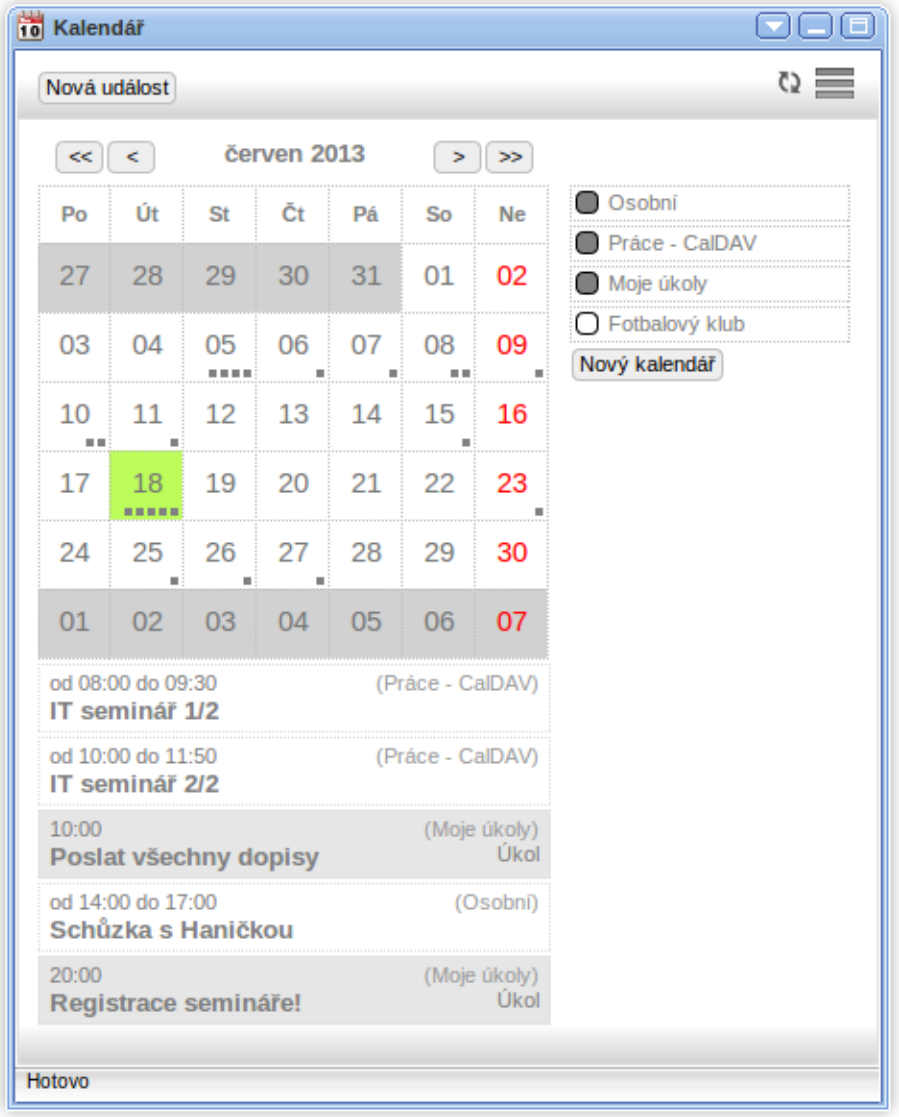
```
1  <!-- Otevření vyskakovacího okna s příslušným pohledem -->
2  <h:form>
3      <a4j:commandButton
4          value="#{text.NewEvent}" title="#{text.NewEvent}"
5          actionListener="#{popupManagerCalendarBean.setPopup('newEvent',
               text.NewEvent)}"
6          render="dynamic-popup-content dynamic-popup-title"
7          oncomplete="#{rich:component('dynamic-popup')}.show();">
8      </a4j:commandButton>
9  </h:form>
10 ...
11 <!-- Vyskakovací okno s měnitelným obsahem -->
12 <rich:popupPanel id="dynamic-popup" modal="true" autosized="true">
13     <f:facet name="header">
14         <a4j:outputPanel id="dynamic-popup-title">
15             <h:outputText value="#{popupManagerCalendarBean.title}" />
16         </a4j:outputPanel>
17     </f:facet>
18     <f:facet name="controls">
19         <h:outputLink value="#" onclick="#{rich:component('dynamic-popup
               ')}.hide(); return false;">
20             #{text.Close}
21         </h:outputLink>
22     </f:facet>
23     <h:form>
24         <a4j:outputPanel id="dynamic-popup-content">
25             <ui:include src="#{popupManagerCalendarBean.view}.xhtml" />
26         </a4j:outputPanel>
27     </h:form>
28 </rich:popupPanel>
```

Ukázka 5.7: Návrh komponenty pro ajaxové ovládání vyskakovacího okna.

otevřít klientským skriptem. Atribut `onComplete` se provede po dokončení AJAXového požadavku. Konstrukce `#rich:component('dynamic-popup')` (řádek 7) vytvoří klientský JavaScriptový kód identifikující klientskou komponentu vyskakovacího okna. Její metoda `show()` okno otevře.

Nyní budou popsány ukázky úvodních stránek portletů a jejich význam. Na obrázcích 5.1, 5.2, 5.3 a 5.4 můžeme vidět výsledné podoby portletů v módu `VIEW` s normálním stavem okna. Všechny portlety obsahují podobnou hlavičku. Hlavička obsahuje vždy v pravé části dvě tlačítka, první k obnovení stránky a druhé k otevření příručního menu portletu. V případě portletu Kalendář a Úkoly obsahuje ještě hlavička tlačítko k vytvoření nové entity. Umístění těchto tlačítek bylo zvoleno záměrně, protože budou zřejmě frekventovaně používané. Ve spodní části všech portletů je stejný barevný přechod, což zajišťuje zmíněný požadavek na uniformní vzhled. Nyní se můžeme podívat na jednotlivé portlety zvlášť.

Tělo portletu Kalendář (5.1) je rozděleno na tři části. V pravém sloupci můžeme vidět uživatelovi kalendáře. V hlavní části vidíme měsíční kalendář s ovládacími tlačítky pro po-



Obrázek 5.1: Portlet Kalendář v módu `VIEW` s *normálním* stavem okna.

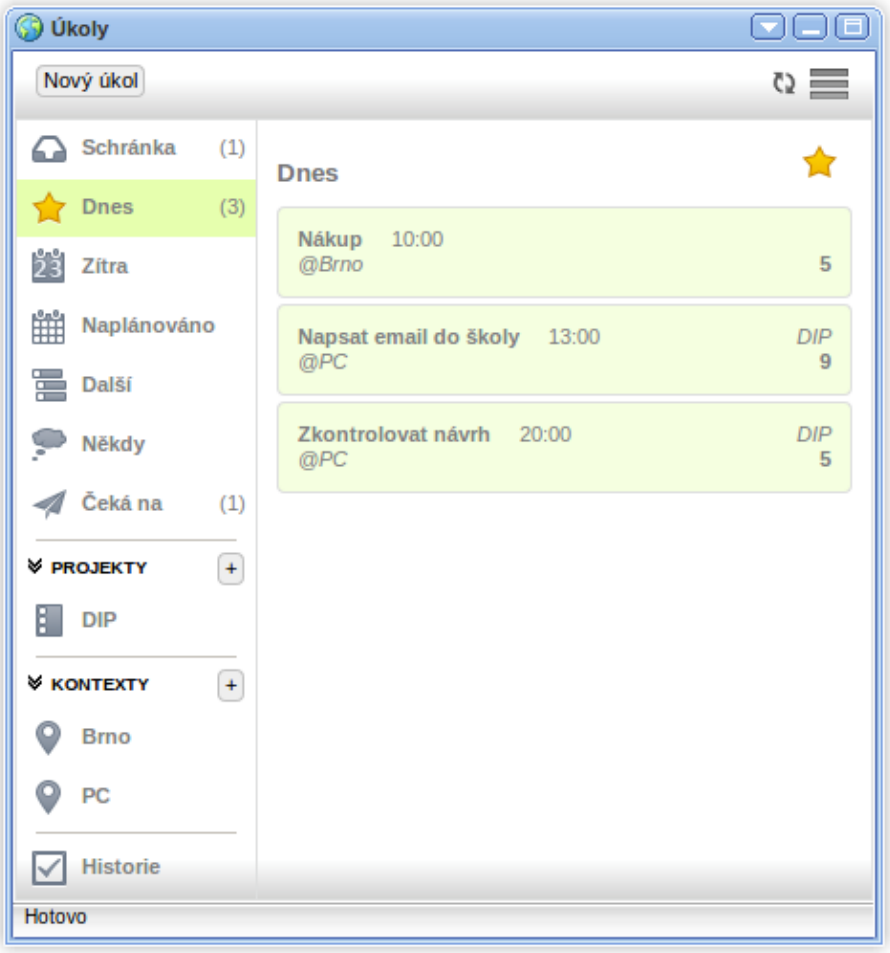
hyb mezi měsíci a roky a u každého dne, kde se nachází nějaké události nebo úkoly můžeme vidět příslušný počet značek ve tvaru malých tmavých čtverečku. Uživatel tedy hned pozná, který den má volný a který den už něco má. Kliknutím na položku dne v kalendáři se zobrazí sjednocené události a úkoly pro daný den ze všech zapnutých kalendářů. Položky jsou seřazeny dle času. Kliknutím na položku se zobrazí detail kalendářové komponenty ve vyskakovacím okně.

Portlet Úkoly (Obrázek 5.2) má tělo stránky rozdělené na dvě části. V levém panelu uživatel vidí jednotlivé složky s úkoly pojmenované dle techniky správy času GTD. U důležitých složek jako je „Schránka“, „Dnes“ a „Čeká na. . . “ je v závorce vidět počet úkolů v dané složce. Pokud měl uživatel nesplněné úkoly z minulého dne, zobrazí se ve složce dnes s červeným nadpisem „Nestihnuto“. Uživatel tedy na první pohled vidí, co ho čeká. Pokud se chce uživatel podívat, jaké úkoly může splnit v konkrétním projektu nebo jaké úkoly může splnit v daném městě, klikne na příslušný projekt nebo kontext. Jednotlivé

položky levého menu tedy vlastně tvoří filtr našich úkolů. Kliknutím na konkrétní úkol se pak zobrazí jeho detail ve vyskakovacím okně. Úkoly lze dále vystavit na RESTové URL a číst jako vzdálený kalendář v portletu Kalendář.

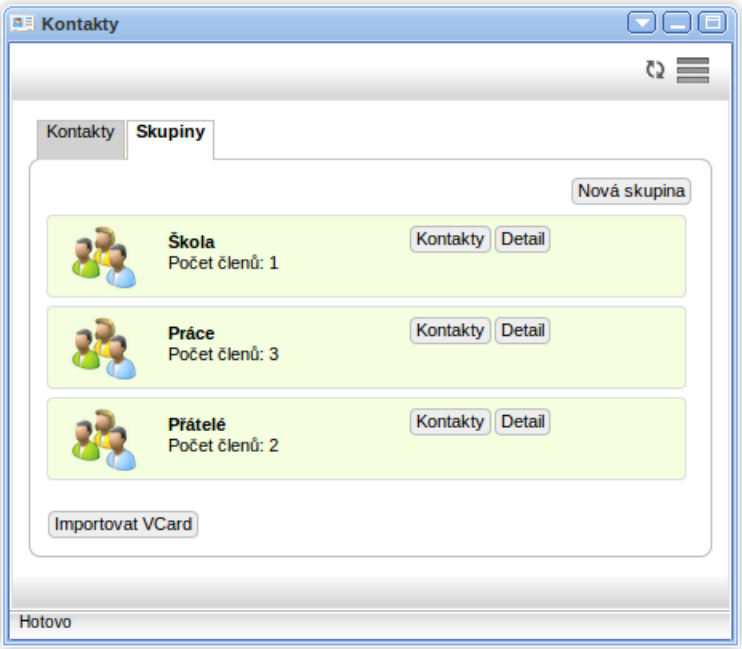
Portlet Kontakty má tělo rozdělené na záložky „Kontakty“ (Obrázek 5.3) a „Skupiny“ (Obrázek 5.4). Zobrazení formou záložek jsem zvolil, protože portlet nebude obsahovat tolik položek, aby mohl v rámci šetření místa vytvořit menu. Záložky navíc vytváří celkem elegantní a přehledný vzhled. Portlet nabízí jednoduché vyhledávání a filtrování kontaktů a obsahuje samozřejmě stránkování kontaktů, které ovšem na obrázku viditelné není. Stránkovat se začne od 10 kontaktů, což je ideální číslo z hlediska spotřeby místa na portálové stránce. Záložka skupiny obsahuje správu skupin kontaktů.

Podrobný popis grafického uživatelského rozhraní a rozebrané případy užití včetně velkého množství snímků obrazovky naleznete v příloze C.



Obrázek 5.2: Portlet Úkoly v módu `VIEW` s *normálním* stavem okna.

Obrázek 5.3: Portlet Kontakty v módu `VIEW` s *normálním* stavem okna.



Obrázek 5.4: Sekce „Skupiny“ v portletu Kontakty.

Kapitola 6

Výsledky a zhodnocení

Tato kapitola si klade za cíl zhodnotit výsledek práce a porovnat ho z hlediska z vybraných aspektů s některými dalšími nástroji pro správu času, o kterých práce pojednávala v sekci 2.3. V této kapitole také zhodnotím poznatky z uživatelské studie a testování (sekce 6.2) a navrhnu možné rozšíření vytvořených portletů.

6.1 Porovnání vytvořených portletů s dalšími nástroji pro správu času

V sekci 2.3 jsme se podívali na nástroje Microsoft Outlook, IBM Lotus Notes, Mozilla Sunbird a Google Apps, které mají určitý průnik v řešení problému správy času s mými vytvořenými portlety. Pokud čtenář odkazovanou sekci nečetl, doporučuji ji nyní přečíst. V tabulce 2.1 jsme si tyto nástroje porovnali z hlediska několika kritérií a nyní se zkusíme podívat na to, kam by se v porovnání zařadily moje portlety. Prvním kritériem bylo rozlišení, zda se jedná o tzv. *groupware*, tedy software pro korporátní využití s možností sdílení některých dat. Portletová technologie sama o sobě je určena pro korporátní využití a moje řešení se dá za groupware určitě považovat z několika důvodů. Zaprvé lze portlety využít, jak na osobní úrovni (uživatelská nástěnka), tak i na skupinové úrovni (portálová stránka vlastněná skupinou). Dále je možné napříč organizací sdílet data přes RESTové rozhraní. Můžeme tedy například zveřejnit firemní kalendář zaměstnancům a zároveň může jeden uživatel (manažer) sdílet pracovní kalendáře svých podřízených. Dalším kritériem bylo rozlišení, zda se jedná o samostatnou aplikaci, nebo o webovou službu. Portlety se samozřejmě budou řadit mezi webové služby. Jediné co potřebujeme pro připojení na portál je mít nainstalovaný prohlížeč a vlastnit přístupové údaje. Co se týče ceny a verze programu, tak vytvořené portlety jsou zdarma (*freeware*) pod licencí LGPL[1]. Verze programu je aktuálně 1.0 k datu odevzdání této práce. Nyní se dostáváme k jádru porovnání. Všechny zmíněné nástroje jsme porovnávali z hlediska podpory standardních formátů pro export a import dat v naší doméně. Implementované portlety využívají pro export a import formát iCalendar. Portlet Kalendář navíc podporuje synchronizaci pomocí protokolu CalDAV. Pro práci s formátem iCalendar využívají portlety knihovnu iCal4j a ta obsahuje plnou podporu formátu iCalendar dle normy RFC-2445, což je předchůdce normy RFC-5545. Samotné portlety nevyužívají všech možností knihovny, ale tak rozsáhlou podporu nemá ani žádný jiný z porovnávaných nástrojů. Portlet Kontakty navíc podporuje standard vCard implementovaný knihovnou ez-vCard, která plně podporuje formát vCard dle norem RFC 2426

[1] http://www.gnu.org/licenses/lgpl.html

	Microsoft Outlook	**IBM Lotus Notes**	**Mozilla Sunbird**	**Google Apps**	**Sada portletů**
Typ softwaru	standalone	standalone	standalone	webová služba	**web. služba**
Verze	2007	8.5.3	1.0beta1	k 18.12.2012	**k 21.5.2013**
Zdarma	ne	ne	ano	ano	**ano**
Groupware	ano	ano	ano	ano	**ano**
iCalendar	ano	ano	ano	ano	**ano**
CalDAV	ne	ano	ano	ano	**ano**
vCard	ano	ano	ne	ano	**ano**
CardDAV	ne	ano	ne	ano	**ne**

Tabulka 6.1: Porovnání softwarových nástrojů pro správu času s vytvořenou sadou portletů.

a RFC 6350. Portlet Kontakty opět nevyužívá všech možností, které knihovna nabízí, ale stejně jako v předchozím příkladě nemají plnou podporu daných norem ani ostatní porovnávané aplikace. Důležitý fakt ale je, že popsané nástroje pro správu času mají více či méně podobnou podporu podmnožiny vlastností použitých formátů, proto i import a export dat nemá kritické ztráty. Tento závěr se týká i mých portletů. V tabulce 6.1 je přehledné shrnutí právě popsaného porovnaní.

6.2 Uživatelská studie

Výsledná sada portletů byla podrobena testování několika uživatelů. Následně jsem sestavil dotazník a požádal tyto uživatele o vyplnění. Uživatelé měli několik společných vlastností. Byli to muži, ve věku 20 – 40 let s vysokoškolským vzděláním a dobrou orientací v oboru informačních technologií. Uživatelé dostali možnost vyzkoušet portlety buď na propůjčeném školním serveru `pcletko2.fit.vutbr.cz`, kam jsem nasadil instanci portálu GateIn, nebo na vlastní instanci. Po seznámení této skupiny s produktem jsem se jich prostřednictvím dotazníku zeptal, jak se jim líbily jednotlivé portlety samy o sobě i sada portletů jako celek. Zajímal jsem se zejména o zhodnocení jednoduchosti, intuitivnosti, vzhledu a hlavně použitelnosti jak celé aplikace, tak jednotlivých jejich částí. Až na dvě otevřené otázky, kde byl udělen respondentům prostor na slovní vyjádření, byly všechny ostatní otázky uzavřené. Uzavřené otázky mají výhodu snadnějšího vyhodnocení, ale bez pár otevřených otázek se dotazník neobejde. Občas potřebujeme od uživatele odpověď, která nelze vyjádřit v nabízených možnostech.

Tento dotazník měl za cíl zjistit, zda má smysl v tomto projektu dál pokračovat a jakým směrem se případně ubírat. Jako sekundární cíl bylo samotné otestování funkčnosti sady portletů a odhalení případných dalších chyb. Použitý dotazník si čtenář může prohlédnout v příloze B.

Dotazník vyplnilo pět respondentů, kteří se před tím seznámili s vytvořenými portlety. Všichni dotázaní hodnotili vzhled portletů jako obyčejný, ale pěkný, kromě jednoho respondenta, který vzhled portletu Úkoly hodnotil jako inovativní a líbil se mu. S tímto výsledkem jsem spokojený a nemyslím si, že je potřeba do vzhledu portletů výrazněji zasahovat. Mnohem důležitější z mého pohledu byly otázky na intuitivnost a jednoduchost ovládání. V tomto hodnocení dopadly portlety také dobře. Většina respondentů je hodnotila jako in-

tuitivní a dobře ovladatelné. Portlet Kalendář obdržel jen jedno chvalitebné hodnocení a portlet Úkoly dvě chvalitebná hodnocení. Chvalitebným hodnocením myslím malou připomínku k jinak kladnému hodnocení. Funkcionalitu exportu a importu dat považuje 60% respondentů za běžnou a očekávanou. Dalších 40% dotázaných funkcionalitu považuje za zdařilou a překročila jejich očekávání. Co se týče podpory protokolu CalDAV, tak tu bohužel 40% dotázaných nezkoušelo, ale zbytek ji hodnotil velice pozitivně. Podporu techniky GTD u správy úkolů všichni dotázání vyzdvihli a považují ji za efektivní způsob řízení času, což hodnotím jako velký úspěch práce. Další otázka zkoumala dostatečnost požadovaných atributů pro entitu kontakt. Jako vyhovující hodnotilo daný stav 60% dotázaných. Ostatní postrádali možnost přidání fotografie nebo kontakt na profil programu Skype[2]. Poslední uzavřená otázka hodnotila možnost spolupráce sady portletů. Všichni, kdo tuto možnost vyzkoušeli, ji hodnotili jako velice zdařilou. Bohužel ji vyzkoušelo pouze 60% dotázaných.

Nyní se pokusím krátce shrnout zhodnocení otevřených otázek. Někteří uživatelé se znovu vyjadřovali pozitivně k intuitivnosti nebo vzhledu výsledných aplikací a potvrzovali tím pozitivní hodnocení. Kladně byl hodnocen i sjednocený vzhled sady portletů. Co se týče funkcionality, tak uživatelé nejčastěji chválili podporu vzdálených kalendářů a možnost exportu a importu dat. Dva dotázaní uvedli, že mají problém s exportem a importem dat u aplikací Google kalendář nebo Microsoft Outlook, ale se sadou portletů žádný problém nenastal. Často zmiňovaným pozitivem byla agregace časových údajů v portletu Kalendář, které si cením, protože to považuji za významný kladný bod, který není v konkurenčních aplikacích dostatečně uspokojen. Jako negativní prvek hodnotil jeden respondent přehnané použití vyskakovacích oken a dalšímu dotázanému se zase nelíbil způsob zavíraní těchto oken a dále nejednoznačnost příslušnosti okna k portletu. Uživatel si zde nebyl jistý, ke kterému portletu vlastně přidává nová data, když bylo na jedné stránce více instancí některého portletu. Jeden uživatel pak nalezl chybu v synchronizaci se vzdáleným kalendářem, která byla následně opravena. Dále jsem byl upozorněn na problém inicializace databáze po restartu aplikačního serveru. Tento nedostatek byl způsoben pozůstatkem jednoho testovacího nastavení v konfiguračním souboru technologie JPA. Problém jsem následně vyřešil.

Uživatelská studie byla velice prospěšná, protože mi pomohla najít několik chyb a identifikovala slabé a silné stránky sady portletů pro správu času. Následující sekce rozebere možné pokračování práce s přihlédnutím na výsledky této studie.

6.3 Návrh pokračování práce

Na základě výsledků z předchozí sekce a vlastních postřehů nyní uvedu několik možných doporučení na pokračování této práce. Z provedené studie konkurenčních aplikací jsem zjistil, že žádná (ani tato) nenabízí plnou podporu formátů iCalendar a vCard, a dokonce si vymýšlejí svoje vlastní atributy, které pak znemožňují automatizované čtení souborů v daném formátu bez speciální podpory pro danou aplikaci. Práce by mohla pokračovat zúplněním této množiny nabízených atributů. Dále si dokážu představit implementaci dynamického zobrazení dat v porletu nejenom na základě portletových módů, ale i na základě stavů oken a rozlišení obrazovky. Dnešní doba prochází obrovským rozvojem mobilních zařízení. Z důvodu plného využití i v těchto zařízeních by bylo vhodné, aby rozměr monitorů nebyl problém a obsah neměnil pouze svou velikost, ale i uspořádání svých částí, obsah a vzhled. Portál GateIn nyní nabízí mód *Mobile*, který tvoří ideální podmínky pro nasazení těchto vylepšených portletů. Více informací obsahuje následující zdroj [55].

[2] http://www.skype.com/cs/

Kapitola 7

Závěr

Předmětem této diplomové práce vyvíjené ve spolupráci s firmou Red Hat bylo vytvoření sady nástrojů pro správu času v prostředí portálu. Hlavním smyslem práce byla demonstrace, že kombinace relativně nových technologií portálů, portletů a portletových mostů je použitelná a vhodná alternativa pro tvorbu webových aplikací a informačních systémů. Tuto práci využije firma Red Hat jako výukový materiál pro tvorbu portletů v těchto technologiích. Implementované portlety pro správu času pak budou nasazeny jako demonstrační ukázky portletů na portále GateIn.

Táto práce detailně popsala architekturu portletových aplikací v prostředí portálů a vysvětlila několik možných přístupů vývoje, a to zejména vývoje pomocí portletový mostů. V další části byl vytvořen architektonický návrh pro zvolené požadavky na sadu portletů a v implementační části práce jsem popsal samotný vývoj.

Výsledek mé práce splnil všechny body zadání a navíc se mi podařilo zajistit vzájemnou spolupráci portletů. Portlet Kalendář dokáže agregovat data svoje s časovými daty portletů Úkoly a Kontakty. Těmito daty myslím časově naplánované úkoly a data narozenin a výročí uživatelských kontaktů. Toto provázání zvýší efektivitu správy času, což mi potvrdily výsledky vyhodnoceného dotazníku uživatelů, kteří portlety testovali. Dále se mi podařilo implementovat nastavení opakování událostí a vytvořil jsem možnost exportu a importu dat do standardních formátů iCalendar a vCard. Zajímavou funkcionalitou je i synchronizační mechanismus kalendářů, který lze využít například s Google kalendářem. Za další důležitý přínos práce považuji vyřešení problému ukládání atributů kontrolérů do portletového sezení. Tuto funkcionalitu standardně portletový most zajistit nedovede a technologie JSF ji implicitně nepotřebuje. Díky tomu je možné nasadit více instancí jednoho portletu na jedné portálové stránce jedním uživatelem v případě, že portlet využívá své portletové sezení k ukládání proměnných a atributů kontrolérů. Mezi zajímavé výstupy mé diplomové práce považuji možnost sdílení dat přes RESTové rozhraní mezi portlety a implementaci techniky „Mít vše hotovo“ (angl. Getting Things Done, GTD) pro správu času v portletu Úkoly.

Tato práce by mohla pokračovat zúplněním podpory datovým formátů iCalendar a vCard, což je stav, kterého nedosahuje žádná z v práci porovnávaných aplikací pro správu času. Dále si dokážu představit implementaci dynamického zobrazení dat v porletu nejenom na základě portletových módů, ale i na základě stavů oken a rozlišení obrazovky, což se velice využije i díky dnešnímu dynamickému rozvoji mobilních zařízení.

Literatura

[1] Allen, D.: *Mít vše hotovo*. Brno: Zoner press, 2008, ISBN 9788090391284.

[2] Angstadt, M.: EZ-vCard Java library. 2013-04-17 [cit. 2013-04-20], [Online].
URL https://code.google.com/p/ez-vcard/

[3] Authority, I. A. N.: MIME Media Types. 2013-01-30 [cit. 2013-05-01], [Online].
URL http://www.iana.org/assignments/media-types

[4] Berners-Lee, T.: Uniform Resource Locators (URL). prosinec 1994 [cit. 2012-12-01], [online].
URL http://tools.ietf.org/html/rfc1738

[5] Brian Leathem, S. R., Lukas Fryc: RichFaces. 2011-04-11 [cit. 2013-04-10], [Online].
URL http:
//docs.jboss.org/richfaces/latest_4_2_X/Developer_Guide/en-US/html/

[6] C. Daboo, L. D., B. Desruisseaux: Calendaring Extensions to WebDAV (CalDAV). březen 2007 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc4791

[7] Covey, S. R.: *Sedm návyků skutečně efektivních lidí*. Praha: Management Press, 2011, ISBN 9788072612413.

[8] Daboo, C.: iCalendar Transport-Independent Interoperability Protocol (iTIP). prosinec 2009 [cit. 2013-01-10], [online].
URL http://tools.ietf.org/html/rfc5546

[9] Daboo, C.: CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV). srpen 2011 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc6352

[10] DeMichiel, L.: JSR 317: Java™ Persistence 2.0. 2004-05-27 [cit. 2013-03-11], [online].
URL http://www.jcp.org/en/jsr/detail?id=317

[11] Desruisseaux, E. B.: Internet Calendaring and Scheduling Core Object Specification (iCalendar). září 2009 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc5545

[12] Edward Burns, C. R. M.: JSR 127: JavaServer Faces. 2004-05-27 [cit. 2012-12-01], [online].
URL http://www.jcp.org/en/jsr/detail?id=127

[13] Edward Burns, R. K.: JSR 314: JavaServer Faces 2.0 (Maintenance Release 2). 2010-11-22 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=314

[14] F. Dawson, D. S.: Internet Calendaring and Scheduling Core Object Specification (iCalendar). listopad 1998 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc2445

[15] F. Dawson, T. H.: vCard MIME Directory Profile. září 1998 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc2426

[16] Fortuna, B.: iCal4j. 2012-09-09 [cit. 2013-01-03], [Online].
URL http://ical4j.sourceforge.net

[17] Foundation, T. A. S.: Apache Maven. 2013 [cit. 2013-04-20], [Online].
URL http://maven.apache.org/maven-features.html

[18] Freedman, M.: JSR 301: Portlet 1.0 Bridge for JavaServer™ Faces 1.2 (Final Release). 2010-07-08 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=301

[19] Freedman, M.: JSR 329: Portlet 2.0 Bridge for JavaServer™ Faces 1.2 Specification (Final Release). 2011-01-13 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=329

[20] Google: Google Calendar CalDAV Developer's Guide. 2012-07-12 [cit. 2012-12-09], [Online].
URL https://developers.google.com/google-apps/calendar/caldav

[21] Group, H.: H2 Database. 2013 [cit. 2013-04-18], [Online].
URL http://www.h2database.com/

[22] Group, O. M.: Unified Modeling Language: Infrastructure. březen 2006 [cit. 2013-01-12], [Online].
URL http://www.omg.org/spec/UML/2.0/Infrastructure/PDF/

[23] Hamberg, E.: GTD in 15 minutes - A Pragmatic Guide to Getting Things Done. 20012-07-27 [cit. 2013-04-05], [online].
URL http://hamberg.no/gtd/

[24] Jappe van der Hel, R. Z., Philipp Eichhorn: Project Lombok. 2013-01-01 [cit. 2013-03-11], [Online].
URL http://projectlombok.org/download.html

[25] JBoss: GateIn Portal documentation. 2013 [cit. 2012-11-05], [Online].
URL https://docs.jboss.org/author/display/GTNPORTAL36/Home

[26] Julian Aubourg, J. S.: XMLHttpRequest. [online], 2012-12-06 [cit. 2013-04-13].
URL http://www.w3.org/TR/XMLHttpRequest/

[27] Ken Finnigan, P. L., Luca Stancapiano: *Gatein Cookbook*. 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing, 2012, ISBN 9781849518628.

[28] King, G.: JSR 299: Contexts and Dependency Injection for the Java™ EE platform. 2009-12-10 [cit. 2013-03-11], [online].
URL http://jcp.org/en/jsr/detail?id=299

[29] Linda DeMichiel, M. K.: Enterprise JavaBeans™ 3.0. 2007-12-17 [cit. 2013-04-10], [online].
URL http://jcp.org/en/jsr/detail?id=220

[30] Mark Birbeck, M. G., Sidewinder Labs: XHTML™ 2.0. [online], 2010-12-16 [cit. 2012-12-01].
URL http://www.w3.org/TR/xhtml2/

[31] Mark Hapner, B. S.: JSR 151: Java™ 2 Platform, Enterprise Edition 1.4 (J2EE 1.4) Specification (Maintenance Draft Review). 2004-04-26 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=151

[32] Nicklous, M. S.: JSR 168: Portlet Specification (Final Release). 2003-10-27 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=168

[33] Nicklous, M. S.: JSR 286: Portlet Specification 2.0 (Final Release). 2008-01-12 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=286

[34] Oracle: Building RESTful Web Services with JAX-RS. leden 2013 [cit. 2013-03-05], [online].
URL http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html

[35] Oracle: Contexts and Dependency Injection for the Java EE Platform. leden 2013 [cit. 2013-03-11], [online].
URL http://docs.oracle.com/javaee/6/tutorial/doc/gjbnr.html

[36] Oracle: Persistence. leden 2013 [cit. 2013-03-20], [online].
URL http://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html

[37] Oracle: Enterprise Beans. leden 2013 [cit. 2013-04-10], [online].
URL http://docs.oracle.com/javaee/6/tutorial/doc/gijsz.html

[38] Paradigm, V.: Visual Paradigm for UML. 2013 [cit. 2013-01-12], [Online].
URL http://www.visual-paradigm.com/

[39] Pelegri-Llopart, E.: JSR 53: Java™ Servlet 2.3 and JavaServer Pages™ 1.2 Specifications (Maintenance Draft Review 2). 2002-05-24 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=53

[40] Perreault, S.: vCard Format Specification. srpen 2011 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc6350

[41] P.Leach, R., M. Mealling: A Universally Unique IDentifier (UUID) URN Namespace. červenec 2011 [cit. 2013-04-09], [online].
URL http://tools.ietf.org/html/rfc4122

[42] Potociar, M.: JSR 311: JAX-RS: The Java™ API for RESTful Web Services. 2009-11-23 [cit. 2013-03-05], [online].
URL http://jcp.org/en/jsr/detail?id=311

[43] RichFaces: RichFaces: Tag fileUpload. 2013 [cit. 2013-04-10], [Online].
URL
http://docs.jboss.org/richfaces/latest_4_X/vdldoc/rich/fileUpload.html

[44] Roberto Chinnici, B. S.: JSR 316: Java™ Platform, Enterprise Edition 6 (Java EE 6) Specification. 2009-12-10 [cit. 2013-05-07], [online].
URL http://jcp.org/en/jsr/detail?id=316

[45] Rod Johnson, A. A., Juergen Hoeller: Spring MVC Portlet Framework. 2013-02-10 [cit. 2013-04-11], [Online].
URL
http://static.springsource.org/spring/docs/2.0.8/reference/portlet.html

[46] Roland H. Alden, S. J. B.: vCalendar — The Electronic Calendaring and Scheduling Exchange Format Version 1.0. 1996-09-18 [cit. 2012-12-09], [Online].
URL http://www.imc.org/pdi/vcal-10.txt

[47] Sarin, A.: *Portlets in Ation*. Shelter Island, NY: Manning Publications, 2012, ISBN 9781935182542.

[48] Shannon, B.: JSR 58: Java™ 2 Platform, Enterprise Edition 1.3 Specification (Final Release). 2001-09-24 [cit. 2012-12-01], [online].
URL http://jcp.org/en/jsr/detail?id=58

[49] Stefan Hepper, O. K.: What's new in the Java Portlet Specification V2.0 (JSR 286)? 2008-03-18 [cit. 2012-12-09], [Online].
URL http://www.ibm.com/developerworks/websphere/library/techarticles/0803_hepper/0803_hepper.html

[50] Wheeler, D. A.: SLOCCount User's Guide. 2004-04-01 [cit. 2013-01-12], [Online].
URL http://www.dwheeler.com/sloccount/sloccount.html

[51] Wikipedia: ICalendar — Wikipedia, The Free Encyclopedia. 2012-11-04 [cit. 2012-12-09], [Online].
URL
http://en.wikipedia.org/w/index.php?title=ICalendar&oldid=521328115

[52] Wikipedia: Microsoft Outlook — Wikipedia, The Free Encyclopedia. 2012-12-02 [cit. 2012-12-09], [Online].
URL http:
//en.wikipedia.org/w/index.php?title=Microsoft_Outlook&oldid=525949133

[53] Wikipedia: Web Service — Wikipedia, The Free Encyclopedia. 2012-12-04 [cit. 2013-01-03], [Online].
URL
http://en.wikipedia.org/w/index.php?title=Web_service&oldid=526373314

[54] Wikipedia: Collaborative software — Wikipedia, The Free Encyclopedia. 2012-12-21 [cit. 2013-01-03], [Online].
URL http://en.wikipedia.org/w/index.php?title=Collaborative_software&oldid=529088436

[55] Wringe, M.: GateIn Mobile Portlet Development. 2012-04-11 [cit. 2013-05-10], [Online].
URL https://community.jboss.org/wiki/GateInMobilePortletDevelopment

[56] Yergeau, F.: UTF-8, a transformation format of ISO 10646. leden 1998 [cit. 2012-12-09], [online].
URL http://tools.ietf.org/html/rfc2279

Příloha A

Obsah DVD

Kořenová složka DVD obsahuje následující soubory:

- Soubor README.TXT
  Soubor obsahuje základní informace o této diplomové práci a návody použití obsahu DVD.
- Archív `dip-xbasov00.war`
  Archív obsahuje již přeložené portlety a je možné ho rovnou nahrát na portál dle návodu v souboru README.TXT.
- Zdrojové soubory sady portletů.
- Zdrojové soubory technické práce v jazyce LaTeX.
- Soubor settings.xml
  Referenční konfigurační soubor s nastavenými zdrojovými repozitáři pro nástroje Maven.

Příloha B

Dotazník

Následující dotazník se skládá ze 4 částí a obsahuje celkem 13 otázek. Před vyplněním dotazníku by se měl respondent detailně seznámit se sadou portletů.

Portlet Kalendář

1. Jak se Vám líbil vzhled portletu Kalendář?
   (a) Vzhled je inovativní a velice se mi líbil.
   (b) Vzhled je obyčejný, ale pěkný.
   (c) Vzhled se mi nelíbí.
   (d) Na vzhled nemám názor.
2. Přišla Vám práce s portletem Kalendář jednoduchá a intuitivní?
   (a) Velice rychle jsem pochopil, jak se s portletem pracuje.
   (b) S portletem se pracuje dobře, ale některé případy užití se ovládají těžkopádně.
   (c) S portletem se mi nepracuje moc dobře, ale nakonec jsem se ovládání naučil.
   (d) S portletem se mi nepracuje dobře a nechápu úplně ovládání.
   (e) Portlet jsem netestoval.
3. Jak se Vám líbí možnost importu a exportu dat do formátu iCalendar?
   (a) Funkce se mi líbí a pracuje lépe než jsem čekal.
   (b) Funkce pracuje jak jsem očekával.
   (c) Funkce je vyhovující, ale s výhradami.
   (d) Funkce nepracuje dobře.
   (e) Funkci se mi nepodařilo zprovoznit.
   (f) Funkci jsem nezkoušel.
4. Jak se Vám líbí možnost synchronizace dat pomocí CalDAV protokolu?
   (a) Funkce se mi líbí a pracuje lépe než jsem čekal.

(b) Funkce pracuje jak jsem očekával.

(c) Funkce je vyhovující, ale s výhradami.

(d) Funkce nepracuje dobře.

(e) Funkci se mi nepodařilo zprovoznit.

(f) Funkci jsem nezkoušel.

Portlet Úkoly

5. Jak se Vám líbil vzhled portletu Úkoly?

   (a) Vzhled je inovativní a velice se mi líbil.

   (b) Vzhled je obyčejný, ale pěkný.

   (c) Vzhled se mi nelíbí.

   (d) Na vzhled nemám názor.

6. Přišla Vám práce s portletem Úkoly jednoduchá a intuitivní?

   (a) Velice rychle jsem pochopil, jak se s portletem pracuje.

   (b) S portletem se pracuje dobře, ale některé případy užití se ovládají těžkopádně.

   (c) S portletem se mi nepracuje moc dobře, ale nakonec jsem se ovládání naučil.

   (d) S portletem se mi nepracuje dobře a nechápu úplně ovládání.

   (e) Portlet jsem netestoval.

7. Jak se Vám líbíla možnost práce s úkoly pomocí metody „Mít vše hotovo“ (angl. Getting things done, GTD)?

   (a) Věřím, že díky této metodě bude správa úkolů efektivnější.

   (b) Tato metoda je zajímavá, ale mně osobně se nelíbí.

   (c) Metoda se mi nelíbí a myslím, že v praxi moc využitelná není.

   (d) Na danou věc nemám názor.

Portlet Kontakty

8. Jak se Vám líbil vzhled portletu Kontakty?

   (a) Vzhled je inovativní a velice se mi líbil.

   (b) Vzhled je obyčejný, ale pěkný.

   (c) Vzhled se mi nelíbí.

   (d) Na vzhled nemám názor.

9. Přišla Vám práce s portletem Úkoly jednoduchá a intuitivní?

   (a) Velice rychle jsem pochopil, jak se s portletem pracuje.

   (b) S portletem se pracuje dobře, ale některé případy užití se ovládají těžkopádně.

   (c) S portletem se mi nepracuje moc dobře, ale nakonec jsem se ovládání naučil.

   (d) S portletem se mi nepracuje dobře a nechápu úplně ovládání.

(e) Portlet jsem netestoval.

10. Podařilo se Vám ke kontaktům uložit všechny informace, které potřebujete?

    (a) Ano a vyhovuje mi to.
    (b) Většina ano, ale některé atributy jsem postrádal.
    (c) Většinu pro mne důležitých atributů jsem ve formuláři nenašel.
    (d) S portletem se mi nepracuje dobře a nechápu úplně ovládání.
    (e) Danou věc jsem netestoval.

    Pokud byla odpověď b) nebo c) připiště prosím seznam některých atributů, které jste postrádali:

Společné prvky portletů

11. Jak se Vám líbí možnost zobrazení časově naplánovaných úkolů z portletu Úkoly a výročí a narozenín z portletu Kontakty v portletu Kalendář?

    (a) Funkce funguje pěkně a velice se mi líbí možnost agregace dat ostatních portletů v porletu Kalendář!
    (b) Funkce je zajímavá, ale nefunguje úplně jak bych čekal.
    (c) Funkce se mi částečně nebo úplně nepodařila zprovoznit.
    (d) Vůbec nevím, že je tato funkcionalita možná.
    (e) O funkci vím, ale nezkoušel jsem ji.

12. Co se Vám na sadě portletů líbilo? (otevřená otázka, max. 3 body)

13. Co se Vám na sadě portletů nelíbilo? (otevřená otázka, max. 3 body)

Děkuji moc za vyplnění dotazníku!

Příloha C

Manuál

Tato kapitola obsahuje uživatelský manuál pro sadu portletů v angličtině.

C.1 User manual

This user guide describes a set of time management portlets for portal environment. This set includes Calendar, Tasks and Contacts.

Portlet Calendar manages user's calendars which consist of events. User can create several types of calendars (including remote ones) and add events to them. Events of all calendars are aggregated and user has his summary of all events from selected date. In addition, this calendar can display scheduled tasks from portlet Tasks and birthdays and anniversaries of his contacts from portlet Contacts.

Portlet Tasks can manage your tasks effectively according to the technique Getting Things Done (GTD). GTD simply creates a special life-cycle of each task and user knows every time the next task he should work on.

And finally, **portlet Contacts** allows user to manage his contacts and group them in custom contact groups and search them quickly.

Lets have a look at each portlet separately!

C.1.1 Portlet Calendar

Portlet Calendar works in three modes which may be switched using window controllers. Controllers are placed in the upper right corner of the portlet window. The first button handles switching between portlet modes. The second button may be used for minimizing the portlet. The third button switches the portlet to the maximized window and back to the normal window state.

VIEW MODE

Now let's focus on the VIEW mode which can be seen in Figure C.1. The window is divided into three parts – header, body and right block. In the header part, you can see button "New Event" which you can use to create a new event. Then you can see two icons on the right side of the header. The first one is "Refresh" button. If any problem occurred, you can try to refresh page by this button. The second button is the "Menu" button. By clicking this button, a new menu where you can switch portlet modes is opened (navigation panel may be missing – the setting is in portal administration mode).

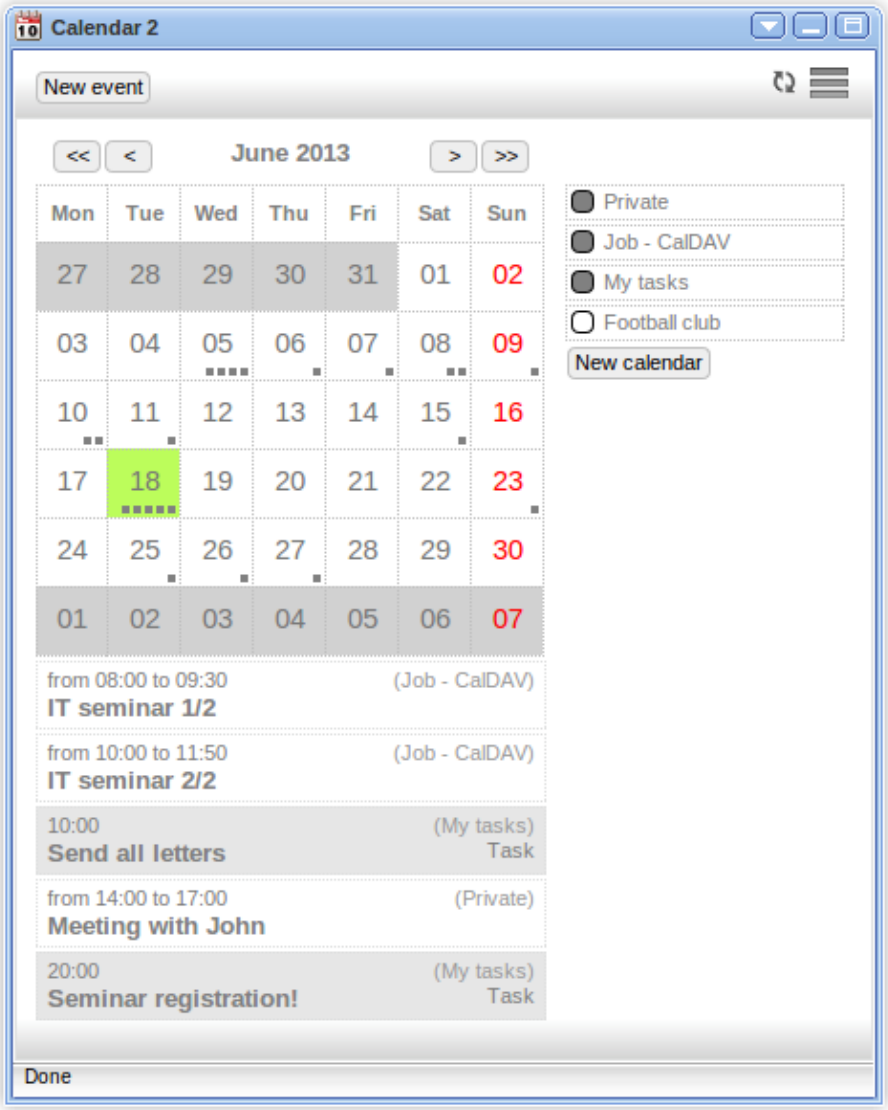


Figure C.1: Calendar portlet in the `VIEW` mode in the normal state window.

The right block of the portlet contains the user's calendars. You can have several calendars of different types. Before you can add new event, you have to own at least one editable calendar (types of calendars are described bellow). In Figure C.1 you can see that the calendars can be switched-off by clicking on their title. The calendar "Football club" in our example is disabled. Events from disabled calendars are not visible in the body part. If you put a mouse pointer on a calendar item, special icon appears on the right side of the item. By clicking on it, calendar menu is opened. In this menu, there are options to open calendar detail and calendar edit pop-up window or you can just delete calendar including all its events or synchronize calendar if it is a remote one.

In the body part of the portlet, you can see a calendar where all events from enabled calendars are merged. By clicking on a date, you can see all events for this date ordered by time. Some dates in calendar might contain small gray squares. These represent the number of events of this date. Events in the list bellow contain title of their calendar in the

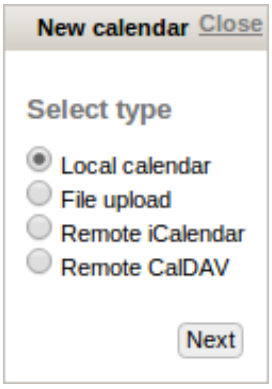


Figure C.2: Selection of type of new calendar.

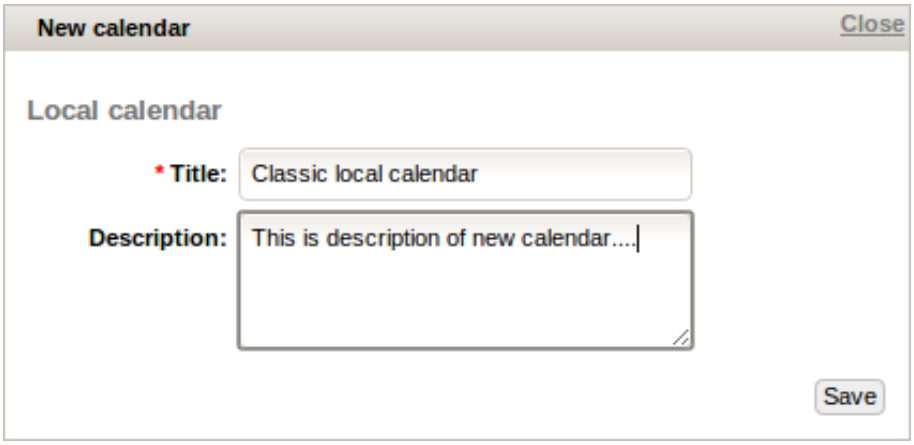


Figure C.3: New local calendar form.

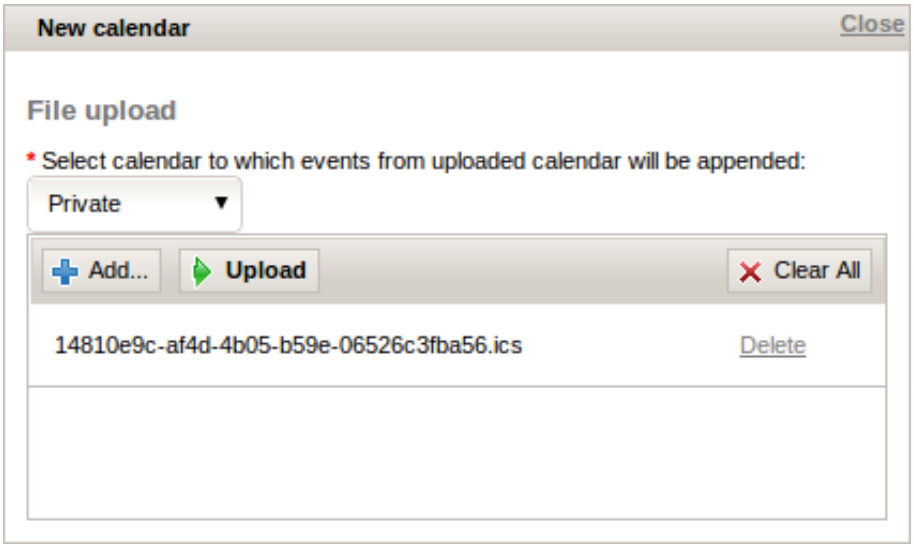


Figure C.4: Calendar upload form.

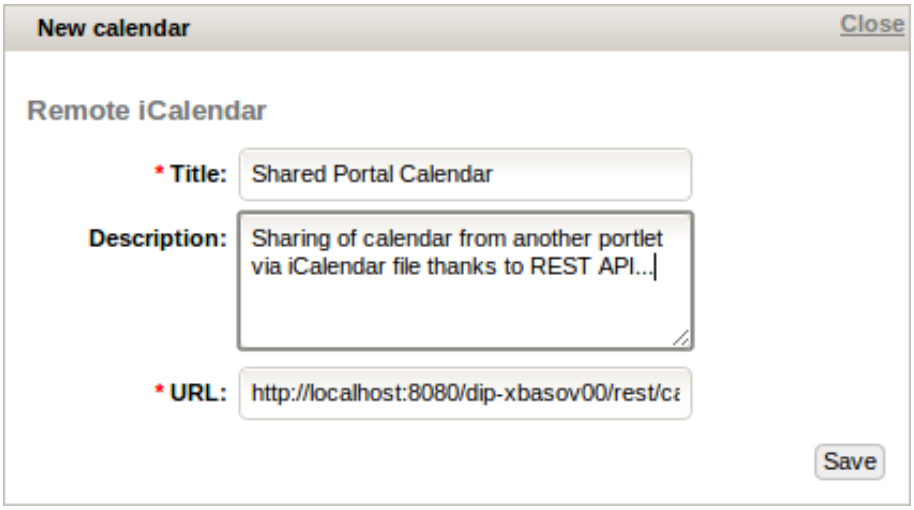


Figure C.5: New remote iCalendar form.

upper right corner. You can also see special events with dark background and small title "Task" in bottom-right corner. This represents the "ToDo" components. ToDo components cannot be created in calendar (Calendar portlet is not targeted for creation of tasks) but you can read tasks from remote calendars or from the Tasks portlet which will be described in the next section. Finally, you can see small buttons above calendar used for switching months and years in both directions (back & forward).

In portlet Calendar, you can create several types of calendars. When you click the "New calendar" button, a pop-up window will open (Figure C.2) and you can choose the calendar type from several options. Option "Local calendar" will create new local calendar on your portal. Local calendar form screen is displayed in Figure C.3. Second option is "File upload". You can import calendar components (VEVENTs and "scheduled" VTODOs) from iCalendar file to an existing editable calendar. Screen of upload calendar form is shown in Figure C.4. The third option is to create "Remote iCalendar". This form is in Figure C.5. You can see field "URL" where you put URL to an iCalendar file. Thanks to this feature you can share calendars on your portal because each calendar is available via REST API. URL of your local calendars can be found in calendar details. Remote calendars are read-only because we cannot modify remote files. The last option is to create new "Remote CalDAV" calendar. These calendars use CalDAV protocol and we can modify them. CalDAV form is in Figure C.6. In this form you have to fill your username and password[1] to remote calendar. With this function you can connect to Google Calendar. Users can manually synchronize all remote calendars.

In Figure C.7, you can see pop-up window with a calendar detail. Detail contains "edit" and "delete" buttons. Moreover, you can see attribute "Calendar URL". This URL can be used to share this calendar via different portlet instance. Refresh icon is used to generate new calendar URL. When you generate new URL, all remote calendars using old URL will be broken. Use this option to stop sharing your calendar by remote readers.

Now let's have a look on creation of a new event. In Figure C.8 we can see a form to create a new event. The meaning of most of the fields is obvious. For each event, you have to select a calendar to which the event will be added. If you want, you can use repeat block

[1]Warning: Passwords are stored in database as plaint-text now!

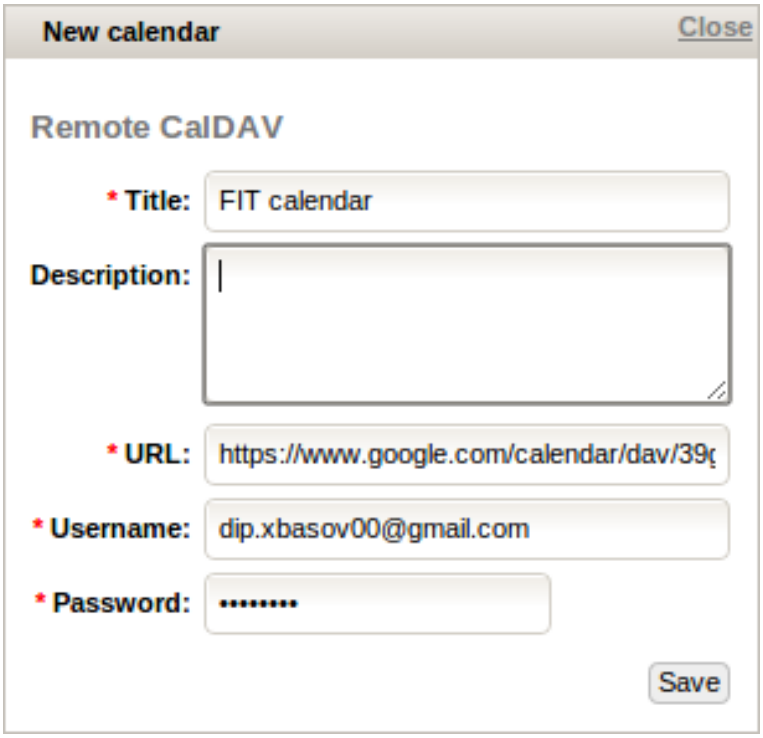


Figure C.6: New remote CalDAV calendar form.

Figure C.7: Local calendar detail.

to setup recurrences of your event. Occurrence types are “daily”, “weekly”, “monthly”, “yearly” and “none” (to switch-off occurrences). Each type has its own attributes. In the figure, there is selected week occurrence and you setup week days when you want to repeat your event. Attribute “Interval” is used for skipping defined occurrences. If the interval is of value 3, every second and third occurrences will be skipped. You can also define a type of end of occurrences. Options are “Never”, “After a number of times” and “Specific date”.

EDIT MODE and HELP MODE

Edit mode contains portlet settings. Portlet settings include attribute “Start of week”. Possible values are “Sunday” and “Monday”. Be careful where is your portlet instance put. In group portal page, every member of group can change this settings. Your private

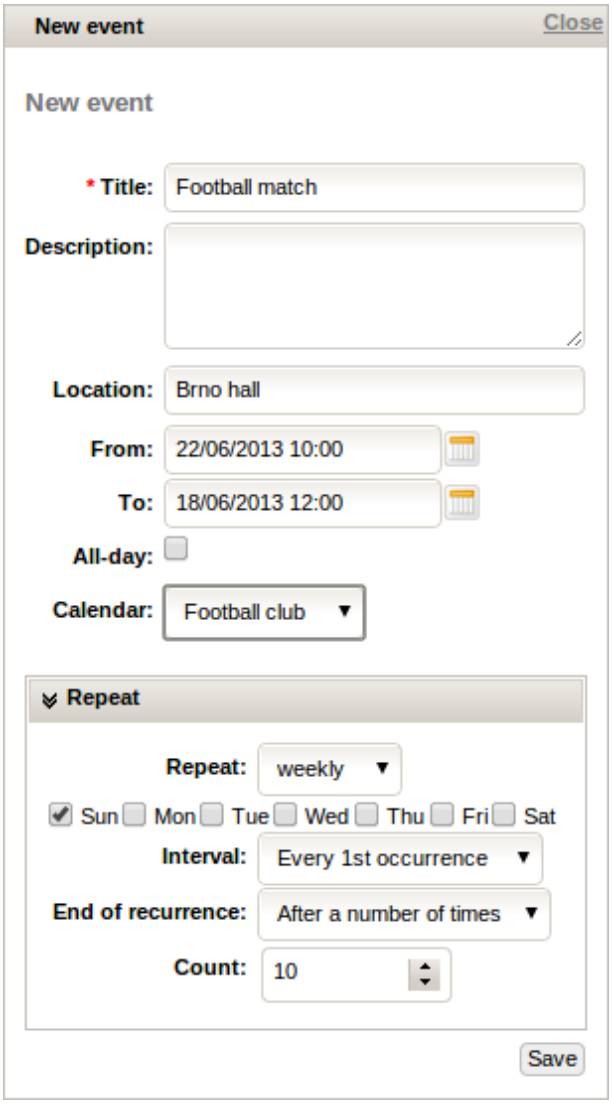


Figure C.8: New event form.

portlet instances should be put on your private dashboard! Only portlet instances on your dashboard are really private!

Help mode contains link to this user manual.

C.1.2 Portlet Tasks

Portlet Tasks works in two modes which may be switched using window controllers. Controllers are placed in the upper right corner of the portlet window. The first button handles switching between portlet modes. The second button may be used for minimizing of the portlet. The third button switches the portlet to the maximized window and back to the normal window state.

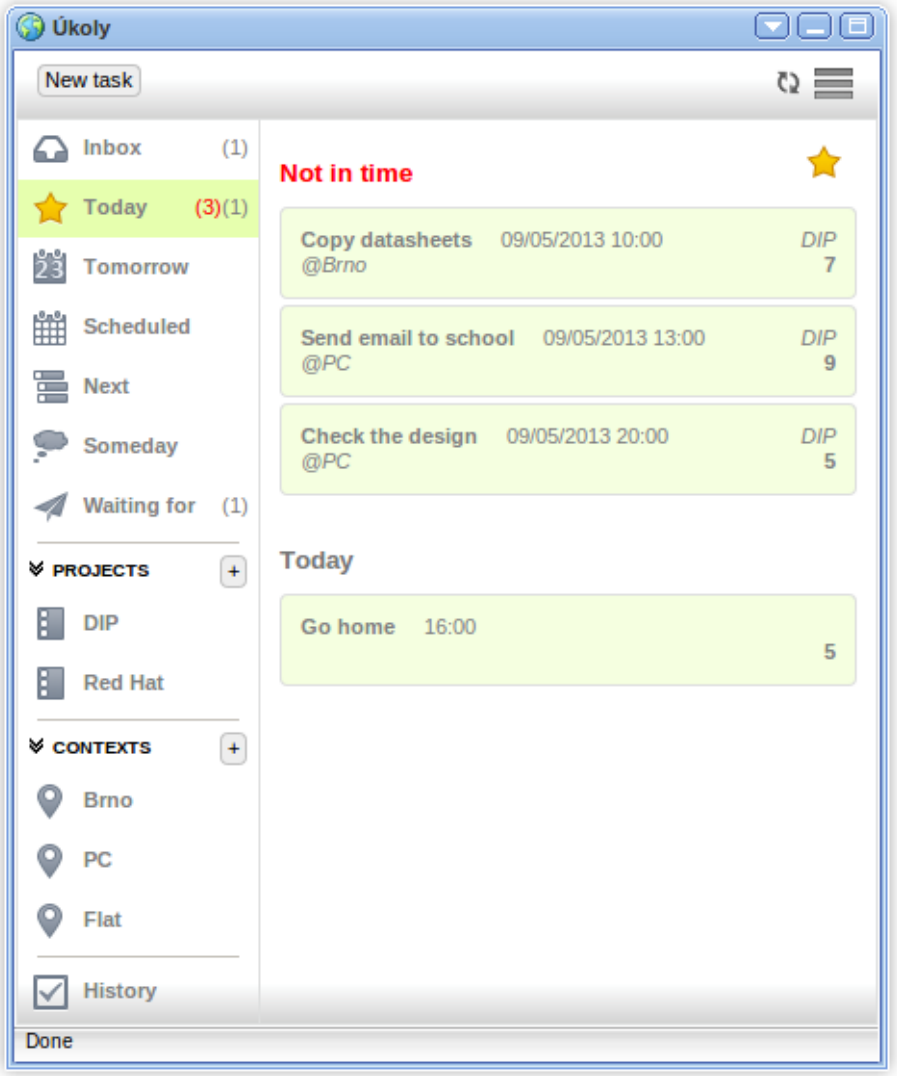


Figure C.9: Portlet Tasks in `VIEW` mode in normal window state.

VIEW MODE

Now let's focus on the VIEW mode which can be seen in Figure C.9. The window is divided into three parts – header, left block and body. In the header part, you can see button "New task", which you can use to create new tasks. Then you can see two icons on the right side of the header. First one is "Refresh" button. If any problem occurred, you can try to refresh page by this button. Second button is "Menu" button. By clicking this button, a new menu where you find several options is opened. Firstly, you can switch portlet modes (navigation panel may be missing – the setting is in portal administration) and secondly, you can open the menu for import and export of tasks. (More information about import and export are described bellow.)

The left block is divided into four parts. In the most top part, you can find menu items "Inbox", "Today", "Tomorrow", "Scheduled", "Next", "Someday" and "Waiting for". These items represent groups of tasks. Groups are named according to the technique GTD[2].

[2] http://hamberg.no/gtd/

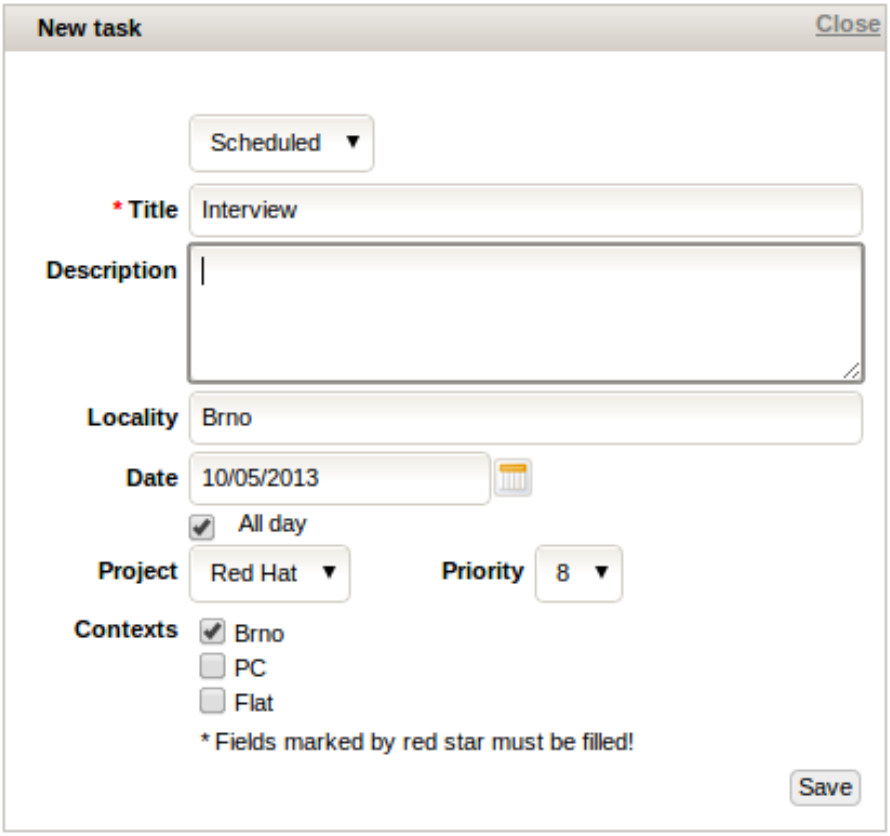


Figure C.10: New task form.

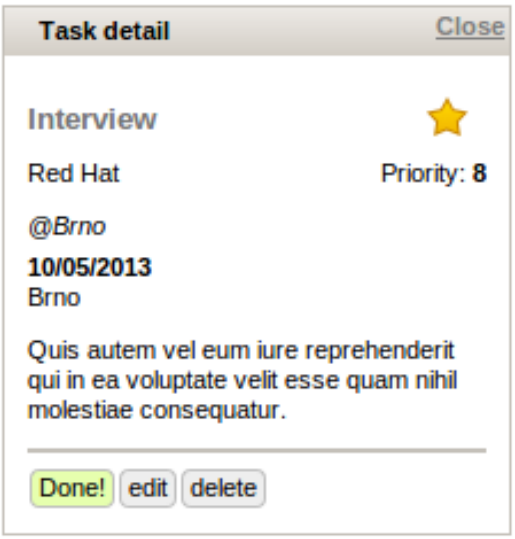


Figure C.11: Task detail.

Group “Scheduled” is in my portlet divided into three groups – “Today”, “Tomorrow” and “Scheduled”. User has better view of the most immediate tasks. Next section in the left block is “Projects” section. Each task can belong to zero or one project. New project can be created by clicking the button “+” in the header of this section. Pop-up window will be opened and you can enter name and description of a new project. If you put mouse pointer on a project item, special icon will appear on the right side of this item. By clicking on it, project menu is opened. In this menu you can find options to edit and delete project. When you delete project, all of its task will be deleted. In section “Contexts” in the left block, you can find possible contexts for your tasks. Manipulating with contexts is similar to manipulating with projects. Each task can belong to several contexts. When you delete a context, its tasks will not be deleted, only this context will be removed from their attributes. The last item in the left block is called “History” and here you can find all tasks which have been completed but not yet definitely removed.

Now let’s have a look at Figure C.10. Task form in this figure contains several common attributes of tasks such as “Title”, “Description”, “Locality” and “Priority”. The first select box in form is used to choose a GTD folder where should be a new task stored. If you choose folder “Scheduled”, new attribute “Date” will be appended. You can also define possible project and some contexts. In figure C.11, there is an example of task detail. You can see common buttons like “Delete” and “Edit”. Important button is the button “Done!”. Clicking this button makes this task completed. Completed task can be found in section “Completed”. If you open a task detail in section “Completed”, it contains the button “Open!”. By clicking it you will clear completed flag and return the task back to an original GTD folder.

You can also export all of your tasks from your portlet instance by clicking the option “Export iCalendar” in the main menu (menu can be opened by clicking on the most left icon in header). Format iCalendar does not support project attributes so, unfortunately, this information will be lost! On the other hand, contexts will be stored in attribute “Categories” of iCalendar format. In an export pop-up window, you can also find URL which can be used to import tasks to portlet Calendar. Import of events is similar to uploading of calendars.

EDIT MODE and HELP MODE

`EDIT` mode does not contain any user settings for this portlet and `HELP` mode contains link to this user manual.

C.1.3 Portlet Contacts

Portlet Contacts works in two modes which may be switched using window controllers. Controllers are placed in the upper right corner of the portlet window. The first button handles switching between portlet modes. The second button may be used for minimizing the portlet. The third button switches the portlet to the maximized window and back to the normal window state.

VIEW MODE

Now let’s focus on the VIEW mode which can be seen in figures C.12 and C.13. The window is divided into two parts – header and body. In the header part on the right side, you can see two icons. The first one is the “Refresh” button. If any problem occurred, you can

Figure C.12: Portlet Contacts in `VIEW` mode and normal window state.

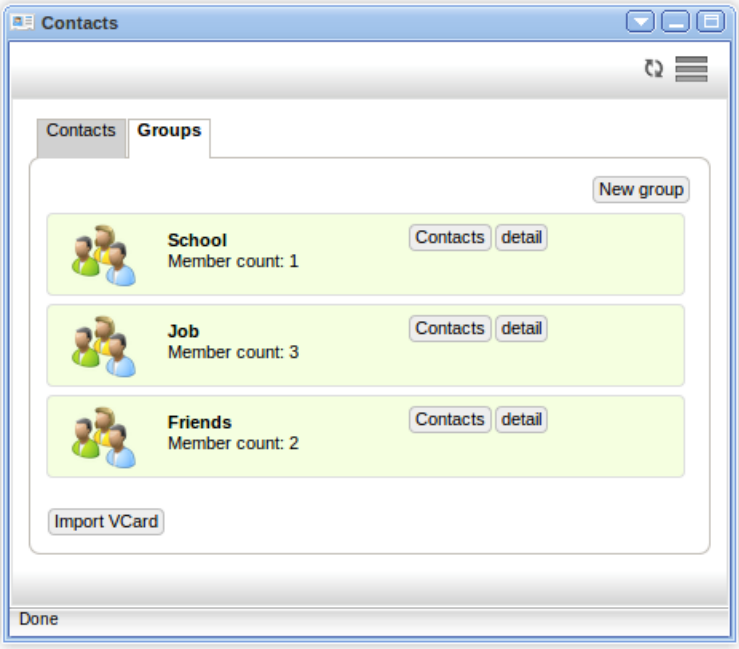


Figure C.13: Section “Groups” in portlet Contacts.

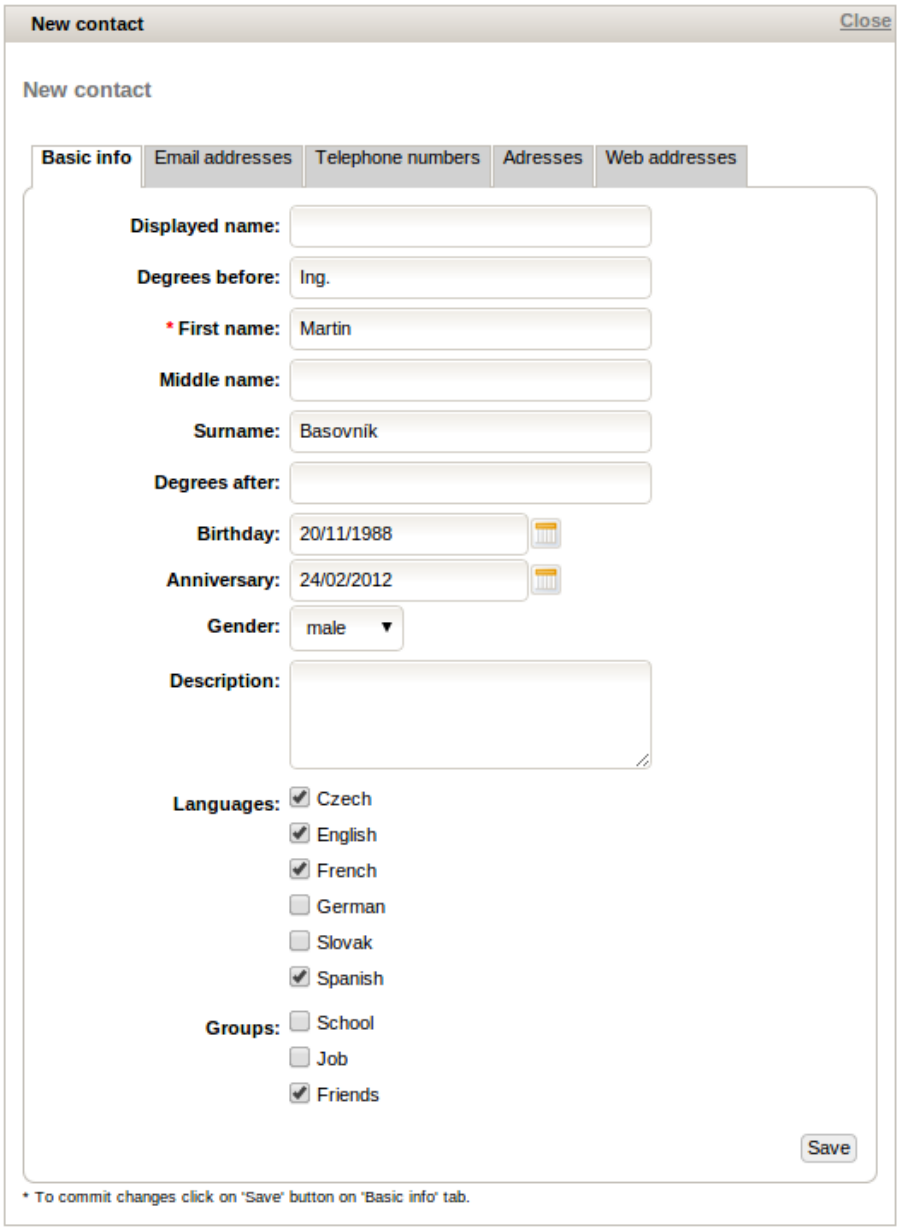


Figure C.14: New contact form.

try to refresh page by clicking this button. The second button is the "Menu" button. By clicking this button, a new menu where you can switch portlet modes is opened (navigation panel may be missing – the setting is in portal administration mode).

Body part contains tabbed panel with two tabs. The first one is called "Contacts" and the second one is called "Groups". Tab "Contacts" contains all user's contacts which can be filtered firstly by groups of contacts and secondly by string which should represent substring of name, surname or nickname. In this tab, you can also find the button "New contact" used to open pop-up window with new contact form (Figure C.14). An example of contact detail can be seen of Figure C.15.

Tab "Groups" contains your groups of contacts. Each contact can belong to several

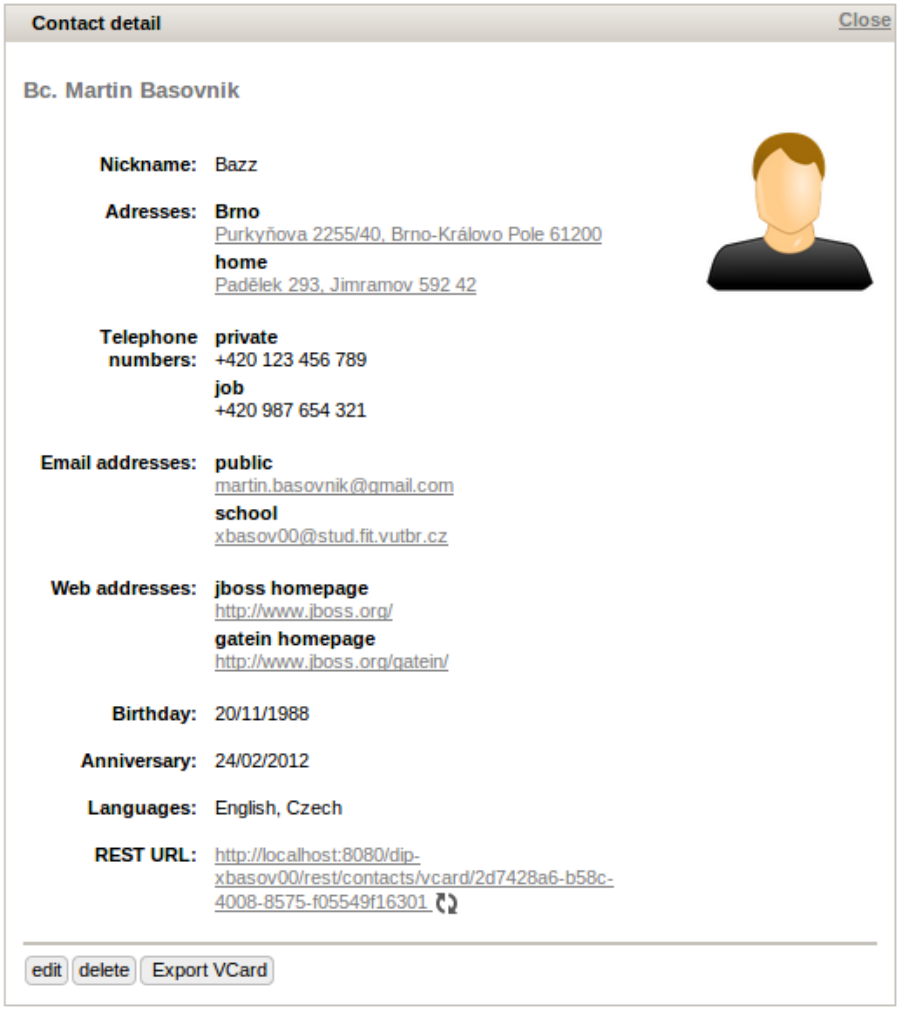


Figure C.15: Contact detail.

contact groups. By clicking the button "New group", a new group form will be displayed in pop-up window. Every group item contains two buttons. The button "Contacts" switches tab to "Contacts" and filters specified group. The second button "Detail" opens pop-up window with detail of selected group. Finally, you can see the button "Import VCard" on the bottom of current tab. By clicking it, pop-up window with upload form will be opened. You can import tasks from specified file and all of its contacts will create new group with a generated name. Name can be changed later.

EDIT MODE and HELP MODE

`EDIT` mode does not contain any user settings for this portlet and `HELP` mode contains link to this user manual.