VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

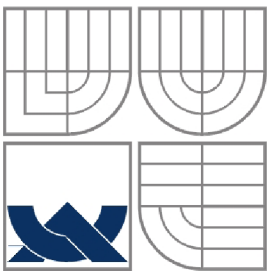
SYSTÉM PRO ŘÍZENÍ INFORMAČNÍCH TABULÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

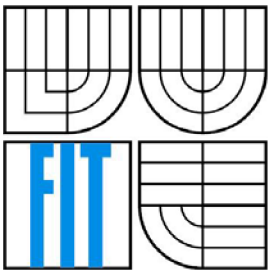
AUTOR PRÁCE
AUTHOR

PETR VOZÁK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO ŘÍZENÍ INFORMAČNÍCH TABULÍ

INFORMATION TABLE CONTROL SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

PETR VOZÁK

VEDOUCÍ PRÁCE
SUPERVISOR

ING RADOMÍR KUREČKA

BRNO 2007

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro správu informačních tabulí. K návrhu systému byl použit modelovací jazyk UML. Implementace je postavena na moderní vývojářské platformě .NET Framework.

Klíčová slova

.NET, ASP, C#, MSSQL, LCD, monitor, panel, informační tabule.

Abstract

This paper deals with design and implementation of the information table control system. It is designed using UML model language and Rational Rose development tool. System implementation is based on the modern Microsoft .NET Framework component.

Keywords

.NET, ASP, C#, MSSQL, LCD, display, information table.

Citace

Petr Vozák: Systém pro řízení informačních tabulí, diplomová práce, Brno, FIT VUT v Brně, 2007

Systém pro řízení informačních tabulí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radomíra Kurečky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Vozák
14.5.2007

Poděkování

Děkuji panu Ing. Radomíru Kurečkovi za odborné vedení diplomové práce a poskytnutí cenných rad při jejím zpracování.

Obsah

1 Úvod

Cílem této diplomové práce bylo navrhnout a vytvořit systém pro správu a řízení informačních tabulí. Informační tabulí je přitom myšlen libovolně velký LCD panel připojený k počítači. Počet připojených panelů je tedy omezen pouze možnostmi operačního systému a počtem grafických karet, resp. počtem jejich výstupů. Navržený systém měl umožňovat rozvržení plochy jednotlivých panelů na dílčí oblasti a následné plánovité zobrazování různého obsahu v těchto oblastech. Důležitou podmínkou přitom bylo, aby celý systém byl přístupný přes webové rozhraní a umožňoval tak správu libovolného panelu „na dálku“. S tím souvisel také požadavek možnosti náhledu na libovolný panel a jeho obsah. Samotné zobrazování obsahu na panelu pak mělo být nezávislé na dostupnosti internetového připojení a zajištěn tzv. „offline“ provoz.

V následující kapitole se budeme zabývat použitými technologiemi a postupy, které souvisí s funkcionalitou tohoto systému a prezentací dat na LCD panelech. V kapitole 3 popíšeme blíže návrh systému a uživatelského rozhraní a znovu si připomeneme některé hlavní požadavky na řešení a jejich vliv na celkový návrh systému. Vlastní implementací se pak zaobírá kapitola 4. V závěrečné kapitole najdeme zhodnocení systému a návrh na některá jeho rozšíření.

2 Použité technologie

Před samotným návrhem bylo nutné provést analýzu požadavků na systém. Na základě této analýzy a zjištěných informací byly zvoleny vhodné vývojové prostředky jak pro návrh tak pro samotnou implementaci celého systému. V neposlední řadě bylo potřeba se seznámit s funkcionalitou současně více zapojených monitorů k jednomu počítači a možnostmi operačního systému, které s tím souvisí.

2.1 UML

Pro návrh modelu celého systému byl zvolen grafický jazyk UML (Unified Modeling Language). Jedná se o jednotný jazyk pro specifikaci, vizualizaci, konstrukci a dokumentaci při objektově orientované analýze a návrhu. Tento jazyk není závislý na konkrétním programovacím jazyce, ve kterém bude program vyvíjen. Cílem návrhu bylo vytvoření diagramu použití (Use Case diagram).

2.1.1 Tvorba diagramu použití

Chování systému, tj. jaké funkce musí systém poskytovat, je dokumentováno diagramem Use Case, který ilustruje zamýšlené funkce systému (Use Case), jejich okolí (aktéry) a vazby mezi Use Case a aktéry.

2.1.1.1 Aktér

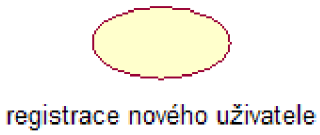
Zde je nutné uvést hlavní rozdíl mezi uživatelem (konkrétní uživatel - objekt komunikující se systémem) a aktérem (role uživatele). Jeden uživatel může vystupovat ve více rolích. Jako aktéři v systému vystupují:

- všechny externí zdroje a příjemci informací
- iniciátor případu použití
- aktér, který získává hodnotu ze systému

Notace aktéra v UML, obr. 2-1.

2.1.1.2 Use case

Use Case modelují dialog mezi systémem a aktérem. Reprezentují funkčnost, prováděnou systémem, tj. jaké schopnosti budou poskytovány systémem pro aktéra. Sada Use Case v systému stanovuje všechny definované cesty systému, které mohou být použity. Každý Use Case je také detailně dokumentován tokem událostí. Tok událostí pro Use Case je popis událostí, potřebných k dosažení požadovaného chování Use Case. V toku událostí by mělo být napsáno, co by systém měl dělat, nikoliv jak by to měl dělat.



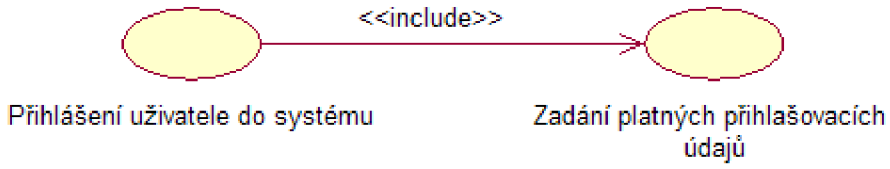
V UML je Use Case reprezentován oválem, obr. 2-2.

2.1.1.3 Vztahy mezi prvky diagramu použití

Mezi aktérem a Use Case mohou existovat vztahy asociace. Asociace může být řízena dvěma směry (aktér – Use Case, nebo Use Case – aktér). Navigační směr asociace reprezentuje, kdo je iniciátorem (počátkem) komunikace (tj. aktér je iniciátor komunikace s Use Case, Use Case je iniciátor komunikace s aktérem). Často není asociace směrována vůbec a spojení je bez šipky. V takovém případě není pro Use Case důležitý iniciátor, spolupráce může probíhat oběma směry a je důležitá jen funkcionalita, kterou Use Case popisuje. Asociace je čára spojující příbuzné prvky. Nepovinně může být asociace popsána jménem. Mezi Use Case samými mohou existovat dva typy vazeb, které pomáhají zjednodušit, zpřehlednit a strukturovat model Use Case, jsou to vztahy *include* a *extend*.

Vztah zahrnutí (include)

Případ, kdy je chování určené jedním případem použití zahrnuto do jiného případu. Takovýto způsob zápisu se používá také při opakování daného Use Case ve dvou a více oddělených případech použití.

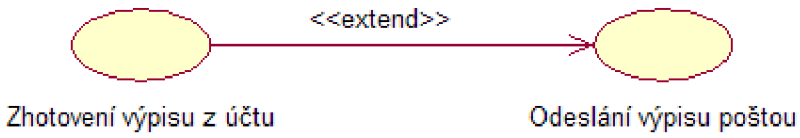


Znázornění vztahu zahrnutí, obr. 2-3.

Vztah rozšíření (extend)

Používá se ke znázornění:

- volitelného chování
- chování, které se projeví jen za určitých podmínek
- několika odlišných toků, které mohou být spuštěny výběrem aktéra

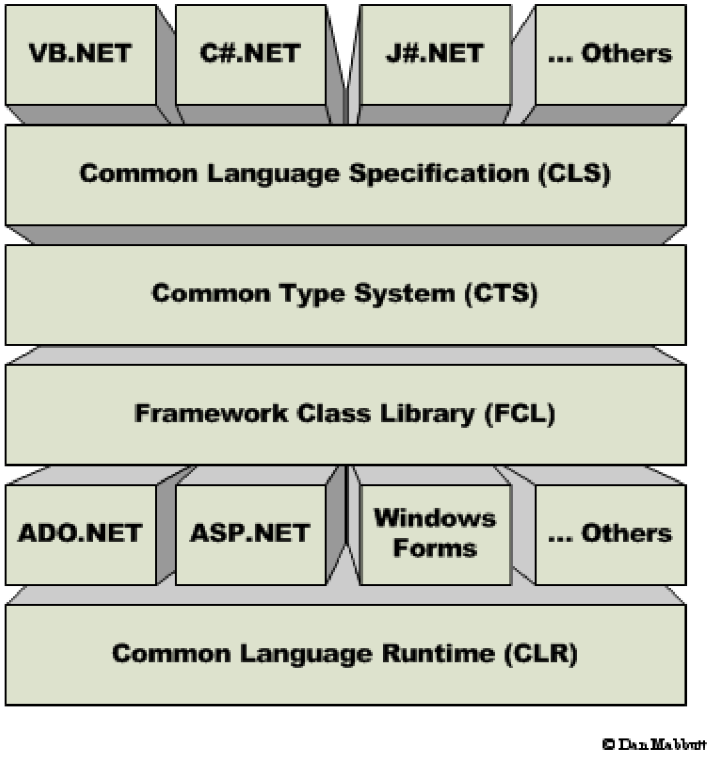


Znázornění vztahu rozšíření, obr. 2-4.

2.2 .NET Framework

V dnešní době existuje mnoho různých implementačních prostředků, které by jistě splňovaly veškeré požadavky na tvorbu tohoto systému. Pro mě však nebylo rozhodování příliš složité, zvolil jsem moderní vývojovou platformu Microsoft .NET Framework. Tato platforma je integrální součástí operačního systému Microsoft Windows pro vytváření a běh nové generace aplikací a XML webových služeb. Zajišťuje vysoce produktivní prostředí, v němž může existovat více programovacích jazyků pro snadnější vývoj různorodých aplikací. Platforma .NET Framework výrazně usnadňuje proces programové integrace s již existujícími softwarovými aplikacemi, moduly či komponenty. Je postavena na dvou základních pilířích, jimiž jsou: společné běhové prostředí (Common Language Runtime, CLR) a jednotná a hierarchicky uspořádaná knihovna tříd, která zapouzdřuje třídy pro vývoj webových aplikací pomocí ASP.NET, klientských aplikací pro Windows (pomocí Windows Forms) a databázových aplikací prostřednictvím subsystému ADO.NET.

Programátoři mohou psát své aplikace pro platformu .NET Framework v mnoha programovacích jazycích, finálním výstupem kompilátoru bude však vždy jazyková mezivrstva, která je tvořena kódem jazyka Microsoft Intermediate Language (MSIL). Takováto aplikace potom může bez potíží spolupracovat s aplikací, která byla vytvořená v jiném programovacím jazyce. Díky společnému běhovému prostředí CLR si mohou programátoři vybrat z dostupných jazyků právě ten „svůj“ programovací jazyk a to plně podle svých preferencí a charakteru řešených úloh.



Základní struktura platformy .NET Framework, obr. 2-5.

2.2.1 Programovací jazyky

Co se programovacích jazyků týče, Microsoft nabízí pro svou moderní platformu čtyři zástupce s odpovídajícími vývojovými prostředími.

2.2.1.1 Visual Basic .NET

Jedná se o nejnovější verzi populárního vývojového nástroje a jazyka, který klade důraz především na vysokou produktivitu práce. Já osobně jsem s tímto jazykem přišel do styku jen okrajově. Příliš mě neoslovil kvůli své velké „upovídanosti“ – to, co se napsalo v jazyce Visual Basic, šlo většinou napsat v jiném programovacím jazyce daleko jednodušším a elegantnějším způsobem.

2.2.1.2 Visual C++ .NET

Programovací nástroj, pro který je typická maximální síla a ovladatelnost. Pomocí tohoto jazyka mohou programátoři překlenout technologie různých platforem a sestavovat tak nativní aplikace pro Windows, jako i moderní řešení pro platformu .NET. S jazykem C++ mám o něco větší zkušenosti ovšem jeho robustnost a síla je vykoupena složitějším způsobem programování.

2.2.1.3 Visual J# .NET

Jazyk, který je zaměřen na programátory v jazyce Visual J++ a snaží se jim nabídnout úplný přístup ke zdrojům platformy .NET Framework a pokročilým XML webovým službám.

2.2.1.4 Visual C# .NET

Moderní a řekl bych až průkopnický programovací jazyk, který disponuje známou syntaxi C++ a společně s unikátními jazykovými konstrukcemi nabízí vývojářům elegantní nástroj pro stavbu aplikací na platformě .NET Framework. Tento programovací jazyk byl od samotného počátku mým velkým favoritem a jednoznačnou volbou pro vývoj tohoto systému.

2.2.2 ASP.NET

Jedná se o velmi produktivní a výkonný vývojářský nástroj pro tvorbu webových aplikací pod platformou .NET Framework. Níže se pokusím popsat a vyzdvihnout jeho největší přednosti.

2.2.2.1 Snadný programovací model

Serverové ovládací prvky dovolují uplatňovat styl programování podobný HTML s jehož pomocí lze dosáhnout stejného výsledku za použití mnohem méně kódu než tomu bylo u tradičního ASP.

2.2.2.2 Nové možnosti ve výběru a použití programovacích jazyků

Na rozdíl od ASP, které podporuje jen interpretovaný VBScript a Jscript, ASP.NET dokáže spolupracovat s více než pětadvaceti programovacími jazyky (včetně vestavěné podpory pro Visual Basic .NET, C# a Jscript .NET), což přináší velkou flexibilitu při výběru programovacího jazyka.

2.2.2.3 Výtečná podpora nástrojů

ASP.NET kód můžete psát samozřejmě i v prostém poznámkovém bloku či jakémkoliv textovém editoru. Pravou silu však poznáte při spojení s primárním vývojovým prostředím Visual Studio .NET, kde můžete využívat řady jeho výhod:

- vizuální návrh ASP.NET webových formulářů
- kvalifikovaná podpora při psaní zdrojového textu včetně dokončování příkazů
- barevné odlišení specifických partií kódu
- vestavěná podpora odlaďování ASP.NET webových aplikací
- přehledná správa celého webového projektu a dílčích knihoven

2.2.2.4 Široký rámec tříd

Techniky, které byly dříve obtížně proveditelné, nebo vyžadovaly komponenty třetích stran, mohou být v prostředí platformy .NET Framework realizovány pomocí několika řádků programového kódu. Celá kolekce tříd zapouzdřuje funkcionalitu potřebnou pro práci s XML, databázemi, práci se soubory, práci s regulárními výrazy, zasílání elektronických zpráv, atd. Možností potenciálního využití je mnohem víc.

2.2.2.5 Dynamická aktualizace spuštěných aplikací

ASP.NET umožňuje aktualizovat kompilované komponenty, aniž by musel být restartován webový server. Dříve tomu tak nebylo. Jediné co je zapotřebí udělat, je nahradit knihovnu DLL stávající komponenty – ASP.NET pak automaticky rozezná změnu a začne používat nový aplikační kód.

2.2.2.6 XML webové služby

Prostřednictvím XML webových služeb mohou aplikace komunikovat a sdílet data přes internet, bez ohledu na použitý operační systém nebo programovací jazyk. ASP.NET tento proces výrazným způsobem usnadňuje.

2.2.2.7 Podpora mobilních zařízení

ASP.NET Mobile Controls nabízejí efektivní programovatelnost mobilních telefonů, PDA a dalších mobilních zařízení. Stačí, když se aplikace napíše pouze jednou a vestavěné podpůrné nástroje a komponenty se postarají o automatické vygenerování WAP/WML, HTML podle požadavků cílového zařízení.

2.3 MS SQL Server 2005

Při práci s „dynamickými daty“, při tvorbě a obsluze webové dynamické aplikace je potřeba někde tato data ukládat – do databáze. Ideálním řešením bylo použít nejnovější produkt od Microsoftu a to sice *Miscrosoft SQL Server 2005*, přičemž již při samotném návrhu bylo myšleno také na zpětnou kompatibilitu s předchozí verzí *Miscrosoft SQL Server 2003*. Tento produkt používá relační databázový jazyk, který pracuje nad daty v podobě tabulek a sloupců. Mezi tabulkami jsou vytvářeny relace. Na základě těchto relací pak můžeme tvořit dotazy do databáze a vybírat z ní potřebná data.

2.4 Zobrazování na více monitorech

Samotný systém Microsoft Windows XP podporuje současné připojení až deseti monitorů k jednomu počítači, což umožňuje vytvoření dostatečně velké plochy, na které je možné zobrazit velké množství programů a oken. Je tedy velice snadné takto pracovat na několika úlohách současně, položky je možné přesouvat mezi monitory nebo je možné je roztáhnout na více monitorů.

2.4.1 Uspořádání více monitorů

Jeden monitor (v našem systému monitor obsluhy) slouží jako primární monitor, na němž se zobrazí dialogové okno *Přihlášení* při spuštění počítače. Na primárním monitoru zobrazí svá okna také většina programů při svém prvním spuštění. U každého monitoru je možné nastavit jiné rozlišení obrazovky a kvalitu barev. Monitory mohou být připojeny k samostatným grafickým kartám nebo ke grafické kartě, která podporuje více výstupů.

2.4.2 Dualview

Funkce Dualview umožňuje v mnoha přenosných počítačích a některých stolních počítačích (pokud grafická karta obsahuje dva grafické porty) rozšířit oblast zobrazení na druhý monitor. Tato funkce je podobná funkci použití více monitorů s tím rozdílem, že není možné zvolit primární monitor. V přenosném počítači je primárním monitorem vždy obrazovka LCD. V případě stolního počítače je primárním monitorem monitor, který je připojen k prvnímu portu grafické karty.

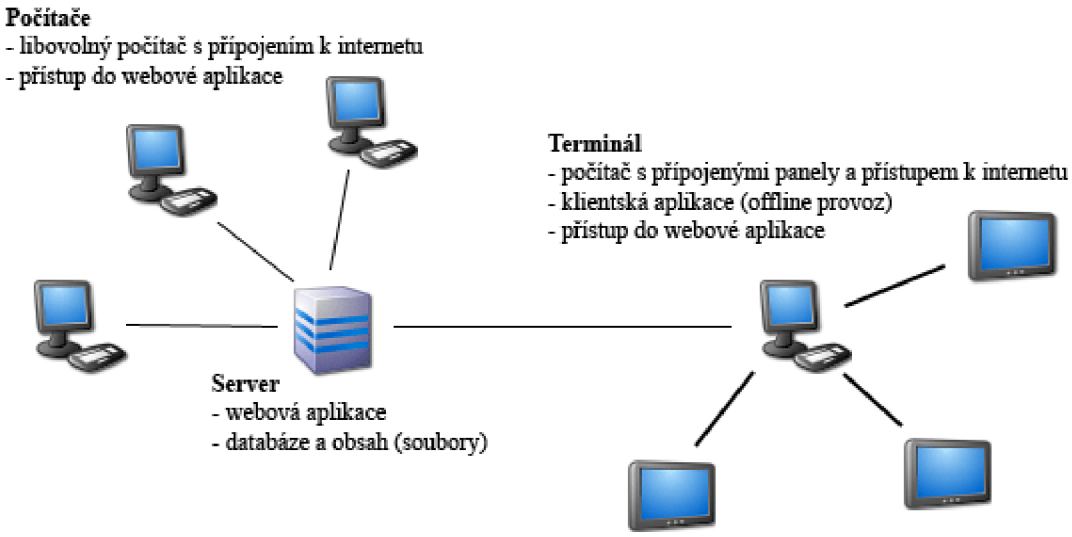
2.4.3 Další možnosti zobrazování

Za použití různých softwarových produktů třetích stran a výkonných grafických stanic lze docílit zobrazení i na více než 10 monitorech současně. Pro běžné uživatele je však „omezení“ ze strany Microsoft Windows XP plně dostačující.

Ukázka možnosti zobrazení na více monitorech, obr. 2-6

3 Návrh

Z již zmíněných základních požadavků na celý systém, především tedy správy přes webové rozhraní a umožnění offline provozu, vyplynulo základní rozdělení aplikace, a to sice na webovou aplikaci, která bude přístupná přes internetový prohlížeč a bude umožňovat spravovat data v celém systému, a klientskou aplikaci, která poběží na jednotlivých klientských počítačích, k nimž budou připojeny jednotlivé LCD panely. Úkolem klientské aplikace bude především plánovité stahování vložených a nadefinovaných dat prostřednictvím webové aplikace, jejich uložení na lokální počítač a následné zobrazování na připojených LCD panelech. V následující kapitole se blíže seznámíme s návrhem a fungováním celého systému a vzájemnou komunikací dílčích aplikací.



Zobrazení celého systému a jeho hlavních prvků, obr. 3-1.

3.1 Terminologie

Před samotným popisem návrhu bude uvedeno několik základních terminologických pojmů, na které se budu v této ale i dalších kapitolách dále odkazovat.

Schéma šablony

- definice rozvržení plochy panelu na příslušné oblasti, ve kterých se bude zobrazovat požadovaný obsah / posloupnost obsahů. Schéma je globální a lze jej přiřadit k více šablonám.

Obsah

- data (obrázek, video, flash, html soubor, odkaz na web, …), která se zobrazí v příslušné oblasti schématu. Obsah je globální a lze jej přiřadit k více šablonám.

Posloupnost obsahů

- několik přiřazených obsahů, které se v definovaném pořadí a po definovanou dobu (sekundy) cyklicky zobrazují v určité oblasti schématu. Doba zobrazení konkrétního obsahu je lokální pro danou oblast. V každé oblasti může být obsah zobrazen jinou dobu.

Šablona

- výsledný „obsah“, který se zobrazí na panelu. Definice šablony zahrnuje přiřazení právě jednoho schématu a následné přiřazení obsahu/ vytvoření posloupnosti obsahů k jednotlivým oblastem přiřazeného schématu. Šablona = schéma + obsah. Šablona je globální a lze ji přiřadit k více panelům.

Posloupnost šablon

- naplánované zobrazování šablon na panelu. Několik přiřazených šablon, které budou zobrazeny na daném panelu v daných termínech (od – do). Zobrazování není cyklické. Pokud platnost šablony vyprší, je zobrazena následující platná šablona, pokud taková šablona není, je zobrazena šablona výchozí. Platnost konkrétní šablony je lokální pro daný panel. Na každém panelu může mít šablona jinou platnost.

Terminál

- fyzický počítač, ke kterému jsou připojeny dílčí panely.

Virtuální panel

- Interpretace fyzického panelu, na kterém se má zobrazovat požadovaný obsah.

Virtuální lokalita

- Virtuální lokalita může obsahovat buď jen virtuální panely (v tomto případě interpretuje terminál) nebo může obsahovat jen další virtuální lokality (v tomto případě slouží pouze jako nadřazený uzel, což umožňuje vytvářet větvenou virtuální strukturu lokalit a panelů).

3.2 Webová aplikace

Prvním krokem při návrhu webové aplikace bylo vytvoření jejího modelu. Tento model měl jednoduchým způsobem popsat její základní funkčnost a jednotlivé činnosti a procesy, které je možno provádět. Cílem bylo vytvoření tzv. Use Case diagramu. V dalším kroku návrhu jsem se zaměřil na návrh datového modelu aplikace, tzn. vytvoření databázových tabulek a relací mezi nimi. V neposlední řadě návrh zahrnuje samotný návrh prostředí webové aplikace.

3.2.1 Use Case diagram

Use Case diagram (model celého systému), představoval základní pilíř, na kterém to všechno stálo. Jeho špatný návrh a následná implementace by později znamenalo vynaložení velkého úsilí pro docílení jakýchkoli změn ve funkčnosti systému. Proto bylo zapotřebí důkladně analyzovat veškeré požadavky na systém a promyslet jejich promítnutí do tohoto modelu. Pro samotnou tvorbu diagramu byl zvolen grafický jazyk UML a prostředí programu Rational Rose. Výsledný Use Case diagram webové aplikace představuje *Příloha 1*.

3.2.1.1 Aktéři

Aktéry, neboli činnými objekty v systému jsou pracovník (obsluha systému) a administrátor (správce systému). Obě role jsou si velice podobné a jejich úlohu v systému přiblíží následující odstavce.

Pracovník

Jeho úkolem je celková obsluha systému, která zahrnuje správu obsahů, schémat, šablon, lokalit, panelů a správu některých nastavení.

Správa obsahů

Při definici obsahu může obsluha volit mezi obsahem webovým (jako zdroj dat bude zadána webová adresa) nebo obsahem datovým (jako zdroj bude na server nahrán příslušný soubor – obrázek, video, atd.). Zde je třeba zmínit fakt, že definicí obsahu je myšleno prosté použití již existujícího zdroje dat (existující soubor, odkaz na existující webové stránky), nikoli jeho samotné vytváření. Zdroj dat je možné kdykoli změnit. Pokud se obsluha rozhodne daný obsah smazat je také zajištěno smazání příslušného datového souboru. Datové soubory jsou ukládány přímo do domovského adresáře webu do složky *Files*. Každému obsahu je přiřazen unikátní identifikátor GUID, který současně slouží k pojmenování souboru uloženém na serverovém disku. Například soubor s názvem *obrazek.jpg* bude

na disku uložen jako *<GUID>.jpg*. Zobrazování obsahu je možné kdykoli zakázat aniž by musel být smazán jeho záznam z databáze nebo datový soubor z disku. Nastavení obsahu a jeho propojení na šablony se v takovém případě uchová, obsah se jen přestane zobrazovat na všech šablonách a to bez ohledu na jejich platnost. Pochopitelně je obsluze umožněno v rámci administračního prostředí vložený obsah také prohlížet. Dále je také možné sledovat, ke kterým panelům a šablonám je daný obsah přiřazen.

Správa schémat

Schéma představuje rozvržení plochy displeje na jednotlivé oblasti (rámce), ve kterých se poté bude zobrazovat příslušný obsah. Samotný návrh schématu spočívá v definici rámců a jejich vzájemném uspořádání, které nakonec vytvoří výsledné schéma. Obsluze je také umožněno sledovat, na kterých šablonách se dané schéma používá.

Správa šablon

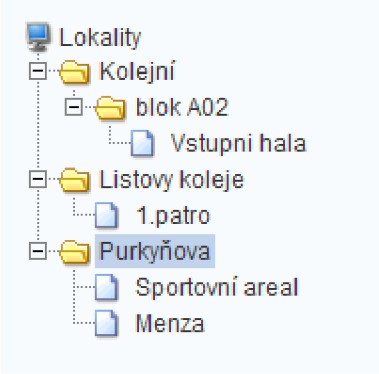
Při tvorbě šablon má obsluha jednu ze dvou možností, buď vytvoří zcela novou prázdnou šablonu (zvolí pouze schéma, které má daná šablona využívat) a nebo může vytvořit kopii již existující šablony (pokud má být výsledná šablona podobná již existující šabloně není zapotřebí ji vytvářet zcela znovu). Jakmile je šablona vytvořena, může obsluha přiřazovat do jednotlivých rámců existující obsah a vytvářet tak posloupnosti obsahů, které se budou v daných rámcích cyklicky zobrazovat. U každého obsahu může zvolit dobu jeho zobrazení a popřípadě nastavit jeho výšku a šířku pro doladění korektního zobrazení obsahu v rámci. Tyto rozměry nemusejí být totožné s originálními rozměry obrázku, videa, apod. Zvětšení/zmenšení obrázku o pár pixelů se na kvalitě žádným výrazným způsobem neprojeví. Každý obsah může být v libovolném rámci přiřazen libovolněkrát a jeho každá doba zobrazení může být taktéž různá. Samozřejmostí je možnost náhledů na vytvořenou šablonu. Obsluha může přitom zvolit náhled na libovolném z již existujících virtuálních panelů, popřípadě zvolit náhled na panelu vlastním (obsluha jednorázově zadá rozlišení panelu). Cílem náhledu je zjistit zda je všechen vložený obsah správně zobrazován a zda šablona vyhovuje rozměrům panelu.

Správa lokalit

Jak již bylo zmíněno, virtuální lokality společně s virtuálními panely vytváří stromovou strukturu. Některé lokality (terminální) mají pod sebou jen virtuální panely a jiné slouží jen jako další kontejner pro jiné lokality. Tuto stromovou struktura lokalit a panelů lze přirovnat ke stromové struktuře adresářů a souborů na disku s tím rozdílem, že pod lokalitou nemohou být současně panely a také další lokality (v adresáři mohou být jak další podadresáře tak i soubory).

Správa panelů

Při vytváření panelů je důležitým parametrem jeho rozlišení. Od něj se odvíjí jak bude přiřazená šablona na daném panelu vypadat. Šablona může mít sice procentuálně zadané rozměry jednotlivých rámců schématu, ne však všechen obsah je schopen automaticky se přizpůsobit rozdílným rozměrům panelů. Proto je většinou zapotřebí si nejdříve ujasnit rozlišení panelu a teprve potom pro něj vytvářet šablonu. Jakmile je panel vytvořen může k němu obsluha přiřazovat šablony a pro každou definovat její platnost. Takto se dá naplánovat zobrazování obsahu na libovolném panelu na libovolnou dobu dopředu. Každá šablona může být přiřazena k danému panelu libovolněkrát a v rámci každého své ho přiřazení může mít jinou platnost. Pořadí zobrazování šablon je tedy dáno jejich platností. Zobrazování není cyklické, jakmile platnost šablony vyprší, přesune se do historie zobrazených šablon u daného panelu a na panelu je zobrazena následující platná šablona. Pokud taková šablona neexistuje je zobrazena šablona výchozí, kterou může být libovolná šablona, závisí na nastavení. Přesunem šablony do historie šablona jako taková nezaniká, ta existuje stále dále a může být přiřazována k dalším panelům. Obsluha má možnost do historie zobrazovaných šablon nahlížet.



Ukázka stromové struktury lokalit a panelů, obr. 3-2.

Administrátor

Administrátor má stejné možnosti správy systému jako pracovník. Navíc má na starost správu uživatelů a jejich oprávnění. Správa uživatelů zahrnuje vytvoření uživatele a přiřazení jeho uživatelského jména a hesla, editaci a mazání uživatelů. Administrátor má možnost zamezit uživateli přístup do systému a to aniž by musel vymazat celý údaj o daném uživateli. Libovolnému uživateli může nastavit příznak administrátora, což znamená, že tento uživatel získává veškerá administrátorská práva.

3.2.2 Datový model

Datový model byl navržen s ohledem na všechny objekty, které se v systému vyskytují a na vazby, které mezi nimi existují. K tomu nám posloužil navrhnutý Use Case diagram a slovní popis jednotlivých objektů a jejich vazeb na objekty ostatní. Model systému odpovídá relační databázi, kde jsou data ukládána do tabulek mezi nimiž existují určité vztahy – relace. Sloupce tabulek představují vlastnosti objektu, který daná tabulka reprezentuje, řádky tabulek pak odpovídají jednotlivým záznamům (konkrétním objektům s konkrétními vlastnostmi). Pro tvorbu datového modelu (databázového diagramu) byl zvolen komplexní nástroj *Miscrosoft SQL Management Studio*, který současně poskytuje prostor pro vytvoření potřebné databáze a uložení dat. Součástí tohoto nástroje je také MS SQL Server, který umožňuje ostatním aplikacím k vytvořené databázi přistupovat a pracovat s jejími daty. Datový model představuje *Příloha 2*.

3.2.2.1 Databázové tabulky

V následující pasáži této kapitoly si popíšeme jednotlivé tabulky databáze, bude zmíněn význam jednotlivých sloupců tabulek a také si přiblížíme význam relací mezi jednotlivými tabulkami. U názvu tabulky je v závorce uveden název objektu, který daná tabulka reprezentuje.

Location (Lokalita)

LocationID	- jedinečný identifikátor lokality
LocationName	- jméno lokality
LocationDescription	- popis lokality
LocationParetnID	- ID nadřazené lokality
LocationIDPath	- řetězec popisující umístění lokality ve stromové struktuře lokalit

Display (Panel)

DisplayID	- jedinečný identifikátor panelu
DisplayName	- jméno panelu
DisplayWidth	- horizontální rozlišení panelu
DisplayHeight	- vertikální rozlišení panelu
DisplayDescription	- popis panelu
DisplayLocationID	- ID lokality (terminálu), ke které panel patří
DisplayOrder	- pořadí panelu v rámci všech připojených panelů k danému terminálu

DisplayIsEnabled - příznak, který určuje zda je panel aktivní, tzn. zda má nastavené všechny potřebné parametry pro to, aby na něm mohl být zobrazován obsah

Template (Šablona)

TemplateID - jedinečný identifikátor šablony
TemplateName - název šablony
TemplateDescription - popis šablony
TemplateLayoutID - ID schématu, který daná šablona využívá
TemplateIsDefault - příznak, který určuje zda je daná lokalita nastavena jako výchozí

DisplayTemplate

DisplayTemplateID - jedinečný identifikátor vztahu přiřazení šablony k panelu
DisplayID - ID panelu, kterého se daný vztah týká
Template ID - ID šablony, které se daný vztah týká
ValidFrom - datum a čas vyjadřující dobu, od které se má daná šablona na daném panelu zobrazovat
ValidTo - datum a čas vyjadřující dobu, do které se má daná šablona na daném panelu zobrazovat

Layout (Schéma)

LayoutID - jedinečný identifikátor schématu
LayoutName - název schématu
LayoutDescription - popis schématu

Frame (Rámec)

FrameID - jedinečný identifikátor rámce
FrameName - název rámce
FrameSize - rozměr (výška/šířka) rámce
FrameLayoutType - příznak rozdělení rámce (horizontálně/vertikálně)
FrameOrder - pořadí rámce v rámci nadřazeného rámce
FrameParentID - ID nadřazeného rámce
FrameLayoutID - ID schématu, ke kterému daný rámec patří
FrameIDPath - řetězec popisující umístění rámce ve stromové struktuře rámců

FrameChildrenCount - počet o jednu úroveň podřazených rámců

Content (Obsah)

ContentID - jedinečný identifikátor obsahu
ContentGUID - jedinečný identifikátor obsahu (slouží pro pojmenování souboru na disku)
ContentName - název obsahu
ContentFileName - původní název souboru
ContentExtension - původní přípona souboru
ContentUrl - webová adresa odkazující na příslušný webový obsah
ContentMimeType - MIME typ obsahu
ContentDuration - doba trvání obsahu (délka videa, animovaného obrázku apod.)
ContentWidth - šířka obsahu (šířka obrázku, horizontální rozlišení videa, apod.)
ContentHeight - výška obsahu (výška obrázku, vertikální rozlišení videa, apod.)
ContentSize - velikost obsahu v bajtech
ContentIsEnabled - příznak, který určuje zda je daný obsah aktivní, tzn. zda může být zobrazován na panelech
ContentDescription - popis obsahu

TemplateFrameContent

ID - jedinečný identifikátor vztahu mezi šablonou, rámcem a obsahem. Tento vztah vyjadřuje přiřazení obsahu do daného rámce na konkrétní šabloně.
TemplateID - ID šablony, které se vztah týká
FrameID - ID rámce, kterého se vztah týká
ContentID - ID obsahu, kterého se vztah týká
ContentOrder - pořadí obsahu v kontextu daného rámce na konkrétní šabloně
ContentActualWidth - šířka obsahu v kontextu daného rámce na konkrétní šabloně
ContentActualHeight - výška obsahu v kontextu daného rámce na konkrétní šabloně
ContentActualDuration - doba trvání obsahu v kontextu daného rámce na konkrétní šabloně

Users (Uživatelé)

UserID - jedinečný identifikátor uživatele
UserLogin - přihlašovací jméno uživatele
UserPassword - přihlašovací heslo uživatele
UserFirstName - jméno uživatele

UserLastName	- příjmení uživatele
UserEmail	- email uživatele
UserIsEnabled	- příznak, který určuje, zda se může daný uživatel přihlásit do systému
UserIsGlobalAdministrator	- příznak, který určuje zda má daný uživatel administrátorská práva

3.2.3 Uživatelské rozhraní

Návrh uživatelského rozhraní byl vytvořen z ohledem na docílení co možná největší přehlednosti a jednoduché ovladatelnosti. Z vlastní zkušenosti s některými aplikacemi a webovými stránkami vím, že pro uživatele je nesmírně důležité, aby pro něj byl systém intuitivní a přehledný a neodradil jej už jen tím jak vypadá a jakým způsobem se ovládá.

Pro návrh a umístění jednotlivých prvků uživatelského prostředí a jejich promítnutí do příslušných datových souborů byl použit nástroj *Lucid Spec*. Samotný grafický návrh prostředí byl pak vytvořen programem *Adobe Photoshop*. Ukázky uživatelského prostředí hotové aplikace představuje *Příloha 3*. V následujících kapitolách si postupně projdeme jednotlivé kroky návrhu a ukážeme si jakým (jednoduchým) způsobem bylo docíleno maximální přehlednosti a jednoduché orientace v systému.

3.2.3.1 Základní rozvržení prostředí webové aplikace

Celé okno aplikace bylo rozděleno na oblasti záhlaví, menší levou plochu a větší pravou plochu. Rád bych zde také zmínil, že toto rozdělení bylo zamýšleno implementovat pomocí rámců, což výrazným způsobem usnadnilo další návrh rozhraní, neboť do každé této oblasti bylo pak možné nahrávat obsah (zdrojový kód) v podobě celých souborů. Dalším aspektem, díky kterému jsem se přiklonil k tomuto řešení je, že při požadavku na aktualizaci konkrétního obsahu nemusí být aktualizován obsah celého okna, ale jen té plochy která aktualizaci vyžaduje, což v důsledku samozřejmě urychlí práci s aplikací.

Záhlaví

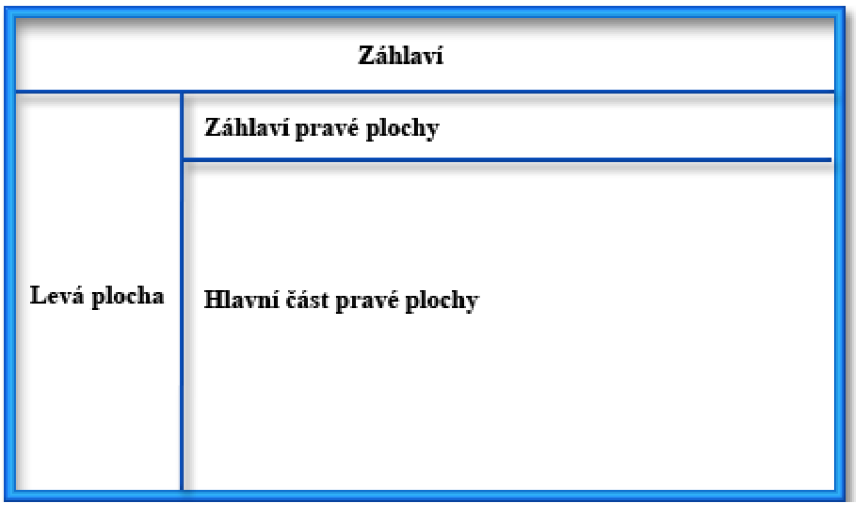
V této oblasti byly zobrazeny záhlaví záložek s odkazy na další sekce v aplikaci (sekce Panely, Administrace a Nastavení) v pravém rohu pak údaj o přihlášeném uživateli a tlačítko s možností odhlášení. Celá oblast se načítá pouze při přihlášení uživatele do systému a dále není potřeba informace v záhlaví žádným způsobem aktualizovat a znovu načítat.

Levá plocha

Jedná se o prostor, ve kterém se zobrazují menu jednotlivých sekcí, jsou to tedy: stromové menu lokalit a panelů (pokud se uživatel nacházel v sekci Panely), administrační menu (pokud se uživatel nacházel v sekci Administrace) a nebo menu nastavení (pokud se uživatel nacházel v sekci Nastavení). Tato oblast se tedy aktualizuje pouze při přejití do dané sekce, dále ji opět není potřeba znovu načítat.

Pravá plocha

Zde probíhá většina interakce mezi uživatelem a aplikací. Zobrazují se zde veškerá hlavní data, která může uživatel editovat, ať už se jedná o různé seznamy prvků (např. seznam vytvořených šablon) nebo přímo jejich detaily. U detailů prvků se ještě chvíli zastavím. Představme si například detail konkrétní šablony, který kromě základních informací o šabloně jako je název, přiřazené schéma a popis, musí poskytnout uživateli místo pro návrh šablony a zobrazení seznamu panelů, ke kterým je daná šablona přiřazena. Bylo nemožné, aby všechny tyto informace byly uživateli poskytnuty najednou v rámci pravé plochy okna. Proto jsem přistoupil k dalšímu rozdělení pravé plochy, které se uplatňuje právě při editaci detailu nějakého prvku, v našem případě šablony. Pravá plocha byla tedy rozdělena na záhlaví a hlavní část. V záhlaví pravé plochy se zobrazují informace o právě editovaném prvku spolu s odkazy na dílčí sekce editace. U šablony to jsou sekce Obecné (obecné informace o šabloně), Návrh (návrh šablony) a Panely (seznam panelů, ke kterým je šablona přiřazena). V hlavní části pravé plochy se pak zobrazuje obsah jednotlivých sekcí. Opět je zde s výhodou využito možnosti aktualizace právě té plochy, která aktualizaci skutečně potřebuje, tedy bez nutnosti znovunačtení celého okna aplikace.



Rozvržení okna webové aplikace, obr. 3-3.

3.3 Klientská aplikace

Jak již bylo zmíněno, nedílnou součástí celého systému je také klientská aplikace, která se instaluje přímo na konkrétní terminál s připojenými panely. Tato aplikace se již nestará o správu obsahu, schémat a šablon a dalších věcí, jejím hlavním úkolem je plánovitě stahovat aktuální obsah ze serveru a zobrazovat jej na konkrétních panelech. Požadavkem bylo dosažení automatického provozu bez nutnosti neustále interakce mezi uživatelem a klientskou aplikací a také zajištění „offline“ provozu.

3.3.1 Způsob zobrazování na panelech

Panely, na kterých se zobrazuje příslušný obsah, a monitor obsluhy jsou připojeny k jednomu počítači. V tomto případě se operační systém zachová tak, že přes všechny tyto monitory roztáhne jednu plochu. Otevření okna na konkrétním panelu pak spočívá ve výpočtu souřadnic levého horního rohu okna, které má být na daném panelu zobrazeno. Současně však nastává problém, kdy by mohla obsluha nechtěně přetáhnout okno jiné aplikace mimo plochu monitoru a tudíž by se toto okno (nebo jeho část) zobrazilo na panelu s obsahem, což je samozřejmě nežádoucí. Stejně nežádoucí také je přetáhnutí kursoru myši mimo plochu monitoru obsluhy. Další věc, kterou bylo třeba vzít v úvahu byl spořič displeje - když dojde k jeho aktivaci, zobrazí se na všech připojených monitorech.

3.3.2 Návrh provozu aplikace

Podobně jak tomu bylo u webové aplikace tak i zde bylo potřeba si ujasnit jednotlivé činnosti, a to jak uživatelů (obsluhy aplikace) tak aplikace samotné, která pak provádí spoustu věcí zcela automaticky. Které kroky to jsou a jak to vše funguje se dozvíte v následujících odstavcích.

3.3.2.1 První spuštění klientské aplikace

1. Aplikace sama automaticky detekuje připojené panely a jejich rozlišení.
2. Obsluha přes rozhraní webové aplikace vytvoří (pokud už tak dříve neučinila) virtuální lokalitu a odpovídající virtuální panely, navrhne rozdělení panelů na jednotlivé oblasti, přiřadí každé oblasti zobrazovaný obsah/posloupnost obsahů. Takto vytvořené šablony přiřadí k dílčím virtuálním panelům a nastaví jejich platnost.
3. V prostředí klientské aplikace požádá webovou aplikaci o zaslání aktuálního seznamu virtuálních lokalit.
4. Obsluha z tohoto seznamu zvolí požadovanou lokalitu. Po této volbě je klientské aplikaci zaslán seznam virtuálních panelů zvolené lokality.

5. Pokud obsluha seznam odsouhlasí, klientská aplikace porovná aktuálně zapojené panely a jejich rozlišení s virtuálními panely a jejich nastavením. Na případné nesrovnalosti upozorní obsluhu. Pokud sobě vše odpovídá, dojde ke „spárování" terminálu s virtuální lokalitou a toto nastavení je uloženo na lokální počítač.
6. Obsluha odsouhlasí stáhnutí aktuálních šablon pro všechny monitory.
7. Obsluha nastaví interval, který určí jak často se má klientská aplikace automaticky dotazovat webové aplikace na nové šablony nebo nový obsah.
8. Obsluha spustí zobrazování obsahu na všech panelech.

3.3.2.2 Další spuštění klientské aplikace

Následující činnosti provádí po dalším spuštění klientská aplikace zcela automaticky.

1. Zkontroluje počet a nastavení virtuálních panelů spárované lokality s nastavením a počtem skutečně připojených panelů a také zkontroluje jejich pořadí. Pokud najde nějaké nesrovnalosti upozorní obsluhu.
2. Na každém připojeném panelu otevře okno, ve kterém zobrazí data uložené šablony (data z lokálního disku, komunikace s webovou aplikaci neprobíhá) příslušného panelu.
3. Pokud pro nějaký panel aktuální data nejsou ještě stažena, požádá o jejich zaslání webovou aplikaci.
4. Po určitých intervalech se ptá webové aplikace na případná nová data pro dílčí panely. Pokud taková data existují (nová šablona, zrušení zobrazování nějakého obsahu, přidání obsahu do zobrazované posloupnosti apod.), stáhne je k sobě na lokální disk a nahradí stávající obsah příslušných panelů nově staženým obsahem.

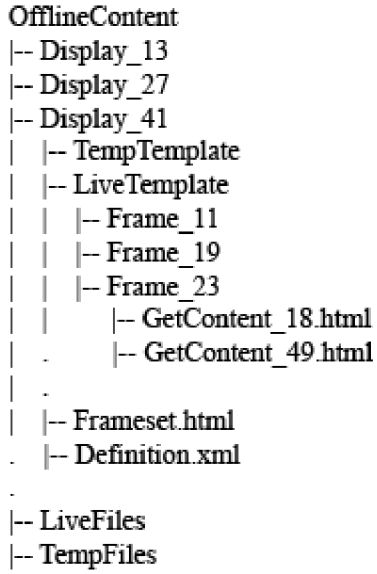
3.3.2.3 Zjišťování platnosti šablony/obsahu panelu

Následující činnosti provádí klientská aplikace opět zcela automaticky.

1. V určitých intervalech žádá webovou aplikaci o zaslání XML schématu (struktura schématu bude popsána v kapitole 4 – Implementace) platné šablony daného panelu.
2. Stáhnuté schéma porovná se schématem právě zobrazované šablony.
3. Pokud se schémata sobě rovnají pokračuje klientská aplikace se zobrazováním již stáhnutého obsahu. Pokud jsou schémata rozdílná provede se aktualizace šablony.

3.3.2.4 Offline úschovna dat

Před popisem procesu aktualizace šablony je potřeba se zmínit o struktuře úschovny stahovaných dat na lokálním disku terminálu. Všechen obsah je uložen ve složce *OfflineContent*. V ní má každý panel svůj adresář, který je rozlišen podle identifikátoru příslušného virtuálního panelu. (např. fyzický panel, který je spárován s virtuálním panelem jehož *PanelID* je rovno 41, má složku *Display_41*). Kromě dílčích složek jednotlivých panelů zde ještě najdeme složky *LiveFiles* a *TempFiles*, které jsou sdílené pro všechny panely. V obou složkách jsou uloženy soubory (obrázky, video, flash, atd.), složka *LiveFiles* obsahuje soubory, které se právě zobrazují na panelech, složka *TempFiles* pak obsahuje soubory, které se právě stahují ze serveru. Každá složka panelu obsahuje soubor s XML schématem příslušného panelu (*definition.xml*), soubor s HTML definicí rámců (*frameset.html*) a dvě další složky, *LiveTemplate* (obsahuje data aktuálně zobrazované šablony panelu) a *TempTemplate* (obsahuje data právě stahované šablony pro daný panel). V obou složkách je struktura obsahu stejná – obsahují složky s obsahem jednotlivých rámců schématu, který daná šablona využívá. Složky rámců jsou pojmenovány obdobným způsobem jako složky panelů, tedy podle identifikátorů příslušných rámců, například tedy pro rámec, jehož *FrameID* je rovno 23 má složka název *Frame_23*. V těchto složkách jsou umístěny soubory s webovým obsahem (*.html), které reprezentují posloupnost obsahů, jež se má zobrazovat v daném rámci (to jakým způsobem je zajištěno časování obsahů a načítání příslušných datových souborů ze složky *LiveFiles* se dozvíte v kapitole 4 – Implementace.)



Struktura offline úschovny dat, obr. 3-4.

3.3.2.5 Aktualizace šablony

Následující činnosti provádí klientská aplikace opět zcela automaticky.

1. Webová aplikace zašle klientovi XML schéma nové šablony pro daný panel.
2. Klientská aplikace začne podle nově staženého schématu stahovat obsah ze serveru a umisťovat jej do příslušných složek na lokálním disku.
3. Během stahování je kontrolováno jestli nenastala nějaká změna v právě stahované šabloně (změna ve schématu, obsahu, časování obsahů apod.). Pokud ano, stáhne se nové XML schéma pro daný panel a začne se podle něj stahovat znovu, přičemž datové soubory (obrázky, video, atd.), které jsou již stažené (jsou obsaženy ve složkách *LiveFiles* nebo *TempFiles*) se znovu nestahují.
4. Po úspěšném stáhnutí celé šablony a všech datových souborů je potřeba začít zobrazovat na panelu nově stažený obsah. To se provede následujícím způsobem. Stažené datové soubory se přesunou ze složky *TempFiles* do složky *LiveFiles*, panel se na chvíli vypne (zobrazí se černé pozadí), složka *LiveTemplate* příslušného panelu se smaže a složka *TempTemplate* příslušného panelu se přejmenuje na *LiveTemplate*.
5. Staré datové soubory ve složce *TempFiles* zůstávají (kdyby byly opět potřeba, tak se již nemusejí znovu stahovat). Jejich mazání je zajištěno automaticky po určité době. Toto nastavení má možnost obsluha měnit.
6. Dojde k opětovnému „zapnutí“ panelu, na kterém se začne zobrazovat nový obsah. Prodleva, která je mezi „vypnutím“ panelu a zobrazením nového obsahu, je tedy zanedbatelná, spočívá jen v době potřebné na přejmenování a smazání složky.

3.3.3 Nastavení

Klientská aplikace během svého provozu automaticky komunikuje s databázovým serverem a webovou aplikací. Iniciátorem této komunikace je vždy klientská aplikace. Ta se v nastavených intervalech spojuje s webovou aplikací nebo databázovým serverem, jimž některá data posílá a nebo naopak některá data od nich přijímá. Nastavení jednotlivých intervalů provádí obsluha před samotným spuštěním automatického provozu, jejich pozdější změna je však samozřejmě také možná. Dále má obsluha možnost nastavit interval pro mazání nepoužívaných datových souborů ze složky *LiveFiles*. V neposlední řadě se jedná o nastavení týkající se omezení pohybu oken aplikací a kurzoru myši mimo prostor pracovního monitoru obsluhy.

4 Implementace

Vytvoření dobrého návrhu systému je dozajista základním předpokladem pro jeho další fáze vývoje. Neznamená to však, že ty se mohou proto opomíjet, ba naopak. Finální fáze projektu, především tedy implementace mnohdy rozhoduje o dalším osudu celého díla. Stejně tak jak se dá v této fázi hodně získat, tak je možné také hodně ztratit. Určitě nebude odvážné prohlásit, že neexistuje program, který by neměl chyby. Jinými slovy, v každé aplikaci se dříve nebo později tyto chyby objeví, popřípadě bude potřeba provést určitá rozšíření. Proto je nutné promyslet jednotlivé postupy v implementaci tak, aby mohl být celý systém v budoucnu snadno udržován. Dalšími důležitými faktory, na které musíme ve fázi implementace aplikace myslet, je její výkon a efektivita s jakou bude plnit požadované úkoly. Mnoho věcí je většinou možné implementovat různými způsoby, volba toho správného nebývá však vždy zcela jasná a nutí nás někdy volit cestu kompromisu. Co z toho všeho zmíněného tedy vyplývá? Ani samotná fáze implementace se neobejde bez svého návrhu.

4.1 Webová aplikace

Implementace tohoto systému jednoznačně vyžadovala objektový přístup. To spočívalo především v návrhu a logickém uspořádání jednotlivých funkcí a metod do tříd (objektů), které byly pak dále podle svého charakteru umístěny do větších celků, tzv. knihoven.

4.1.1 Popis knihoven, tříd a funkcí

V této kapitole se blíže podíváme na knihovny, které využívá webové aplikace.

4.1.1.1 DataEngine

Jedná se o knihovnu, která představuje komunikační vrstvu s databází. Veškeré dotazy do databáze, volání databázových procedur a jiné operace nad databází zprostředkovává právě tato knihovna. Obsahuje výčtový typ (seznam symbolických konstant) *QueryTypeEnum* a třídu *DataConnection*.

QueryTypeEnum

Nabývá dvou hodnot, a to sice: *SQLQuery* (tato hodnota říká, že budeme po databázi chtít provést nějaký SQL dotaz) a *StoredProcedure* (tato hodnota říká, že bude požadováno provedení nějaké

uložené databázová procedury). Tyto konstanty vstupují jako parametr do některých metod třídy DataConnection.

DataConnection

Jelikož se jedná o jedinou třídu této knihovny a současně je tato třída velmi významná (obsahuje metody pro spojení s databází a práci s jejími daty), podíváme se na její funkce a vlastnosti (Properties) trochu více detailněji.

ConnectionString

Textový řetězec s potřebnými informacemi pro spojení s databází. Tento řetězec má přesně definovaný formát a je načítán z konfiguračního souboru webové aplikace. Obsahuje údaje jako je IP adresa databázového serveru, jméno databáze, ke které se hodláme připojit, přihlašovací údaje a další specifická nastavení databáze.

Connection

Představuje *SqlConnection* objekt inicializovaný patřičným spojovacím řetězcem. Každá metoda, která určitým způsobem přistupuje k databázi, musí tento objekt vytvořit.

GetSqlConnectionString()

Inicializuje *ConnectionString* hodnotou zadanou v konfiguračním souboru webové aplikace.

Open()

Otevře spojení s databází.

Close()

Ukončí spojení s databází.

IsOpen()

Indikuje stav spojení s databází (spojení je otevřeno/spojení je ukončeno).

BeginTransaction()

V rámci daného spojení s databází nastartuje databázovou transakci. Pokud je tato funkce zavolána, tak všechny další spojení s databází, která využívají stejné spojení, běží v rámci jedné transakce a v případě komplikací je možné udělat tzv. *rollback* (vrácení změn v databázi, které byly provedeny v rámci jedné transakce, do původního stavu) .

IsTransaction()

Indikuje stav transakce v rámci daného spojení s databází (transakce běží/transakce neběží).

CommitTransaction()

Potvrdí provedení všech změn v databázi v rámci jedné transakce.

RollbackTransaction()

Vrátí změny v databázi, které byly provedeny v rámci jedné transakce, do původního stavu.

ExecuteEuery()

Obsahuje tří parametry. První představuje text databázového dotazu/jméno volané databázové procedury. Druhým parametrem je pole parametrů a jejich hodnot, které vstupují do daného databázového dotazu nebo procedury. Třetím parametrem je konstanta výčtového typu *QueryTypeEnum*. Výsledkem volání této metody je buď provedení určitých změn v databázi (aktualizace, smazání) nebo navrácení požadovaných dat z databáze.

4.1.1.2 SiteProvider

Knihovna sdružující třídy, které reflektují objekty databázových tabulek, a třídy, které s těmito objekty pracují. Význam vlastností (Properties) jednotlivých objektů je stejný jako význam sloupců příslušných databázových tabulek, proto již budu uvádět jen jejich název (pro lepší přehlednost je totožný s názvem příslušného sloupce tabulky). Jejich popis naleznete v kapitole *3.2.2.1 Databázové tabulky*.

LocationInfo

Obsahuje vlastnosti LocationID, LocationName, LocationParentID, LocationDescription, LocationIDPath.

LocationInfoProvider

Třída provádějící operace nad objektem LocationInfo a tabulkou Location.

GetLocationInfo()

Na základě parametru LocationID vybere příslušný záznam z tabulky Location, ze získaných dat vytvoří objekt LocationInfo a tento objekt vrátí.

SetLocationInfo()

Vstupním parametrem je objekt LocationInfo. V tabulce Location provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteLocationInfo()

Z tabulky Location smaže záznam s příslušným LocationID.

GetLocations()

Přetížená metoda:

1. Bez parametru: Vrátí všechny záznamy z tabulky Location.
2. S parametrem: Pro specifikovanou lokalitu vrátí všechny její podřazené lokality z první úrovně (nikoli z celého stromu podřazených lokalit).

GetLocationsWithoutSublocations ()

Vrátí všechny lokality z tabulky Location, které neobsahují žádné další podřazené lokality.

DisplayInfo

Obsahuje vlastnosti DisplayID, DisplayName, DisplayWidth, DisplayHeight, DisplayDescription, DisplayOrder, , DisplayLocationID, DisplayIsEnabled.

DisplayInfoProvider

Třída provádějící operace nad objektem DisplayInfo a tabulkou Display.

GetDisplayInfo()

Na základě parametru DisplayID vybere příslušný záznam z tabulky Display, ze získaných dat vytvoří objekt DisplayInfo a tento objekt vrátí.

SetDisplayInfo()

Vstupním parametrem je objekt DisplayInfo. V tabulce Display provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteDisplayInfo()

Z tabulky Display smaže záznam s příslušným DisplayID.

GetDisplays()

Přetížená metoda:

1. Bez parametru: Vrátí všechny záznamy z tabulky Display.
2. S parametrem: Vrátí všechny panely specifikované lokality.

GetContentDisplays()

Vrátí všechny panely, na kterých je zobrazen obsah s příslušným ContentID.

GetTemplateDisplays()

Vrátí všechny panely, které používají šablonu s příslušným TemplateID.

MoveDisplayDown()

Sníží pořadí panelu s příslušným DisplayID v rámci lokality daného panelu.

MoveDisplayUp()

Zvýší pořadí panelu s příslušným DisplayID v rámci lokality daného panelu.

GetNewDisplayOrder()

Vrátí pořadí nově vkládaného panelu do lokality s příslušným LocationID.

DisplayTemplateInfo

Obsahuje vlastnosti DisplayTemplateID, DisplayID, TemplateID, ValidFrom, ValidTo.

DisplayTemplateInfoProvider

Třída provádějící operace nad objektem DisplayTemplateInfo a tabulkou DisplayTemplate.

GetDisplayTemplateInfo()

Na základě parametru DisplayTemplatetID vybere příslušný záznam z tabulky DisplayTemplate, ze získaných dat vytvoří objekt DisplayTemplateInfo a tento objekt vrátí.

SetDisplayTemplateInfo()

Vstupním parametrem je objekt DisplayTemplateInfo. V tabulce DisplayTemplate provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteDisplayTemplateInfo()

Z tabulky DisplayTemplate smaže záznam s příslušným DisplayTemplateID.

LayoutInfo

Obsahuje vlastnosti LayoutID, LayoutName, LayoutDescription.

LayoutInfoProvider

Třída provádějící operace nad objektem LayoutInfo a tabulkou Layout.

GetLayoutInfo()

Na základě parametru LayoutID vybere příslušný záznam z tabulky Layout, ze získaných dat vytvoří objekt LayoutInfo a tento objekt vrátí.

SetLayoutInfo()

Vstupním parametrem je objekt LayoutInfo. V tabulce Layout provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteLayoutInfo()

Z tabulky Layout smaže záznam s příslušným LayoutID.

GetLayouts()

Vrátí všechny záznamy z tabulky Layout.

GetLayoutDefinitionWithoutSource ()

Vrátí definici schématu s příslušným LayoutID v podobě XML dokumentu. Definice neobsahuje prvotní obsah příslušných rámců.

GetLayoutDefinitionWithSource ()

Vrátí definici schématu, který využívá šablona s příslušným TemplateID, v podobě XML dokumentu. Definice obsahuje prvotní obsah příslušných rámců.

FrameInfo

Obsahuje vlastnosti FrameID, FrameName, FrameOrder, FrameLayoutType, FrameSize, FrameParentID, FrameLayoutID, FrameChildrenCount, FrameIDPath.

FrameInfoProvider

Třída provádějící operace nad objektem FrameInfo a tabulkou Frame.

GetFrameInfo()

Na základě parametru FrameID vybere příslušný záznam z tabulky Frame, ze získaných dat vytvoří objekt FrameInfo a tento objekt vrátí.

SetFrameInfo()

Vstupním parametrem je objekt FrameInfo. V tabulce Frame provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteFrameInfo()

Z tabulky Frame smaže záznam s příslušným FrameID.

GetBaseFrameInfo ()

Pro schéma s příslušným LayoutID vrátí základní rámec v podobě FrameInfo objektu.

GetFrames()

Přetížená metoda:

1. Bez parametru: Vrátí všechny záznamy z tabulky Frame.
2. S parametrem: Vrátí všechny rámce schématu s příslušným LayoutID.

GetFramesWithoutChildren ()

Pro šablonu s příslušným TemplateID vrátí seznam rámců, které již nemají žádné další podřazené rámce.

GetFrameChildren ()

Pro schéma s příslušným LayoutID a nadřazený rámec s příslušným FrameID vrátí seznam podřazených rámců.

GetFrameNewChildOrder ()

Pro schéma s příslušným LayoutID a nadřazený rámec s příslušným FrameID vrátí pořadí nově vkládaného podřazeného rámce.

MoveFrameDown()

Sníží pořadí rámce s příslušným FrameID v kontextu nadřazeného rámce.

MoveFrameUp()

Zvýší pořadí rámce s příslušným FrameID v kontextu nadřazeného rámce.

ContentInfo

Obsahuje vlastnosti ContentID, ContentGUID, ContentName, ContentFileName, ContentExtension, ContentUrl, ContentMimeType, ContentDuration, ContentWidth, ContentHeight, ContentSize, ContentIsEnabled, ContentDescription.

ContentInfoProvider

Třída provádějící operace nad objektem ContentInfo a tabulkou Content. Součástí této třídy jsou také metody pro práci se soubory.

GetContentInfo()

Na základě parametru ContentID vybere příslušný záznam z tabulky Content, ze získaných dat vytvoří objekt ContentInfo a tento objekt vrátí.

SetContentInfo()

Vstupním parametrem je objekt ContentInfo. V tabulce Content provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteContentInfo()

Z tabulky Content smaže záznam s příslušným ContentID.

GetContents()

Přetížená metoda:

1. Bez parametrů: Vrátí všechny záznamy z tabulky Content.
2. S parametry: Vrátí všechny obsahy, jenž se nacházejí v rámci s příslušným FrameID na šabloně s příslušným TemplateID.

GetEnabledContents()

Vrátí všechny aktivní obsahy.

GetNextContentInfo ()

Vrátí aktivní obsah (objekt ContentInfo), jenž se nachází v rámci s příslušným FrameID na šabloně s příslušným TemplateID a jenž následuje za obsahem s daným pořadím. Pokud za daným obsahem již žádný další aktivní obsah není, je vrácen první aktivní obsah (tímto způsobem je zajištěno cyklení obsahů, které se uplatňuje při „online“ náhledech na panel).

GetContentContainer ()

Pro daný obsah vrátí „schránku“ v podobě HTML kódu. Typ „schránky“ záleží na typu obsahu. Například pro obrázky je vrácen klasický HTML tag `<img />` s nastavenými příslušnými parametry.

GetFile ()

Pro obsah s příslušným ContentID vrátí binární data souboru uloženého na serverovém disku.

SaveFileToDisk ()

Uloží vstupní binární data do požadovaného souboru na serverovém disku.

DeleteFile ()

Smaže požadovaný soubor na serverovém disku.

GetFileRelativePath ()

Vrátí relativní cestu k souboru v rámci daného webu.

GetFilePhysicalPath ()

Vrátí fyzickou cestu k souboru na serverovém disku.

TemplateFrameContentInfo

Obsahuje vlastnosti TemplateFrameContentID, TemplateID, FrameID, ContentID, ContentOrder, ContentActualDuration, ContentActualWidth, ContentActualHeight.

TemplateFrameContentInfoProvider

Třída provádějící operace nad objektem TemplateFrameContentInfo a tabulkou TemplateFrameContent.

Get TemplateFrameContent Info()

Na základě parametru TemplateFrameContentID vybere příslušný záznam z tabulky TemplateFrameContent, ze získaných dat vytvoří objekt TemplateFrameContentInfo a tento objekt vrátí.

SetTemplateFrameContentInfo()

Vstupním parametrem je objekt TemplateFrameContentInfo. V tabulce TemplateFrameContent provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteTemplateFrameContentInfo()

Z tabulky TemplateFrameContent smaže záznam s příslušným TemplateFrameContentID.

GetNewContentOrder ()

Vrátí pořadí nově vkládaného obsahu do rámce s příslušným FrameID na šabloně s příslušným TemplateID.

MoveDisplayDown()

Sníží pořadí obsahu v příslušném rámci na dané šabloně.

MoveDisplayUp()

Zvýší pořadí obsahu v příslušném rámci na dané šabloně.

UserInfo

Obsahuje vlastnosti UserID, UserFirstName, UserLastName, UserLogin, UserPassword, UserEmail, UserIsEnabled, UserIsGlobaAdministrator.

UserInfoProvider

Třída provádějící operace nad objektem UserInfo a tabulkou Users.

GetUserInfo()

Na základě parametru UserID/UserLogin vybere příslušný záznam z tabulky Users, ze získaných dat vytvoří objekt UserInfo a tento objekt vrátí.

SetUserInfo()

Vstupním parametrem je objekt UserInfo. V tabulce Users provede vložení/aktualizaci záznamu, který je reprezentován tímto objektem.

DeleteUserInfo()

Z tabulky Users smaže záznam s příslušným UserID.

SetPassword()

Pro daného uživatele nastaví požadované heslo. Hesla jsou ukládána do databáze v kódovaném formátu, konkrétně je použito kódování SHA1.

4.1.1.3 ITMembershipProvider

Třídy této knihovny obstarávají především správu přihlašování uživatelů do systému neboli jejich autentifikaci. K tomu je použita standardní třída ASP.NET MembershipProvider, jejíž metody jsou přepsány tak, aby mohly pracovat s uživateli uloženými v tabulce Users a současně využívat výhody objektového modelu UserInfo a metody třídy UserInfoProvider.

4.1.1.4 ITHelper

Obsahuje dvě třídy, které uchovávají a poskytují informace o aktuálně přihlášeném uživateli a o výchozí lokalitě pro daný počítač.

CurrentUserInfo

Združuje informace o řádně autentifikovaném uživateli. Dědí všechny vlastnosti třídy UserInfo a také metody rozhraní ISerializable, které umožňují danému objektu převzít kontrolu nad svou serializací a deserializací. Dále obsahuje metodu IsAuthenticated(), která indikuje zda byl daný uživatel autentifikován.

DefaultLocationInfo

Pracuje s informacemi o výchozí lokalitě pro daný počítač.

DefaultLocationID

Představuje identifikátor LocationID výchozí lokality. Toto přiřazení je možné změnit v rámci nastavení webové aplikace a lze jej provést i tehdy pokud k našemu počítači, ze kterého jsme připojeni k webové aplikaci, nejsou připojeny žádné panely. Nejedná se tedy o spárování virtuální lokality s terminálem jak jsme si to popsali již dříve, nýbrž pouze o výběr výchozí lokality pro příslušný počítač. Volba výchozí lokality nám pak usnadní pohyb v administraci a to sice tehdy, pokud se často přihlašujeme z daného počítače k webové aplikaci a měníme šablony panelů určité lokality. Pokud máme nastavenou výchozí lokalitu, tak po přechodu do sekce lokalit a panelů je nám vždy tato lokalita zobrazena jako první. Hodnota této vlastnosti je uložena do cookie na klientském počítači, ze kterého je do aplikace přistupováno.

PrimaryDisplayWidth

Hodnota této vlastnosti je opět uložena do cookie a říká jaké horizontální rozlišení má monitor počítače, ze kterého jsme přihlášeni do webové aplikace. Slouží pro správné určení šířky otevřeného okna s náhledem šablony. Minimální a současně výchozí hodnota je 800px.

PrimaryDisplayHeight

Vlastnost jejíž význam je totožný s vlastnosti PrimaryDisplayWidth, rozdíl je v tom, že místo šířky otevřeného okna s náhledem šablony určuje jeho výšku. Minimální a současně výchozí hodnota je 600px.

ITContext

Třída reprezentující pomyslný kontext webové aplikace. Obsahuje dvě vlastnosti:

CurrentUser

Přes tuto vlastnost je možné přistupovat k objektu typu CurrentUserInfo, který je ukládán do aktuální session. Z tohoto důvodu byla také u tohoto typu objektu zajištěna jeho vlastní serializace a deserializace.

DefaultLocation

Zprostředkovává přístup k informacím o výchozí lokalitě pro daný počítač.

4.1.1.5 GlobalHelper

Pomocná knihovna, jejíž třídy a metody usnadňují práci s daty, typovou kontrolu, validaci uživatelských vstupů, zpracování XML dokumentu, manipulaci a vkládání kódu v jazyce Javascript, a také práci s tzv. *cookies*.

DataHelper

Obsahuje pomocné metody pro práci s daty.

DatasourceIsEmpty()

Indikuje zda je vstupní datový zdroj prázdný či nikoli.

GetDataRowValue()

S příslušného objektu *DataRow* vrátí hodnotu požadovaného sloupce.

SetDataRowValue()

Příslušnému sloupci objektu *DataRow* nastaví požadovanou hodnotu.

ValidationHelper

Združuje pomocné metody pro validaci a přetypování různých typů dat.

IsBoolean()

Určí zda je vstupní objekt typu *bool* (booleovská hodnota *true*/*false*).

IsInteger()

Určí zda je vstupní objekt typu *int* (celé číslo).

IsDouble()

Určí zda je vstupní objekt typu *double* (desetinné číslo).

IsPositiveNumber()

Určí zda je vstupní objekt kladné číslo.

IsIdentificator()

Určí zda je vstupní objekt identifikátor (může začínat pouze písmenem, žádným jiným znakem).

IsEmail()

Určí zda vstupní objekt splňuje formát emailové adresy.

Následující metody obsahují dva vstupní parametry: objekt a výchozí hodnotu. Slouží pro přetypování objektu na požadovaný typ. Pokud se přetypování nepodaří vrátí metoda výchozí hodnotu.

GetBoolean()

Přetypuje objekt na typ *bool* (booleovská hodnota *true*/*false*).

GetInteger()

Přetypuje objekt na typ *int* (celé číslo).

GetDouble()

Přetypuje objekt na typ *double* (desetinné číslo).

GetString()

Přetypuje objekt na typ *string* (textový řetězec).

GetDateTime()

Přetypuje objekt na typ *DateTime* (datum a čas).

GetGuid()

Přetypuje objekt na typ *Guid* (jedinečný identifikátor).

Validator

Třída pro validaci uživatelských vstupů v rámci uživatelského rozhraní aplikace. Dokáže postupně validovat všechny zadané vstupy uživatele, přičemž validace se zastaví na té vstupní hodnotě, která validaci nevyhovuje. Dále se ve validaci nepokračuje a je navrácena příslušná chybová hláška, která odpovídá validovanému vstupu.

Příklad:

```
string errorMessage = new Validator().
   NotEmpty(firstName, "Zadejte prosím jméno uživatele").
   NotEmpty(lastName, "Zadejte prosím příjmení uživatele").Result;
```

Pokud je hodnota *firstName* prázdná je navrácena hláška *"Zadejte prosím jméno uživatele"*, hodnota *lastName* se již nekontroluje. Pokud jsou hodnoty *firstName* i *lastName* neprázdné, je navrácen prázdný řetězec (validace proběhla v pořádku).

Metody třídy *Validator* tedy vždy kontrolují zadaný objekt a v případě, že kontrola skončí neúspěchem, je kontrola přerušena a výsledek validátoru se nastaví na příslušnou chybovou hlášku, která je jedním ze vstupních parametrů příslušné metody:

NotEmpty()

Kontroluje zda je vstupní objekt prázdný.

IsBoolean()

Kontroluje zda je vstupní objekt booleovské hodnoty.

IsInteger()

Kontroluje zda je vstupní objekt celé číslo.

IsDouble()

Kontroluje zda je vstupní objekt desetinné číslo.

IsRegularExp()

Kontroluje zda vstupní objekt vyhovuje zadanému regulárnímu výrazu.

XmlHelper

Obsahuje pomocné metody pro práci s XML dokumentem.

XmlAttributeIsEmpty()

Pokud příslušný XML atribut existuje, vrátí metoda booleovskou hodnotu *true*. Pokud atribut neexistuje, vrátí booleovskou hodnotu *false*.

GetXmlAttributeValue()

Pokud příslušný XML atribut existuje, vrátí metoda jeho hodnotu. Pokud atribut neexistuje, vrátí příslušnou výchozí hodnotu zadanou jako druhý vstupní parametr funkce.

SetXmlNodeAttributes ()

Pro daný XML element nastaví hodnoty jeho parametrů na příslušné hodnoty. Prvním parametrem funkce je tedy příslušný element, druhým parametrem je pole názvů XML atributů a jejich hodnot, které obsahuje nebo má obsahovat požadovaný element. Metoda pak hodnoty atributů, které již existují, aktualizuje, atributy, které neexistují, vloží a nastaví na požadovanou hodnotu.

CookieHelper

Obsahuje pomocné metody pro práci s tzv. Cookies.

GetValue()

Vrátí hodnotu požadované cookie.

SetValue()

Nastaví požadovanou cookie na specifikovanou hodnotu.

Remove()

Odstraní příslušnou cookie z klientského počítače.

JavascriptHelper

Obsahuje pomocné metody pro práci s kódem v jazyce Javascript.

GetScript()

Tato metoda je využívána, když je potřeba vložit do stránky javascript na straně serveru. Vstupním parametrem je kód v jazyce Javascript. Výstupem je pak řetězec, který se skládá z párového HTML tagu pro vložení kódu v jazyce Javascript a daného vstupního kódu, který je uvnitř tohoto tagu. Pro vstupní kód `alert("ahoj")` bude výstupní řetězec následující: `<script type="text/javascript" language="javascript">alert("ahoj")</script>`.

GetAlertScript()

Využívá předchozí metody a vrací kód v jazyce Javascript pro zobrazení požadované hlášky uživateli.

4.1.1.6 Controls

Knihovna uživatelských komponent (Web Server Controls), které byly vytvořeny za účelem zjednodušení a zpřehlednění implementace webové aplikace. Jedná se o tyto komponenty: *Calendar*, *FrameView*, *Tabs*. Bližší náhled na tyto komponenty a jejich využití je součástí následující kapitoly.

4.1.2 Uživatelské komponenty

ASP.NET pro programátory připravilo velice flexibilní základnu pro vytváření komponent. Na rozdíl od ASP, kde jedinou praktickou možností jak simulovat komponentový přístup byla direktiva *#include*, poskytuje ASP.NET velice bohaté možnosti vytváření komponent, modulů a mnoha dalších. Dvěma základními typy uživatelských komponent jsou *Web User Control* a *Web Server Control*.

4.1.2.1 Web User Controls

Web User Control je v podstatě část kódu normální *.aspx* stránky, který je vložen do souboru s příponou *.ascx*. Těchto typů komponent jsem také patřičně využil při implementaci webové aplikace. Své komponenty jsem pojmenoval *PageTitle* a *Unigrid*. Nyní se na ně podívejme detailněji.

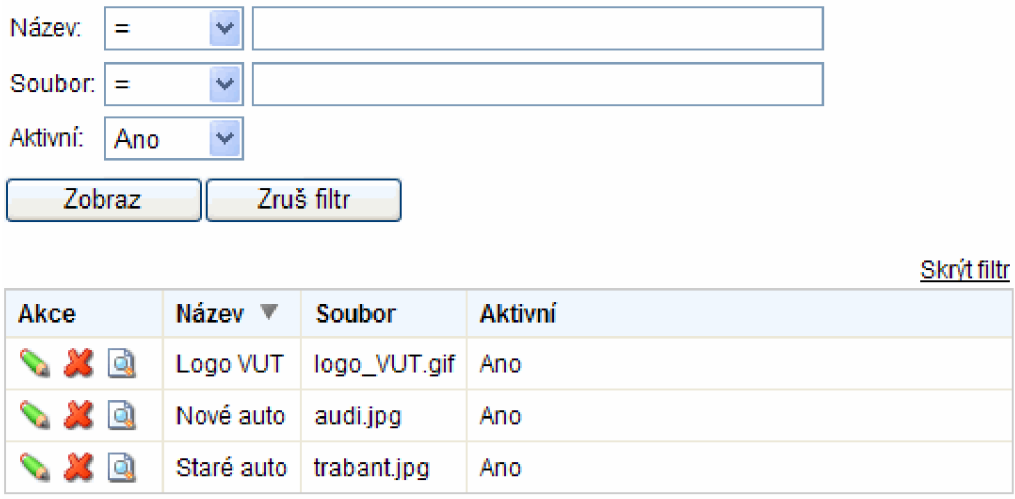
PageTitle

Tato komponenta je použita pro zobrazování nadpisu na dílčích stránkách uživatelského rozhraní. Skládá se z obrázku nadpisu, samotného textu nadpisu a navigačních odkazů. Celá komponenta a její jednotlivé prvky lze jednoduchým způsobem stylovat a docílit tak jedinečného vzhledu. Její použití a inicializace je daleko jednodušší než opakované vytváření nadpisů na každé stránce pomocí zmiňovaných částí komponenty.

Ukázka komponenty PageTitle, obr. 4-1.

UniGrid

Návrh a implementace této komponenty mně sice zabrali hodně času, ale její jednoduché a velice efektivní použití tento vynaložený čas několikanásobně vrátilo. Jedná se o komponentu na zobrazování seznamu záznamů (seznam uživatelů, seznam šablon, seznam schémat, atd.). Než se pustíme do jejího popisu, podívejme se na ni jak vypadá. To nám pomůže se v jejím dalším výkladu lépe orientovat.



Ukázka komponenty UniGrid, obr. 4-2.

Základním prvkem celé komponenty je standardní ASP.NET komponenta *GridView*, která umožňuje výpis určitého seznamu dat do tabulky. Tato komponenta byla rozšířena o:

- Filtr, který dokáže třídit zobrazované záznamy.
- Sloupec s akcemi, které jsou provedeny po kliknutí na dílčí ikonu.
- Ukazatel sloupce, podle kterého jsou záznamy v celém seznamu seřazeny.
- XML definici celé komponenty

Na XML definici celé komponenty se podíváme detailněji. Tato definice je načítána z externího souboru, který je jedním z parametrů komponenty *UniGrid*. Níže je uvedena odpovídající definice komponenty z obrázku 4-2.

```
<grid>
  <actions parameters="ContentGUID">
    <action name="edit" tooltip="Editovat" icon="edit.gif" />
    <action name="delete" tooltip="Smazat" icon="delete.gif"
      confirmation="Opravdu chcete daný obsah smazat?" />
    <action name="view" tooltip="Náhled" icon="view.gif"
      onclick="GetContent('{0}'); return false;"/>
  </actions>
  <columns>
    <column source="ContentName" type="normal" caption="Název" >
      <filter type="text" />
    </column>
    <column source="ContentFileName" type="normal" caption="Soubor" >
      <filter type="text" />
    </column>
    <column source="ContentIsEnabled" type="boolvalue" caption="Aktivní"
      width="100%">
      <filter type="boolean" />
    </column>
  </columns>
</grid>
```

Actions

Tento element definuje, které akce se zobrazí v levém sloupci akcí. Jejich pořadí je dáno pořadím v xml definici. Možnosti nastavení akcí jsou:

- Povinný jedinečný identifikátor (name).
- Popisek (tooltip), který se zobrazí po najetí myši na ikonku akce.
- Obrázek (icon), který je zobrazen na místě odpovídající akce.
- Potvrzení (confirmation), pokud je definováno zobrazí se po kliknutí na příslušnou akci hláška s příslušným textem, která vyzve uživatele k potvrzení dané akce. Pokud akci uživatel potvrdí, bude provedena. V opačném případě se nic neděje.
- Vlastní akce po kliknutí myši (onclick). V ukázkovém příkladě je volána javascript funkce *GetContent()* s parametrem, jehož hodnotu představuje hodnota ve sloupci *ContentGUID* příslušného řádku. Tato funkce zajistí zobrazení náhledu uloženého obsahu.

Columns

Obsahuje dílčí definice zobrazených sloupců. Každý sloupec může obsahovat následující parametry:

- Zdroj dat (source), který představuje jméno sloupce z datového zdroje, jehož data budou zobrazena v daném sloupci tabulky.
- Typ sloupce (type) říká jakým způsobem mají být získaná data zobrazena. *Normal* – to co je v datovém sloupci je přímo zobrazeno. *BoolValue* – v datovém sloupci jsou hodnoty True/False, zobrazeny jsou hodnoty Ano/Ne. *Custom* – hodnota datového sloupce může být doplněna o libovlný text.

- Nadpis sloupce (caption) představuje text, který je zobrazen v záhlaví daného sloupce.
- Šířka (width) sloupce, procentuální nebo pevná.
- Filtrové pole (filter - type). Zadáme u sloupců, podle jejichž hodnot chceme mít možnost filtrovat záznamy. Typ filtru může být *text* (pro zadání hodnoty se ve filtru zobrazí textové pole) nebo *boolean* (pro výběr hodnoty ve filtru se zobrazí menu s možnostmi *Ano*/*Ne*)

4.1.2.2 Web Server Controls

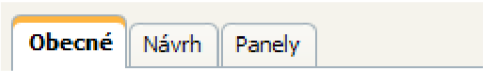
Poměrně širokou sadu těchto komponent má vývojář k dispozici již jako součást ASP.NET. Nic však nebrání tomu udělat si vlastní. Na rozdíl od komponent *Web User Controls* je jejich vytváření složitější, protože musí mít svůj výstup (HTML kód) programován ručně. Jejich použití je pak však více efektivnější. V následujících odstavcích si popíšeme komponenty, které jsem vytvořil pro použití v rozhraní webové aplikace. Jedná se o komponenty z knihovny *Controls*.

Tabs

Podobně jako komponenta *PageTitle* , tak i komponenta *Tabs* patří mezi základní, které se v rozhraní uplatnili v hojné míře. Jejím úkolem je zobrazování odkazů na jednotlivé sekce detailu daného objektu v systému (např. odkaz na sekci Návrh v detailu Šablony). Styl těchto odkazů byl upraven do záložkového vzhledu, který opět přispěl k lepší orientaci uživatele na stránce. Inicializace komponenty spočívá v inicializaci pole, jehož počet prvků odpovídá počtu požadovaných záložek.

Každý prvek pak má tyto vlastnosti:

- Zobrazený text na záložce.
- Url adresa, na kterou bude uživatel přesměrován po kliknutí na danou záložku.
- Jméno rámce, ve kterém se odkazovaná stránka otevře.
- Příznak *SELECTED*, který řekne, že má být daná záložka označena a její obsah zobrazen.



Ukázka komponenty Tabs, obr. 4-3.

Calendar

Jelikož jsem kladl na jednoduchost uživatelského rozhraní velký důraz, nemohl jsem nechat uživatele zadávat datové a časové údaje (např. při zadávání platnosti šablony) ručně. Proto vznikla komponenta *Calendar*. Základem je editovatelné textové pole s obrázkem.

Calendar – textové pole s odkazem na zobrazení kalendáře, obr. 4-4

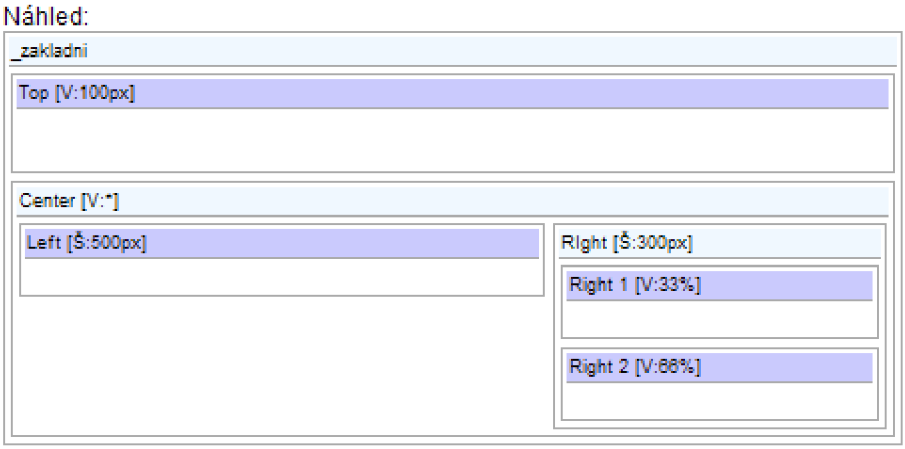
Po kliknutí na obrázek se otevře malé dialogové okno s kalendářem a možností výběru požadovaného data a času. Po výběru je tento údaj vrácen ve správném formátu do textového pole a dialogové okno je zavřeno. Textové pole má možnost uživatel dodatečně editovat. Samozřejmostí je finální kontrola formátu zadaného údaje v textovém poli.

Calendar – dialogové okno s kalendářem, obr. 4-5.

FrameView

Způsob jakým zpřístupnit uživateli náhled na právě navrhované schéma nebyl hned od začátku zcela jasný. Řešení situace ale na sebe nenechalo dlouho čekat a na světě byla komponenta *FrameView*. Ta umožňuje zobrazit náhled daného schématu v jakékoli fázi jeho návrhu. Komponenta pracuje tak, že rekurzivně do sebe vykresluje vnořené tabulky, které představují jednotlivé rámce a jejich rámce podřazené. Přitom je brán zřetel na rozměry rámců, které mohou být jak pevné tak relativní.
Vstupními parametry jsou:

- ParentFrameID – identifikátor rámce, od kterého se má začít náhled vykreslovat. V praxi je tento parametr nastaven vždy na základní rámec schématu.
- CompleteWidth – celková šířka prostoru, ve kterém se náhled vykresluje
- CompleteHeight – celková výška prostoru, ve kterém se náhled vykresluje



Náhled schématu vygenerovaný za pomoci komponenty FrameView, obr. 4-6.

Tento náhled říká, že panel bude rozdělen na oblasti záhlaví (Top) o výšce 100px a střední část (Center), jejíž výška není určena, a bude tak vysoká jak bude vysoký obsah uvnitř. Vnitřní část střední části je dále rozdělena na levou část (Left) o šířce 500px a pravou část (Right) o šířce 300px. Pravá část je rozdělena na dvě horizontální části (Right 1, Right 2), jejichž výška je zadána procentuálně a tudíž skutečná výška závisí na výšce, která bude celkem k dispozici. Z definice schématu tedy vyplývá, že takovéto schéma by bylo použitelné pro panel, který má libovolné vertikální (střední část se dokáže přizpůsobit) a pevné horizontální (800px) rozlišení. Obsah bude zobrazen v rámcích Top, Left, Right 1, Right 2.

4.1.3 Zobrazování a časování obsahu

V poslední kapitole implementace webové aplikace se seznámíme se způsobem vkládání obsahů do jednotlivých rámců schématu a s jejich časováním. Tento způsob je uplatňován jednak při náhledech šablon v rámci webové aplikaci a jednak tvoří základ časování a zobrazování obsahu pro klientskou aplikaci.

4.1.3.1 Kód schématu

Plocha monitoru je rozdělena podle nadefinovaného schématu na jednotlivé rámce. Kód, který takovéto rozvržení zajistí, je klasické HTML za použití tagů *FRAMESET* a *FRAME*. Podívejme se tedy pro lepší představu na vygenerovaný kód pro naše navrhnuté schéma podle obrázku 4-6.

```
<frameset name="frame_48" rows="100,*" >
      <frame name="frame_49" src="" />
      <frameset name="frame_50" cols="500,300" >
            <frame name="frame_65" src="" />
            <frameset name="frame_64" rows="33%,66%" >
                  <frame name="frame_66" src="" />
                  <frame name="frame_67" src="" />
            </frameset>
      </frameset>
</frameset>
```

Pro lepší přehlednost neobsahuje kód schématu doplňující parametry, např. *border="0"*, *frameborder="0"*, *noresize*. První dva z nich zamezí zobrazení rámečku kolem jednotlivých rámců, poslední pak zabrání zobrazení rolovacích lišt v rámci v situaci, kdy jeho vnitřní obsah bude větší než je samotný rámec. Tyto parametry jsou však v praxi samozřejmě v kódu použity. Kódový název rámce (name) je dán složením slova *'frame_'* a identifikátoru příslušného rámce neboli *FrameID*. Tento název slouží pro adresaci příslušných rámců při vkládání obsahu. Rámce, do kterých bude obsah vkládán, obsahují navíc parametr *src*, který zatím není nastaven. Výhoda použití tohoto kódu spočívá v možnosti načítat do jednotlivých rámců přímo celé soubory.

4.1.3.2 Zobrazování obsahu

Před popisem procesu celého náhledu šablony je potřeba objasnit, jakým způsobem je obsah zobrazován. Tento postup je použit jak při dílčích náhledech na jednotlivé obsahy, tak při vkládání obsahů do oblastí šablony. Způsob zobrazení závisí na typu obsahu.

Podporovanými typy obsahů jsou:

- Obrázek (soubory s příponou *bmp*, *jpg*, *gif*, *png*, atd.)
- Video (soubory s příponou *avi*, wmv, *mpg*, *mov*)
- Flash (soubory s příponou *swf*)
- „Živý" webový obsah (odkaz na webovou stránku, která se pak zobrazuje v dané oblasti podobně jako jiný obsah)
- Lokální webový obsah (soubor s příponou *htm*, *html*, *mht* uložený na serveru, který představuje obsah nějaké webové stránky uložený do souboru)

Zobrazení obsahu pak zajišťuje kód souboru *GetContent.aspx*:

1. Tento soubor je zavolán s parametrem *GUID*, jehož hodnota odpovídá jedinečnému identifikátoru *ContetnGUID* požadovaného obsahu.
2. Podle typu obsahu je vygenerován kód v jazyce Javascript, který zajistí zobrazení příslušného datového zdroje obsahu (obrázek, video, flash, lokální webový obsah) nebo přesměrování na požadovanou url adresu („živý" webový obsah).

4.1.3.3 Časování obsahu

Časování obsahu je zajištěno opět pomocí kódu jazyka Javascript. S výhodou je využito vestavěné funkce tohoto jazyka, která umožňuje zavolat jinou funkci za určitý časový interval zadaný v milisekundách. Současně tedy s kódem, který zajistí zobrazení příslušeného obsahu v konkrétním rámci schématu, je vygenerován kód, který zajistí načtení dalšího obsahu, jakmile uplyne doba zobrazení obsahu původního.

Příklad: `setTimeout("GetNextContent(34, 2)", 5000);`

Nejdříve se podívejme na jednotlivé hodnoty:

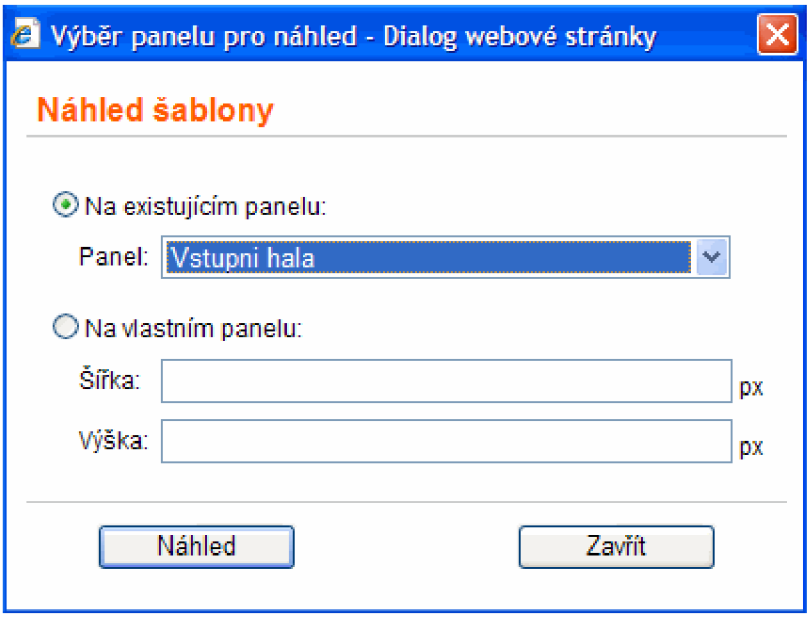
- 34 – jedinečný identifikátor rámce, ve kterém je aktuální obsah zobrazen
- 2 – pořadí aktuálního obsahu v daném rámci
- 5000 – doba zobrazení (v milisekundách) aktuálního obsahu v daném rámci

Tato ukázka tedy říká, že za 5 sekund bude do rámce jehož *FrameID* je rovno 34 nahrán první aktivní obsah, který následuje za aktuálním obsahem v tomto rámci. Pokud takovýto obsah neexistuje je nahrán obsah, který je v celé posloupnosti obsahů daného rámce na prvním místě (tím je zajištěno cyklické zobrazování obsahů). Nahrání nového obsahu je zajištěno voláním funkce *GetNextContent*.

4.1.3.4 Náhled šablony

Náhled šablony se děje v nově otevřeném okně a je spravován kódem souborů: *GetContent.aspx*, *GetDisplayContent.aspx* a *TemplateManager.aspx*. Co všechno tento proces obnáší je popsáno v následujících krocích:

1. Do náhledového okna je načten soubor *GetDisplayContent.aspx*. Tento soubor může být volán s parametry *templateID*, *displayID*, *width*, *height*. Možnosti jsou tyto:
   - jen parametr *displayID* - chceme zobrazit náhled aktuálně zobrazované (platné) šablony daného panelu
   - parametry *templateID*, *displayID* – chceme zobrazit náhled požadované šablony na požadovaném panelu. Jinými slovy, chceme zjistit jak by dané šablona vypadala na konkrétním existujícím virtuálním panelu.
   - parametry *templateID*, *width*, *height* – chceme zobrazit náhled požadované šablony na vlastním panelu, jehož rozlišení je dáno parametry *width* (horizontální rozlišení) a *height* (vertikální rozlišení)



Výběr panelu pro náhled šablony, obr. 4-7.

2. Každá z předcházejících možností vede k získání schématu šablony, jejíž náhled nám bude zprostředkován. Kód v souboru *GetDisplayContent.aspx* zajistí vygenerování HTML kódu schématu pro danou šablonu. Tento kód ještě neobsahuje odkazy na první obsahy jednotlivých rámců.
3. Vygenerovaný kód schématu je rozšířen o skrytý přídavný rámec, do kterého se později načte soubor *TemplateManager.aspx* s parametrem náhledované šablony. Kód tohoto souboru zajišťuje načítání a správné časování obsahů v jednotlivých rámcích.
4. Kód schématu je dále vložen do vnějšího rámce, který má rozměry požadované panelu (podle parametru *DisplayID*) nebo rozměry vlastní (podle parametrů *height* a *width*). Tento vnější rámec je přidán kvůli možnému přibližování a oddalování náhledu na šablonu, a to proto, aby náhledové okno vědělo, jaký poměr stran šablony má při přibližování a oddalování nastavit. Tento poměr totiž nemusí být vždy ze schématu panelu zřejmý.
5. Pro takto obohacený kód schématu je na serveru vygenerován příkaz v jazyce Javascript, který zajistí na straně klienta načtení tohoto kódu do zobrazovaného okna.
6. Po jeho načtení začne ihned pracovat kód, který je obsažen v souboru *TemplateManager.aspx*.
7. Kód tohoto souboru zajistí zobrazení prvních obsahů v jednotlivých rámcích. Současně také vygeneruje do svého rámce kód, který zajistí načtení a zobrazení následujících obsahů v každém rámci.

4.2 Klientská aplikace

V následující kapitole se již nebudeme zabývat popisem fungování klientské aplikace, tento proces je popsán v kapitole návrhu. Zmíním se zde o struktuře XML definice panelu, podíváme se na způsob zobrazování a časování staženého obsahu a na závěr bude nabídnut přehled knihoven, tříd a funkcí, které klientská aplikace využívá.

4.2.1 XML definice panelu

Jak již bylo zmíněno v kapitole návrhu, tato definice popisuje strukturu celé šablony, která má být zobrazena na příslušném panelu. Níže je uveden příklad XML definice šablony, která využívá schéma z obrázku 4-6.

```
<Display DisplayID="32" DisplayName="Hlavni panel">
  <Template TemplateID="29" LayoutID="33">
    <Frame FrameID="49" FrameName="Top">
      <Content TFCID="138" GUID="B5AB8D0D-82BB-4910-BDA9-A05B828299F3"
        Extension=".gif" Width="0" Height="100" Duration="0" Order="1"/>
    </Frame>
    <Frame FrameID="65" FrameName="Left">
      <Content TFCID="144" GUID="A1CA9DB1-533D-406A-9079-3045400F7560"
        Extension=".jpg" Width="500" Height="500" Duration="0" Order="1"/>
    </Frame>
    <Frame FrameID="66" FrameName="Right 1">
      <Content TFCID="142" GUID="A152DFF9-EBBB-4474-9EC8-ED64EADE4FF4"
        Extension=".jpg" Width="300" Height="0" Duration="30" Order="1"/>
      <Content TFCID="143" GUID="D12EAF4A-120E-4BCF-9BE8-30AED0EE9A8F"
        Extension=".jpg" Width="300" Height="0" Duration="20" Order="2"/>
    </Frame>
    <Frame FrameID="67" FrameName="Right 2">
      <Content TFCID="145" GUID="B5B1D296-2FAE-43F8-B818-ADCDB23F1A23"
        Extension=".gif" Width="300" Height="0" Duration="0" Order="1"/>
   </Frame>
  </Template>
</Display>
```

Definice tedy poskytuje informace o:

- panelu, pro který je určena (*DisplayID, DisplayName*)
- šabloně a schématu (*TemplateID, LayoutID*)
- rámcích schématu (*FrameID, FrameName*)
- zobrazovaném obsahu

Všimněme si, že definice obsahuje jen rámce, které jsou určeny pro zobrazování obsahu. Jejich rámce nadřazené již tato definice neobsahuje. Důvod je jednoduchý. Informaci o rozmístění rámců a jejich velikosti poskytuje již soubor *frameset.html*, který obsahuje html definici schématu s využitím

tagů *FRAMESET* a *FRAME* a není proto potřeba tuto informaci mít zde ještě jednou. Co zde však určitě musí být, je informace o zobrazovaném obsahu. Tuto informaci poskytuje vnořený element *Content* a hodnoty jeho atributů (v závorce jsou pak uvedeny sloupce tabulek, kterým tyto atributy odpovídají):

- *TFCID* (TemplateFrameContentID) – jedinečný identifikátor vztahu mezi šablonou, rámcem a obsahem. Tento vztah vyjadřuje přiřazení obsahu do daného rámce na konkrétní šabloně.
- *GUID* (ContentGUID) – jedinečný identifikátor obsahu
- *Extension* (ContentExtension) – přípona původního souboru obsahu
- *Width* (ContentActualWidth) – šířka obsahu v kontextu daného rámce na konkrétní šabloně
- *Height* (ContentActualHeight) – výška obsahu v kontextu daného rámce na konkrétní šabloně
- *Duration* (ContentActualDuration) – doba trvání (v sekundách) obsahu v kontextu daného rámce na konkrétní šabloně, pokud je nastavena na hodnotu "0", znamená to, že obsah je stálý a nemění se.

4.2.2 Zobrazování a časování obsahu

Vidíme, že definice poskytuje všechny potřebné informace pro to, abychom mohli stáhnout všechen potřebný obsah a soubory, které jsou nutné pro správné zobrazení celé šablony. Připomeňme si, že se jedná o soubory datové (obrázek, video, atd.) a o soubory, umístěné ve složkách jednotlivých rámců (viz kapitola 3.3.2.4 Offline úschovna dat).

Nyní se podíváme na význam a obsah těchto „druhých" souborů detailněji. Především je třeba říct, že celý jejich obsah je generován na serveru a posílán na terminál při stahování nové šablony.

Obsahem těchto souborů je:

- HTML kód (závisí na typu obsahu) s odkazem na příslušný datový zdroj obsahu – jedná se o tentýž kód, který byl vkládán příkazem v jazyce Javascript do souboru při náhledu obsahu nebo při náhledu šablony z prostředí webové aplikace. Jediný rozdíl je v odkazu na datový zdroj obsahu.
- Kód v jazyce Javascript, který zajistí načtení dalšího obsahu. Opět se jedná o tentýž kód, se kterým jsme se setkali při časování obsahů při náhledu šablony.

V čem se tedy liší zobrazování a časování obsahů při náhledu na serveru a při „živém" zobrazení na konkrétních fyzických panelech? Odpověď je jednoduchá. Při náhledu jsou veškeré obsahy a jejich časování generovány dynamicky během náhledu, tzn. během náhledu se ptáme na následující aktivní obsah, během náhledu zjišťujeme, který datový soubor patří požadovanému

obsahu, který chceme zobrazit, během náhledu získáváme kódem v jazyce Javascript HTML kód pro zobrazení daného datové zdroje obsahu. Při zobrazování obsahu na fyzickém panelu již není klientská aplikace ve spojení se serverem nebo webovou aplikací. Proto má ve stažených souborech příslušné šablony již vše dopředu nachystáno. Již nezjišťuje jak dlouho má daný obsah zobrazovat, který datový zdroj má pro tento obsah načíst a jaký obsah bude následovat. Vše je již definováno v kódu souborů, které jsou obsaženy v jednotlivých adresářích příslušných rámců. Jakmile má klientská aplikace stažen obsah pro celou šablonu, je zobrazování šablony a všech obsahů na fyzickém panelu absolutně nezávislé na internetovém připojení. Aplikace jednoduše použije to co si stáhla a zobrazí to. Lze to přirovnat tomu, když si uložíte obsah nějaké webové stránky na svůj lokální disk. Stáhnou se vám veškeré potřebné soubory (obrázky, definice CSS stylů, javascript, flash, video, atd.) plus soubor s HTML kódem stažené stránky. Pak můžete kdykoli v budoucnu bez nutnosti internetového připojení tento HTML soubor otevřít. Při jeho otevření se načtou potřebné uložené soubory a stránka se vám korektně zobrazí.

4.2.3 Zobrazování webového obsahu

Pozorného čtenáře by určitě mohla napadnout otázka jakým způsobem je docíleno zobrazení webové stránky, která je nastavena jako zdroj dat pro určitý obsah. Bylo přece řečeno, že zobrazování obsahu na fyzickém panelu je absolutně nezávislé na internetovém připojeni. Ano. Tímto problémem jsem se zabýval nezanedbatelnou dobu. Variantu detekce jednotlivých datových zdrojů ve webové stránce a jejich následné stahování jsem zavrhnul již na samém počátku úvah. Mé úmysly byly jiné. Chtěl jsem docílit zabalení kompletního obsahu příslušné webové stránky do jediného souboru, tzv. webového archivu (souboru s příponou *mht*). Jelikož stejnou možnost uložení webové stránky nabízí také internetový prohlížeč Microsoft Internet Explorer 7, bylo zřejmé, že možné to je. Za pomoci internetových vyhledávačů jsem nakonec objevil dvě knihovny, které jsem potřeboval. Jedná se o knihovny *Collaboration Data Objects* (CDO) and *Microsoft ActiveX Data Objects* (ADO). S využitím jejich funkcí jsem pak docílil požadovaného výsledku – zabalení kompletní webové stránky do jediného souboru, který posléze umožňoval korektní zobrazení celé stránky bez nutnosti internetového připojení.

4.2.4 Popis knihoven, tříd a funkcí

V závěrečné kapitole implementace se seznámíme s knihovnami, které byly navrženy pro běh klientské aplikace.

4.2.4.1 ContentManager

Sdružuje třídy, které určitým způsobem pracují s obsahem, datovými soubory obsahů a informacemi, které jsou potřeba pro správné zobrazení šablon na fyzických panelech.

ContentInfo

Obsahuje informace o zobrazovaném obsahu: *ContentGuid*, *ContentExtension*, *ContetnOrder*, *ContentDuration*, *ContentWidth*, *ContentHeight*, *ContentUrl*. Tyto informace jsou jakýmsi výtahem informací o obsahu, které jsou k jeho správnému zobrazení potřeba.

ContentInfoProvider

Obsahuje pouze dvě metody pracující nad objektem ContentInfo:

GetContentHTMLFileName()

Podle informací o daném obsahu vrátí název příslušného HTML souboru (soubor umístěný v adresáři příslušného rámce).

GetContentBinaryFileName()

Podle informací o daném obsahu vrátí název jeho datového souboru (soubor umístěný v adresáři *LiveFiles* nebo *TempFiles*). Název se skládá z identifikátoru obsahu (*ContentGuid*) a přípony souboru.

FrameInfo

Představuje informace o konkrétním rámci schématu: *FrameId*, *FrameName* a kolekce *Contents*. Tato kolekce obsahuje objekty *ContentInfo*, které reprezentují obsahy zobrazované v příslušném rámci.

LocalDisplayInfo

Objekt poskytující informace o konkrétním panelu, jehož obsah máme uložený na fyzickém disku: *DisplayId*, *DisplayName*, *TemplateId*, *LayoutId* a kolekce *Frames*. Kolekce sestává z objektů typu *FrameInfo*, které odpovídají rámcům schématu konkrétního panelu. Všechny tyto vlastnosti objektu jsou načteny ze staženého XML schématu příslušného panelu.

LocalDisplayInfoProvider

Třída pracující nad objektem *LocalDisplayInfo*. Obsahuje následující metody:

DownloadNewLocalDisplayDefinition()

Parametrem metody je identifikátor panelu *DisplayId*. Pro tento panel stáhne ze serveru novou XML definici. Na základě této definice vytoří objekt *LocalDisplayInfo* a ten vrátí.

DownloadNewLocalDisplayContentFiles ()

Parametrem metody je objekt *LocalDisplayInfo*. Na základě informací z toho objektu stáhne ze serveru definici rozvržení rámců daného panelu a uloží jej do souboru *frameset.html*. Dále stáhne HTML a datové soubory obsahů, které opět uloží do příslušných složek. HTML soubory jsou ukládány do složek příslušných rámců, datové soubory pak do složky *TempFiles*. Dále platí, že HTML soubory jsou stahovány vždy, datové soubory (obrázky, video, atd.) jsou stahovány pouze tehdy, pokud ještě neexistují ve složkách *TempFiles* nebo *LiveFiles* (mohou být už totiž využívány jiným panelem).

SetFirstContents()

Parametrem metody je objekt *LocalDisplayInfo.* Tato metoda nastaví do jednotlivých rámců definice schématu odkaz na první obsah (odkaz na příslušný HTML soubor obsahu), který se má v každém rámci zobrazit.

VirtualDisplayInfo

Objekt, který uchovává informace o „spárování“ virtuálního a fyzického panelu. Virtuální panel je popsán vlastnostmi *DisplayId, DisplayName, DisplayWidth, DisplayHeight ,* jemu odpovídající fyzický panel je pak určen vlastností *DisplayOrder*. Informace o spárování panelů, je ukládána do souboru *Location.xml*. Vlastnost *DisplayId* říká o jaký virtuální panel se jedná, vlastnost *DisplayOrder* pak říká jakému fyzickému panelu tento virtuální panel odpovídá. Fyzické panely jsou totiž identifikovatelné pouze podle pořadí v jakém jsou k terminálu připojeny.

WebDownloader

Základní třída pro stahování obsahu ze serveru. Poskytuje následující metody:

SaveTextResponseToFile()

Vstupními parametry jsou url adresa a cesta k souboru. Metoda uloží do zadaného souboru textovou odezvu na volání příslušné url adresy. Jedná se o text, který za normálních okolností přijde jakémukoli internetovém prohlížeči jako odezva na volání nějaké stránky, kterou následně zobrazí.

SaveBinaryResponseToFile()

Obdoba metody předcházející. Místo textu odezvy ukládá do příslušného souboru binární data odezvy. Tato metoda slouží pro stahování datových souborů obsahů.

SaveWebPageToMHTFile()

Vstupními parametry jsou opět url adresa a cesta k souboru. Metoda uloží kompletní obsah dané webové stránky do příslušného souboru webového archivu (souboru s příponou *mht*).

CheckWebApplicationConnection()

Slouží pro ověření spojení klientské aplikace s aplikací webovou.

4.2.4.2 DMContext

Poskytuje informace o aktuálně „spárované" virtuální lokalitě s terminálem a další metody, které s těmito informacemi pracují.

CurrentLocationInfo

Představuje objekt aktuální lokality, který je definován těmito vlastnostmi:

- LocationId – identifikátor odpovídající virtuální lokality
- LocationName – název odpovídající virtuální lokality
- VirtualDisplays – kolekce objektů *VirtualDisplayInfo*
- LocalDisplays – kolekce objektů *LocalDisplayInfo*

Dále obsahuje následující metody:

LoadVirtualDisplays()

Pro každý pár virtuální – fyzický panel z definice přiřazení, která je uložena v souboru *Location.xml,* vytvoří objekt *VirtualDisplayInfo*. Tyto objekty postupně vloží do kolekce *VirtualDisplays*

SaveVirtualDisplays()

Informace o přiřazení virtuálních panelů k panelům fyzickým (informace z kolekce *VirtualDisplays*) převede do XML formátu a uloží do souboru *Location.xml*.

LoadLocalDisplays()

Pro každý objekt *VirtualDisplayInfo* z kolekce *VirtualDisplays* načte z disku XML definici příslušného panelu a z ní vytvoří objekt *LocalDisplayInfo*. Z těchto objektů postupně vytvoří kolekci *LocalDisplays*.

4.2.4.3 GlobalHelper

Pomocná knihovna. Obsahuje několik tříd, které najdeme také v knihovně *GlobalHelper* webové aplikace. Jsou to tyto: *DataHelper*, *ScriptHelper*, *ValidationHelper*, *XmlHelper*. Jejich funkce a význam je tedy stejný jako u stejnojmenné knihovny webové aplikace. Navíc zde ale najdeme třídu *FileHelper*.

FileHelper

Obsahuje metody pro získání názvů a fyzických cest jednotlivých složek a souborů „Offline úschovny dat" (viz kapitola 3.3.2.4 Offline úschovna dat). Jelikož tyto složky existují vždy ve dvou variantách (jedny pro aktuálně zobrazovaný obsah, druhé pro právě stahovaný obsah), je součástí každá metody parametr, kterým určíme požadovanou variantu.

GetContentFolderPath()

Vrátí fyzickou cestu složky pro datové soubory obsahů, tedy kompletní cestu ke složce *LiveFiles* nebo *TempFiles*.

GetDisplayFolderPath()

Vrátí fyzickou cestu složky požadovaného panelu.

GetTemplateFolderPath()

Pro daný panel vrátí fyzickou cestu složky pro uložení obsahu šablony, tedy kompletní cestu ke složce *LiveTemplate* nebo *TempTemplate* příslušného panelu.

GetFrameFolderPath()

Pro požadovaný rámec konkrétního panelu vrátí fyzickou cestu složky pro uložení obsahu rámce.

GetDisplayFramesetDefinitionFilePath()

Vrátí fyzickou cestu k definici schématu zadaného panelu, tedy cestu k souboru *frameset.html* příslušného panelu.

GetDisplayFramesetDefinitionFilePath()

Vrátí cestu k XML definici panelu, tedy cestu k souboru *definition.xml* příslušného panelu.

4.2.4.4 SettingsManager

Veškerá nastavení, která se týkají chodu klientské aplikace jsou ukládána přímo do konfiguračního souboru aplikace. O správu těchto nastavení se stará tato knihovna.

DMKeys

Každé nastavení je specifikováno svým jedinečným názvem. Tato třída obsahuje konstanty, které představují názvy jednotlivých nastavení v konfiguračním souboru.

SettingsProvider

Obsahuje metody pro získání/změnu hodnot jednotlivých nastavení.

GetKeyValue()

Vrátí hodnotu požadovaného nastavení.

SetKeyValue()

Změní požadované nastavení na zvolenou hodnotu.

5 Závěr

Cílem této práce bylo navrhnout a implementovat systém pro prezentaci dat na LCD panelech. Tohoto cíle bylo úspěšně dosaženo. Přestože implementovaný systém splňuje veškeré počáteční podmínky a je plně funkční, je možné uvažovat o jeho dalších rozšířeních.

Jedním z požadavků na celý systém bylo dosažení určitého stupně automatizace. Ta spočívala v automatickém zobrazení obsahu na připojených panelech po startu aplikace a následné jeho automatické aktualizaci. Rozšíření je možné tedy vidět v ještě větší automatizaci celého systému, například tedy v možnosti automatické aktualizace klientské aplikace. Dalšími směry, kterými je možné se ubírat při úvahách o rozšíření systému jsou vzdálená správa klientské aplikace (spuštění počítače a klientské aplikace) a připojených panelů (zapnutí panelů). Samotné spuštění celého procesu zobrazování by tak nemuselo být vázáno pouze na jeden konkrétní počítač, k němuž jsou připojeny panely. Dále je samozřejmě možné se zaměřit na rozšíření podpory typů zobrazovaných obsahů, včetně podpory tzv. „streamovaného videa“, nebo indikaci stavů připojených panelů.

A čím byla práce přínosem pro mě samotného? V praxi jsem si prošel celým procesem vývoje většího systému, prohloubil si znalosti prostředí .NET Framework a seznámil se s novými postupy a možnostmi, které souvisí s prezentací dat na LCD panelech. Nejvíce si cením zkušeností s prací s více současně připojenými panely k jednomu počítači a zobrazováním a časováním jednotlivých obsahů. S tím souviselo také nemalé množství překážek, s nimiž jsem se musel vypořádat. Jednalo se například o zamezení pohybu myši a oken mimo plochu pracovního monitoru (dosaženo pomocí kontroly pozice aktivního okna a kurzoru s využitím funkcí knihovny *ManagedWinapi*), offline provoz systému (dosaženo za pomoci lokální úschovny dat), stahování kompletního obsahu webové stránky do jediného souboru – webového archivu, detekci připojených panelů a jejich parametrů (dosaženo za pomoci knihovny *System.Windows.Forms.Screens* a jejich metod) nebo o samotné zobrazování obsahů v rámcích tak, aby obsahy při své vzájemné výměně neproblikávaly (dosaženo za pomoci tzv. „asynchronních callbacků“). Důraz byl také kladen na vytvoření intuitivního a přehledného uživatelské rozhraní, čehož bylo dosaženo díky efektivním uživatelským komponentám a důmyslné navigaci v prostředí celé aplikace. V některých případech bylo hledání řešení daného problému opravdu složité. V jiných případech se řešení nabízelo hned několik a jejich postupnou analýzou jsem pak vybral to nejvíce vyhovující. Ať už celý proces návrhu a implementace skýtal všemožná úskalí a překážky, troufám si říct, že se mi je podařilo úspěšně překonat.

Literatura

[1] Evjen, B., Hanselman, S., Muhammad, F., Sivakumar, S., Rader, D. *ASP.NET 2.0 Programujeme profesionálně*, Computer Press.

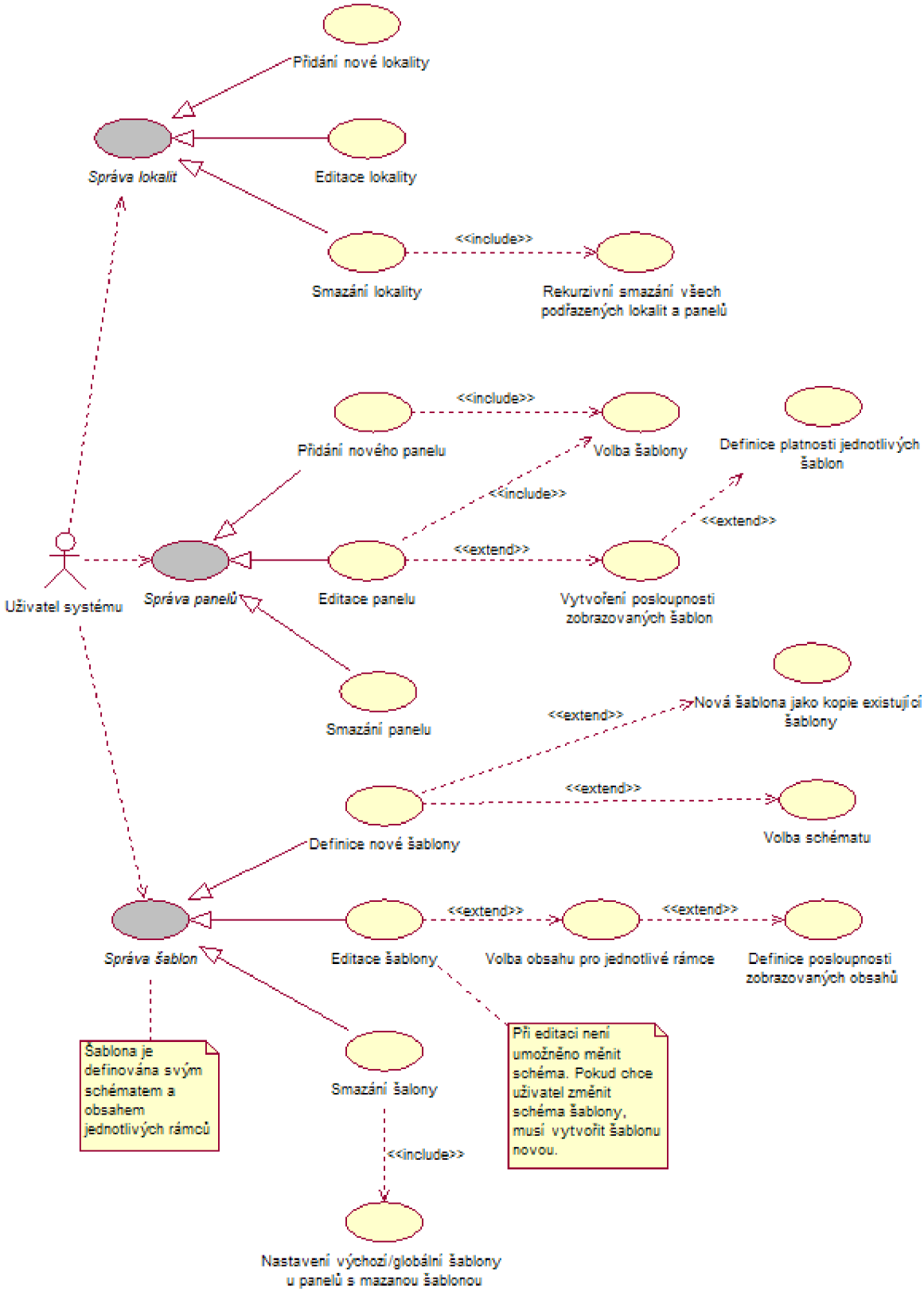
[2] Kačmář, D. *Programujeme .NET aplikace*, Computer Press 2001.

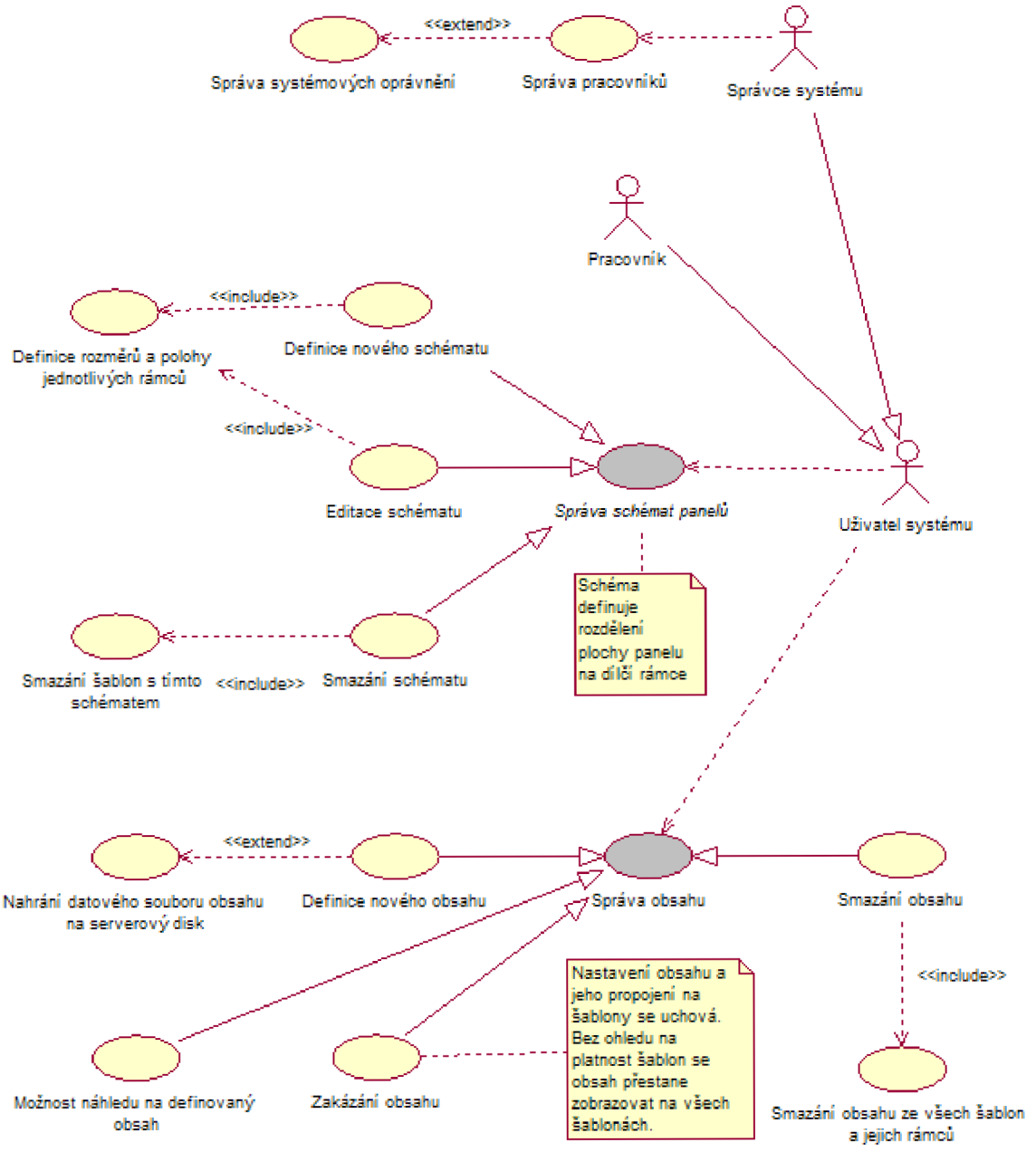
[3] MacDonald, M. *ASP.NET 2.0 a C#*, Zonner Press.

[4] Web: *www.msdn.com*

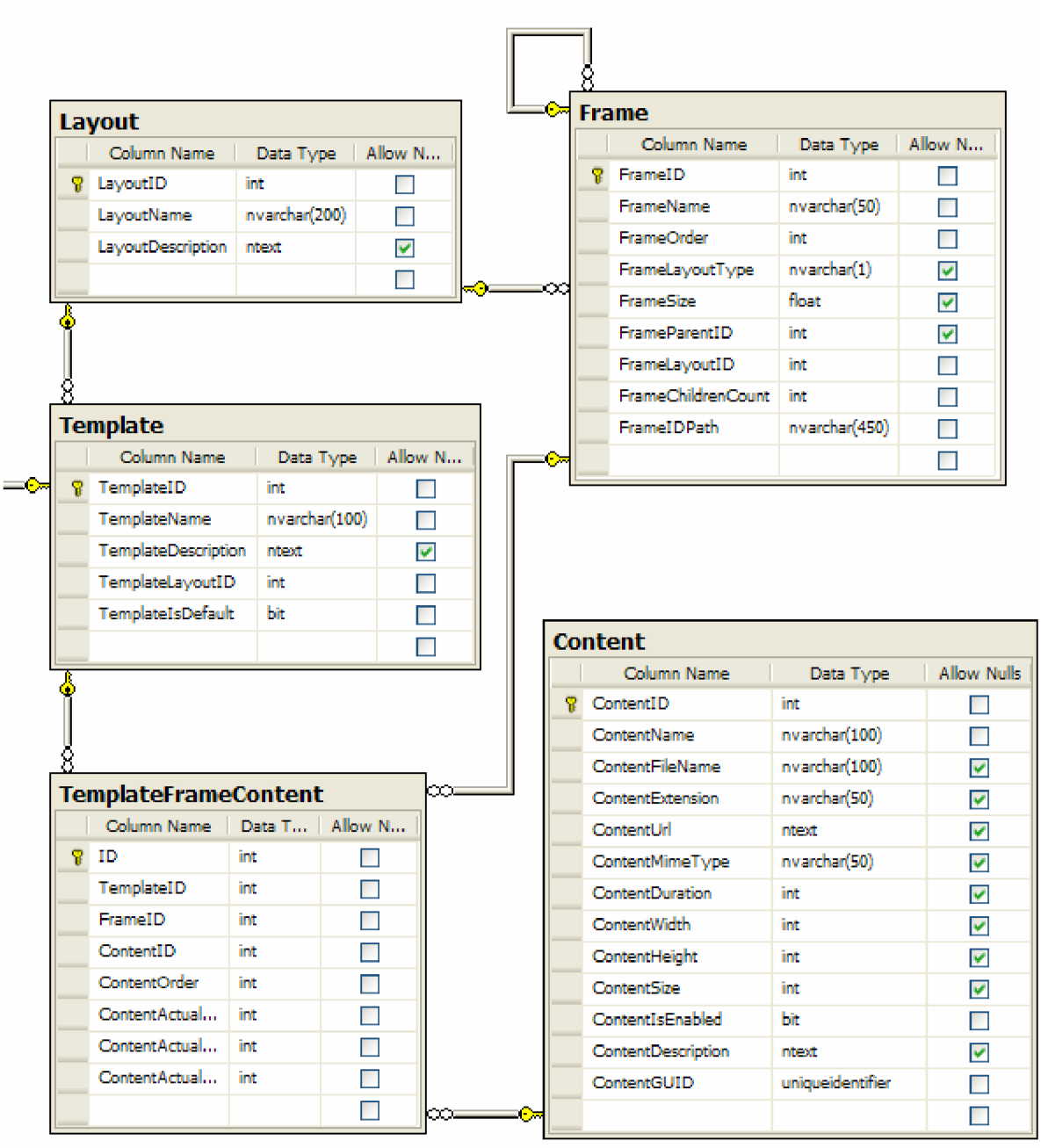
Seznam příloh

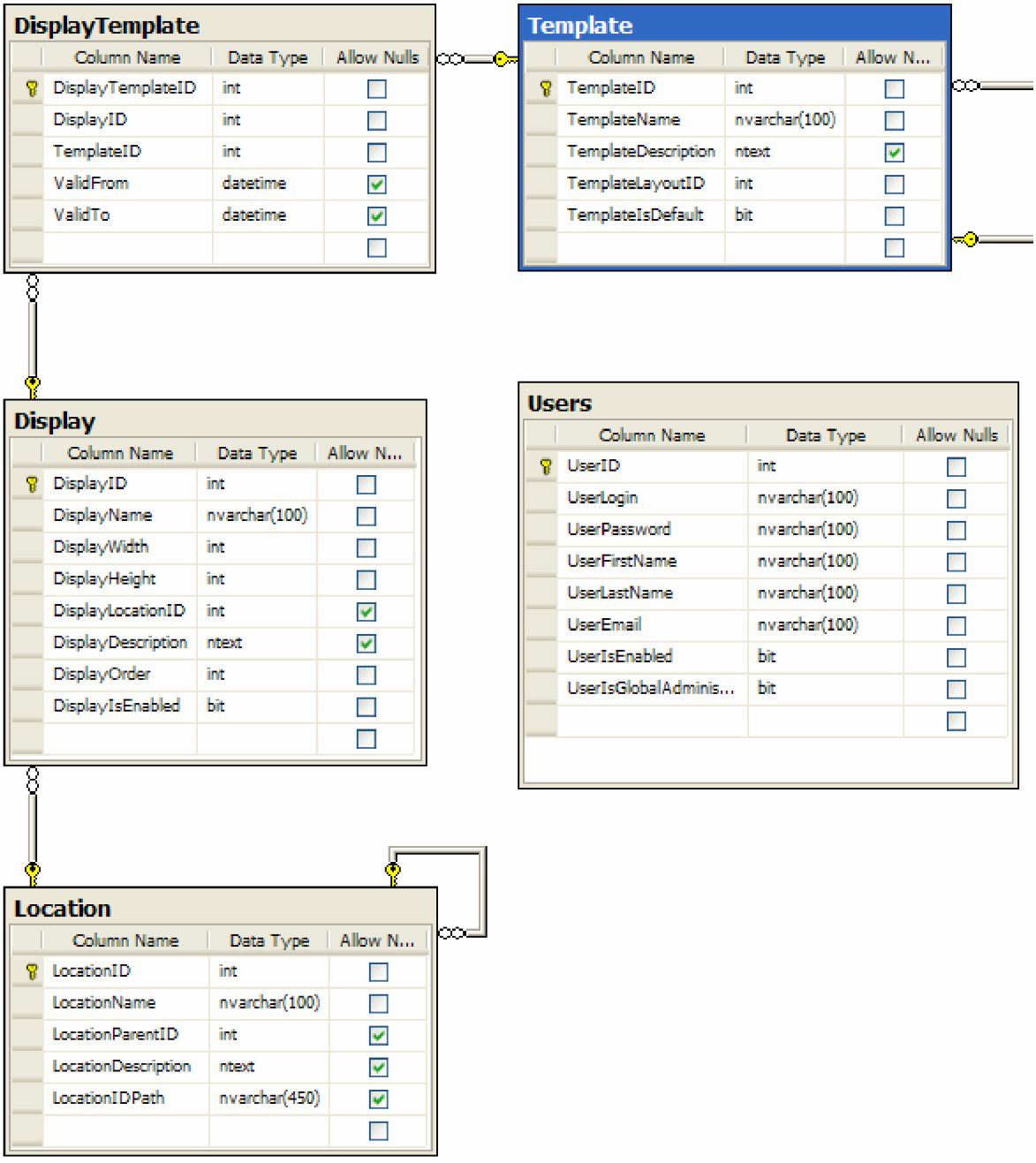
Příloha 1


Přidání nové lokality
Správa lokalit
Editace lokality
Smazání lokality
<<include>>
Rekurzivní smazání všech podřazených lokalit a panelů
Přidání nového panelu
<<include>>
Volba šablony
Definice platnosti jednotlivých šablon
<<include>>
<<extend>>
Uživatel systému
Správa panelů
Editace panelu
<<extend>>
Vytvoření posloupnosti zobrazovaných šablon
Smazání panelu
Nová šablona jako kopie existující šablony
<<extend>>
<<extend>>
Definice nové šablony
Volba schématu
Správa šablon
Editace šablony
<<extend>>
Volba obsahu pro jednotlivé rámce
<<extend>>
Definice posloupnosti zobrazovaných obsahů
Šablona je definována svým schématem a obsahem jednotlivých rámců
Smazání šablony
Při editaci není umožněno měnit schéma. Pokud chce uživatel změnit schéma šablony, musí vytvořit šablonu novou.
<<include>>
Nastavení výchozí/globální šablony u panelů s mazanou šablonou

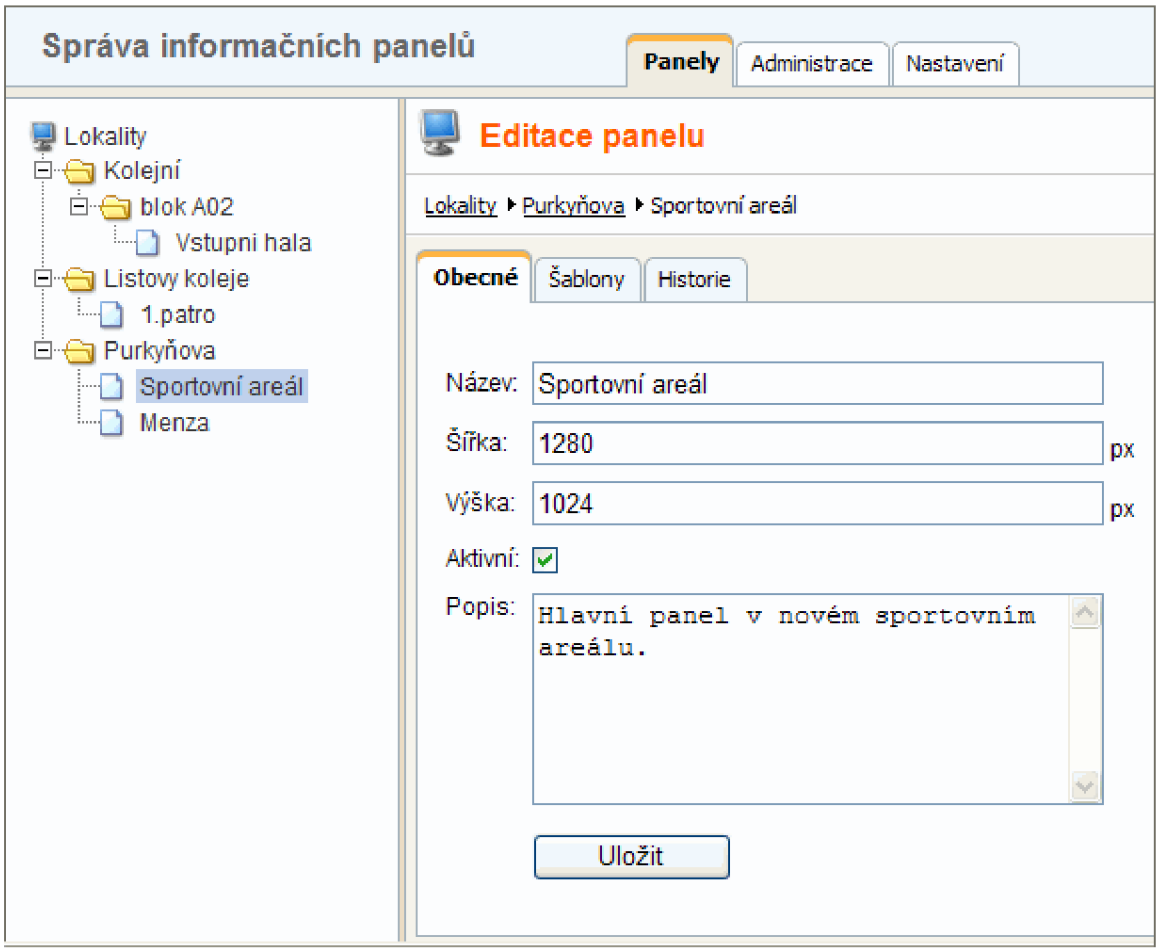

<<extend>>
Správa systémových oprávnění
Správa pracovníků
Správce systému
Pracovník
<<include>>
Definice rozměrů a polohy jednotlivých rámců
Definice nového schématu
<<include>>
Editace schématu
Správa schémat panelů
Uživatel systému
Schéma definuje rozdělení plochy panelu na dílčí rámce
Smazání šablon s tímto schématem
<<include>>
Smazání schématu
<<extend>>
Nahrání datového souboru obsahu na serverový disk
Definice nového obsahu
Správa obsahu
Smazání obsahu
<<include>>
Nastavení obsahu a jeho propojení na šablony se uchová. Bez ohledu na platnost šablon se obsah přestane zobrazovat na všech šablonách.
Možnost náhledu na definovaný obsah
Zakázání obsahu
Smazání obsahu ze všech šablon a jejich rámců

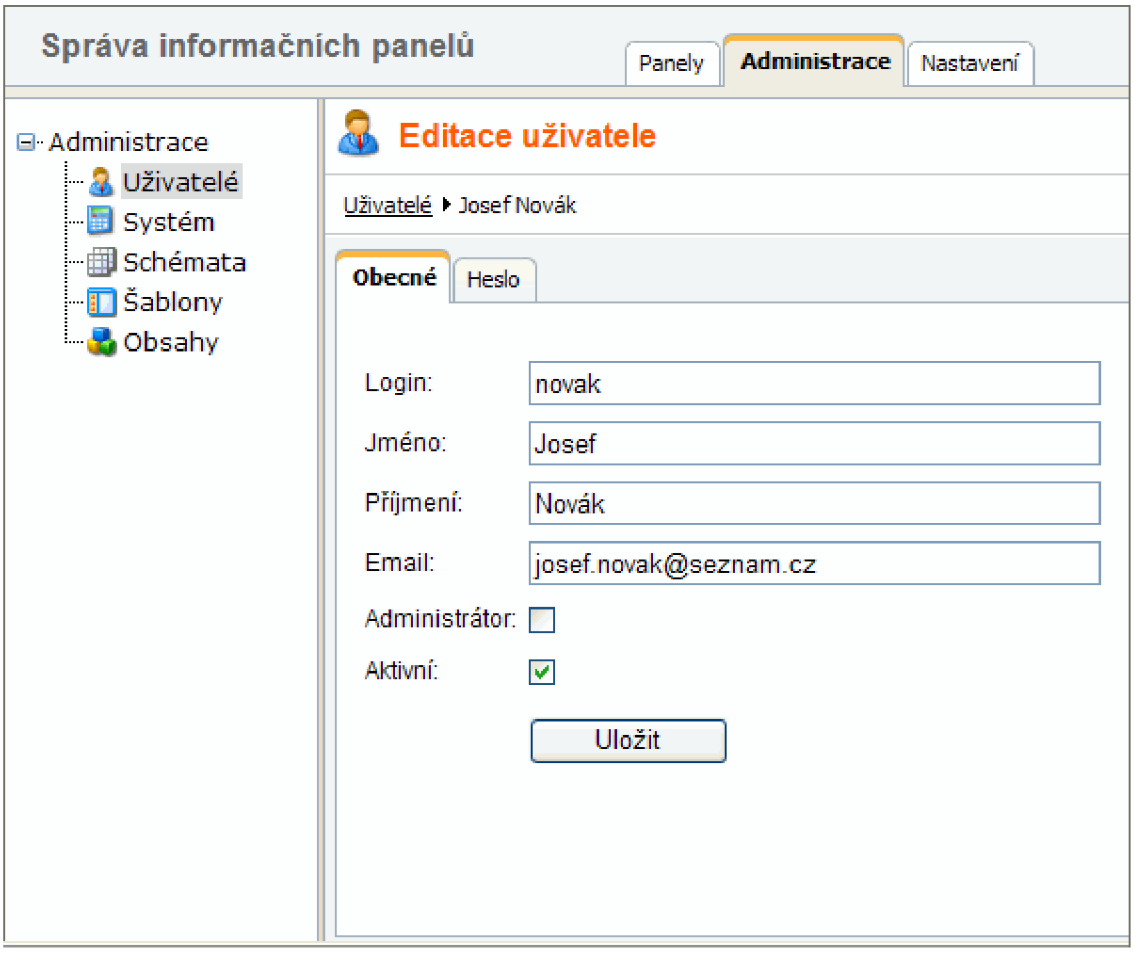
Příloha 2


Layout
Column Name
Data Type
Allow N...
LayoutID
int
LayoutName
nvarchar(200)
LayoutDescription
ntext
Frame
Column Name
Data Type
Allow N...
FrameID
int
FrameName
nvarchar(50)
FrameOrder
int
FrameLayoutType
nvarchar(1)
FrameSize
float
FrameParentID
int
FrameLayoutID
int
FrameChildrenCount
int
FrameIDPath
nvarchar(450)
Template
Column Name
Data Type
Allow N...
TemplateID
int
TemplateName
nvarchar(100)
TemplateDescription
ntext
TemplateLayoutID
int
TemplateIsDefault
bit
Content
Column Name
Data Type
Allow Nulls
ContentID
int
ContentName
nvarchar(100)
ContentFileName
nvarchar(100)
ContentExtension
nvarchar(50)
ContentUrl
ntext
ContentMimeType
nvarchar(50)
ContentDuration
int
ContentWidth
int
ContentHeight
int
ContentSize
int
ContentIsEnabled
bit
ContentDescription
ntext
ContentGUID
uniqueidentifier
TemplateFrameContent
Column Name
Data T...
Allow N...
ID
int
TemplateID
int
FrameID
int
ContentID
int
ContentOrder
int
ContentActual...
int
ContentActual...
int
ContentActual...
int

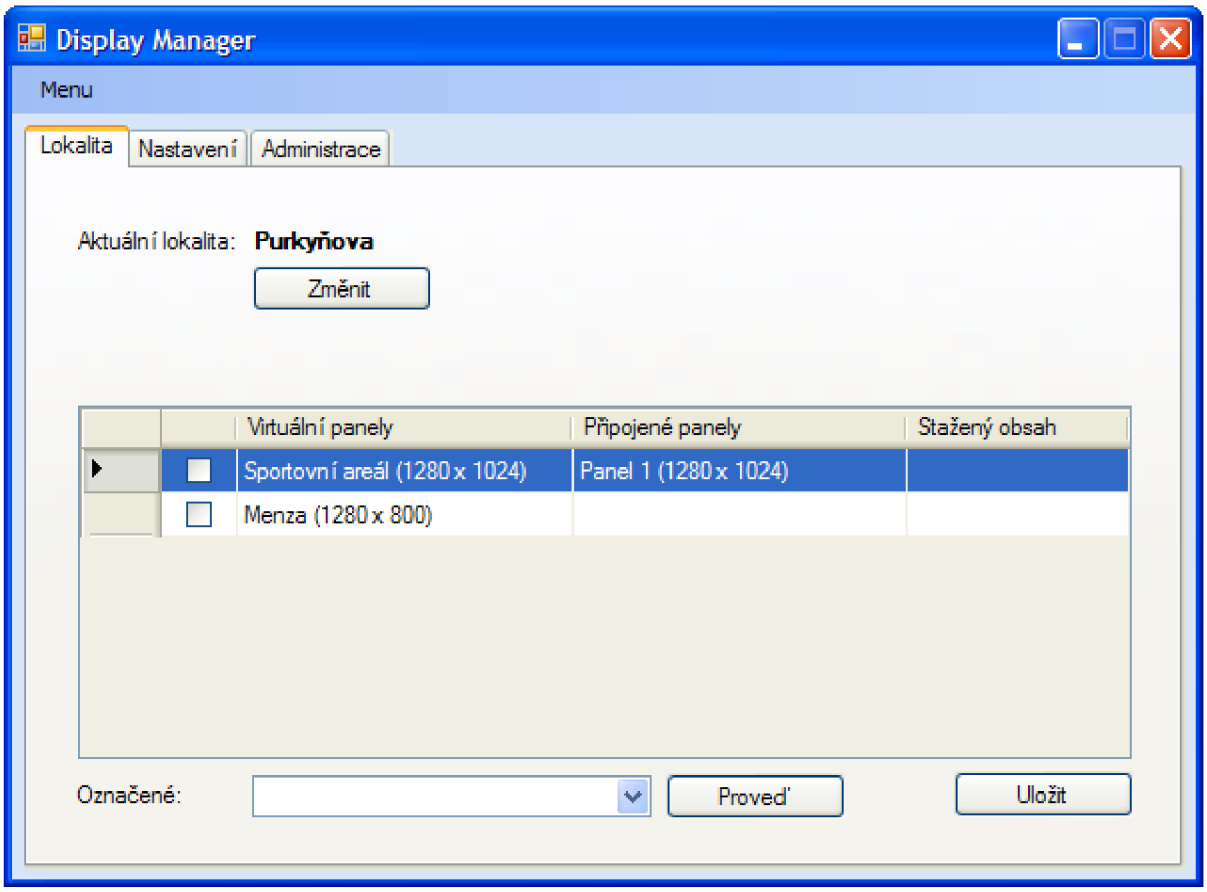

DisplayTemplate
Column Name
Data Type
Allow Nulls
DisplayTemplateID
int
DisplayID
int
TemplateID
int
ValidFrom
datetime
ValidTo
datetime
Template
Column Name
Data Type
Allow N...
TemplateID
int
TemplateName
nvarchar(100)
TemplateDescription
ntext
TemplateLayoutID
int
TemplateIsDefault
bit
Display
Column Name
Data Type
Allow N...
DisplayID
int
DisplayName
nvarchar(100)
DisplayWidth
int
DisplayHeight
int
DisplayLocationID
int
DisplayDescription
ntext
DisplayOrder
int
DisplayIsEnabled
bit
Users
Column Name
Data Type
Allow Nulls
UserID
int
UserLogin
nvarchar(100)
UserPassword
nvarchar(100)
UserFirstName
nvarchar(100)
UserLastName
nvarchar(100)
UserEmail
nvarchar(100)
UserIsEnabled
bit
UserIsGlobalAdminis...
bit
Location
Column Name
Data Type
Allow N...
LocationID
int
LocationName
nvarchar(100)
LocationParentID
int
LocationDescription
ntext
LocationIDPath
nvarchar(450)

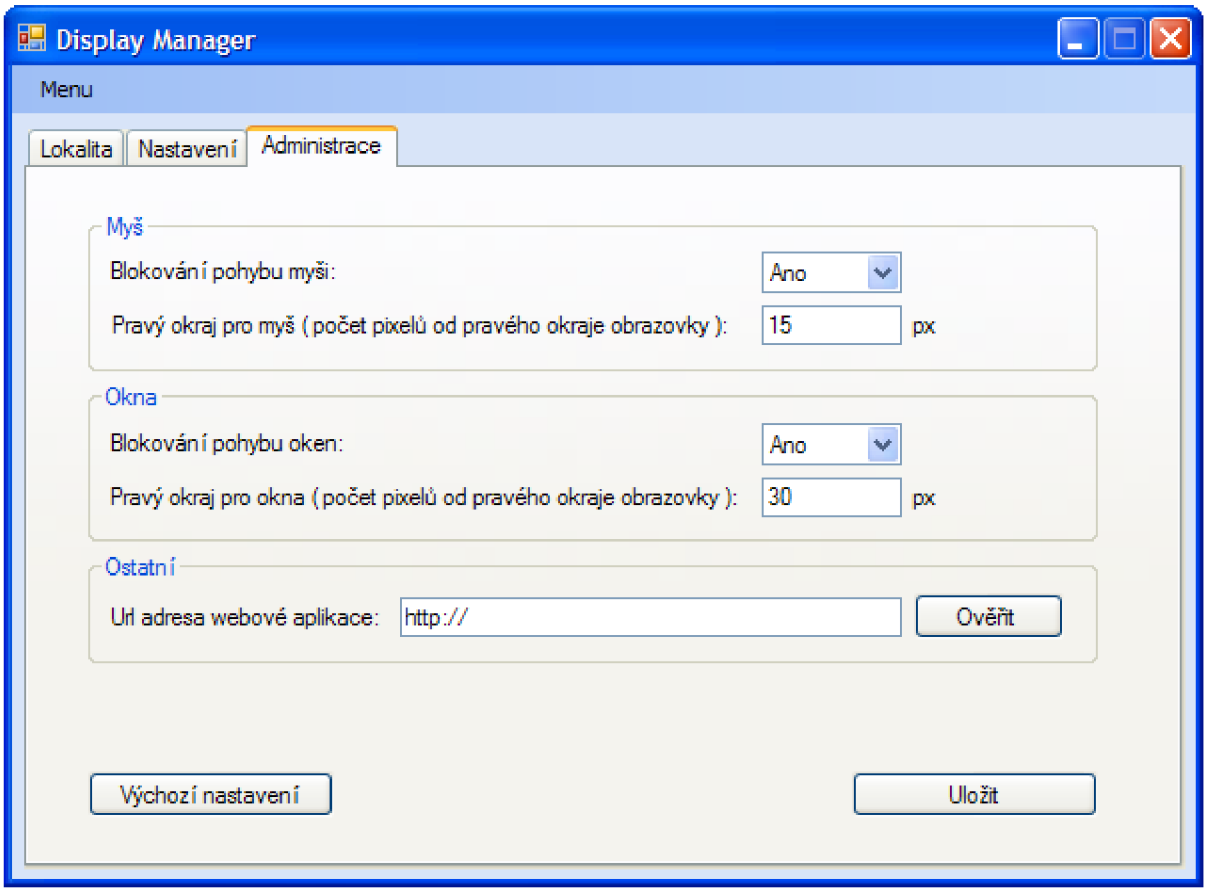
Příloha 3


Správa informačních panelů
Panely
Administrace
Nastavení
Lokality
Kolejní
blok A02
Vstupni hala
Listovy koleje
1.patro
Purkyňova
Sportovní areál
Menza
Editace panelu
Lokality ▸ Purkyňova ▸ Sportovní areál
Obecné
Šablony
Historie
Název:
Sportovní areál
Šířka:
1280
px
Výška:
1024
px
Aktivní:
Popis:
Hlavní panel v novém sportovním areálu.
Uložit


Správa informačních panelů
Panely
Administrace
Nastavení
Administrace
Uživatelé
Systém
Schémata
Šablony
Obsahy
Editace uživatele
Uživatelé ▸ Josef Novák
Obecné
Heslo
Login:
novak
Jméno:
Josef
Příjmení:
Novák
Email:
josef.novak@seznam.cz
Administrátor:
Aktivní:
Uložit

Příloha 4


Display Manager
Menu
Lokalita
Nastavení
Administrace
Aktuální lokalita:
Purkyňova
Změnit
Virtuální panely
Připojené panely
Stažený obsah
Sportovní areál (1280 x 1024)
Panel 1 (1280 x 1024)
Menza (1280 x 800)
Označené:
Proveď
Uložit


Display Manager
Menu
Lokalita
Nastavení
Administrace
Myš
Blokování pohybu myši:
Ano
Pravý okraj pro myš (počet pixelů od pravého okraje obrazovky):
15
px
Okna
Blokování pohybu oken:
Ano
Pravý okraj pro okna (počet pixelů od pravého okraje obrazovky):
30
px
Ostatní
Url adresa webové aplikace:
http://
Ověřit
Výchozí nastavení
Uložit