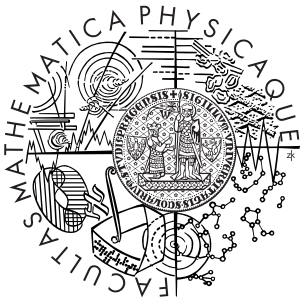
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Široký

Aplikace pro prezentaci dat naměřených v systému SCADA

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek
Studijní program: Informatika, programování

2010

Na tomto místě bych rád poděkoval společnosti Proteco Expert s.r.o. za podporu a svolení zpracovávat jeden z jejích projektů v rámci této bakalářské práce.

Dále děkuji mému vedoucímu, Mgr. Pavlovi Ježkovi, za konzultace a cenné rady a připomínky, které mi v průběhu celé práce a při úpravě textu poskytoval.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Jan Široký

Obsah

Název práce: Aplikace pro prezentaci dat naměřených v systému SCADA
Autor: Jan Široký
Katedra (ústav): Katedra distribuovaných a spolehlivých systémů
Vedoucí bakalářské práce: Mgr. Pavel Ježek
e-mail vedoucího: pavel.jezek@mff.cuni.cz

Abstrakt: Cílem práce bylo vytvořit webovou aplikaci, která slouží k zobrazování a prezentaci dat naměřených v systému SCADA. Skládá se ze dvou částí - klientské a serverové. Serverová část slouží k načtení a zpracovávání dat z archivní databáze, kde jsou uloženy naměřené hodnoty a následně k jejich odeslání klientovi. Klientská část tyto data prezentuje uživateli ve formě tabulek, grafů nebo grafických schémat. Pro grafická schémata bylo nutné navrhnout jejich formát a reprezentaci a implementovat pro ně jednoduchý editor. V něm je možné sestavit nové schéma ze základních geometrických tvarů, bitmapových obrázků, textových popisků, apod. a propojit vybrané vlastnosti těchto objektů s datovými zdroji, takže se grafické schéma dynamicky mění v závislosti na datech.

Klíčová slova: SCADA, data, prezentace, server, klient

Title: Application for Presentation of SCADA System Measurements
Author: Jan Široký
Department: Department of Distributed and Dependable Systems
Supervisor: Mgr. Pavel Ježek
Supervisor's e-mail address: pavel.jezek@mff.cuni.cz

Abstract: The goal of this thesis was to create a web application which will show and present the data measured in the SCADA system. It consists of two halves - the server and the client. The server fetches the data from the archive database, processes it and sends it to the client. The client side presents these data in tables, graphs or graphical schemes. Next part of the thesis was to design a format and a representation of these schemes and implement a simple editor for them. The editor provides functionality to create a new scheme from basic geometric shapes, bitmap images, text labels, etc. Some properties of these objects may be linked to the data sources, so the scheme, in dependence on the data, dynamically changes.

Keywords: SCADA, data, presentation, server, client

Kapitola 1

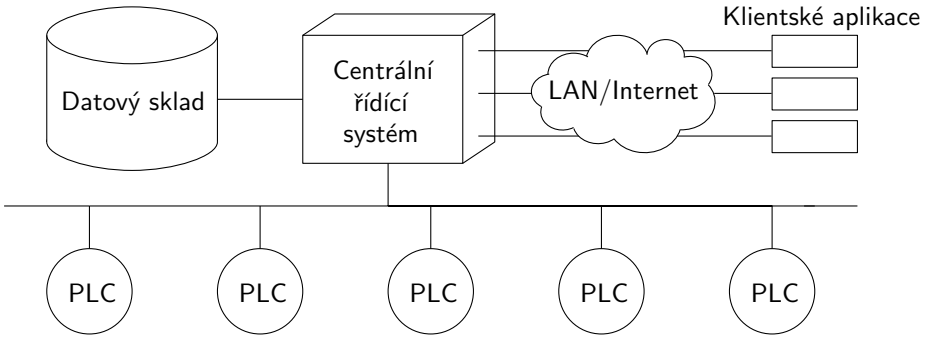
Úvod

V rámci této bakalářské práce bude vytvářena aplikace, která bude velice úzce spolupacovat s již existujícím SCADA systémem StoneBase společnosti Proteco Expert s. r. o. Tento systém je ve vývoji již delší dobu a je nasazen v mnoha průmyslových provozech, takže koncepce a architektura tohoto systému nejsou předmětem této práce a ani není možné je měnit.

1.1 Přehled SCADA systému StoneBase

1.1.1 SCADA systém

SCADA systém[1] (Supervisory Control And Data Acquisition system) je zpravidla průmyslový řídící systém, který slouží k monitorování a řízení určitého průmyslového provozu. Skládá z několika komponent:



Obrázek 1.1: Struktura SCADA systému.

Centrální řídící systém je komponenta SCADA systému, která zajišťuje především sběr měřených dat, jejich vyhodnocení, zpracování a ukládání do datového skladu. Provádí rovněž distribuci povelů na podřízené technologie, generování alarmů a stará se o průběžné zálohování archivu.

Jednotky PLC (Programmable Logic Controller) jsou univerzální programovatelná zařízení přímo v místě provozu, která bezprostředně sbírají data ze senzorů a odesílají je řídícímu systému. Jsou také schopné reagovat na naměřená data a vydávat podle toho patřičné příkazy provozu.

Komunikační infrastruktura zajišťuje přenos dat z PLC jednotek do řídícího systému.

Klientské aplikace slouží zejména dispečerům a operátorům pro nepřetržité sledování provozu, vyhodnocování alarmových situací a vydávání požadovaných povelů. Jiným typem jsou konfigurační aplikace, které slouží k nastavování parametrů systému.

1.1.2 StoneBase

SCADA systém StoneBase[2] je distribuovaná aplikace učená pro prostředí Microsoft Windows 2000 a vyšší a je složena z celé řady serverových a klientských komponent, které vzájemně sdílejí data uložené v centrální databázi. Tato databáze obsahuje nejen veškerá naměřená a zpracovaná data (okamžité i archivní hodnoty, alarmová hlášení), ale také celou parametrizaci a všechny informace potřebné pro prezentaci dat. Systém StoneBase je schopen pracovat nad různými databázovými servery, v současné době jsou podporovány Microsoft SQL Server a MySQL.

Struktura databáze

Protože v rámci této práce bude vytvářena jedna z klientských aplikací, nebude nás podrobněji zajímat, jak fungují PLC jednotky, kumunikační infrastruktura nebo zpracovávání dat v centrálním řídícím systému. Z vlastností systému pro nás bude nejdůležitější centrální datový sklad, neboť to bude jediná komponenta systému, se kterou bude naše aplikace komunikovat. Zde jsou uvedeny jen ty nejdůležitější tabulky, které tvoří jakousi základní kostru systému. Databáze samozřejmě obsahuje daleko více tabulek a zde

uvedené mají řadu dalších sloupců. Ty však není nezbytně nutné znát, abychom získali základní povědomí o systému. Mezi nejdůležitšjší tabulky, jak je vidět na obrázku 1.2, patří:

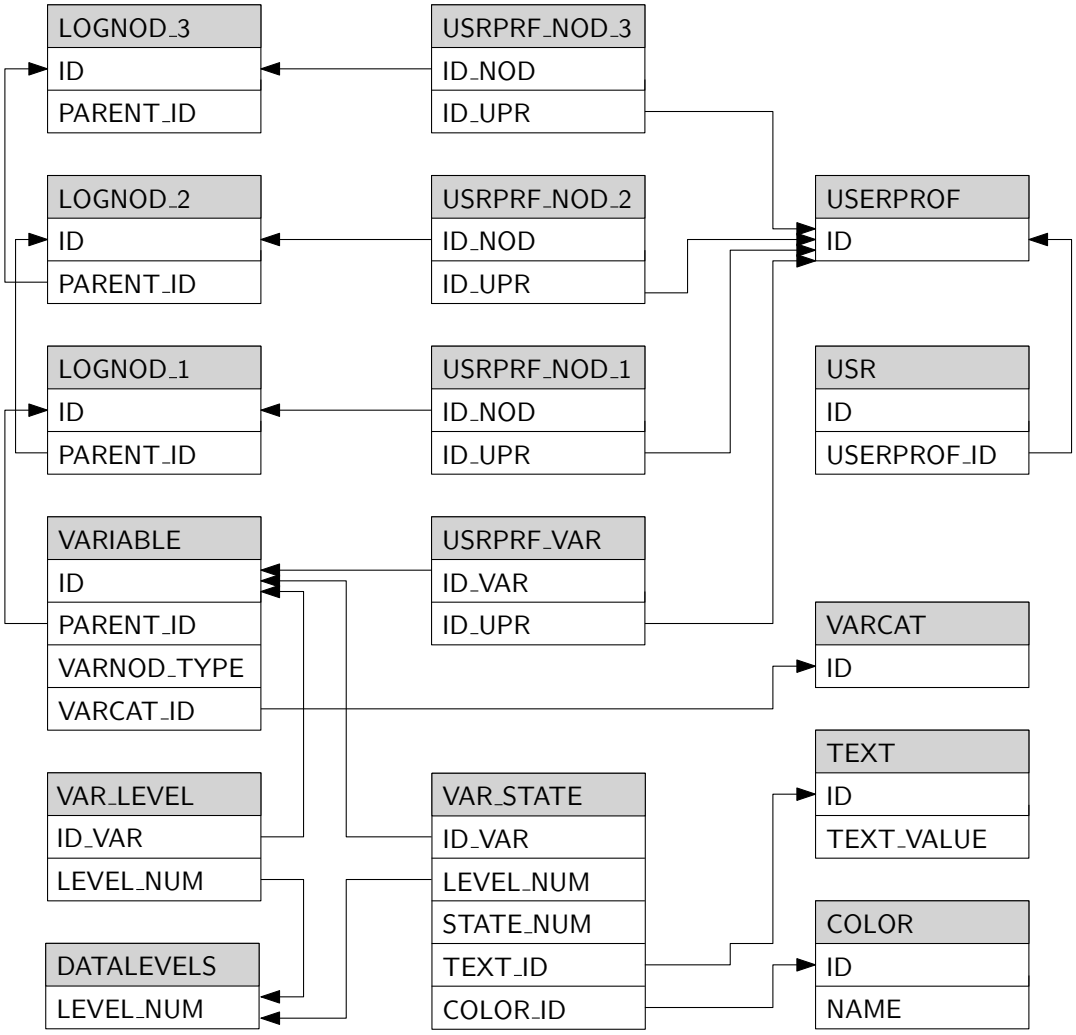
`LOGNOD_X` kde `X` může být 1, 2, 3 atd. Systém má stromovou strukturu a tyto tabulky reprezentují jednotlivé úrovně tohoto stromu. Typicky úroveň 3 je nejvyšší, tedy hned pod kořenem, a odpovídá oblastem v systému, úroveň 2 je rozdělení oblastí na lokality, 1 je zjemnění lokalit na místa. Každý uzel má nadefinovaného rodiče z vyšší úrovně, jméno, identifikátor apod.

`VARIABLE` obsahuje informace o proměnných v systému. PLC jednotky obvykle v logické struktuře odpovídají místům, různé senzory napojené na tuto jednotku pak odpovídají proměnným. Může jít o teplotu v peci či aktuální směr větru. Proměnné mají podobné vlastnosti jako uzly logické struktury, obsahují navíc odkaz na kategorii, do které proměnná patří, typ proměnné a další. Nejdůležitější typy proměnných jsou „analog“, což je třeba výše zmíněná teplota a „binár“, což je proměnná, která může nabývat konečně mnoha hodnot (název binár implikuje dva stavy, ale může jich být 2-16). Příklad binára je proměnná „dveře“, jejíž hodnota může být buď „otevřeno“ nebo „zavřeno“.

`DATALEVELS` neboli datové úrovně vyjadřují časové hladiny, na kterých jsou data měřeny. Obvykle jsou v systému datové úrovně „sekunda“, „minuta“, „hodina“, „den“, „měsíc“ a „rok“. Znamená to, že data jsou fyzicky měřeny každou sekundu. Každou minutu se vezmou sekundová data naměřená za poslední minutu a podle konfigurace se třeba zprůměrují a uloží. Každou hodinu se vezmou minutová data naměřená za poslední hodinu a zpracují do hodinové hodnoty. A tak až do nejvyšší úrovně.

`VAR_LEVEL` odpovídá vlastnostem konkrétní proměnné pro danou datovou úroveň. Definuje se zde například jednotka, protože u proměnné „hmotnost spáleného uhlí“ má na úrovni „sekunda“ smysl zobrazovat hodnotu v kilogramech, zatímco na úrovni „den“ už v tunách. Také jsou zde definovány meze, tedy hodnoty, při jejichž překročení je vyvolán alarm. Je zase logické, že tyto meze musí být pro různé datové úrovně různé.

`VAR_STATE` se uplatňuje hlavně u binárních proměnných. Jeden řádek v tabulce odpovídá jednomu stavu dané proměnné na dané datové úrovni.



Obrázek 1.2: Kostra databáze systému StoneBase.

Pro proměnnou „dveře“ a datovou úroveň „sekunda“ se nachází dva záznamy („otevřeno“, „zavřeno“). Na úrovni „hodina“ už ale „dveře“ můžou mít čtyři stavy („stále otevřeno“, „povětšinou otevřeno“, „povětšinou zavřeno“ a „stále zavřeno“). Stav má také další parametry jako barvu a text, kterými má být prezentován.

`USR` obsahuje informace o uživatelích systému. Tím jsou myšleni operátoři a lidé, kteří pracují s klientskými aplikacemi. Každý uživatel má jméno, heslo, práva (například právo měnit parametry systému) a odkaz na svůj uživatelský profil.

`USERPROF`, `USRPRF_NOD_X`, `USRPRF_VAR_X` obsahuje uživatelské profily, které definují vlastnosti shodné pro skupiny uživatelů (administrátoři, dispečeři). V dalších tabulkách jsou pak práva profilů k jednotlivým uzlům logické struktury a k proměnným.

Datové tabulky jsou tří typů a to pro archivní data, poslední data (hodnoty, které byly změřeny naposledy) a průběžné data. Průběžné data mají smysl jen u vyšších datových úrovní, např. v půlce měsíce se sice nedá určit přesná hodnota za „rozpracovaný“ měsíc, ale z dosud naměřených hodnot je možné extrapolovat hodnotu průběžnou. Každý záznam v datové tabulce obahuje odkaz na proměnnou, odkud hodnota pochází, datum a čas, kdy byla změřena, samotnou hodnotu a další dodatečné informace, které byly na základě hodnoty vygenerovány řídícím systémem.

`_SB_SYSTEMS` uchovává informace o systémech v databázi (jméno systému, unikátní kód systému). Takže je možné do jedné databáze ukládat data a konfiguraci více systémů, přičemž všechny výše zmíněné tabulky mají před jménem, jako prefix, unikátní kód systému, do kterého patří. Zatím se ale nikdy této možnosti nevyužilo, vždy jedna databáze obsahuje jen jeden systém.

Klientská aplikace SBExplorer

SBExplorer[3] je nejdůležitější klientská aplikace systému, která umožňuje nejen prezentaci naměřených dat, ale také vydávání povelů a ovládání samotného průmyslového provozu. Nabízí velmi bohaté prezentační schopnosti, data je možno zobrazovat ve formě grafů, tabulek, grafických schémat apod. SBExplorer komunikuje přímo s datovým serverem, což v rámci LAN nevadí.

Datový server ale neumožňuje přímý přístup k databázi z internetu, takže v případě, že uživatel potřebuje přistupovat k datům vzdáleně, musí si doinstalovat ještě podpůrnou aplikaci, která vytváří SQL tunel mezi SBExplorerem a aplikačním serverem. Tato podpůrná aplikace na klientovi všechny SQL dotazy vkládá do HTTP požadavků, které odesílá na aplikační server. Zde jsou SQL dotazy odeslány na datový server, jejich výsledky jsou opět zabaleny do HTTP odpovědí a vráceny klientovi. Podpůrná aplikace na klientovi odpověďi rozbalí a předá SBExploreru, který vlastně vůbec neví, že komunikace probíhala po internetu.

1.2 Cíle práce

Vzhledem k nevýhodám, které se projevují při použití aplikace SBExplorer po internetu, vznikl na straně zadavatele požadavek vytvořit webovou aplikaci, která bude sloužit k zobrazování a prezentaci naměřených dat. Serverová část bude sloužit k získávání a zpracovávání dat z datového skladu, kde jsou uloženy naměřené hodnoty a následně k jejich odeslání klientovi. Klientská část aplikace bude tato data prezentovat v různých formách uživateli. Nebude již tedy nutné pro každý SQL dotaz vytvářet nový HTTP požadavek, sníží se objem přenášených dat a vytvářená aplikace bude lépe reagovat na delší latenci způsobenou komunikací po internetu. Cílem práce je tedy vytvořit aplikaci, která se v prezentačních schopnostech bude blížit SBExploreru. Nebude sice, co se funkcionality týče, tak robustní, ale měla by rychlostí a snadností použití v internetovém prostředí předčit SBExplorer spolu s podpůrnou aplikací. Následuje popis požadavků zadavatele:

1.2.1 Požadavky na serverovou část aplikace

Protože SCADA systém StoneBase je postaven na platformě Microsoft Windows, měla by být vytvářená aplikace přizpůsobena pro operační systém Microsoft Windows a to ve verzi XP a novější. Nepočítá se s přechodem na jiný operační systém, takže není třeba, aby byla multiplatformní. Jako datový sklad je používán Microsoft SQL Server nebo MySQL, přičemž oba databázové servery by měly být podporovány. Serverová část by také měla být vytvářena s přihlédnutím na to, že v budoucnu se databázový server může změnit například na Oracle, takže by bylo dobré, aby v případě takovéto změny nebyly nutné zásadnější úpravy.

K serveru se klient bude moci připojovat jak v rámci LAN, tak i v rámci internetu, kde bude nutné komunikaci šifrovat (například využitím protokolu HTTPs). Způsob zabezpečení také záleží na společnostech, které systém používají, a na jejich bezpečnostní politice. Zabezpečení by tedy mělo být uděláno variabilně, aby se dalo v závislosti na požadavcích zákazníka změnit.

1.2.2 Požadavky na klientskou aplikaci

Hlavní okno klientské aplikace by se mělo skládat z několika částí. Nahoře hned pod titulkem aplikace bude panel nástrojů, ve kterém půjde nastavit parametry dat, které chce uživatel zobrazit a to bez ohledu na jejich formu. Tzn. datum a čas, kdy byly data naměřeny, jedná-li se o data archivní, poslední čí průběžné apod. Bude také obsahovat další akce globální pro celý program jako třeba tlačítko „zpět“ nebo tlačítko „obnovit“. Naopak úplně dole se bude nacházet stavový řádek, který bude informovat uživatele o stavu aplikace, probíhajících akcích a případných chybách.

V menší části okna nalevo se bude zobrazovat logická struktura systému, pro kterou bude použito stromové zobrazení, kde kořen je celý systém, pod ním jsou pak oblasti, lokality, místa a v listech proměnné. Strom je nutné načítat dynamicky, tedy jen ty uzly, které jsou momentálně dostupné. Při otevření uzlu se pak donačtou jeho potomci, při zavření se potomci naopak odstraní. Logická struktura systému se totiž v průběhu práce s aplikací může změnit a toto je nutné reflektovat. Existují také systémy, které mají velmi mnoho proměnných a uzlů a není nutné držet v paměti celý strom po celou dobu běhu aplikace. Ve zbývající části okna se budou nacházet samotné prezentační „stránky“, které bude možno přepínat pomocí záložek. Bude jich pět typů a to:

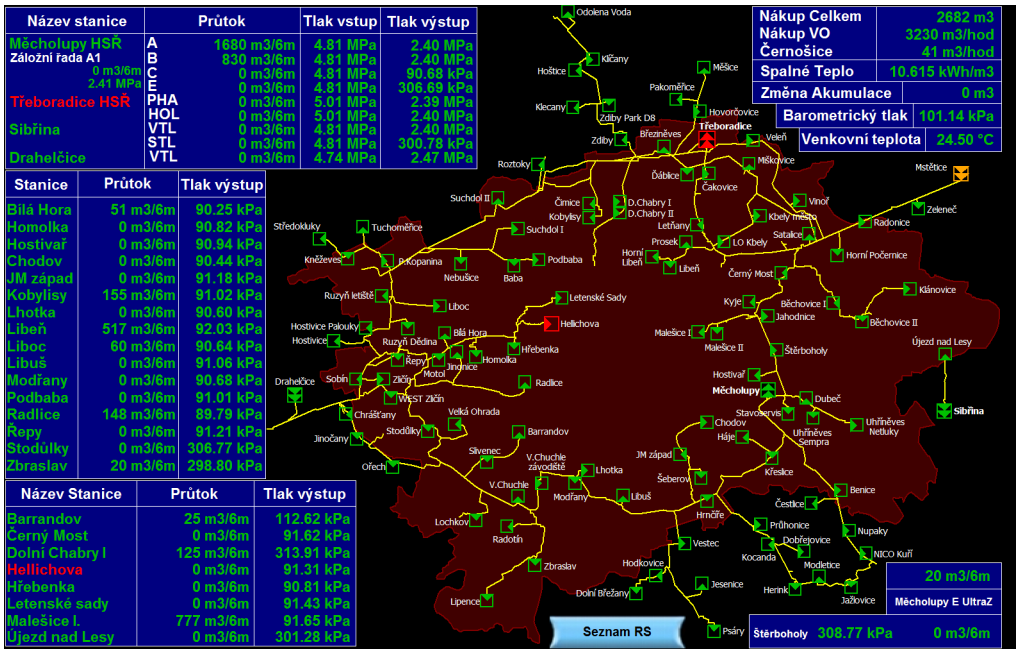
Stavy jsou variabilní přehled naměřených vzorků v řádkovém seznamu, který by měl umožňovat stránkování, třídění a filtraci podle různých vlastností. Měla by být také možnost zobrazit sumarizované stavy vyšších logických úrovní systému.

Grafy (čarové, sloupcové, větrné růžice), do kterých jsou vyneseny časové trendy naměřených vzorků. Tato prezentační „stránka“ by měla umět stránkování, různý počet grafů v jednom okně, filtraci grafů. Grafy by měly mít technický vzhled a respektovat barevnou konvenci odpovídající stavu vzorku, zobrazovat meze a podobně.

Tabulka časových trendů s obdobnými parametry jako u grafů. Navíc doplněno o statistiku (minimum, maximum, průměr, apod.) pro každý trend. Tabulka by měla být synchronizována s grafy, takže po přepnutí na grafy se nezmění naměřené hodnoty, jen způsob zobrazení.

Alarmy jsou podobná prezentační „stránka“ jako stavy s tím rozdílem, že se zobrazují vzorky, které vyvolaly alarmovou událost, doplněné o další informace související s alarmem.

Schéma. V této prezentační „stránce“ budou zobrazována grafická schémata, která jsou podrobněji popsána v následující části.



Obrázek 1.3: Ukázka grafického schématu.

1.2.3 Požadavky na grafická schémata

Grafická schémata jsou uživatelem vytvořené vektorové obrázky skládající se ze základních geometrických tvarů, textových popisků, případně i bitmapových obrázků. Na takovéto statické vektorové obrázky jsou napojeny skripty

v jazyku Visual Basic, které v závislosti na zobrazovaných datech upravují vlastnosti prvků ve schématu. Například plní textová pole naměřenými hodnotami nebo obarvují některé prvky v závislosti na naměřených datech.

Stávající schémata jsou ve formátu, ke kterému nemá zadavatel dokumentaci ani popis. K jejich vytváření a zobrazování jsou používány zakoupené komponenty, jejichž zdrojový kód není k dispozici, takže není možné schéma rekonstruovat z binární reprezentace uložené v databázi. Je tedy nutné navrhnout novou koncepci a reprezentaci grafických schémat, ideálně tak, aby obě verze mohly být používány současně. Aplikace SBExplorer bude používat stávající schémata, zatímco vytvářená aplikace bude zobrazovat již schémata nová.

Schémata by měla být složena z vrstev, které budou obsahovat různé grafické objekty: čáry, obdélníky, elipsy, kruhy, textové popisky a vložené bitmapové obrázky (obrázky se nacházejí v databázi). Většina vektorových objektů by měla mít klasické vlastnosti jako pozice, výška, šířka, barva okraje a výplně, u textových popisků text a vlastnosti fontu a podobně.

1.2.4 Požadavky na editor schémat

Pokud uživatel v klientské aplikaci přejde do prezentační „stránky“ se schématy, mělo by být možno zobrazené schéma editovat. To znamená, že se spustí editor, ve kterém bude odpovídající schéma připravené k úpravám. Editor schémat by měl umožnit výše zmíněné grafické objekty vytvářet, upravovat a organizovat do vrstev. Bitmapové obrázky nebude možné vytvářet, pouze vkládat ve formě odkazu do konfigurační databáze. Další vlastnosti a funkce editoru nebyly specifikovány, protože se odvíjí od navrhnutého konceptu schémat.

Kapitola 2

Analýza

2.1 Výběr technologie

Hned na začátku bylo rozhodnuto, že vytvářená aplikace bude postavena nad nejnovějším .NET frameworkem (verze 4.0) a to hlavně z důvodu nejlepší integrace tohoto frameworku do operačního systému Windows a velmi kvalitní podpory této aplikační platformy ze strany jak Microsoftu, tak i programátorské komunity. Framework .NET také nabízí velmi bohaté grafické možnosti aplikací napsaných ve WPF, které již předčí klasické Windows Forms aplikace.

Tato platforma nám nabízí několik možných technologií, které splňují výše zmíněné požadavky.

2.1.1 ASP.NET aplikace

ASP.NET je technologie používaná k tvorbě dynamických webových stránek a s použitím AJAXu by mohla vcelku solidně emulovat chování klasických desktopových aplikací. AJAX by dovolil přenést část čistě prezentační logiky na klienta, což může, ale i nemusí být výhoda, vzhledem k tomu, že neznáme výkon klientského počítače a generování složitějších částí HTML stránek JavaScriptem není výkonově optimální. Mezi největší negativa této technologie patří zejména náročnost vývoje uživatelského rozhraní, s čímž souvisí i nedostatek kvalitních nekomerčních komponent pro tvorbu UI.

Výhody: nulové nároky na klienta, snadná distribuce nových verzí.

Nevýhody: náročný vývoj, uživatelské rozhraní ochuzeno o obecně známé prvky (TreeView, ListView).

2.1.2 Silverlight aplikace

Silverlight je zatím nepříliš rozšířená obdoba technologie Adobe Flash, která umožňuje tvorbu aplikací nad ořezanou verzí .NET frameworku. Silverlight aplikace jsou spouštěny v rámci webového prohlížeče, který si aplikaci sám stáhne ze serveru. Uživatel kromě pluginu do prohlížeče nemusí nic instalovat. Silverlight bývá využíván hlavně pro aplikace, které cílí na větší počet uživatelů, protože není náročný na instalaci. Serverová část by fungovala jako webová služba, kterou by klientská aplikace volala.

Výhody: velmi malé nároky na klienta (instalace pluginu), snadná distribuce nových verzí.

Nevýhody: chudší grafické možnosti oproti WPF, klientská aplikace běží v režimu „partial-trust“, což například omezuje práci se souborovým systémem.

2.1.3 XBAP

XBAP[4] (XAML browser application) kombinuje možnosti klasických desktopových aplikací a webových aplikací. Uživatel k aplikaci přistupuje skrze webový prohlížeč, který aplikaci stáhne ze serveru, podobně jako u Silverlight aplikace. Rozdíl je v tom, že na klientském počítači musí být nainstalován celý .NET framework, což přináší mnoho výhod (plné využití WPF a knihoven .NET frameworku) a aplikace může běžet nejen v „partial-trust“ režimu, ale i ve „full-trust“ režimu. Serverová část by byla webová služba jako v případě Silverlightu.

Výhody: poměrně malé nároky na klienta (instalace .NET frameworku), snadná distribuce nových verzí, neomezené využití knihoven.

Nevýhody: delší doba spouštění, způsobená stahováním ze serveru.

2.1.4 WPF desktopová aplikace

V případě volby této technologie by klient a server byly dvě úplně oddělené aplikace. Uživatel by si nainstaloval klientskou část, což by byla normální

desktopová aplikace a ta by komunikovala se serverovou částí - webovou službou. Hlavní výhodou oproti Silverlightu a XBAP je nezávislost na webovém prohlížeči, plná možnost využití knihoven .NET frameworku a neomezenost bezpečnostními režimy.

Výhody: neomezené využití knihoven, přívětivost uživatelského rozhraní, rychlý start.

Nevýhody: největší nároky na klienta (instalace frameworku a aplikace), složitější distribuce nových verzí.

2.1.5 Zvolená technologie

Nejdříve byly zamítnuty technologie ASP.NET a Silverlight, protože jejich cílová skupina jsou projekty pro velké masy uživatelů, ne jednotky až pár desítek uživatelů. To velmi omezuje možnosti uživatelského rozhraní, které zdaleka nemůže být tak bohaté, jako u zbývajících dvou technologií. Rozhodování mezi XBAP a WPF aplikací probíhalo déle, rozdíly mezi nimi nejsou tak markantní. Zadavateli se zamlouvala technologie XBAP z toho důvodu, že není potřeba distribuovat nové verze aplikace, uživateli se při spuštění vždy stáhne aktuální. WPF aplikace je na druhou stranu nejvíce podobná technologii SBExploreru, takže by pro uživatele byla nejpřívětivější. Nakonec byla zvolena desktopová WPF aplikace, s tím, že v případě poptávky by neměl být problém upravit aplikaci tak, aby fungovala i jako XBAP aplikace.

2.1.6 Komunikace klienta se serverem

Framework .NET obsahuje mnoho technologií určených pro vzájemnou komunikaci distribuovaných aplikací. Jedná se například o ASP.NET webové služby (ASMX), .NET Remoting, použití objektů z jmenného prostoru System.Net a další. Od verze 3.0 však .NET framework obsahuje WCF (Windows Communication Foundation), což je nejnovější technologie určená pro jednoduchou tvorbu servisně orientovaných aplikací. WCF sjednocuje a zastřešuje všechny původní technologie a poskytuje široké možnosti v oblasti zabezpečení a volby přenosového protokolu. WCF bylo vybráno z důvodu snadné implementace webové služby i klienta, kromě vzdáleného volání metod umožňuje například i odchytávání výjimek vyhozených na serveru až na klientovi. Konfigurace webové služby je také snadná. Veškeré informace

o zabezpečení, komunikačním protokolu apod. jsou uloženy v konfiguračním XML souboru, takže případné změny zabezpečení či zapnutí autentifikace pomocí serverového certifikátu je docíleno změnou parametrů v tomto souboru. Více informací o technologii WCF je k nalezení na [5], konkrétně k oblasti zabezpečení pak na [6].

2.2 Řešení přístupu k databázi

Jedním z požadavků na aplikaci je i schopnost pracovat s různými databázovými servery a to nejlépe tak, že typ databáze se zvolí v konfiguračním souboru na serverové straně. Možností, jak toto řešit je mnoho. Od dvou sad metod, kde každá z nich bude vykonávat dotazy nad svým typem databáze, až po komplexní databázové abstrakční vrstvy. Z důvodu velikosti aplikace a počtu různých SQL dotazů bylo rozhodnuto, že nejlepší bude použít nějakou abstrakci databáze. Ta zabrání psaní dvou, ve většině případů dosti podobných, SQL dotazů a v případě, že by zadavatel chtěl rozšířit portfolio podporovaných databázových serverů, nebude nutné psát pro každý dotaz další verzi. Byla zvažována tato řešení:

Linq to SQL a Linq to MySQL. Zde je hlavní problém v tom, že vývoj provideru pro MySQL databázi se zastavil někdy kolem roku 2007 a od té doby nebyl aktualizován. Existují samozřejmě komerční a jistě i kvalitní providery, ale ty nebylo možno z finančních důvodů použít.

Open-source databázový framework, například NHibernate, což je port frameworku Hibernate z Javy do .NETu, či DbEntry.Net[7], které jistě řeší přístup k různým databázím velmi dobře. Oba výše uvedené navíc podporují námi požadované databázové servery. Jediný problém může nastat až v případě, kdy zjistíme, že daný framework neumí něco, co potřebujeme. To se obvykle neprojeví hned na začátku vývoje, ale až později, takže je nutné buď změnit framework nebo se pokusit ho upravit pro naše potřeby. Tak jako tak je to poměrně časově náročné.

Vlastní databázový framework napsaný přímo na míru. Na jedné straně to obnáší „objevování kola“, na druhé straně je takováto abstrakce databáze plně přizpůsobena našim potřebám a i případné úpravy v průběhu vývoje jsou snazší.

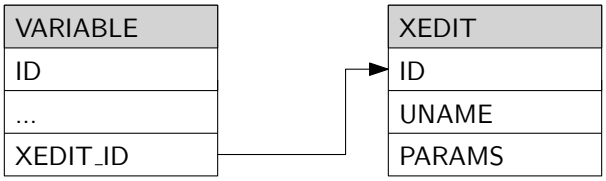
Protože vlastní databázový framework byl k dizpozici a jeho koncepce se již osvědčila v předchozích projektech, bylo rozhodnuto, že bude s úpravami a

vylepšeními použit i tentokrát. Pokud bychom však takovouto vrstvou pro přístup k databázi nedisponovali, pravděpodobně by byl použit jeden z výše zmíněných open-source frameworků.

2.3 Koncepce grafických schémat

2.3.1 Ukládání schémat

Stávající schémata jsou uložena v databázi v tabulce `XEDIT`, která obsahuje sloupce `ID` (jednoznačný identifikátor schématu), `UNAME` (unikátní jméno schématu) a `PARAMS` (binární reprezentace schématu). Tabulky `LOGNOD_X` a `VARIABLE` obsahují mimo jiné sloupec `XEDIT_ID`, což je cizí klíč do tabulky `XEDIT`, kterým je uskutečněna vazba mezi uzlem logické struktury a jeho schématem.



Obrázek 2.1: Stávající ukládání schémat v databázi.

Protože jedním z požadavků je, aby současná schémata a nová schémata mohla být používána současně, není možné tyto tabulky využít pro ukládání nových schémat. Aby byly změny ve struktuře databáze co nejmenší, bylo zvoleno takové řešení, kdy schémata budou ukládány do tabulky `SCHEME`, která je velmi podobná tabulce `XEDIT`. Vazba mezi uzly logické struktury a schématy bude pak zajištěna pomocí vazební tabulky `NOD_SCHEME`, která obsahuje sloupce:

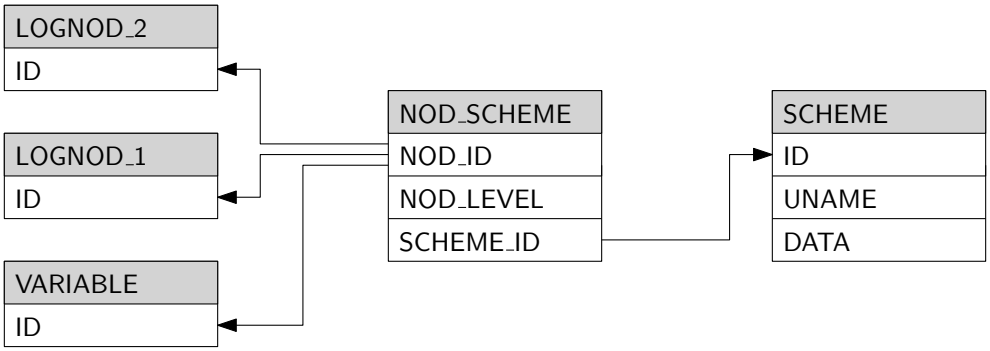
`NOD_ID` obsahující jednoznačný identifikátor uzlu a je to cizí klíč do tabulky uzlů nebo proměnných.

`NOD_LEVEL` určuje logickou úroveň uzlu (1, 2, atd.) nebo proměnné (0).

`SCHEME_ID` je cizí klíč do tabulky `SCHEME`.

Kdyby sloupec `NOD_LEVEL` použit nebyl, bylo by nutné vytvořit redundantní kopie tabulky `NOD_SCHEME` pro každou logickou úroveň systému (`VAR_SCHEME`,

NOD_1_SCHEME, atd.). Nad kombinací sloupců NOD_ID a NOD_LEVEL je vytvořen unikátní klíč, který zaručí, že každý uzel logické struktury bude moci mít přiřazeno pouze jedno schéma.



Obrázek 2.2: Ukládání nových schémat v databázi.

Navržené tabulky nebude nutné při instalaci serverové části vytvářet nebo „ručně“ přidávat do databáze. Serverová služba vždy při startu zkontroluje, zda se v databázi nachází a v případě, že ne, je vytvoří.

2.3.2 Reprezentace schémat

Statické grafické schéma, tedy schéma bez návaznosti na data, bude objekt, který bude mít definovanou šířku, výšku, barvu pozadí, jméno a jednoznačný identifikátor. Bude sestávat ze setříděné kolekce vrstev, kde pozice v kolekci bude odpovídat tomu, jak vysoko se vrstva nachází. Schéma bude složeno z prvků, které budou určeny jednoznačným identifikátorem v rámci schématu. Každý prvek bude dále obsahovat identifikátor vrstvy, do které patří, pozici, na které se nachází, výšku a šířku. Od takovéhoto obecného prvku lze podědit již konkrétní prvky, které budou vlastosti rozšiřovat o další specifické. Například obdélník bude mít oproti obecnému prvku navíc barvu výplně, barvu okraje a tloušťku okraje. Měly by být implementovány tyto základní prvky: čára, obdélník, elipsa, textové pole a z databáze vložený bitmapový obrázek. Protože stávající schémata mají poměrně technický vzhled, bylo po konzultaci se zadavatelem rozhodnuto, že tyto prvky by měly k vytváření schémat stačit. I praxe ukazuje, že uživatelé mají tendenci vytvářet schémata tak, že jako pozadí je použit nějaký bitmapový obrázek a pouze prvky, které se mají v závislosti na datech měnit, jsou vytvořeny

v editoru schémat. Není tedy nutné implementovat například křivky nebo jiné funkce plnohodnotných grafických editorů.

2.3.3 Propojení statických schémat s daty

Při prohlížení stávajících schémat se ukázalo, že většina z nich se skládá z textových polí, do kterých skripty dynamicky plní vybrané hodnoty naměřených vzorků. Dále pak typické schéma obsahuje grafickou reprezentaci provozu, stanice nebo přenosové sítě, která se v závislosti na datech mění. V drtivé většině případů jsou tyto změny jednoduchého charakteru, například nastavení barvy prvku přímo na barvu odpovídající hodnotě daného vzorku a podobně. Skripty tedy neobsahují mnoho logiky, většinou jen nastavují konkrétní vlastnost konkrétního prvku, což je možno vidět i na ukázce schématu na obrázku 1.3, ve kterém se v závislosti na datech mění textové popisky u okrajů a barva ikonek reprezentujících jednotlivé stanice.

Proto bylo rozhodnuto, že koncepce propojení schémat s daty bude pozměněna. Skripty v podobě, ve které byly dosud používány, budou zrušeny a nahradí je dvě technologie, které nabídnou stejnou funkcionalitu, jakou měly skripty, ale o poznání jednodušší použití.

Datové zdroje

Při vytváření bude možné do schématu přidat tzv. datové zdroje, které budou zpřístupňovat naměřené vzorky dat. Každý datový zdroj bude určen:

Uzlem logické struktury, ve kterém byl vzorek dat změřen.

Typem dat, který je požadován. Tedy buď archivní, poslední nebo průběžný vzorek.

Datovou úrovní, na které byl vzorek dat měřen.

Datumem a časem vzorku dat, což má smysl jen u archivního typu dat.

Pokud bude uživatel vytvářet nebo upravovat prvky ve schématu, bude moci nastavit hodnotu vlastností nejen konstantně (např. nastavit barvu výplně obdélníku na červenou), ale i propojit danou vlastnost s datovým zdrojem (např. propojit barvu výplně obdélníku s barvou vzorku vybraného datového zdroje). Datové zdroje budou poskytovat několik hodnot, které lze využít jako „cíl“ propojení (binding target). Bude to už výše zmíněná barva vzorku,

hodnota, hodnota s jednotkou, stav vzorku, vlastnosti uzlu, kde byl vzorek změřen a jiné.

Uživatelské funkce

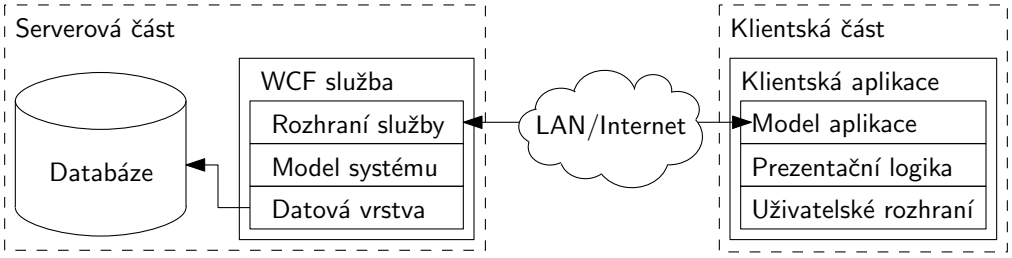
Datové zdroje však nemohou zcela nahradit původní skripty, které v některých případech provádějí s daty složitější operace. Aby i tato funkčnost zůstala zachována, vzniknou tzv. uživatelské funkce. Bude se jednat o uživatelem naprogramované funkce, které jako parametry budou přijímat 1-n datových zdrojů. V těle funkce bude možné přistupovat k hodnotám datových zdrojů předaných v parametrech a na základě těchto hodnot provádět výpočty. Návratovou hodnotou pak bude nějaký primitivní datový typ. Kromě konstantní hodnoty a hodnoty z datového zdroje bude možné propojit vlastnost prvku ve schématu i s návratovou hodnotou z nějaké uživatelské funkce (samozřejmě pouze v případech, kdy si budou vlastnost a návratová hodnota typově odpovídat).

Kapitola 3

Implementace

3.1 Architektura aplikace

Základní rozdělení aplikace na serverovou a klientskou část je dáno požadavky. Tyto dvě části jsou z důvodu poměrně velké velikosti ještě dále rozčleněny na další logické vrstvy. Každá takováto logická vrstva odpovídá buď samostatné knihovně nebo konkrétní aplikaci. Datová vrstva a model systému jsou samostatné knihovny, rozhraní služby je aplikace typu WCF Service Application. Model klientské aplikace je opět knihovna, prezentační logika spolu s uživatelským rozhraním jsou součástí WPF aplikace.



Obrázek 3.1: Architektura klientské a serverové části aplikace.

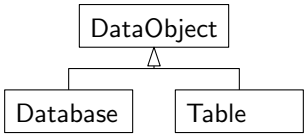
3.2 Datová vrstva

Zdrojové soubory datové vrstvy se nachází v projektu `Server.Data` a jsou součástí jmenného prostoru `SBWebExplorer.Server.Data`. Účelem datové

vrstvy je sjednocovat přístup k různým databázovým serverům a poskytovat co možná největší průnik funkcionalit dostupných na jednotlivých databázích. Pokud některý databázový server něco neumožňuje, pak datová vrstva, v případě, že je to možné, emuluje toto chování na aplikační úrovni. Příkladem může být neexistence klauzule `LIMIT` v `SELECT` dotazu na Microsoft SQL Serveru (dále MSSQL), která však u MySQL existuje a v některých případech velmi pomáhá ušetřit objem přenášených dat. Pro MSSQL je tato klauzule emulována a na venek se tedy vrstva chová jednotně bez ohledu na typ databáze. Nejdůležitější objekty této vrstvy by se daly rozdělit do tří skupin: datové objekty, konektory a adaptéry.

3.2.1 Datové objekty

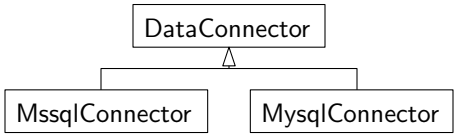
`DataObject` je objekt, který reprezentuje obecnou databázovou entitu a slouží jako rozhraní se kterým uživatel datové vrstvy pracuje. Jeho nejpoužívanější potomek je `Table`, nad kterým lze provádět většinu operací (pracujících s databázovými tabulkami) známých z jazyka SQL (`CREATE`, `INSERT`, `UPDATE`, `SELECT`, `DELETE` a další). Umožňuje třeba také testovat, jestli daná tabulka v databázi existuje, což nemá žádný ekvivalent v jazyce SQL. `Database` obsahuje pouze jedinou metodu testující připojení k databázi. `DataObject` i jeho potomci jsou nezávislé na typu databáze, dalo by se říci, že jde o objektovou reprezentaci toho, co od databáze naše aplikace požaduje. Tedy například jaké tabulky má databáze obsahovat, což lze specifikovat poděděním konkrétních tabulek od `Table`, z jakých sloupců se tyto tabulky mají skádat apod. Neříkají už ale, jak konkrétně má být toto realizováno na úrovni databáze.



Obrázek 3.2: Hierarchie datových objektů.

3.2.2 Datové konektory

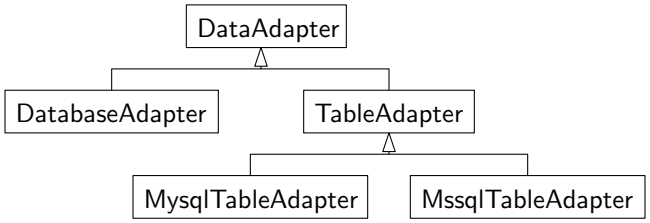
Objekt `DataConnector` definuje abstraktní metody pro vytvoření spojení s databázovým serverem (`System.Data.DbConnection`[1]) a pro vytvoření dotazu do databáze (`System.Data.DbCommand`). Další metody tyto dvě využívají a provádějí různé typy dotazů na databázi (vracející jednu hodnotu, vracející tabulku, nevracející nic). Potomci `MssqlConnector` a `MysqlConnector` pouze definované abstraktní metody implementují.



Obrázek 3.3: Hierarchie datových konektorů.

3.2.3 Datové adaptéry

`DataAdaptery` se nachází mezi `DataObjecty` a `DataConnectory` a slouží k propojení plně abstraktního světa `DataObjectů` s konkrétním `DataConnectorem`. Například datový objekt `Table` má pro podporované typy databázových serverů své adaptéry (`MssqlTableAdapter` a `MysqlTableAdapter`). Adaptéry na základě informací obsažených v objektu `Table` sestavují již konkrétní SQL dotazy v odpovídajícím dialektu jazyka SQL.

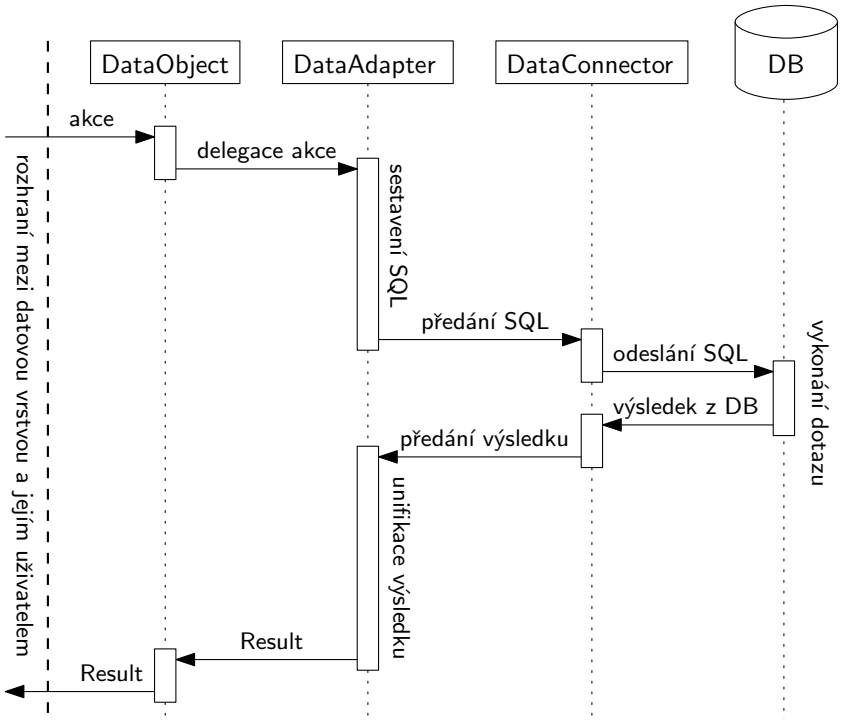


Obrázek 3.4: Hierarchie datových adaptérů.

Typický scénář je, že na `DataObjectu` je uskutečněna nějaká akce (například vkládání do tabulky), `DataObject` tuto akci deleguje na konkrétní `DataAdapter`

[1] Všechny třídy z jmenného prostoru `System.Data` jsou součástí frameworku .NET, konkrétně části ADO.NET.

a připojí k tomu dodatečné informace (například data, která se mají vložit). `DataAdapter` podle održených informací sestaví samotný SQL dotaz (`INSERT INTO [tabulka] ...`) a předá ho `DataConnectoru`, který tento dotaz odešle na databázový server. Sequence diagram komunikace těchto objektů je na obrázku 3.5.



Obrázek 3.5: Sequence diagram komunikace objektů datové vrstvy.

3.2.4 Další objekty

Knihovna obsahuje také mnoho „podpůrných“ objektů, které zpravidla slouží k uchovávání nějaké skupiny informací. Je to třeba `Column` (informace o sloupci) či `JoiningTable` (informace o propojení dvou tabulek při selectu, tedy typ spojení, sloupce přes které spojení probíhá a podmínky spojení). Nemají však v sobě žádnou složitou logiku, používají se víceméně jako kontejnery na data a podle informací v nich uložených `DataAdapter` sestavuje SQL dotaz.

Result

Objekt `Result` slouží k zapouzdření hodnot získaných při selekci z databáze. Pokud vybíráme nespecifikované data, tak je `Result` naplněn `DataConnectorem` hned po provedení SQL dotazu. V případě, že ale vybíráme data z nějaké tabulky specifikované v objektu `Table`, je nutné aby měl `TableAdapter` přístup přímo k datům vráceným z databáze. Takže `DataConnector` nevrací `Result`, ale pouze `System.Data.DbDataReader`, který je pak dále zpracován konkrétním `TableAdapterem`. `TableAdapter` v tomto případě data z databáze transformuje (např. `DbNull` na `null`), sjednocuje (stejná data získaná z různých databázových serverů mohou být reprezentována různě a je tedy nutné je převést do jednotného formátu) a pak teprve plní objekt `Result`, který je vrácen uživateli knihovny. V ranných fázích vývoje byl tento objekt také serializovatelný, aby ho bylo možné posílat přes WCF službu klientovi, ale od toho bylo později upuštěno. Pole působnosti `Resultu` a vlastně celé datové vrstvy je tak pouze na serveru, klient s žádnými objekty z této knihovny nepřijde do styku.

Configuration

Objekt `Configuration` zajišťuje uchovávání a načítání konfiguračních dat uložených v souboru `Web.config`. Jde o typ databázového serveru (MSSQL, MySQL), port, kterým má být přistupováno k databázovému serveru, jméno databáze, uživatelské jméno a heslo pro přístup k databázi.

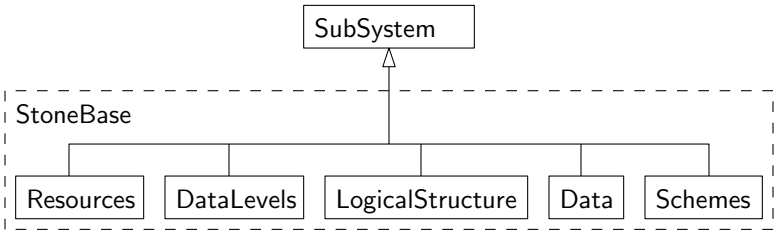
3.2.5 Budoucnost

Do vrstvy pro přístup k databázi jsou rovněž zahrnuty objekty poděděné od objektu `Table` odpovídající jednotlivým tabulkám v databázi. Asi čistější řešení by bylo udělat z této knihovny úplně autonomní projekt a vyčlenit z naší aplikace. V aplikaci by byly už jen výše zmíněné konkrétní tabulky. To se s největší pravděpodobností v budoucnu také stane a přestože je knihovna již nyní znovupoužitelná, bude nutné udělat menší úpravy, aby opětovné použití v jiných projektech bylo především jednoduché a pohodlné.

3.3 Model systému

Jedná se o objektový model systému pracující nad parametrizačními daty z databáze. Všechny jeho třídy se nacházejí v projektu `Server.Model` a

spadají do jmenného prostoru `SBWebExplorer.Server.Model`. Systém, jako takový, je reprezentován objektem `StoneBase`, jenž se skládá z logických podsystémů. Každý podsystém je potomkem třídy `SubSystem` a stará se o jednu logickou oblast v systému (např. logická struktura, zdroje). Další pomocné třídy jsou definovány v adresářích pojmenovaných stejně jako podsystém, se kterým logicky souvisí. Zároveň jsou některé objekty serializovatelné, takže se využívají i pro přenos dat a informací ze systému klientovi.



Obrázek 3.6: Rozdělení modelu systému na systém a podsystémy.

3.3.1 Třída `StoneBase`

Statická část třídy `StoneBase` obsahuje dvě metody:

`GetSystems()` při prvním zavolání projde tabulku `_SB_SYSTEMS` a podle dat v ní obsažených vytvoří instance třídy `StoneBase` reprezentující jednotlivé systémy v databázi. Kolekci objektů `StoneBase` si uloží pro pozdější použití a vrátí ji. Při každém dalším zavolání této metody nejsou objekty `StoneBase` znovu vytvářeny, vrátí se ty, které byly zkonstruovány při prvním zavolání. Jelikož třída `StoneBase` definuje jediný privátní konstruktor, představuje tato metoda factory method na objekty této třídy.

`GetSystem(string systemCode)` v uložené kolekci najde instanci systému podle jednoznačného kódu a vrátí ji.

Objekt `StoneBase` si ve svém konstruktoru načte všechny parametry z databáze a inicializuje své podsystémy. Tento objekt však neobsahuje mnoho logiky, je mimo jiné zodpovědný za autentizaci uživatelů. Autentizace probíhá tím způsobem, že v rozhraní vrstvy je zveřejněna ověřovací metoda, která má jako parametry kód systému, uživatelské jméno a heslo. Podle

kódu systému předá odpovídajícímu objektu `StoneBase` přihlašovací údaje. Systém údaje ověří a v případě úspěchu je vrácen objekt `User` s informacemi o přihlášeném uživateli. V ostatních případech je vyhozena výjimka, která projde až ke klientovi, kde je odchycena.

3.3.2 Podsystém `Resources`

Podsystém `Resources` je zodpovědný za poskytování systémových zdrojů. Zdroje by se daly rozdělit do dvou základních skupin a to neměnné a dynamické. Neměnné zdroje se načtou z databáze po spuštění aplikace a po celou dobu běhu se nemění, nelze je změnit ani pomocí aplikace pro konfiguraci systému. Jedná se například o textové popisy alarmových událostí či možné stavy naměřených vzorků. Naopak dynamické zdroje není možné načíst při spuštění aplikace a pamatovat si je po celou dobu běhu, protože mohou být libovolným uživatelem s patřičnými právy v průběhu pozměněny, přidány nebo odebrány. Jsou z databáze načítány vždy až ve chvíli, kdy jsou potřeba. Jde o texty, barvy, zvuky, bitmapové obrázky atd.

3.3.3 Podsystém `DataLevels`

`DataLevels` obsahuje informace o datových úrovních systému a provádí nad nimi různé operace. Každá datová úroveň má několik vlastností, nejdůležitější je vzorkovací perioda (`SamplePeriod`), což je délka intervalu mezi dvěma měřeními a posun vzorku (`SampleShift`). Posun vzorku je typicky na úrovních s vzorkovací periodou vyšší než den a jeho hodnota je šest hodin, na nižších úrovních je nulový. Jeho význam je, že vzorky naměřené mezi 00:00:00 a 06:00:00 spadají do předcházejícího dne. Některé datové úrovně navíc nejsou vyjádřeny pevně, například měsíční datová úroveň má vzorkovací periodu závislou na aktuálním měsíci, podle toho kolik má zrovna dní. S těmito informacemi podsystém pracuje a umožňuje:

Zaokrouhlování datumů na dané datové úrovni (`RoundDateTime`). Například čas 10:13:13 na úrovni „30 minut“ je zaokrouhlen na 10:00:00.

Posun datumů na datové úrovni o konkrétní počet vzorků (`ShiftDateTime`). Pokud například posuneme 1.1.2010 10:00:00 na úrovni „30 minut“ o 5 vzorků dopředu, vyjde 1.1.2010 12:30:00. Pokud bychom však posouvali na úrovni „den“, vyšlo by 6.1.2010 10:00:00.

Formátování datumů podle datové úrovně (`GetDateTimeFormat`).

Převod datumů z prezentačních na databázové a naopak (`PresentationDateTimeToDatabase`, `DatabaseDateTimeToPresentation`). Čas měření vzorku je totiž v databázi ukládán jinak, než má být prezentován uživateli.

A další podobné operace.

3.3.4 Podsystém `LogicalStructure`

`LogicalStructure` pracuje s logickou strukturou systému. Protože vytvářená aplikace nemá za úkol nějak tuto strukturu měnit, je tento podsystém zodpovědný pouze za získávání požadovaných dat o uzlech, proměnných, datových úrovní proměnných a stavech proměnných. Tyto data zopouzdřuje do odpovídajících objektů (`Node`, `Variable`, `VariableLevel`, `VariableState`), které jsou serializovatelné a v případě potřeby jsou odeslány klientovi. Při získávání informací o logické struktuře systému je nutné kontrolovat práva uživatele, který aplikaci zrovna používá, aby mu nemohly být zobrazeny uzly, které nesmí vidět. Uzly logické struktury se chovají obdobně jako dynamické zdroje, tedy je nutné počítat s tím, že v průběhu mohou vznikat a zanikat uzly, proměnné nebo dokonce celé větve tohoto stromu.

3.3.5 Podsystém `Schemes`

Podsystém `Schemes` zařizuje vytvoření tabulek pro ukládání schémat v databázi pokud se tam již nenacházejí. Zajišťuje také serializaci objektové reprezentace schématu do binární podoby při ukládání do databáze a deserializaci při načítání z databáze. Do tohoto podsystému spadá poměrně hodně tříd, které odpovídají různým částem grafického schématu. Všechny jsou serializovatelné kvůli ukládání do databáze a implementují rozhraní `System.ICloneable`, aby bylo jednoduše uskutečnitelné kopírování a vkládání.

Objekty `Scheme`, `Layer`, `DataSource`, `Element`

Tyto čtyři objekty reprezentují grafické schéma, vrstvu, datový zdroj a obecný prvek schématu. Obsahují takové vlastnosti a chovají se tak, jak je popsáno v analýze.

Objekt `UserFunction`

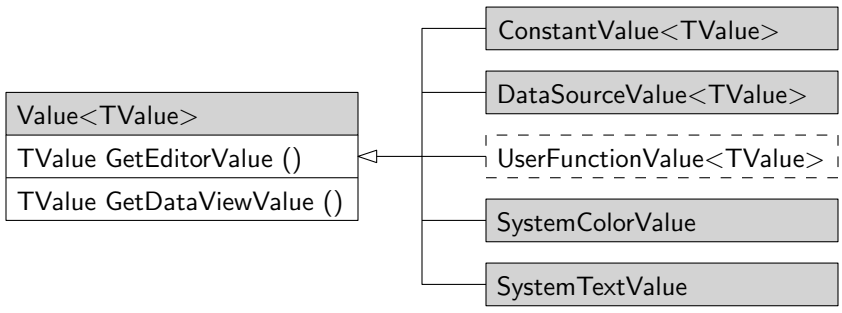
Objekt `UserFunction` zatím implementován není. Měl by reprezentovat uživatelskou funkci a mezi jeho vlastnostmi by mělo být jméno funkce, typ návratové hodnoty, počet a jména parametrů (všechny budou stejného typu `Sample` viz 3.3.6), programovací jazyk, jakým je funkce napsána a samotný kód těla funkce. Podporované programovací jazyky budou C# a Visual Basic .NET, které je možné kompilovat za běhu.

Třída `Value`

`Value` je velmi důležitá generická třída, díky níž je v důsledku možné propojovat vlastnosti prvků schématu s datovými zdroji nebo uživatelskými funkcemi. Vlastnost prvku, u které má být možnost propojení není primitivního typu (barva výplně obdélníku není typu `Color`), nýbrž je typu `Value` (barva výplně obdélníku je typu `Value<Color>`). Třída `Value` definuje dvě metody, které vracejí skutečnou hodnotu (již primitivního typu). Je to `GetEditorValue` vracející hodnotu, která se má použít v editoru schémat a `GetDataViewValue`, která se použije při zobrazení schématu spolu s daty v prezentační stránce. Od třídy `Value` jsou poděděny další třídy (obrázek 3.7):

`ConstantValue` odpovídá konstantní hodnotě, kterou uživatel zadá v editoru schémat a která při změně dat zůstává stejná.

`DataSourceValue` je hodnota propojená s datovým zdrojem. V konstruktoru dostává identifikátor zdroje a jméno vlastnosti datového zdroje, se kterou je propojena.



Obrázek 3.7: Podtřídy třídy Value.

`UserFunctionValue` zatím není (stejně jako `UserFunction`) implementována. Bude ale reprezentovat hodnotu získanou voláním uživatelské funkce. V konstruktoru obdrží jméno funkce a seznam identifikátorů datových zdrojů, jejichž vzorky dat se mají uživatelské funkci předat jako parametry.

`SystemColorValue` je poděděná od `Value<Color>` a reprezentuje systémovou barvu. Vazba je uskutečněna přes identifikátor systémové barvy, takže když se hodnota systémové barvy změní, změní se i hodnota všech vlastností, které jsou na ni napojeny.

`SystemTextValue` dědí od `Value<string>` a je podobná `SystemColorValue` s tím rozdílem, že reprezentuje systémový text.

Konkrétní prvky schématu

Mezi konkrétní prvky schématu patří `LineElement`, `RectangleElement`, `EllipseElement`, `LabelElement` a `ImageElement`. Z vlastností rodičovské třídy `Element` dědí pozici, výšku a šířku. Tyto zděděné vlastnosti však zatím není možné propojit s datovým zdrojem nebo uživatelskou funkcí. Každý prvek pak definuje pro něj specifické vlastnosti, například `RectangleElement` definuje barvu výplně (`FillColor`) typu `Value<Color>`, barvu okraje (`StrokeColor`) typu `Value<Color>` a tloušťku okraje (`StrokeThickness`) typu `Value<double>`. Vlastnosti, které jsou propojitelné s datovými zdroji či funkcemi jsou opatřeny atributem `BindableProperty` a jejich typ je primitivním typem parametrizovaná třída `Value<TValue>`.

3.3.6 Podsystém `Data`

Hlavní náplní práce objektu `Data` je získávání naměřených dat, propojení těchto dat s uzly, kde byly naměřeny, dopočítání hodnot, které se v databázi nenachází a vrácení tohoto výsledku klientovi. Tento podsystém obsahuje metody pro získání dat odpovídajících jednotlivým prezentačním stránkám, tedy `GetStates` pro stavy, `GetTable` pro tabulku a grafy, `GetAlarms` pro přehled alarmů a `GetSchemeResult` pro grafická schémata. Samotný požadavek o data má ve všech případech stejné vlastnosti, které klient zabalí do objektu `RequestProperties`, podle kterého jsou data vybírány z databáze. Jsou to tyto vlastnosti:

Uzel nebo uzly ve kterých byly vzorky naměřeny.

Datová úroveň na které byly vzorky měřeny nebo dopočítány.

Typ dat požadovaných vzorků (archivní, poslední nebo průběžné).

Datum a čas určující, kdy byly vzorky změřeny.

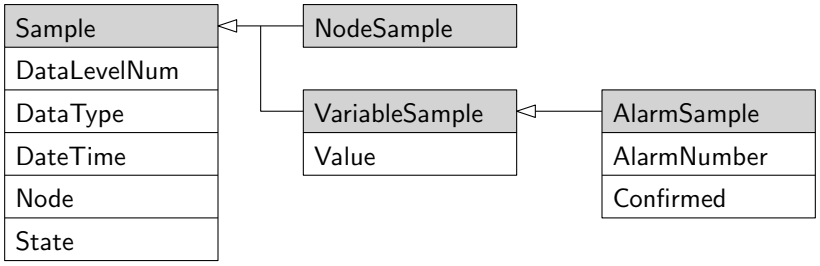
Informace o stránkování obsahující velikost jedné stránky a číslo požadované stránky. Tohoto se využívá ve všech případech kromě `GetSchemeResult`, protože velikost výsledků může být velmi velká a je tedy nutné je rozdělit do několika stránek.

Počet vzorků v trendu hraje roli u tabulky a grafů, kde nestačí zobrazit vzorky jen z datumu a času určeného v parametru uvedeném výše. Například na měsíční datové úrovni bývá obvykle dvanáct vzorků v trendu, takže v tabulce nebo grafu jsou vidět data všech dvanácti měsíců daného roku.

V případě získávání dat schématu (`GetSchemeResult`) jsou data z databáze vybírány primárně podle vlastností jednotlivých datových zdrojů. Pokud je však některá vlastnost datového zdroje nedefinovaná (resp. definovaná jako „zvolená uživatelem klientské aplikace“) použije se hodnota z objektu `RequestProperties`.

Sample

Důležitými objekty souvisejícími s tímto podsystémem jsou také `Sample` a od něj poděděné `NodeSample`, `VariableSample` a `AlarmSample`, které odpovídají obecnému vzorku dat, resp. sdruženému vzorku dat pro uzel, vzorku dat změřenému v proměnné a vzorku, který způsobil alarmovou událost.



Obrázek 3.8: Hierarchie vzorků.

Tyto objekty slouží k zapouzdření informací o jedné konkrétní naměřené hodnotě. Každý vzorek má podobné vlastnosti jako `RequestProperties` (kromě stránkování a počtu vzorků v trendu), dále pak stav vyjadřující jestli jsou naměřená data platná, neplatná, neměřená či chybová (`State`). V závislosti na tomto stavu je vzorku přiřazena barva a text. `VariableSample` obsahuje navíc hodnotu (`Value`). Vzorek naměřený v uzlu žádnou hodnotu nemá, má pouze stav, který shrnuje stav všech potomků. Od `VariableSample` je ještě poděděn `AlarmSample`, který přidává číslo alarmové události a datum, kdy byl alarm potvrzen. Tyto objekty jsou serializovatelné a jsou odesílány klientovi v rámci odpovědi na nějaký požadavek o data, `GetStates` tedy vrací seznam objektů `Sample`, `GetTableData` vrací dvourozměrné pole objektů `Sample` atd.

Při získávání dat pro grafické schéma je každému datovému zdroji přiřazen jemu odpovídající `Sample`. Když je pak na nějaké vlastnosti prvku schématu zavolána metoda `GetDataViewValue`, podle identifikátoru datového zdroje se přistoupí k jemu odpovídajícímu objektu `Sample` a z něj se teprve získá hodnota, která se má použít a kterou metoda `GetDataViewValue` vrátí.

3.4 Architektura klienta

Klientská aplikace je logicky rozdělena do tří vrstev na model klientské aplikace, vrstvu prezentační logiky a uživatelské rozhraní. Model aplikace zapouzdřuje komunikaci se serverem, uchovává všechny informace, které uživatel zadal a jsou potřebné při komunikaci s webovou službou a vyvolává některé události (např. událost oznamující příjem dat ze serveru). Jeho třídy jsou součástí projektu `Client.Model` a jsou obsaženy ve jmenném prostoru `SBWebExplorer.Client.Model`. Vrstva prezentační logiky a uživatelské rozhraní částečně splývají a jsou součástí jediného projektu `Client`. Třídy těchto dvou vrstev spadají do jemnného prostoru `SBWebExplorer.Client`.

K definici uživatelského rozhraní byl použit jazyk XAML[2]. Částečně se pak vrstva uživatelského rozhraní nachází i v souborech code-behind (v jazyce C#), pokud na požadovanou funkčnost nestačí jazyk XAML.

Vrstva prezentační logiky má za úkol reagovat na uživatelské akce a podle nich provádět změny v modelu nebo volat patřičné metody. Reaguje také

[2]XAML[8] (eXtensible Application Markup Language) je deklarativní jazyk založený na XML, který slouží k vytváření uživatelského rozhraní v cílové technologii WPF.

na události vyvolané modelem klientské aplikace a v závislosti na nich mění uživatelské rozhraní. Tato „vrstva“ se volně řečeno rozprostírá ve zbývajících částech code-behind souborů a dalších spolupracujících třídách.

3.4.1 Model aplikace

Model aplikace sestává z několika vzájemně spolupracujících objektů, veškerá komunikace se serverovou částí probíhá v oddělených vláknech prostřednictvím objektů `System.ComponentModel.BackgroundWorker`[3]. Tato vrstva není moc rozsáhlá, protože při vývoji byla snaha, aby pokud možno veškerá důležitá logika byla co nejblíže datům, tedy na straně serveru.

Třída `Session`

Objekt `Session` odpovídá jedné uživatelské ralaci s aplikací. Umožňuje navázat spojení se serverem, zajišťuje přihlašování uživatele do systému na serveru, se kterým je aktuálně vytvořené spojení a uchovává informace o systému a uživateli. Tento objekt také vyvolává několik souvisejících událostí, jde o signalizaci zahájení připojování (`Connecting`), připojení k serveru (`Connected`), selhání připojení k serveru (`NotConnected`) a obdobné události pro přihlašovací proces.

Třída `SessionNodes`

`SessionNodes` slouží k dynamickému načítání částí stromu logické struktury systému, podle toho, jak uživatel stromem prochází. Obsahuje také identifikaci uživatelem zvoleného uzlu v zobrazeném stromu struktury systému, která je použita při vytváření objektu `RequestProperties` a získávání dat ze serveru. Podobně jako objekt `Session` vyvolává události oznamující načítání a obdržení/neobdržení dat ze serveru.

Třída `SessionData`

Objekt `SessionData` zapouzdřuje v objektu `DataProperties` vlastnosti dat, které uživatel zvolil (datum a čas, datová úroveň apod.) a na základě nich vytváří objekt `RequestProperties`. Jedinou veřejnou metodou objektu `SessionData` je `RequestData`, která načítá podle aktuálně zvolené prezentační

[3] `System.ComponentModel.BackgroundWorker` je standardní součástí .NET frameworku a je schopen provádět definované operace ve vláknech na pozadí.

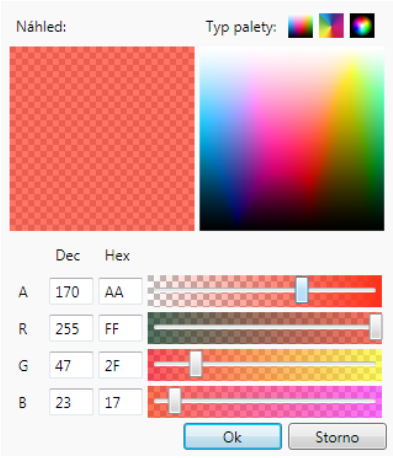
stránky odpovídající data ze serveru. O průběhu načítání informuje podobnými událostmi, jako předchozí dva objekty.

Třída `SchemeEditor`

`SchemeEditor` je objekt, který reprezentuje instanci editoru schémat. V konstruktoru obdrží schéma, které má být editováno spolu s dalšími informacemi potřebnými k editaci schématu (seznam systémových barev, textů, obrázků). Uchovává uživatelem zvolenou hodnotu přiblížení, velikost mřížky a jestli je zapnuta, referenci na vybranou vrstvu, seznam referencí na vybrané elementy, seznam referencí na elementy, které se zrovna nacházejí ve schránce a další. Náplní jeho práce je ukládání schématu, načítání dat obrázků ze serveru, operace se schránkou (vyjmout, kopírovat, vložit) a také, pokud je zapnuto přichytávání k mřížce, zaokrouhluje body na nejbližší průnik dvou čar mřížky. Mezi události, které vyvolává patří notifikace o změnách ve schránce, vybraných prvcích schématu nebo přiblížení a také události informující o průběhu ukládání schématu a načítání obrázků.

3.4.2 Uživatelské rozhraní

Uživatelské rozhraní se skládá jednak z obecných, znovupoužitelných prvků (namespace `SBWebExplorer.Client.Controls`) a druhak z „pohledů“, což jsou okna, user controls a custom controls (`SBWebExplorer.Client.Views`) dávající smysl jen ve spojení s vytvářenou aplikací, a proto nejsou rozumně znovupoužitelné.



Obrázek 3.9: ColorPicker.

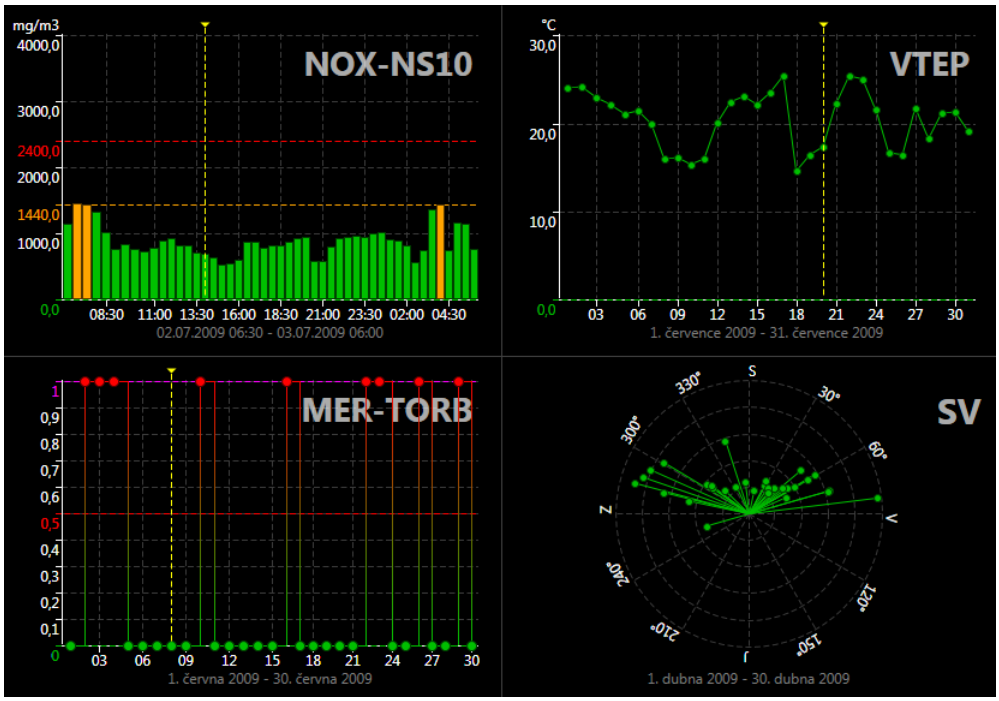
Controls

Mezi tyto prvky spadá několik druhů textových polí (poděděné od `TextBoxu`), které akceptují jen povolenou hodnotu (celé číslo, reálné číslo, neprázdný text). Zajímavější je kalendář umožňující zadání datumu a času a barevná paleta (`ColorPicker`). Barevná paleta je založena na objektu `WPFColorPicker`[9] od Sachy Barbera a je rozšířena o možnost

přesného zadání barvy hexadecimální i dekadickou hodnotou a o posuvníky, které volí hodnotu jednotlivých složek barvy.

Views

Mezi nejdůležitější views patří okna (přihlašovací okno, hlavní okno aplikace, editor schémat, okno pro přidání datových zdrojů do schémat) nebo různé user controls (`TreeView` s logickou strukturou systému, jednotlivé prezentační stránky, přehled vrstev schématu apod.), ale tyto třídy nejsou z programátorského hlediska až zas tak zajímavé.



Obrázek 3.10: Různé typy grafů.

Zajímavější je objekt `Graph` a od něj poděděné `LineGraph`, `BarGraph` a `RoseGraph`, což jsou grafové komponenty používané v prezentační stránce s grafy. Protože nebylo možné si dovolit zakoupit nějakou komponentu zobrazující grafy a ani nebyla nalezena dostatečně robustní volně šiřitelná alternativa, bylo rozhodnuto, že grafy budou naprogramovány vlastními silami. Existují čtyři typy grafů a to: sloupcový, čarový, čarový pro binární

proměnné a větrná růžice. V tomto pořadí jsou také zobrazeny na obrázku 3.10 (zleva doprava a dolů). Grafy umí adaptovat svou velikost podle dostupné výšky a šířky, hodnoty a rozsah osy y volí až podle dat, které mají být zobrazeny, zobrazují meze a umožňují kliknutím zvolit čas vzorku.

Druhým netriviálním objektem je `EditorCanvas`, což je plátno editoru schémat. Jsou v něm vykresleny jednotlivé prvky schématu, které je možno vybírat, měnit jejich pozici a velikost. S tím souvisí i přidružené objekty `MoveThumb`, `ResizeThumb` a jiné, které reprezentují „úchopy“, za které je možné tahat a měnit tak velikost nebo pozici prvku. Většina této funkcionality však byla prakticky bez úprav převzata z editoru diagramů, který je popsán v článku WPF Diagram Designer, Part 2 [10]. Vytváření nových prvků ve výše zmíněném editoru diagramů je pro naše účely řešeno nešikovně. Proto je nad `EditorCanvas` umístěno další plátno (`ToolCanvas`), které v případě, že je zvolen tvořící nástroj (čára, obdélník), zachytává události myši a v závislosti na nich vykresluje nově vytvářený objekt. Jakmile je prvek dokončen, je z tohoto plátna vyjmut a přidán ke všem ostatním prvkům do `EditorCanvas`u.

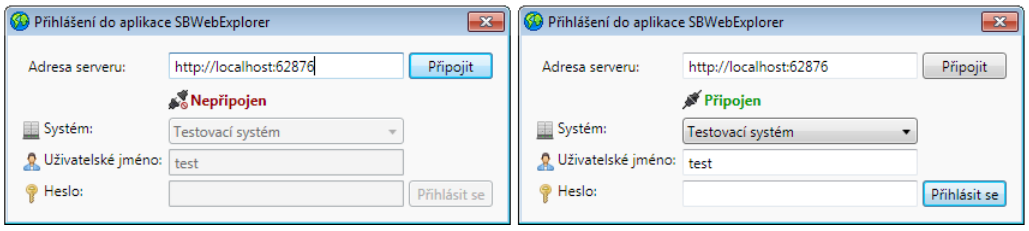
3.4.3 Vrstva prezentační logiky

Tato „pseudovrstva“ není nějak konkrétně vymezena, nachází se z valné většiny v metodách reagujících na chování uživatele, které ovlivní model aplikace. A opačně v metodách reagujících na události vyvolané modelem, které znamenají změny pro uživatelské rozhraní. Původně tato vrstva byla samostatná (jednalo se o vrstvu Presenters známou z návrhového vzoru Model-View-Presenter), ale v průběhu vývoje se ukázalo toto rozdělení jako zbytečně složité a zhoršující orientaci v projektu.

Kapitola 4

Aplikace z pohledu uživatele

Po spuštění aplikace se zobrazí okno (obrázek 4.1), do kterého je nutné zadat adresu serveru. Až po připojení k serveru je možné vybrat systém, zadat uživatelské jméno a heslo a přihlásit se. Systém i uživatelské jméno jsou na obrázku vyplněné již před připojením, protože aplikace si pamatuje tyto údaje z poslední relace.

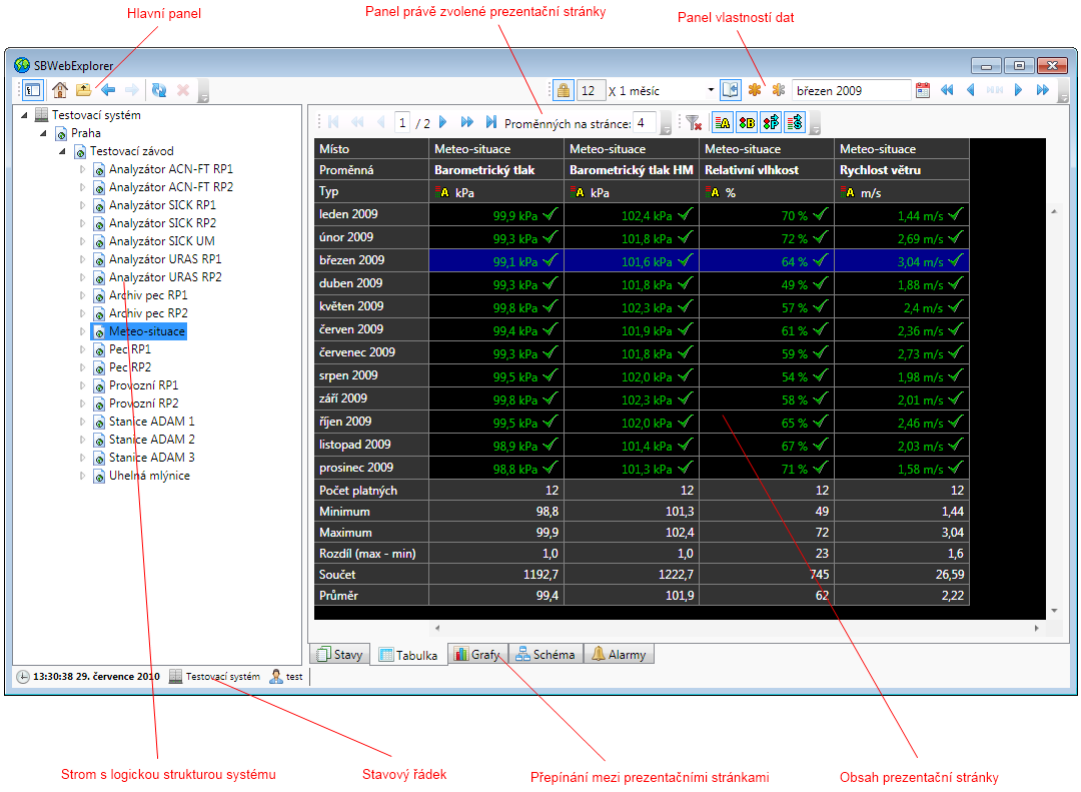


Obrázek 4.1: Úvodní okno před připojením k serveru a po připojení.

Jakmile přihlášení úspěšně proběhne, zobrazí se hlavní okno aplikace určené pro prezentaci dat.

4.1 Prezentace dat

Hlavní okno (obrázek 4.2) se skládá z několika částí, nahoře se nachází panel nástrojů, úplně dole pak stavový řádek. Asi čtvrtinu zbylé plochy zabírá strom logické struktury systému umístěný nalevo, ostatní část okna je vyplněna právě zvolenou prezentační stránkou.



Obrázek 4.2: Hlavní okno aplikace.

Strom s logickou strukturou systému umožňuje procházet uzly a proměnné a vybírat ten, jehož data mají být zobrazeny.

Hlavní panel obsahuje několik tlačítek, první tlačítko slouží k zapnutí/vypnutí dokování stromu s logickou strukturou systému. V druhé části se nacházejí tlačítka usnadňující pohyb v logické struktuře systému a v historii jejího procházení. Třetí část pak obsahuje tlačítko pro obnovení dat a tlačítko pro zrušení načítání dat, které je aktivní jen v průběhu obnovování.

Panel vlastností dat umožňuje, pokud je zámek počtu vzorků v trendu odemknutý, tento počet zadat. Dále se zde nachází volba datové úrovně, typu dat a datumu s časem. Skupina tlačítek na konci posunuje datum

o jeden vzorek nebo o počet vzorků v trendu zpět a vpřed. Neaktivní tlačítko uprostřed této skupiny je k dispozici jen, pokud je zámek počtu vzorků v trendu odemknutý a nastavuje tento počet na standardní.

Stavový řádek zobrazuje serverové datum a čas, jméno systému, ke kterému je uživatel přihlášen, jméno uživatele a ve zbylé části se objevují informace o právě probíhajících akcích či případných chybách.

Přepínání mezi prezentačními stránkami slouží k výběru požadované prezentační stránky.

Panel prezentační stránky se mění v závislosti na tom, která je právě zvolená a obvykle umožňuje ještě blíže specifikovat požadovaná data. Kromě panelu u prezentační stránky se schémata, se zde nachází i tlačítka pro pohyb po stránkách obdržených dat.

Obsah prezentační stránky závisí na právě zvolené, nachází se zde samotná prezentovaná data.

4.2 Editor schémat

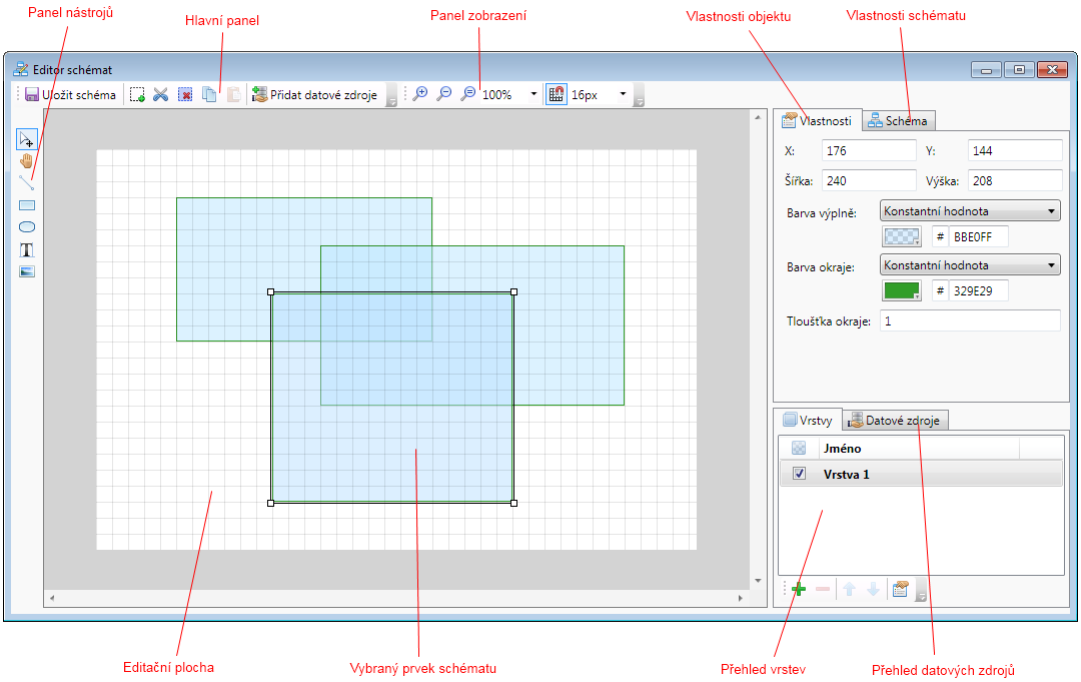
Editor schémat se spustí po kliknutí na tlačítko „editovat schéma“ v prezentační stránce se schématy. Okno editoru (obrázek 4.3) se skládá z těchto částí:

Panel nástrojů obsahuje nástroj „ukazatel“ (vybírání, přemisťování a měnění velikosti prvků schématu), nástroj „ruka“ (posun editační plochy při velkém přiblížení) a další tvořící nástroje, jejichž funkce je zřejmá z ikonek.

Hlavní panel se skládá z několika skupin tlačítek zprostředkujících nejdůležitější operace a práci se schránkou.

Panel zobrazení umožňuje měnit přiblížení editační plochy, zapnout/vypnout přichycování k mřížce a určit hustotu této mřížky.

Vlastnosti objektu zobrazují všechny vlastnosti právě vybraného prvku ve schématu. V případě, že je používán nějaký tvořící nástroj, jsou zobrazeny vlastnosti nově vytvářeného prvku.



Obrázek 4.3: Hlavní okno editoru grafických schémat.

Vlastnosti schématu se skládají z editačních polí pro zadání jména, výšky, šířky a barvy pozadí schématu.

Přehled vrstev prezentuje v řádkovém seznamu vrstvy schématu, které je možné přidávat, mazat, přeskupovat a upravovat jejich vlastnosti.

Přehled datových zdrojů je velmi podobný přehledu vrstev, jen jsou zde zobrazeny datové zdroje.

Kapitola 5

Závěr

Během vývoje aplikace nedošlo k žádným velkým problémům, nebo neřešitelným situacím, které by mohly odsoudit projekt k neúspěchu. Velkou výhodou bylo to, že ze strany zadavatele nebyl vyvíjen tlak a ani nebyly stanoveny žádné termíny dokončení. Vyvíjená aplikace totiž nebude sloužit jako hlavní prezentační aplikace na dispečincích, které mají přístup k datovému serveru prostřednictvím lokální sítě. Bude spíše doplňkovou aplikací, která bude zpříjemňovat a zjednodušovat sledování provozu vzdáleně nebo z domova.

5.1 Zhodnocení aplikace vzhledem k cílům práce

Požadavky na serverovou část aplikace byly splněny bezezbytku, aplikace je schopna spolupracovat s oběma podporovanými databázovými servery. Ani rozšíření o nový databázový server by nemělo znamenat větší problémy. Při hodnocení klientské části bude rozlišováno mezi prezentací dat, koncepcí grafických schémat a grafickým editorem schémat.

5.1.1 Prezentace dat

Všechny základní požadavky na část aplikace prezentující data byly splněny, bylo implementováno všech pět požadovaných typů prezentačních stránek. Samotné prezentační stránky se poměrně hodně přiblížily aplikaci SBExplorer, velmi však oproti této aplikaci zaostávají v oblasti konfigurovatelnosti a vícemonitorového zobrazení. Uživatel si například nemůže navolit,

jakou konkrétní sestavu grafů chce vidět, musí respektovat rozdělení podle logické struktury systému.

5.1.2 Koncepce grafických schémat

V analýze popsaná koncepce grafických schémat splňuje na ni kladené požadavky, zavedení datových zdrojů a možnosti propojení vlastností prvků schématu s datovým zdrojem výrazně zjednoduší vytváření obvyklých schémat, kterých je drtivá většina. Uživatelské funkce, kterými lze dosáhnout stoprocentní funkcionality a naprosté volnosti při dynamizaci schémat zatím implementovány nebyly. Koncepce schémat je však navržena tak, aby po rozšíření o uživatelské funkce byla zpětně kompatibilní se schématy vytvořenými v době před rozšířením.

Hlavním důvodem nezvládnutí implementace těchto funkcí je nutnost vytvořit textový editor, ve kterém půjde funkce upravovat a kompilovat, což by ještě více rozšířilo, již tak velký, objem práce. Proto byl raději kladen důraz na kvalitní implementaci datových zdrojů.

5.1.3 Editor schémat

Editor schémat umožňuje vytvářet a upravovat navrhnutá grafická schémata, přidávat datové zdroje a snadno je propojovat s vlastnostmi prvků. Z obecnějších vlastností umožňuje zoom, přichytávání k mřížce a to i u změny velikosti objektů, práci se schránkou a organizaci schématu do vrstev. Inspirace při vytváření byla brána hlavně z vektorového editoru Ipe[11], ve kterém byly vytvořeny diagramy použité v této bakalářské práci. Tomuto editoru a ani většině jiných se však editor schémat ať už svojí komplexností nebo uživatelskou přívětivostí nemůže rovnat.

5.2 Podobné aplikace

Jediná aplikace, která měla sloužit ke stejnému účelu a měla splňovat stejné požadavky se nazývala WebAccess. Byla postavena na platformě ASP.NET, takže serverová část řešila vše včetně generování uživatelského rozhraní v podobě HTML stránek. Přestože nebyly využity možnosti AJAXu, který do značné míry stírá rozdíly mezi tlustým a tenkým klientem, podařilo se aplikaci napsat tak, že se chovala podobně jako velmi ořezaný SBExplorer. Nebyly však implementovány všechny typy prezentačních stránek a možná

i z důvodu náročného vývoje uživatelského rozhraní se vývoj zastavil a projekt nikdy nebyl dokončen.

5.3 Možná rozšíření do budoucna

Mezi rozšíření nejen možná, ale i velmi pravděpodobná, patří implementace uživatelských funkcí, které jsou nezbytné pro tvorbu složitých schémat a s tím souvisí i implementace jednoduchého textového editoru, ve kterém bude možné funkce vytvářet a upravovat. Další z pravděpodoných rozšíření je podpora historie v editoru schémat (akce „zpět“ a „vpřed“).

Prezentace dat je na velmi slušné úrovni, přesto by se dala přidat lepší podpora filtrování a třídění na prezentačních stránkách se stavy a alarmy, do grafů by se dal přidat izometrický trojrozměrný vzhled grafů, na který jsou uživatelé zvyklí z SBExploreru a další podobná rozšíření. Tato cesta je ale spíše nepravděpodobná.

Koncepci a editor schémat je možné rozšiřovat neustále, hlavním faktorem nejspíše budou požadavky, které vzejdou ze zpětné vazby uživatelů. Jeden z již obdržených námětů souvisí s editorem a datovými zdroji. Po kliknutí na datový zdroj v jejich přehledu by bylo zajímavé vizuálně zvýraznit ty prvky, jejichž vlastnosti jsou s tímto zdrojem propojeny. Takto bude uživatel hned znát všechny vazby a neztratí v nich přehled.

Literatura

[1] *Wikipedia, the free encyclopedia: SCADA*
http://en.wikipedia.org/wiki/SCADA

[2] *Proteco Expert, Produkty: StoneBase*
http://www.proteco-expert.cz/web/155-stonebase.aspx

[3] *Proteco Expert, Produkty, StoneBase: Grafický prohlížeč*
http://www.proteco-expert.cz/web/166-graficky-prohlizec.aspx

[4] *MSDN: WPF XAML Browser Applications Overview*
http://msdn.microsoft.com/en-us/library/aa970060.aspx

[5] *MSDN: Introducing Windows Communication Foundation in .NET Framework 4*
http://msdn.microsoft.com/en-us/library/ee958158.aspx

[6] *MSDN: Windows Communication Foundation Security*
http://msdn.microsoft.com/en-us/library/ms732362.aspx

[7] *Codeplex: DbEntry.Net (Lephone Framework)*
http://dbentry.codeplex.com/

[8] *XAML jako deklarativní jazyk*
http://xaml.cz/wpf/xaml-jako-deklarativni-jazyk/

[9] *WPF Color Picker*
http://sachabarber.net/?p=424

[10] *Code Project: WPF Diagram Designer - Part 2*
http://www.codeproject.com/KB/WPF/WPFDiagramDesigner_Part2.aspx

[11] *The Ipe extensible drawing editor*
http://tclab.kaist.ac.kr/ipe/

Přílohy

Obsah CD

Adresář Thesis obsahuje text práce ve formátu PDF a obrázky a diagramy použité v této práci.

Adresář Install obsahuje instalační soubory a návod k instalaci testovacího systému, serverové a klientské části.

Adresář Source obsahuje zdrojové kódy serverové a klientské aplikace.

Adresář Documentation obsahuje vygenerovanou programátorskou dokumentaci.