VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

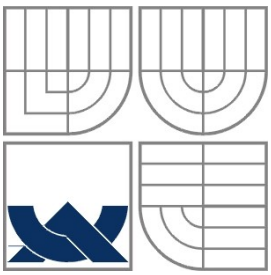
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DISTRIBUOVANÝ INFORMAČNÍ SYSTÉM MALÉ FIRMY

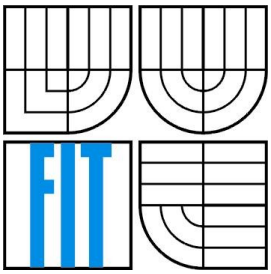
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Ondřej Pajgrt
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DISTRIBUOVANÝ INFORMAČNÍ SYSTÉM MALÉ FIRMY

DISTRIBUTED INFORMATION SYSTEM FOR A SMALL FIRM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Ondřej Pajgrt

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. Tomáš Hruška, Csc.

BRNO 2010

Abstrakt

Práce se zabývá realizací distribuovaného informačního systému malé stavební firmy od návrhu až po nasazení. Seznámíme se v ní s problematikou prostředí distribuovaných aplikací a s technologiemi, které se tohoto prostředí buď přímo týkají, nebo slouží jako podpůrné nástroje pro realizaci zadaného projektu. Dále nás pak tato práce provede kompletním návrhem a implementací výsledného produktu.

Abstract

This thesis deals with implementation of distributed information systém for a small construction engineering firm all the way from design to deployment. We will be introduced to distributed applications problems and technologies involved either throught direct relevance or as a support tool for the implementation of the project. In addition, this work will guide us throught complete design and implementation of a final product.

Klíčová slova

Distribuovaný informační systém, Middleware, RPC, ORB, Klient-Server, SQL, .NET Framework, .NET Remoting, Windows Forms, Windows Service, ClickOnce

Keywords

Distributed information system, Middleware, RPC, ORB, Client-Server, SQL, .NET Framework, .NET Remoting, Windows Forms, Windows Service, ClickOnce

Citace

Ondřej Pajgrt: Distribuovaný informační systém malé firmy, diplomová práce, Brno, FIT VUT v Brně, 2010

Distribuovaný informační systém malé firmy

Zadání

1. Seznamte se s problematikou distibuovaných informačních systémů.
2. Navrhněte distribuovaný systém pro správu faktur a zakázek malé firmy, řešení datové výměny mezi klientem a serverem a způsob zabezpečení přenosu dat.
3. Navržený systém implementujte.
4. Zhodnoťte použití systému v praxi u konkrétní firmy.
5. Diskutujte další možnosti rozvoje projektu.

Část požadovaná pro obhajobu SP: body 1 a 2.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Prof. Ing. Tomáše Hrušky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ondřej Pajgrt
15.5.2010

Poděkování

Chtěl bych poděkovat vedoucímu práce, Prof. Ing. Tomáši Hruškovi, CSc., za poskytnutí odborných rad a připomínek při řešení dané práce. Dále bych chtěl poděkovat firmě Portico s.r.o., zvláště pak Ing. Zdeňku Hýskoví za úzkou spolupráci při řešení projektu.

Obsah

1 Úvod

Tématem této práce je distribuovaný informační systém, který byl vyvíjen pro menší stavební firmu. Měl posloužit jejímu managementu pouze jako přehledný zdroj informací o finančních tocích ve firmě, nikoli jako plnohodnotný ekonomický software. Práce byly započaty v létě roku 2008, přibližně po roce byl systém zaveden do zkušebního provozu a drobné úpravy jsou na něm prováděny dodnes. Tato práce nám přiblíží veškerá úskalí takového projektu, uvede nás do problematiky distribuovaných aplikací a provede nás kompletním vývojem výsledného software od návrhu až po nasazení.

1.1 Zadavatel

Zadavatelem tohoto projektu je firma Portico s.r.o. Jedná se o menší firmu se sídlem v Brně, založenou v roce 2000, s průměrným ročním obratem kolem 30 milionů korun a s počtem 15ti zaměstnanců, jejíž hlavní směr činnosti je inženýring staveb a zajištění komplexních služeb v oblasti stavebnictví od stavebního povolení až ke kolaudaci. Těžiště působnosti je převážně orientováno na Jihomoravský kraj, realizovány však již byly i zakázky v Jihočeském, Východočeském a Moravskoslezském kraji a v Praze.

1.2 Výchozí bod

Prvotním impulsem k realizaci tohoto projektu byla nespokojenost zadavatele se stávajícím informačním systémem pro přehlednou správu zakázek a faktur, který byl implementován jako dokument Microsoft Excel za pomocí maker v jazyce Visual Basic. Systém byl již poměrně zastaralý a ve firmě nebyl nikdo, kdo by jej dokázal udržovat. Při firemním personálním růstu a postupném rozšiřování oblasti působnosti tak navíc vznikala nutnost zadávat do systému data i z více míst současně, což daný stav neumožňoval a tato skutečnost byla obcházena spoustou práce navíc (nutnost neustálého hlídání aktuálnosti dokumentu, jeho přeposílání emailem, rezervace „prázdných faktur“ a jejich pozdější vyplnění atd.). Vznikl tak požadavek na systém, který bude obsahovat funkcionalitu předchozího, popř. ji v pozdějších fázích projektu ještě rozšířit, a tato omezení mít nebude. V rámci vyhrazených finančních prostředků však na trhu chyběl systém, který by výše uvedeným požadavkům vyhovoval. Proto se zadavatel rozhodl pro software na míru.

1.3 Cíle požadovaného systému

Hlavním zaměřením požadovaného systému tedy měly být funkce, které umožňoval jeho předchůdce, konkrétně:

- komplexní správa a přehled zakázek
- správa a přehled jednotlivých faktur a ostatních daňových dokladů
- správa a přehled obchodních partnerů.
- správa a přehled uživatelů v systému a podrobné přidělování jejich pravomocí

Systém by měl dále umět vytvářet z uložených dat různé finanční přehledy a statistiky pro zvýšení transparentnosti jednotlivých zakázek. Jedním z hlavních požadavků byla centralizace dat a vzdálený přístup k nim prostřednictvím internetu, v neposlední řadě by měl systém umožňovat práci vice osob současně, ať už se jedná o operace čtecí či zapisovací.

1.4 Vize aplikace

Po zvážení požadavků na systém bylo upuštěno od možnosti modifikace, popř. rozšíření stávajícího systému a bylo rozhodnuto vytvořit zcela novou aplikaci, která bude plně vyhovovat podmínkám zadavatele a bude více otevřená konkrétním požadavkům budoucích uživatelů a případným dalším možnostem rošíření.

Z nutnosti současného přístupu více uživatelů k platným centralizovaným datům také vzešel koncept realizace systému formou distribuované aplikace s architekturou klient-server (Více o této problematice pojednává kapitola 2.)

1.5 Technické možnosti zadavatele

Jelikož požadovaný systém nebude sloužit jako plnohodnotný ekonomický software (firemní účetnictví je outsourcováno třetí stranou), ale pouze jako jakási interní správa účetnictví pro managment, nechtěl zadavatel do tohoto systému investovat vic, než je bezpodmínečně nutné. Aby se vyhnul zbytečným poplatkům za hostování aplikačního serveru, byl v brněnské kanceláři firmy přestavěn jeden z kancelářských PC na fyzický server se systémem Windows XP se vzdáleným přístupem, sloužící převážně pro potřeby vyvíjeného systému, a změněn internetový provider, který firmě poskytl rychlejší a spolehlivější připojení a veřejnou IP adresu.

Při následné implementaci systému už byl kladen důraz na využití volně dostupných programů a technologií, takže nebyly vyžadovány žádné další investice.

2 Prostředí distribuovaných systémů

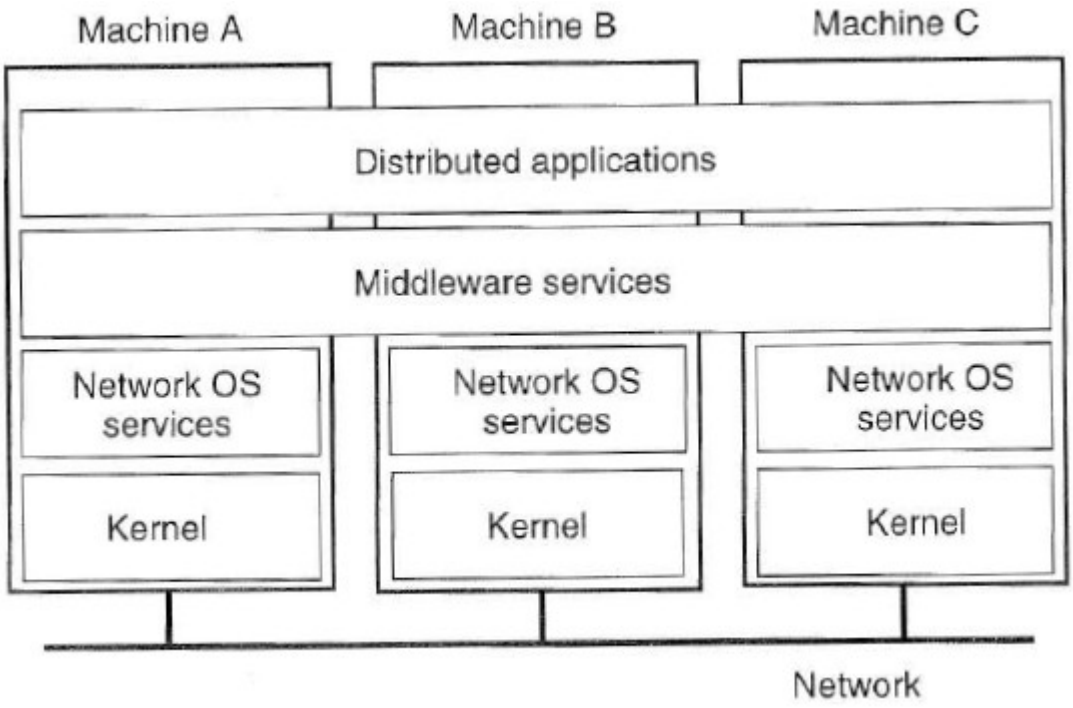
Distribuovaným systémem rozumíme soubor nezávislých počítačů, které spolu komunikují po síti a spolupracují za účelem splnění konkrétního cíle, koncovému uživateli se však jeví jako jednotný systém. To je v praxi vyžadováno např. z důvodů zrychlení či centralizace aplikace (viz literatura [1]).

Měl by splňovat tyto základní vlastnosti:

- ***Transparentnost*** – chovat a prezentovat se pro koncové uživatele a aplikace jako jednotný počítačový systém a veškerou komunikaci před nimi ukrýt.
- ***Otevřenost*** – poskytovat ostatním aplikacím přístup ke svému rozhraní a umožnit jim tak komunikaci
- ***Konkurentnost*** – umožňovat současné využití systému pro více uživatelů či procesů.
- ***Rozšiřitelnost*** – poskytovat možnost rozšíření systému, ať už na úrovni softwareové (přidání dalších modulů), nebo geografické (přesun části fyzického systému na jiné místo)

2.1 Middleware

Většina dnešních distribuovaných aplikací je implementvána pomocí tzv. Middleware. Jedná se o mezivrstvu, která zapouzdřuje rozdělení aplikace do více systémů a představuje tak jakýsi most, který tyto systémy spojuje (viz obrázek 2.1). Zaujímá funkci konverzační či překladatelskou a ve specifických případech umožňuje realizaci spolupráce komponent od různých výrobců či směřovaných na odlišné platformy.



Obrázek 2.1 – Middleware

Middleware zprostředkovává distribuované aplikaci možnost přístupu ke zdrojům na různých fyzických systémech, zodpovídá za persistenci dat mezi nimi a za bezchybnost transakcí (atomické operace). Musí také zajišťovat bezpečnost, což bývá jeden z hlavních problémů.

2.2 Problémy

Distribuované systémy tak svou podstatou nabízí mnoho rozšíření nad běžné aplikace, souvisí s nimi však také pár problémů navíc, které je potřeba řešit.

2.2.1 Synchronizace

Vzhledem k tomu, že distribuované systémy často běží na vice fyzických počítačích, objevuje se tak problém absence globálních hodin, které by byly dostupné všem účastníkům systému. Ty jsou potřeba např. pro časové značky u souborů, nebo pro některé algoritmy zajištující vzájemné vyloučení. Přesné sladění interních hodin u všech zapojených počítačů není prakticky možné z důvodu různých časovacích krystalů – problém clock-skew.

Jistým řešením by bylo zavedení nějakého centrálního časového serveru, reálně to však není vhodné jednak z důvodů latence komunikace a dále by tím vzniklo slabé místo v systému, jehož selhání by se projevilo v globálním měřítku (viz literatura 2]).

V praxi se pro eliminaci tohoto problému nakonec využívá následujících přístupů:

- ***Logické hodiny*** – nemají návaznost na reálný čas, ale řeší pořadí jednotlivých ukonů v systému (co se stane před čím).
- ***Fyzické hodiny*** – navenek konsistentní hodiny, bez zpoždění. Lze realizovat za pomocí synchronizačních algoritmů, které do výsledného času zapracovávají komunikační zpoždění.

2.2.2 Replikace a konzistence

Distribuované systémy také často využívají replikaci dat, ať už za účelem spolehlivosti (pokud jeden uzel selže, je potřeba aby byla data dostupná jinde), či zvýšení výkonu systému (přiblížení dat k místu, které je často využívá - caching). S tím vzniká problém zachování konsistence replikovaných dat (viz literatura [3]).

Existuje několik modelů správy konsistence dat, které využívá buď samotná aplikace, nebo middleware. Ty představují druh kontrakt mezi procesem a úložištěm dat, resp. soubor určitých pravidel, která musí daný proces dodržovat při přístupu k datům.

- ***Striktní konsistence*** – operace zápisu se okamžitě projeví ve všech kopiích konkrétních dat. V reálných systémech se velmi težko realizuje vzhledem ke komunikačnímu zpoždění.
- ***Sekvenční konsistence*** – pracuje s pořadím prováděných operací a následným sekvenčním provedením těchto operací jednotlivými systémy.
- ***Linearizace*** – rozšíření sekvencní konzistence, přesné pořadí provedení konkrétních operací v konkrétních systémech závislé také na čase dokončení jednotlivých procesů.
- ***Kauzální konsistence*** – přeuspořádání prováděných operací tak, aby nedošlo k porušení konsistence.

2.2.3 Vzájemné vyloučení

V situaci, kdy více částí distribuovaného systému chce využívat stejná data, je zapotřebí zajistit vzájemné vyloučení přístupu do kritické sekce. Tou může v těchto systémech být například zápis dat. Ve většině případů je souběžné čtení více procesů povoleno, konkrétní data lze tedy číst současně z vice zdrojů bez komplikací. Problém nastává v momentě, kdy je jednou ze souběžně vykonávaných operací zápis. Pokud chce jeden proces číst data, která v tu danou chvilí jiný proces zapisuje, popř. pokud chce zapisovat data, která již zapisuje jiný proces, může to vést k jejich znehodnocení a různým dalším nepředvídatelným výsledkům (viz literatura [4]).

Ohledně přístupu do kritické sekce musí v distribuovaných systémech platit určitá pravidla:

- Veškeré operace, při jejichž souběžném provedení by došlo ke znehodnocení dat, musí být v kritické sekci.

- Procesu, který žádá o vstup do kritické sekce, musí být vstup umožněn v konečném čase. Nesmí nastat deadlock či vyhladovění.
- Pokud v kritické sekci není žádný proces, tak libovolnému procesu, který zažádá, musí být umožněn okamžitý vstup.
- Každý proces stráví uvnitř kritické sekce konečný čas.

Problémem vzájemného vyloučení v disribuovaných systémech řeší množství algoritmů, které se dají podle své hlavní myšlenky rozdělit do 3 kategorií:

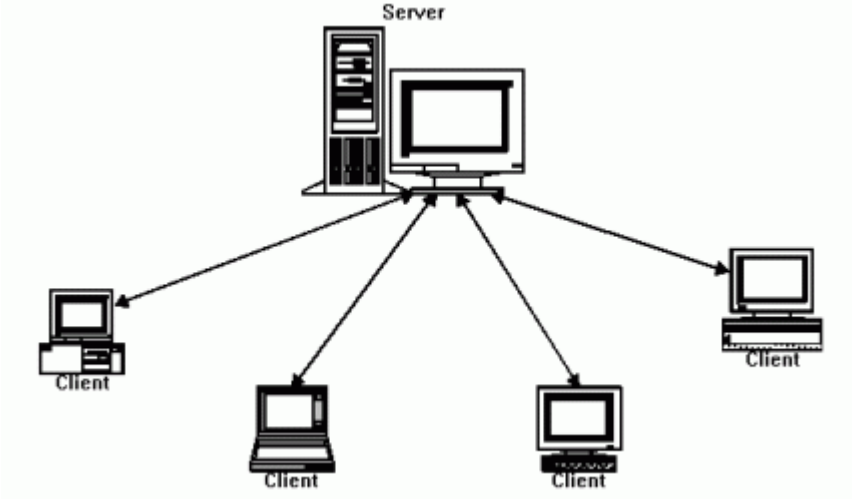
- ***Centralizované*** – Založené na koordinátorovi. Tím se stává jeden z procesů, který pak rozhoduje o přidělování kritické sekce. Tento přístup je relativně rychlý (ve většině případů stačí k rozhodnutí o přidělení kritické sekce pouze 3 zprávy) a vylučuje vyhladovění. V případě nedostupnosti koordinátora však musí být podle nějakých pravidel (většinou za využití nějakého election algoritmu) zvolen nový, aby mohl systém pokračovat v činnosti. Při větším počtu takto komunikujících procesů také vzniká u koordinátora úzké hrdlo, což může celý systém značně zpomalovat.
- ***Plně distribuované*** – Založené na časových značkách. Proces žádající o vstup do kritické sekce rozešle požadavek obsahující časovou značku všem ostatním procesům a podle přijatých časových značek se rozhodne, zda odpoví či pozdrží odpovědi na všechny došlé žádosti od ostatních procesů do doby, než on sám kritickou sekci opustí. Po obdržení všech odpovědí na svoji žádost může vstoupit do kritické sekce. Tento přístup je náchylný na výpadky procesu a ve většině případů vyžaduje jejich neustále monitorování, je tedy vhodný spíše pro menší a stabilní systémy.
- ***Algoritmy založené na předávání tokenu*** – Veškeré procesy v systému jsou seřazeny do logického kruhu, ve kterém koluje „token“, čímž rozumíme jakousi unikátní entitu, kterou v danou chvíli vlastní právě jeden proces. Aktuální vlastník tokenu má jako jediný přístup do kritické sekce bez jakýchkoli dalších omezení. Po výstupu z ní, či v případě, že o vstup do ní nemá zájem, pošle token svému následníkovi v kruhu. Tento přístup je ovšem náchylný na ztrátu tokenu a výpadek procesů.

2.3 Architektury v distribuovaných systémech

V prostředí distribuovaných systému rozlišujeme dva základní typy architektur, které se liší v pojetí komunikace: model klient-server a model peer-to-peer.

2.3.1 Obecný model Klient-Server

V dnešních distribuovaných aplikacích se jedná o nejrozšířenější model architektury. Aplikace jsou rozděleny na klienstkou část, která většinou zajišťuje interakci s uživatelem a iniciuje komunikaci, a část serverovou, která převážně spravuje data a provádí nad nimi žádané operace. Klient je tak v této architektuře aktivním prvkem, obvykle bývá připojen k malému množství serverů. Oproti tomu server pasivně naslouchá a obsluhuje příchozí dotazy od většího množství klientů (viz literatura [5]).

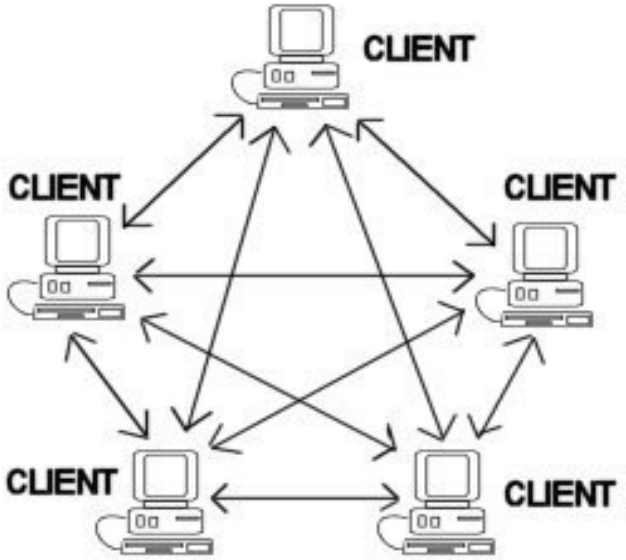


Obrázek 2.2 – Klient-Server

Tento centralizovaný přístup tak s sebou nese určité výhody z hlediska bezpečnosti a správy systému. Server má přehled o všech účastnících systému a rozhoduje o jejich autentizaci. Rovněž data jsou uložena na straně serveru, což redukuje hrozbu jejich neoprávněné modifikace klientem a značně redukuje náklady na jejich případnou modifikaci. Tyto výhody jsou ovšem kompenzovány vyššími nároky kladenými na server v případě rozšiřování sítě. Mnoho současných požadavků může daný server snadno přetížit. Také zde vniká riziko výpadku serverového uzlu, který často ohrozí celý systém.

2.3.2 Obecný model Peer-to-Peer

Modelem Peer-to-Peer (označovaný také jako Klient-Klient, či zkratkou P2P) rozumíme architekturu, která je jakýmsi opakem modelu Klient-Server. Síťová komunikace zde probíhá přímo mezi jednotlivými klienty. Ti jsou si navzájem rovnocenní a mohou zastávat úlohu jak iniciátora(Klienta), tak příjemce(Serveru) komunikace. Teoreticky se v takovém systému nevyskytuje žádný centrální prvek, v praxi se však často pro zjednodušení využívá specializovaných serverů, které slouží pro navázání komunikace a jakési počáteční „seznámení“ klientů (viz literatura [6]).

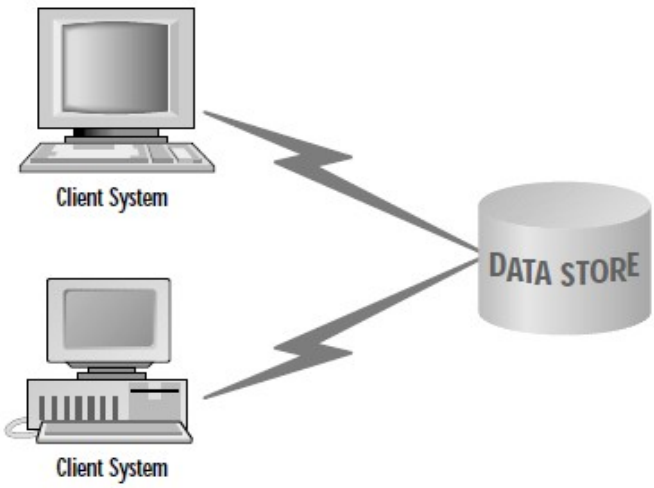


Obrázek 2.3 – Peer-to-Peer síť

Ve srovnání s modelem Klient-Server není tato architektura tolik ohrožena výpadkem uzlu. Také při rozšiřování P2P sítě se její celková přenosová rychlost zvyšuje, u architektury Klient-Server je tomu v důsledku nutnosti sdílení serverového uzlu přesně naopak.

2.3.3 Dvojvrstvá architektura Klient-Server

Jedná se o nejjednodušší verzi klient-server architektury, kdy se celý systém skládá pouze ze dvou vrstev. Ve většině případů představuje klientská část vrstvu prezentační, jejímž hlavním úkolem je interakce s uživatelem. Ta pak komunikuje se serverem, který reprezentuje vrstvu datovou, jejímž primárním cílem jsou datové operace. Typickým příkladem této architektury je tedy jakákoli databázová aplikace, kde data spravuje dedikovaný databázový server, či jiné datové úložiště, a manipulace s nimi probíhá v nějakém jiném procesu. Existují však i vyjímky (např. X Window System), ve kterých jsou tyto role prohozené (viz literatura [23]).

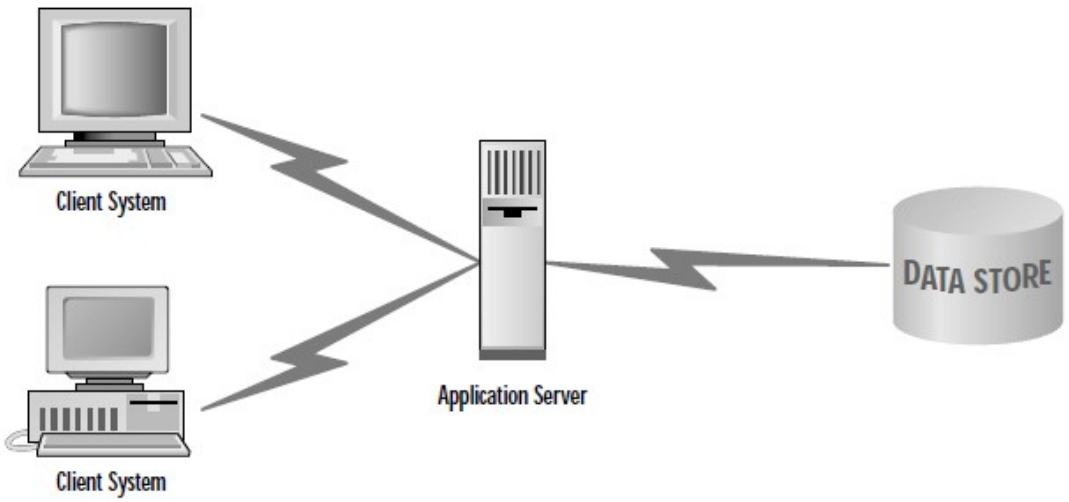


Obrázek 2.4 – Dvouvrstvá klient-server architektura

Pro jednoduchou distribuovanou aplikaci, za kterou se dá pokládat i obyčejná statická HTML stránka, je tato architektura plně dostačující. Problém ovšem nastává v momentě, kdy je nad daty vyžadována nějaká pokročilejší logika. Samotné datové úložiště touto možností disponuje pouze ve značně omezené míře a implementovat tuto logiku do klientské vrsty se v mnoha případech jeví jako neefektivní, u tenkých klientů (viz kapitola 2.3.5) to dokonce ani nemusí být možné v důsledku zabezpečení či dostupných prostředků.

2.3.4 N-vrstvá architektura Klient-Server

Přidáním další vrstvy (případně i více), jejímž hlavním úkolem je právě implementace pokročilejší aplikační logiky, vzniká v dnešní době asi nejrozšířenejší architektura – třívrstvá (v případě dalších vrstev dále označována jako N-vrstvá). Nachází se mezi datovou a prezentační a podle své podstaty se nazývá logická či business vrstva (viz literatura [23]).



Obrázek 2.5 –Trojvrstvá klient-server architektura

Zavedením této vrstvy dochází ke značnému zefektivnění systému, umožňujícím tak např. znovupoužití objektů pro různé další aplikace, či využití serverových prostředků pro předzpracování náročnějších operací. Tato architektura s sebou ovšem také nese poněkud výšší požadavky na kvalitu návrhu a následnou správu.

2.3.5 Tenký klient

Aplikace založené na klient-server architektuře můžeme také rozlišovat na základě vzájemné závislosti jejich částí a umístění aplikační logiky, resp. logické vrstvy. V případě, kdy je její drtivá většina přítomna přímo na serverové straně, klient je na ní plně závislý a sám realizuje funkce převážně prezentační, mluvíme o tzv. tenkém klientovi. Ten nachází hojné uplatnění např. na poli webových aplikací. V takovém případě se klientem stává internetový prohlížeč a jednotně tak zastává tuto funkci i pro velké množství různých aplikací.

Takový přístup nabízí mnohem větší kontrolu nad klientskými aplikacemi např. z hlediska bezpečnosti. Každé připojení k serveru představuje potencionální riziko, proto může být nevhodné svěřovat mu nadbytečné množství informací či logiky. Server, který sám sobě plně důveřuje, tak klientům předkládá data už zpracovaná, v ideálním případě bez jakékoli redundance. Také z hlediská správy verzí je tento přístup značně zjednodušen, vzhledem k centralizaci aplikace na straně serveru. Ten se tak mnohdy stává jediným místem, nad kterým je potřeba provádět případné aktualizace. To, spolu s minimální požadavky, které jsou na samotný klientský hardware kladeny, značně rozšiřuje oblast potencionálního využití aplikace.

Vedle těchto výhod však leží i zápory. Tím hlavním je riziko výpadku serveru, který představuje kritické místo v systému (Single point of failure). Bez přítomnosti serveru celý systém zkolabuje a stává se nepoužitelným (viz literatura [24]).

2.3.6 Tlustý klient

Jakýmsi opakem k tenkému klientovi je pak druh klienta označovaný jako tlustý. Ten obsahuje již větší část (mnohdy většinu) aplikační logiky. I nadále sice vyžaduje spojení se serverem, vzhledem ke schopnosti provádět množství ukonů bez jeho účasti ho však využívá v mnohem menším měřítku.

Omezené množství komunikace se také pozitivně promítne na celkovém objemu přenesených dat, diky čemuž může být tlustý klient upřednostňován v prostředí pomalejšího (např. mobilního) připojení. Obecně bývá tlustý klient robustnější a úzce spjatý s konkrétní aplikací. Umožňuje pracovat i offline, bez akutního přístupu k serveru (např. při výpadku).

V tom však také spočívá i jeho hlavní nevýhoda. Tím, že aplikační logika a mnohdy i samotná zpracovávaná data nejsou centralizována, může docházet k situaci, kdy data u různých klientů nebudou konzistentní vzájemně či s daty na serveru. V důsledku rozšíření značné váhy aplikace mezi klienty je také správa verzí značně komplikovanější, než je tomu u tenkých klientů (viz literatura [25]).

2.4 Komunikace v distribuovaných systémech

Způsob komunikace v distribuovaných aplikacích je založen převážně na transakčním zpracování dotaz/odpověď. Komunikační protokoly často tvoří další vrstvu nad hojně rozšířenými transportními protokoly, jako je TCP, HTTP atd. Liší se v závislosti na konkrétním systému, hojně se však kromě aplikačně specifických protokolů využívají i různé autentizační a autorizační protokoly.

Veškerou takovou komunikaci by měl zapouzřovat Middleware. Vzhledem k faktu, že distribuované systémy často nedisponují sdílenou pamětí, je komunikace na nižších úrovnich abstrakce realizována formou zasílání zpráv (viz literatura [7]).

- Podle typu komunikace rozlišujeme synchronní a asynchronní zasílání zpráv:
  - ***Synchronní*** – odesílatel se po odeslání zprávy zablokuje a čeká na odpověď od příjemce.
  - ***Asynchronní*** – odesílatel po odeslání nečeká na odpověď a pokračuje dále ve svém normálním běhu
- Podle doby, po kterou je přenášená zpráva uchována, rozlišujeme persistentní a transientní komunikaci

- ***Persistentní*** – zpráva je uložena do okamžiku, než si ji příjemce převezme. Využívá fronty zpráv (Message-queuing systems).
- ***Transientní*** – zpráva je zpracována pouze v okamžiku, kdy se jak odesílatel tak příjemce učastní komunikace. Může tak dojít k její ztrátě.

Způsob, kterým middleware zapouzdřuje síťovou komunikaci, je založen nejčastěji na některém ze 3 základních přístupů – Zasílání zpráv vlastním či standardním protokolem, Remote Procedure Call (RPC) a Object Request Broker (ORB).

2.4.1 Vlastní či standardní protkol

Jedná se o nejjednodušší formu komunikace, kdy mezi sebou klient a server komunikují prostřednictvím zasílání zpráv, které jsou základní jednotkou nesené informace. Používá se v případech, kdy je potřeba dosáhnout optimálního výkonu nebo kompatibility. Nevýhodou je náročná implementace a obtížná realizace, zvláště u složitějších protokolů. V oblasti vícevrstvých aplikací se používá pouze okrajově z důvodu velké závislosti na protokolu – při špatném návrhu může i malá změna způsobit katastrofální dopad na celý systém (viz literatura [8]).

Middleware, který využívá tohoto přístupu se označuje jako Message-oriented middleware. Jeho hlavní výhodou je značné zvýšení přenositelnosti a celkové flexibility systému tím, že zjednodušuje jeho nasazení a používání v heterogením prostředí a na různých platformách. Dále umožňuje tyto zprávy za běhu modifikovat či přeposílat dalším příjemcům (viz literatura [9]).

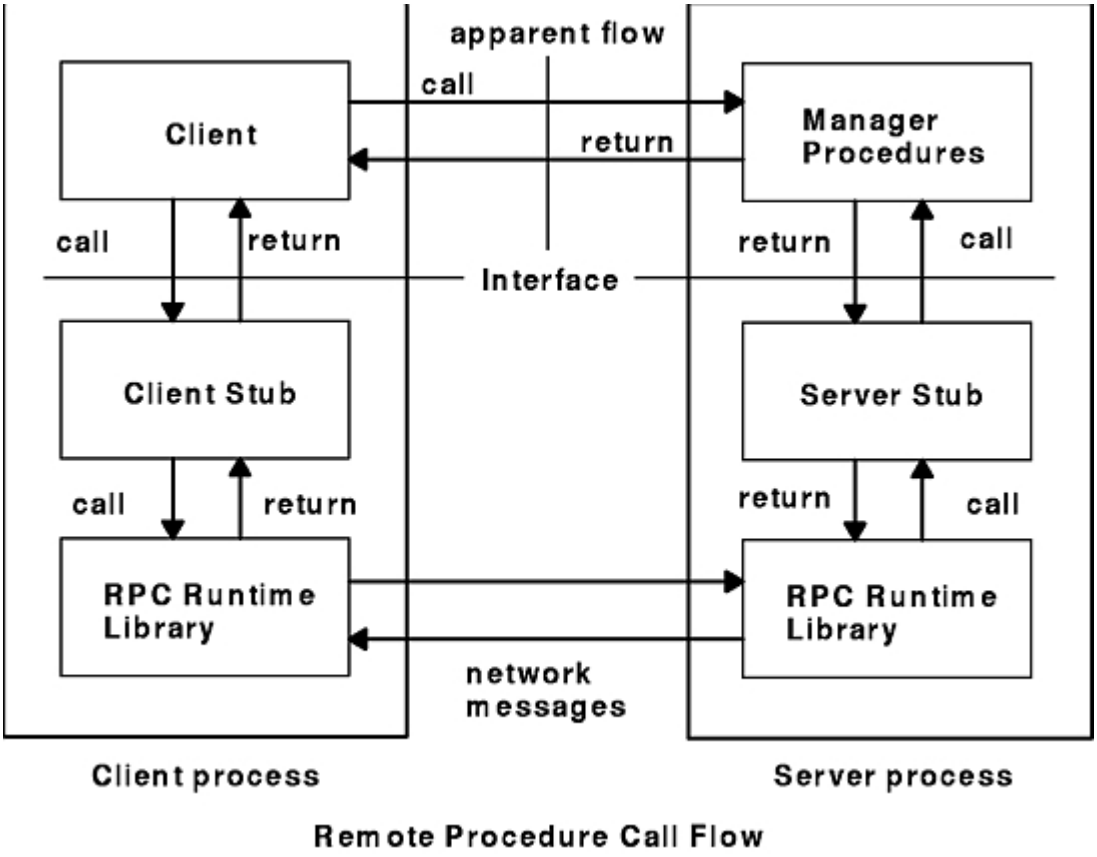
2.4.2 RPC – Remote Procedure Call

Tento způsob vzešel ze snahy o skrytí komunikace a jejího zprůhlednění navenek. Hlavní myšlenkou bylo zavolání procedury na straně klienta a její provedení na straně serveru. Vzhledem k faktu, že se jedná o dva oddělené systémy s nezávislými adresovými prostory, je nutné nejen dát serverové straně vědět, jakou funkci má zavolat, ale také jí předat veškeré parametry. K tomu se využívají tzv. „stubs“. Jedná se o rozdělení procedury mezi klientskou a serverovou část, kdy klientská verze této funkce realizuje volání a předání parametru serverové části, ta provede samotné tělo procedury a výsledek poté vrátí zpět na stranu klienta. Vzniká tak vlastně jakási simulace sdílené paměti (viz literatura [10], [11]).

Komunikace probíhá konkrétně následujícím způsobem:

1. Klient zavolá danou funkci, jakoby se jednalo o obyčejnou lokální.
2. Klientský „stub“ vygeneruje zprávu o volané funkci a zabalí do ní informace o jejích parametrech (marshaling).
3. Tuto zprávu odešle na server.

4. Serverový „stub“, nazývaný také „skeleton“, tuto zprávu rozbalí (unmarchaling) a danou funkci zavolá.
5. Výsledek je opět zabalen do zprávy a odeslán zpět klientovi.



Obrázek 2.6 – Remote procedure call

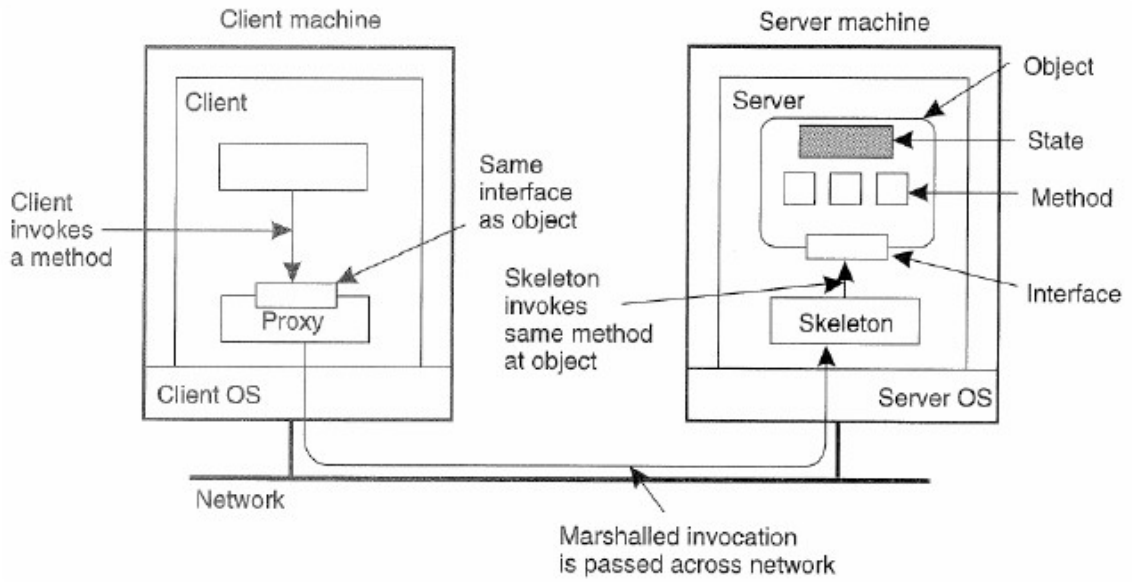
Podle způsobu předávání parametrů vzdálené proceduře rozlišujeme 3 typy volání:

- ***Call by value*** – předání parametrů hodnotou – funkci je jako parametr předána konkrétní hodnota (objekt). Jedná se pouze o kopii původních dat, proto na ně jakákoli modifikace na straně serveru nemá vliv.
- ***Call by reference*** – předání parametrů odkazem – funkci je předán pouze odkaz na konkrétní hodnotu (objekt), přístup k těmto datům dále zajišťuje RPC middleware. Jakákoli změna těchto dat na straně serveru je reflektována u klienta.
- ***Call by copy/restore*** – speciální případ Call by reference – pro konkrétní volání se vytvoří nová kopie dat. Při návratu volání přepíší pozměněná data ty původní.

2.4.3 ORB – Object Request Broker

S příchodem objektově orientovaných jazyků se začaly objevovat nedostatky RPC a vyvstala nutnost tento přístup ke vzdálené komunikaci více přiblížit objektovému paradigmatu. Z potřeby sdílení objektů a jejich metod v rámci distribuovaného systému vznikl koncept ORB. Ten silně vychází z RPC, na rozdíl od vzdáleného volání procedur však realizuje vzdálený přístup ke konkrétním objektům na straně serveru a jejich metodám (viz literatura [7]).

Podobně jako v případě RPC je daný objekt rozdělen mezi klienta a server. Klientská část, tzv. „proxy“, popisuje pouze rozhraní, serverová část pak obsahuje samotný objekt, rozšířený o tzv. „skeleton“, který zpracovává zprávy od klientského proxy a volá požadované metody.



Obrázek 2.7 – Object request broker

Proxy objekt je na straně klienta vytvořen při kompilaci, následně se s jeho pomocí už vzdáleně přistupuje k atributům objektu, či volají jeho metody, zcela totožně, jako by se jednalo o lokální objekt, a to buď staticky – object.method(int), nebo dynamicky – invoke(object, method, int). Dynamický přístup nevyžaduje při změně skeletonu opětovnou kompilaci proxy objektu.

2.5 Hrozby

Ve většině případů využívají distribuované aplikace ke komunikaci mezi jejich jednotlivými částmi nějakou síť, ať už intranet, či internet. Každou takovouto síť je nutno implicitně považovat za

potencionálně nebezpečnou a předpokládat v ní výskyt možného útočníka. Je tedy nutné tuto aplikaci důsledně zabezpečit, v prvé řadě ovšem musíme vědět před čím (viz literatura [12]).

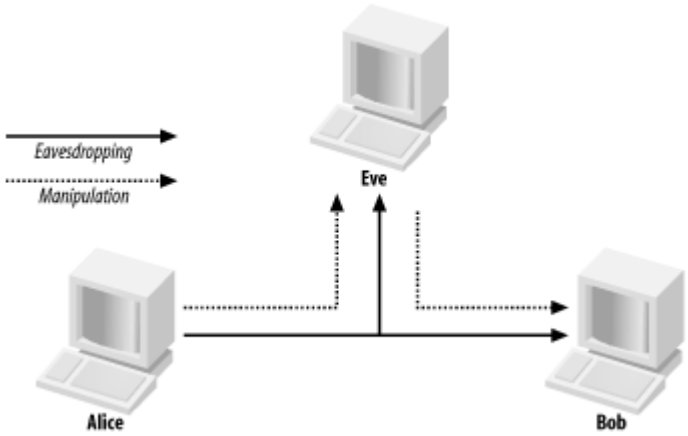
V základu můžeme rozlišit hrozby útoku na distribuované aplikace v závislosti na jejich primárním cíli do 3 kategorií:

- Únik informací
- Neoprávněná manipulace s daty
- Útok na funkčnost systému

2.5.1 Únik informací

Hlavním cílem tohoto druhu útoku je zisk nějakých důvěrných informací. Díky nutnosti síťové komunikace se aplikace stává zranitelná zejména vůči tzv. odposlouchávání (Eavesdropping). Jde o situaci, kdy útočník získá pasivní přístup k přenosovému kanálu mezi dvěma body komunikace. Získaná data pak může použit nezávisle na odposlouchaném systému, v případě, že se jedná o nějaké důvěrné informace, či k další manipulaci s ním například formou zneužití přístupových informací.

Řešením tohoto problému je zavedení šifrované komunikace za pomoci např. asynchronního šifrování (viz kapitoly 2.6.1 a 2.6.2).



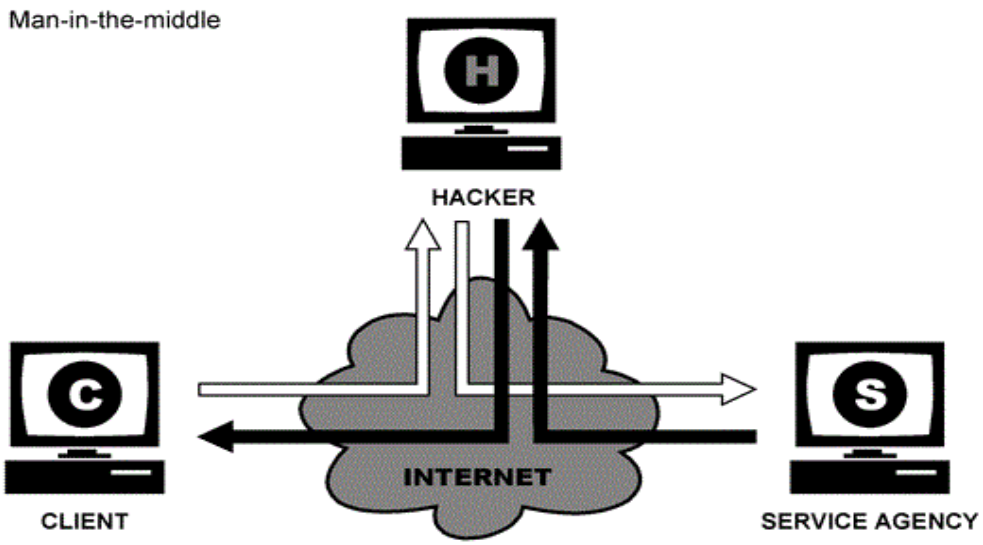
Obrázek 2.8 – Eavesdropping útok

Rovněž přístup do systému by neměl být veřejný (kromě systémů k tomu přímo určených) a v době, kdy je téměř jakákoli informace zneužitelná, zvláště v obchodní oblasti, by měl být takový přístup jednotlivým uživatelům přidělován pouze v minimální možně míře, kterou potřebují k využití tohoto systému, např. formou uživatelských účtů či přístupových práv.

2.5.2 Neoprávněná manipulace s daty

Ve většině případů však útočníkovi nestačí pouze pasivní přístup k datům, ale vyhledává též způsob, jak je modifikovat. Příkladem takového útoku je tzv. „člověk uprostřed“ (Man-in-the-middle)

(viz literatura [13]). Útočník získá aktivní přístup k přenosovému médiu tím, že celou komunikaci přesměruje tak, aby procházela přes něj a zároveň aby oba původně komunikující uzly nic nezpozorovaly. V tu chvíli získává kontrolu nad veškerými přenášenými daty.



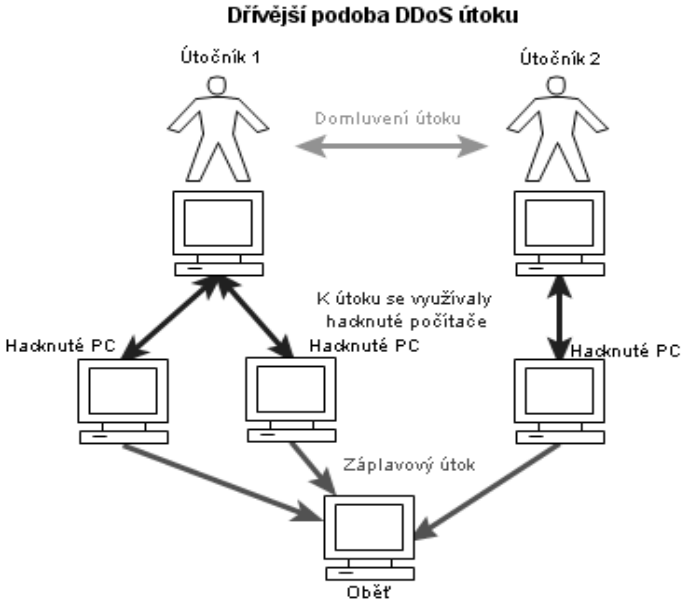
Obrázek 2.9 – Man-in-the-midle útok

Tento problém už pouhé asynchronní šifrování nevyřeší, vzhledem k možnostem útočníka zmanipulovat samotnou výměnu veřejných klíčů. Proto je nutné zavést nějakou formu elektronického podpisu např. formou certifikační autority, která jednoznačně vzájemně identifikuje komunikující strany (viz kapitola 2.6.3).

Dalším z takových útoků může být např. použití replikace. K té dochází v případě, že útočník odposlechne nějakou zprávu, uloží si ji a posléze neoprávněně použije (klidně i vicekrát). Systém tedy musí zaručit a rozpoznat čerstvost přijatých zpráv.

2.5.3 Útok na funkčnost systému

Cílem útoku nemusí být vždy data. V některých případech se jím může stát samotný systém. Hlavním představitelem takovýchto útoků bývá odmítnutí služby (DoS - Denial of Service, resp. DDoS – Distributed Denial of Service). Jedná se o případ, kdy útočník má za cíl výrazně zpomalit, či úplně zablokovat komunikaci v systému. Toho docílí např. zaplavením sítě, resp. cílového serveru náhodnými daty či dotazy, čímž zabrání protékání skutečných dat. K těmto útokům se často využívá podpůrný malware, který se v delším časovém horizontu rozšíří na větší množství pracovních stanic. V domluvený okamžik pak realizují útok současně, čímž výrazně zvýší objem dat, kterým je server v tu chvíli vystaven.



Obrázek 2.10 – Man-in-the-midle útok

Ochrana před těmito útoky závisí ve značné míře na serverové infrastruktuře, vhodné konfiguraci firewall a filtrování příchozí komunikace. Zpravidla se tak vymyká spektru aplikace.

2.6 Zabezpečení

Aplikaci je tedy třeba řádně zabezpečit proti výše zmíněným hrozbám a útočníkovi napadení systému v ideálním případě zcela znemožnit, či alespoň natolik dostatečně zkomplikovat, aby zdroje nutné k úspěšnému útoku (ať už finanční, fyzické či časové) byly natolik obrovské, že předčí jeho případný zisk (viz literatura [33]).

V distribuovaných systémech je nejvíce zranitelným místem pro případný útok síťová komunikace. Základním nedostatkem je její potencionální otevřenost. Pro případného útočníka je minimálním problémem její sledování, ať už formou napojení na některý ze síťových prvků na trase komunikace, či s využitím malware na některé z komunikujících stanic. Vedle zabezpečení datového úložiště je tak právě komunikace hlavní náplní této kapitoly.

2.6.1 Symetrické šifrování

Při běžné komunikaci, kterou je útočník schopen sledovat, mu volně vystavujeme potencionálně důvěrná data. Pokud však přenášená data zašifrujeme, značně je tím pro něj znehodnotíme.

V moderní kryptografii není pro šifrování hlavním tajemstvím samotný algoritmus, u kterého se obecně předpokládá, že je veřejně známý, ale šifrovací klíč, resp. heslo, pomocí kterého jsou data

kódována. Vztah tohoto hesla ke kodéru a dekodéru pak rozděluje dva základní způsoby kódování dat. Tou jednodušší metodou je šifrování symetrické. V takovém případě jsou data kódována i dekódována stejným tajným klíčem, který musí obě komunikující strany předem znát, popř. si je nějak bezpečně předat (viz literatura [26], [34]). Tímto druhem šifrování se zabývá celá řada algoritmů. Těmi nejrozšířenejšími jsou:

- ***3DES*** – bloková šifra se 112 bitovým klíčem, 3x aplikuje dnes již zastaralý DES.
- ***AES*** – bloková šifra, více variant délky klíče – 128, 192 nebo 256 bitů. Vychází z algoritmu Rijndael.
- ***RC4*** – zřejmě nejrozšířenější zástupce řetězových šifer, použitý např. v SSL či WEP.

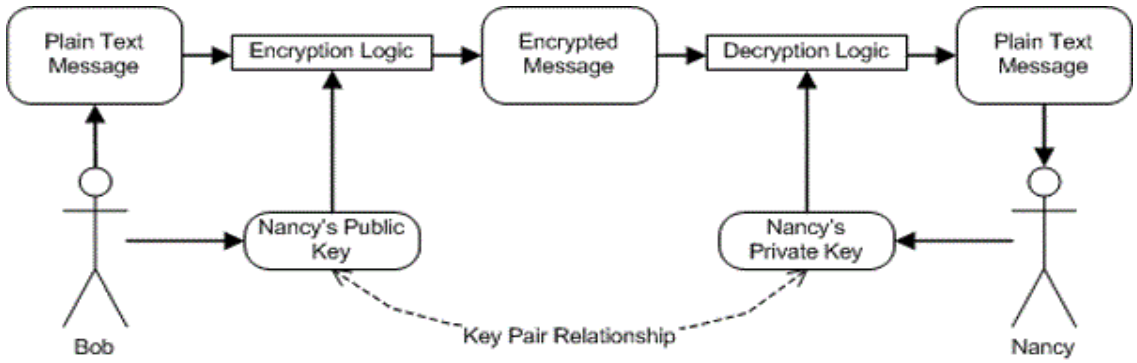
Obecně se dá říct, že jsou symetrické šifry bezpečné v případě, že je jejich klíč dostatečně dlouhý, aby odolal útoku silou (postupnému procházení stavového prostoru všech možných kombinací klíče), resp. aby takový útok trval i s využitím veškerých moderních prostředků nereálně dlouho.

2.6.2 Asymetrické šifrování

Druhou, složitější variantou je šifrování asymetrické. V takovém případě je klíč složen ze dvou matematicky provázaných částí:

- ***Veřejný klíč*** – pomocí něj lze data pouze zašifrovat.
- ***Soukromý klíč*** – slouží pouze k dešifrování.

Každý účastník komunikace má svůj pár klíčů. Soukromý uchovává jako tajemství, zatímco ten veřejný je volně k dispozici ostatním stranám. Odesílatel zašifruje zprávu veřejným klíčem adresáta, která pak může být rozšifrována pouze jeho odpovídajícím soukromým klíčem (viz literatura [27], [28], [34]).



Obrázek 2.11 – Asymetrické šifrování

Zatímco u symetrického šifrování lze jako klíč použít libovolnou bitovou kombinaci o zadané délce, u asymetrického je vyhovujících párů klíčů o poznání méně. V důsledku zachování dostatečně velkého stavového prostoru možných klíčů je jejich velikost až několikanásobně větší, než u symetrických (v dnešní době nejrozšířenější asymetrická šifra RSA s klíčem o velikosti 2048 bitů je ohledně bezpečnosti ekvivalentní symetrické šifře s klíčem o velikosti 112 bitů, viz literatura [29]). Také samotné šifrování a dešifrování trvá déle, proto se v některých případech využívá asymetrického šifrování pouze k bezpečné výměně symetrického klíče, kterým jsou pak zakódována už samotná data.

2.6.3 Digitální podpis

Speciální variantou asymetrického šifrování je digitální podpis, resp. certifikace. Rozdíl je v opačném použití klíčů, kdy odesílatel data (případně jejich část) zašifruje svým soukromým klíčem, čímž jasně prokáže, že pochází od něj. Adresát poté využije odesílatelův veřejný klíč k rozšifrování těchto dat a může mít jistotu, že nebyla cestou měněna či podvrhnuta (viz literatura [30], [34]).

2.6.4 Zabezpečení databáze

V naprosté většině IS jsou data tím nejcennějším. Je tedy nutné věnovat zvýšenou pozornost zabezpečení databáze samotné. V prvé řadě jde o její bezpečnou administraci formou omezení přístupu k samotnému databázovému souboru či správné nastavení uživatelských pravomocí. Tyto úkony jsou však většinou plně v kompetenci systémového administrátora a vymykají se spektru aplikace. Implementačně je však nutné aplikaci zabezpečit proti speciálnímu typu útoku s názvem SQL Injection. Jedná se o velmi častý a populární typ útoku, proti kterému je relativně velké množství aplikací, převážně těch webových, nedostatečně chráněno. Využívá nekvalitně ošetřených vstupů v aplikaci, jako jsou např. různá textová pole pro přihlašovací údaje, za účelem vložení a následného vykonání vlastního, pozměněného SQL příkazu. Tímto způsobem lze tak např. obejít autentizaci, či zcela ovládnout databázi. Ochranou proti takovému útoku je buď důsledné ošetření vstupů pomocí escapování (označování některých znaků speciální značkou) všech potencionálně nebezpečných znaků, či využití parametrizovaných dotazů na databázi (viz literatura [31]).

2.6.5 Hash

Se zabezpečením databáze souvisí také forma ukládání uživatelských hesel. Tato hesla by se nikde neměla vyskytovat v čisté podobě, odkud by je bylo možné např. s využitím některých dalších útoků přečíst. Využívá se zde hashovacích algoritmů, které vytváří z libovolně dlouhého vstupního řetězce jeho hash (označovaný také jako otisk) pevné délky, ze kterého je prakticky nemožné zkonstruovat

zpět původní řetězec. Drobná změna ve vstupním řetězci znamená velkou změnu výsledného otisku, zároveň je však velmi malá pravděpodobnost, že by dva různé vstupy měly stejný otisk. Tuto pravděpodobnost však vzhledem k faktu, že množina možných vstupních dat je větší než množina kombinací výsledného otisku, nelze zcela vyloučit (viz literatura [32], [33]).

Každé nové heslo vkládané do systému při registraci je tedy zpracováno jednocestnou hashovací funkcí a teprve její výsledek je uložen do databáze. Při následné autentizaci se pak heslo zadané uživatelem zpracuje stejnou funkcí a porovná s uloženým výsledkem. Při úspěšném prolomení databáze tak útočník získá pouze hash, nikoli plnohodnotné heslo.

Při běžném hashování nějakým standardním obecně využívaným algoritmem (např. MD5, SHA-2) odpovídají stejné vstupní řetězce vždy stejným otiskům. Existují však velice rozsáhlé databáze (tzv. „rainbow“ tabulky) obsahující velké množství standardně hashovaných řetezců. K dovršení míry bezpečnosti je proto vhodné vstupní data při hashování „prosít“ - připojit k nim určitý řetězec (nazvaný „salt“), či je s jeho pomocí nějakým jiným způsobem zmodifikovat a znemožnít tak využití těchto tabulek.

3 Technologie

V situaci, kdy již známe cíle systému a jsme seznámeni s prostředím distribuovaných aplikací, můžeme přistoupit k volbě technologií, které použijeme. Pozornost bude soutředěna mimo volby cílové platformy a implementačního jazyka převážně na výběr vhodné technologie pro část databázovou, serverovou i klientskou.

Tato kapitola si neklade za cíl představení veškerých technologií, které je možné pro daný projekt použít, ale pouze bližší seznámení s těmi, se kterými se v rámci aplikace opravdu experimentovalo, či se o nich alespoň reálně uvažovalo.

3.1 Implementační prostředí

Za cílovou platformu byl vzhledem k povaze firemní aplikace vybrán systém MS Windows. K implementaci byl z čistě subjektivních důvodů zvolen jazyk C# a platforma .NET Framework v3.5, převážně v důsledku mých předchozích zkušeností s ní a kvůli možnostem, které poskytuje IDE MS Visual Studio 2008.

3.2 Databáze

Vzhledem k datovému zaměření aplikace vyvstala nutnost využití nějakého databázového systému. Při jeho výběru byl kladen důraz převážně na možnost interakce s jazykem C# a celkové možnosti. Z těchto požadavků tak vzešli dva hlavní kandidáti: Oracle a MS SQL (viz literatura [14]).

3.2.1 Oracle

První z nich, Oracle, představuje velice robustní systém řízení báze dat (DBMS – Database management system), který podporuje nejen standartní dotazovací jazyk T-SQL, ale také pokročilé objektové databáze a vlastní imperativní programovací jazyk PL/SQL, rozšiřující možnosti vlastního SQL. Je multiplatformní s podporou všech známých serverových platforem. Obecně se ve srovnání s ostatními databázovými systémy řadí spíše mezi dražší a vyžaduje pokročilejší znalosti pro správnou konfiguraci. Jazyk PL/SQL také poskytuje více možností, než standartní T-SQL využívaný v jiných databázích.

3.2.2 MS SQL

Druhý jmenovaný je variantou databázového serveru od firmy Microsoft. Pracuje se standartním dotazovacím jazykem T-SQL a ANSI SQL. Ve srovnání se systémem Oracle je levnější na pořízení a

je poměrně jednoduchý na instalaci a údržbu. Navíc poskytuje mezi množstvím různých edic i verzi MS SQL Express. Jedná se o volně šiřitelnou edici databázového serveru, na kterou se vztahují jistá omezení výkonu avšak počet interních databází či uživatelů není omezen. Navíc umožňuje udržování kompletní databáze ve formě souboru s libovolným umístěním.

3.3 Server a datová část

Z požadavku současného přístupu a nutnosti centralizace dat je také nutný určitý spojovací uzel celé aplikace ve formě nějakého aplikačního serveru. Nejedná se o serverovou část výsledného produktu v pravém slova smyslu, nýbrž o část software, která je zodpovědná za přístup k databázi a provádí základní operace nad daty.

3.3.1 Databázový server

Tato varianta vychází z předpokladu, že serverová část aplikace bude tvořena pouze databázovým serverem, který bude zapouzdřovat většinu aplikační logiky ve formě vložených procedur, a veškerou komunikaci s databází tak bude zajišťovat už samotný klient. Takový přístup však nabízí velice omezený prostor pro pokročilou manipulaci s daty na straně serveru a neposkytuje možnosti zabezpečení v takové míře jako varianty ostatní.

3.3.2 Web Service

Další možností bylo implementovat serverovou část aplikace jako webovou službu, hostovanou ve webovém serveru. Ta s jeho pomocí vystaví svoje API potencionálním klientům a skrz protokoly HTTP(S) a SOAP (viz kapitola 3.5.1) s nimi komunikuje. Díky textové reprezentaci veškerých přenášených dat, standardizovanému popisu služby a obecně volné vazbě mezi službou a klienty je tak ideálním řešením v případě, kdy je vyžadována spolupráce různých platforem či frameworků, ze stejného důvodu je však také vyšší objem přenášených dat (viz literatura [15]).

3.3.3 Windows Aplikace

Serverovou část je také možné implementovat jako klasickou aplikaci windows, ať už formou konzole, nebo prostřednictvím windows forms. Tento přístup nevyžaduje žádnou pokročilou instalaci, nabízí jednoduchou správu (zapnutí a vypnutí aplikace) a umožňuje značný komfort při ladění. Pro praktické nasazení na serverovém stroji však není příliš vhodný vzhledem k tomu, že vyžaduje manuální (popř. pomocí startup nabídky) spuštění pod nějakým uživatelským účtem.

3.3.4 Windows Service

Nevýhody takové klasické aplikace řeší windows service. Ta umožňuje spuštění programu, který reprezentuje, v některém ze systémových účtů (nejčastěji SYSTEM, LOCAL SERVICE, NETWORK SERVICE) ihned po nabootování počítače. Omezuje se tak interakce programu s uživatelem, což ale v případě serverové aplikace není požadováno, může být dokonce i nežádoucí. Tento způsob běhu programu však značně komplikuje možnosti ladění aplikace. Navíc každá verze takového programu musí být znovu řádně nainstalována, což vývoj software zbytečně zpomaluje.

3.4 Klient a uživatelské rozhranní

Označení „klient“ v tomto případě znamená tu část výsledné aplikace, která zprostředkovává interakci s koncovým uživatelem. .NET Framework nabízí dva základní způsoby řešení uživatelského rozhranní (UI) systému a to formou buď webové aplikace (Web Forms) spustitelné v internetovém prohlížeči, či formou nativní windows aplikace (Windows Forms).

3.4.1 Web Forms

Webové aplikace postavené na technologii ASP.NET využívají k realizaci uživatelského rozhranní prvky Web Forms. Aplikace ke svému provozu vyžaduje webový server IIS. UI je vytvořeno na straně serveru ve formě jazyka HTML a zobrazeno u klienta v obyčejném webovém prohlížeči. Ten však představuje pomyslné hranice možností uživatelského rozhranní vysledné aplikace, které jsou tak limitovány na základní HTML prvky s omezenými vizuálními možnostmi. Také pokročilé operace s daty na klientské straně jsou relativně komplikované, protože většinou vyžadují další komunikaci se serverem.

Tyto nevýhody však kompenzuje dostupnost takového klienta. K jeho spuštění potřebuje uživatel většinou pouze libovolný webový prohlížeč a URL aplikace, což umožňuje přístup téměř odkudkoli. Rovněž také odpadá problém aktuálnosti. V momentě zaktualizování na straně serveru je nová verze aplikace ihned dostupná všem připojeným uživatelům.

3.4.2 Windows Forms

Aplikace s uživatelským rozhraním realizovaným pomocí Windows Forms vychází z WindowsAPI (Win32 API), což jí otevírá mnohem větší škálu možností, než které nabízí aplikace využívající Web Forms. Tento přístup také nabízí širší prostor pro aplikační logiku na klientské straně aplikace, což může v případě nutnosti zmírnit vytížení serveru.

Jedná se o samostatně spustitelný program, který ke svému běhu vyžaduje přítomnost platformy .NET na cílovém počítači, na serverové straně pak není podmíněn žádnou další technologií. V praxi však často vyžaduje instalaci na cílový počítač uživatele a pravidelnou kontrolu a obsluhu aktualizací, což ho dělá náročnějším na správu.

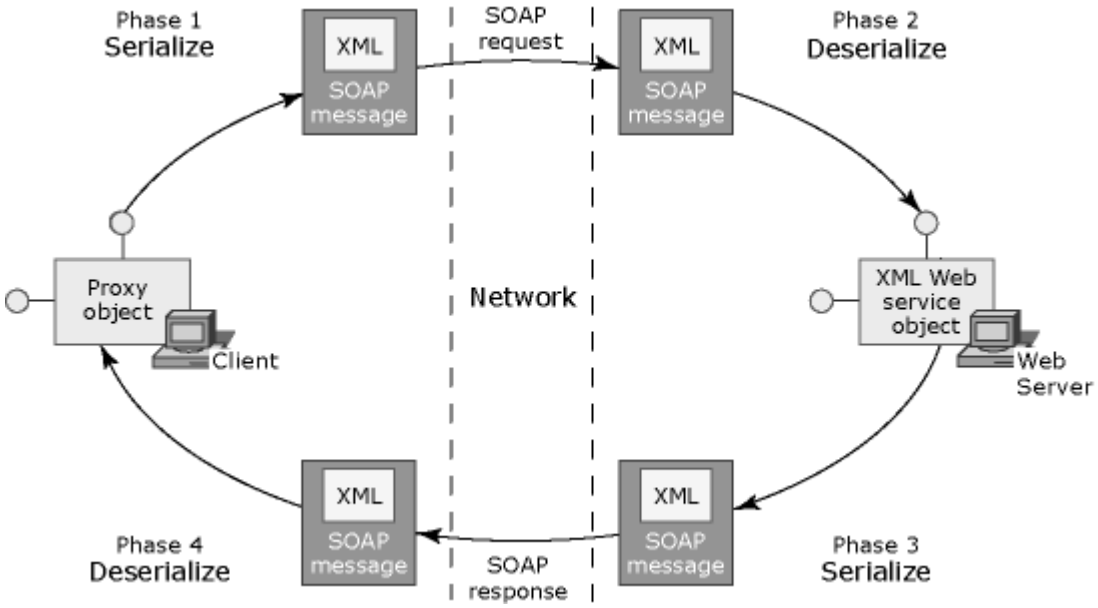
3.5 Komunikace

Dalším problémem, který bylo třeba řešit vhodnou technologií je komunikační propojení serverové a klientské části. Už v úvodu bylo upuštěno od nízkoúrovňové komunikace přes sockety a pozornost byla směřována spíše na možnost využití nějakého existujícího API .

3.5.1 ASP.NET Web Service

Komunikace založená na principu webové služby využívá volné vazby mezi komunikujícími stranami. Jako server zde působí samotná služba běžící pod webovým serverem IIS. Klienti, často označování také jako konzumenti, pak nepotřebují o dané službě vědět nic ohledně cílové platformy, objektovému modelu či programovacím jazyce, ve kterém je implementována. Jediné, co potřebují, je její URL a popis. Ten služba nabízí ve formě WSDL (Web Service Description Language) souboru, který představuje standardizovaný popis jejího API ve formátu XML.

Samotná komunikace probíhá mapováním přenášených zpráv na konkrétní koncové metody služby pomocí protokolu SOAP (Simple Object Access Protocol), který využívá standardní webové technologie, konkrétně XML pro popis a HTTP pro transport. Obecná implementace webové služby ASP.NET je také velmi jednoduchá proti ostatním metodám (viz literatura [15]).



Obrázek 3.1 – Komunikace přes SOAP

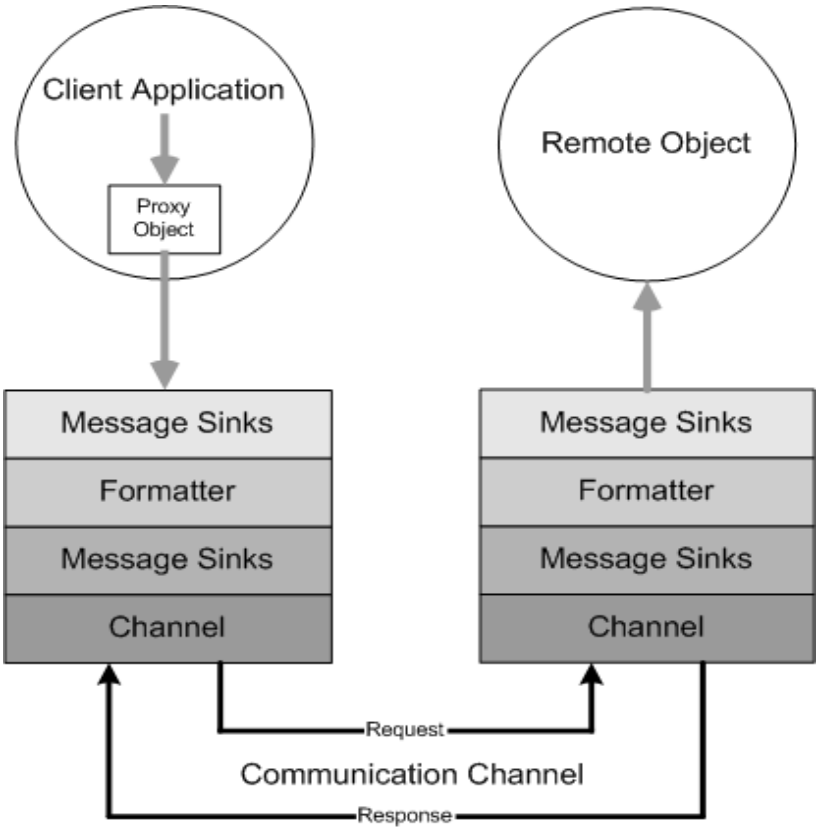
Komunikace pomocí webových služeb díky otevřenosti svého rozhraní umožňuje jednoduché propojení a spolupráci všech účastníků bez vzájemné nutné znalosti bližších implementačních detailů a to i skrze firewall. Platí za to však několika nevýhodami. Tou hlavní je bezesporu rychlost - komunikace webových služeb je omezena pouze na HTTP(S) kanál, který není tak efektivní jako TCP přenos. Reprezentace přenášených dat v textové formě XML navíc výrazně zvyšuje jejich objem. Webové služby jsou také ze své podstaty bezstavové. To může v některých případech představovat komplikace a nutnost zavedení možnosti uchovávání stavů nějakou alternativní cestou. Dalším omezením je možnost předávání parametrů volaných funkcí pouze hodnotou.

Ohledně bezpečnosti je ASP.NET Web Service odkázána na nastavení webového serveru IIS, kterému je podřízena.

3.5.2 .NET Remoting

.NET Remoting je implementaci ORB a představuje API pro meziprocesní komunikaci. Jeho hlavní funkcí je zpřístupnění vzdáleného objektu za hranicí procesu, domény či dokonce jiného počítače v síti pomocí jeho proxy. Ten vystavuje na straně klienta stejné rozhraní jako vzdálený objekt a realizuje komunikaci s ním skrz kanál, a to buď formou HTTP nebo jako TCP spojení. Data jsou přenášena v binární podobě či protokolem SOAP, který je ale o poznání pomalejší.

.NET Remoting se skládá z několika vrstev, které zodpovídají za postupné vytvoření komunikační zprávy z volání metody proxy objektu na straně klienta, její zpracování a zavolání konkrétní metody na straně serveru a předání odpovědi zpět. Mezi ně lze poměrně snadno vložit nové, uživatelem definované formátovací vrstvy (formattery), což mu poskytuje obrovský prostor pro přizpůsobení komunikace konkrétním požadavkům (viz literatura [16]).



Obrázek 3.2 – Komunikace v .NET Remoting

Úzká komunikační vazba mezi objektem a jeho proxy dovoluje realizovat i komplexnější požadavky na správu vzdálených objektů, jako je např. předávání parametrů jak hodnotou, tak referencí, či reakce na události. Remoting také nabízí kontrolu nad životním cyklem vzdáleného objektu. Ten může být realizován jako:

- ***SingleCall*** – bezstavový, stejně jako webová služba. Nový objekt je vytvořen pro obsluhu každého volání.
- ***Singleton*** – stavový. Veškerá volání všech klientů obsluhuje jediná instance objektu.
- ***Client-activated*** – hybridní. Jedna instance objektu obsluhuje veškerá volání jednoho klienta.

Zabezpečení .NET Remoting závisí na volbě používané komunikace. Při použití HTTP kanálu podporuje autentizaci pouze v případě, že je vzdálený objekt hostován ve webovém serveru IIS, Šifrování dat je pak možné pouze v případě, že je tento server nakonfigurován pro SSL. TCP kanál podporuje autentizaci i šifrování s omezením na IWA (Integrated Windows Authentication). Implementací bezpečnostní vrstvy a jejím přídáním do řetězu komunikace lze však relativně snadno docílit pokročilejšího zabezpečení.

3.5.3 WCF – Windows Communication Foundation

WCF představuje jakousi novou generaci API pro tvorbu aplikací orientovaných na služby (SOA). Sjednocuje v sobě možnosti SOAP komunikace, kterou nabízejí webové služby, optimalizovaného binárního spojení, jaké realizuje Remoting, a dalších komunikačních technologií (COM+, WSE, MSMQ). Vzniká tak jednotné robustní API, které si odnáší to lepší ze všech těchto přístupů a zároveň se vyhýbá některým jejich chybám. To umožňuje uživateli realizovat téměř libovolný komunikační problém lépe a efektivněji (viz literatura [17]).

Způsob propojení WCF serveru a klienta vychází z velké části z webových služeb. Server je v tomto případě služba, definovaná tzv. koncovým bodem (End Point). Ten se skládá ze 3 částí, často označovaných ABC:

- ***Adresy (Address)*** – definuje KDE se služba nachází. Určuje konkrétní adresu a transportní protokol.
- ***Vazby (Binding)*** – definuje JAK se služba bude používat. Určuje komunikační protokol, kódování či jaké zabezpečení bude použito.
- ***Kontraktu (Contract)*** – definuje CO služba umožňuje. Vystavuje metody služby, často impelmentován jako rozhranní (lze ovšem použít i třídu).

Přistupující klient tak potřebuje být pouze seznámen s koncovým bodem dané služby a nepotřebuje vědět žádné další implementační detaily.

Při srovnání s výše zmíněnými technologiemi musíme brát v úvahu, že WCF je z nich nejnovější. To se projevilo v prvé řadě na obecné rychlosti, která je o přibližně 25% vyšší než u .NET Remoting, v případě ASP.NET Web Services je uváděn dokonce 50% nárůst.

3.6 Distribuce

V neposlední řadě bylo nutné zvážit také technologii distribuce výsledného produktu. Zvláště u klientské části aplikace je nutná co nejjednodušší instalace a údržba, včetně správy nových verzí, aby tyto úkony mohly být prováděny běžnými uživateli, nejlépe bez nutnosti zásahu administrátorů.

3.6.1 Windows installer

Jednoduchou formu distribuce aplikace poskytuje Microsoft ve formě Windows Installeru. Ten umožňuje uživatelsky přívětivou správu aplikace od instalace, přes údržbu až po odinstalování. Za pomocí Visual Studia je tak možné vytvořit .msi balík, který provede uživatele kompletní instalací, umožní vytvořit různé zástupce na ploše, či zapsat potřebná data do registru. Důležitou vlastností je

také možnost vrácení celé instalace (Rollback) v případě neúspěchu či nějaké nepředvídané události (viz literatura [18]).

3.6.2 ClickOnce

Microsoft také nabízí technologii ClickOnce pro distribuci Windows aplikací (viz literatura [19]). Ta je zaměřena na absolutní zjednodušení správy konkrétního software a uživateli umožňuje kompletní instalaci a spuštění aplikace pouhým kliknutím na její webový odkaz. Řeší přitom 3 další problémy při nasazení aplikací:

- ***Správu nových verzí*** – aplikace nasazené pomocí technologie ClickOnce mají schopnost automatického zjištění, stáhnutí a nahrazení starých souborů novými. Konkrétní chování v případě updatu software je ve velké míře konfigurovatelné.
- ***Dopad software na systém uživatele*** – aplikace je do systému nainstalována pro konkrétního přihlášeného uživatele. Každá taková aplikace je tedy od ostatních izolovaná.
- ***Nutnost administrátorských práv při instalaci*** – aplikace nasazená pomocí ClickOnce je instalována do aplikačních dat daného uživatele, proto nevyžaduje administrátorská přístupová práva. Cílovou cestu instalace však kvůli tomu nelze níkterak změnit.

3.7 Zvolené technologie

Po prozkoumání a zvážení možností, které v případě tohoto projektu připadaly v úvahu, byly pro výsledný software vybrány tyto technologie:

Databáze je realizována, vzhledem k finanční stránce projektu, jako MS SQL databázový soubor. Ten poskytuje pro požadavky této aplikace postačující výkon a umožňuje její relativně zjednodušené zálohování a obnovu ve formě kopírování samotného souboru.

V případě serverové části aplikace je vzhledem k rychlosti komunikace, konkrétnímu spojení server-klient, které nevyžaduje otevřenost, a absenci IIS na serverovém PC u zadavatele zvolena implementace jak formou Windows Service pro finální nasazení, tak i jako nativní konzolová windows aplikace, která je využita pro vývoj a ladění.

Klientská část software je implementována jako Windows Forms aplikace, a to hlavně z důvodů omezení nutné komunikace na minimum, které umožňuje tlustý klient, zjednodušení vytváření tiskových sestav faktur a obecně uživatelsky přívětivější GUI oproti webové aplikaci.

Komunikace je realizována technologií .NET Remoting převážně z důvodu rychlosti a jednostraného zaměření aplikace, čímž z výběru vyřadila ASP.NET Web Service. V průběhu prací na projektu byla komunikace řešena také pomocí WCF, avšak zde chyběla potřebná dokumentace pro rozšíření a implementaci asymetrického šifrování přenosu dat.

Otázka distribuce je rozdělena na dvě části. Aplikační server je nasazen pomoci Windows Installeru, vzhledem k nutnosti administrátorských práv a pokročilejší konfigurace. Také je zapotřebí spolu se samotnou serverovou službou doinstalovat administrační software pro správu, což tento přístup podporuje. Klientská aplikace je pak nasazena technologií ClickOnce, což umožňuje i běžnému uživateli plnou práci se systémem bez dalších nutných práv a znalostí.

4 Návrh

Po zvážení všech aspektů projektu a teoretické přípravě, věnované technologiím, bylo možné přejít na samotný návrh aplikace. Ten probíhal v úzké spolupráci se zadavatelem za účelem maximálního naplnění požadavků. Tvorba produktu probíhala převážně formou iterativního životního cyklu software, ovšem v některých případech se spíše blížila agilnímu vývoji.

4.1 Neformální specifikace

Zákazník potřebuje nahradit starý informační systém pro správu zakázek, pohledávek a faktur novým, který bude lépe a aktuálněji splňovat současné požadavky firmy.

Hlavním požadavkem na systém je jeho distribuované provedení, které na rozdíl od jeho předchůdce umožní přístup k aktuálním datům, současnou práci více uživatelů a vystavování nových daňových dokladů „v terénu“. Tato realizace v sobě proto nepřímo nese i nutnost zabezpečeného přenosu a přístupu k informacím formou šifrované komunikace a uživatelských účtů. Dalším hlavním požadavkem byla případná možnost využití systému i některým obchodním partnerem. Proto je kladen důraz na maximální variabilitu uživatelských pravomocí a možnost libovolnému uživateli nastavit konkrétní kombinaci přístupových práv.

Systém by měl umožňovat evidenci zakázek firmy v celé jejich šíři, včetně správy investorů, detailních nákladů i příjmů v podobě přijatých a vystavených faktur, či jiných daňových dokladů (např. obyčejných paragonů). Každý daňový doklad může obsahovat různé množství jednotlivých položek, které se mohou vztahovat k různým zakázkám, faktury musí spolu s dalšími doplňujícími informacemi umožňovat uchovávat data vystavení, plnění, splatnosti a doručení. Všechny tyto informace by měl zobrazovat v přehledném seznamu realizovaném např. tabulkou a umožňovat jejich detailní editaci a přehledy formou různých filtrů a barevného rozlišení zaplacených a nezaplacených faktur.

Systém by měl dále umožňovat zadané faktury tisknout či ukládat do přenositelného souboru (např .pdf nebo .xls), spravovat platby a částečné úhrady jednotlivých faktur formou jejich platební historie a vystavovat „platební příkaz“ – seznam faktur k placení včetně čísla účtu dodavatele, resp. investora a cílové častky – pro osobu, která tyto faktury finančně realizuje.

V neposlední řadě by měl být systém schopen zobrazovat detailní přehledy o zakázkách, kde bude patrná jejich finanční bilance v jednotlivých měsících včetně podrobného seznamu nákladů a příjmů, a jakýsi celkový přehled finanční stránky všech zakázek v aktuálním roce.

Implementace bude provedena formou klientské a serverové aplikace. Důraz je kladen hlavně na minimální objem přenášených dat z důvodu častého připojení přes pomalejší mobilní internet.

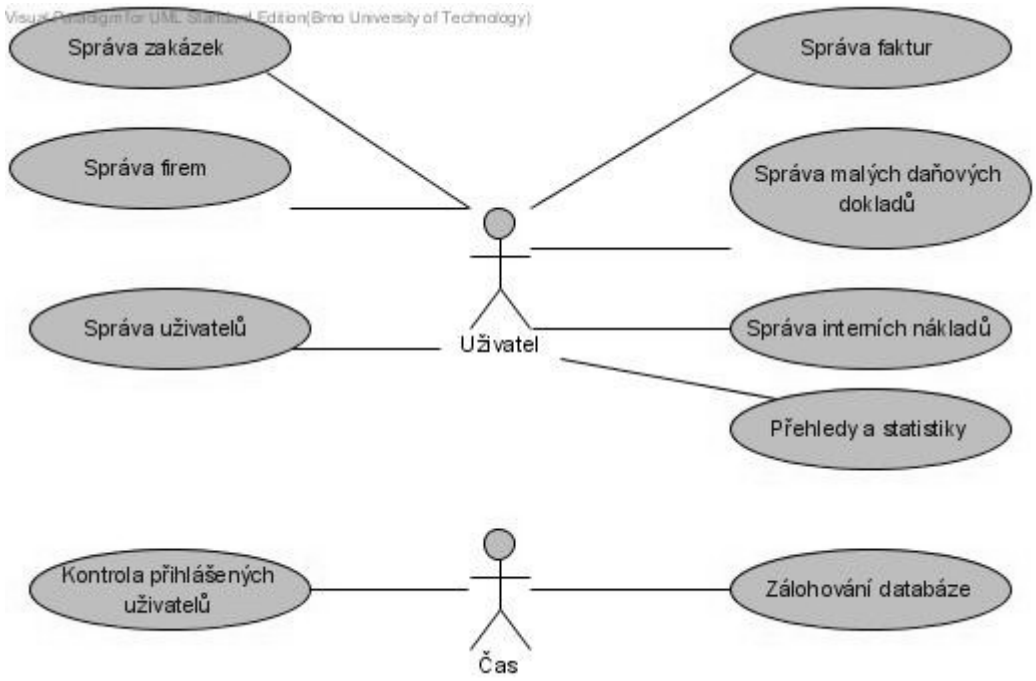
Server bude pracovat nad databází, zajišťovat její aktuálnost a provádět zálohování, řídit veškerý přístup k datům pro jednotlivé připojené klienty a provádět datově náročné operace (zejména sestavovat výše zmíněné přehledy). Klient komunikuje se serverem a umožňuje stažení dat, lokální změnu a jejich opětovné uložení na server.

4.2 Analýza požadavků

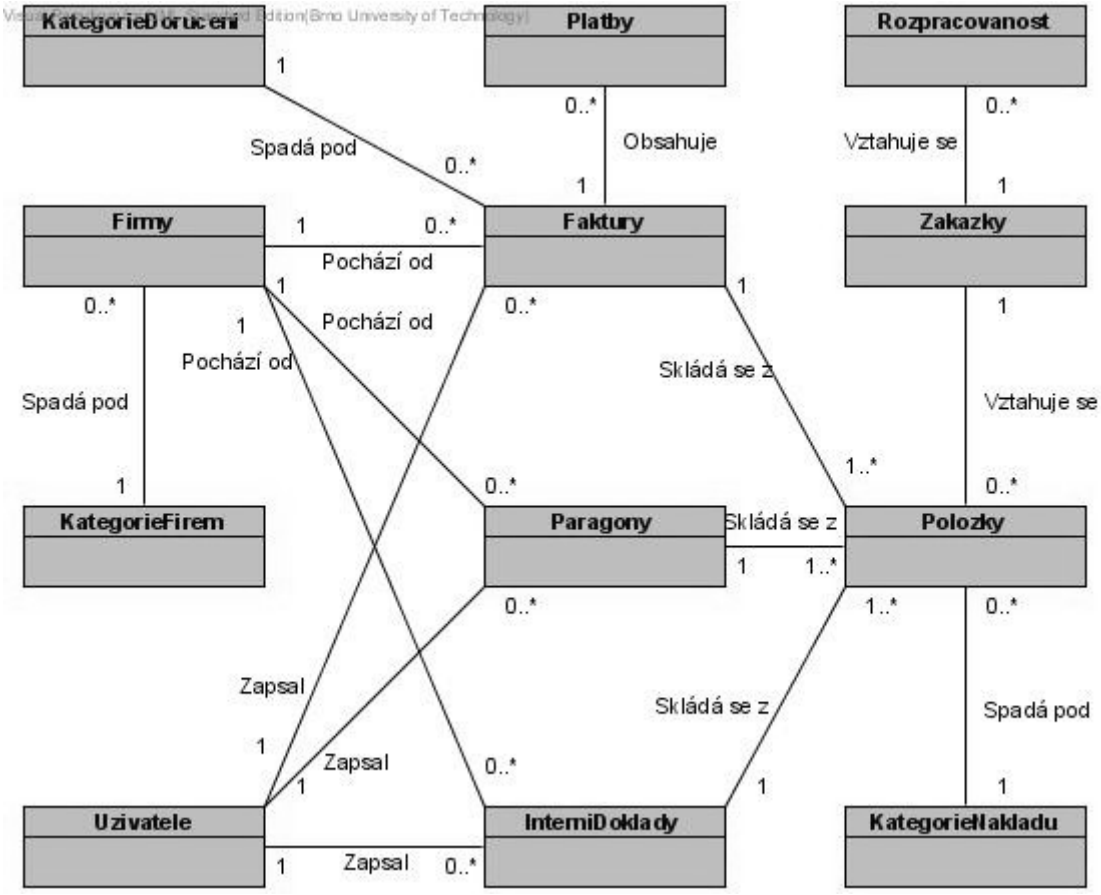
Z neformální specifikace vyplynulo, že systém bude obsahovat dva základní druhy aktérů:

- ***Uživatel*** – Vzhledem k požadované variablitě uživatelských přístupových práv je systém realizován pouze jednou základní uživatelskou úrovní, která dovoluje vykonávat veškeré činnosti v systému. Tyto činnosti jsou pak autorizovány jednotlivými přístupovými právy konkrétních uživatelů. Všichni uživatelé systému jsou potomkem tohoto aktéra.
- ***Čas*** – Bude provádět zálohování databází a kontrolu připojených klientů.

Následující diagram případů užití ilustruje případy užití systému. Jedná se o zjednodušený diagram, kde z důvodů přehlednosti jsou CRUD akce nad jednotlivými typy dat shrnuty do souhrného celku „správa“.



Obrázek 4.1 – Zjednodušený diagram případu užití systému



Obrázek 4.2 – Konceptuální diagram tříd

4.3 Plán projektu

Zákazník požadoval, aby byl systém schopen základního zpracovávání dat (primárně aby bylo možné ho začít plnit daty) v co nejkratším čase. Proto byla jeho realizace rozdělena na 4 iterace v rámci uvažovaného životního cyklu vývoje:

1. V první iteraci je implementován prototyp aplikace realizující základní komunikaci a správu požadovaných celků (faktur, zakázek, uřivatelů a firem), tzn. jejich zobrazení, přidávání a editaci, aby bylo možné začít vkládat data.
2. Ve druhé iteraci je systém rozšířen o správu zbylých, časově nekritických celků (malé daňové doklady-paragony a interní náklady-mzdy) a správu částečných plateb pro jednotlivé faktury. V tuto chvíli je možné systém plnit všemi potřebnými daty.
3. Třetí iterace je více zaměřena na zabezpečení systému. Jedná se převážně o implementaci zálohování databáze a správu konkrétních záloh, přiřazení jednotlivých pravomocí

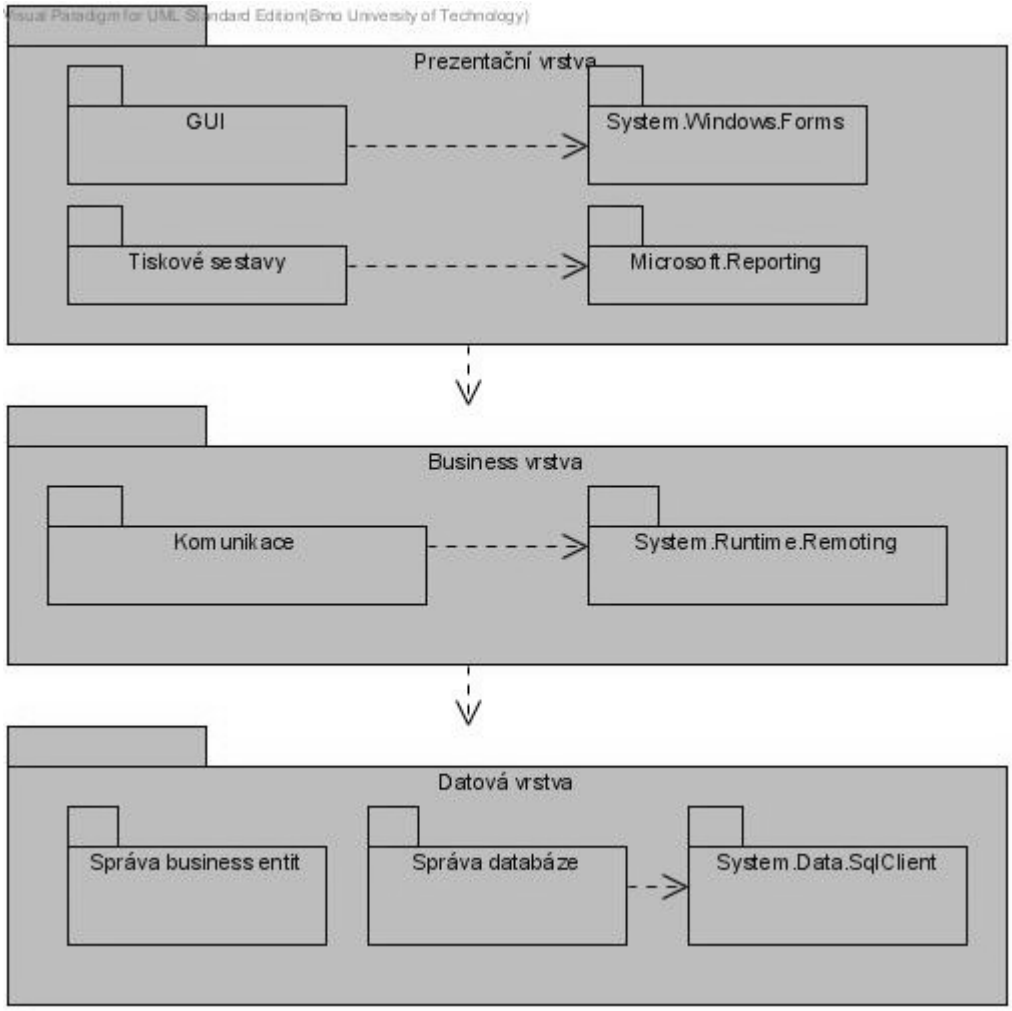
odpovídajícím úkonům v systému a rozšíření stávající komunikace o šifrování a uživatelsky přívětivější práci s objemnými daty.

4. Poslední iterace pak implementuje převážně nemodifikující statistické operace a přehledy nad větším množstvím dat.

4.4 Kostra aplikace

Aplikace je rozdělena do tří logických vrstev v rámci klasické třívrstvé architektury (viz literatura [20]):

- ***Prezentační vrstva*** – nejvyšší vrstva aplikace, realizuje interakci s uživatelem, implementuje GUI a řídí vytváření tiskových sestav. Je kompletně obsažena v klientské části aplikace.
- ***Business vrstva*** – často označovaná také jako logická vrstva, realizuje převážně komunikaci. Jedná se o most mezi klientskou a serverovou částí aplikace, je rozdělena do obou z nich.
- ***Datová vrstva*** – zajišťuje kompletní práci s databází a správu systémových entit. Je kompletně obsažena v serverové části.

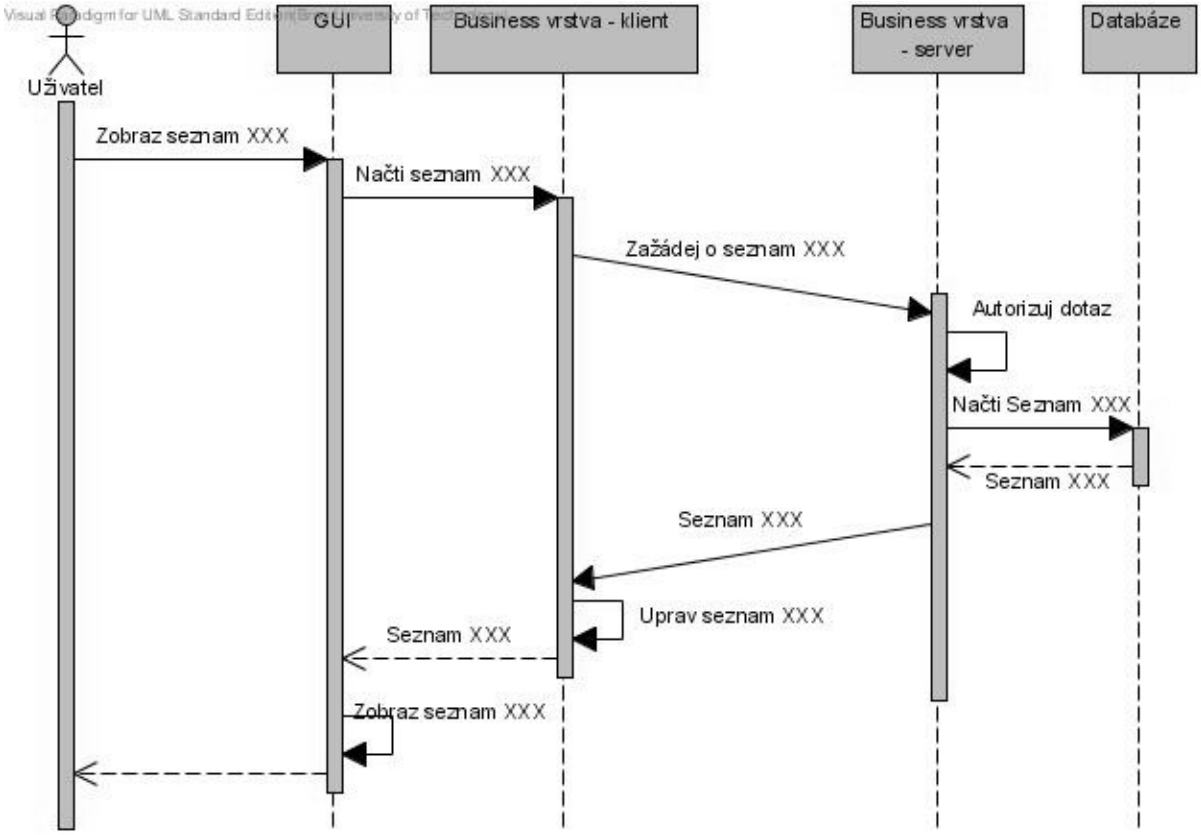


Obrázek 4.3 – Návrh architektury

4.5 Datová výměna

Komunikace v systému je formou dotaz-odpověď a je tak vždy iniciována ze strany klienta. Vzhledem k povaze systému je také možné očekávat přenos poměrně objemných dat ve formě např. seznamu faktur za celý uplynulý rok, což by převážně na pomalejším mobilním připojení mohlo působit komplikace. Proto je nutné umístit komunikaci do samostatného vlákna, uživateli zobrazovat průběžně její pokrok a nabídnout mu možnost ji případně i bezpečně stornovat.

Základní koncept výměny dat zobrazuje následující obrázek. Jedná se o obecný princip komunikace, kde byla pro zjednodušení a generalizaci případu datové výměny použita abstraktní operace „Zobraz seznam XXX“.



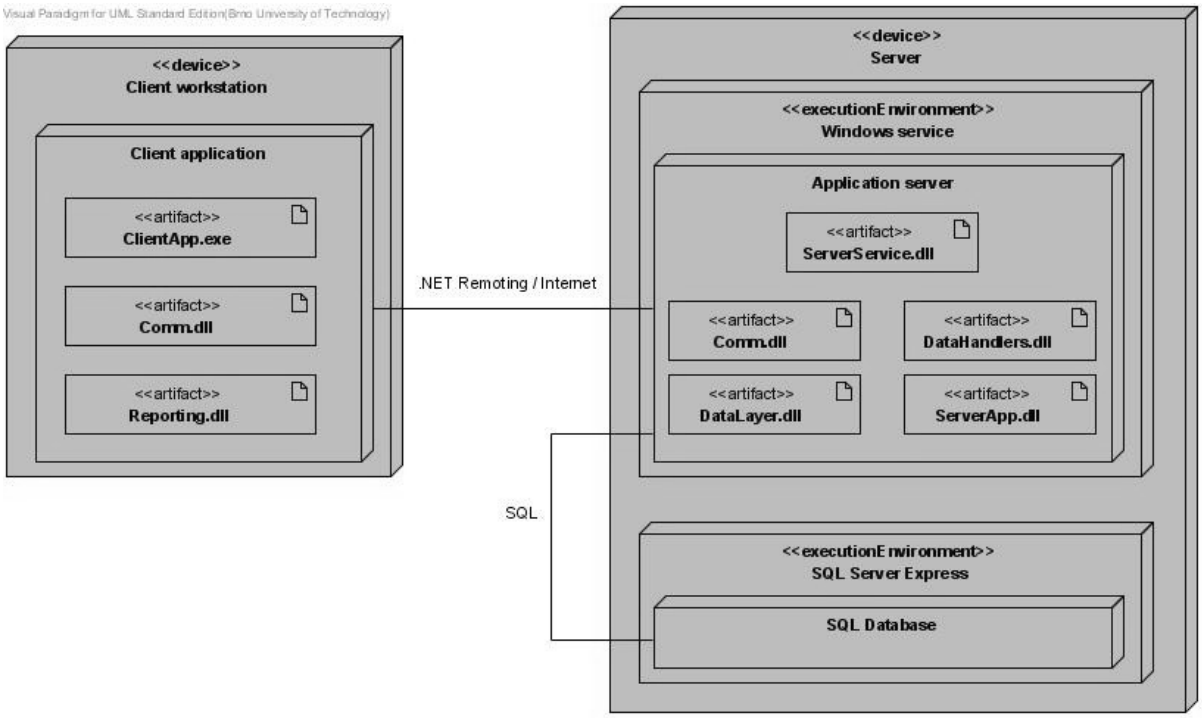
Obrázek 4.4 – Návrh datové výměny

4.6 Zabezpečení dat

Kromě základního použití v intranetu zákazníka je aplikace také koncipována pro použití i v mnohem více nepřátelském prostředí, kterým je internet. Proto je nutné klást velký důraz na zabezpečení komunikace a eliminaci možných bezpečnostních rizik. Přenos dat proto musí být šifrován, nejlépe asymetricky, a jakýkoli dotaz na server před jeho vykonáním náležitě autentizován. Dále je potřeba důsledně logovat přístupy do systému a vykonané změny pro případ zpětného odhalení útoku či chyby.

4.7 Nasazení

Vzhledem k možnostem zadavatele je MS Winows cílovou platformou jak pro klienta, tak pro server. Aplikační server je nasazen formou windows service a využívá databázový server, který je v původním plánu provozován na stejném fyzickém stroji, jako server aplikační. Není to však nutná podmínka a v rámci pozdější úpravy požadavků to může být snadno změněno.

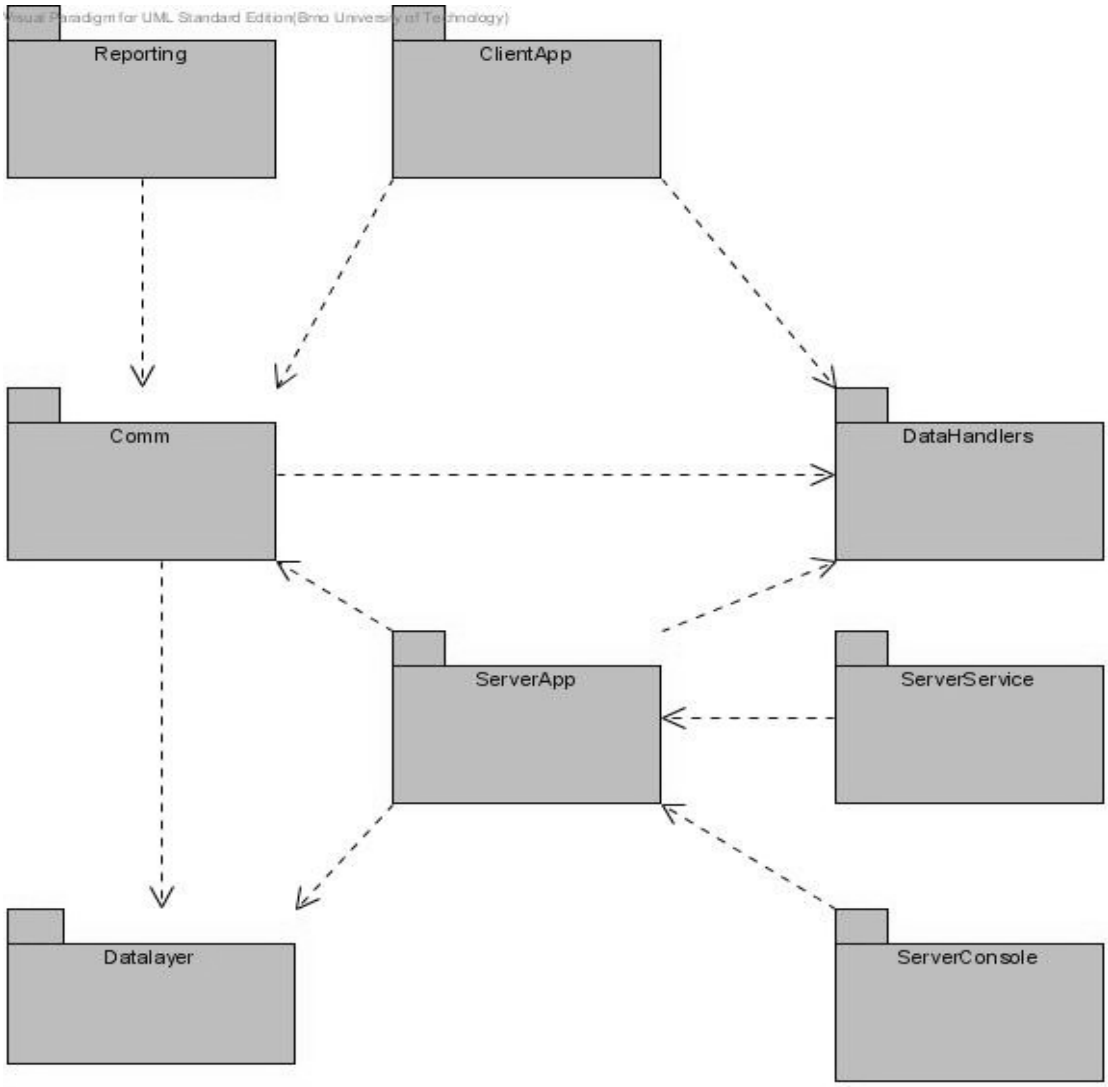


Obrázek 4.5 – Diagram nasazení

5 Implementace

V této kapitole budou probrány hlavní principy implementace všech částí výsledné aplikace. Ta je složena z následujících modulů, jejichž logickou návaznost ilustruje obrázek 5.1:

- ***DataLayer*** – reprezentuje datovou vrstvu a konkrétní fyzický přístup k databázi. Část této logiky je také obsažena v ní samotné ve formě pohledů a vložených procedur.
- ***ServerApp*** – vykonává serverovou logiku aplikace.
- ***Comm*** – modul realizující komunikaci mezi klientem a serverem, obsahuje také definici komunikačního rozhranní a protokolu.
- ***ClientApp*** – reprezentuje spustitelnou klientskou část aplikace, vykonává klientskou logiku aplikace a tvoří prezentační vrstvu.
- ***DataHandlers*** – knihovna obsahující podpůrné prostředky pro chod aplikace, jakými jsou např. funkce kryptografické, logovací, či funkce pro práci s emailem.
- ***Reporting*** – modul zajišťující vytvoření tiskové sestavy faktur resp. platebních příkazů.
- ***ServerConsole*** – spustitelná serverová část ve formě konzolové aplikace. Ta je využívána při ladění projektu.
- ***ServerService*** – spustitelná serverová část aplikace ve formě windows service, která je použita v praktickém nasazení.
- ***ServerService_Administration*** – spustitelná administrační aplikace pro správu a konfiguraci serverové služby.
- ***ServerService_Down*** – miniaplikace ve formě windows service pro komunikaci s uživatelem v době, kdy není serverová služba k dispozici, např. z důvodu údržby.



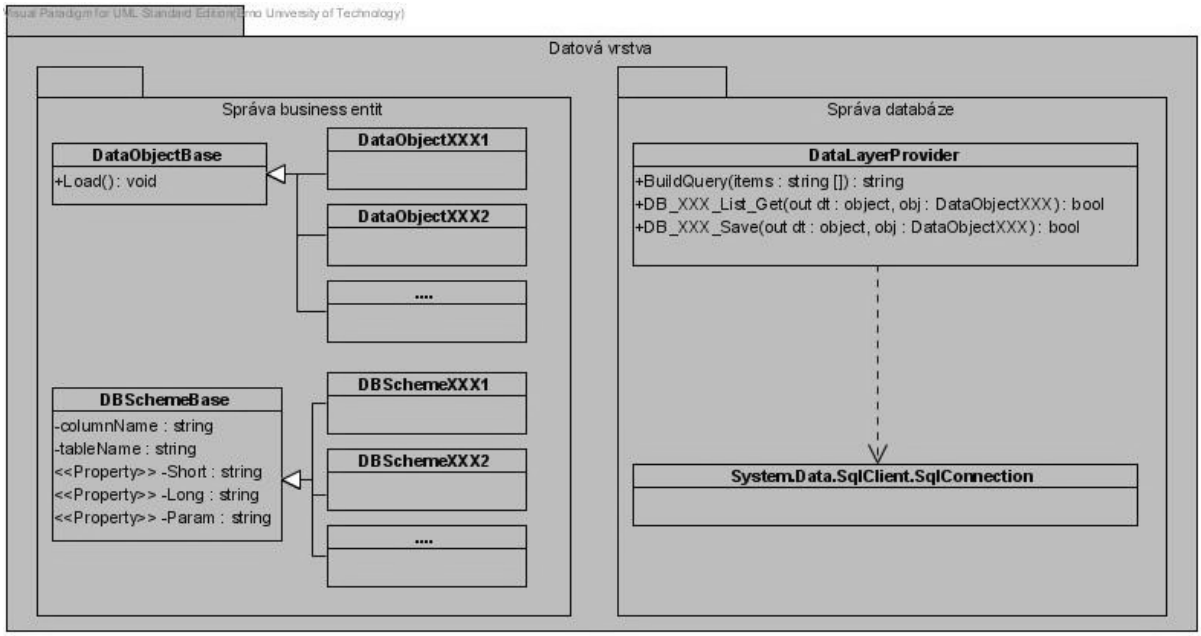
Obrázek 5.1 – Schéma logické závislosti aplikačních modulů

5.1 Databáze

Databáze je implementována jako MS SQL Express databázový soubor. To umožňuje snadnou manipulaci, modifikaci a zálohování kompletních dat a splňuje požadavek zadavatele umožnit pověřené osobě její uložení na přenosné medium kdykoli a bez nutnosti účasti administátora aplikace. Samotná databáze obsahuje vedle tabulek, reprezentujících entity v systému, a pohledů, které usnadňují manipulaci a agregaci dat, také množství vložených procedur, ve kterých je kromě podpůrných procesů pro CRUD operace obsažena i část datové logiky.

5.2 Datová vrstva

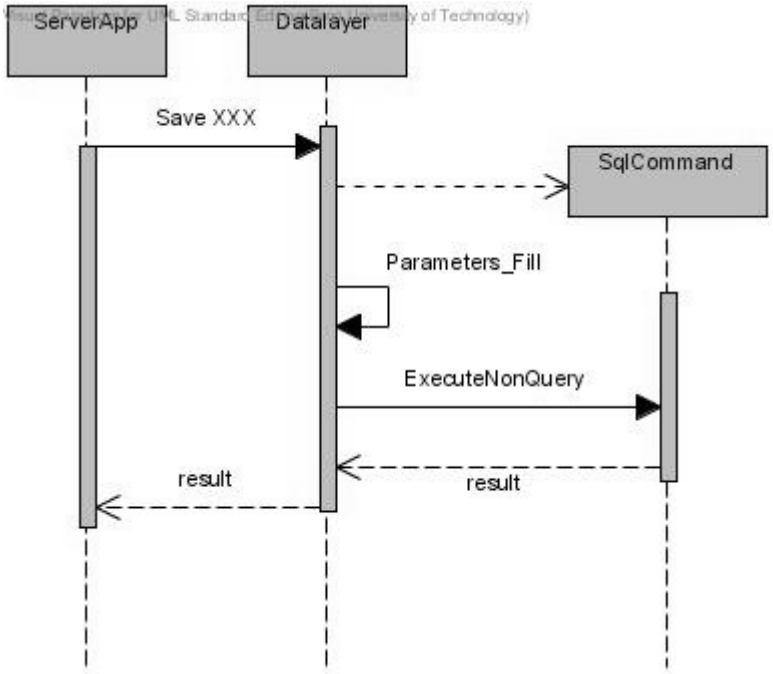
Datová vrstva aplikace realizuje fyzický přístup do databáze. Zajišťuje transakční přístup k logicky provázaným operacím, které pak probíhají atomicky, čímž předchází vzniku nekonzistentního stavu (např. uložením položek k neexistující faktuře). Provádí také mapování databázových entit na objekty, se kterými se dále v systému pracuje.



Obrázek 5.2 – Zjednodušený diagram návrhových tříd datové vrstvy

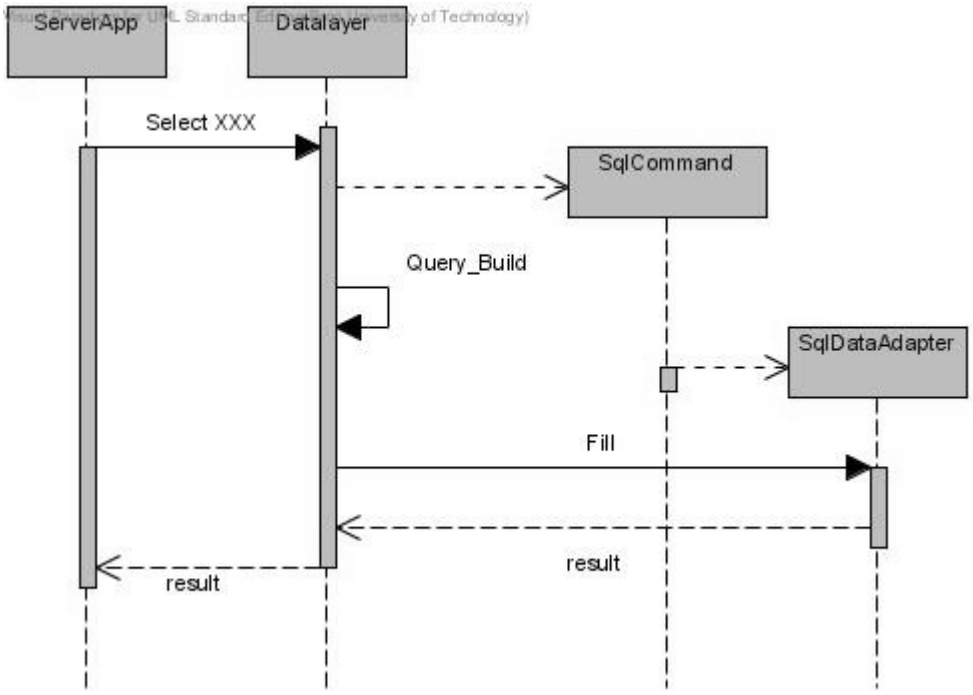
5.2.1 Komunikace databáze-server

Pro výměnu dat mezi databází a serverovou částí aplikace jsou použity dva přístupy. První z nich je zaměřen převážně na CRUD operace pro jednotlivé záznamy dat (konkrétní faktura, firma, zakázka) a využívá parametrizovaných vložených procedur a jejich volání za pomocí ADO.NET. Uplatňuje se při práci s detaily jednotlivých faktur či zakázek. Průběh této operace ilustruje diagram 5.3 pomocí abstraktní operace „Save XXX“.



Obrázek 5.3 – Diagram sekvence operace Save XXX nad jediným záznamem

Druhý přístup je taktéž založen na ADO.NET, místo parametrizovaných vložených procedur však využívá DataAdapter. Pracuje se silně typovaným DataSetem a je tak zaměřen na současné zpracování většího množství dat. To je využito například při načítání seznamů (faktur, zakázek), či načítání a ukládání fakturových položek.

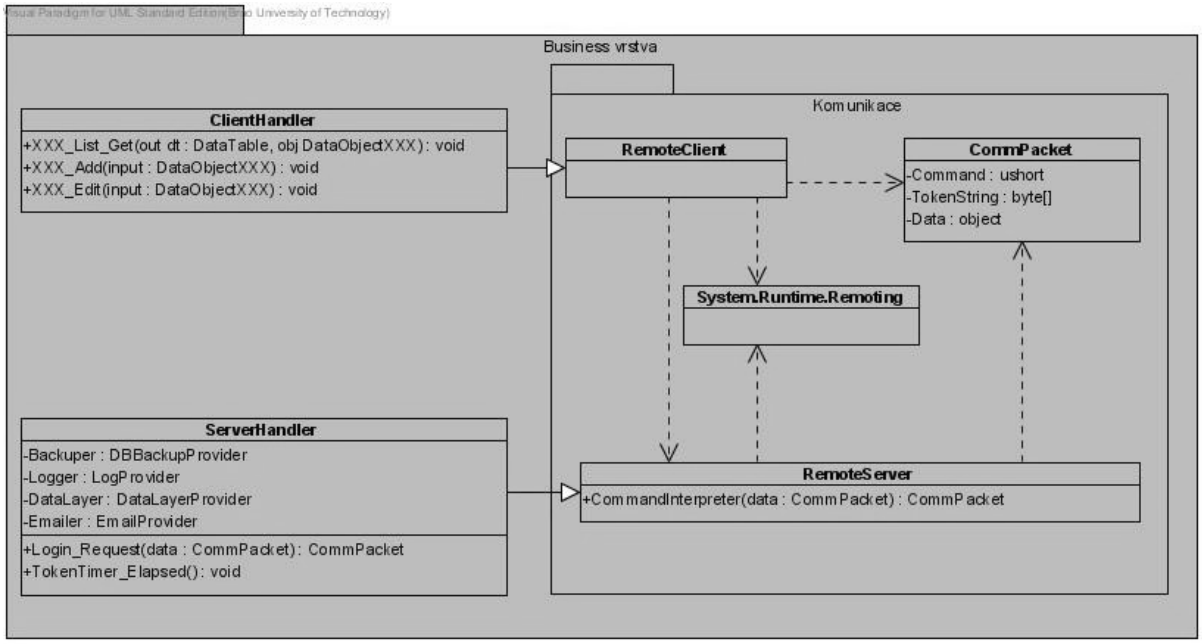


Obrázek 5.4 – Diagram sekvence operace Select XXX nad seznamem záznamů

Vzhledem k použití typovaných parametrizovaných vložených procedur také vzniká další bezpečnostní vrstva ochrany proti útokům typu SQL injection.

5.3 Business vrstva

Vzhledem k tomu, že je aplikace realizována formou tlustého klienta, je bussiness vrstva rozdělena mezi serverovou i klientskou část. Zajištění komunikace mezi těmito dvěma celky je její hlavní funkcí. V tomto distribuovaném systému tak představuje middleware. Zodpovídá také za správu přihlášených uživatelů a realizaci plánovaných záloh databáze. V neposlední řadě poskytuje aplikaci další podpůrné funkce, např. pro práci s emailem či logování chyb a operací.



Obrázek 5.5 – Zjednodušený diagram návrhových tříd business vrstvy

5.3.1 Zálohování

Protože aplikace pracuje s větším množstvím dat, měla by zajistit jejich bezpečné zálohování a případnou obnovu. To probíhá formou přímých operací nad databázovými soubory, jejich zkopírováním do bezpečného uložiště, resp. přepsáním v případě obnovy. Systém nabízí vytváření záloh jak manuálních, které jsou spuštěny k tomu pověřeným uživatelem, tak automatických podle předem nastaveného plánu.

Z důvodu rozložení záloh do širšího záběru jsou automatické zálohy rozděleny do 3 kategorií, které se ukládají nezávisle na sobě:

- ***Denní*** – vytváří se každý den v předem nastavený čas (nejlépe v noci), udržuje se posledních 7 záloh.

- ***Týdenní*** – vytváří se každý nastavený den v týdnu ve stejném čase jako denní zálohy, udržuje se posledních 5 záloh.
- ***Měsíční*** – vytváří se každý nastavený den v měsíci ve stejném čase jako denní zálohy, udržuje se neomezený počet záloh.

Samotné vytvoření zálohy může trvat i relativně delší dobu (stále se ale pohybujeme v časech menších než 1s), během níž není databáze k dispozici. To je způsobeno umělým časovým oknem mezi odpojením databáze a fyzickým přístupem k databázovému souboru. Čekání o délce v řádech desítek až stovek milisekund poskytuje SQL serveru dostatečně dlouhou dobu na jeho uvolnění. Během této doby nemůže aplikace přistupovat do databáze, aplikační server tedy na všechny příchozí dotazy reaguje odpovídajícím informačním hlášením.

5.3.2 Komunikace server-klient

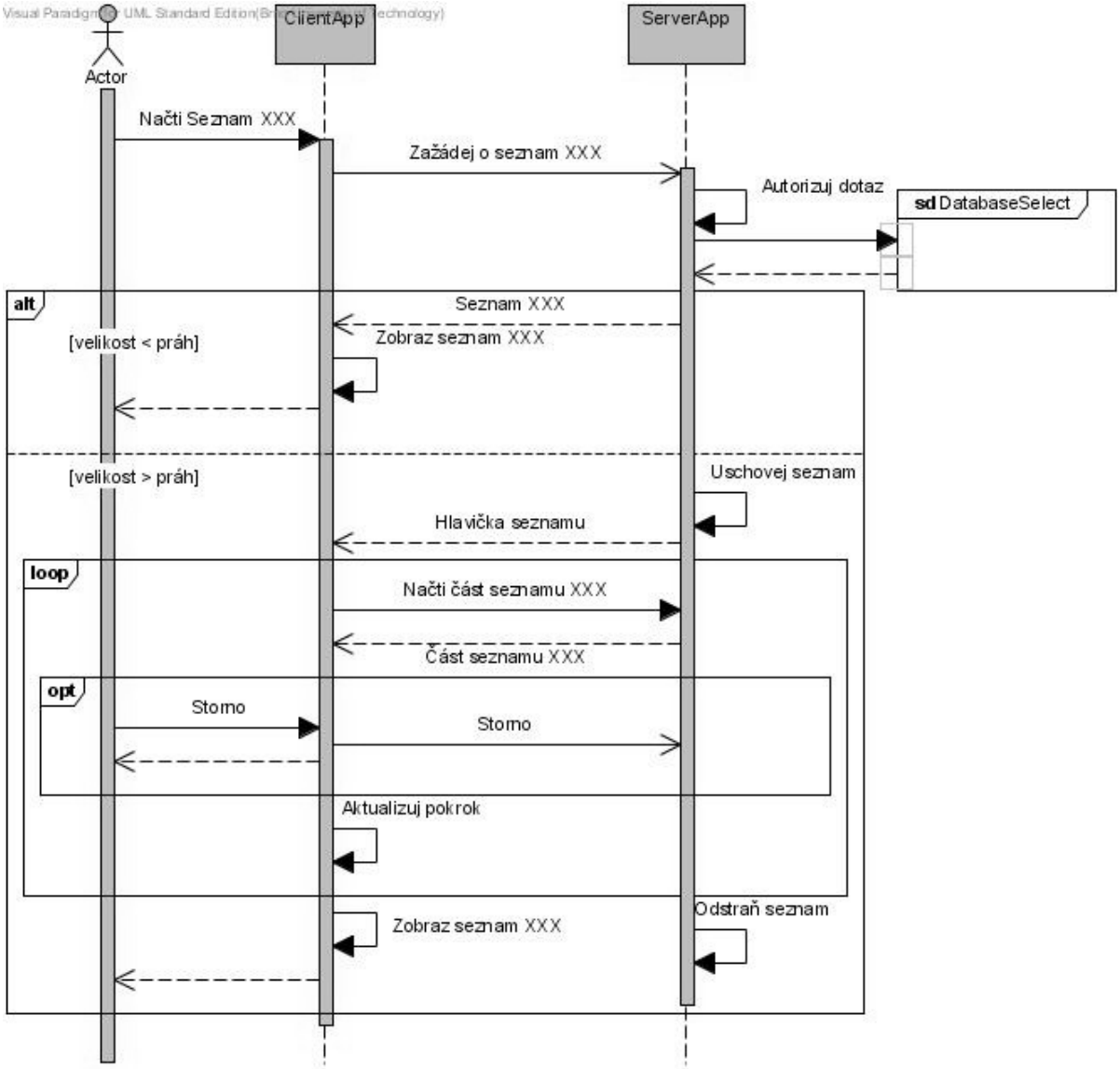
Výměna dat mezi serverem a klientem je realizována technologií .NET Remoting prostřednictvím TCP kanálu pro optimální rychlost. Celá serverová aplikace je v podstatě vzdálený objekt, ke kterému klienti přistupují voláním jeho metod. Veškerá komunikace je zprostředkována jedinou metodou, která přijímá i vrací komunikační „packet“ obsahující identifikátor operace, identifikaci uživatele a samotná přenášená data. Tato metoda je volána v samostatném vlákně. Pokud na ni nedostane aplikace odpověd v přednastaveném „timeout“ čase, vyhodnotí tuto situaci jako nemožnost navázat spojení a toto vlákno ukončí.

Typ přenášených dat je pak závislý na konkrétní operaci a podle jejího druhu se liší. V případě méně objemných dat, představujících většinou jediný záznam (faktura, zakázka, atd.), se jedná o samotný objekt mapující odpovídající entitu. Při načítání objemných dat, jakými jsou např. různé statistické přehledy a seznamy těchto entit, je pak jednotkou přenášené informace celá tabulka ve formě DataSet/DataTable. Tyto seznamy však mohou být poměrně rozsáhlé (počet vystavených faktur za jeden kalendářní rok se může pohybovat v řádech stovek či dokonce tisíců), proto je třeba k těmto přenosům přistupovat poněkud odlišně, o čemž pojednává následující kapitola.

5.3.3 Přenos objemných dat

Transport rozsáhlých tabulek dat, jakými jsou např. různé přehledy či seznamy, může zvláště na pomalejším připojení trvat znatelně dlouhou dobu, během níž by měl uživatel vědět, co a zda se vůbec něco děje. Samotný přístup volání vzdálených metod a jeho zapouzdřená režie komunikace však navenek neposkytuje žádný přehled o aktuálním stavu a objemu dosud přenesených dat. Proto transport těchto tabulek vyžaduje zvláštní režii:

1. Pokud server vrací na nějaké volání objekt DataTable, je nejprve zjištěna jeho velikost (počet řádků). V případě, že je tato velikost menší než nastavený práh (v důsledku experimentování s ruznými hodnotami byla zvolena velikost 12 řádků), je s ním zacházeno jako s jakýmkoli jiným malým objektem a je standardně vrácen ve volání.
2. V případě, že je jeho velikost větší než práh, je tato tabulka uložena do speciálního úložiště na serveru a je jí vygenerován unikátní identifikátor. Klientské aplikaci je pak vrácen objekt představující pouze jakousi hlavičku, která obsahuje informace o uložené tabulce, její velikost a identifikátor.
3. Klient pak postupně volá načítání jednotlivých částí DataTable ze serveru, skláda je dohromady a informuje uživatele o pokroku. Během tohoto procesu má uživatel také možnost akci stornovat.
4. V případě načtení poslední části přenášené tabulky, či v případě stornování uživatelem, je tato tabulka odstraněna z dočasného úložistě.



Obrázek 5.6 – Diagram sekvence přenosu objemných dat

Celkový objem dat se sice při použití takového postupu nepatrně zvětší vzhledem k režii nutné pro volání více metod, možnost průběžného informování uživatele či stornování požadavku však tento mírný nárůst načítacího času dostatečně vykompenzuje.

5.4 Prezentační vrstva

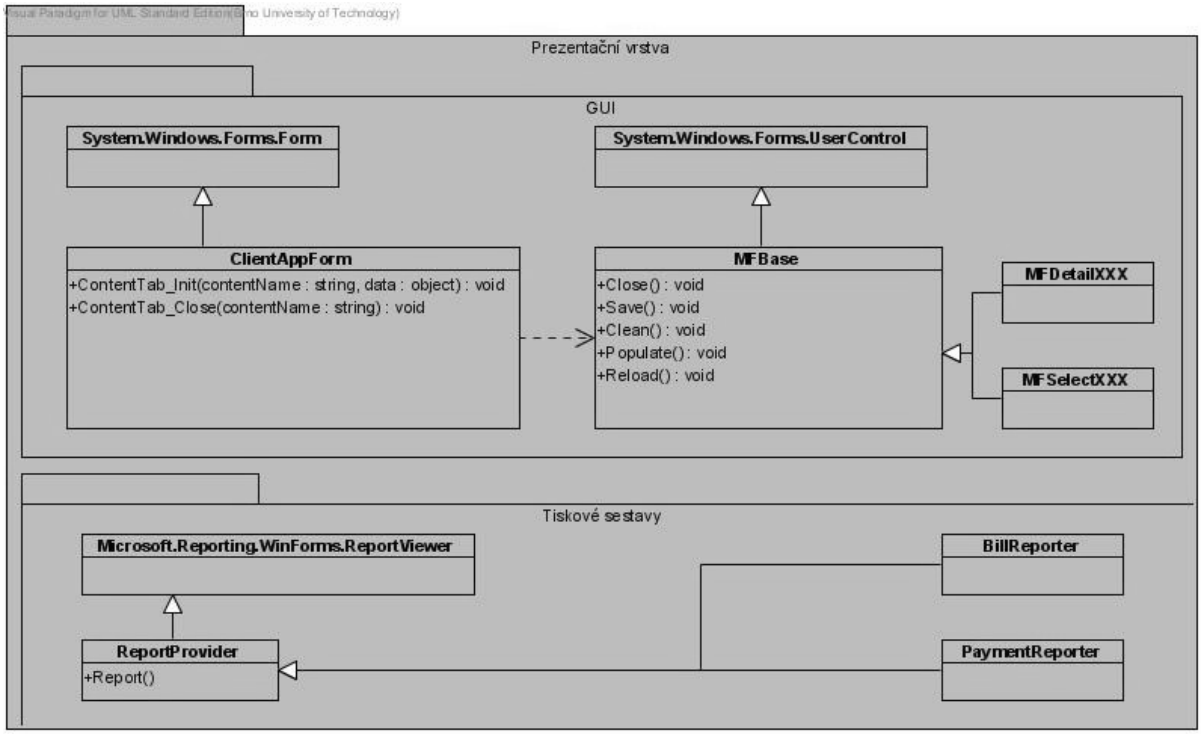
Prezentační vrstva klientské aplikace je realizována prostřednictvím rámce Windows Forms. Spravuje zobrazení načtených dat uživateli a umožňuje jejich lokální modifikaci. Manipulace s daty je pak z grafické stránky řešena formou nezávislých záložek (Tabs) v hlavním okně aplikace, díky čemuž je

umožněna práce s více druhy dat současně. Jsou zde použity převážně dva druhy záložek vztahující se na systémové datové entity:

- ***Seznam*** (Select) – Zobrazuje načtený seznam záznamů.
- ***Detail*** (Detail)– Zobrazuje veškerá data jednoho konkrétního záznamu. Přístup do detailu je možný pouze ze seznamu.

Kromě těchto dvou nejčastěji používaných typů se zde objevují ještě některé atypické záložky pro specifické účely.

Prezentační vrstva má za úkol také vytváření tiskových sestav z načtených dat, kterými umožňuje grafický export faktur ze systému, popř. usnadňuje jejích placení formou interního platebního příkazu.



Obrázek 5.7 – Zjednodušený diagram návrhových tříd prezentační vrstvy

5.4.1 Seznam

Hlavní část seznamové záložky tvoří tabulka s načtenými daty. Ta je realizována prvkem DataGridView, který dává uživateli možnost libovolné práce se sloupci, jako je změna velikosti či pořadí, a např. v případě faktur umožňuje přehledně rozlišit jednotlivé záznamy podle jejich splatnosti v závislosti na aktuálním datu. Zbytek této záložky pak tvoří soustava vyhledávacích filtrů, vztahující se ke konkrétnímu typu dat. Vzájemně se doplňují a z důvodů minimalizace množství přenesených dat se aplikují na serverové straně. Při každé jejich změně je tedy nutné data znovu načíst.

Seznam je také vzhledem k datům nemodifikující. Slouží pouze pro přehledný výpis většího množství dat a jako mezičlánek v navigaci k detailu konkrétního záznamu.

5.4.2 Detail

Záložka detailu už je poté specifická pro každý druh datových entit. Kromě editovatelných textových polí pro modifikaci jejich základních atributů obsahují např. seznam položek (v případě faktur, paragonů a interních dokladů), přehlednou roční bilanci příjmů a nákladů (v detailu zakázky), či seznam uživatelských práv (v detailu uživatele).

Záložka detailu také monitoruje veškeré změny v datových polích. V případě, že chce uživatel záložku zavřít a tyto změny v ní ještě nejsou uloženy, je vyzván k obsluze této události buď explicitním uložením, jejich ignorováním, či stornováním celé akce. Totéž platí při zavírání celé aplikace, kdy jsou postupně ukončovány a stejným způsobem kontrolovány všechny otevřené karty záložek.

5.4.3 Atypické záložky

Mezi záložky zobrazované v hlavním okně aplikace patří vedle výše zmiňovaných typů seznam a detail také některé další, úzce se vztahující ke specifickým požadavkům zadavatele.

- ***Rozvaha*** – Jedná se o komplexní celoroční přehled nákladů a příjmů ve vztahu k jednotlivým zakázkám a měsícům ve formě finanční bilance.
- ***Náklady/příjmy*** – Detailní měsíční rozpis zakázky pro zadaný rok. Pro vybrané měsíce zobrazuje seznam položek ze všech faktur, paragonů a interních nákladů, které se k této zakázce vztahují, a umožňuje rychlou navigaci přímo na jejich konkrétní zdroj.
- ***Nastavení serveru*** – Realizuje interface pro správu záloh na serveru a nastavení registračního emailu.
- ***Platby*** – Představuje informační panel pro průběžně označené faktury určené k platbě či částečné úhradě.

5.4.4 Tiskové sestavy

Pro grafický export faktury ze systému jsou použity tiskové sestavy. Jedná se o předem připravenou šablonu vytvořenou podle stávajících papírových faktur zadavatele vyplněnou konkrétními daty ze systému. Takto vytvořenou fakturu je dále možné přímo tisknout či převést do souboru ve formátu PDF nebo excel.

Obdobným způsobem je řešen seznam plateb. Po vybrání požadovaných faktur či částečných úhrad je uživateli předložen jejich přehledný seznam, seřazený po jednotlivých transakcích podle

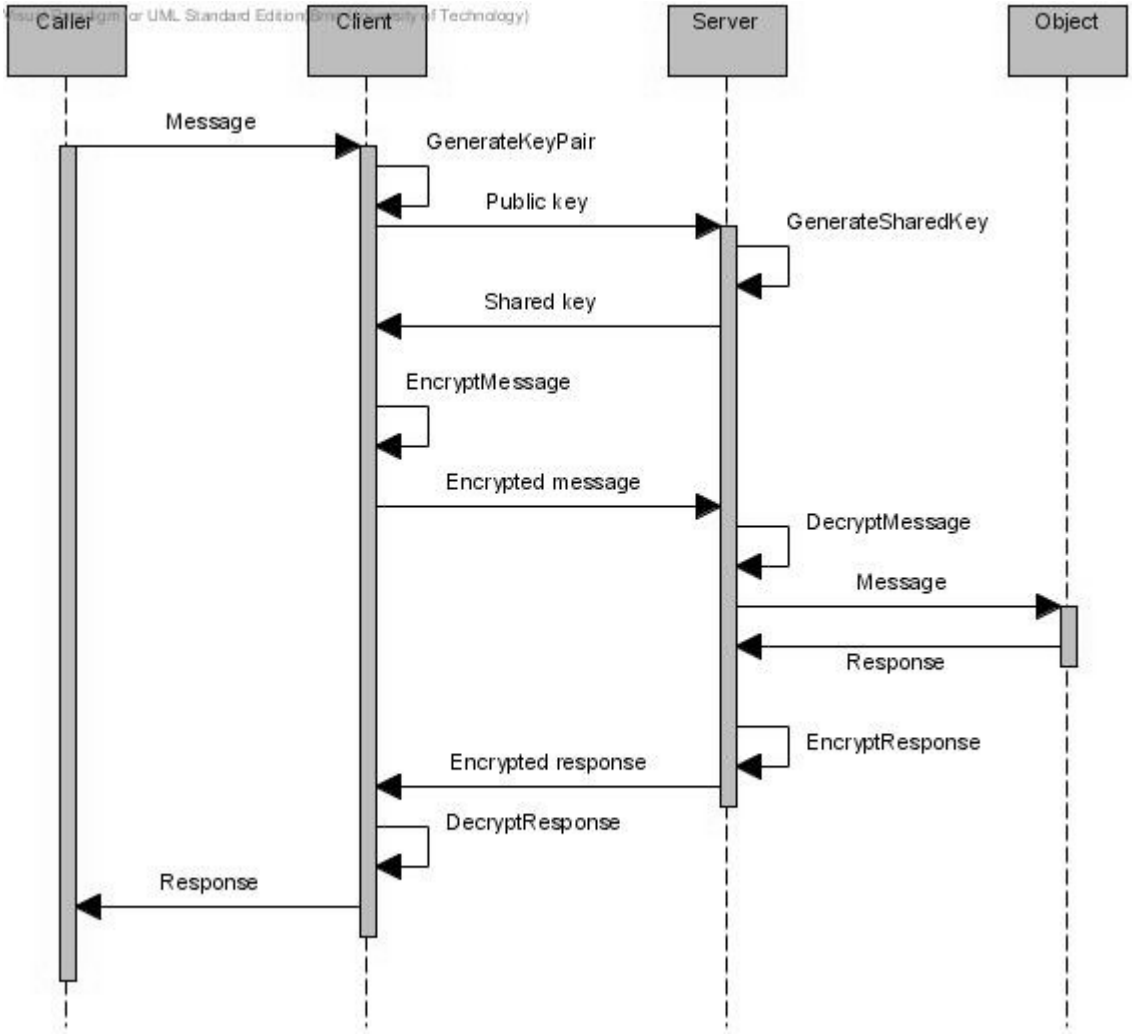
cílových účtů (firem), umožňující snažší realizaci samotné platby. Tento seznam lze opět vytisknout či exportovat do souboru.

5.5 Zabezpečení

Každá aplikace pracující s neveřejnými daty musí nějakým způsobem zaručit jejich zabezpečení proti odcizení či neoprávněné modifikaci. V klasické nedistribuované aplikaci by takovým zapezpečení bylo omezení přístupu k datovému úložišti a případné rozlišení uživatelských rolí a jejich přístupových práv ke konkrétním datům. V případě aplikace, která je navíc provozována v tak nebezpečném prostředí, jakým je internet, jsou nároky na její zabezpečení navýšeny o nutnost zabezpečení komunikace samotné.

5.5.1 Komunikace

Technologie .NET Remoting, kterou je komunikace v tomto systému realizována, v základu nabízí pouze omezené možnosti zabezpečení komunikačního kanálu. Pro specifické případy, jaký nastal právě v této aplikaci, je nutné šifrování komunikace zajistit explicitně formou specializované vrstvy umístěné mezi formátovací část komunikace (formatter). Pro vyšší bezpečnost je zde použito asymetrické šifrování, vrstva tedy kromě samotného zakódování a dekódování realizuje i počáteční výměnu komunikačních klíčů.



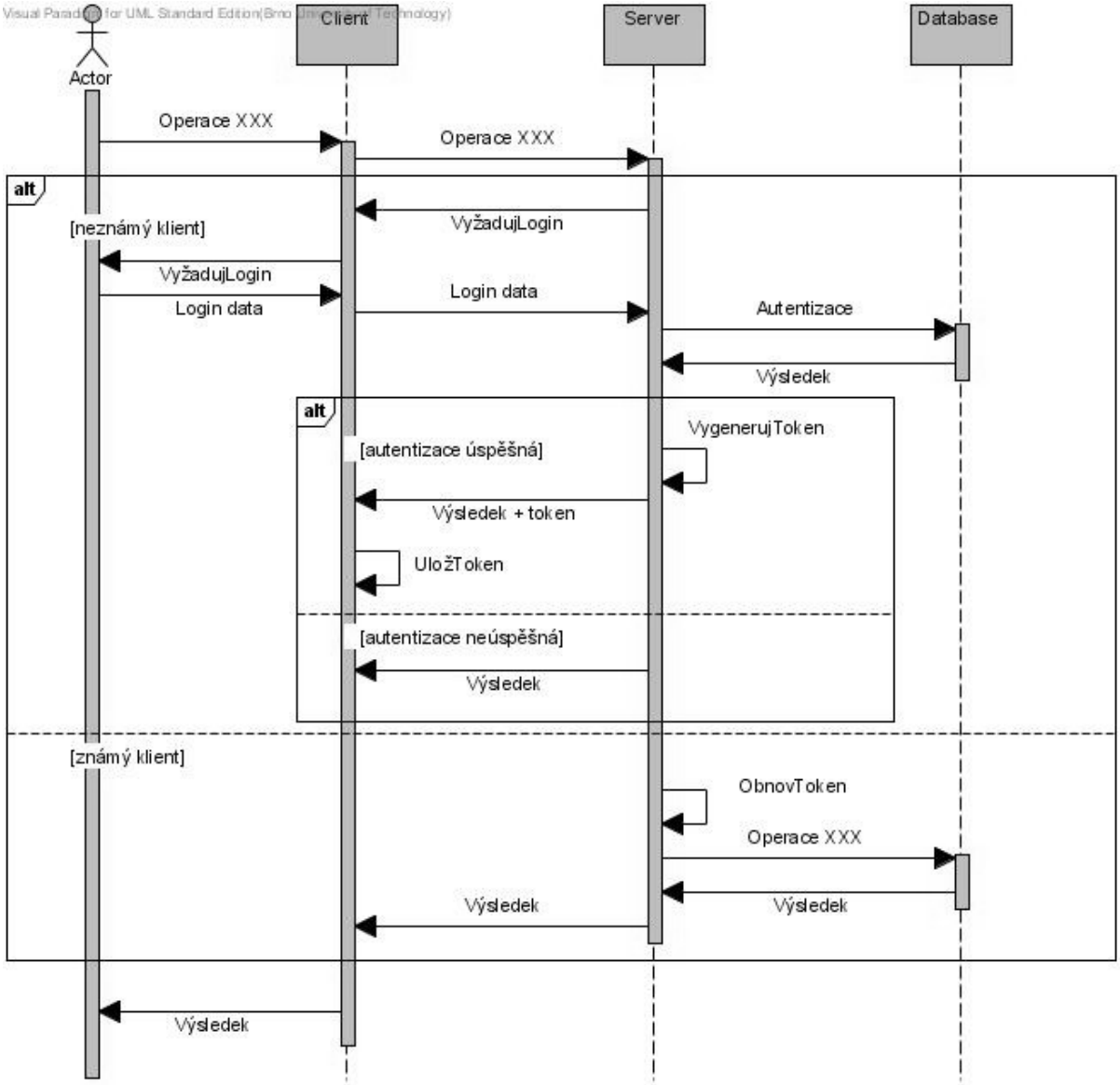
Obrázek 5.8 – Schéma šifrování komunikace

Implementace této šifrovací vrstvy je z větší části převzata, vyžadovala ovšem drobné úpravy a přizpůsobení cílové aplikaci (viz literatura [21],[22]).

5.5.2 Přihlášení uživatelů

Každá operace vykonávaná na serveru, která nějakým způsobem pracuje s daty, musí být náležitě autorizována vzhledem k uživateli, který ji chce vykonat. Server ho proto potřebuje jednoznačně identifikovat. Udržuje seznam platných přihlášených uživatelů ve formě bezpečnostních tokenů s omezenou časovou platností. V případě žádosti o operaci od klienta, jenž se v tomto seznamu nenachází, jsou nejprve serverem vyžádány bezpečnostní údaje ve formě přihlašovacího jména a hesla. Ze zadaného hesla je vytvořen MD5 hash a porovnán s databází. Po úspěšné autentizaci je vygenerován unikátní 128-bitový bezpečnostní token s předem nastavenou dobou platnosti, který pak

zodpovídá za budoucí autentizaci dotazů. Životnost tokenu je obnovena při každém platném dotazu, po jejím vypršení je však uživatel ze seznamu vyřazen a je vyžadováno opětovné přihlášení.



Obrázek 5.9 – Schéma přihlášení uživatele

5.5.3 Uživatelské účty

K bezpečnostnímu tokenu se také vztahuje autorizace uživatele ke konkrétním operacím. V rámci systému se zde vyskytuje pouze jediná skupina uživatelů, neexistuje tu žádné speciální administrátorské či podobné uživatelské účty. Jejich vzájemné rozlišení vzhledem k přístupu ke konkrétním úkonům je realizováno formou pravomocí, které jim mohou být přiděleny.

Kromě některých specifických pravomocí se jedná o čtyřstupňový systém přístupu k jednotlivým datovým entitám, které jsou na sobě navzájem nezávislé.

- ***Nezobrazit*** – uživatel není oprávněn tyto entity ani zobrazit. Vyjímku tvoří seznam firem, který lze zobrazit ve zkrácenné verzi, obsahující pouze jejich názvy, z důvodu využití ve vyhledávacích filtrech.
- ***Zobrazit*** – uživatel má přístup k veškerým detailům konkrétních dat, nemůže je však nikterak modifikovat ani přidávat nové.
- ***Vkládat*** – uživatel může do systému zadávat nová data, není však oprávněn modifikovat ty, které v něm už jsou. Vyjímku tvoří položky daňových dokladů, pravomoci uživatelů a položky rozpracovanosti, které mají vzhledem ke způsobu realizace tento stupeň přístupu vynechán.
- ***Upravovat*** – uživatel má umožněnu plnou správu nad konkrétními daty.

Mimo tento čtyřúrovňový systém stojí pravomoci pro zobrazení rozvahy, administrace serveru a s ní spjaté vytváření databázových záloh a jejich obnovení, a tzv. „superuser“, který umožňuje přihlášení za libovolného uživatele bez znalosti jeho hesla, což bylo využito hlavně při ladění aplikace.

Z hlediska implementace se jedná o bitové pole, kde každý bit odpovídá jedné konkrétní pravomoci. Je načteno při autentizaci uživatele a uloženo jako součást bezpečnostního tokenu. U výše zmiňovaného čtyřstupňového systému je zaručena provázanost mezi jednotlivými stupni, takže při nastavení nějaké úrovně získá uživatel automaticky všechna práva úrovní nižších.

Při vytváření nového uživatelského účtu je vyžadován pouze login a emailová adresa. Server se pak na tuto adresu pokusí odeslat email s registračními údaji, včetně unikátního vygenerovaného hesla. V případě, že administrátorem v aplikaci nebyl nastaven odchozí SMTP server, či se email z nějakého důvodu nepodařilo odeslat, je heslo nastaveno na stejnou hodnotu jako přihlašovací jméno. Uživatel by si jej pak po přihlášení měl sám změnit.

6 Provoz v praxi

Práce na systému započaly už v létě roku 2008, první prototyp byl pak pro testovací účely spuštěn v březnu 2009 a po celý zbytek roku běžel souběžně se starým IS systémem firmy. Během této doby probíhala úzká spolupráce se zadavatelem a systém byl upravován k jeho konkrétním potřebám a požadavkům. Probíhaly zde tedy minoritní i razantnější zásahy do datového zpracování či vizuálního konceptu aplikace. Samotné nasazení však také doprovázely určité komplikace.

6.1 Problémy nasazení

V prvé řadě jimi byly problémy infrastruktury v brněnské kanceláři, kde běžel aplikační server. Nejdříve bylo nutné vyměnit nekvalitní směrovač, který sám od sebe obnovoval své tovární nastavení hned několikrát během týdne, čímž pokaždé znemožnil přístup k aplikaci z internetu. Dále zde pak vyvstávaly komplikace s internetovým připojením, které bylo agregováno v rámci celé kancelářské budovy a se svou velice nízkou upload rychlostí se tak ve špičkách stávalo pro provoz serveru nepoužitelné. Firma tedy přešla k jinému poskytovateli internetu a získala tak dedikované připojení pouze pro svoji kancelář a s dostatečně širokým přenosovým pásmem.

Vedle těchto problémů se zde objevovaly také problémy implementační, převážně pak jeden konkrétní, týkající se komunikace, jehož „vyřešení“ zabralo hned několik měsíců. Objevoval se pouze při přenosu velkého množství dat, zejména při načítání seznamu všech faktur v systému. Načítání tohoto seznamu „zamrzlo“ zhruba ve čtvrtině a komunikační port se zablokoval. Přibližně jednu hodinu pak byla veškerá další komunikace na tomto portu odmítnuta, jakoby na něm nenaslouchal žádný proces. Tato chyba se také vyskytovala, pouze pokud byl server zapojen v brněnské kanceláři firmy (na libovolném počítači) a pouze při připojování z vnějšího prostředí. Za účelem vyřešení tohoto problému bylo 3x od základu přepracováno komunikační jádro a použity různé technologie, než se tato chyba přestala objevovat. Její příčina nebyla nikdy odhalena, pravděpodobně se ale jednalo o nějakou kombinaci nastavení komunikace, síťových prvků, na které byl fyzický server napojen, či bezpečnostních opatření u poskytovatele internetu.

6.2 Současný stav a vize do budoucna

Od počátku roku 2010 běží systém v ostrém provozu jako primární IS firmy. Ten původní se zcela přestal využívat v březnu téhož roku. Aktivně s ním každodenně pracuje 6 osob, připojujících se převážně z různých lokací, včetně využití mobilního spojení.

Vývoj systému však stále není ukončen. Vedle drobných změn a oprav na již implementovaných částech probíhají v současné době práce na dalších smluvených vylepšeních. Jedná se především o rozšíření nabídky přehledů nákladů a příjmů a jejich případné grafické zobrazení formou srozumitelných grafů, rozšíření tisknutelného exportu dat i nad rámec faktur a zavedení nějaké primitivní matematické logiky při zadávání jednotlivých položek (umožnit aspoň základní operace sčítání, odčítání, násobení a dělení). V delším časovém horizontu je pak plánováno umožnit uživateli i offline přístup k lokální databázi (např. ke stáhnuté serverové záloze), nejspíš však s omezeným či zcela zakázaným zápisem.

V poslední době se zadavatel začal zabývat také myšlenkou redistribuce tohoto IS. V současnosti tak probíhá jednání se třemi nejbližšími obchodními partnery o případném rozšíření systému i do jejich firmy. Vzhledem k vývoji úzce spjatým s konkrétní firmou zadavatele by však tento krok vyžadoval některé zobecňující implementační změny, které jsou však z hlediska priorit plánovaných rozšíření systému momentálně spíše odsunuty do pozadí.

7 Závěr

Na základě požadavků zadavatele byl tedy navrhnut a implementován distribuovaný informační systém pro přehlednou správu zakázek, faktur, obchodních partnerů a různých finančních toků firmy, který plně nahradil předchozí zastaralý IS. Hlavní důraz byl přitom kladen na minimalizaci množství přenesených dat a možnost využití i pomalého mobilního připojení, čímž byl uživateli umožněn přístup ke všem důležitým datům i z „terénu“, a to i z více míst současně.

Výsledný software je doposud zdaleka největším projektem, který jsem zpracovával.Vyžadoval proto poněkud komplexnější přístup a využití softwarového inženýrství. Vyzkoušel jsem si tak vývoj software od naprostého začátku, v podobě zpracování požadavků, přes důsledný návrh, implementaci a nakonec i samotné nasazení. Během implementačních prací jsem nabyl nespočet cenných zkušeností v oblasti objektově orientovaného programování, osvojil si vývojové praktiky desktopových aplikací a rámce .NET Framework a blíže se seznámil s pokročilejšími databázovými operacemi. Vzhledem k delšímu časovému období, které vývoj pokrývalo, jsou na implementaci znát rozdíly mezi částmi, které vznikly v rámci prvních prototypů aplikace a těmi, které byly do systému přidány až v „dokončovacích“ fázích. Tyto rozdíly korespondují s postupně narůstajícím množstvím mých zkušeností a celkovým přehledem v této oblasti.

Vzhledem k pozitivním ohlasům, které tento systém vyvolal jak v rámcí zadavatelské firmy tak i u oslovených partnerů si troufám říct, že byl projekt zpracován úspěšně. Práce na něm však dále pokračují a systém je stále otevřen pro další možná rozšíření. O jeho budoucím rozvoji však bude rozhodnuto až v průběhu času.

Literatura

[1] *Principles and characteristics of distributed systems and environments* [online]. [2009] [cit. 2009-12-22]. Dostupný z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/PDI-IT/lectures/prednaska_1.pdf>.

[2] *Synchronization in Distributed Systems* [online]. [1996] [cit. 2009-12-21]. Dostupný z WWW: <http://phoenix.goucher.edu/~kelliher/cs43/apr09.html>.

[3] *Consistency and Replication* [online]. [2009] [cit. 2009-12-22]. Dostupný z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/PDI-IT/lectures/prednaska_5.pdf>.

[4] FORNACIARI, William. *Mutual Exclusion in Distributed Systems* [online]. [s.l.] : [s.n.], 2003 [cit. 2010-03-25]. Dostupné z WWW: <http://home.dei.polimi.it/fornacia/didattica/labsw0304/2003DistribME.pdf>.

[5] *Klient-server* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Klient-server>.

[6] *Peer-to-peer* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Peer-to-peer>.

[7] *Interprocess communication of distributed systems* [online]. [2009] [cit. 2009-12-22]. Dostupný z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/PDI-IT/lectures/prednaska_2.pdf>.

[8] KIM, Jungkee. *Message-Oriented Communication* [online]. [2005] [cit. 2009-12-22]. Dostupný z WWW: <http://kjk.skhu.ac.kr/MOM.html>.

[9] *Message-oriented middleware* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Message-oriented_middleware>.

[10] *How RPC Works* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/aa373935%28VS.85%29.aspx>.

[11] *RPC: Remote Procedure Call protocol* [online]. [2007] [cit. 2009-12-22]. Dostupný z WWW: <http://www.javvin.com/protocolRPC.html>.

[12] AMIR, Yair, STANTON, Jonathan. *Threats in Distributed Systems* [online]. [2009] [cit. 2009-12-22]. Dostupný z WWW: <http://www.cs.jhu.edu/~yairamir/cs437/week12/sld005.htm>.

[13] *Man in the middle* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Man_in_the_middle>.

[14] CHIGRIK, Alexander. *SQL Server 2000 vs Oracle 9i* [online]. 2005 [cit. 2009-12-22]. Dostupný z WWW: <http://www.mssqlcity.com/Articles/Compare/sql_server_vs_oracle.htm>.

[13] SHERIFF, Paul. *Introduction to ASP.NET and Web Forms* [online]. 2001 [cit. 2009-12-22]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms973868.aspx>.

[14] HOWARD, Rob. *Web Services with ASP.NET* [online]. 2001 [cit. 2009-12-22]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms972326.aspx>.

[15] DHAWAN, Priya, EWALD, Rim. *ASP.NET Web Services or .NET Remoting: How to Choose* [online]. 2002 [cit. 2009-12-22]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms978420.aspx>.

[16] *.NET Remoting* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/.NET_Remoting>.

[17] *Windows Communication Foundation* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Windows_Communication_Foundation>.

[18] *Windows Installer* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Windows_Installer>.

[19] *ClickOnce* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/ClickOnce>.

[20] *Multitier architecture* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture>.

[21] TOUB, Stephen. *.NET Remoting: Writing an Asymmetric Encryption Channel Sink* [online]. 2003 [cit. 2009-12-22]. Dostupný z WWW: <http://msdn.microsoft.com/cs-cz/magazine/cc300447%28en-us%29.aspx#S5>.

[22] SHAKED, Motti. *.NET Remoting Customization Made Easy: Custom Sinks* [online]. 2003 [cit. 2009-12-22]. Dostupný z WWW: <http://www.codeproject.com/KB/IP/customsinks.aspx>.

[23] *Distributed Application Architecture* [online]. Dostupný z WWW: <http://java.sun.com/developer/Books/jdbc/ch07.pdf>.

[24] *Thin client* [online]. 2010 [cit. 2010-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Thin_client>.

[25] *Fat client* [online]. 2010 [cit. 2010-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Fat_client>.

[26] *Symmetric-key algorithm* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Symmetric-key_algorithm>.

[27] *Public-key cryptography* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Public-key_cryptography>.

[28] *Cryptography 101 for the .NET Framework* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://www.codeproject.com/KB/security/Crypto.aspx?msg=1833034>.

[29] *Key size* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Key_size>.

[30] *Digital signature* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Digital_signature>.

[31] *SQL Injection* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/SQL_injection>.

[32] *Hash function* [online]. 2010 [cit. 2010-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Hash_functionn>.

[33] HANÁČEK, Petr. *Bezpečnost informačních systémů 1* [online]. 2007 [cit. 2010-04-29]. Dostupný z WWW: <https://www.fit.vutbr.cz/study/courses/BIS/private/bis01.pdf>.

[34] HANÁČEK, Petr. *Bezpečnost informačních systémů* 3 [online]. 2007 [cit. 2010-04-29]. Dostupný z WWW: <https://www.fit.vutbr.cz/study/courses/BIS/private/bis03.pdf>.

Seznam příloh

[1] Ukázka aplikace – Seznam přijatých faktur

[2] Ukázka aplikace – Detail faktury

[3] Ukázka aplikace – Detail nákladů zakázky

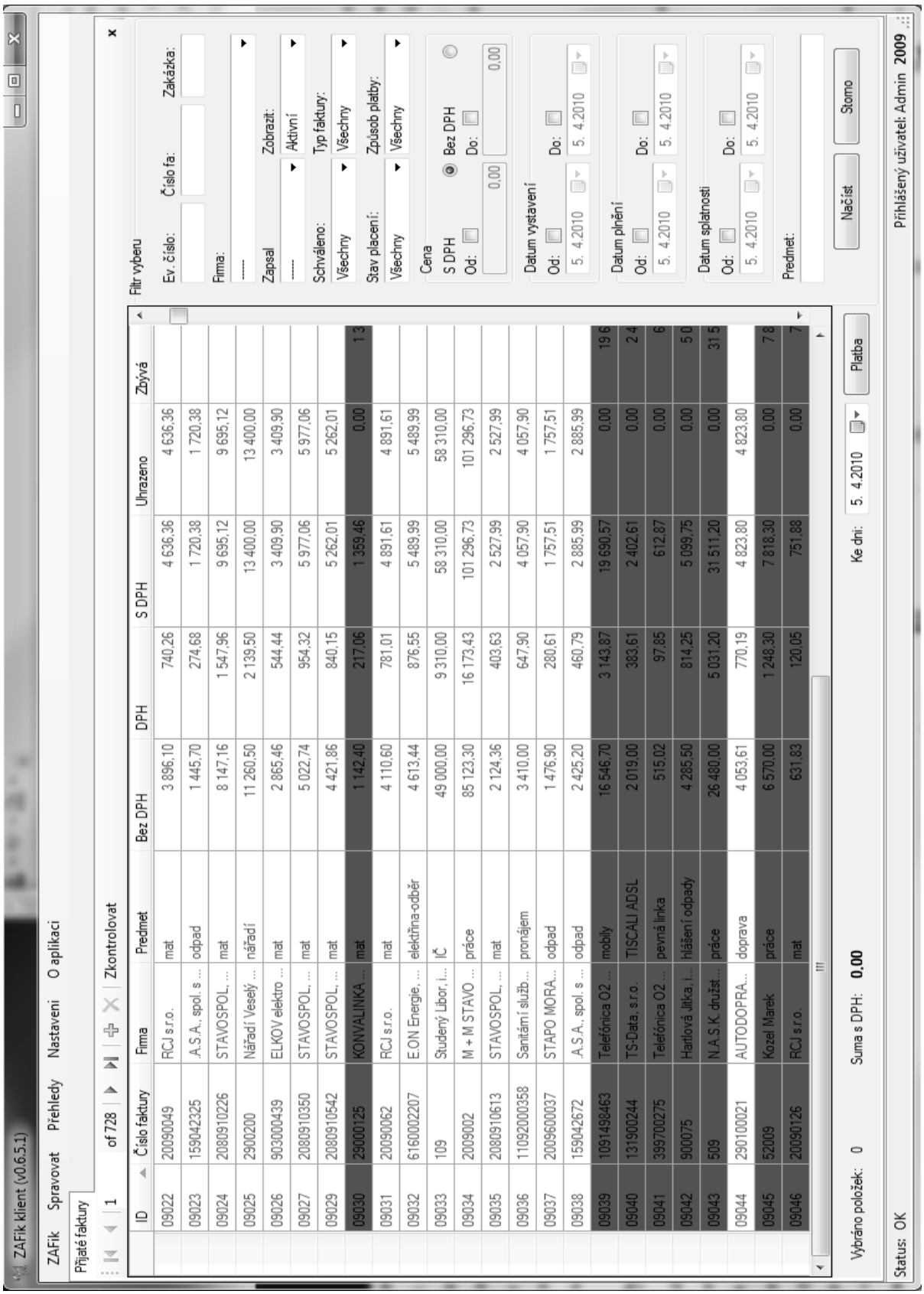
[4] Ukázka aplikace – Administrace serveru

[5] Ukázka aplikace – Detail uživatele

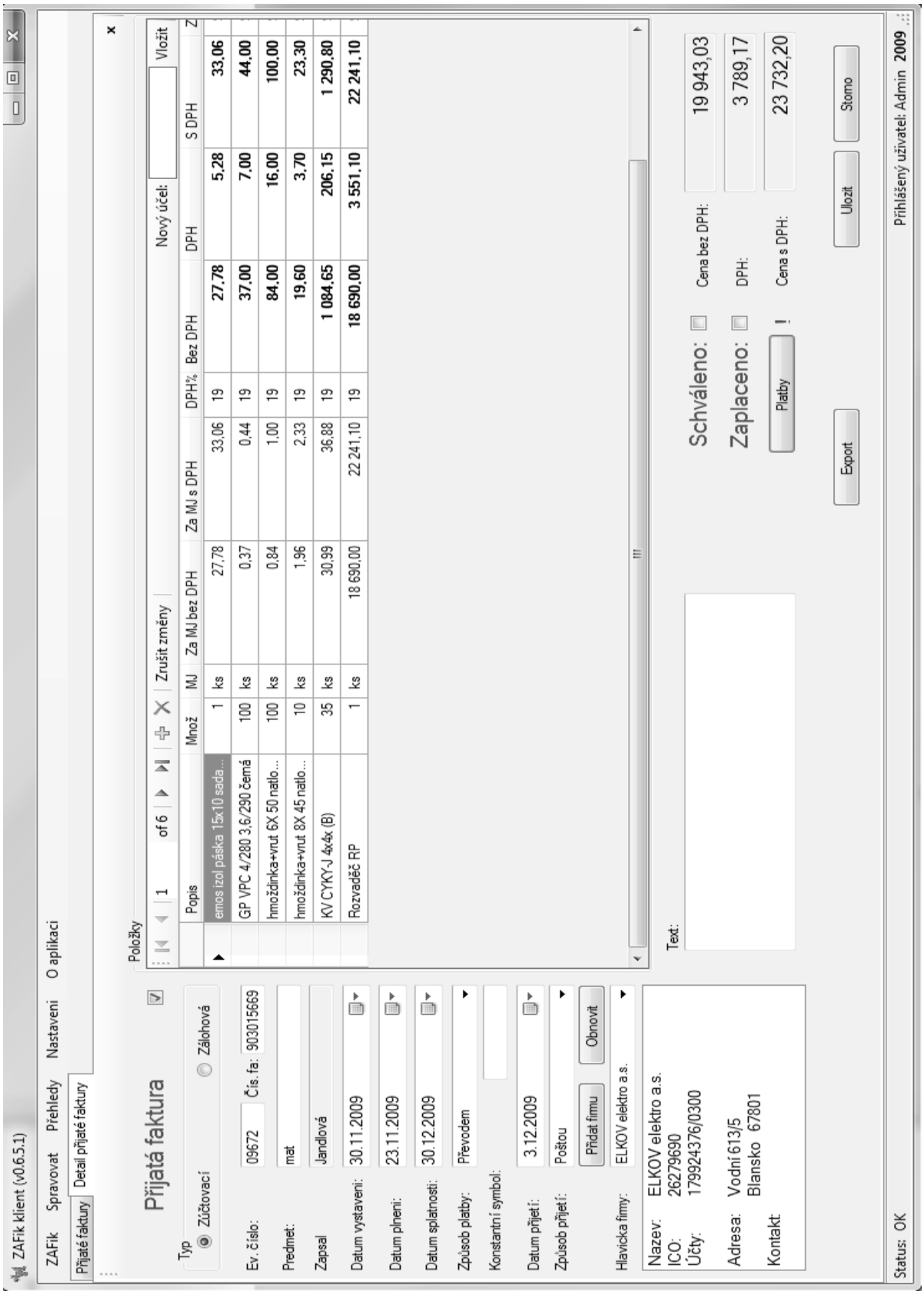
[6] Ukázka aplikace – Detail zakázky

[7] CD se zdrojovými kódy aplikace

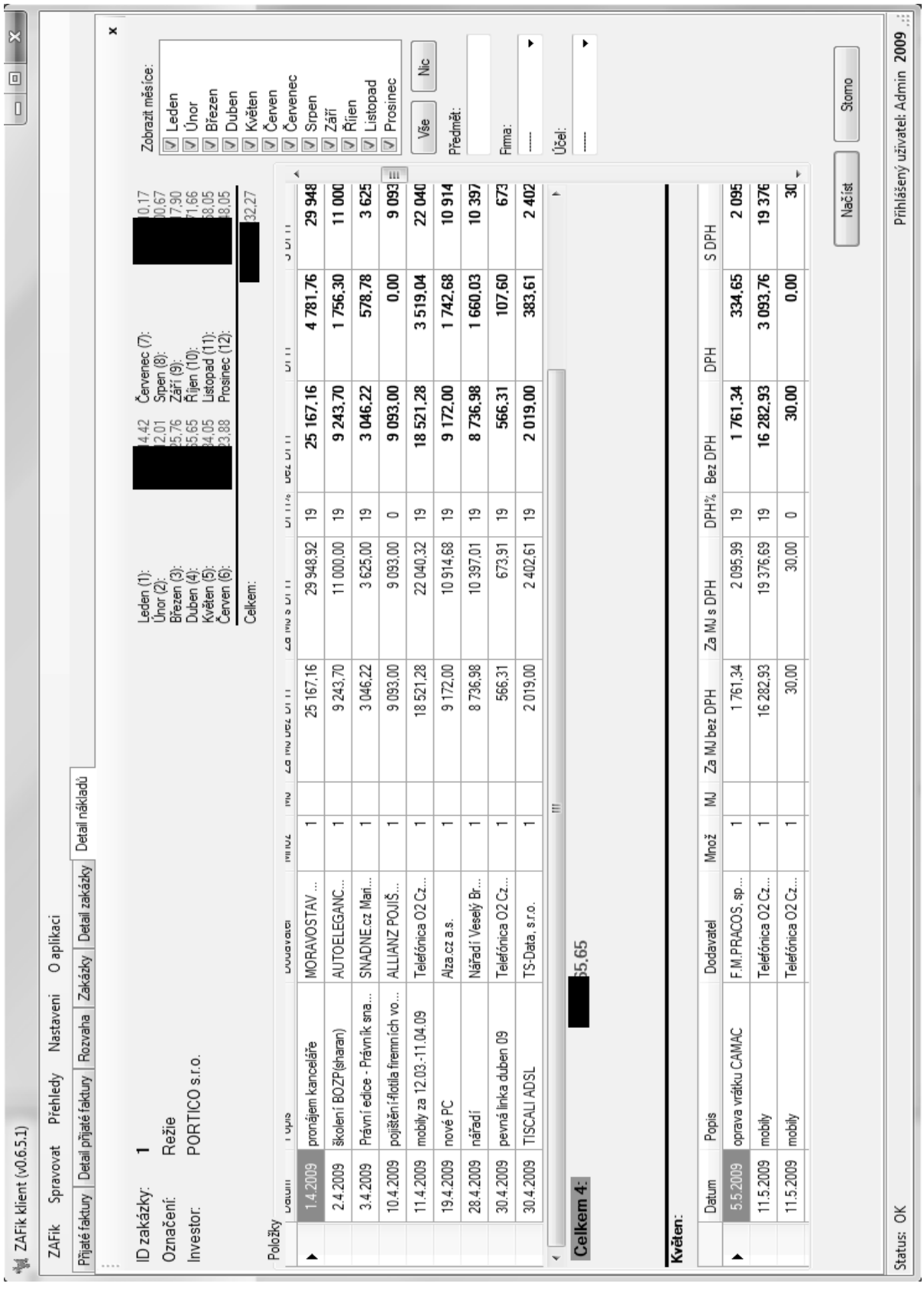
Příloha 1


ZAFik klient (v0.6.5.1)
ZAFik Spravovat Přehledy Nastaveni O aplikaci
Přijaté faktury
1 of 728 Zkontrolovat
Filtr vyberu
Ev. číslo: Číslo fa: Zakázka:
Firma:
Zapsal Zobrazit:
Schváleno: Typ faktury:
Stav placení: Způsob platby:
Cena
S DPH Bez DPH
Od: Do:
Datum vystavení
Datum plnění
Datum splatnosti
Předmět:
Načíst Storno
Vybráno položek: 0 Suma s DPH: 0,00 Ke dni: 5. 4.2010 Platba
Status: OK
Přihlášený uživatel: Admin 2009

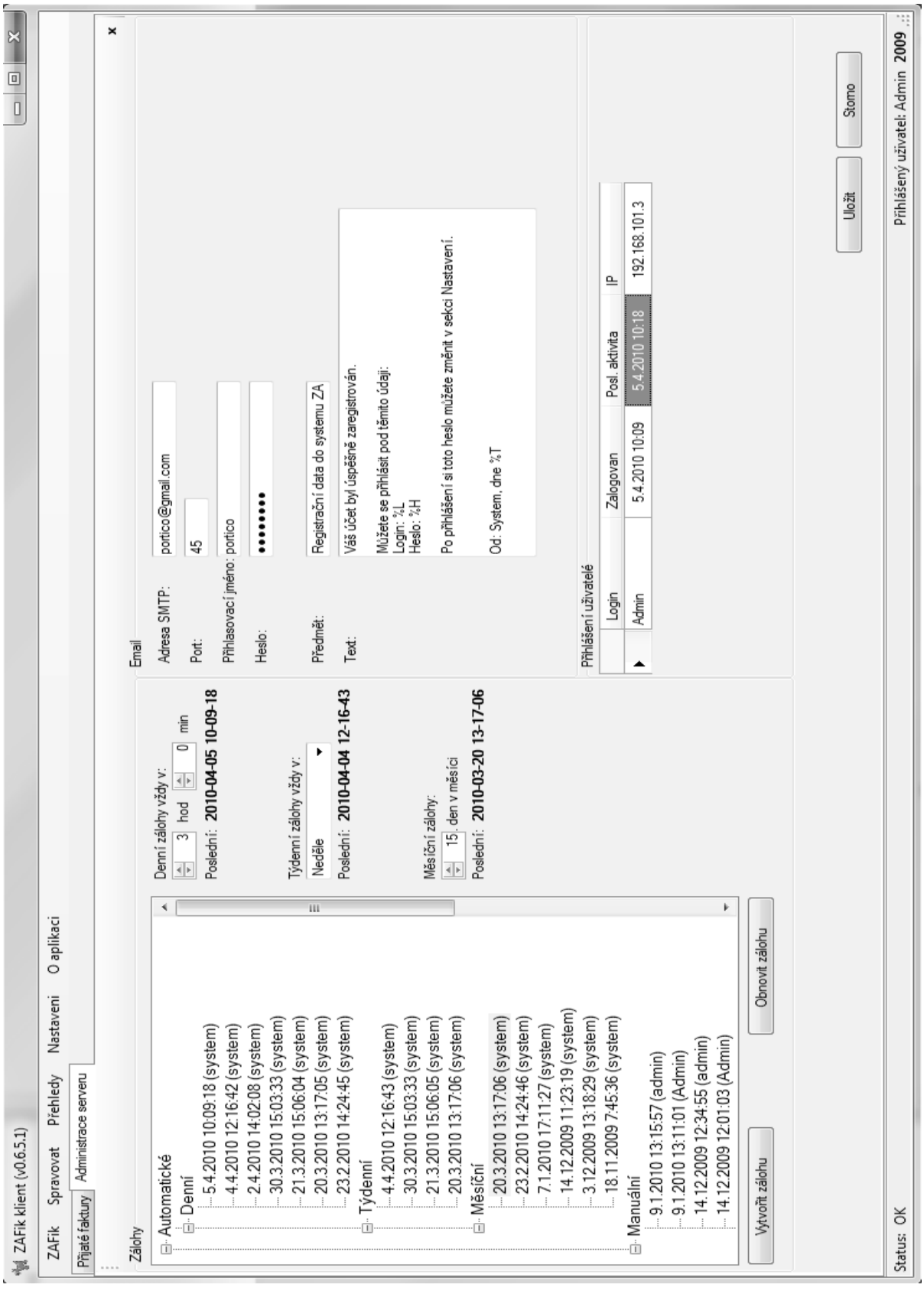
Příloha 2


ZAFik klient (v0.6.5.1)
ZAFik Spravovat Přehledy Nastaveni O aplikaci
Přijaté faktury Detail přijaté faktury
Přijatá faktura
Typ
Zúčtovací
Zálohová
Ev. číslo: 09672 Čís. fa: 9030156669
Předmět: mat
Zapsal Jandlová
Datum vystaveni: 30.11.2009
Datum plneni: 23.11.2009
Datum splatnosti: 30.12.2009
Způsob platby: Převodem
Konstantní symbol:
Datum přijetí: 3.12.2009
Způsob přijetí: Poštou
Přidat firmu
Obnovit
Hlavicka firmy: ELKOV elektro a.s.
Nazev: ELKOV elektro a.s.
ICO: 26279690
Účty: 179924376/0300
Adresa: Vodní 613/5
Blansko 67801
Kontakt:
Položky
1 of 6
Zrušit změny
Nový účel:
Vložit
Popis
Množ
MJ
Za MJ bez DPH
Za MJ s DPH
DPH%
Bez DPH
DPH
S DPH
emos izol páska 15x10 sada...
GP VPC 4/280 3,6/290 černá
hmoždinka+vrut 6X 50 natlo...
hmoždinka+vrut 8X 45 natlo...
KV CYKY-J 4x4x (B)
Rozvaděč RP
Text:
Export
Schváleno:
Zaplaceno:
Platby
Cena bez DPH: 19 943,03
DPH: 3 789,17
Cena s DPH: 23 732,20
Uložit
Storno
Status: OK
Přihlášený uživatel: Admin 2009

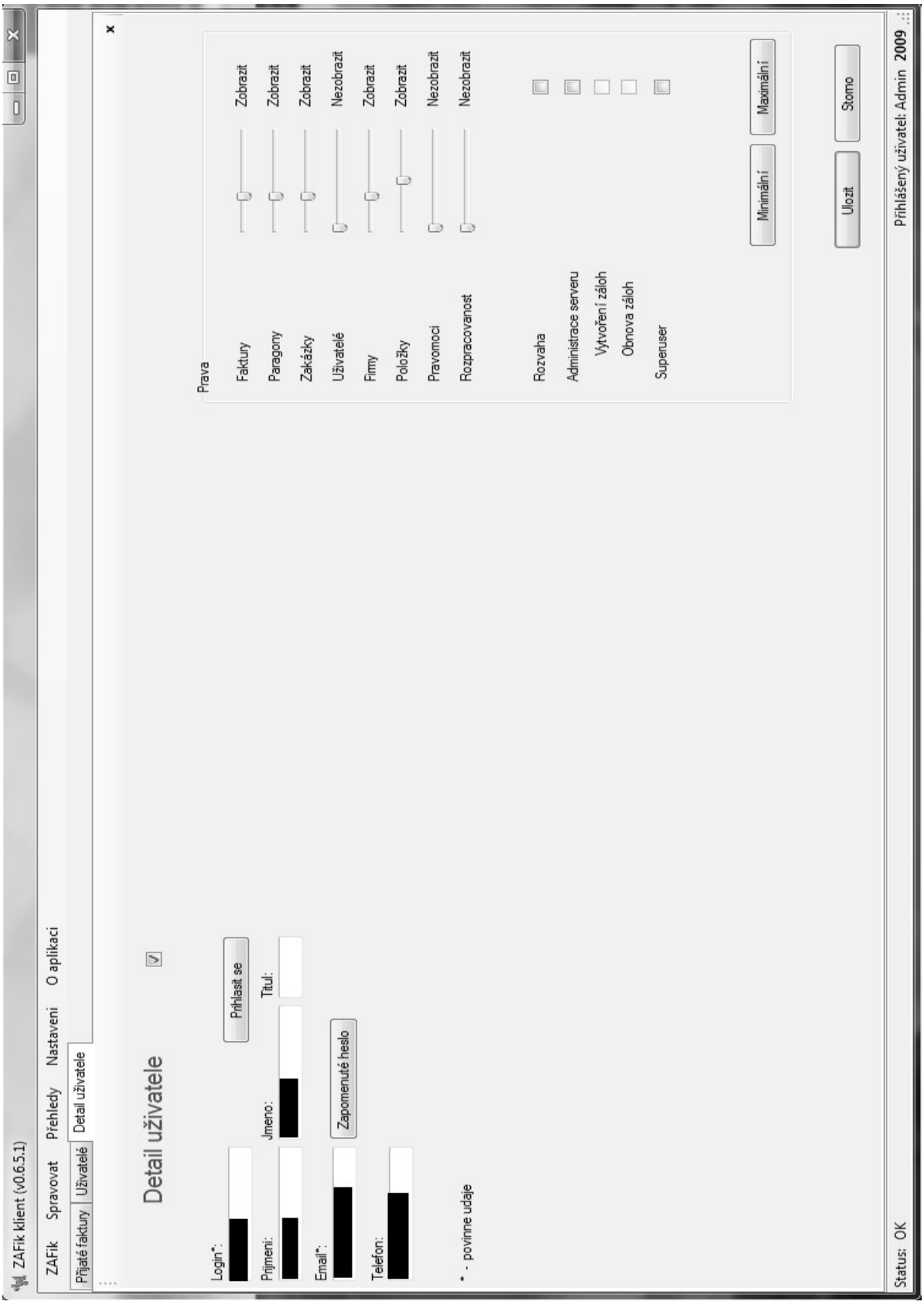
Příloha 3


ZAFik klient (v0.6.5.1)
ZAFik Spravovat Přehledy Nastaveni O aplikaci
Přijaté faktury | Detail přijaté faktury | Rozvaha | Zakázky | Detail zakázky | Detail nákladů
ID zakázky: 1
Označení: Režie
Investor: PORTICO s.r.o.
Leden (1):
Únor (2):
Březen (3):
Duben (4):
Květen (5):
Červen (6):
Červenec (7):
Srpen (8):
Září (9):
Říjen (10):
Listopad (11):
Prosinec (12):
Celkem:
Zobrazit měsíce:
Leden
Únor
Březen
Duben
Květen
Červen
Červenec
Srpen
Září
Říjen
Listopad
Prosinec
Vše
Nic
Předmět:
Firma:
Účel:
Položky
Celkem 4:
Květen:
Načíst
Storno
Status: OK
Přihlášený uživatel: Admin 2009

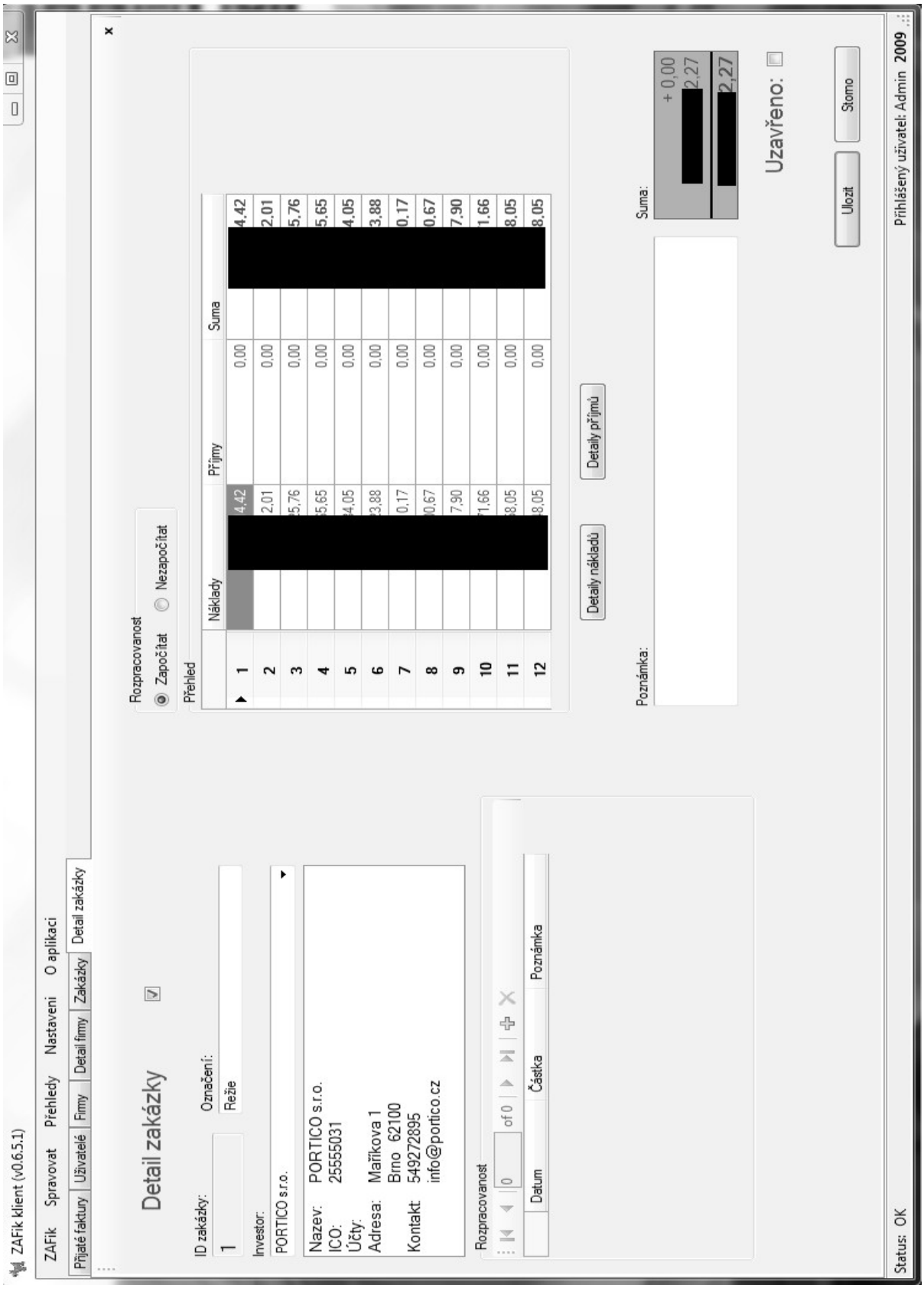
Příloha 4


ZAFik klient (v0.6.5.1)
ZAFik
Spravovat
Přehledy
Nastaveni
O aplikaci
Přijaté faktury
Administrace serveru
Zálohy
Automatické
Denní
5.4.2010 10:09:18 (system)
4.4.2010 12:16:42 (system)
2.4.2010 14:02:08 (system)
30.3.2010 15:03:33 (system)
21.3.2010 15:06:04 (system)
20.3.2010 13:17:05 (system)
23.2.2010 14:24:45 (system)
Týdenní
4.4.2010 12:16:43 (system)
30.3.2010 15:03:33 (system)
21.3.2010 15:06:05 (system)
20.3.2010 13:17:06 (system)
Měsíční
20.3.2010 13:17:06 (system)
23.2.2010 14:24:46 (system)
7.1.2010 17:11:27 (system)
14.12.2009 11:23:19 (system)
3.12.2009 13:18:29 (system)
18.11.2009 7:45:36 (system)
Manuální
9.1.2010 13:15:57 (admin)
9.1.2010 13:11:01 (Admin)
14.12.2009 12:34:55 (admin)
14.12.2009 12:01:03 (Admin)
Vytvořit zálohu
Obnovit zálohu
Denní zálohy vždy v:
3
hod
0
min
Poslední: 2010-04-05 10-09-18
Týdenní zálohy vždy v:
Neděle
Poslední: 2010-04-04 12-16-43
Měsíční zálohy:
15. den v měsíci
Poslední: 2010-03-20 13-17-06
Email
Adresa SMTP:
portico@gmail.com
Port:
45
Přihlasovací jméno:
portico
Heslo:
Předmět:
Registrační data do systemu ZA
Text:
Váš účet byl úspěšně zaregistrován.
Můžete se přihlásit pod těmito údaji:
Login: %L
Heslo: %H
Po přihlášení si toto heslo můžete změnit v sekci Nastavení.
Od: System, dne %T
Přihlášení uživatelé
Login
Zalogovan
Posl. aktivita
IP
Admin
5.4.2010 10:09
5.4.2010 10:18
192.168.101.3
Uložit
Storno
Status: OK
Přihlášený uživatel: Admin 2009

Příloha 5


ZAFik klient (v0.6.5.1)
ZAFik
Spravovat
Přehledy
Nastaveni
O aplikaci
Přijaté faktury
Uživatelé
Detail uživatele
Detail uživatele
Login*:
Přihlasit se
Příjmení:
Jméno:
Titul:
Email*:
Zapomenuté heslo
Telefon:
* - povinné údaje
Práva
Faktury
Zobrazit
Paragony
Zobrazit
Zakázky
Zobrazit
Uživatelé
Nezobrazit
Firmy
Zobrazit
Položky
Zobrazit
Pravomoci
Nezobrazit
Rozpracovanost
Nezobrazit
Rozvaha
Administrace serveru
Vytvoření záloh
Obnova záloh
Superuser
Minimální
Maximální
Uložit
Storno
Status: OK
Přihlášený uživatel: Admin 2009

Příloha 6


ZAFik klient (v0.6.5.1)
ZAFik
Spravovat
Přehledy
Nastaveni
O aplikaci
Přijaté faktury
Uživatelé
Firmy
Detail firmy
Zakázky
Detail zakázky
Detail zakázky
ID zakázky:
1
Označení:
Režie
Investor:
PORTICO s.r.o.
Nazev: PORTICO s.r.o.
ICO: 25555031
Účty:
Adresa: Mařikova 1
Brno 62100
Kontakt: 549272895
info@portico.cz
Rozpracovanost
Započítat
Nezapočítat
Přehled
Náklady
Příjmy
Suma
Detaily nákladů
Detaily příjmů
Poznámka:
Suma:
+ 0,00
2,27
2,27
Uzavřeno:
Uložit
Storno
Rozpracovanost
Datum
Částka
Poznámka
Status: OK
Přihlášený uživatel: Admin 2009