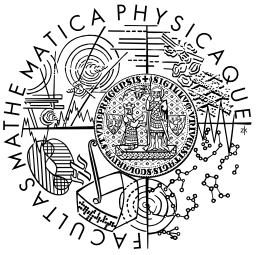


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Jan Navrátil

Integrace legacy databází do soudobých informačních systémů

Katedra softwarového inženýrství

Vedoucí diplomové práce:	RNDr. Michal Kopecký, Ph.D.
Studijní program:	Informatika
Studijní obor:	ISS

Praha 2016

Poděkování:

Děkuji vedoucímu své diplomové práce, RNDr. Michalovi Kopeckému, Ph.D., za konzultace a odbornou pomoc při koncipování práce. Děkuji firmě Ryston Electronics s r.o., zejména Ing. Petru Pokornému a Robertu Pěknicovi z firmy Ryston Electronics s.r.o. za poskytnutí úryvků zdrojových kódů a konzultace při řešení problémů. Hlavně děkuji svým rodičům, své partnerce Barboře a celé rodině za pochopení a velkou trpělivost.

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 28.7.2016 Jan Navrátil

Název práce: Integrace legacy databází do soudobých informačních systémů

Autor: Jan Navrátil

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Michal Kopecký, Ph.D.

Abstrakt:

Cílem této práce je navrhnout a implementovat framework pro podporu přístupu k Legacy systémům, tedy k databázím, využívajícím nekompatibilní zastaralé technologie, které přesto nelze jednoduše opustit. Framework umožňuje abstrakci aplikační logiky od databázové platformy a umožní v budoucnu úplný či inkrementální přechod na novou, moderní platformu. Návrh zahrnuje také případné zapouzdření fungující Legacy aplikace a její integraci do nového systému v podobě černé skříňky. Framework je vysoce konfigurovatelný a nezávislý na kontextu nebo databázové platformě. Systém byl pilotně nasazen ve firmě, kde úspěšně funguje. Umožnil přechod firmy na nový informační systém se zcela odlišnou databázovou platformou. Praxe ukazuje, že návrh frameworku je životaschopný.

Klíčová slova: legacy systém, legacy databáze, integrace dat, návrh frameworku, Visual FoxPro

Title: Integration of legacy databases to current information systems

Author: Jan Navrátil

Department: Department of Software Engineering

Supervisor: RNDr. Michal Kopecký, Ph.D.

Abstract:

The goal of this thesis is to design and implement a framework to support Legacy system access. Legacy systems are databases that use incompatible and obsolete technologies and can not be easily abandoned. The framework allows the abstraction of application logic from database platform and will enable full or incremental migration to a new, modern platform in the future. The framework also considers the option of encapsulation of an existing legacy application into the new system as a black box. The framework is well configurable and extendable and is independent on database platform or data context. A system based on proposed framework has been succesfully deployed in a company. The system facilitated the migration of the company to a new information system with an entirely different database platform. The practice shows the viability of the framework design.

Keywords: legacy systems, legacy databases, data integration, framework design, Visual FoxPro

Obsah

Úvod

Současná doba si žádá, aby každá firma, škola, úřad či jiná organizace měla informační systém. Velikost, funkcionalita a cena takového systému se odráží v potřebách a možnostech organizace a pokud si systém nevyvine sama, je na trhu široká paleta komerčních i otevřených řešení. Některé instituce mají svůj systém již dlouho. Pokud se však tento systém náležitě neudržuje a nemodernizuje, může se časem stát spíše přítěží. Cílem této práce je právě odstranění problémů s používáním zastaralých systémů a zjednodušení přechodu na moderní systém.

0.1 Legacy systémy

Pojemem **Legacy systém** se označuje stará metoda, technologie, počítačový systém nebo program, který je i přes svou zastaralost používán "z historických důvodů". Takový systém typicky znemožňuje plynulý přechod na nové technologie a stává se brzdou v rozvoji infrastruktury firmy.

Ačkoli byl systém kdysi uspokojivě funkční, rostoucí nároky postupně přesáhly jeho možnosti. Nejedná se ani tak o hardwarové nedostatky, jako je nízký výpočetní výkon, nedostatečná kapacita úložiště nebo pomalá konektivita, ale především o softwarovou zastaralost, jakou je použití slepé vývojové větve technologie, uzavřeného systému, neefektivita práce nad velkými daty a další.

Systém je tak žádoucí nahradit modernějším, flexibilnějším, rozšiřitelnějším a v neposlední řadě vývojáři stále podporovaným řešením. Přejít na nový systém najednou však může být velký problém. Můžeme narazit na nejrůznější typy překážek:

Nikdo se v aplikaci nevyzná
: Tato situace nastává, když původní vývojář opustí firmu nebo poskytovatel systému (firma) zkrachuje nebo z jiného důvodu odmítá nadále poskytovat své služby. Dokumentace neexistuje a zdrojové kódy, pokud jsou vůbec k dispozici, jsou nedostatečně komentované.

Aplikace je monolitická
: Vazby uvnitř apliakce jsou vzájemně propletené do té míry, že systém nejde modernizovat po částech. Tato situace často nastává dlouhodobým záplatováním a doplňováním bez celkového konceptu.

Aplikace je závislá na vnějším faktoru
: Např. když firma nechce nebo nemůže přejít na jinou technologii z licenčních, právních, strategických nebo finančních důvodů.

Kromě celých aplikací se vyskytují také **legacy databáze**, představující podobný problém na datové úrovni. Bývají technologicky zastaralé a existují-li pro ně vůbec moderní adaptéry, zaostávají ve flexibilitě nebo výkonu.

0.2 Cíl práce

Cílem tohoto projektu je proto navrhnout modulární a rozšiřitelné řešení popsané situace. Framework, který dovede propojit staré technologie s novými. Jeho struktura by měla být objektová, pokud možno robustní a přizpůsobitelná libovolnému použití. Framework by měl izolovat business logiku od datové vrstvy tak, aby usnadnil odstranění problematických elementů a umožnil plynulý přechod na nové technologie.

Myšlenkou frameworku je skrýt a sjednotit reprezentaci dat, se kterými aplikace pracují. Proto je zvolen objektový přístup, který dokáže obsáhnout data v SQL i no-SQL databázích, stejně jako výstupy z legacy aplikací.

0.3 Osnova práce

Obsah této práce je logicky rozdělen do dvou částí. První část se věnuje návrhu obecného řešení, druhá se zabývá nasazením navrhovaného řešení v pilotním projektu.

V kapitole 1 je provedena analýza problému legacy systémů představených zde v úvodu a diskutuje jejich zapojení do soudobých aplikací. Je uvedena souvislost s obecným databázovým přístupem a je představeno několik existujících řešení, která se zabývají oddělením aplikační logiky od formátu dat a databáze (ORM). Kapitola 2 obsahuje analytickou část případové studie na reálné situaci v české firmě. Představuje legacy informační systém a další software přítomný ve firmě. Specifikuje požadavky na novou aplikaci pro webovou službu, licenční a technologické podmínky pro vývoj a provoz.

V kapitole 3 je představen návrh frameworku, který by měl umožnit sjednocení různorodých databázových platforem a wrapperů pro legacy systémy. Popisuje předpokládaný tok dat a vhodné rozhraní. Kapitola 4 se pak zabývá implementací návrhu do konkrétní podoby. Rozebírá podobu jádra a objektového rozhraní a realizaci vzorových modulů. V kapitole 5 je popsána implementace požadovaného díla na základě navrhovaného frameworku.

1. Analýza

V praxi se často setkáme s tzv. *legacy systémy*, jak bylo rozebráno v úvodu. Takové systémy přesluhují, ale je příliš obtížné je nahradit něčím novým. Často však v sobě skrývají "know-how" firmy, nebo se jednoduše neví, jak vlastně pracují. Takový systém lze zkoumat a analyzovat[12], ale tento proces je nesmírně náročný a hlavně pro každý systém individuální.

V této práci je rozebírána varianta zakomponování legacy systému jako celku do moderních softwarových celků. Je třeba rozlišit dva základní druhy legacy systémů. Prvním jsou **legacy databáze**. Jedná se o zastaralé nebo slepé vývojové větve databázových technologií (například xBase FoxPro), které byly překonány a už nejsou podporovány. Druhým typem jsou **legacy aplikace**, tedy celé softwarové systémy nebo aplikace, které obsahují netriviální logiku a fungují jako celek. Tato práce bere v úvahu obě varianty.

Právě proto, že se jedná o zapomenuté nebo přinejmenším nepodporované technologie, jejich kombinace a další nestandardní situace, nelze očekávat, že by existoval nějaký univerzální nástroj pro jejich integraci do moderních systémů. Snad jedinou výjimkou jsou webové aplikace, pro něž existuje projekt *Selenium WebDriver*[11]. Aplikaci je tedy třeba individuálně poskytnout nějaké přístupové rozhraní, kterým by mohla komunikovat.

Pozornost je třeba věnovat databázovým technologiím. Pro většinu dohledatelných technologií existuje nějaká implementace přístupového API (ODBC, SOAP...), je tedy třeba je využít a zakomponovat do celku.

Pro zaintegrování legacy systémů, resp. jejich wrapperů, do současných systémů je potřeba sjednotit paradigma, s nímž dokáží systémy pracovat. Jako nejvhodnější prostředek se jeví **objekty**, protože jsou flexibilní a dokáží být dostatečně obecné pro libovolný koncept dat. V důsledku tedy hledáme způsob, jak mapovat data z různých platforem do objektů.

1.1 Existující řešení

Obecně je třeba očekávat různou povahu dat, tedy jak relační tak nerelační. Koncept transformace relačních dat do objektů se nazývá *Object-Relation Mapping (ORM)*. Tento proces je běžně využíván a existuje celá řada děl, která se jím zabývají.

Významným hráčem je Javový framework *Hibernate ORM*, spolu se svým portem do .NET *NHibernate*[1]. Oproti dalšímu kandidátu, *Entity Framework* má řadu předností, jak je popsáno v článku [7]. Další možnosti selhávají buď pro omezenou podporu platforem nebo kvůli nevyhovujícím licenčním podmínkám použití (Oracle TopLink). Náskok získal NHibernate především šířkou podpory různých databázových platforem a propracovanosti možností definice mapování.

Pro nerelační, takzvané *NoSQL* databázové platformy je objektový přístup často přímo podporován (ObjectDB, Perst, OrientDB, MongoDB ...).

Nepodařilo se mi najít software, který by umožnil agregaci různorodých modulů do jediného celku. Možná protože se pouze vzácně vyskytne situace složitější než hlavní databáze + speciální případ navíc a vývojáři jsou zahlceni obtížemi konkrétního případu. Cílem projektu se tedy stalo navrhnout mnohostranně použitelný, modulární, objektově orientovaný systém, umožňující pro-

pojení aplikační logiky postavené na moderních technologiích s nejen soudobými, ale i legacy systémy a databázemi. Systém by měl právě svou modularitou a objektovou abstrakcí usnadnit přechod od zmíněných zastaralých technologií na moderní.

2. Případová studie

V této kapitole je provedena případové studie. Zahrnuje analýzu situace a požadavky zadavatele na její řešení, načež je v kapitole 5 popsána pilotní implementace.

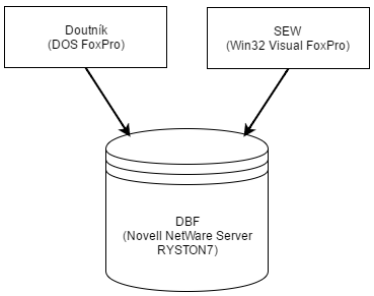
Pilotní firmou je menší česká firma, zabývající se velkoobchodem s elektronickými součástkami a zakázkovým vývojem elektroniky. Firma má zastaralý informační systém, a chce rozšířit nabídku služeb o webový portál.

2.1 Analýza stávajícího prostředí

2.1.1 Historie

Pro svůj interní informační systém pilotní firma dosud využívá zastaralý DOSovský in-house program *Doutník*, napsaný ve FoxPro. Ten funguje nad databází v podobě DBF tabulek verze 2.5 a neexistuje k němu ucelená dokumentace.

V minulosti byl proveden pokus o modernizaci systému vytvořením programu *Sklad Elektroniky Windows (SEW)*. SEW je napsaný ve Visual FoxPro, což mělo umožnit opuštění DBF tabulek, protože novější Visual FoxPro dokáže pracovat i s moderními SQL databázemi. Celý systém měl být převeden do programu SEW, což se dělo postupným přesouváním funkcionality, ale dělo se tak velmi pomalu. Důvodem bylo souběžné přidávání nové funkcionality a "vylepšování" programu. Poté, co klíčový vývojář firmu opustil, zamrzl systém na mrtvém bodě. Nyní tedy existují dva programy s různou funkcionalitou, které se oba používají a navzájem se doplňují (ilustrováno na obrázku 2.1).



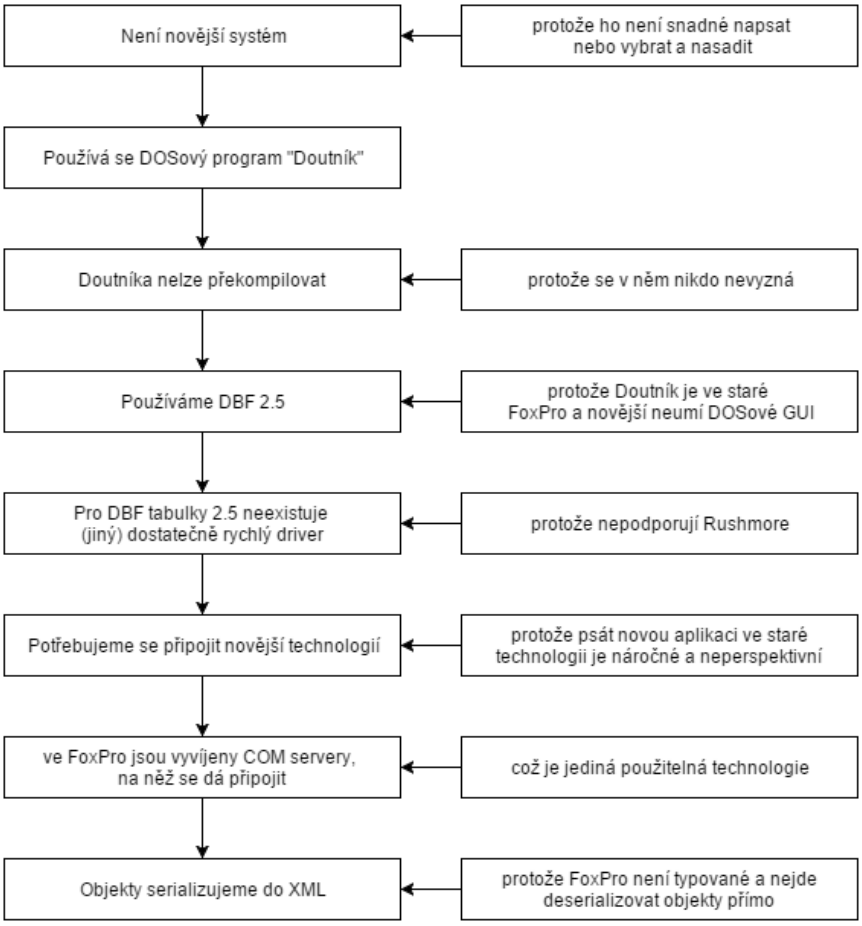
Obrázek 2.1: Schéma informačního systému v podobě programů Doutník a SEW. Oba programy používají tutéž DBF databázi ve verzi 2.5, poslední podporovaná starou DOSovou FoxPro.

Vývojář, který měl *Doutník* a *SEW* na starosti, nezanechal žádnou použitelnou dokumentaci, čímž ztratila firma možnost program Doutník opustit. SEW byl, jako nástupní software, psaný přehledněji a bylo ho do jisté míry možno dále rozvíjet. Pro Doutník nebyl k dispozici ani proces správného překompilování, a tak zůstal program nadále bez jakékoli změny.

Vedle aplikací *Doutník* a *SEW* se na pracovních stanicích používá celá řada malých, specializovaných aplikací. Tyto aplikace byly psány v průběhu let na míru specifickým problémům,

které nebylo možno řešit úpravou SEWu. Jsou napsány většinou ve Visual FoxPro, a to buď to celé nebo jejich podstatné části. Jejich autorem a správcem je jediný vývojář a dokumentace je chabá. Představují riziko a jsou součástí balastu zastaralé technologie.

Aplikace Doutník a SEW tvoří společně *Legacy systém*, jak je definován v úvodu. Logickým řešením by bylo napsat anebo zakoupit moderní systém, který by obě aplikace plně nahradil. Tento proces je ale velmi zdlouhavý, protože znamená překonání celé řady problémů. Hlavní problém je absence podrobné procesní a systémové analýzy a to především proto, že nikdo nedovede specifikovat, co všechno má systém umět.



Obrázek 2.2: Řetězec příčinných souvislostí legacy systému.

Proti napsání vlastního systému ve firmě existuje několik pádných argumentů:

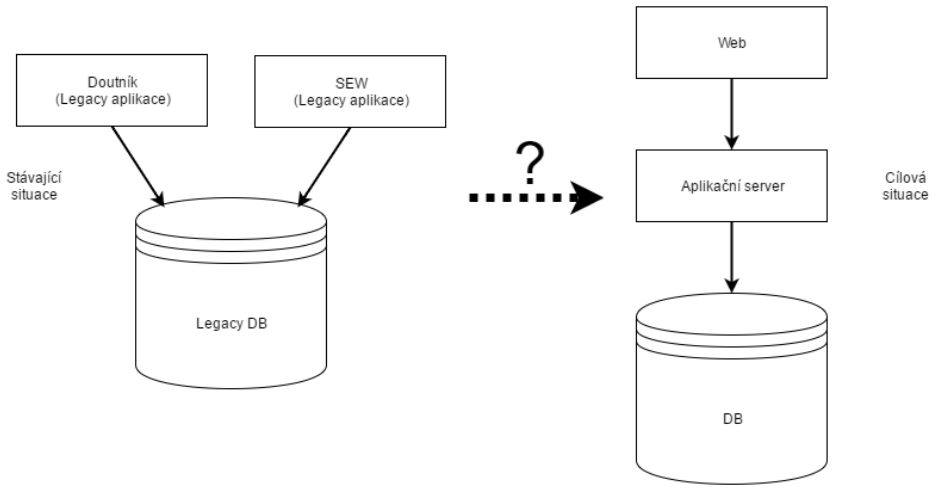
1. V minulosti se to neosvědčilo.

2. Nejsou na to pracovní síly.

Vývoj vlastního systému by tedy trval velmi dlouho a vývojáři by museli opustit svoje ostatní činnosti. Najmutí nových pracovníků nemá smysl. Bylo by lepší najmout si celou firmu, přednostně s hotovým systémem.

V průběhu psaní této diplomové práce se firma rozhodla nasadit nový, proprietární informační systém *Flores* [1], klon programu Abra, který funguje nad databázovou platformou *Firebird SQL*. Učinila tak v době, kdy už byla nasazena do provozu aplikace, postavená na zde navrženém frameworku. To se stalo jedním z prubířských kamenů použitelnosti frameworku. Stačilo framework rozšířit o nový backend modul pro podporu nové databázové platformy. Programu Flores je věnována pasáž na konci této sekce.

Jak je popsáno výše, interní firemní systém je DOSovský program, psaný ve FoxPro, s nadstavbou a částečnou náhradou napsanou ve Visual FoxPro. Přestože byly podnikuty kroky k modernizaci, i ona "novější" technologie je dnes již zastaralá[4] a je potřeba překlenout obě aplikace jako celek.

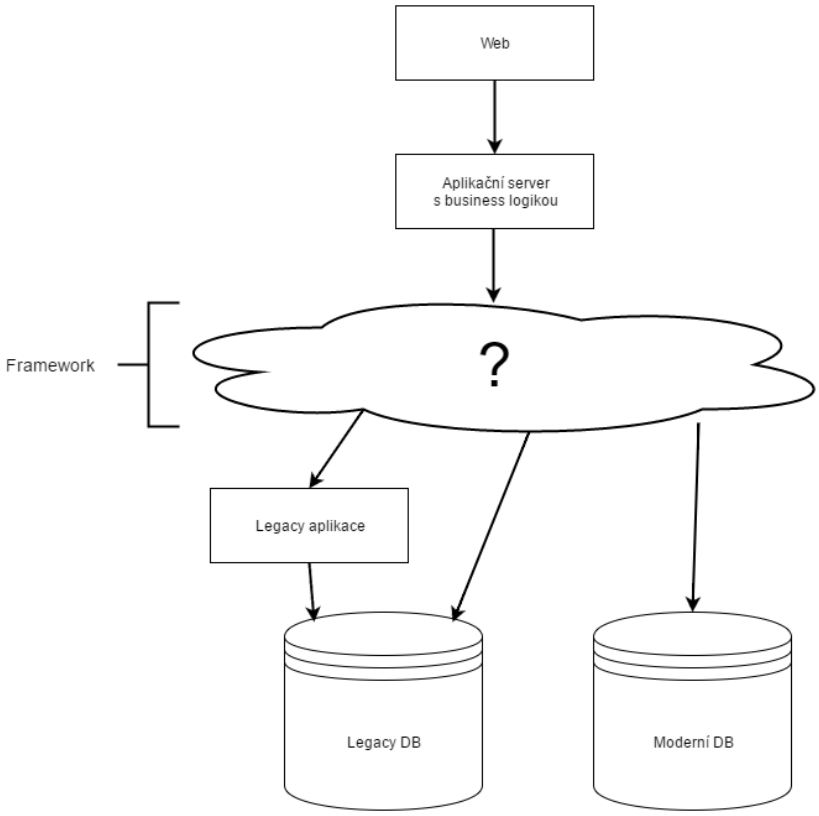


Obrázek 2.3: Vizualizace překlopení schémat z legacy aplikace na moderní. Vlevo je zobrazena stávající stituace která neumožňuje použití moderní prezentační vrstvy. Naznačena je tendence transformovat situaci do podoby, jak jsou budovány moderní aplikace (vpravo).

Z řetězce na obrázku 2.2 plyne, že je potřeba najít vhodnou nadstavbu nad platformou databázových tabulek tak, aby umožnila modernějším technologiím v jakékoli aplikaci pracovat s daty. Vize plánovaného přechodu na jiný systém s modernější platformou motivuje k navržení řešení modulárně tak, aby bylo možno konkrétní databázový adaptér jednoduše vyměnit za jiný. Schéma 2.3 ilustruje plánovaný přesun od současného stavu ke struktuře moderní aplikace.

[1] `http://www.floresps.cz/`

Firma nemá ve svém softwarovém vybavení implementovanou požadovanou funkcionalitu pro webovou službu, takže v pilotním nasazení nebude potřeba ovládat existující legacy aplikaci. Řešení by ale mělo být navrženo tak, aby tuto možnost podporovalo. Představa řešení v tuto chvíli by se dala vyjádřit obrázkem 2.4.



Obrázek 2.4: Abstraktní představa řešení situace ve firmě. Cílem je vytvořit framework tak, aby mohly být technologie používány najednou a aby byl možný přesun od stávající situace s pouze legacy systémem k situaci zcela bez legacy technologií.

2.1.2 SW vybavení firmy

Na všech stanicích, kde se pracuje s programem Doutník, je nainstalován systém *Microsoft Windows XP*. Na stanicích, kde se používá pouze SEW, může být i novější verze systému Windows. Tyto systémy bohužel nepodporují 32 bitové běhové prostředí FoxPro, které vyžaduje Doutník. Znamená to, že řešení musí být kompatibilní se systémem Windows XP. Firma neplánuje měnit platformu na např. Linux.

Firma má zakoupeny licence na software a vývojové nástroje, jako jsou *Microsoft Visual Studio* přes účet (subscribtion) na MSDN. Preferuje tedy použití těchto technologií, aby udržela svůj software v jednotné podobě a mohla jej bez zbytečných komplikací spravovat. Z licenčních důvodů nechce firma používat technologii *Internet Information Services (IIS)*.

2.1.3 Databáze

Jak už bylo uvedeno výše, informační systém je postaven na platformě FoxPro (*xBase*), která ukládá data do tabulek v souborech s příponou ".DBF". Tyto soubory jsou umístěny na síťovém úložišti, *Novell NetWare Server*. S touto databází pracují i ostatní programy, které využívají data z informačního systému. Ačkoli proběhl pokus o migraci na modernější platformu Visual FoxPro, nepodařilo se eliminovat program Doutník a ten vyžaduje ke svému běhu starou verzi DBF tabulek 2.5 (pro DOS, před povýšením na Visual). Pro tuto verzi neexistuje dostatečně rychlý, moderní konektor, neboť se firma Microsoft po zakoupení technologie FoxPro rozhodla nevyvíjet zpětnou kompatibilitu a zaměřila se na vývoj vpřed.

K Doutníku existují zdrojové kódy, ale ví se, že kompilace a vydání (release) je netriviální proce, který ovládal pouze bývalý vývojář. Systém, který nelze překompilovat, ale který je kriticky citlivý na nekonzistence v databázi, je nesmírně křehký. Z toho důvodu firma odmítá jakékoli úpravy struktury databáze, protože by mohl zhavarovat celý systém a zastavila by se práce. Tento fakt je důležitý především pro situaci, kdy bude potřeba přidat data, například pro zavedení uživatelských účtů.

Odpojování tabulek

Jedním z důsledků používání databázového systému na bázi DBF tabulek, respektive mechanismů pro jejich používání je, že dochází k údržbové odstávce databáze. Odstávka databáze obnáší odpojení serveru, aby se zamezilo konfliktnímu otevření tabulek.

Důvodem pro údržbu je v prvé řadě přepočítání indexů, které vyžaduje exkluzivní přístup k tabulkám. Souběžným používáním DOSové FoxPro a Windowsové Visual FoxPro dochází k porušení indexových souborů ".cdx". Tento problém je znám, replikovatelný, ale neodstranitelný. Přepočítání indexů se provádí plánovaně jednou týdně.

Nad důležitými tabulkami, ve kterých se indexy využívají při hledání, je indexů i několik desítek. Právě tyto důležité tabulky jsou velmi velké - statisíce až miliony záznamů, např. neustále rostoucí historie nabídek dodavatelů. Jejich přepočítání tak může zabrat i hodinu. Během odstávky databáze musí být všechny služby odpojeny.

2.1.4 Přechod na Flores

V roce 2014 nasadila firma zakoupený informační systém Flores. Tento přechod obnášel migraci dat do databáze Firebird SQL s předem danou strukturou. Podařilo se opustit dvojici programů Doutník a SEW a nahradit jejich používání aplikací Flores, ale neobešlo se to bez ztráty funkcionality. Protože systém Flores nenabízel pro systém Online podporu, přibyla do požadavků na framework podpora DB Firebird.

Platforma Firebird SQL je otevřená platforma a je hojně využívaná (zajímavé srovnání s MySQL nabízí P. Gulutzan ve svém článku [5]). Je dokonce podporována frameworkem NHibernate.

Po roce byl z licenčních důvodů provoz systému Flores ukončen. Následovala migrace zpět do DBF tabulek a návrat k duu Doutník a SEW. Došlo při tom ke ztrátě nebo porušení některých

dat.

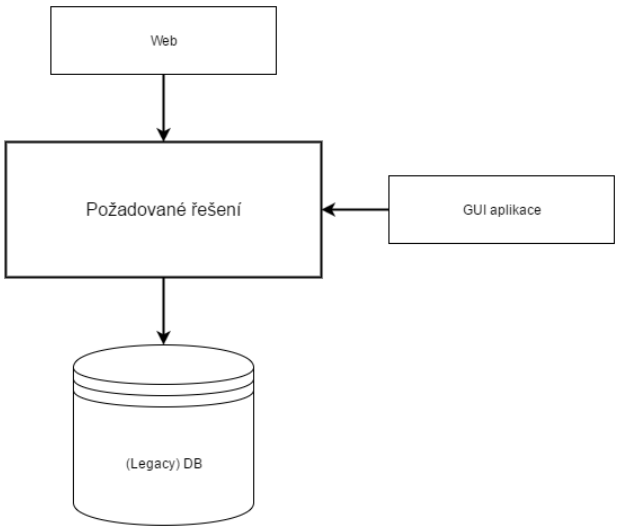
2.2 Specifikace

V této sekci jsou popsány požadavky na nový systém s pracovním názvem *Online*. Jedná se o přístup k informacím interního informačního systému přes webové rozhraní. V kapitole jsou uvedeny požadavky na funkcionalitu, dostupné nebo požadované technologie a předpokládané chování v kritických situacích.

2.2.1 Systém Online

Systém Online je aplikací navrhovaného frameworku. Požadovanou funkcionalitou je zpřístupnění některých obchodních informací pro zákazníky na webových stránkách. Do budoucna by to měl být vlastně e-shop, ale v současnosti je koncept omezen na "readonly" přístup, aby ani vzdáleně nehrozilo porušení databáze. Samotné webové rozhraní bylo vyvinuto již dříve v rámci firmy.

Schéma 2.5 znázorňuje hrubou představu systému. Jak bylo uvedeno výše, funkcionalita systému Online není implementována v žádné existující *legacy aplikaci*, takže ji nebude třeba zakomponovat do návrhu jako stavební prvek.



Obrázek 2.5: Základní hrubé schéma požadované podoby systému Online. Mělo by být shora přístupné pro web, pracovat na obecné databázové platformě a obsahovat grafické uživatelské rozhraní. Neobsahuje žádnou *Legacy aplikaci*, přístup k datům je přímo do databáze.

Hledané řešení má podporovat databázové platformy FoxPro i FirebirdSQL a zpřístupnit je pro prezentační vrstvu v podobě webové stránky. Do řešení patří také vestavěna vnitřní logika, především autentizace a autorizace, a správcovské rozhraní pro kontrolu a monitorování služby.

2.2.2 Zákazníci

Ve stávajícím informačním systému není zavedena podpora pro provoz Online systému. Konkrétně jde o databázi (o několika málo tabulkách), kde by byly uloženy přístupové údaje pro uživatelské účty. Pro firmu není žádoucí o tyto údaje stávající systém rozšiřovat a i vzhledem k dalšímu vývoji je požadována oddělená správu uživatelských účtů pro systém Online.

Firma požaduje plnou kontrolu nad vytvářením a správou uživatelských účtů. Požadována je multifunkční aplikace, kterou bude ovládat pověřený pracovník. Ten bude ručně schvalovat žádosti o vytvoření účtu a další úkony. V rámci této práce je implementována pouze nezbytná funkcionalita této aplikace postavené nad frameworkem. Volitelná funkcionalita může být doplněna časem podle požadavků provozu.

Zákazníci jsou typicky firmy, které mohou mít více obchodních zástupců. V informačním systému jsou však vedeni pod jednou firmou a systém Online by měl umožnit používání více nezávislých účtů.

2.2.3 Databáze

Situace je spjata se zastaralou databázovou platformou .DBF (dBase/xBase) tabulek. Řešení by mělo být komponováno s myšlenkou na přechod na platformu Firebird SQL, na níž funguje systém Flores. Zároveň by mělo řešení být dostatečně robustní, aby bylo schopno pracovat s oběma systémy souběžně, anebo i s úplně jinou platformou, pokud by to bylo potřeba.

DBF

Komplikace, které přináší databáze s DBF tabulkami, je rozebrána v analýze. Firmou otestovaná a preferovaná technologie pro práci s daty je skriptovací jazyk *Visual FoxPro*. Jako jediný totiž dokáže využít optimalizační engine *Rushmore*, bez nějž se provoz ukázal být neúnosně pomalý. Ve FoxPro napsaný *COM server*, zkompilovaný jako dynamická knihovna, je totiž použitelný v rámci frameworku .Net.

Problém servisních odstávek je vysvětlen výše. Tuto údržbu nelze provádět za provozu systému, z čehož plyne požadavek na robustní přístup jakékoli aplikace, která s touto databází pracuje. Navržené řešení by tedy mělo mít nějaký mechanismus nebo vrstvu, která by byla schopna odizolovat business logiku od této technické obtíže, nebo alespoň dát tuto informaci k dispozici.

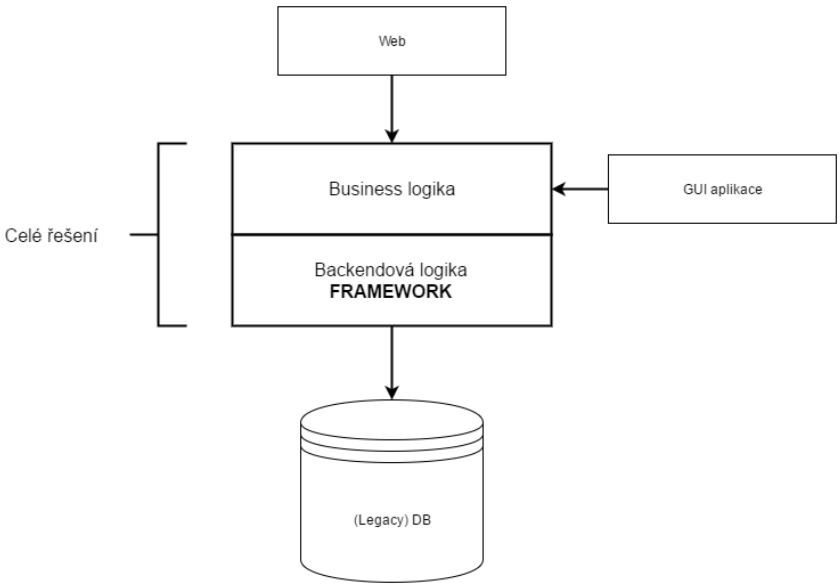
Firebird

Proprietární informační systém Flores je provozován nad databázovou platformou Firebird SQL. To je otevřený relační systém řízení báze dat, poskytující mnoho rysů ANSI SQL. Jsou k dispozici průběžně doplňované a opravované ovladače pro 32 i 64bitové verze systémů Windows i Linux. Zajímavým faktem je, že je podporován frameworkem NHibernate.

Struktura databáze systému Flores se však naprosto liší od té stávající. Všechny dotazy do databáze je potřeba naimplementovat znovu podle vazeb mezi tabulkami a v závislosti na přidaných nebo chybějících datech. Navíc je celká část funkcionality realizovaná uloženými pro-

cedurami (stored procedures), což znesnadňuje automatické mapování objektů případným ORM frameworkem.

Shrneme-li situaci s databázemi, je třeba zohlednit v představě řešení persistenci některých jeho částí. Business logika uvnitř aplikačního serveru by neměla být závislá na momentálně používané databázi, a zároveň by databázová vrstva neměla podléhat změnám v business logice. Z tohoto důvodu se nám celek rozděluje na dvě části, jak je vidět na obrázku 2.6.



Obrázek 2.6: Stále hrubé schéma systému Online, ale s ohledem na databáze. Představa řešení se rozpadla na aplikační Jádro a backendový Framework.

2.2.4 Web

Firma si nechala vytvořit webové rozhraní podle svých představ, úkolem systému Online je zpřístupnit mu data z informačního systému. Web má ucelenou strukturu, přesně vymezený výčet funkcí, které nabízí uživateli a které potřebuje volat na serveru. Volání je realizováno pomocí protokolu *SOAP* a klienta *SoapClient*.

2.3 GUI aplikace

Jádrem programu je software, propojující datovou a prezentační vrstvu systému. V případě systému Online je to aplikace s grafickým uživatelským rozhraním, která umožňuje správci monitorovat stav a provoz na službě a vykonávat některé správcovské aktivity.

2.3.1 Požadovaná funkcionalita

Pro nasazení systému byla zadavatelem specifikována konkrétní funkcionalita společně s představou uživatelského (systémově správcovského) rozhraní.

Monitoring stavu služby
Vzhledem k problematické povaze databáze je žádoucí, aby v systému existoval monitorovací mechanismus pro připojení k datům. Je požadována možnost vizuálně kontrolovat připojenost k databázovým tabulkám (viz sekce 2.2.3) a manuální nebo periodicky plánované odpojení tabulek (vysvětleno v sekci 2.1.3).

Přehled přihlášených uživatelů
Je požadována možnost sledování přihlášených uživatelů, včetně možnosti je manuálně odhlásit.

Sledování provozu a žurnálování
V prototypu kvůli ladění, v hotovém produktu kvůli dohledání aktivit a jevů v programu je požadována evidence akcí v podobě žurnálu.

GUI aplikace
Součástí požadavků firmy je, aby výše uvedené úkony bylo možno vykonávat z přehledné aplikace s grafickým uživatelským rozhraním, aby je v případě potřeby mohl vykonávat i středně kvalifikovaný pracovník.

2.3.2 Vzhled

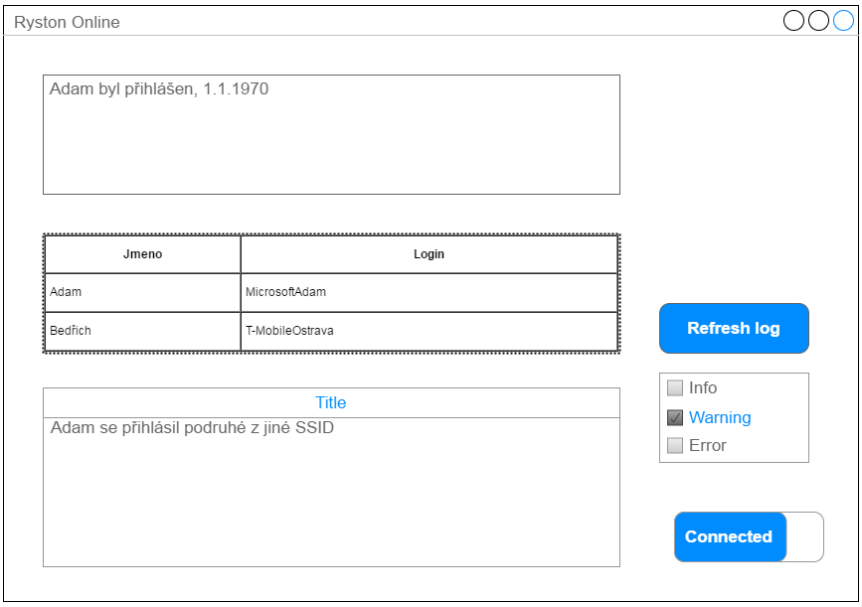
V rámci firemní politiky pro vzhled aplikací (uživatelských, správcovských i smíšených) má zadavatel poměrně přesnou představu o podobě rozhraní. Je požadován jednoduchý formulář s datovými a ovládacími prvky, odolný proti neúmyslným incidentům. Mockup viz na obrázku 2.7

2.4 Priority požadavků

Zde jsou zrekapitulovány a shrnuty požadavky na implementaci jednotlivých rysů návrhu. Každý má určitou prioritu je třeba učinit některé kompromisy, co se týče realistických nároků.

Konfigurovatelné jádro Zcela zásadním prvkem je jádro backendové logiky, které je možno bez zásahu do kódu konfigurovat při postupné migraci mezi platformami. To souvisí s požadavkem na co nejuniverzálnější podporu databázových platforem.

Dvě backend varianty
Na základě specifických požadavků pilotní firmy by měly být systémem podporovány obě dvě databáze, tedy jak legacy DBF tabulky verze 2.5 pro stávající databázi, tak Firebird SQL pro novější systém Flores. Návrh rozhraní bude umožňovat postupné přidávání podpory i dalších databází.



Obrázek 2.7: Mockup podoby GUI rozhraní aplikace Online.

Jeden přístup shora

Pro nasazení do systému Online byla primárně požadována podpora technologie WCF pro svou univerzální použitelnost (viz [10]), hlavně pro jednoduchý přístup ze stávajících webových aplikací, což je vyžadováno.

Přístup k datům prioritně pro čtení

Krátkodobým cílem firmy je mít webové uživatelské rozhraní, které umožní prohlížení firemních dokumentů zákazníky. Dlouhodobým cílem bude ale interaktivní e-shop. Prioritou je tedy zajistit v první řadě "Read only" přístup k datům s tím, že nebude problém přidat i opačný směr.

V této kapitole jsme shrnuli analýzu situace ve firmě, její relevantní historii a softwarové vybavení. Dále byly specifikovány požadavky na její nový online systém, který bude nasazen jako varianta obecného návrhu frameworku. Tento návrh je popsán v následující kapitole.

3. Design

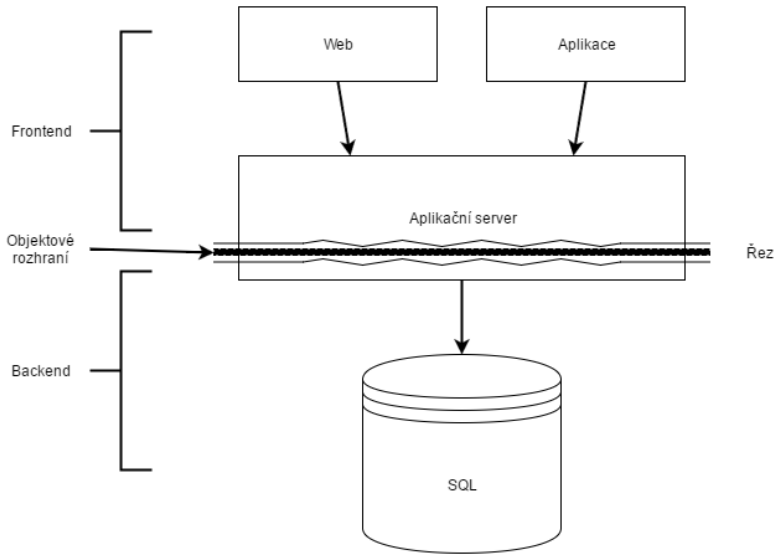
V této kapitole je popsán návrh frameworku. Je popsána základní stavba, komponenty a tok dat. Je demonstrováno, jakým způsobem jádro spravuje backendové moduly a jak je konfigurovatelné.

Cílem projektu je navrhnout mnohostranně použitelný, modulární, objektově orientovaný systém, umožňující propojení aplikační logiky postavené na moderních technologiích s nejen soudobými, ale i legacy systémy a databázemi. Systém by měl právě modularitou usnadnit přechod od zmíněných zastaralých technologií na moderní.

Základní podmínkou pro návrh byla velká flexibilita vůči databázovým platformám. Protože celkový návrh má sdružovat a využívat pozitiva více modelů, je nasnadě inspirovat se způsobem fungování velmi rozšířeného frameworku Hibernate. Důsledky tohoto rozhodnutí se projevují v dále rozebraných sekcích. Pro implementaci byl zvolen jazyk C#, protože je to dobrý standard v oboru vývoje informačních systémů[9], díky projektu *Mono*[8] ho bude možné použít i na linuxových systémech a protože bude možno framework využít při implementaci v případové studii.

3.1 Celkový pohled

Prvním krokem při tvorbě návrhu je představa transformace existující moderní aplikace pro použití frameworku. Hlavním zásahem je izolovat frontendovou část od backendu. Je tedy potřeba aplikaci na vhodném místě rozdělit a buď to využít existující, nebo vložit nové rozhraní, jako je na obrázku 3.1.

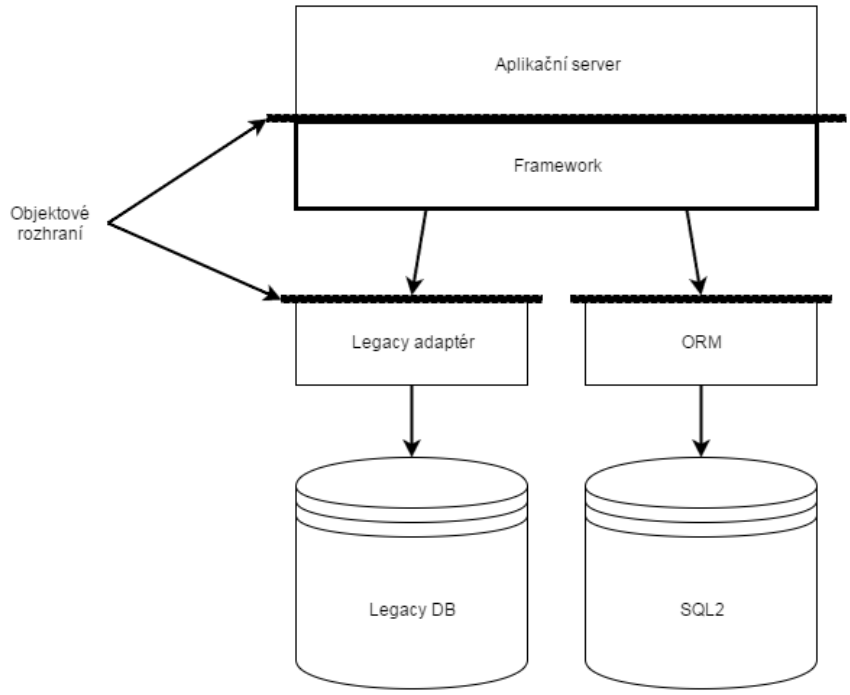


Obrázek 3.1: Schéma řezu moderní aplikace. Objektové rozhraní umožňuje aplikaci komunikovat s datovou vrstvou.

Z backendové části se tak stává izolovaný celek, sestávající se z databáze a adaptéru pro ni. Dále ho budeme nazývat *backendový modul*, protože je ho díky rozhraní vyměnit za jiný.

Místo rozdělení je třeba vybrat tak, aby byl možný přenos dat v objektové podobě. Objekty totiž nejsnáze sjednotí paradigma dat z různých zdrojů (viz kapitola 1). Z toho také vyplývá vhodnost využití nějakého *Objektově-Relačního Mapování (ORM)* pro SQL databáze.

Do vzniklé mezery můžeme vložit novou vrstvu - framework, který odejme aplikačnímu serveru jeho backendovou logiku, tedy správu databáze. Pokud stojí systém nad více databázemi, je tato netrivialita pro business logiku zcela skryta. Právě sem také můžeme vložit přístup do legacy systému. Dostaneme tedy vrstevnatou strukturu, znázorněnou na obrázku 3.2.



Obrázek 3.2: Nahrazení backendové logiky frameworkem s objektovým rozhraním - stejným, jako nabízí ORM wrapper pro SQL databázi i adaptér pro legacy systém.

Nyní probereme stavební prvky, z nichž jsou poskládány jednotlivé vrstvy.

3.2 Rozhraní

Klíčovou úlohu v návrhu hraje objektové rozhraní. Jeho úloha je zapouzdřit databázové platformy do uniformní podoby, aby frontend nebyl závislý na spodních vrstvách. Mezi základní stavební kameny patří následující úvahy:

CRUD operace
: tedy zprostředkování operací, odpovídajících příkazům Create (INSERT), Read (SELECT), Update a Delete. Konceptuálně odpovídají nejběžnějším potřebám při práci s databází.

PerformAction
: Odpovídá provedení ne-CRUDové akce. V prostředí SQL se může jednat o zavolání uložené procedury, v případě ovladače na legacy aplikaci např. stisknutí tlačítka “odeslat ke kontrole”.

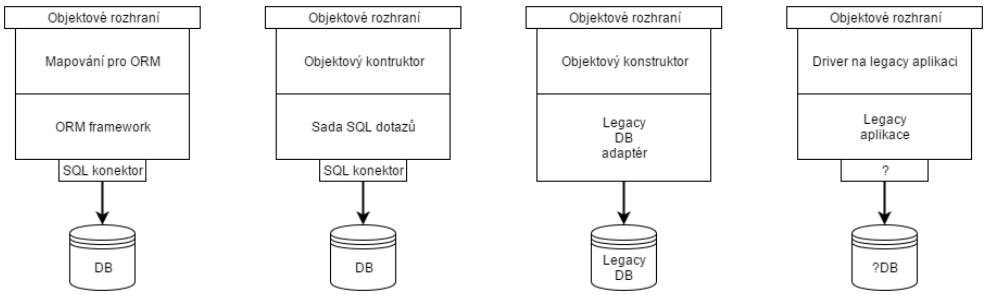
Rozšiřitelnost
: Důležitou vlastností frameworku je možnost rozšíření funkcionality, pokud to nasazení vyžaduje. První bod by měl pokrýt většinu situací, ale je možné přidat vlastní ad-hoc metody pomocí funkce PerformAction.

Podrobnější implementace je popsána v následující kapitole.

3.3 Backend moduly

Backend moduly jsou pro framework vstupní branou do databází. Pro moderní aplikace se jedná o některou formu ORM, ať framework (jako např. Hibernate ORM anebo Entity Framework), nebo vlastní implementaci s explicitními SQL dotazy.

V případě legacy systémů jde buď to o adaptér pro legacy databázi, nebo v nejhorším případě ovladač (driver) na celou legacy aplikaci. Situace je ilustrována na obrázku 3.3.



Obrázek 3.3: Příklady podob backend modulů. (Zleva) ORM framework, vlastní implementace objektového wrapperu, legacy adaptér (s konektorem do legacy databáze) a ovladač na legacy aplikaci.

Backend moduly obecně mohou implementovat různé podmnožiny celkové funkcionality. To, které dotazy nebo požadavky jsou na nich volány, závisí (kromě možnostií modulu) na konfiguraci jádra (vysvětleno v následující sekci).

Z popisu návrhu plynou dvě možnosti využití. První je implementovat v každém modulu všechnu funkcionalitu. Toho lze využít např. při testování nové databázové platformy pro jednorázový přechod. Není od věci zmínit variantu dvou instancí stejného modulu, lišících se připojovacím řetězcem (Connection string) - např. pro provoz nad testovacími daty.

Druhá možnost je implementovat v modulech různé podmnožiny celkové funkcionality. Typické využití je při postupném přechodu mezi systémy (legacy $\rightarrow$ moderní) nebo v případě multiplatformní aplikace. V konfiguračním souboru frameworku je pro každou definovanou metodu určen modul, ze kterého má být volána. Pokud pro metodu není specifikován modul, je použit výchozí.

3.4 Jádro frameworku

Jádro frameworku představuje společnou sdílenou část řešení. Obsahuje inteligenci pro správu backendových modulů a na základě své konfigurace rozesílá jim volání metod. Jádro je konfigurovatelné co se týče přítomnosti modulů a funkcionality, kterou implementují. V konfiguračním souboru je popsáno, jaké moduly má jádro používat a jaké dotazy má který modul dostávat.

Předpokládá se, že veškerá business logika se provádí aplikační vrstvě a do jádra se dostanou jen dotazy, které vyžadují práci s daty z databáze nebo jinou operaci s backend modulem (např. stisknutí tlačítka v legacy aplikaci). Jádro implementuje stejné rozhraní jako backendové moduly. Uvnitř metod je tedy prováděna pouze "rozřaďovací" logika na základě identifikace dotazu (např. "getCustomer") a zavolání odpovídající metody (rovněž z uvedeného rozhraní) ve zvoleném modulu.

Konfigurace jádra tedy obsahuje seznam dostupných modulů a mapování dotazů na moduly. Zde se hodí podotknout, že dotazy nemusí být mapovány všechny explicitně. Souvislost s legacy systémy nás totiž navádí na situaci, kdy např. pouze malá část funkcionality je realizována "legacy modulem". Takový dotaz je jmenovitě namapován, a zbylých 99% je směřováno do defaultního modulu nad standardní databází.

3.5 Objekty

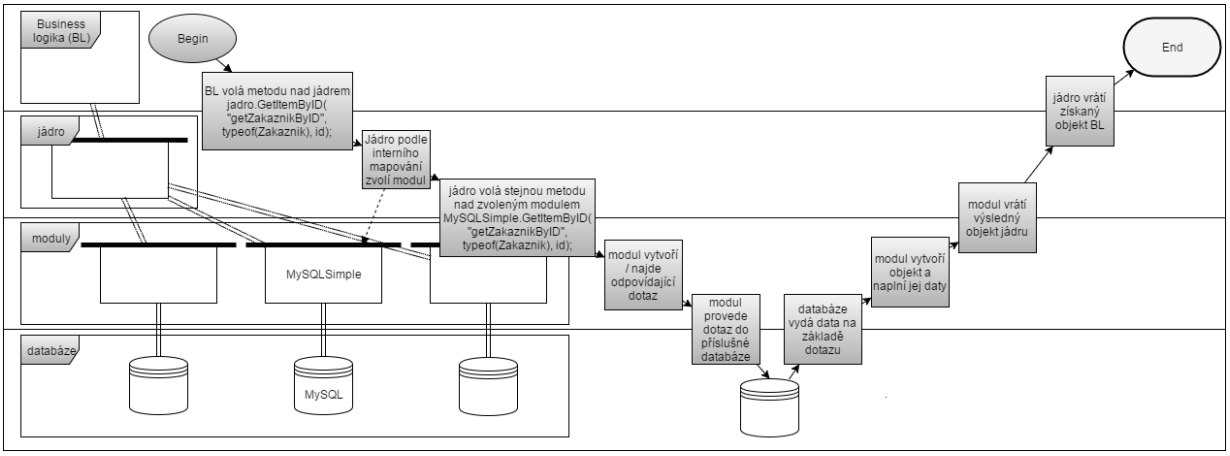
Backendové moduly a business logika musejí znát strukturu objektů, se kterými pracují. Jádro frameworku však může s daty pracovat zcela obecně. Stačí, aby za běhu bylo schopno instanciovat objekty z jiných sestavení.

3.6 Příklad dotazu

Uveďme příklad předpokládaného průběhu událostí při zavolání požadavku na získání objektu z databáze.

1. Business logika zavolá na jádře metodu pro získání objektu, například takto:
   `_core.GetItemByID("getCustomerByID", typeof(Customer), id);`
2. Jádro se na základě vnitřního mapování rozhodne, že dotaz přepošle právě modulu jménem "MySQLSimple".
3. Jádro zavolá na tomto modulu opět metodu `GetItemByID` se stejnými argumenty
4. Modul na základě své implementace provede dotaz do databáze.
5. Modul vytvoří objekt požadovaného typu a naplní jeho vlastnosti obsahem z databáze
6. Modul vrátí výsledný objekt jádru
7. Jádro vrátí objekt business logice

Obrázek 3.4 zachycuje výše popsaný dotaz pomocí UML diagramu.



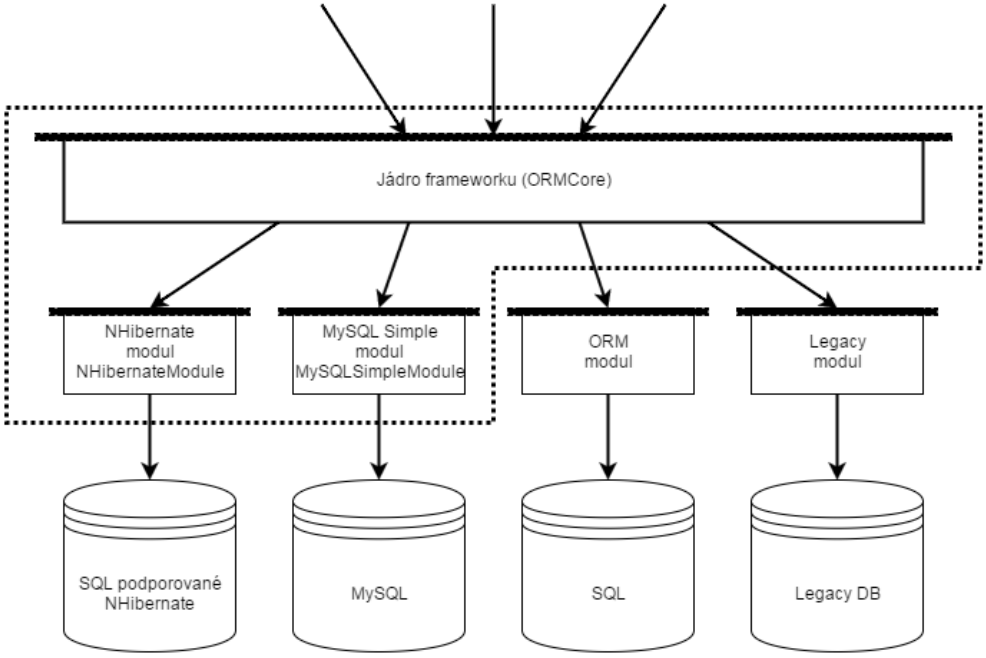
Obrázek 3.4: UML diagram popisovaného příkladu dotazu.

Tok dat je celkem zřejmý. Nepřibývá zbytečně velká režie, modul může podle vlastních schopností provádět optimalizaci dotazů nebo cachování, to už je věc implementace.

V této kapitole byl představen návrh frameworku a způsob jeho začlenění do aplikace. Bylo popsáno společné objektové rozhraní a způsob práce s backendovými moduly. Byl popsán proud dat jednotlivými vrstvami včetně příkladu. V následující kapitole bude popsána ukázková implementace.

4. Implementace frameworku

V této kapitole je popsána implementace návrhu v jazyce C# nad platformou .Net. Softwarové dílo, označované jako framework, se skládá z jádra frameworku a dvou využitelných implementací ORM modulů - viz obrázek 4.1. Tyto části jsou napsány pro univerzální použití nad libovolným kontextem, který je kompatibilní s objektovým konceptem dat.



Obrázek 4.1: Schéma implementace frameworku. Uvnitř tečkovaného výběru je jádro frameworku `ORMCore` a dva vzorové backend moduly - `NHibernateModule` a `MySQLSimpleModule`. Pro ilustraci jsou zahrnuty předpokládané další možnosti - libovolný ORM modul a legacy modul.

Jak je uvedeno v minulé kapitole, návrh i implementace je do značné míry inspirována frameworkem *NHibernate ORM* a způsobem, jakým se s ním pracuje. Právě NHibernate je nástrojem, který využívá jeden z modulů. Znamená to také, že vývojáři, který je zvyklý s NHibernate pracovat, bude způsob práce s frameworkem blízký.

4.1 Rozhraní

Základním stavebním prvkem frameworku je objektové rozhraní, poskytující metody pro manipulaci s daty v databázi. V návrhu jsou uvedeny základní požadavky, zde jsou pak převedeny do kódu. Rozhraní podporuje CRUD operace a poskytuje prostor pro rozšíření o vlastní ad-hoc funkcionalitu. Také obsahuje režijní metody pro práci s databází jako s celkem.

Rozhraní je implementováno pod názvem `IORM` a deklaruje následující metody:

Read

Nejpoužívanější funkce při vývoji softwaru bývá získání dat z databáze. V návrhu je realizována třemi metodami, podle povahy výsledku a "netriviality" jeho získání. Toto rozdělení usnadňuje vývojáři práci při volání metody z aplikace, tak zlepšuje granularizaci při mapování objektů pro ORM framework. Nejjednodušší varianta je

```
Object GetItemByID(string method, Type type, object id);
```

kde `id` je primární klíč pro určení objektu. Druhá možnost je vyhledat objekt komplikovanějším způsobem, například voláním uložené procedury s parametry. Od toho existuje metoda

```
Object GetItemByQuery(string method, Type type, object[] args);,
```

kde parametr `args` je pole argumentů pro volanou metodu. Obě tyto metody vracejí instanci třídy, jejíž typ je předán jako argument při volání.

Třetí možnost je dotaz, který může vrátit jiný počet výsledků než jeden. K tomu slouží metoda

```
Object GetItemsByQuery(string method, Type type, object[] args);
```

která je vlastně zobecněním druhého případu. Vrací kontejner s objekty typu `typ`. Typ kontejneru závisí na použitém modulu, takže po přemapování se může vracet něco jiného (pole místo kolekce apod.).

Create/Update

Tato metoda sdružuje funkce Create a Update. Její podoba

```
void SaveItem(string method, Object item, object id = null);,
```

odpovídá jak možnosti aktualizovat existující položku (s předvyplněným identifikátorem, načteným z databáze), tak vytvořit novou, a to jak s implicitně tak explicitně daným identifikátorem.

Delete

Posledním článkem životního cyklu objektu je jeho smazání. K tomu slouží metoda

```
void DeleteItem(string method, Object item);.
```

Vlastní operace Úkony, které nespadají do výše zmíněných kategorií mohou být provedeny pomocí metody

```
Object PerformAction(string method, Type type, object[] args);,
```

která je nejobecnější podobou interakce aplikační a datové vrstvy. Spadá sem např. "Odeslat rozvrh ke kontrole" apod.

Transakce Transakce jsou užitečný koncept při provádění komplikovanějších dotazů. Jsou však záležitostí přímo databázového adaptéru, který je provádí na základě jednoduché instrukce. Příkazy `BeginTransaction`, `Commit` a `Rollback` by tak měly být skryty před aplikační logikou, už proto že některé databázové moduly je nemusí podporovat. V rozhraní tudíž tyto metody nejsou.

Připojování V mnoha případech může být podstatná informace, zda je databáze dostupná či nikoli. Proto v rozhraní explicitně exisuje metoda `bool IsConnected();` která vrací dostupnost databáze. K této funkci patří neodmyslitelně i metody `void Connect();` a `void Disconnect();`.

Jak je patrno, datové metody mají jako první parametr identifikátor dotazu. To pomáhá jádru při větvení a rovněž modulům, které mohou mít více implementací pro některé operace (např. seznam sedadel v letadle vs. seznam nejčastěji obsazovaných sedadel napříč historií).

4.2 Jádro

Jádro frameworku je esenciální součástí celého řešení, protože sdružuje moduly do funkčního celku, využitelného aplikační vrstvou. Implementace jádra ve třídě `ORMCore` má dvě hlavní části - načítání konfigurace a větvení v metodách.

4.2.1 Konfigurace

Konfigurace jádra obnáší dvě části - definice použitých modulů a mapování dotazů na moduly. Konfigurace je dodána v podobě XML souboru, který může mít následující podobu:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <modules>
    <module name="MySQL"
      classname="WCFonline.Moduly.MySQLNHibernateModule, WCFonline"
      mappingsfolder="D:\ORM\MySQLDialectMappings"/>
  </modules>
  <defaultmodule name="MySQL"/>
  <modulemappings>
    <mapping method="getUzivatel" module="MySQL"/>
  </modulemappings>
</configuration>
```

V ukázce kódu vidíme deklaraci jediného modulu, který je tedy i výchozí.

Jádro má člen `public Dictionary<string, IORM> Modules`, který eviduje moduly. Pro mapování dotazů do modulů slouží člen `public Dictionary<string, string> Mapping`, který eviduje relaci $metoda \rightarrow modul$.

Při dotazu do jádra je jako parametr předán pouze identifikátor dotazu, `string method`, takže je využita metoda `private IORM GetModule(string method)`, která vrací příslušný modul (výchozí, pokud není specifikován).

4.3 Modul NHibernate

Velmi užitečným nástrojem pro seznámení se a práci s frameworkem je backendový modul `NHibernateModule`. Modul převádí volání metod z rozhraní `IORM` na metody frameworku NHibernate. Vývojář zběhlý v NHibernate jistě ocení velmi podobný způsob práce.

Modul `NHibernateModule` je přizpůsoben situaci, kdy je použit ve více instancích s různými sadami mapovacích souborů (typicky pro dvě různé databázové platformy). V konstruktoru přijímá cestu k adresáři, kde vyhledává soubory s koncovkou `.hbm.xml`. Tato separace je nutná, protože NHibernate si neporadí s kolizemi při deklaraci mapování.

4.4 Modul MySQLSimple

Druhá varianta využitelného řešení je modul `MySQLSimpleModule`. Jak jeho název napovídá, je navržen na jednoduchou práci s MySQL databází. Narozdíl od modulu pro NHibernate neobsahuje žádný ORM framework. Funguje pomocí explicitně psaných SQL dotazů a vlastnosti objektu očekávaného návratového typu se pokusí namapovat na sloupce výsledku dotazu. Tento modul není úplně vhodný pro komerční nasazení, ale slouží jako "proof of concept". Je také dobrým základem pro implementaci vlastního modulu nebo jako rychlá pomoc při absenci sofistikovanějšího řešení.

Modul využívá primitivní podobu konfigurace. Všechny údaje jsou v jednom XML souboru, tedy připojovací řetězec, definice dotazů i mapování na vlastnosti objektů. Viz příklad:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionstring>
    server=satan.ryston.cz; user id=diplomka;
    password=diplomka; database=online; pooling=false;
  </connectionstring>
  <query name="getUzivatel"
      resultclass="WCFonline.Struktury.Uzivatel, WCFonline">
    <sql>
      select * from uzivatele where uzivatele.login = @1
    </sql>
    <mapping property="Uzivatelid" columnname="login"/>
  </query>
</configuration>
```

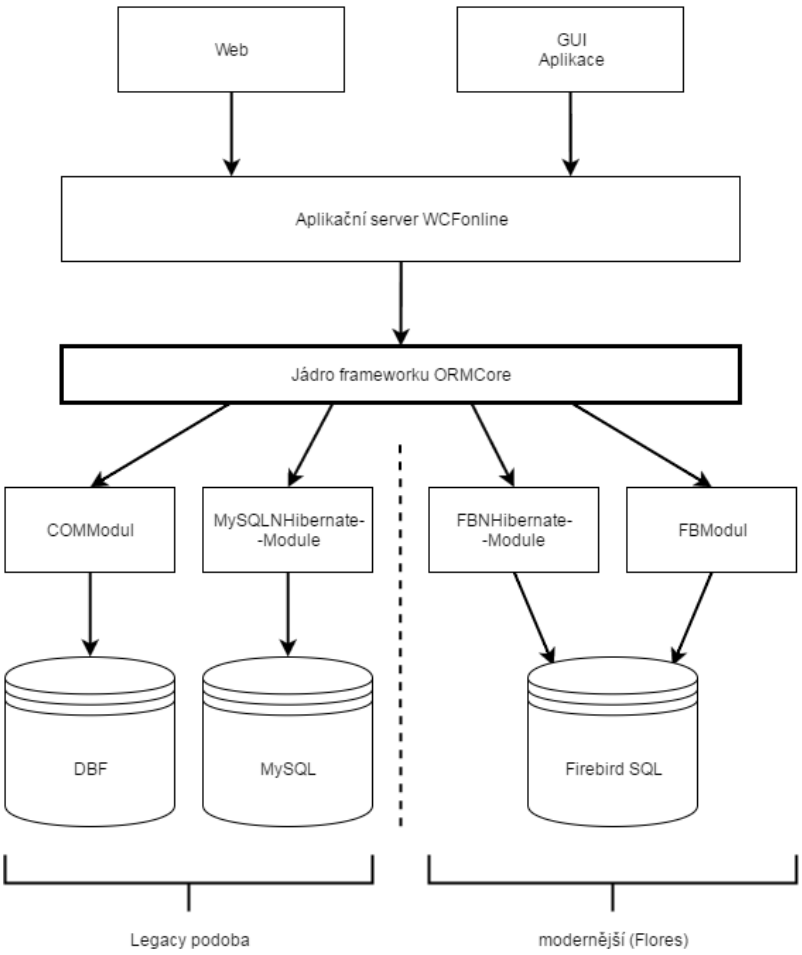
Jak bylo již řečeno, tento modul je úmyslně velmi triviální, aby prokázal snadnou použitelnost frameworku. V uvedeném příkladě je, když pomineme databázovou stranu, přítomno velmi málo předpokladů - Existence třídy `WCFonline.Struktury.Uzivatel` v sestavení `WCFonline`, která má vlastnost `Uzivatelid`.

5. Pilotní implementace

V této kapitole je popsána pilotní implementace, tedy konkrétní podoba aplikace vystavěné nad příkladovým použitím navrhovaného frameworku. Systém Online, vtělený do aplikace `WCFonline`, slouží jako aplikační server pro webovou aplikaci.

5.1 Struktura

Podoba aplikace je zachycena na schématu 5.1.



Obrázek 5.1: Schéma aplikace Online. K datům přistupuje aplikační server přes framework, nad aplikačním serverem je vystavěna GUI aplikace pro monitorování provozu v systému Online a WCF služba pro web. Ve spodní části obrázku jsou dvě varianty fungování systému - pro starou a novou podobu databází.

5.2 Technologie

Zásadním rozhodnutím pro nasazení díla je výběr vhodných technologií. V obecném případě jsou možnosti široké, pro systém Online jsou předem některé aspekty dány - technologie databáze (DBF), protokol prezentační vrstvy (SOAP) a preferovaná rodina softwaru (Microsoft). Základní rozhodnutí je samotný programovací jazyk. Skládat projekt z více jazyků by představovalo budoucí zvýšenou zátěž pro vývojáře. Bylo tedy potřeba vybrat jeden jazyk, který by uspokojil rozličné požadavky, které vyplývají z analýzy a specifikace. Bude v něm napsáno jádro a bude sjednocovat ostatní moduly do fungujícího celku.

Vítězným kandidátem je jazyk C#, s platformou *.NET*. Poskytuje obrovské množství podpory pro všemožné aplikace a technologie, v kontextu zmiňme databázové konektory, interoperabilitu, aplikační server SOA, GUI aplikace a mnoho dalších. Přímo pro C#je napsán port Javového frameworku Hibernate *NHibernate*, velmi rozšířený nástroj na ORM, který bere v úvahu samotný návrh frameworku. Licenčně je C#pro firmu přijatelný, navíc v něm jsou již vytvořeny další aplikace, které v budoucnu mohou spolupracovat.

V úvahu byla brána rovněž Java jako open source alternativa k technologiím od společnosti Microsoft. Nabízí jak konektory pro DBF tak pro Firebird, nástroje pro vývoj GUI aplikací a je rovněž široce podporovaná. Pro Javu původně vznikl framework Hibernate. Silný argument proti ní je bezpečnostní riziko spojené s používáním zastaralé verze Javy na nadále nepodporovaném systému Windows XP. Zásadním je ale fakt, že driver *JDBC* nedokáže využít technologii Rushmore pro FoxPro a na velkých tabulkách je příliš pomalý. Existuje sice projekt *JACOB: A JAva-COM Bridge*[6], ten ale stojí na OLE DB provideru od firmy Microsoft a nepřináší tudíž žádnou explicitní výhodu.

Další možností bylo *C++*. Bez platformy .NET se nepřipojí k Firebirdu a s ní nenabízí nic víc než C#, naopak mu schází řada vymožeností C#, například Reflexe, esenciální pro ORM. Nabízely se i další možnosti jako *Python*, ale firma byla proti nasazování jimi neotestovaných kombinací technologií.

5.2.1 Backend

FoxPro

Volba technologie pro primární databázi je poměrně jednoduchá. V roce 1994 koupila technologii *FoxPro* firma Microsoft, aby ji začlenila pod názvem *Visual FoxPro* do své rodiny vývojových prostředí, a poskytuje proto řadu nástrojů pro zpětnou kompatibilitu. Před tím, než v roce 2007 definitivně ukončil její vývoj, vyvinul Microsoft pro zpřístupnění FoxPro ve vyšších jazycích technologii *Object Linking and Embedding (OLE)* na základě standardu *Component Object Model (COM)*. Pro starou verzi DBF databáze neexistuje jiný dostatečně rychlý prostředek než *Visual FoxPro*, který dokáže využít optimalizační technologii *Rushmore*. Pomocí technologie *OLE Automation*[1] lze vytvářet automační server, použitelný jednoduše ve vyšších jazycích jako objekt s volatelnými metodami.

[1] `https://en.wikipedia.org/wiki/OLE_Automation`

Pro přístup k databázi v podobě (Visual) FoxPro .DBF tabulek je použit jednovláknový COM server. Pro každý požadavek existuje separátní metoda. Každá metoda realizuje "plnění" jistého objektu, ať jde o jednotlivý doklad nebo seznam položek. Tento modul provádí dotazy (kurzorové obdoby SQL příkazů `SELECT` a `JOIN` do tabulek) a transformuje výsledky do (interních) objektů. Tento výsledný objekt je následně "serializován" ručním skládáním řetězců, nikoli standardní metodou, a vrácen jako XML string. Jazyk FoxPro je slabě typovaný, takže objekty z FoxPro bohužel není možné jednoduše posílat do vyššího jazyka, jako je silně typovaný C#.

COM je široce používané rozhraní, které pro svou jednoduchost zůstává podporovanou technologií. Je tedy perspektivním řešením situace s nepodporovanou platformou. Navíc je implementováno přímo od Microsoftu třídou OLEPUBLIC, která je součástí běhového prostředí Visual FoxPro, používaného pro vývoj některých dalších aplikací ve firmě. Tím odpadá potřeba vlastní implementace protokolu.

Kód COM serveru vychází ze zdrojových kódů Doutníka, které byly pro vývoj k dispozici. Naštěstí se dochovaly zdrojové kódy právě pro manuální serializaci objektů do XML z dřívějších pokusů o implementaci, což ušetřilo spoustu náročné a neperspektivní práce. Ačkoli by tato část snesla údržbu kódu, fungovala a byla ponechána ve "vývojové" podobě. Nečeká se její rozšiřování. Další podrobnosti o FoxPro a v ní tvořených programech by obnášely spoustu historie, kterou lze přečíst jinde ([2],[3]), a spoustu kódu, který je bez znalostí FoxPro těžko čitelný.

COM server je zakomponován do "legacy modulu" jménem `COMmodul`. Tento modul implementuje rozhraní navrhovaného frameworku a je vlastně jen C# wrapperem nad kódem pro FoxPro.

Nově přidaná funkcionalita (resp. data) pro systém online byla vložena do databáze MySQL. Pro dotazy do této části je použit modul `MySQLNHibernateModule`, který standardními prostředky frameworku NHibernate ppropojuje požadované objekty s daty. Skvěle tak demonstruje použitelnost frameworku při integraci legacy databází do moderních systémů.

Firebird modul

Sekundární databáze, Firebird, je platforma standardu SQL a je pro ni vyvinuta řada driverů včetně toho pro .NET. Příkazy (SQL queries) lze psát v libovolném nástroji a vložit do použitého jazyka. Firebird ADO.NET Data Provider je dokonce dostupný přes NuGet.

Protože databáze systému Flores má úplně jinou strukturu než DBF, nad níž pracuje Doutník, musely být funkce znovu implementovány v nových SQL dotazech. Významná část odpovídající funkcionality byla ale implementována dotazy v takové podobě (v kombinaci s očekávanou objektovou strukturou), že se autorovi nepodařilo správně nakonfigurovat mapování pro NHibernate. Důsledkem jsou dva moduly, jejichž kombinací je funkcionalita pokryta. Prvním je `FBNHibernateModule`, potomek `NHIbernateModule` z frameworku. Tento modul obstarává jednodušší případy, kdy lze objekt namapovat na výsledky dotazu nebo přímo na řádek tabulky. Druhým modulem, který zajišťuje ty obtížnější případy, je `FBmodul`, který implementuje logiku pro specifický výčet metod. Přípustnost takového řešení potvrzuje použitelnost návrhu frameworku.

5.2.2 Frontend

Frontendovým řešením se v tomto případě rozumí tři odlišné věci - aplikační server, webové rozhraní a GUI aplikace. Aplikační server implementuje business logiku, GUI rozhraní slouží k monitorování a manuálnímu ovládání služby a web prezentuje data zákazníkům.

Webové rozhraní představuje prezentační vrstvu celého projektu. K implementaci lze za důležité uvést to, že právě web svou stavbou určuje podobu dotazů. Dotazy mají v podstatě jen dva různé účely, a to přihlašování a dotaz na dokument. Tyto dotazy se liší počtem a typem argumentů i očekávanou odpovědí. Do jisté míry je možné je sjednotit a implementace aplikačního serveru to také dělá, ale web je tak postavený. Byl vytvořen před zahájením tohoto projektu a firma trvá na jeho použití.

Aplikační server

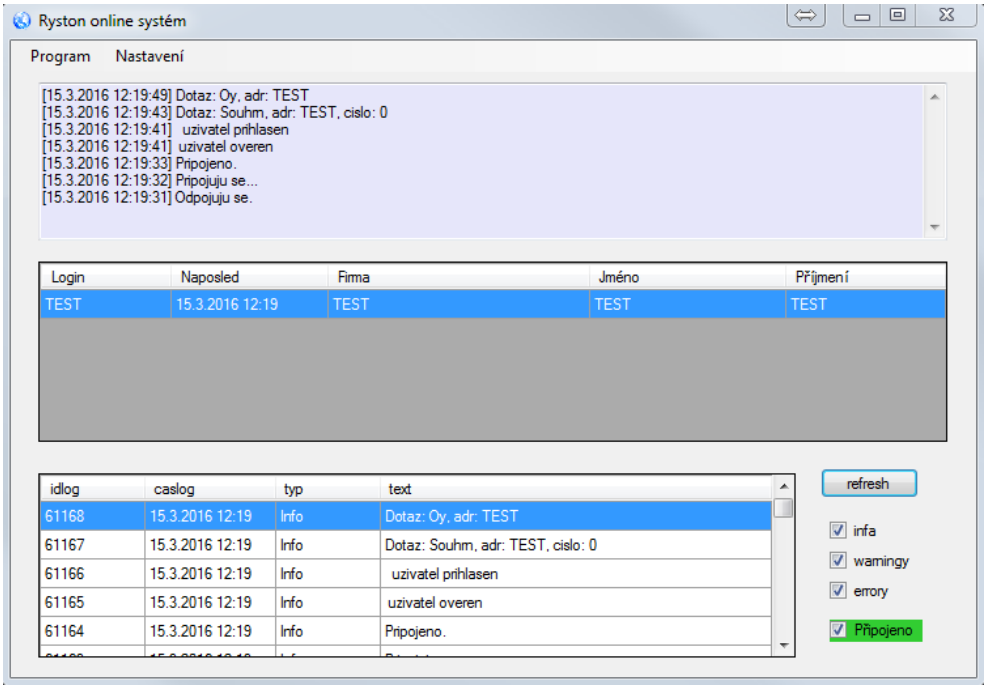
Aplikační server obsahuje business logiku a je základem pro zbývající frontendové části. Je implementován jako třída `WCFservice`. Implementuje rozhraní `IWCFservice`, relizující technologii *Windows Communication Foundation (WCF)*. Rozhraní je anotováno jako `ServiceContract` a každá deklarovaná metoda je anotována jako `OperationContract`. Pro naši situaci, kdy jde o přenos dat spíše než o volání funkcí, WCF kromě formátu JSON podporuje i protokol SOAP, čímž vyhovíme požadavkům specifikace. Třída `WCFservice` obsahuje definice jednotlivých metod, deklarovaných v rozhraní zmíněném výše. Tyto definice jsou u dokladových dotazů jen jednořádkové, protože jsou převedeny na sdružující metodu `ProvedDotaz`, popsanou výše. Ostatní metody, jako třeba `(prihlasit)`, jsou implementovány v rámci business logiky.

5.3 Formulářová aplikace

Účelem GUI aplikace je umožnit správci kontrolovat stav systému. Nabízí se vybudovat GUI rozhraní přímo jako součást aplikačního serveru, protože přidat ho z vnějšku by pouze přineslo možné komplikace. Vzhledem k již učiněnému rozhodnutí použít framework .NET se nabízejí dvě platformy - *Windows Forms (WinForms)* a novější *Windows Presentation Foundation (WPF)*. Dopad na funkcionalitu je minimální, a proto bylo rozhodnuto použít WinForms, protože byl autor blízce seznámen s fungováním a vývojem.

Firma měla představu o vzhledu aplikace, s níž bude pracovat správce služby. Vychází z firemních směrnic a jedná se o jediný formulář s několika prvky. Mockup byl uveden minulé sekci (obrázek 2.7), výsledek je zachycen na obrázku 5.2.

Formulářová aplikace je postavena nad aplikačním serverem. To znamená, že celá aplikace běží na jednom počítači; aplikační server, na nějž se připojuje SOAP klient z webové stránky **a zároveň** GUI aplikace. Toto řešení bylo zvoleno pro snadnost kontroly a manipulace se službou - ruční odpojení databáze, restartování služby, manuální odhlášení uživatelů, sledování provozu a prohlížení žurnálu. Aplikace funguje také jako pohled (View) a řadič (Controller) nad modelem, který si udržuje aplikační server. Bylo by obtížné celý program monitorovat a spravovat, pokud by běžel např. jako konzolová aplikace.



Obrázek 5.2: Snímek běžící aplikace WCFonline - Ryston Online systém. Nahoře lokální, podrobný textový log, uprostřed seznam přihlášených uživatelů, dole Dataview se záznamy z logu na MySQL.

5.4 Konvence

Bylo velmi těžké udržet konzistentní pojmenovávací konvence v celém díle, obsahujícím hotové projekty jiných autorů. Bylo třeba vzít v potaz firemní politiku a zvyklosti, pojmenovávací konvence v hotových částech softwaru i praktiky v tom kterém programovacím jazyce. Jinými jazyky je myšleno hlavně PHP na webu a nomenklatura v DBF tabulkách a převzatém kódu FoxPro skriptů.

Přes snahu o jednotnost tak - hlavně z důvodu zamezení kolizí a zachování terminologie a chování programu - nekonzistence přetrvaly. Největší komplikace v tomto směru vnáší do práce netypovaný skriptovací jazyk FoxPro. Formát jmen pro proměnné se odvíjel od úmyslného typu, dávaje mu tak odpovídající prefix, například řetězcová proměnná `cZprava`, obsahující prefix `'c'` podle názvu typu `Character`.

5.5 Objekty

Jak plyne z konceptu frameworku, definice objektů je zcela věcí požité aplikace. Framework je nepotřebuje znát, stačí, aby moduly dokázal objekty instanciovat. K tomu stačí předat jako argument typ objektu z načteného sestavení a modul se postará o zbytek.

Je vhodné uvést, že k datovým atributům je v objektech přidána sada servisních atributů -

osvědčená praxe z práce s FoxPro. Jedná se o pole `lChyba, nChyba` a `cChyba`, tedy příznak, kód a hláška vyskytnuté chyby. K chybě může dojít kdykoli během cesty, v databázovém modulu (chyba databáze), v jádře (chyba deserializace) anebo v business logice (neoprávněný dotaz). Záleží na implementaci (aplikace i modulu), kde přesně je chyba odchycena a jak je s ní naloženo. V případě řešení chyb v business logice by stačilo vyhodit výjimku, ale Online má implementované odchytávání (některých) chyb na webu, a proto si nesou objekty příznaky celou cestu vrstvami.

5.6 Zpracování požadavků

V programu existují dva druhy požadavků - dotazy na dokumenty a požadavky na organizační akci. Organizační akcí je například požadavek na přihlášení uživatele, který přijde z webu, nebo dotaz na žurnál (service log) ze správcovské aplikace. Tyto dotazy nejsou přeposílány do databázových modulů, ale jsou zpracovány v aplikaci.

Dotazy na dokumenty přicházejí z webu do aplikačního serveru. Dotazy jsou navzájem strukturálně stejné, takže jsou implementovány voláním společné metody `ProvedDotaz`, jejíž přesné chování je ovlivněno parametry. Postup business logiky je vcelku přímočarý:

1. Init - zpracování příchozích parametrů, identifikace druhu dotazu
2. Kontrola připojení k databázi (potřebná jen u DBF[2])
3. Kontrola přihlášení uživatele (zda smí pokládat dotazy)
4. Získání datového objektu (zavolání příslušné metody na jádře frameworku)
5. Kontrola vnořených chyb (některé chyby mohou vyžadovat reakci na této úrovni)
6. Kontrola oprávnění tazatele (zda má nárok na tento výsledek)
7. Vrácení výsledku.

Lze si všimnout, že průběh operací připomíná průběh dat od jádra frameworku dolů, a skutečně se implementace aplikace tímto schématem inspirovala.

V ideálním případě proběhne tento řetězec akcí a výsledkem je objekt s naplněnými vlastnostmi, který je vrácen jako odpověď na volanou funkci. V každé fázi může pochopitelně dojít k odklonu od ideálního workflow; tyto situace jsou ošetřeny příznakem vyskytnuté chyby, číslem a chybovým hlášením (viz přiložený číselník `chyby.xml`, vysvětlení výše v sekci 5.5).

Díky modulárnímu návrhu frameworku je bod (4.) oproštěn od konkrétní backendové implementace. Kód je tak obecně použitelný a nezávislý na používané databázové platformě.

[2]Jedná se o kontrolu, zda není právě odstávka. Samotná kontrola připojení databáze proběhne v modulu.

5.7 Konfigurace jádra

Aplikace, resp. jádro frameworku, které aplikace používá, je konfigurovatelné tak, jak je popsáno v sekci 4.2.1. Využívá XML soubor o třech částech - definice a popisy modulů, které mají být použity, výchozí modul a mapování dotazů na moduly.

Možnosti definice více modulů, nikoli modulu jediného, plyne z typické potřeby používat data z více zdrojů. Konfigurace obsahuje pracovní název modulu, pomocí nějž se následně mapují dotazy a plné jméno třídy, která daný modul implementuje (jmenný prostor, jméno třídy, sestavení). Případně obsahuje také cestu k adresáři, v němž jsou konfigurační soubory. Toto rozlišení je výhodné pro situaci, kdy jsou použity různé sady konfiguračních souborů, a pro modul (např. NHibernateModule a jeho NHibernate jádro) by byl problém s duplicitními definicemi.

Pro jednotlivé dotazy je pak v souboru specifikováno, z kterého modulu mají být volány. Aby nebylo nutno po přidání nového dotazu doplňovat konfigurační soubor, je pro nespecifikovaný dotaz vybrán výchozí modul. Explicitně definované tak může být jen minimum dotazů, které zůstávají v legacy modulu.

6. Závěr

6.1 Shrnutí

Tato práce představila návrh frameworku, který dokáže usnadnit problém s přístupem moderních aplikací do legacy systémů. Framework je silně konfigurovatelný, nezávislý na kontextu dat, nezávislý na použité databázové platformě a v návrhu také nezávislý na programovacím jazyce. Je objektově orientovaný, aby dokázal překlenout co nejširší možnosti konceptu dat (relační, objektové, dokumentové, key-value atd.).

Tento framework spravuje "backend moduly", tedy adaptéry pro libovolnou databázovou technologii, zapouzdřené do objektového wrapperu. Backendovým modulem může být implementovaná sada SQL dotazů, zabalená do API, wrapper pro ORM framework, ale může to být také adaptér pro dynamicky připojený skript v jiném jazyce anebo ovladač na neopustitelnou aplikaci.

Framework byl implementován v jazyce C#a kromě jádra obsahuje dvě vzorové implementace modulů. Jedním z modulů je velmi jednoduchý adaptér pro MySQL, dobře demonstrující obecnost návrhu. Druhý modul je nadstavba frameworku NHibernate, což je široce použitelný ORM nástroj.

Byla prozkoumána situace v konkrétním případě firmy, která se potýká s legacy systémem. Byly prezentovány softwarové a hardwarové nároky a možnosti firmy a rozebrány důsledky přechodu na nový proprietární systém. Byla specifikována přání, požadavky a vize firmy na nový systém pro webové rozhraní, které má přistupovat k datům z informačního systému. Vyvstala konkrétní omezení na práci s databází, kterou je kvůli údržbě potřeba odstavovat. Požadavky na funkcionalitu byly seřazeny podle priority, zvítězilo především nasazení do provozu i s omezenou funkcionalitou.

Systém "Online" byl implementován a nasazen. Kvůli omezením původního systému byla část dat umístěna do MySQL databáze mimo hlavní úložiště. Tím se ale zároveň ověřila možnost současně přistupovat do legacy databáze i do standardní DB. Během provozu došlo ke změně firemní databáze, na což byl systém navržen a připraven a tudíž obstál.

6.2 Budoucí práce

V této kapitole jsou uvedeny vize a návrhy na další vývoj projektu. Kapitola má dvě části, první se zabývá frameworkem jako návrhem, v obecné podobě a druhá se týká pilotního nasazení frameworku v systému Online.

6.2.1 Obecný návrh frameworku

Navrhovaný framework se osvědčil v pilotním nasazení. Pro skutečně univerzální nasazení existuje několik vylepšení nebo rozšíření.

Další moduly

V praxi existuje celá řada dalších legacy technologií, nad nimiž jsou provozovány funkční systémy, ale které teď brzdí vývoj infrastruktury dané firmy. V případě potřeby by mohly být implementovány další odpovídající moduly. Příkladem by mohl být modul pro ovladače na GUI aplikace[1].

Framework však lze díky NHibernate použít i pro komunikaci s mnoha moderními databázovými systémy.

Klon v jiném programovacím jazyce

Nasazení frameworku v konkrétních podmínkách pilotní firmy přineslo celou řadu rozhodnutí, která by v případě jiných požadavků mohla vést k dosti odlišnému výsledku. Například v případě striktního požadavku na multiplatformnost by bylo možné framework implementovat v Javě.

Rozšíření rozhraní

Experiment ukázal, že původní představy se mohou lišit od konečného výsledku. Důkazem toho jsou metody `Connect, Disconnect` a `IsConnected`, které se při vývoji frameworku ukázaly jako užitečné. Použití v dalších případech by mohlo navrhnout další takové rozšíření, například pro podporu transakcí.

6.2.2 Online

V této sekci jsou rozebrány plány a vize na další vývoj systému Online. V rámci práce byl systém uveden do pilotního provozu, což byl hlavní požadavek firmy. Existuje na něm však mnoho prostoru pro rozvoj.

Obousměrný provoz

Z důvodu bezpečnosti je systém Online pouze "jednosměrný", tedy umožňuje zákazníkům číst data, ale nikoli je měnit nebo provádět nějaké operace nad daty v informačním systému. Do budoucna je žádoucí rozvinout systém Online do e-shopu s funkcionalitou, jaká je běžně dostupná.

Predikční logika

V zájmu firmy je mít rovněž systém predikce prodeje. V projektu Online od něj bylo upuštěno, protože nebyl prioritou pro funkčnost systému a neměl konceptuální podklady. Do budoucna by mohl být do systému integrován AI systém, který by zákazníkům nabízel relevantní zboží nebo slevy na základě jejich nákupní historie a vnějších trendů.

[1] `https://en.wikipedia.org/wiki/List_of_GUI_testing_tools`

Prostor pro zlepšování

Žádná implementace se neobejde bezkompromisů. V případě systému Online bylo nutné využít části kódu, které nevyhovují standardům kvalitně psaného kódu. Tyto části byly výsledkem předchozího vývoje uvnitř firmy z období let 1999-2007. Přestože se je podařilo scelit do funkčního celku, připomínají, jak palčivá může být situace s legacy systémem.

Technologie WCF (dříve Indigo) byla představena v rámci frameworku .NET verze 3.0 v roce 2005 a stále se těší velké oblibě. Umožňuje připojení přes HTTP pomocí protokolu SOAP, které vyžaduje existující webové rozhraní. Pokud by byl web vyvíjen znovu, bylo by na místě použít nějakou jinou metodu, např. REST. Ten je také podporován službou WCF, ale používá místo XML formát JSON. Ve výsledku by mohla být u velkých dat propustnost systému znatelně rychlejší. Stejně tak by při vývoji webu od základu znova stálo za úvahu použití frameworku *ASP.NET*, který by jednodušeji pracoval s .NETovou aplikací.

Co se týče formulářové aplikace, určitě by se hodilo použít místo Windows Forms novější Windows *Presentation Foundation (WPF)*. Je modernější, flexibilnější a Windows Forms byly použity pro časem prověřenou spolehlivost a snadnou práci s neklíčovou součástí software.

V rámci práce se podařilo navrhnout, naimplementovat, nasadit a dlouhodobě provozovat systém na základě prezentovaného frameworku. Framework tím prokázal svou životaschopnost a to, že se dokáže vypořádat s obecným problémem propojení legacy a moderních technologií.

Seznam použité literatury

[1] BAUER, Christian a Gavin KING. Hibernate in action. Greenvich: Manning, 2004. ISBN 193239415X. dostupné z http://www.schraderduncan.com/pdf/pdf_of-investor_Compliance-20130911101453-hibernate_in_action.pdf

[2] BOURKE, Alan. Painless Legacy FoxPro Applications. Hentzenwerke Publishing, Inc., 2003. EISBN 9781930919556.

[3] EGGER, Markus. Modern Application Development: Visual FoxPro and .NET [online]. CODE Magazine, 2004. http://www.codemag.com/article/030024.

[4] GRANOR Tamar E. , Doug Hennig, Jim Slater (Editor), Jim Slater, Editor-Jim Slater, Tamar Ezekiel Granor, Doug Henning What's New in Visual FoxPro 8.0. Paperback, 200 Pages, Published 2003 by Hentzenwerke Publishing ISBN-13: 978-1-930919-40-2, ISBN: 1-930919-40-9

[5] GULUTZAN, Peter. MySQL versus Firebird [online]. Ocelot Computer Services Inc. blog, 2014. http://ocelot.ca/blog/blog/2014/03/09/354/.

[6] JACOB - Java COM Bridge, homepage [online] https://sourceforge.net/projects/jacob-project/

[7] KUCINSKAS, Darius. Entity Framework 6 vs NHibernate 4. Devbridge Group [online], 2014. https://www.devbridge.com/articles/entity-framework-6-vs-nhibernate-4/

[8] MAMONE, Mark. Practical Mono. Apress, Berkeley, CA, 2006. ISBN: 978-1-59059-548-0 (Print) 978-1-4302-0097-0 (Online)

[9] NAGEL Christian Ovid. Professional C# 2012 and .NET 4.5 . Indianapolis, Ind. : Wiley Pub., Inc., 2013. ISBN 9781118314425. http://site.ebrary.com/lib/cuni/Doc?id=10657793

[10] PATHAK, Nishith. Pro WCF 4: Practical Microsoft SOA Implementation. 2. vydání, Apress, 28. 8. 2011. 472 stran. ISBN 1430233699, 9781430233695.

[11] Selenium WebDriver, homepage [online] http://www.seleniumhq.org/projects/webdriver/

[12] WARREN, Ian. The renaissance of legacy systems: method support for software-system evolution. Springer Science & Business Media, 2012.

Seznam použitých zkratek

- COM - Component Object Model; objektově orientovaný systém od firmy Microsoft pro komunikaci aplikací napříč jazyky [2]
- DBF - dBase database file; souborový formát tabulek pro databázi platformy FoxPro [3]
- ODBC - Open Database Connectivity; standardizované API pro přístup k databázím[4]
- OLE - Object Linking and Embedding [5]
- ORM - Objektově relační mapování [6]
- SEW - Sklad Elektroniky Windows; Windows aplikace napsaná ve VFP, určená především pro obchodní odd. firmy Ryston, nadstavba programu Doutník
- VFP - Visual FoxPro; modernější verze platformy FoxPro, vyvíjená firmou Microsoft [7]
- WCF - Windows Communication Foundation; framework součástí .NETu, umožňuje vytvářet servisně orientované aplikace [8]
- WPF - Windows Presentation Foundation, framework pro tvorbu grafického uživatelského rozhraní od firmy Microsoft [9]

[2] https://en.wikipedia.org/wiki/Component_Object_Model
[3] https://en.wikipedia.org/wiki/.dbf
[4] https://cs.wikipedia.org/wiki/Open_Database_Connectivity
[5] https://cs.wikipedia.org/wiki/Object_Linking_and_Embedding
[6] https://cs.wikipedia.org/wiki/Objektově_relační_mapování
[7] https://en.wikipedia.org/wiki/Visual_FoxPro
[8] https://en.wikipedia.org/wiki/Windows_Communication_Foundation
[9] https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx

Přílohy - obsah přiloženého CD

Práce

V adresáři "Prace" je umístěna tato práce ve formátu PDF i zdrojové soubory pro LaTeX, včetně snímků obrazovky a schémat.

Zdrojové kódy

V adresáři "src" jsou umístěny zdrojové kódy aplikace WCFonline. To znamená projekt ve Visual Studio 2012 "WCFonline" a projekt ve Visual FoxPro "OnlineComServer". Součástí projektů jsou binární soubory nezbytné pro běh aplikace. Ve složce řešení "WCFonline" je také projekt "ORMFramework".

V kořenové složce je umístěn soubor README.TXT, kde je obsah popsán podrobněji. Na disku nejsou uložena data z firemní databáze.

Programátorská dokumentace

Nosič dále obsahuje ve složce "doc" programátorskou dokoumentaci, která se týká používání frameworku a implementace databázových modulů.