Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Jan Blaha

Framework pro vývoj databázových aplikací

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Jan Votrubec
Studijní program: informatika, programování

2007

Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu bakalářské práce panu Mgr. Janu Votrubcovi za celkové vedení práce.

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

V Praze 4. srpna 2007 Jan Blaha

Obsah

1 Úvod **6**
1.1 Softwarový framework 6
1.2 Cíle práce 8

2 NCore **9**
2.1 Použité komponenty 9
2.2 Splnění stanovených cílů 11
2.3 Technické požadavky 12

3 Deployované komponenty **13**
3.1 Motivace 13
3.2 Specifikace deployovaných komponent 13
3.3 Deployment komponenty 14
3.4 Získávání komponent 16
3.5 Základní deploymenty 16
3.6 Značkování 17

4 Práce s databází **19**
4.1 ORM v NCore 19
4.2 Základní manipulace s datovými objekty 22
4.2.1 Načtení objektu z databáze 22
4.2.2 Smazání objektu z databáze 23
4.2.3 Ukládání objektů do databáze 23
4.2.4 Modifikace objektů 23
4.3 Transakce 24
4.4 Exekutoři 25
4.5 NHibernate exekutor 25
4.6 ADO exekutor 26
4.7 DBFacade 26

4.8 Konfigurace NHibernate 28

5 Základní komponenty **29**
5.1 Logování 29
5.1.1 Logování v NCore 30
5.1.2 Vytváření vlastních komponent pro logování 30
5.2 Cachování 31
5.3 Číselníky 32

6 Tvorba vlastních komponent **33**

7 Podpora uživatelského rozhraní **36**
7.1 Hlavní elementy 36
7.2 Modul 37
7.3 Work item 37
7.4 View 38
7.5 Controller 40
7.6 Služba - ModuleManagerService 40
7.7 Business rule objects 42

8 Build NCore aplikace **43**
8.1 Konfigurace App.config 43
8.2 Třída Components 45

9 Ukázková aplikace Contact Manager **46**
9.1 Instalace 46
9.2 Struktura projektu 47
9.2.1 Projekt Contact 47
9.2.2 Projekt ContactManager 48
9.3 Závěr 48

10 Shrnutí **50**
10.1 Práce více týmů 51
10.2 Rychlost 51
10.3 Směry dalšího vývoje 51
10.4 Závěr 52

Literatura **53**

A Obsah CD **54**

Název práce: Framework pro vývoj databázových aplikací
Autor: Jan Blaha
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Jan Votrubec
e-mail vedoucího práce: votrubec@ufal.mff.cuni.cz

Abstrakt: Práce představuje referenční příručku frameworku NCore, který jsem vyvinul jako součást bakalářské práce v jazyce c#. NCore slouží programátorům zejména ke zjednodušení práce s relační databází. Dále pak obsahuje sadu komponent, které programátorům usnadní práci v oblasti cachování, logování, číselníků a také uživatelského rozhraní. V závěru je použitelnost frameworku předvedena na ukázkové aplikaci.

Klíčová slova: databáze, framework, ORM, NHibernate

Title: Framework for developement of database applications
Author: Jan Blaha
Department: Institute of formal and applied linguistics
Supervisor: Mgr. Jan Votrubec
Supervisor's e-mail address: votrubec@ufal.mff.cuni.cz

Abstract: Main goal of this work is to create reference guide to database framewrok NCore that I have developed in c# language as a part of the bachelor thesis. NCore is mainly useful for developers when working with database. Other goals of NCore are in fields like caching, logging, list tables and even user interface. In the end the framework usability is shown on simple application.

Keywords: database, framework, ORM, NHibernate

Kapitola 1

Úvod

V současné době potřebuje každá větší firma svuj vlastní podnikový informační systém, který usnadní její dynamický rozvoj. Takový informační systém může například zpracovávat odpracované hodiny zaměstnanců, generovat podklady pro mzdy zaměstnanců nebo nabízet další funkce specifické pro daný podnik. Tyto informační systémy jsou obvykle velice rozsáhlé, a proto může být jejich vývoj časově náročný. Navíc, pokud vývojový tým nedodržuje základní architektonická pravidla, může časem v systému zavládnout chaos. Je potřeba vývoj těchto systémů zrychlit a standardizovat.

Řešením jsou obvykle softwarové frameworky. Ty vnášejí do systému pevný řád a architektonická pravidla. Umožňují tak programátorům soustředit se na implementaci funkcí, které jsou od systému skutečně požadovány. Například vývojový tým používající softwarový framework se může při vývoji webových stránek pro bankovní společnost zaměřit na implementaci bankovního účtu a nemusí se starat o správu navigace mezi jednotlivými stránkami.

1.1 Softwarový framework

Jak je uvedeno v [5], softwarový framework představuje znovupoužitelnou architekturu pro softwarový systém. Je možné si jej představit jako skupinu abstraktních tříd a styl komunikace jejich instancí.

Softwarový framework bývá většinou objektově orientovaný, ale je možné jej implementovat i v neobjektovém jazyku. Může obsahovat pomocné programy, knihovny, skriptovací jazyky nebo jiný software, který usnadňuje programování částí projektu.

Frameworky jsou navrhované tak, aby usnadnily softwarový vývoj. Umožňují programátorům soustředit se na požadavky softwaru více než na to, jak systém celkově funguje. Nevýhodou frameworku může být jejich komplexnost, protože je potřeba strávit čas jejich učením. Tento obětovaný čas může být někdy pro vývoj menších aplikací zbytečný.

Někteří vývojáři si myslí, že použitím frameworku se jejich aplikace zbytečně „nafoukne" a přibude do ní spousta nepruhledného kódu. Zastávají často postoj: „ Jsme schopni si vše naprogramovat sami." Tento postoj je ale pro vývoj rozsáhlejších informačních systémů špatný. Uvedu proto seznam pěti nejdůležitějších důvodů, proč při vývoji aplikací používat již existující řešení v podobě frameworku.

1. **Používání frameworku umožní pracovat rychleji.**
   Rychlost je hlavní důvod, proč se frameworky stávají tak oblíbené. Při dobré znalosti používání frameworku je možné aplikace vyvinout mnohem rychleji.

2. **Používání frameworku dá psanému kódu strukturu.**
   Frameworky se nám snaží zjednodušit život tím, že poskytují univerzální způsob strukturalizace aplikace. Různé aplikace vyvinuté ve stejném frameworku mají podobnou strukturu. To je umožněno tím, že framework je systém, který určuje logický běh aplikace a neponechává jej jednotlivým programátorům, kteří by mohli do aplikace vnést zmatek. Díky tomu se mohou programátoři rychle zorientovat v již dokončených projektech.

3. **Při používání frameworku pracujete s nejnovějšími technologiemi.**
   Každý dobrý framework má kolem sebe skupinu nadšenců, kteří stále sledují svět nových informačních technologií a jimi framework vylepšují. Protože jsou frameworky většinou navrženy dostatečně obecně, není problém do nich nové technologie přidávat.

4. **Používáním frameworku získáte zdarma podporu.**
   Ke každému frameworku jsou k dispozici stovky stránek, návodů, diskuzí a spousty dalších informací. Nemusíte sami zjišťovat, jak se má framework chovat. Navíc nemusíte sepisovat zbytečné dokumentace, stačí použít existující materiály.

5. **Použití frameworku donutí psát lepší kód i začínající programátory.**
   Frameworky svým způsobem omezují programátory v jejich kreativitě. Nutí je například oddělovat práci s databází[1] od uživatelského rozhraní a logiky aplikace. Díky tomu dostává zdrojový kód standardní strukturu a styl.

1.2 Cíle práce

Ve světě Javy je softwarových frameworků celá řada, dobře známé jsou např. Struts [6] a Spring [7]. Ve světě .NETu již tolik možností není. Microsoft totiž razí myšlenku usnadnit vývoj programátorům co nejvíce již na straně jazyka a platformy. Tato řešení však nejsou na takové úrovni, aby mohla zcela zastoupit framework. Proto jsem se rozhodl vyvinout softwarový framework použitelný ve světě .NET.

Cílem práce je vyvinout softwarový framework v jazyce c#, usnadňující vývoj aplikací pracujících s databází. Vzniklý framework by měl splňovat následující požadavky:

- Umožnit jednoduchou práci s daty z relační databáze.
- Usnadnit rozšiřitelnost frameworku a jeho konfigurovatelnost.
- Poskytnout rozhraní pro zapisování ladicích zpráv.
- Poskytnout rozhraní pro ukládání do dočasné paměti cache.
- Usnadnit práci s uživatelským rozhraním, navigaci mezi obrazovkami a rozdělení aplikace na nezávislé části.
- Předvést základní funkčnosti framework na ukázkové aplikaci.

[1]Databází je v této práci myšlena vždy relační databáze.

Kapitola 2

NCore

NCore je framework, který jsem vyvinul jako součást bakalářské práce v jazyce c#. Programátorům slouží zejména ke zjednodušení práce s relační databází. NCore dále obsahuje sadu komponent, které programátorům usnadní práci v oblasti cachování, logování, číselníků a také uživatelského rozhraní. Dále v textu budu termínem framework označovat právě NCore.

2.1 Použité komponenty

Vyvinout na zelené louce framework, který by pokryl většinu nejčastěji požadovaných funkcí jedním programátorem, by bylo z časového hlediska velice náročné. Proto jsem se rozhodl co nejvíce využít již hotové nástroje a knihovny. NCore je vlastně takový deštník, který propojuje již existující knihovny do jednoho celku. Představu složení NCore z existujících řešení naznačuje obrázek 2.1.

NCore využívá následující komponenty:

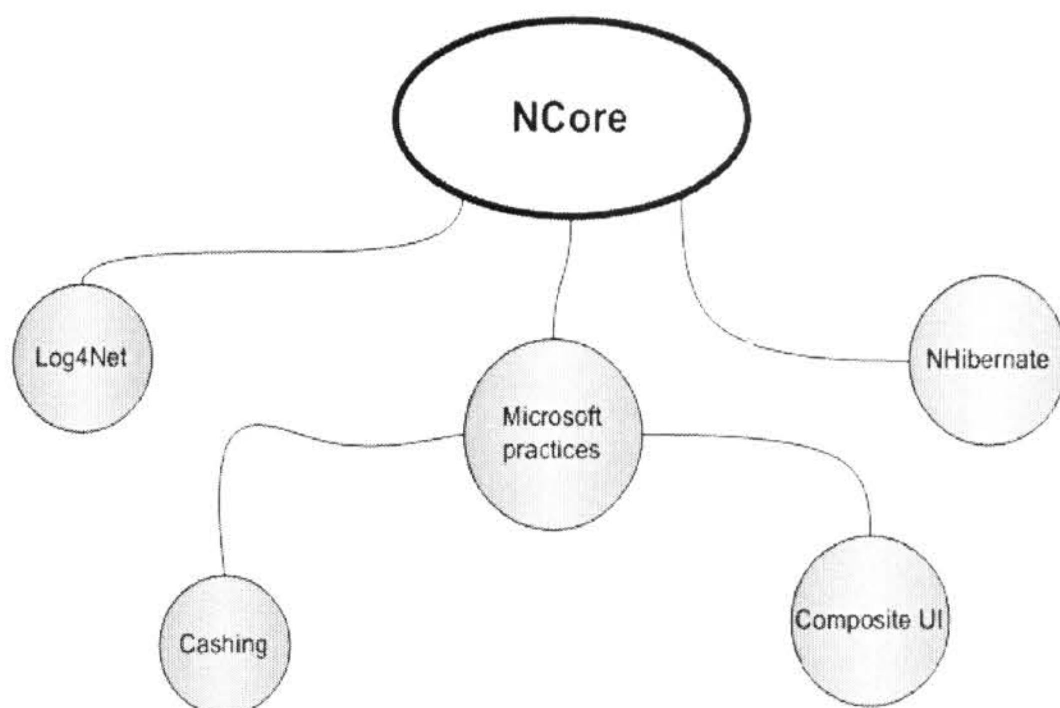
- **NHibernate** je framework umožňující pracovat s daty z relační databáze v prostředí jazyka c#.
- **Log4Net** je framework usnadňující zapisování ladicích zpráv.
- **Composite UI Application Block** slouží k tvorbě rozsáhlých uživatelských rozhraní.

- **Caching Application Block** poskytuje programátorům maximálně konfigurovatelnou komponentu umožňující ukládání do dočasné paměti.

NCore se zasluhuje o spojení těchto frameworků a knihoven v jeden celek. Použité komponenty jsou samostatně jen řešení jednotlivých problémů. Jako celek již NCore představuje koncové řešení pro vývoj aplikací.

Všechny použité komponenty mají samozřejmě různé způsoby konfigurování a používání. NCore zavádí jeden jediný obecný mechanizmus konfigurace, který aplikuje na všechny použité komponenty. Díky jednotnému mechanizmu pak uživatel nemusí znát všechny detaily jednotlivých komponent. Dokonce nemusí ani poznat, že právě používá externí komponentu, která je pouze součástí NCore.

Další přínos NCore je, že je možné jej okamžitě začít používat, protože obsahuje všechny potřebné knihovny. Stačí spustit projekt NCoreStartUp z přiloženého CD, jehož strukturu popisuje příloha A, a je možné začít vyvíjet aplikace.



Obrázek 2.1: Představa složení NCore z existujících řešení.

2.2 Splnění stanovených cílů

V této sekci popíšu, jak NCore splňuje jednotlivé cíle stanovené v kapitole 1.2.

NCore nabízí uživatelům zásadní usnadnění práce s relační databází. Umožňuje nadefinovat mapování c# třídy na tabulku databáze a následně pracovat s reálnými daty z databáze v prostředí jazyka c# bez použití SQL dotazů. Práce s databází je v NCore postavena na frameworku NHibernate. Práci s databází i frameworkem NHibernate se dále zabývá kapitola 4.

Rozšiřitelnost a konfigurovatelnost zajišťuje framework díky svému mechanizmu pro práci s komponentami. Celý framework je složen z dílčích částí nazvaných komponenty. NCore obsahuje například komponenty pro logování, cachování a další. Každá tato komponenta se inicializuje pomocí svého konfiguračního XML souboru. Jádro frameworku je pak mechanizmus, který vytváří instance komponent a inicializuje je pomocí dat z XML souborů. Jak se pracuje v NCore s komponentami popisuje kapitola 3.

NCore poskytuje velké množství možností jak zapisovat ladicí zprávy (logovat). Můžete například zapisovat do souboru, standardního výstupu i do databáze. Způsob zapisování zpráv se konfiguruje v XML souborech, takže je možné vypnout či zapnout zápis zpráv bez nutnosti překládání aplikace. NCore poskytuje pouze rozhraní pro zapisování ladicích zpráv. Samotnou implementaci ponechává nástroji log4net. Logováním se dále zabývá kapitola 5.1.

Framework poskytuje maximálně konfirugovatelnou dočasnou paměť. Je možné například nastavit velikost paměti, kolik prvků se má odstranit při přetečení paměti, nebo algoritmus hledající prvky, které se mají odstranit. NCore pouze poskytuje rozhraní pro práci s dočasnou pamětí. Samotnou implementaci přenechává knihovně Caching Application Block. O ukládání do dočasné paměti dále pojednává kapitola 5.2.

NCore obsahuje také podporu uživatelských rozhraní na bázi WinForms[1]. Usnadňuje například rozdělení aplikace na nezávislé moduly nebo navigaci

[1] WinForms je knihovna obsahující sadu uživatelských prvků, používaných v prostředí .NET.

mezi jednotlivými funkčnostmi. Podpora uživatelského rozhraní vychází z Composite UI application blocku (CAB) modifikovaného tak, aby co nejvíce zapadl do NCore. Práce s uživatelským rozhraním je popsána v kapitole 7.

Praktické použití NCore je předvedeno na ukázkové aplikaci Contact Manager. Tato aplikace simuluje jednoduchého správce kontaktů. Contact Manager využívá většinu výhod frameworku a může představovat první krok před skutečným používáním frameworku NCore. Aplikace Contact Manager je popsána v kapitole 9.

2.3 Technické požadavky

Softwarové frameworky jsou obecně komplikované a s jejich učením musí mít vývojáři trpělivost. Stejně je na tom i framework NCore. Můžete strávit i půl dne jeho studiem, než jej budete schopni efektivně používat.

Uvedu ještě seznam produktů a znalostí, které jsou k práci s NCore potřebné.

- Microsoft Visual Studio Express 2005.
- .NET Framework 2.0 Software Development Kit.
- Microsoft SQL Server 2005 Express.
- Minimální znalost jazyka c#.
- Minimální znalost jazyka SQL.
- Minimální znalost jazyka XML.
- Znalost objektově relačního mapování pomocí nástroje NHibernate.

Všechny použité komponenty i software potřebný k vývoji aplikací v NCore je možné používat i v komerční sféře. Samotný NCore bude šířen pod GPL open-source licencí a bude jej tedy možné libovolně používat.

Kapitola 3

Deployované komponenty

3.1 Motivace

Jeden ze základních problému, který musí autor databázového frameworku vyřešit, je oddělení dotazu do databáze od zdrojového kódu. NCore umísťuje dotazy do databáze do speciálních XML souboru nazvaných deploymenty. Aby bylo možné dotazy vykonávat různými způsoby, obsahuje každý deployment název třídy, která databázový dotaz obslouží. NCore pak obsahuje mechanizmus, který nalezne třídu spojenou s deploymentem, inicializuje ji a poskytne její instanci uživateli frameworku. Takto vytvořená instance už zná dotaz z deploymentu a je schopná jej vykonat. Takovéto spojení deploymentu a třídy je označeno jako deployovaná komponenta.

3.2 Specifikace deployovaných komponent

Deployovaná komponenta je základní kámen frameworku NCore umožňující vytvářet nezávislé, samostatně konfigurovatelné části aplikace. Většina funkčností frameworku je uvnitř těchto komponent. Existují například komponenty pro cachování, logování, přístup do databáze atd.

Deployovaná komponenta spojuje XML soubor se c# třídou. c# třída může např. zprostředkovávat spouštění dotazů na databázi a vyhodnocení její odpovědi. XML soubor k ní přidružený pak může obsahovat samotný SQL dotaz. Při práci s NCore nebudete většinou implementace těchto komponent vůbec vytvářet, protože jsou již naprogramované. Budete pouze vytvářet nové, nebo modifikovat existující XML soubory zvané deploymenty.

Deployovaná komponenta se tedy skládá ze dvou částí.

- Deployment komponenty obsahující konfigurační část komponenty uložený v XML souboru.
- Implementaci komponenty, kterou představuje c# třída.

Je potřeba podotknout, že k jedné implementaci komponenty zpravidla existuje několik jejích deploymentů. Například máme jednu implementaci komponenty realizující SQL dotazy do databáze a k ní tolik deploymentu, kolik chceme mít dotazů. Pro každý dotaz vytvoříme nový deployment, ve kterém dotaz specifikujeme. Tuto situaci zobrazuje obrázek 3.1.

3.3 Deployment komponenty

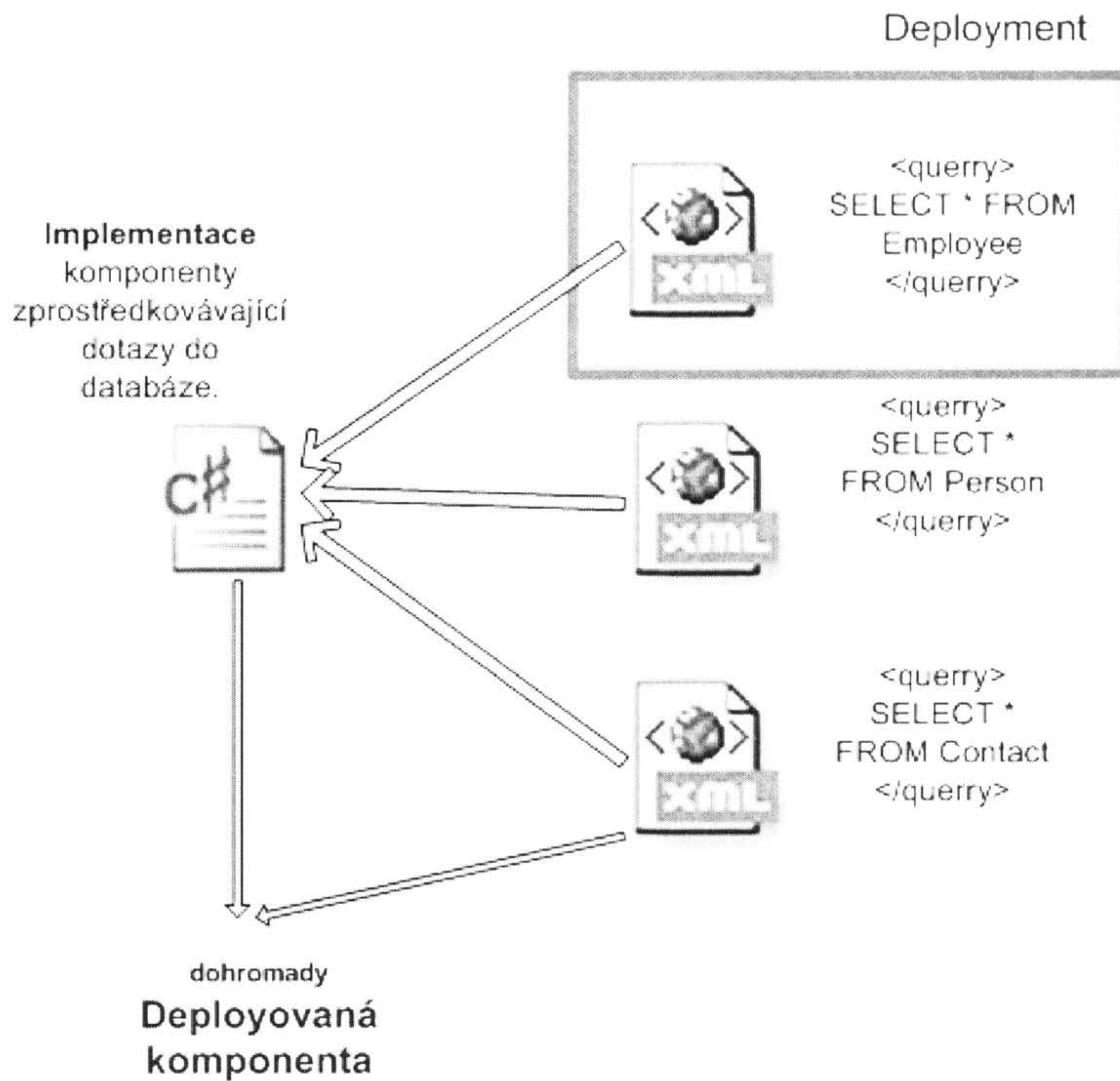
Deployovaná komponenta přináší univerzální způsob, jak načítat XML data do aplikace. Vyskytuje se téměř ve všech částech frameworku NCore a při jeho používání se s ní mnohokrát setkáte, proto je tedy důležité pochopit její princip. Seznámím vás blíže se strukturou deploymentu a ukážu, jak je propojen s implementací komponenty. Jeho ukázka je na obrázku 3.2. Každý takovýto deployment musí mít jméno ve formátu *.depl.XML. Podmínka na jeho koncovku je důležitá kvůli mechanizmu kompilace aplikace, kterým se zabývá kapitola 8.

Každý deployment musí mít kořenový uzel pojmenovaný „deployedComponent". V atributech tohoto uzlu jsou nastaveny základní parametry, které určují, ke které implementaci komponenty má být připojen. Jsou to atributy:

- name – název komponenty, musí být unikátní, jedině s pomocí názvu lze získat instanci komponenty,
- type – má strukturu „Typ, Assembly[1]" , určuje třídu, která představuje implementaci komponenty,
- factoryType – stejná struktura jako type, určuje třídu, která je zodpovědná za vytvoření instance komponenty a její inicializaci.

Hlavička vypadá na první pohled složitě, ale ve skutečnosti ji lze zkrátit jen na název a „značku" komponenty. Toto zkracování řeší tzv. značkování, o kterém pojednává kapitola 3.6.

[1] Assembly v prostředí .NET představuje knihovnu (dll).



Obrázek 3.1: Ukázka deployovaných komponent pro dotazy do databáze.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent
  name="GetPersons"
  type="GetPersons, Test"
  factoryType="NCore.DBComponents.Implementation.DAOComponentFactory,
              NCore.DBComponents">
  <querryString>
    <![CDATA[
   select * from person p1, person p2
   where p1.firstName = p2.firstName and
         p1.id != p2.id
    ]]>

  </querryString>
</deployedComponent>
```

Obrázek 3.2: Ukázka struktury deploymentu.

3.4 Získávání komponent

K získání deployované komponenty je potřeba znát její název uvedený v jejím deploymentu. V NCore je implementován mechanizmus získávání komponenty podle jejího jména. Veškerou práci od vyhledání deploymentu, konstrukce, až po inicializaci komponenty zapouzdřuje třída ComponentManager. Na obrázku 3.3 je ukázáno, jak získat komponentu DBFacade, pomocí které je možné přistupovat přímo do databáze.

```
IComponentManager manager = ComponentManagerFactory.GetComponentManager();
DBFacade facade = (DBFacade)manager.GetComponent("DBFacade");
facade.ExecuteQuerry("insert into...");
```

Obrázek 3.3: Ukázka získání deployované komponenty.

3.5 Základní deploymenty

NCore nabízí celou řadu základních deploymentů, tedy základních konfigurací komponent. Tyto deploymenty můžete modifikovat, a libovolně tak konfigurovat základní chování aplikace.

Základní deploymenty jsou:

- DBFacade.depl.xml – obsahuje konfiguraci databázového připojení, je nutné si ji vždy nakonfigurovat pro svou databázi, viz. 4.7.
- NHibernateSession.depl.xml - obsahuje konfiguraci nástroje NHibernate.
- InternalLogger.depl.xml – konfigurace loggeru[1] , který loguje dění v NCore.
- NCoreLog.depl.xml – předpřipravená konfigurace loggeru, který slouží k logování dění mimo NCore.
- SimpleCache.depl.xml – předpřipravený deployment cache.

3.6 Značkování

NCore nabízí způsob, jak hlavičku deploymentu zjednodušit. Díky značkování umožňuje NCore dát libovolné implementaci komponenty značku a tu pak používat v deploymentu namísto parametrů type a factoryType. Značky se ukládají v souboru sign.xml, tento soubor můžete libovolně editovat a přidávat do něj vlastní značky. Struktura souboru XML je zobrazena v obrázku 3.4.

```
<?xml version="1.0"encoding="utf-8"?>
<signs>
  <sign name="NHibernateDAOExecutor"
        type="NCore.DBComponents.Implementation.NHibernate.NHibernateDAOExecutor, NCore.DBComponents"
        factoryType="NCore.DBComponents.Implementation.NHibernate.DAONHibernateExecutorComponentFactory,
NCore.DBComponents"/>
  <sign name="DTO"
      type="NCore.DBComponents.Implementation.NHibernate.HibernateMapping, NCore.DBComponents"
     factoryType="NCore.DBComponents.Implementation.NHibernate.HibernateMappingFactory, NCore.DBComponents"/>
</signs>
```

Obrázek 3.4: Soubor sign.xml.

V souboru signs.xml je vždy pro každou značku uvedeno její jméno, typ implementace komponenty a typ třídy realizující vytváření komponenty. Tyto typy byste jinak bez značkování museli psát do každého deploymentu. S použitím značkování stačí do hlavičky uvést značku komponenty a její název. Ukázku použití značkování zobrazuje příklad 3.5.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent name="GetContactFromPhone"
                   sign="NHibernateDAOExecutor">
```

Obrázek 3.5: Ukázka použití značkování.

Kapitola 4

Práce s databází

NCore nabízí uživatelům zásadní usnadnění práce s relační databází. Umožňuje nadefinovat mapování c# třídy na tabulku databáze a následně pracovat s reálnými daty z databáze v prostředí jazyka c# bez použití SQL dotazů.

4.1 ORM v NCore

V současné době jsou data nejčastěji ukládána v relační databázi a zpracovávána nějakým objektově orientovaným jazykem. Problém je, že uložená data jsou jinak reprezentována v databázi a jinak v objektově orientovaném jazyce. Programátor tedy musí ke stejným datům přistupovat dvěma rozdílnými způsoby. Překonáváním těchto rozdílů vzniká režie, která zabírá až desítky procent času stráveného vývojem aplikace. Proto vznikla technika objektově relačního mapování (ORM), která usnadňuje převedení dat z prostředí relační databáze do prostředí OOP jazyka. Základní princip je takový, že jedna databázová tabulka představuje jednu třídu, jednotlivé řádky její instance. Jejich vzájemný převod je pak zprostředkován technikou ORM pomocí mapovacích informací, které programátor specifikuje pro danou třídu. NCore implementuje ORM pomocí open source nástroje NHibernate. Při používání NCore se bez znalosti NHibernate většinou obejdete, pouze při vytváření mapovacích schémat je znalost NHibernate nutná. Díky ORM je možné v NCore pracovat s daty i bez použití jediného SQL dotazu do databáze. Dále v textu bude termínem mapování myšleno právě ORM.

Samotné mapování je vloženo do deploymentů komponenty HibernateMap-

ping. Základní mapování objektu na tabulku z relační databáze ukážu na jednoduchém příkladu. Představte si, že potřebujete pracovat s daty z tabulky Person 4.1.

Person

	Column Name	Data Type
🔑	ID	bigint
	FirstName	varchar(50)
	Surname	varchar(50)
	BirthDate	datetime

Obrázek 4.1: Tabulka Person.

Co potřebujete, je namapovat tabulku na třídu DTOPerson. Předponou DTO je naznačeno, že se jedná o datový objekt. Tato předpona není nutná, ale dodržováním konvence, že se každému datovému objektu vloží před název předpona DTO, zůstane kód přehlednější. Datové objekty, které chcete mapovat na tabulky databáze, musí vždy dědit od třídy DTOBase. DTOBase zapouzdřuje mapovací data a také identifikátor záznamu v databázi. Datové objekty by měly obsahovat pouze data, nejlépe pouze property [1]. Žádná logika do datových objektů nepatří. Třída DTOPerson, kterou budeme mapovat na tabulku Person, je na příkladu 4.2.

Třídu DTOPerson nyní namapujeme na tabulku. Mapování se vytváří pomocí deploymentu k implementaci komponenty HibernateMapping nebo značky DTO. Konkrétní mapování je na příkladu 4.3.

Z ukázky je vidět, že do hlavičky deploymentu přibyla nová položka. InitOnStartUp slouží k tomu, aby ComponentManager věděl, že má tuto komponentu inicializovat při startu aplikace. Tento atribut je pro mapovací deploymenty povinný, proto jej nezapomeňte nastavit. V deploymentu také přibyl, na první pohled komplikovaný, uzel hibernate-mapping. Tento uzel popisuje mapování tabulky Person na třídu DTOPerson. Přesný popis struktury mapování je nad rámec této práce, všechny potřebné informace jsou v [2]. Mapování vkládané do deploymentů může používat všech technik

[1] Property jsou speciální konstrukce jazyka c#, sloužící k zapouzdřování vlastností tříd.

```
public class DTOPerson : DTOBase
{
    private string firstName;
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    private string surname;
    public string Surname
    {
        get { return surname; }
        set { surname = value; }
    }

    private DateTime birthDate;
    public DateTime BirthDate
    {
        get { return birthDate; }
        set { birthDate = value; }
    }
}
```

Obrázek 4.2: Třída DTOPerson.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent name="DTOPerson"
                   sign="DTO"
                   initOnStartUp="true">

  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
    <class name="Example.DTOPerson, Example"table="Person"lazy="false">
      <id name="Id"type="Int32"column="Id">
        <generator class="identity">
        </generator>
      </id>
      <property name="FirstName"column="FirstName"length="50"type="string"/>
      <property name="Surname"column="Surname"length="50"type="string"/>
      <property name="BirthDate"column="BirthDate"type="DateTime"/>
    </class>
  </hibernate-mapping>

</deployedComponent>
```

Obrázek 4.3: Mapování tabulky Person na třídu DTOPerson.

NHibernate s jedinou výjimkou. NCore nepodporuje techniku líného načítání objektů. Proto je nutné v každém mapování nastavit atribut lazy na hodnotu false.

Nyní po rekompilaci aplikace máme připravený objekt namapovaný na databázi. V další kapitole ukážu, jaké možnosti NCore nabízí při práci s takovými objekty.

4.2 Základní manipulace s datovými objekty

Základní CRUD[2] operace s datovými objekty zapouzdřuje statická třída NHibernateFacade. Tato třída obsahuje metody, které skutečně zrychlí vývoj databázových aplikací. S touto třídou zvládnete bez použití jediného SQL skriptu načítat objekty z databáze, vkládat objekty do databáze a mnoho dalších operací. To vše je možné díky tomu, že pokud existuje mapování objektu do databáze, framework zvládne SQL skripty automaticky generovat.

4.2.1 Načtení objektu z databáze

Načtení jednoho objektu z databáze zapouzdřuje metoda Load třídy NHibernateFacade. Její parametry jsou:

- Typ datového objektu, který chcete načíst. K tomuto objektu musí existovat mapování vytvořené způsobem naznačeným v předchozí kapitole.
- Primární klíč záznamu, který chcete načíst.

Metoda Load obsahuje i alternativu pro práci s generickými typy[3]. Použití metody Load ukazuje příklad 4.4.

```
//load person with id 958
DTOPerson person = (DTOPerson)NHibernateFacade.Load(typeof (DTOPerson), 958);
//load person with id 957 using generic type of method Load
DTOPerson personGenerics = NHibernateFacade.Load<DTOPerson>(957);
```

Obrázek 4.4: Přečtení objektu DTOPerson z databáze.

[2]CRUD jsou označovány operace vytvoř, přečti, modifikuj, smaž (angl. create, read, update, delete).

[3]Generické typy v .NET představují obdobu šablon jazyka c++.

4.2.2 Smazání objektu z databáze

Smazání objektů z databáze zapouzdřuje metoda Delete. Její jediný parametr je instance objektu, který chceme smazat z databáze. Ukázka smazání objektů z databáze je vidět na obrázku 4.5.

```
//load person
DTOPerson person = (DTOPerson)NHibernateFacade.Load(typeof(DTOPerson),
                                                    958);
//remove person from db
NHibernateFacade.Delete(person);
```

Obrázek 4.5: Smazání objektu z databáze.

4.2.3 Ukládání objektů do databáze

Ukládání objektů zajišťuje metoda Save. Její jediný parametr je instance objektu, který chceme uložit do databáze. Návratová hodnota je výsledný objekt, který byl uložen do databáze. Ukládání objektů ukazuje příklad 4.6.

```
DTOPerson person = new DTOPerson();
person.BirthDate = DateTime.Parse("01-27-1985");
person.FirstName = "Jan";
person.Surname = "Blaha";
object id = NHibernateFacade.Save(person);
```

Obrázek 4.6: Ukládání nových objektů do databáze.

4.2.4 Modifikace objektů

Modifikaci objektů zajišťuje metoda Update. Její jediný parametr je instance objektu, který chceme modifikovat v databázi. Modifikaci objektu ukazuje příklad 4.7.

Alternativa metody Update a Save je metoda SaveOrUpdate, která objekt uloží, pokud ještě v databázi neexistuje, a modifikuje, pokud již existuje.

```
//load person with id 958
DTOPerson person = (DTOPerson)NHibernateFacade.Load(typeof (DTOPerson),
                                                     958);
//modify person
person.Surname = "Novák";
//update old person with new one
NHibernateFacade.Update(person);
```

Obrázek 4.7: Modifikace objektů v databázi.

4.3 Transakce

NCore samozřejmě podporuje transakce. Transakce jsou nezbytné pro zachování konzistentnosti databáze při vykonávání několika závislých, po sobě jdoucích databázových operací. Zpracovávají se pomocí třídy NHibernateFacade. Základní metody pro jejich zpracování jsou BeginTransaction, CommitTransaction a RollbackTransaction. Ukázka práce s transakcemi je vidět na obrázku 4.8.

```
//start transaction
NHibernateFacade.BeginTransaction();
try
{
    //more updates here
    NHibernateFacade.SaveOrUpdate(person);
}
catch (Exception ex)
{
    log.Error("Unable to create contact. ", ex);
    NHibernateFacade.RollbackTransaction();
    return false;
}
//save transaction
NHibernateFacade.CommitTransaction();
```

Obrázek 4.8: Práce s transakcemi.

4.4 Exekutoři

NCore separuje DML[4] dotazy do deploymentů, díky tomu je možné je měnit a ladit bez nutnosti rekompilace. K získávání seznamů objektů z databáze podle různých kritérií slouží tzv. exekutoři. Existují dva základní typy exekutorů.

4.5 NHibernate exekutor

NHiberbate exekutor slouží k získávání seznamu objektů stejného typu z databáze. K těmto objektům musí existovat mapování. Data, která se mají do seznamu vybrat, se popisují pomocí jazyka Hibernate Query Language (HQL)[5]. HQL je velice podobný jazyku SQL a jeho použití je intuitivní. Složitější konstrukce HQL jako spojení více tabulek či agregační funkce jsou popsány v dokumentaci NHibernate [2]. HQL se nezapisuje nikam jinam než do deploymentů. Je potřeba vytvořit nový deployment k implementaci komponenty NHibernateDAOExecutor, zapsat svůj HQL dotaz do elementu querry a nakonec už jen získat komponentu standardním způsobem pomocí ComponentManageru a zavolat metodu Execute. Ta nám vrátí požadovaný seznam objektů. Nejlépe celý postup ukáže následující ukázka. Deployment exekutoru je vidět na obrázku 4.9 a samotné spuštění na obrázku 4.10.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent name="GetPersonByName"
                   sign="NHibernateDAOExecutor">
  <querryString>
    <![CDATA[
    select person from DTOPerson person
    where
     phone.Surname =:surname
    ]]>

  </querryString>

</deployedComponent>
```

Obrázek 4.9: Deployment k NHibernate exekutorovi.

[4]DML(Data manipulation language) jsou SQL příkazy select, insert, delete, update.
[5]HQL implementuje nástroj NHibernate.

```
//prepare params for executor
Hashtable param = new Hashtable();
param.Add("surname", "Blaha");

//get executor from ComponentManager
IDAOExecutor<DTOContact> executor =
    (IDAOExecutor<DTOContact>)ComponentManager.GetComponent(GetPersonByName, param);

//execute executor - get every person with surname Blaha
IList<DTOPerson> list = executor.Execute();
```

Obrázek 4.10: Spuštění NHibernate exekutoru.

4.6 ADO exekutor

ADO[6] exekutor slouží také k získání seznamu objektů stejného typu. Tyto objekty už nemusí být namapované. Data, která se mají do seznamu vybrat, se popisují pomocí libovolného SQL. Problém však je, že pokud nejsou objekty namapované, je nutné, aby programátor sám naplnil objekty daty, které získá z SQL dotazu. Z tohoto důvodu je doporučeno používat spíše NHibernate exekutor a Ado exekutor pouze v případech, kdy vám nestačí možnosti HQL, nebo se jedná o rychlostně kritickou oblast. SQL dotazy se zapisují opět do deploymentů do elementu querry. Na rozdíl od NHibernate exekutorů však implementaci komponenty nemůžete uvést žádnou standardní. Musíte nejprve vytvořit vlastní implementaci komponenty, která bude překládat data z databáze do objektů. Takovéto implementace dědí od třídy AdoDAOExecutor a překrývají její metodu ReadObject. Nejlépe vše ukáže následující příklad. Deployment Ado exekutoru je na obrázku 4.11 a implementace komponenty je na obrázku 4.12. Exekutor se pak spouští stejným způsobem jako NHibernate exekutor. Pomocí ComponentManagera se získá instance exekutoru, a pak už jen stačí zavolat metodu Execute.

4.7 DBFacade

Komponenta DBFacade tvoří přístupový most do databáze. NCore obsahuje její standardní deployment v souboru DBFacade.depl.xml. Ten obsahuje parametry pro nastavení databázového spojení. Ten nejdůležitější parametr je connectionString, který obsahuje připojovací řetězec do databáze. Pomocí DBFacade můžete přímo spouštět skripty na databázi, tento způsob ale není

[6] ADO je skupina .NET tříd, které slouží programátorům k základní práci s databází.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent
  name="GetPersons"
  type="GetPersons, Test"
  factoryType="NCore.DBComponents.Implementation.DAOComponentFactory,
              NCore.DBComponents">
  <querryString>
    <![CDATA[
   select * from person p1, person p2
   where p1.firstName = p2.firstName and
         p1.id != p2.id
    ]]>

  </querryString>
</deployedComponent>
```

Obrázek 4.11: Deployment k Ado exekutorovi.

```
public class GetPersons : AdoDAOExecutor<DTOPerson>
{
    protected override DTOPerson ReadDTO(IDataReader reader)
    {
        DTOPerson person = new DTOPerson();

        person.Id = (int)reader["Id"];
        person.BirthDate = (DateTime) reader["BirthDate"];
        person.FirstName = (string) reader["FirstName"];
        person.Surname = (string) reader["Surname"];

        return person;
    }
}
```

Obrázek 4.12: Implementace Ado exekutoru.

doporučeno používat. Všechny skripty by měly být umístěny v deploymentech a jejich spouštění by měli zajišťovat exekutoři. Ukázka práce s DBFacade je vidět na obrázku 3.3.

4.8 Konfigurace NHibernate

NHibernate můžete libovolně konfigurovat pomocí standardního deploymentu NHibernateSession.depl.xml. Jediná část konfigurace NHibernate, kterou nejspíše použijete, bude konfigurace logu. Ta je obsažena v uzlu log4net a její struktura odpovídá struktuře konfigurace nástroje log4net. Dokumentaci k tomuto nástroji najdete v [1].

Kapitola 5

Základní komponenty

5.1 Logování

Jeden z požadavků na jakékoliv vývojové prostředí je, aby bylo schopné efektivně ladit kód (debugovat). Během vývoje má programátor možnost využití mnoha technik, jako jsou breakpointy, výpis do standardního výstupu apod. Představte si ale, že aplikace běží na serveru, kde se nechová správně. Pokud se rozhodnete zapisovat běh aplikace do log souborů, budete pak muset při nasazování aplikace do produkce zakomentovávat kód, aby se aplikace logováním zbytečně nezpomalovala. Další možností je použití nějakého boolean flagu, který bude určovat, zda se má logovat či nikoliv. Při změně tohoto flagu však musíte aplikaci rekompilovat, což je zcela zbytečné. Při využití dobrého nástroje pro logování můžete dynamicky povolovat nebo zakazovat logování bez nutnosti rekompilace. Navíc můžete zapisovat do logu zprávy s různou úrovní důležitosti, nebo dynamicky měnit jejich formát, určovat, zda se bude zapisovat do databáze, souboru či konzole. Jedním z takovýchto nástrojů je log4net [1].

Právě na tomto API je postavené logování v NCore. NCore nabízí základní nastavení logování, které lze používat bez bližší znalosti log4net. Pokud však budete chtít logování v NCore konfigurovat, doporučuji si blíže nastudovat dokumentaci log4net [1].

5.1.1 Logování v NCore

Logování je umístěno v jmenném prostoru NCore.Logging. Nejprve ukážu, jak jednoduché je logování v NCore na obrázku 5.1.

```
ILog log = LogManager.GetLogger(this.GetType(), "NCoreLogger");
log.Info("Logging something on info level...");
```

Obrázek 5.1: Ukázka zápisu do logu.

K logování se používají komponenty, tzv. loggery. Nejjednodušeji je získáte pomocí třídy LogManager a její metody GetLogger. K identifikaci loggeru jsou potřeba dva parametry. První je typ loggeru, ten může být libovolný a slouží k identifikaci, kde se do logu zapisovalo. Druhý parametr je jméno loggeru, což je jméno deployované komponenty. NCore poskytuje základní přednastavený deployment NCoreLogger, který můžete libovolně používat.
Zprávy do logu můžou být zapsány na třech úrovních:

- Info úroveň – označuje zprávy s vysokou důležitostí, například spuštěna aplikace nebo navázáno spojení s databází.
- Debug úroveň – označuje zprávy s nízkou důležitostí, například obnovení stránky.
- Error úroveň – slouží k zapisování chybových hlášení a výjimek.

Díky těmto úrovním je možné filtrovat zprávy do různých souborů podle jejich úrovně. Přednastavený logger NCoreLogger například zapisuje do tří souborů podle úrovně zprávy.

NCore loguje své chování do interního logu. Styl interního logování můžete libovolně konfigurovat v deploymentu komponenty InternalLogger.

5.1.2 Vytváření vlastních komponent pro logování

Můžete si samozřejmě vytvořit svoje vlastní loggery a libovolně si je nakonfigurovat. Stačí vytvořit deployment k implementaci komponenty Logger. Jeho název pak stačí předat LogManageru a můžete zapisovat. Deployment nově nadefinovaného loggeru ukazuje obrázek 5.2.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent
  name="MyLogger"
  type="NCore.Log.Implementation.Logger, NCore.Log"
  factoryType="NCore.Log.Implementation.LoggerComponentFactory, NCore.Log"
  >

  <log4net>
      <!-- konfigurace log4net -->
  </log4net>

</deployedComponent>
```

Obrázek 5.2: Nově nadefinovaný logger.

Deploymenty loggeru mají základní strukturu, navíc však obsahují uzel log4net, který obsahuje konfiguraci loggeru. Tato konfigurace je stejná jako ve známém nástroji log4net. Popis struktury konfigurace a všechny její možnosti naleznete v [1].

5.2 Cachování

Cachování slouží k dočasnému ukládání dat. Má pouze omezenou kapacitu, proto není zaručené, že bude obsahovat vždy všechna data, která byla do ní vložena. Pomocí cache můžete mnohonásobně zvýšit efektivitu práce s daty. S cache se pracuje opět pomocí deployovaných komponent. Stačí si vytvořit deployment k implementaci komponenty Cache, získat jí standardně pomocí ComponentManagera a můžete začít ukládat. Nejlépe vše ukáže příklad 5.3.

```
IComponentManager manager = ComponentManagerFactory.GetComponentManager();
//get cache component
cache = (ICache)manager.GetComponent("SimpleCache");
//try to read data from cache
items = (IList) cache.GetData("dataKey");
if (items == null)//data arent in cache, read from db
items = NHibernateFacade.LoadAll(typeof (DTOData));
```

Obrázek 5.3: Práce s cache.

Jak dlouho cache svá data uchovává v paměti apod. můžete konfigurovat pomocí parametrů deploymentu cache. NCore poskytuje standardní deployment SimpleCache, který je určen k libovolnému použití.

5.3 Číselníky

Číselníky jsou seznamy dat stejného významu. Číselníky můžou být např. dny v týdnu, pohlaví apod. NCore poskytuje jednoduché API pro práci s číselníky. Pro vytvoření číselníku v NCore je potřeba vykonat několik kroků.

1. Vytvořit datový objekt, který bude představovat položku číselníku. Tento objekt musí být potomek DTOCodeTableItem a musí k němu existovat mapování.

2. Vytvořit deployment k implementaci CodeTable. Tento deployment slouží ke konfiguraci samotného číselníku. Je potřeba mu přidat parametr itemType. Jeho hodnota bude typ objektu, který byl vytvořen v předchozím kroku, tedy objekt představující položku číselníku. Typ objektu se zapisuje standardním způsobem používaným v NCore: celé jméno, assembly.

Jak může deployment číselníku vypadat, najdete v obrázku 5.4.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent name="PersonTypeCodeTable"
                   type="NCore.CodeTables.Implementation.CodeTable, NCore.CodeTables">

  <parametrs>
    <parametr key="itemType"value="Contact.API.DTOPersonType, Contact"></parametr>
  </parametrs>

</deployedComponent>
```

Obrázek 5.4: Práce s cache.

Protože jsou číselníky cachované, nemusíte se obávat, že jejich opakované načítání zatíží vaši aplikaci.

Kapitola 6

Tvorba vlastních komponent

Architektura deployovaných komponent je natolik obecná, že vám nic nebrání vytvořit si vlastní komponenty.

Nejjednodušší komponenty jsou ty, které zpracovávají ze svých deploymentů pouze parametry. Mechanizmus získávání parametrů je totiž do každé komponenty automaticky zahrnut. Parametry jsou v deploymentech jen jednoduché dvojice hodnota, klíč. Vytvoření takovéto komponenty ukážu na obrázku. Představte si aplikaci, která má mnoho různých typů výstupů a velké množství z nich jsou data. Potřebujete, aby měla data jednotný formát a také, aby tento formát bylo možné změnit bez nutnosti kompilace aplikace. Např. datum v anglické verzi aplikace bude mít jiný formát než v české. Pro formátování dat vytvoříme komponentu DateFormatProvider. Jak by mohl vypadat její deployment, ukazuje příklad 6.1.

```
<?xml version="1.0"encoding="utf-8"?>
<deployedComponent
    name="CzechFormat"
    type="Startup.Implementation.DateFormatProvider ,Startup">

  <parametrs>
    <parametr key="dateFormat"value="dd.mm.yyyy"/>
  </parametrs>

</deployedComponent>
```

Obrázek 6.1: Deployment pro DateFormatProvider.

Jediné, co budeme zatím potřebovat konfigurovat, bude formát data. Tento formát specifikuje parametr „dateFormat“. Nyní vytvoříme implementaci. Každá implementace komponenty musí být potomek třídy Component-

Base. Komponenty mají přístup k datům ze svých deploymentů pomocí property Description, která vrací rozhraní IDeployedDescription. Z něj máme přístup na parametry specifikované v deploymentu, tedy i na parametr „dateFormat". Celá implementace je vidět na obrázku 6.2.

```
namespace Startup.Implementation
{
    public class DateFormatProvider : ComponentBase
    {
        private string dateFormat;

        //parametr key in deployment
        private const string FORMAT_PARAM = "dateFormat";

        public override void OnInit(IComponentDescription description)
        {
            //get data from deployment
            dateFormat = (string)description.Parametrs[FORMAT_PARAM];
        }

        public string FormateDateTime(DateTime date)
        {
            return date.ToString(dateFormat);
        }

        public string DateFormat
        {
            get { return dateFormat; }
        }

    }
}
```

Obrázek 6.2: Implementace DateFormatProvider.

Nyní už máme implementaci i deployment, můžeme tedy začít komponentu používat. Instanci komponenty získáme standardním způsobem pomocí ComponentManagera a můžeme formátovat, postup ukazuje obrázek 6.3. Velká výhoda použití deployované komponenty je, že pokud potřebujeme na různých místech aplikace používat různý formát, stačí nám vytvořit další deployment a používat pak DateFormatProvider ve dvou verzích.

```
IComponentManager manager =(IComponentManager)
                ComponentManagerFactory.GetComponentManager();
DateFormatProvider provider = (DateFormatProvider)manager.GetComponent("CzechFormat");
textBoxNow.Text = provider.FormatDateTime(DateTime.Now);
```

Obrázek 6.3: Práce s komponentou DateFormatProvider.

Pokud vám nestačí konfigurace pomocí parametrů a potřebujete například vkládat do deploymentů celé XML uzly, NCore nabízí postup pro vytvoření i těchto komponent. Potřebujete navíc vytvořit:

- Třídu, která bude představovat data z deploymentů. Tato třída musí být potomek DeployementDescription.
- Třídu, která bude zodpovědná za zpracování deploymentu. Tato třída musí být potomek ComponentFactoryBase. Překryje metodu DoFinishDeployment, ve které zpracuje deployment.

Kapitola 7

Podpora uživatelského rozhraní

NCore obsahuje elegantní podporu uživatelských rozhraní na bázi Win-Forms [4]. Rozděluje projekt na nezávislé moduly tak, aby umožnil jednotlivým vývojovým týmům pracovat současně na více modulech. Modul představuje skupinu funkčností, kterým se říká work items. Work items dále obsahují jednotlivé formuláře.

Podpora uživatelského rozhraní v NCore vychází z Composite UI application blocku (CAB), modifikovaným tak, aby co nejvíce zapadl do NCore. CAB vznikl v Microsoft and practicess a je to framework usnadňující vývoj uživatelských rozhraní. Jeho dokumentaci je možné nalézt v [8]. Snažil jsem se, aby uživatel při práci s uživatelským rozhraním v NCore nemusel příliš znát detaily CAB. Tento framework je celkem složitý a jeho pochopení zabere více času. S NCore je možné jej používat velice snadno a zároveň se držet základních architektonických pravidel.

7.1 Hlavní elementy

- **Modul** představuje jednu nezávislou jednotku. Je to skupina funkčností, které tvoří jeden logický celek.
- **Work item** představuje jednu funkčnost, jako např. zobrazení seznamu zaměstnanců a vytvoření zaměstnance.
- **View** je prezentační část představující formulář nebo user control[1].

[1] User control představuje v .NET skupinu ovládacích prvků. Formulář je pak složen z jednotlivých user control nebo samotných ovládacích prvků.

- **Controller** slouží k obsloužení událostí vzniklých na view spojených s controllerem.
- **Služba** slouží k propojení všech elementů.
- **Business rule object** je objekt obsahující logiku aplikace, slouží k oddělení GUI[2] od databáze a logiky aplikace.

7.2 Modul

Modul představuje jednu nezávislou jednotku. Je to skupina funkčností, které tvoří jeden logický celek. V aplikaci pro správu zaměstnanců můžeme mít například modul zaměstnanec, docházka, nastavení atd.

Každý modul by měl být reprezentován jedním projektem v Microsoft Visual Studiu(VS). Moduly se registrují v souboru ProfileCatalog.xml, viz. ukázka 7.1.

```
<?xml version="1.0"encoding="utf-8"?>
<SolutionProfile xmlns="http://schemas.microsoft.com/pag/cab-profile">
  <Modules>
    <ModuleInfo AssemblyFile="Contact.dll"/>
  </Modules>
</SolutionProfile>
```

Obrázek 7.1: ProfileCatalog.xml.

Fyzický představitel modulu je potomek třídy NCoreModule. Jak jsem uvedl, modul obsahuje všechny work items pracující s jednou konkrétní nezávislou částí aplikace. Work items se registrují v metodě RegisterWorkItem. Ukázka registrování work items je na obrázku 7.2.

7.3 Work item

Work item představuje jednu funkčnost, jako např. zobraz seznam zaměstnanců a vytvoř zaměstnance. Obsahuje všechny pohledy (view), které

[2] GUI (graphical user interface) je rozhraní, které přímo používá uživatel ke komunikaci s aplikací.

```
public class ContactModule : NCoreModule
{
    public override void RegistrWorkItems()
    {
        //registr all workitems that use contacts
        RegistrWorkItem(typeof (ContactListWorkItem));
        RegistrWorkItem(typeof (CreateContactWorkItem));

        //run main usecase
        Manager.LaunchWorkItem(typeof(ContactListWorkItem));
    }
}
```

Obrázek 7.2: Registrování work itemů.

se v dané funkčnosti používají. Fyzicky je work item potomek třídy NCoreWorkItem. Work item si musí zaregistrovat všechny view, které chce používat. View si registruje v metodě OnInitialization. Registrování view je ukázáno na obrázku 7.3.

```
public class ContactListWorkItem : NCoreWorkItem
{
    protected override void OnInitialization()
    {
        //set workitem localized name
        WorkItemLocalizedName = ContactListResources.WorkItemName;
        //registr main view
        RegistrView(typeof(ContactListView), true);
    }
}
```

Obrázek 7.3: Registrování view.

7.4 View

View může být libovolný formulář či user control. Na view klade NCore jediný požadavek. Každé view musí obsahovat property pojmenovanou Controller, která slouží ke spojení view a controlleru. Ukázka implementace view je na obrázku 7.4.

```
public partial class ContactListView : UserControl
{
    private ContactListController controller;

    [CreateNew]
    public ContactListController Controller
    {
        set
        {
            controller = value;
            controller.View = this;
        }
        get
        {
            return controller;
        }
    }

    public ContactListView()
    {
        InitializeComponent();
    }
}
```

Obrázek 7.4: Ukázka spojení view a controlleru.

7.5 Controller

Controller slouží ke zpracovávání událostí od uživatele, které se vyskytly na view připojeném ke controlleru.

View a controller jsou velice úzce propojeny, tuto vazbu musí vždy programátor specifikovat. View je propojen s controllerem pomocí odkazu uloženého v property Controller. Controller specifikuje, s jakým typem view bude pracovat pomocí svého generického typu.

Controller je potomek třídy NCoreController. Tato třída poskytuje svým potomkům velké množství metod, které jim usnadní práci. Například přístup na view, spouštění work items a podobně.

NCoreController poskytuje dvě základní metody, které mohou jeho potomci překrýt. Jsou to metody OnAfterLoad a InitializeEvents.

- **OnAfterLoad** je spouštěna pokaždé, když je spuštěno view asociované ke controlleru, nebo když se na něj přesune uživatel aplikace. Tato metoda je vhodná pro načtení dat z databáze apod.

- **InitializeEvents** je spouštěna pouze jednou, vždy když je vytvořeno asociované view. V této metodě je vhodné přidat handlery na události vzniklé na view.

Každý controller vždy pracuje s jedním datovým objektem, který reprezentuje jeho stav. K tomuto objektu přistupuje controller pomocí property MainDTO.

Ukázka implementace controlleru je v obrázku 7.5.

7.6 Služba - ModuleManagerService

Znalost služeb není pro používání NCore potřebná, proto nebudu způsob jejich použití a implementace popisovat. NCore obsahuje jedinou službu ModuleManagerService, ke které můžete přistupovat z controllerů, work items i modulů. ModuleManagerService je služba usnadňující spouštění jednotlivých funkčností. Obsahuje metody pro spouštění work items, jejich zavírání i přepínání mezi jednotlivými view.

ModuleManagerService umožňuje předat jednotlivým funkčnostem hlavní objekt, se kterým mají pracovat. Work items k tomuto objektu přistupují pomocí property MainDTO. Předávání datového objektu do funkčnosti ukazuje obrázek 7.6.

```
public class CreateTaskController : NCoreController<CreateTaskView>
{
    public override void InitializeEvents()
    {
        //here subscribe event handlers
        //View property is main access to controls
        View.btnSave.Click += new EventHandler(btnSave_Click);
    }

    public override void OnAfterLoad()
    {
        //load data and set bindings here
        View.taskBindingSource.DataSource = MainDTO;
    }

    private void btnSave_Click(object sender, EventArgs e)
    {
        //save MaindDTO using BRO methods
    }
}
```

Obrázek 7.5: Ukázka implementace controlleru.

```
Manager.LaunchWorkItem(typeof (CreateContactWorkItem),
                       typeof (CreateContactWorkItem).FullName + dto.Id, dto);
```

Obrázek 7.6: Ukázka spouštění work itemů.

7.7 Business rule objects

Implementovat aplikace, které používají GUI komponenty pracující přímo s databází, není dobré. Aplikace stavěné tímto způsobem nejsou objektově orientované. Takovéto dvouvrstvé aplikace porušují jeden ze základních principů objektově orientovaného návrhu a to zapouzdřenost. Zapouzdřenost umožňuje klientovi pracovat s objektem bez znalosti jeho implementačních detailů, tedy umožňuje spojovat objekty pomocí slabých vazeb (loose coupling). V takových dvouvrstvých aplikacích jsou klient a databáze velice silně spojeni. GUI, business logika a SQL jsou pak protkány v jednom místě. Výsledek je neudržitelná aplikace, protože každá změna databáze může kaskádově vyvolat nepředvídatelné selhání aplikace.

Daleko lepší způsob je oddělit práci s databází a business logiku[3] do tzv. business rule objektů (BRO). BR objekty představují vždy reálnou entitu, jako třeba zaměstnanec, produkt, osoba. BR objekty úplně zapouzdřují chování očekávané od entity, již představují.

NCore nabízí třídu BROBase jako předka pro všechny BR objekty. Ta přijímá jako parametr konstruktoru entitu, se kterou má pracovat. Pro práci s databází a pro zapouzdření business logiky používejte výhradně BR objekty. Neumísťujte žádnou logiku na controller! Ten má sloužit pouze k práci s view.

[3]Business logika nemá přesnou definici, ale obecně je business logikou označována vrstva v 3-vrstvé aplikaci zpracovávající data z databáze pro vrstvu uživatelského rozhraní.

Kapitola 8

Build NCore aplikace

Aplikace pracující s NCore používají speciální mechanizmus spouštění buildu. Z optimalizačních důvodů je potřeba, aby aplikace před spuštěním měla seznam deploymentů a jeho umístění. Seznam deploymentů vytváří aplikace NCore.Building.exe. Ta před každým buildem prochází všechny projektové adresáře a hledá soubory *.depl.xml. Umístění deploymentů a jejich názvy ukládá do mapovacího souboru mapping.xml, ten je pak využíván po spuštění aplikace. Spouštění aplikace NCore.Building.exe je nutné manuálně nastavit jako pre-build událost ve Visual Studiu. Pre-build události se nastavují ve vlastnostech spouštěcího projektu. Nastavování pre-build event ukazuje obrázek 8.1.

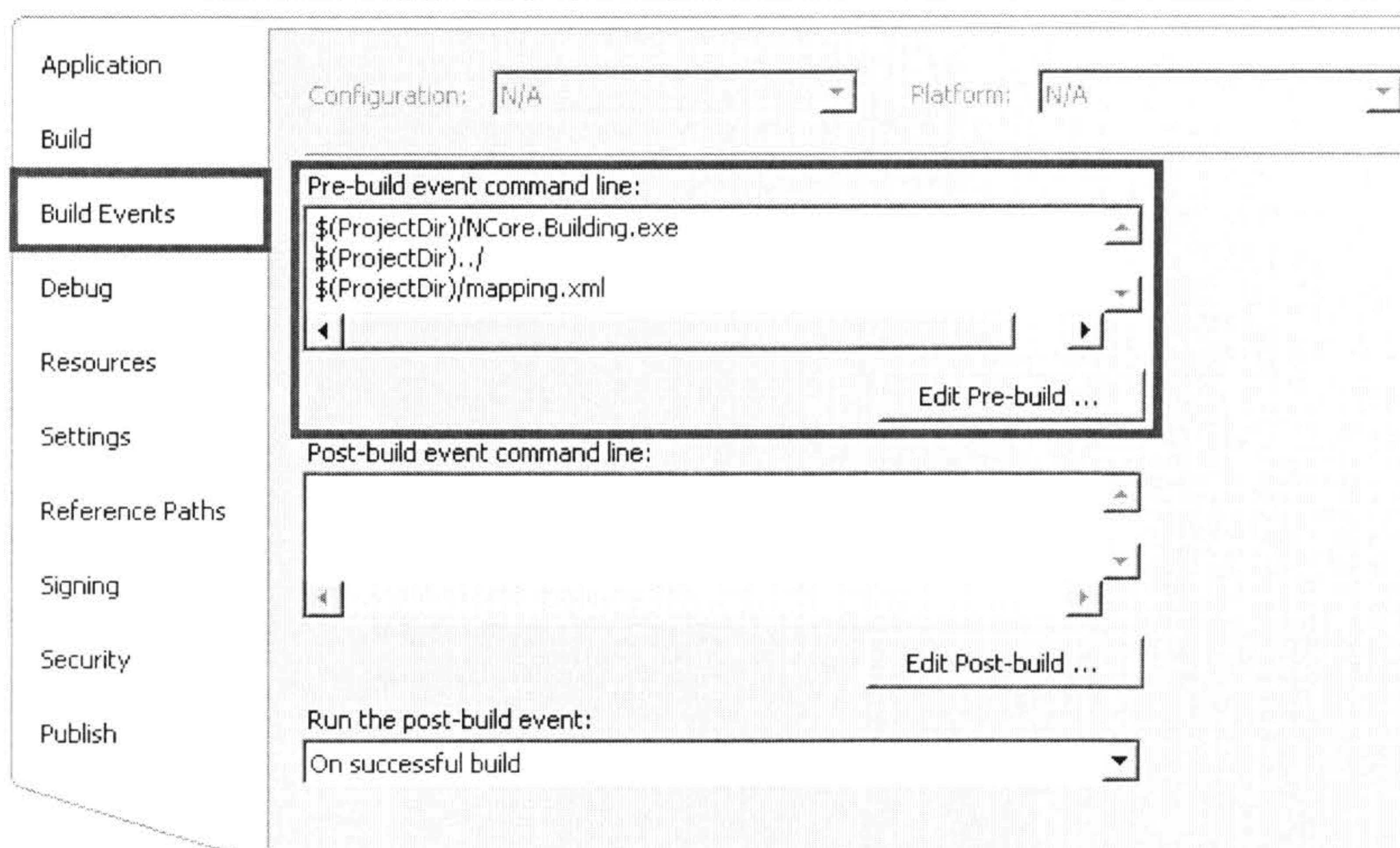
Občas se může stát, že NCore.Building.exe skončí s chybou a build aplikace tedy neproběhne. To znamená, že některý deployment nemá korektní formát nebo unikátní název. Jaká chyba přesně nastala, lze zjistit ze standardního výstupu, který vypisuje VS do panelu Output.

8.1 Konfigurace App.config

NCore se snaží umístit veškerou konfiguraci do deploymentů, ale některé konfigurace do nich umístit nemůže. Proto je nutné vložit do App.config[1] tyto položky:

- AppRoot - umístění adresáře, ve kterém má hledat NCore deploymenty, typicky je to adresář, ve kterém je projekt.

[1] App.config je XML soubor sloužící k základní konfiguraci .NET aplikací.



Obrázek 8.1: Nastavení pre-build event v VS 2005.

- MappingFile - umístění souboru s mapováním.
- SignsFile - umístění souboru se značkami.

8.2 Třída Components

Aplikace NCore.Building.exe má jednu velice užitečnou funkčnost. Vytváří soubor Components.cs, do kterého ukládá názvy všech deployovaných komponent. Soubor Components.cs je vytvářen tak, aby měl formát korektní třídy. Díky tomu, že je NCore.Building.exe pouštěn ještě před samotným buildem VS, stihne vygenerovat Components.cs, který je při následném buildu VS překompilován. Máme tedy vždy k dispozici překompilovanou, automaticky vygenerovanou třídu Components obsahující názvy všech deployovaných komponent. Tato třída velice usnadní vývoj, s její pomocí si už nemusíte pamatovat přesné řetězce názvů komponent.

Kapitola 9

Ukázková aplikace Contact Manager

Contact Manager je jednoduchý projekt ukazující základní funkčnosti frameworku NCore. Aplikace simuluje správce kontaktů. Pod pojmem kontakt si můžete představit jednu osobu, tedy jméno, příjmení, email apod. Ke každému kontaktu je možné přiřadit úkol. Každý úkol obsahuje svůj popis a datum, kdy má být splněn. Aplikace poskytuje typické funkčnosti, které jsou od ní očekávány. Například zobraz seznam kontaktů a úkolů, založ nový kontakt a úkol, vyhledej kontakt atd. Contact Manager je implementován jako tlustý klient v prostředí WinForms. Aplikaci si můžete také prohlédnout bez instalace, protože na přiloženém CD je video zobrazující základní funkčnosti aplikace.

9.1 Instalace

Protože aplikace používá databázi jako úložiště dat, je nutné ji nejprve připravit. Pokud nemáte na svém počítači Microsoft SQL Server 2005 Express, musíte si jej nejprve nainstalovat. Je možné si jej zdarma stáhnout na adrese
http://msdn.microsoft.com/vstudio/express/sql/register/default.aspx. Po nainstalování SQL serveru je potřeba do něj ještě nahrát předpřipravená data. K nahrání dat slouží standardní import v Management Studiu.

- Spusťte Management Studio a připojte se k vaší lokální databázi.

- V pop-up menu nad polem Database v Object Browseru vyberte Restore Database.
- Vyplňte nový název databáze na ncore, vyberte výběr z disku a z přiloženého cd vyberte soubor ncore.bak z adresáře ContactManager-release/DBBak.
- Zaškrtněte checkbox Restore.
- Spusťte vytvoření databáze tlačítek OK.

Nyní máte připravená data a můžete aplikaci spustit. Aplikace se spouští příkazem TestProject.exe.

Dále se již budu zabývat pouze implementací ContactManageru, protože samotný vzhled a funkčnost aplikace nejsou tolik důležité. Po pochopení implementace této aplikace již budete schopní sami používat framework NCore, a usnadníte si tak vývoj vašich databázových aplikací.

9.2 Struktura projektu

Contact Manager je vyvinut v MS Visual studiu 2005, má tedy jeho typickou strukturu a vlastnosti. Celý projekt spustíte pomocí ContactManager.sln. Po jeho spuštění zjistíte, že je rozdělen na složku s implementací NCore, projekt ContactManager a projekt Contact. ContactManager je spouštěcí projekt aplikace, obsahuje nastavené události spouštěné před buildem a hlavní formulář. Projekt Contact představuje modul pro práci s kontakty.

9.2.1 Projekt Contact

Pro větší přehlednost je projekt rozdělen na adresáře.

- API – obsahuje všechny datové objekty.
- Deployments – obsahuje deploymenty potřebné pro běh modulu Contact.
- Implementation – obsahuje implementace BR objektů.
- UI – složka pro objekty uživatelského rozhraní, je rozdělena dále na jednotlivé funkčnosti.

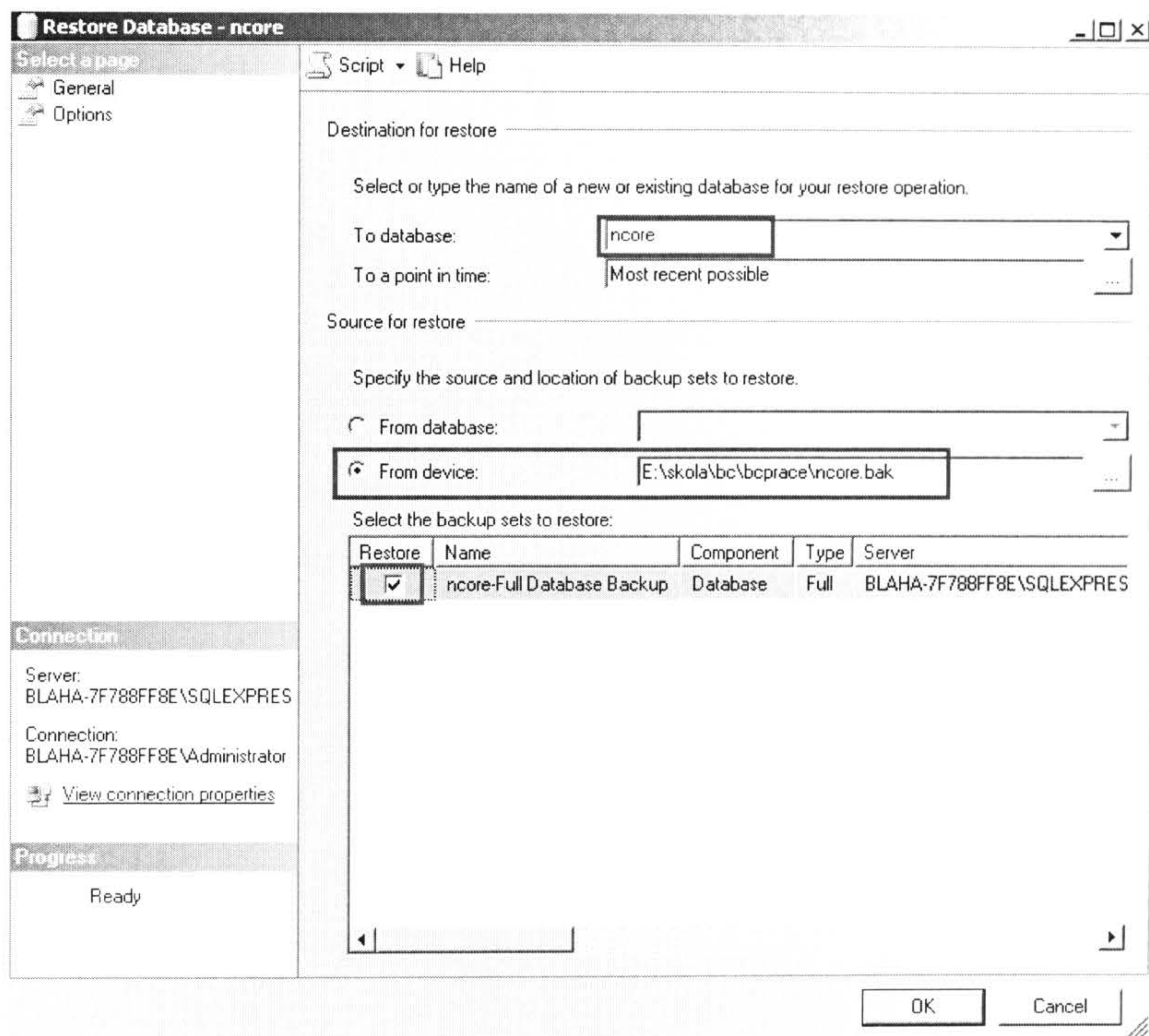
Toto rozdělení je dobré dodržovat u všech dalších projektů. Dokonce je u větších projektů dobré ještě rozdělit modul na více projektů, například Contact-API, Contact-Implementation a Contact-UI. Takovéto dělení aplikace na projekty neslouží jen k přehlednosti, ale také ke zvýšení efektivity překladu aplikace. Visual Studio totiž i při změně jediného souboru překládá minimálně celý projekt. Tedy čím budou projekty menší, tím bude překlad aplikace rychlejší.

9.2.2 Projekt ContactManager

ContactManager je spouštěcí projekt pro aplikaci. Obsahuje tedy metodu Main. Ta je uvnitř třídy Program. Třída Program má poněkud komplikovanější deklaraci, tou se ale nebudu zabývat, protože je v naprosté většině aplikací stejná. Důležité však je, že každá aplikace využívající NCore WinUI musí obsahovat jeden hlavní work item a jeden hlavní formulář. Formulář a work item se pak předávají do spouštěcí třídy pomocí generických typů předka FormShellApplication. Contact Manager má hlavní formulář třídu MainForm a hlavní work item ShellWorkItem.

9.3 Závěr

Nejlépe samozřejmě proniknete do implementace aplikace pomocí jejich zdrojových kódů. Aplikace je opravdu jednoduchá, tak že by to neměl být velký problém. Největší překážka je určitě mapování objektů na tabulky databáze pomocí NHibernate. Nastudování tohoto frameworku zabere poměrně hodně času, o to více času vám to však ušetří do budoucna při implementaci rozsáhlejších aplikací.



Obrázek 9.1: Vytvoření nové databáze pro Contact Manager.

Kapitola 10

Shrnutí

NCore představuje hotové řešení pro vývoj aplikací pracujících s relační databází. Umožňuje zrychlit vývoj aplikací a zároveň udržet v systému standardní styl.

Framework splňuje všechny cíle stanovené v kapitole 1.2.

- Díky nástroji NHibernate umožňuje pracovat s daty z databáze v prostředí jazyka c#.
- Mechanizmus práce s deployovanými komponentami umožňuje framework dále libovolně rozšiřovat.
- NCore poskytuje rozhraní pro robustní logování postavené na nástroji log4net.
- Poskytuje rozhraní pro práci s dočasnou pamětí.
- Díky CAB usnadňuje framework tvorbu uživatelských rozhraní. Usnadňuje např. rozdělení rozhraní na nezávislé části a přepínání mezi obrazovkami.
- Použitelnost frameworku je předvedena na aplikace Contact Manager.

Vývoj aplikace v NCore by měl probíhat tak, že se nejprve nadefinují mapování na všechny objekty, a pak se pracuje s databází výhradně pomocí třídy NHibernateFacade. Programátoři by neměli psát žádné SQL dotazy, pokud to není nutné. Díky tomu, že NCore generuje SQL místo programátorů, je vývoj aplikace rychlejší a méně náchylný na chybovost programátorů.

10.1 Práce více týmů

NCore rozděluje aplikaci na vrstvy, čímž umožňuje současnou práci více vývojových týmů zároveň. Jeden tým může například obstarávat databázovou vrstvu a mapovat tabulky na datové objekty. Další tým může implementovat business logiku v BR objektech. A poslední se může starat o uživatelské rozhraní. Toto schéma může navíc být ještě dále rozděleno, protože aplikace v NCore se skládá typicky z několika samostatných modulů, na kterých mohou týmy odděleně pracovat.

10.2 Rychlost

Při používání NCore by neměly vznikat větší rychlostní problémy, protože framework stihne přemapovat i tisíce objektů z databáze na třídy během sekundy.

Problémy s rychlostí mohou vzniknout jedině při načítání velkých seznamů objektů. Pro načítání seznamů by měly být vždy použity NCore exekutory a datové objekty, které obsahují namapované pouze ty sloupce, které jsou v seznamu skutečně potřeba. Exekutory umožňují zapsání dotazovacího jazyka databáze do externích souborů, jejich vyvolání a následné zpracování. Výsledkem spuštění exekutora je seznam objektů, který je už plně použitelný v jazyce c#.

10.3 Směry dalšího vývoje

Díky mechanizmu práce s deployovanými komponentami je framework maximálně rozšiřitelný. Je možné přidat libovolnou novou komponentu a okamžitě ji začít používat. Nové komponenty mohou být například komponenty pro správu uživatelů, audit log apod.

Další možnosti rozšíření jsou v oblasti webových aplikací a ASP. V těchto aplikacích je NCore použitelný, protože všechna práce s databází, logováním atd., je stejná jak pro webové tak pro desktopové aplikace, ale podpora uživatelských rozhraní v NCore zatím zahrnuje jen ty desktopové.

10.4 Závěr

Tato práce především popisovala databázový framework NCore, který jsem jako součást bakalářské práce vyvinul v jazyce c#. V úvodu práce jsou obecně popsány softwarové frameworky a jejich nezbytnost při vývoji rozsáhlejších systémů. V dalších částech jsou uživateli ukázány základní funkce frameworku. Skutečná použitelnost NCore je pak demonstrována na ukázkové aplikaci Contact Manager.

NCore slouží programátorům zejména ke zjednodušení práce s relační databází. Dále pak obsahuje sadu komponent, které programátorům usnadní práci v oblasti cachování, logování, číselníků a také uživatelského rozhraní.

Literatura

[1] http://logging.apache.org/log4net (29.7.2007)

[2] http://www.nhibernate.org (29.7.2007)

[3] Gamma E., Helm R., Johnson R.,Vlissides J. (2000): *Návrh programů pomocí vzorů.* Grada Publishing, Praha

[4] http://msdn2.microsoft.com/en-us/library/8bxxy49h(vs.80).aspx (29.7.2007)

[5] Johnson R., Foot R. (1988) *Designing Reusable Classes.* Oriented Programming

[6] Cavaness Ch. (2002) *Programming Jakarta Struts.* O'REILLY

[7] http://www.springframework.org/ (29.7.2007)

[8] msdn.microsoft.com/library/en-us/dnpag2/html/cab.asp (29.7.2007)

Dodatek A

Obsah CD

Součásti bakalářské práce je CD, které obsahuje všechny zdrojové kódy i ukázkovou aplikaci ContactManager.

Vlastní obsah CD:

- Adresář **ContactManager-src** obsahuje VS projekt se zdrojovými kódy aplikace ContactManager.
- Adresář **ContactManager-release** obsahuje spustitelnou aplikaci ContactManager. Jak se tato aplikace spouští popisuje kapitola 9.
- Adresář **NCoreStartUp** obsahuje VS projekt, který umožňuje okamžitě začít pracovat ve frameworku NCore. Také obsahuje zdrojové kódy frameworku.
- Adresář **Bakalářská práce** obsahuje elektronickou podobu této práce.
- Adresář **Dokumentace** obsahuje dokumentaci frameworku NCore.