Jihočeská univerzita – Pedagogická fakulta

Katedra informatiky

Bakalářská práce

Databáze Caché a prostředí .NET

Vypracoval: Petr Kohlíček

Vedoucí bakalářské práce: Mgr. Miloš Prokýšek

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 28.4.2011

Petr Kohlíček

Anotace

Caché je databáze post-relačního typu od společnosti InterSystems. K datům je možno přistupovat jak pomocí objektového přístupu, tak prostřednictvím jazyka SQL. Díky objektovému přístupu programátorům odpadá pracné objektově-relační mapování, které je potřeba použít při ukládání objektů z objektově orientovaných programovacích jazyků do relačních databází. Databáze Caché má vlastní vývojové studio pro definování objektových tříd, které se ukládají do databáze, a podporuje export těchto tříd do programovacích jazyků, jako je například Java a .NET (C#). Tato bakalářská práce se zabývá problematikou využití databáze Caché s prostředím .NET a experimentálně ověřuje objektovou kompatibilitu Caché a .NET.

Abstract

Caché is a post-relation database, developed by the InterSystems corporation. Data are available either with the use of an objective access or via the SQL language. Using the objective access, the developers are free of the complicated objective-relation mapping, which is necessary when the objects are being stored from the objective-oriented programming languages into the relation databases. Caché database involves an own visual studio, designed for an easy-to-use object class definition. The supported programming languages, for the export of the created classes, are, for example, Java and .NET (C#). This bachelor thesis deals with the use of the Caché database within the .NET Framework and evaluates the compatibility of Caché and .NET.

Klíčová slova

Caché, .NET, C#, InterSystems, DB

Zadání

Student ve své práci vypracuje databázové schéma v prostředí databázového systému Caché a následně vybuduje GUI pro tuto databázi v prostředí .NET. V teoretické části práce se student zaměří na problematiku importu dat s databáze Caché do prostředí .NET a to především za použití jazyka C#.

Poděkování

Rád bych na tomto místě poděkoval vedoucímu mé bakalářské práce Mgr. Miloši Prokýškovi za odborné vedení, ochotnou pomoc a cenné rady, které mi při vypracování bakalářské práce poskytoval.

Obsah

1 Úvod

V současné době většina aplikací pracujících s daty používá pro jejich ukládání nějaký druh databáze, mezi kterými stále vedou relační databáze. Nejrozšířenější programovací jazyky jsou dnes objektové. Při vývoji aplikace pak vzniká problém, jak ukládat objekty do relační databáze. Pro tento účel vznikla celá řada objektově relačních mapovacích frameworků. Jinou cestou jde post-relační databáze Caché, kde můžeme pracovat přímo s objekty. Zachovává však i možnost relačního přístupu k datům.

Mezi jeden z nejrozšířenějších programovacích jazyků patří C# v .NET Frameworku od firmy Microsoft.

V této bakalářské práci je popsán vývoj aplikace pomocí Caché a prostředí .NET. Práce se z velké části zabývá kompatibilitou Caché a prostředí .NET, která v české literatuře nebyla stále popsána. Dále se zabývá problematikou exportu objektových tříd z Caché do jazyka C# a následně importu dat z databáze Caché do prostředí .NET.

V praktické části byla vyvinuta aplikace, jejíž databázové schéma s definicemi objektových tříd bylo vypracováno v prostředí databázového systému Caché. Následně byly vyexportovány definice objektových tříd do jazyka C# v prostření .NET od společnosti Microsoft, kde bylo naprogramováno i grafické uživatelské prostředí aplikace. Aplikace je určena pro operační systém Windows a demonstruje použití databáze Caché společně s programem v jazyce C#.

2 Teorie

V této kapitole jsou vysvětleny pojmy post-relační databázový systém Caché a .NET Framework, kterými se dále zabývá tato práce.

2.1 Databázové technologie

Relační databázové systémy jsou založeny na datech uložených v relačním modelu. Při vývoji objektově orientovaných aplikací s využitím relačních databází pro ukládání dat je potřeba provádět převod objektů do relačních tabulek a při jejich načítání opět převod z relační tabulky na objekt. Existuje řada objektově relačních mapovacích frameworků, mezi něž patří např. Hibernate.

Objektově-relační databázové systémy jsou relační databázové systémy rozšířené o některé objektové vlastnosti. Jsou to dnes nejrozšířenější databázové systémy.

Objektové databázové systémy nebyly dlouhou dobu příliš rozšířené. Jedním z důvodů byl chybějící standard pro ukládání objektů do těchto databází. Proto vznikla skupina Object Data Management Group (ODMG), která dala vzniknout standardu ODMG, který je dnes ve verzi 3.0.

Post-relační databázový systém Caché

Problémy s objektově-relačním mapováním řeší databáze Caché od společnosti InterSystems. Tato databáze se nazývá databází post-relačního typu. Databáze Caché pracuje s objekty a není tedy třeba řešit převod do relační databáze, což vývojářům zjednodušuje práci, zrychluje a zlevňuje tak vývoj aplikace. Databázový systém Caché se drží standardu ODMG. Kromě objektového přístupu je ale stále možné přistupovat k datům pomocí ODBC. Pro snadný vývoj aplikací pro databázi Caché má Caché rozhraní pro programovací jazyky jako je Java, ActiveX, .NET. Je také možno tvořit webové stránky CSP (Caché Server Pages) podobné stránkám JSP (Java Server Pages) a snadno vytvářet webové služby.

2.2 .NET Framework

V této práci je používán silně typový, objektově orientovaný jazyk C#, který je součástí .NET Frameworku, což je prostředí pro vývoj a běh aplikací od společnosti Microsoft. V .NET Frameworku lze vyvíjet aplikace pomocí několika různých programovacích jazyků. Překlad z těchto jazyků probíhá do CIL (Common Intermediate Language), dříve označovaný MSIL (Microsoft Intermediate Language), což je „mezijazyk“ používaný v .NET Frameworku. CIL není závislý na hardwaru daného počítače a je objektový. Při spuštění programu pak probíhá překlad z CIL pomocí CLR (Common Language Runtime), který tvoří jádro .NET Frameworku, přímo do strojového jazyka daného počítače. Z toho také plyne, že pro program přeložený v .NET Frameworku je tento framework potřeba i pro

jeho spuštění a musí tak být nainstalovaný jak na počítači, který je použit pro vývoj daného programu, tak i na počítači, kde je pak spouštěn.

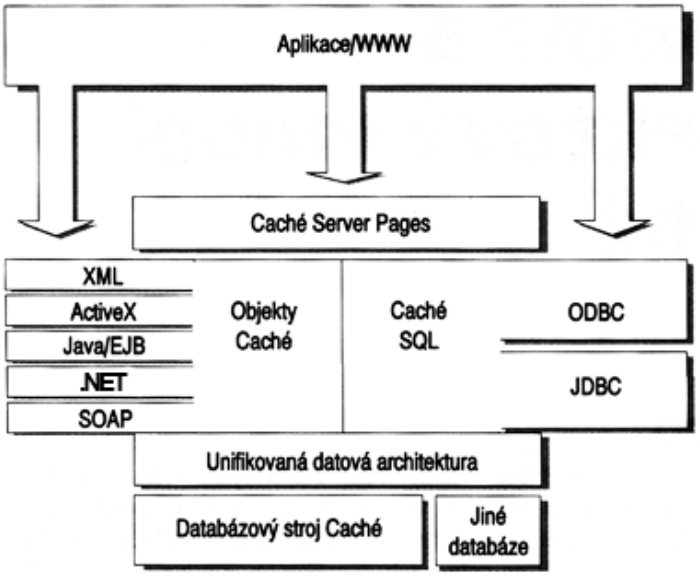
Programovací jazyky pro vývoj .NET aplikací jsou: C#, Visual Basic .NET, F#, J#, IronPython, a další. Pro vývoj .NET aplikací v této práci je používáno vývojové prostředí Visual Studio, které je stejně jako .NET od společnosti Microsoft. Visual Studio pro snadnější vývoj aplikací obsahuje propracovaný editor kódu, který podporuje IntelliSense. Dále mimo jiné obsahuje designer formulářů, sloužící pro vizuální návrh formulářů.

Pro příklady v této práci je potřeba Microsoft .NET Framework verze 2 nebo vyšší. Verze 2 .NET Frameworku je určena pro operační systém Microsoft Windows 98 a vyšší. Pro operační systém Windows Mobile je určen .NET Compact Framework, který je upravený pro běh na mobilních zařízeních. Oproti .NET Frameworku pro Microsoft Windows jsou v .NET Compact Frameworku vynechány některé třídy a u některých tříd je omezena jejich funkčnost.

3 Práce s Caché

Databázový systém Caché umožňuje vytvořit struktury pro ukládání dat několika způsoby. Lze nadefinovat objektové třídy nebo definovat tabulky pomocí SQL.

Unified Data Architecture poskytuje společnou vrstvu jak pro objektový přístup k datům, tak pro přístup prostřednictvím jazyka SQL. Díky tomu mohou starší programy přistupovat k datům relačně a není tak třeba program měnit. Nově naprogramovaná funkcionalita pak může využívat objektový přístup ke stejným datům a databázi Caché a využívat tak vlastnosti objektového programování, aniž by se v kódu vyskytovaly dotazy SQL.



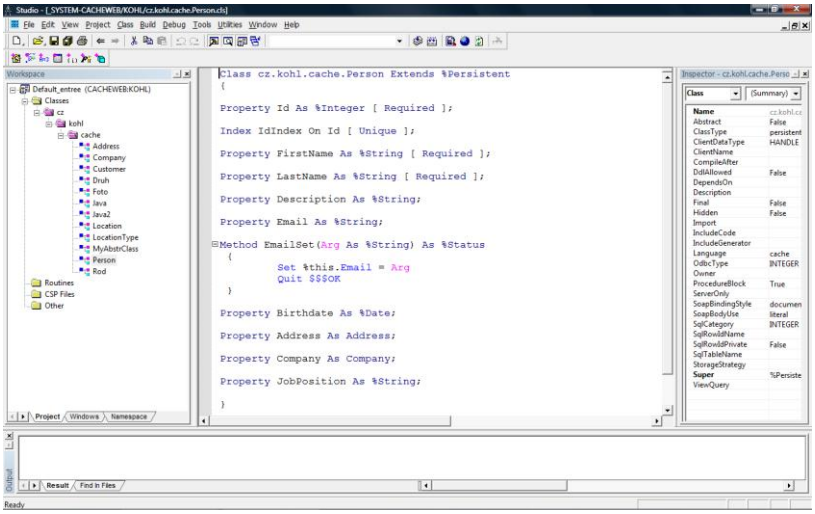
Obrázek 3.1 Systémová architektura Caché
(převzat z [1] str. 40 a rozšířen)

Pro usnadnění vývoje aplikací Caché podporuje export objektových tříd do několika programovacích jazyků, mezi něž patří například: Java, Java Jalapeno, .NET (C#).

Caché má pro správu a vývoj nástroje Studio, Terminal a System Management Portal, které jsou stručně popsané ve zbytku této kapitoly.

Studio

Databázový systém Caché má vlastní vývojové prostředí, které se nazývá Studio. Studio slouží pro definování objektových tříd a jejich překlad.



Obrázek 3.2 Caché Studio

Tvorba tříd v Caché Studio

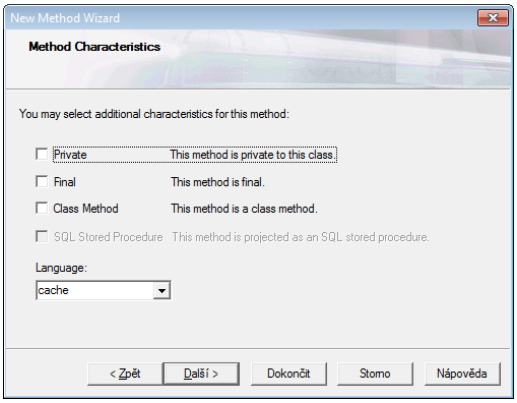
V Caché Studiu se dají vytvářet projekty a v nich třídy, které je možno rozdělovat do namespace. Při vytváření nové třídy je třeba zvolit, jakého typu má třída být. Nabízené možnosti jsou: Persistent, Serial, Registered, Abstract, Datatype, CSP a třída odděděná od jiné třídy. Volba Persistent je pro třídy, které mají být uložené samostatně v databázi a nemají dědit od jiné, dříve vytvořené, třídy. Pro třídy, které mají být uložené v databázi pouze jako součást jiné persistentní třídy, je pak volba Serial. Více informací o typech tříd v kapitole 4.

Po vytvoření třídy je třeba přidat do třídy příslušné vlastnosti. To lze buď přímo zápisem kódu dané vlastnosti v okně s kódem třídy, nebo lze kliknout na ikonku New Property, která spustí průvodce New Property Wizard. Je třeba zadat název dané vlastnosti, její typ a zda má být vlastnost požadovaná, indexovaná, unikátní nebo vypočítaná (více viz. kapitola 5 a 6).

Pro přidání metody je opět na výběr přímo zápis kódu nebo průvodce New Method Wizard, který se spustí po kliknutí na ikonku New Method. U metody je třeba zadat její název, návratový typ metody, parametry metody. Dále je třeba zvolit, zda má být metoda privátní, finální a zda se jedná o metodu třídy. U metody je také na výběr programovací jazyk, kterým má být její obsah naprogramován.

V kódu se metoda definuje klíčovým slovem `Method`, pro metody instance, nebo `ClassMethod`, pro metody třídy. Za klíčovým slovem následuje název metody a kulaté závorky, mezi nimiž je možné definovat seznam argumentů metody. Pokud má metoda

vracet nějaké hodnoty, je uvedena klauzule `As` s návratovým typem. V hranatých závorkách pak mohou být uvedená klíčová slova, jako například `Abstract`, `Final`, `Private`. Nakonec je uvedeno tělo metody uzavřené ve složených závorkách.



Obrázek 3.3 New Method Wizard

Třída může také obsahovat dotazy Caché. Dotazy umožňují definovat SQL dotaz, nebo dotaz pomoci jazyka Caché Object Script. Dotazy mohou mít parametry stejně jako metody. Pro tvorbu dotazu je možno opět použít průvodce New Query Wizard. U dotazu definovaného pomocí SQL se nadefinují případné parametry dotazu, vyberou se vlastnosti třídy, které mají být výstupem, dále se sestaví podmínky a jako poslední je možné nadefinovat třídění výstupu. Pro dotaz definovaný uživatelským kódem, je potřeba vytvořit tři metody třídy. Jsou to: `nameOfQueryExecute`, `nameOfQueryClose`, `nameOfQueryFetch`, kde se místo řetězce nameOfQuery v názvech metod zadá název daného dotazu.

Tvorba webové služby v Caché Studio

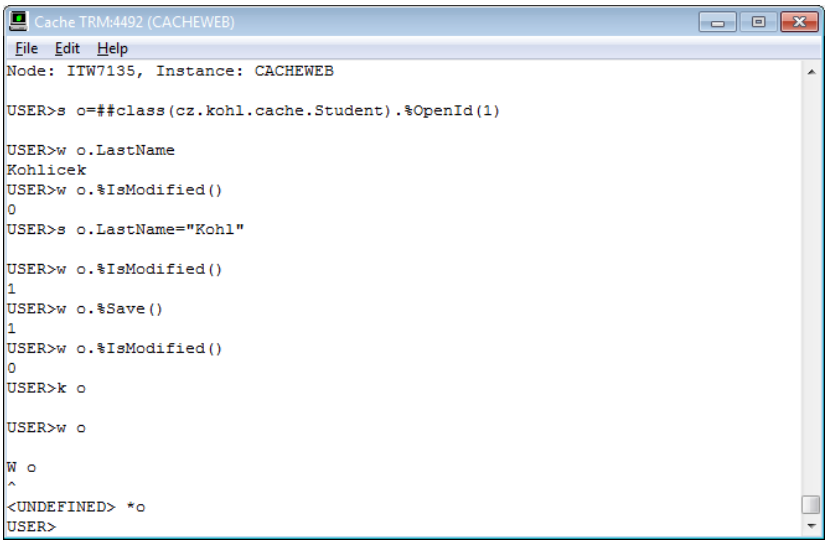
Jednou z možností, jak zpřístupnit data z databáze Caché, jsou webové služby založené na SOAP. Vytvoření webové služby v Caché Studio je velice jednoduché. Po kliknutí na položku New v menu File se otevře okno pro výběr typu nového souboru. V záložce Custom je jedna z voleb Web Service. Po výběru této položky se zobrazí okno Caché Web Service Wizard, ve kterém je třeba vyplnit název balíčku a třídy, která bude obsahovat kód pro danou webovou službu. Poté se zvolí název webové služby a její namespace a názvy metod této služby. Po kliknutí na tlačítko se vytvoří třída odděděná od třídy %SOAP.WebService s předpřipravenými metodami. Zbývá doplnit kód metod, tak aby vracely požadovaný výsledek.

Po zkompilování třídy je webová služba ihned dostupná na url:
http://<server>:<port>/csp/<namespace>/<class>.cls
Význam jednotlivých částí url je popsán v následující tabulce.

<server>	jméno serveru, kde je nainstalován databázový systém Caché
<port>	číslo portu, kde je nainstalován databázový systém Caché
<namespace>	název namespace, kde je třída uložená
<class>	jméno třídy i s balíčkem

Terminál

Terminál slouží pro přihlášení k lokálnímu nebo vzdálenému systému Caché. Na příkazovém řádku je možno zadávat příkazy pomocí Caché Object Scriptu a dotazovat se tak na objekty, nebo s nimi manipulovat.



Obrázek 3.4 Terminál Caché

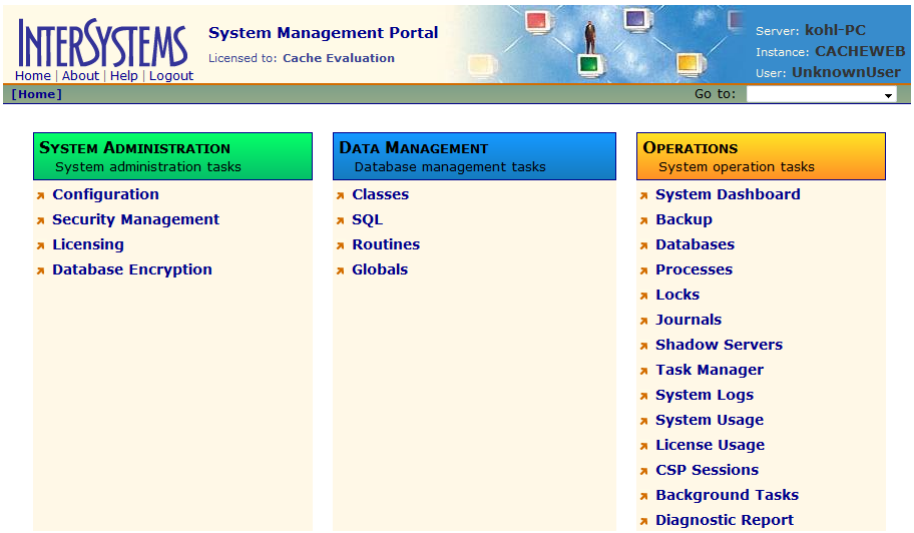
System Management Portal

System Management Portal je webové rozhraní pro správu databázového systemu Caché. Rozděluje se na tři sekce: systémová administrace, správa dat a systémové operace.

V sekci systémové administrace se nacházejí systémová nastavení databázového systému Caché, dají se zde spravovat uživatelské účty a editovat nastavení ohledně zabezpečení Caché.

V sekci pro správu dat lze procházet v databázi uložené třídy, zobrazit jejich nápovědu, popřípadě třídy kompilovat. Dále je možné prohlížet rutiny a procházet globály, popřípadě je editovat. Jak vypadají SQL tabulky pro uložené třídy a obsažená data, lze zjistit ve složce SQL. V podsložce Browse SQL Schemas lze také spouštět dotazy definované v uložených třídách.

V sekci pro systémové operace se nachází položky jako je zálohování databáze, procházení logů, reportů určených pro monitoring či diagnostiku.



Obrázek 3.5 System Management Portal

4 Tvorba programu v .NET Frameworku spolupracujícího s Caché

K databázovému systému Caché je možno přistupovat pomocí Caché Managed Provider, což jsou vygenerované proxy třídy spolu s knihovnou umožňující s těmito proxy třídami pracovat. Proxy třídy představují programový kód, umožňující pracovat s třídami definovanými v databázi Caché v .NETu. Zároveň je zajištěna vazba mezi instancí objektu v .NETu a korespondujícím objektem v databázi.

Druhým možným přístupem je pomocí knihovny ADO.NET společně s knihovnou, obsahující jmenný prostor `InterSystems.Data.CacheClient`, ve kterém se nacházejí třídy, podobné třídám ve jmenném prostoru `Systém.Data.OleDB` od společnosti Microsoft. Tyto třídy umožňují přistupovat k datům uloženým v tabulkách v databázi Caché pomocí knihovny ADO.NET.

Další možností jak přistupovat k datům uloženým v databázi Caché je pomocí SOAP (Simple Object Access Protocol). SOAP je protokol pro webové služby postavený na XML. Tvorba webové služby v databázovém systému Caché byla popsána v kapitole 3 Práce s Caché. WSDL (Web Services Descriptor Language) k webové službě lze zobrazit přidáním parametru WSDL=1 k url webové služby.

4.1 Komponenty pro CMP

Pro použití CMP (Caché Managed Provider) je potřeba Caché 5.1 nebo vyšší a .NET Framework verze 2.0 nebo vyšší. V Caché 5.1 je třeba CMP stáhnout separátně. Potřebné soubory se nacházejí v adresáři <caché install>\dev\dotnet.

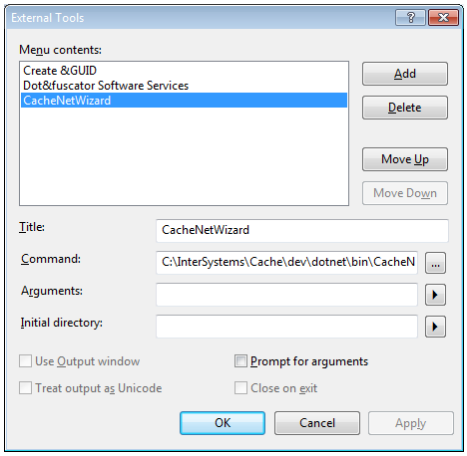
Důležité soubory jsou uvedeny v následující tabulce.

CacheNetWizard.exe	Nástroj Caché Object Binding Wizard for .NET pro vygenerování proxy tříd pro .NET z tříd definovaných v databázovém systému Caché.
InterSystems.Data.CacheClient.dll	DLL knihovna potřebná pro vytvoření spojení s databází Caché, relační přístup k datům a pro práci s proxy třídami.
InterSystems.Data.CacheClientCF.dll	DLL knihovna pro vytvoření spojení s databází Caché, relační přístup k datům a pro práci s proxy třídami v .NET Compact Framework.

Přidání komponent CMP do projektu je popsáno v kapitole 4.3 Vytvoření projektu .NET ve Visual Studiu.

4.2 Přidání nástroje Caché Object Binding Wizard for .NET do Visual Studia

Nástroj Caché Object Binding Wizard for .NET lze spouštět samostatně, je však možné si tento nástroj přidat ve Visual Studiu do menu Tools. Toho lze docílit tak, že ve Visual Studiu vybereme položku External Tools… v menu Tools, což otevře okno External Tools (viz. obrázek 4.1), kde klikneme na tři tečky u položky Command a vybereme soubor CacheNetWizard.exe (popsán v kapitole 4.1) a potvrdíme. Zadáme požadovaný název a potvrdíme tlačítkem OK. To přidá do menu Tools námi vybraný nástroj.



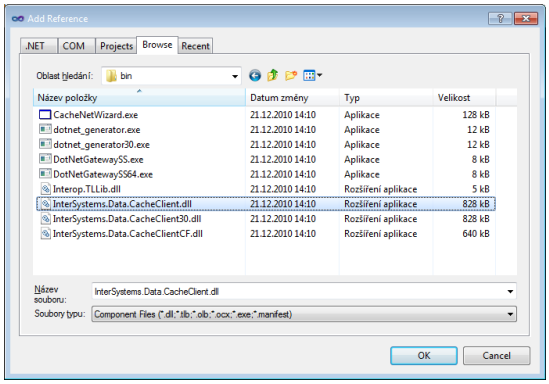
Obrázek 4.1 Přidání Caché .NET Wizardu do Visual Studia

4.3 Vytvoření projektu .NET ve Visual Studiu

Projekt ve Visual Studiu se vytváří klasickým způsobem, ale pak je třeba přidat reference s namespace Caché Managed Provider. To se provede tak, že se v Solution Exploreru u projektu klikne pravým tlačítkem myši na References a vybere se Add Reference… V okně, které se otevře (viz. Obrázek 4.2 Přidání reference na CMP ve Visual Studiu), se vybere záložka Browse a následně soubor `InterSystems.Data.CacheClient.dll` (popsaný v kapitole 4.1). Tím se do projektu přidá reference na jmenné prostory `InterSystems.Data.CacheClient` a `InterSystems.Data.CacheTypes`. Dále je potřeba přidat do zdrojového kódu nezbytné direktivy `using` pro dostupnost CMP namespace, které jsou popsány v následující tabulce.

using InterSystems.Data.CacheClient;	třídy pro připojení ke Caché
using InterSystems.Data.CacheTypes;	specifické typy používané proxy objekty
using InterSystems.Data.CacheClient.ObjBind;	bázové typy pro Caché proxy obj.
using System.Data;	ADO.NET třídy

Tím zajistíme, že v programu budeme mít dostupné všechny jmenné prostory potřebné pro práci s databázovým systémem Caché.



Obrázek 4.2 Přidání reference na CMP ve Visual Studiu

4.4 Vytvoření spojení s Caché

Pro práci jak s proxy třídami, tak pro relační přístup je v programu nejprve potřeba vytvořit spojení s databází Caché. Pro tento účel se v souboru InterSystems.Data.CacheClient.dll, přidaného jako reference, nachází balíček `InterSystems.Data.CacheClient`, který obsahuje třídu `CacheConnection`. Nejprve vytvoříme novou instanci této třídy a přiřadíme jí `ConectionString`.

```
CacheConnection conn = new CacheConnection();
conn.ConnectionString = "Server=localhost; Port=1972; Namespace=USER;
Password=sys; User ID=_system;";
conn.Open();
```

`ConnectionString` obsahuje název serveru a číslo portu, kde běží databázový systém Caché. Dále obsahuje namespace, uživatelské jméno a uživatelské heslo. Pokud nechceme v programu zadávat connection string, je možno zavolat metodu `ConnectDlg`, která otevře okno, ve kterém se vybere Caché server, následně se zadá uživatelské jméno a heslo a nakonec zvolí požadovaný namespace. Po úspěšném spojení s databází Caché vrací metoda connection string se zvolenými parametry.

```
conn.ConnectionString = CacheConnection.ConnectDlg();
```

Poté, co je určen ConnectionString, je třeba zavolat metodu Open, která otevírá spojení s databází Caché. Pokud se nepodaří navázat spojení s databází Caché, je vyvolána vyjímka `InterSystems.Data.CacheClient.CacheException`, na kterou by měl program vhodně reagovat.

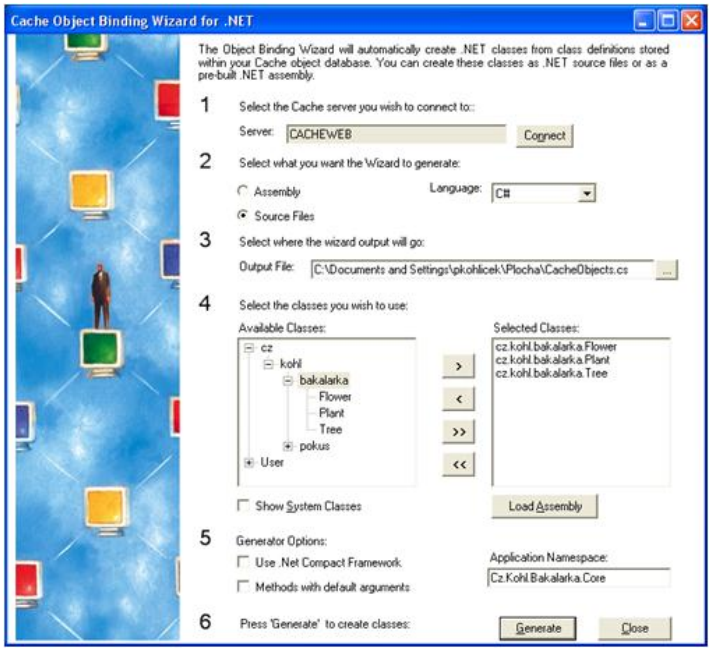
Po ukončení práce s databází je třeba spojení uzavřít metodou Close.

```
conn.Close();
```

4.5 Vygenerování proxy tříd

Pro objektový přístup k databázi Caché z programovacího jazyka C# je třeba vytvořit proxy třídy, pomocí kterých je možno přistupovat k objektům uloženým v databázi Caché. Proxy třídy pro .NET se vytvářejí pomocí Caché Object Binding Wizard for .NET. Tento nástroj je součástí instalace databázového systému Caché. Jeho přidání do Visual Studia bylo popsáno v kapitole 4.2 Přidání nástroje Caché Object Binding Wizard for .NET do Visual Studia.

Po spuštění tohoto nástroje se objeví okno (viz. Obrázek 4.3), kde je nejprve potřeba zvolit připojení k databázi Caché. Je na výběr, zda se má vygenerovat Assembly nebo soubory se zdrojovým kódem, a to buď v jazyce C# nebo ve Visual Basicu. Je třeba též zadat adresář, kam se mají umístit vygenerované soubory. Dále se vyberou třídy z databáze, pro které se mají vygenerovat proxy třídy v .NETu. Lze také zvolit, zda vygenerované třídy budou používány v .NET Compact Frameworku. Pokud je třeba k názvům balíčků, v kterých jsou umístěny Caché třídy, přidat namespace aplikace, zadá se ještě jeho název. Po kliknutí na tlačítko Generate jsou do vybraného adresáře umístěny výsledné soubory, které je pak možno přidat do projektu.



Obrázek 4.3 Caché Object Binding Wizard for .NET

Přidání proxy tříd do projektu ve Visual Studiu

Ve Visual Studiu v okně Solution Explorer je třeba kliknout pravým tlačítkem myši na projekt, do kterého chceme proxy třídy přidat a vybrat položku Add a následně Existing Item. Otevře se okno, ve kterém vybereme soubor s vygenerovanými proxy třídami a potvrdíme tlačítkem Add. Tím se soubor přidá do projektu a je možné začít pracovat s proxy třídami.

4.6 Práce s proxy třídami v C#

Poté, co je vytvořen projekt s přidanou referencí na knihovny umožňující práci s databázovým systémem Caché a jsou přidány příslušné příkazy `using` pro jmenné prostory Caché, je potřeba navázat spojení s databázovým systémem Caché. Když je spojení úspěšné, je možno začít pracovat s proxy třídami Caché.

Třídy, které jsou uchovávány v databázi Caché, jsou potomky třídy `%Persistent`. Z třídy `%Persistent` dědí perzistentní třídy následující metody: `%Open`, `%OpenId`, `%Exists`, `%ExistsId`, `%IsModified`, `%Save`, `%Delete`. Proxy třídy perzistentních objektů obsahují ekvivalentní metody, některé z nich jsou popsány dále.

Vytvoření nové instance

Pro vytvoření nové instance objektu je třeba použít konstruktor s připojením na databázi Caché jako parametrem. Vznikne tak nová instance objektu a zároveň vznikne korespondující objekt v databázi Caché.

```
cz.kohl.cache.Student student = new cz.kohl.cache.Student(connection);
```

Metoda OpenId(CacheConnect, id)

Jedná se o metodu třídy, která vrací instanci dané třídy, nebo její podtřídy, s daným `id`, načtenou z databáze Caché.

```
cz.kohl.cache.Student student = cz.kohl.cache.Student.OpenId(
  connection, "1"
);
```

Metoda ExistsId(CacheConnection, id)

Metoda třídy, která vrací nullable typ `bool?` určující, zda existuje objekt dané třídy se zadaným `id`. Hodnotu null vrací pokud nastane chyba.

```
bool? exists = cz.kohl.cache.Student.ExistsId(connection, "1");
```

Metoda Save()

Metoda `Save` instance perzistentní třídy uloží danou instanci do databáze Caché. Pokud instance byla ukládána poprvé, je vygenerováno nové `id` objektu. Metoda vrací `CacheStatus`, který určuje, zda bylo uložení úspěšné.

```
CacheStatus st = student.Save();
bool saveOk = st.IsOK;
```

Metoda DeleteId(CacheConnect, id)

`DeleteId` je metoda třídy, která smaže objekt s daným `id` z databáze.

```
InterSystems.Data.CacheTypes.CacheStatus status =
    cz.kohl.cache.Student.DeleteId(connection, "1");
bool isOK = status.IsOK;
```

Použití dotazů Caché definovaných ve třídě

Tvorba dotazů Caché jako součásti třídy byla popsána v kapitole 3 Práce s Caché. V programu se dotaz používá jako metoda třídy, která vrací `CacheCommand`. K tomuto příkazu lze přiřadit parametry, a pak s ním dále pracovat, například pomocí CacheDataReader jak je vysvětleno v kapitole 4.7 Přístup k datům přes ADO.NET.

```
CacheCommand cmd = cz.kohl.cache.Student.FromName(connection);
CacheParameter param = new CacheParameter("name", CacheDbType.NVarChar);
param.Value = "P";
cmd.Parameters.Add(param);
```

V příkladu je použit dotaz definovaný ve třídě `Student`. Je mu přiřazen parametr `name` s hodnotou „P“. Při spuštění dotazu jsou pak vrácena jména studentů začínajících na písmeno „P“.

Instance tříd potomků třídy `%SerialObject`, na rozdíl od instancí podtříd třídy `%Persistent`, nemohou být samostatně ukládány do databáze. Mohou však být uloženy v rámci persistentní třídy, jak ukazuje následující příklad.

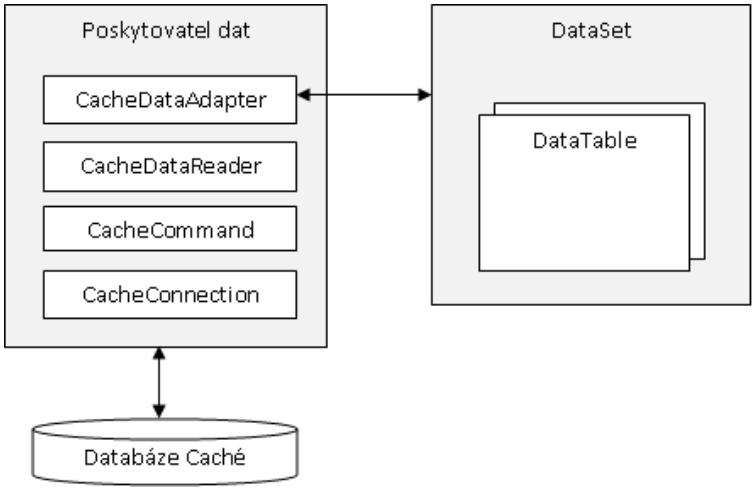
```
cz.kohl.cache.Address address = new cz.kohl.cache.Address(connection);
address.City = "Praha";
address.Street = "Vinohradská";
student.Address = address;
student.Save();
```

V příkladu je vidět vytvoření instance třídy Address, která je podtřídou třídy `%SerialObject`, přiřazení hodnot jejím vlastnostem a následně přiřazení do persistentní třídy `Student`, s kterou je pak uložena do databáze.

V příloze 12.3 je uveden stručný příklad pro práci s proxy třídami v jazyce C#.

4.7 Přístup k datům přes ADO.NET

V příloze 12.4 je uveden zdrojový kód jednoduchého programku přistupujícího k databázi Caché pomocí relačního přístupu. V této kapitole je na tomto příkladu vysvětlen relační přístup k datům v databázi Caché.



Obrázek 4.4 CMP s ADO.NET

Pro přístup k datům uloženým v databázi Caché je v programu nejprve potřeba vytvořit spojení s databázovým serverem, což bylo popsáno v kapitole 4.4 Vytvoření spojení s Caché. Pokud je již vytvořené spojení s databází, je možno provádět dotazy. K tomu slouží třída `CacheCommand`. Pomocí konstruktoru této třídy, kterému je třeba předat řetězec s SQL dotazem a připojení, se vytvoří instance třídy `CacheCommand`.

```
CacheCommand cmd = new CacheCommand(
  "select * from cz_kohl_cache.Student",
  conn
);
```

Na této instanci třídy `CacheCommand` je pak možno zavolat metodu `ExecuteReader`, která provede dotaz a vrátí instanci `CacheDataReader`.

```
CacheDataReader reader = cmd.ExecuteReader();
```

Zavoláním metody `Read` na objektu pro čtení dat je načten jeden řádek výsledku dotazu. Pokud již další řádek neexistuje, je vrácena hodnota `false`. Umístěním tohoto příkazu do cyklu je tak možno načíst všechny řádky výsledku dotazu. Ke sloupcům přistupujeme pomocí indexeru, a to buď pomocí číselného indexu začínajícího od nuly, nebo názvu sloupce.

Následující příklad tak vypíše jméno a město všech studentů ve výsledku dotazu.

```
while (reader.Read())
{
  Console.WriteLine("{0} {1} - {2}",
   reader["LastName"], reader["FirstName"], reader["Address_City"]);
}
```

Přistupuje-li se ke sloupcům pomocí indexeru, je vždy návratový typ `object`. Typově bezpečný přístup zajišťují typované přístupové metody. Ty začínají prefixem `Get` následovaným názvem datového typu. Pokud je tedy prvním sloupcem id objektu, které je typu `Integer`, lze použít k získání id metodu `GetInt32`, jak ukazuje následující příklad.

```
int id = reader.GetInt32(0);
```

Pokud je potřeba v programu pracovat s datovými sadami, je možno je naplnit pomocí třídy `CacheDataAdapter`. Instanci této třídy lze vytvořit podobně jako instanci třídy `CacheCommand`. Konstruktoru se předá řetězec s SQL dotazem a instance připojení.

```
CacheDataAdapter adapter = new CacheDataAdapter(
  "select * from cz_kohl_cache.Student",
   Connection
);
```

Po vytvoření datového adaptéru je již možné vytvořit datovou sadu a nechat ji naplnit pomocí instance adaptéru.

```
DataSet dataSet = new DataSet();
adapter.Fill(dataSet);
```

Datová sada tak bude obsahovat tabulku, v tomto případě se všemi studenty načtenými z databáze Caché.

```
DataTable table = dataSet.Tables[0];
```

Výpis tabulky se provede vypsáním všech řádků a sloupců v tabulce, jak ukazuje následující část příkladu.

```
foreach (DataRow row in table.Rows)
{
    foreach (DataColumn col in table.Columns)
    {
        Console.Write((""+row[col]).PadLeft(30, ' '));
    }
    Console.WriteLine();
}
```

Datová sada je realizována pomocí třídy DataSet z knihovny ADO.NET, která je součástí .NET Frameworku a má mnoho použití. Asi nejčastějším je zobrazení v DataGridView, což demonstruje následující příklad.

```
DataGridView dataGrid = new DataGridView();
dataGrid.Parent = this;
dataGrid.Dock = DockStyle.Fill;
```

```
dataGrid.DataSource = table;
```

V příkladu je vytvořena komponenta DataGridView a je umístěna na formulář, v kterém je daný kód uvedený. Nakonec je datovému zdroji této komponenty přiřazena tabulka z dříve vytvořené datové sady. To zajistí, že se na formuláři zobrazí tabulka s daty.

Po skončení načítání dat z databáze je třeba uzavřít objekt pro čtení dat pomocí metody `Close` a vytvořené spojení.

```
…
}
finally
{
  reader.Close();
  conn.Close();
}
```

5 Základní prvky objektového programování

V této kapitole jsou popsány základní prvky objektového programování. U každého prvku objektového programování je uvedeno, jak se definuje v databázovém systému Caché a jak je převedena pomocí CMP do jazyka C#. Následně je zhodnoceno, zda je tento prvek objektového programování kompatibilní mezi Caché a jazykem C# v .NET Frameworku.

5.1 Třída

Třída je základním stavebním kamenem v objektově orientovaných programovacích jazycích. Může obsahovat vlastnosti a metody. Třída je vzor pro vytvoření instance třídy. Instancí jedné třídy může být několik, a každá má svůj stav, který je dán vlastnostmi třídy.

Caché:

```
Class cz.kohl.cache.MyClass
{
}
```

C#:

```
namespace cz.kohl.cache {
 public partial class MyClass
      :InterSystems.Data.CacheTypes.CacheSerialObject {

   /// <summary>Server class name</summary>
   public const string ServerClassName = "cz.kohl.cache.MyClass";

   …

   /// <summary>Class server name</summary>
   public override string ClassServerName() {
     return cz.kohl.cache.MyClass.ServerClassName;
   }
 }
}
```

Výpis kódu v C# byl zkrácen. U definice třídy nedochází k žádné nekompatibilitě mezi Caché a .NETem.

Finální třída

Od finální třídy není možno dědit jiné podtřídy. V caché se definuje finální třída pomocí klíčového slova Final.

Caché:

```
Class cz.kohl.oop.FinalClass Extends %Persistent [ Final ]
{
  …
}
```

C#:

```
public partial class FinalClass :
InterSystems.Data.CacheTypes.CachePersistent {

    ...
}
```

Do jazyka C# je finální třída převedena jako normální třída, od které lze vytvářet podtřídy.

Abstraktní třída

Z abstraktní třídy nelze vytvářet instance, ale slouží jako šablona pro podtřídy. V Caché lze napsat abstraktní třídu, která se při převodu do .NETu převede jako třída odděděná od CacheSerialObject a její vlastnosti a metody jsou virtual.

Caché:

```
Class cz.kohl.cache.MyAbstrClass [ Abstract ]
{
Property Text As %String;
Method GetText() As %String
{
      Quit Text
}
}
```

C#:

```
public class MyAbstrClass : InterSystems.Data.CacheTypes.CacheSerialObject {

    public new const string ServerClassName = "cz.kohl.cache.MyAbstrClass";

    public virtual string Text {
        get { ... }
        set { ... }
    }

    public virtual string GetText() {
        ...
    }
}
```

5.2 Vlastnosti třídy

Vlastnosti uchovávají data uvnitř instance a definují tak její stav. Každá vlastnost má definovaný svůj typ.

Caché:

```
Property Name As %String;
```

C#:

```
public virtual string Name {
    get { … }
    set { … }
}
```

Vlastnost se z Caché převede do C# jako vlastnost s bloky get a set, které obsahují kód pro získání vlastnosti objektu z databáze Caché. Bloky get a set byly ve výpisu zkráceny a jejich celý kód je uveden v příloze 12.1 a 12.2.

Finální vlastnost

Pomocí klíčového slova Final u definice vlastnosti lze v Caché definovat finální vlastnost. Tu pak nelze překrýt v podtřídách dané třídy.

Caché:

```
Property FinalProperty As %String [ Final ];
```

C#:

```
public virtual string FinalProperty {
    get { ... }
    set { ... }
}
```

Finální vlastnost v Caché se do jazyka C# převede opět jako vlastnost virtual. Proto v podtřídách lze tuto vlastnost překrýt.

Vlastnost určená jen pro čtení

V Caché lze definovat vlastnost určenou jen pro čtení přidáním klíčového slova ReadOnly za definici vlastnosti.

Caché:

```
Property ReadOnlyProperty As %String [ ReadOnly ];
```

C#:

```
public virtual string ReadOnlyProperty {
    get { ... }
}
```

Pokud v Caché nadefinujeme vlastnost ReadOnly, je v převedeném kódu jen blok get, což zajistí, že danou vlastnost jde jen číst.

Vypočítané vlastnosti

V Caché je možno definovat takzvané vypočítané vlastnosti (Calculated Property). Hodnota této vlastnosti je dána příslušnou metodou Get. Hodnota této vlastnosti není uložena v databázi, ale je vypočítána. To může být například použito pro vlastnost `Age` ve třídě `Person`, kde je věk vypočítán z datumu narození.

Caché:

```
Property Birthdate As %Date;
Property Age As %Integer [ Calculated ];
Method AgeGet() As %Integer [ ServerOnly = 1 ]
{
	set vek=+$h-..Birthdate\365
	Quit vek
}
```

C#:

```
public virtual int Birthdate {
    get { ... }
}
```

Do jazyka C# je vygenerována vlastnost s blokem get, která volá příslušnou metodu v databázi a vrací její výsledek.

5.3 Metody

Metoda obsahuje kód a definuje chování objektu. Metoda může mít definované parametry. Každý parametr má svůj typ. U metody lze také definovat návratový typ. Caché neumožňuje přetížení metod.

Metoda instance

Metody instance lze volat jen v kontextu instance, která metodu obsahuje. V caché se metody instance definují klíčovým slovem `Method`.

Caché:

Definice metody s jedním parametrem typu `%String`. Metoda tento string také vrací.

```
Method MyMethod(Arg1 As %String) As %String
{
	quit Arg1
}
```

C#:

```
public virtual string MyMethod(string Arg1) {
    ...
}
```

Do jazyka C# je převedena metoda s klíčovým slovem `virtual`. Má parametr typu `string` a vrací očekávaný řetězec.

Metoda třídy (statická metoda)

Metoda třídy se, na rozdíl od metody instance, volá pomocí třídy.

Caché:

Metoda třídy se v Caché definuje pomocí klíčového slova `ClassMethod`.

```
ClassMethod ClassMethod() As %String
{
	quit "classMethod"
}
```

C#:

```
public static string
ClassMethod(InterSystems.Data.CacheClient.CacheConnection conn) {
    ...
}
```

Do jazyka C# se metoda třídy převede s klíčovým slovem static. Doplní se však ještě parametr, neboť proxy třída nemá na rozdíl od instance této třídy definované spojení s databází Caché. Proto je nutné jako parametr předat připojení k databázi, aby mohla být volána související metoda v databázi.

Finální metody

Pokud je v Caché definována metoda jako finální, není možné tuto metodu překrýt v podtřídách.

Caché:

```
Method FinalMethod() As %String [ Final ]
{
	quit "final"
}
```

C#:

```
public virtual string FinalMethod() {
    ...
}
```

Finální metoda v Caché se do jazyka C# převede jako metoda s modifikátorem `virtual`. V podtřídách lze proto tuto metodu libovolně překrýt.

5.4 Polymorfismus

Pomocí polymorfismu lze v Caché pracovat se třídami odvozenými od stejné bázové třídy, pomocí této bázové třídy. Parametrický polymorfismus v Caché není podporován.

5.5 Kompozice

Kompozice vyjadřuje možnost skládání objektů z objektů jiných. Dále je kompozice v Caché vysvětlena na třídě Person, jejíž vlastnost `Address` obsahuje instanci třídy `Address`.

Caché:

```
// Definice třídy Address
Class cz.kohl.cache.Address Extends %SerialObject
{
Property Street As %String;
Property Number As %Integer;
Property City As %String;
}
```

Nadefinování vlastnosti `JobAddress` ve třídě `Person`, která může obsahovat instanci třídy `Address`.

```
Property JobAddress As %Address;
```

C#:

Vyexportovaná třída `Address`.

```
public partial class Address :
InterSystems.Data.CacheTypes.CacheSerialObject
{ ... }
```

Ve třídě `Person` je definovaná vlastnost `JobAddress`.

```
public virtual cz.kohl.cache.Address JobAddress {
    get { ... }
    set { ... }
}
```

Je-li třeba z instance třídy `Person` zjistit, v jakém městě daný člověk pracuje, přistupujeme k vlastnosti `City` ve vlastnosti `JobAddress`.

```
string city = person.JobAddress.City;
```

5.6 Zapouzdření

Díky zapouzdření jsou data určitého objektu skrytá mimo tento objekt. Pomocí modifikátorů přístupu je možno nastavit, jaké vlastnosti a metody mají být viditelné mimo daný objekt a jaké ne. Tak se definuje rozhraní tohoto objektu, pomocí kterého se s objektem navenek pracuje a objekt nelze jinak měnit. Brání se tak nekonzistenci dat.

Caché umožňuje k vlastnostem a metodám uvést klíčové slovo `Private`. Privátní vlastnosti jsou pak v Caché přístupné jen z metod objektu a jeho podtříd. Pokud není uvedeno klíčové slovo `Private`, je vlastnost veřejná a lze k ní přistupovat i mimo objekt.

Privátní vlastnost

```
Property PrivateProperty As %String [ Private ];
```

Pokud je v Caché nadefinována privátní vlastnost, do kódu v C# není převedena. Proměnná lze používat jen v metodách dané třídy nebo podtřídy v Caché. Pokud je v C# nadefinována podtřída dané proxy třídy, vlastnost dostupná není.

5.7 Dědičnost

Pomocí dědičnosti lze vytvářet nové více specializované podtřídy ze stávající bázové třídy. Podtřída dědí od bázové třídy její vlastnosti a metody. K těmto vlastnostem a metodám lze přidávat nové vlastnosti a metody. Je také možné vlastnosti nebo metody bázové třídy v podtřídě přepsat.

Jednoduchá dědičnost

Lze dědit jen z jedné třídy. Je podporována jak databází Caché, tak i v jazyce C# v .NETu.

Vícenásobná dědičnost

Caché podporuje vícenásobnou dědičnost, což znamená, že třída může dědit od více tříd současně. V prostředí .NET není vícenásobná dědičnost podporována, třída může dědit jen od jedné třídy a implementovat libovolné množství rozhraní.

Na následujícím příkladu je ukázáno, jak Caché převádí třídy s vícenásobnou dědičností do .NETu. V Caché jsou nadefinovány třídy Student, Employee a třída StudentEmployee, která dědí od předchozích dvou tříd.

Caché:

Třída `Student` a `Employee` slouží jako bázové třídy pro třídu `EmployeeStudent`.

```
Class cz.kohl.cache.Employee Extends
 %Persistent
{
Property FirstName As %String;
Property LastName As %String;
Property EmployeeNumber As %String;
Property Salary As %Integer;
Property Address As Address;
Method GetInfo() As %String
{
	quit "Employee"
}
}
```

```
Class cz.kohl.cache.Student Extends
 %Persistent
{
Property FirstName As %String;
Property LastName As %String;
Property StudentNumber As %String;
Property Schoolfee As %Integer;
Property Address As Address;
Method GetInfo() As %String
{
	quit "Student"
}
}
```

Třída `EmployeeStudent` odděděná od tříd `Student` a `Employee`.

```
Class cz.kohl.cache.EmployeeStudent
 Extends (cz.kohl.cache.Student, cz.kohl.cache.Employee)
{
Property Earnings As %Integer [ Calculated ];
Method EarningsGet() As %Integer
{
	set e=..Salary - ..Schoolfee
	quit e
}
}
```

Vlastnosti `FirstName` a `LastName` jsou definované jak v třídě `Student`, tak ve třídě `Employee`. Stejně je tomu tak u metody `GetInfo()`. Třída `EmployeeStudent`, která využívá vícenásobné dědičnosti, v Caché nejprve zdědí vlastnosti a metody z první bázové třídy, což je v tomto případě třída `Student`. Následně dědí vlastnosti a metody z druhé bázové třídy, což je v tomto případě `Employee` a vlastnosti a metody se stejným jménem jsou přepsány. Z toho plyne, že pokud je volána metoda `GetInfo()` třídy `EmployeeStudent`, bude vrácen řetězec „Employee“.

Převedená třída `EmployeeStudent` pak v jazyce C# vypadá následovně:

```
public partial class EmployeeStudent : Cz.Kohl.cache.Student
    {
        /// <summary>
        /// Projection of property Earnings
        /// </summary>
        public virtual System.Nullable<long> Earnings
        {
            get { ... }
        }

        /// <summary>
        /// Projection of property EmployeeNumber
        /// </summary>
        public virtual string EmployeeNumber
        {
```

```
            get { ... }
            set{ ... }
        }

        /// <summary>
        /// Projection of property Salary
        /// </summary>
        public virtual System.Nullable<long> Salary
        {
            get { ... }
            set { ... }
        }
    }
```

Z výpisu kódu byly vypuštěny metody, které nesouvisejí s uváděnou vícenásobnou dědičností a getry a setry byli opět zkrácené (obsah getru a setru je uveden v příloze, liší se jen název vlastnosti a její typ).

Jak je vidět z příkladu, třída s vícenásobnou dědičností je převedena do .NETu tak, že je odvozena od první bázové třídy a vlastnosti a metody z ostatních bázových tříd jsou do ní doplněny. Třída tak má všechny vlastnosti a metody svých předků, ale nastává tu problém u polymorfizmu. Pokud s touto třídou chceme pracovat pomocí referenční proměnné deklarované s typem původní bázové třídy, která byla při převodu do .NETu vypuštěna.

5.8 Rozhraní

V databázovém systému Caché není rozhraní jako takové, ale pracuje se s abstraktní třídou.

5.9 Balíčky

Třídy mohou být umístěny v balíčcích, což umožňuje seskupit související třídy do jednoho jmenného prostoru. V Caché existují dva předdefinované balíčky: %Library a User. V balíčku %Library jsou všechny třídy začínající na znak %. Třídy které nezačínají na % a u kterých není žádný balíček definován jsou součástí balíčku User. Balíček v Caché definujeme uvedením jeho názvu před název třídy a je oddělen znakem tečky.

Caché:

```
Class cz.kohl.cache.MyClass { ... }
```

C#:

```
namespace cz.kohl.cache {
    public partial class MyClass{ ... }
}
```

6 Speciální datové typy Caché

6.1 Relationship

Datový typ Relationship představuje v Caché obousměrnou vazbu mezi objekty. Jsou podporovány vazby 1 : 1, N : M a vazba rodič, potomek.

Pokud je vazba vytvořena v jednom z objektů, je vytvořena automaticky i ve vázaném objektu.

Caché:

Ve třídě `Rod` je definována vlastnost `Druhy`, která obsahuje přiřazené druhy k tomuto rodu.

```
Relationship Druhy As cz.kohl.pokus.Druh [ Cardinality = children,
Inverse = Rod ];
```

Ve třídě `Druh` je nadefinovaná vlastnost `Rod`, která tvoří společně s vlastností `Druhy` ve třídě `Rod` vazbu.

```
Relationship Rod As cz.kohl.pokus.Rod [ Cardinality = parent, Inverse =
Druhy ];
```

C#:

Do kódu v jazyce C# pro třídu `Rod` je vygenerována vlastnost typu `CacheRelationshipObject`, která umožňuje přistupovat k uloženým rodům.

```
public virtual InterSystems.Data.CacheTypes.CacheRelationshipObject Druhy
{
    get { ... }
    set { ... }
}
```

Ve třídě Druh je pak vygenerována vlastnost Rod, která odkazuje na rodičovský rod.

```
public virtual cz.kohl.pokus.Rod Rod {
    get { ... }
    set { ... }
}
```

Pokud je potřeba projít všechny druhy přiřazené k danému rodu, je možné použít cyklus `foreach`, neboť třída `CacheRelationshipObject` implementuje rozhraní `ICollection` a `IEnumerable`.

```
cz.kohl.pokus.Rod rod = cz.kohl.pokus.Rod.OpenId(connection, "1");
foreach (cz.kohl.pokus.Druh druh in rod.Druhy)
{
    Console.WriteLine(druh.Name);
}
```

7 Závěr

V této bakalářské práci byla popsána tvorba objektových tříd pomocí nástroje Studio v databázovém systému Caché od společnosti InterSystems. Dále byly popsány způsoby exportu objektových tříd do jazyka C#, který je součástí prostředí .NET a práce s těmito třídami. Detailně byla experimentálně prověřena a popsána problematika objektové kompatibility Caché a prostředí .NET. V práci jsou uvedeny základní prvky objektového programování a popsány jak v databázovém systému Caché, tak i v jazyku C# v prostředí .NET.

V praktické části byla vyvinuta aplikace demonstrující použití databáze Caché společně s programem v jazyce C#. Pro tuto aplikaci byla navrhnuta definice objektových tříd, které byly implementovány pomocí databázového systému Caché. Pro tyto třídy byly vygenerovány proxy třídy, které byli použity v projektu v .NETu. V tomto projektu bylo pomocí jazyka C# naprogramováno grafické uživatelské rozhraní aplikace.

8 Použité zkratky

.NET	Prostředí od společnosti Microsoft, které obsahuje programovací jazyky jako například C# Visual Basic .NET, F#, J#, IronPython.
ADO.NET	Knihovna ADO.NET je součástí .NET Frameworku. ADO.NET je sada tříd umožňující programátorům připojení k databázi a práci s ní.
CMP	Caché Managed Provider
UDA	Caché Unified Data Architecture

9 Seznam literatury

[1] Kirsten, W., Ihringer, M., Kühn, M., Röhrig, B.: Caché – Databáze postrelačního typu a tvorba aplikací. CP Books, a.s. 2005. ISBN 80-251-0491-5.

[2] InterSystems Caché - World's fastest database [online]. c1996-2010 [cit. 2010-01-20]. Dostupný z WWW: <http://www.intersystems.cz/cache/>.

[3] InterSystems Caché - dotnet Managed Provider [online]. c1996-2010 [cit. 2010-01-20]. Dostupný z WWW: <http://www.intersystems.com/cache/dotnet/>.

[4] Microsoft Corporation [online]. c2010 [cit. 2010-01-20]. Dostupný z WWW: <http://www.microsoft.com/>.

[5] Microsoft Corporation. *Overview of the .NET Framework* [online]. 2011 [cit. 2011-04-27]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/a4t23ktk.aspx>.

[6] Agarwal, V., Huddleston, J.: Databáze v C# 2008 – Průvodce programátora. Computer Press, a.s. 2009. ISBN 978-80-251-2309-6.

[7] Petzold, Ch.: Programování Microsoft Windows Forms v jazyce C#. Computer Press, a.s. 2006. ISBN 80-251-1058-3

10 Seznam obrázků

11 Rejstřík

12 Přílohy

12.1 Úplný kód Getru

Kompletní kód pro blok get používaný pro všechny vlastnosti. Liší se vždy jen jméno vlastnosti a její typ.

```
get {
    try {
       System.Threading.Monitor.Enter(this.conn);
       this.AssertIsConnected();
       this.conn.GeneratedAssembly =
System.Reflection.Assembly.GetExecutingAssembly();
       InterSystems.Data.CacheTypes.CacheMethodSignature mtdSignature =
         this.conn.GetMtdSignature();
            try {
                mtdSignature.SetReturnType(this.conn, 0);
                this.GetPropertyS("NázevVlastnosti", 3, 0, mtdSignature);
                return
       ((Třída vlastnosti)
       (((InterSystems.Data.CacheTypes.CacheObjReturnValue)
       (mtdSignature.ReturnValue)).Value));
            }
            finally {
                mtdSignature.Clear();
            }
        }
        finally {
            System.Threading.Monitor.Exit(this.conn);
        }
    }
```

12.2 Úplný kód Setru

Kompletní kód pro blok set používaný pro všechny vlastnosti obsahující blok set. Liší se vždy jen jméno vlastnosti a její typ.

```
set {
        try {
            System.Threading.Monitor.Enter(this.conn);
            this.AssertIsConnected();
            this.conn.GeneratedAssembly =
       System.Reflection.Assembly.GetExecutingAssembly();
            InterSystems.Data.CacheTypes.CacheMethodSignature mtdSignature =
       this.conn.GetMtdSignature();
            try {
                mtdSignature.Add(value, this.conn, false);
                this.SetPropertyS("NázevVlastnosti", mtdSignature, 3, 0, 3);
            }
            finally {
                mtdSignature.Clear();
            }
        }
        finally {
            System.Threading.Monitor.Exit(this.conn);
        }
    }
```

12.3 Příklad použití proxy tříd pro načítání objektů z databáze Caché

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Text;
using InterSystems.Data.CacheClient;
using System.Net;

namespace CacheSample
{
    class Program
    {
        static void Main(string[] args)
        {
CacheConnection connection = null;
CacheDataReader reader = null;
try
{
    connection = new CacheConnection();
    connection.ConnectionString = CacheConnection.ConnectDlg();
    connection.Open();

    //vytvoření nového studenta
    cz.kohl.cache.Student newStudent;
    newStudent = new cz.kohl.cache.Student(connection);
    //přiřazení hodnot vlastnostem objektu
    newStudent.FirstName = "Petr";
    newStudent.LastName = "Kohl";
    newStudent.Address.City = "Rudolfov";
    newStudent.Address.Street = "Ne Točně";
    //uložení do databáze Caché
    CacheStatus st = newStudent.Save();
    if (st.IsOK)
        Console.WriteLine("Uložení porběhlo v pořádku");
    //uzavření
    newStudent.Close();
    newStudent.Dispose();

    //načtení studenta s id 1
    cz.kohl.cache.Student student;
    student = cz.kohl.cache.Student.OpenId(connection, "1");
    Console.WriteLine(student.FirstName + " " + student.LastName);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
finally
{
    connection.Close();
}
        }
    }
}
```

12.4 ADO.NET příklad

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Text;
using InterSystems.Data.CacheClient;
using cz.kohl.cache;

namespace CachePokus
{
    public class CachePokus
    {
        CacheConnection conn = new CacheConnection();
        CacheDataReader reader = null;
            try
            {
                conn.ConnectionString = "SERVER = localhost; PORT = 1972;
NAMESPACE = KOHL; USER ID = _system; PASSWORD = SYS";
                //conn.ConnectionString = CacheConnection.ConnectDlg();
                conn.Open();

                CacheCommand cmd = new CacheCommand("select * from
cz_kohl_cache.Student", conn);
                reader = cmd.ExecuteReader();

                while (reader.Read())
                {
                    Console.WriteLine("{0} {1} - {2}", reader["LastName"],
reader["FirstName"], reader["Address_City"]);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
                MessageBox.Show("Nepodařilo se vytvořit spojení.");
                this.Close();
            }
            finally
            {
                reader.Close();
                conn.Close();
            }
    }
}
```

13 Obsah DVD

- bakalářská práce ve formátu PDF
- zdrojové kódy programu