VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta elektrotechniky a komunikačních technologií
Ústav biomedicínského inženýrství

DATABÁZOVÝ SYSTÉM InterSystems CACHÉ

Zhodnocení z hlediska jeho vlastností jako objektově orientovaného systému při praktickém vývoji ambulantního modulu nemocničního informačního systému.

Martin Plchút

květen 2004

Obsah

Úvod

V úvodních fázích návrhu nové aplikace musí vývojáři rozhodnout o přístupu k modelování dat. Většinou se rozhodování redukuje na volbu mezi tradičním modelováním dat pomocí relačních tabulek nebo novějším modelováním pomocí objektů. Práce se složitými daty vedla mnoho vývojářů k přesvědčení, že modelování pomocí objektů je efektivnější.

V dnešní době existují produkty, které umožňují relační (tedy SQL) i objektový přístup k datům, protože každý z nich má své opodstatnění. Kdy použít který z přístupů, a proč současní vývojáři dávají všeobecně přednost modelování dat pomocí objektů, pomůže objasnit to, jak a kdy tyto přístupy vznikly.

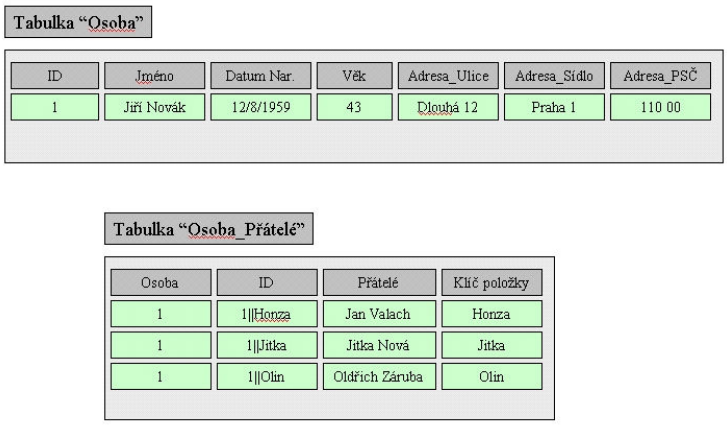
Relační technologie

V průkopnických dobách se zpracovávaly informace na sálových počítačích (mainframe) a přístup k datům byl omezen ve většině případů na odborníky v oblasti informačních technologií.

Databáze byly vytvářeny na koleně a získávání nebo aktualizace dat vyžadovala dokonalou znalost příslušné databáze a jejího uspořádání. Pokud uživatel potřeboval nějakou sestavu, musel zadat její vytvoření přetíženým programátorům. Požadovaná sestava pak obvykle stejně nebyla hotova v čas, aby mohla mít vliv na přijímaná rozhodnutí.

S příchodem osobních počítačů vstoupil svět do doby počítačového zpracování „zaměřeného na uživatele“. Na základě potřeby uživatelů, požadující přístup k centrálně udržovaným podnikovým datům přímo ze svých počítačů, vznikla relační databázová technologie. Uživatelé chtěli vytvářet vlastní sestavy a ad-hoc dotazy.

Relační technologie s sebou přinesly dotazovací jazyk SQL, který umožňuje formulovat dotazy na různá data pomocí konzistentního jazyka. SQL umožňuje nahlížet na data ve velmi jednoduchém standardizovaném formátu tvořeném dvourozměrnými tabulkami s řádky a sloupci. Stal se základem celé generace nástrojů, pomocí kterých uživatelé zadávají dotazy a vytvářejí sestavy.



Obr. 1 Relační technologie

Za to, že tento jednoduchý datový model umožňuje konstrukci elegantního dotazovacího jazyka, je třeba zaplatit. Skutečná složitost vazeb mezi daty neodpovídá jednoduše řádkům a sloupcům. Data jsou často rozdělena do více tabulek, které je i v případě jednoduchých úloh nutno „spojovat“. To vede ke dvěma problémům: a) nutnost spojování více tabulek značně komplikuje zápis dotazu, b) při zpracování složitých dat v relačních databázích může docházet k neúměrnému nárůstu režie.

Jazyk SQL se stal standardem v nástrojích pro vytváření sestava a při zajišťování interoperability databází. I když jazyk SQL vznikl pro potřeby relačních databází, nemusí se omezovat pouze na ně.

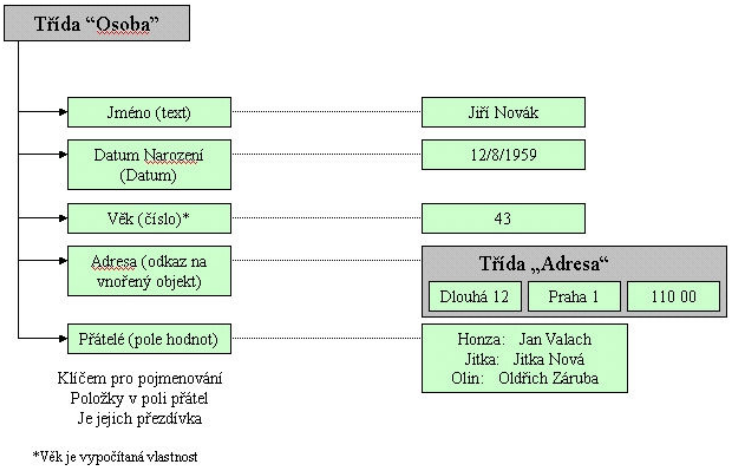
Objektová technologie a objektové databáze

Objektové programování a objektové databáze jsou praktickým výsledkem práce plynoucí z potřeby modelovat složitou činnost mozku. Bylo zjištěno, že mozek je schopen ukládat velmi složité a různorodé typy dat a zároveň s takto zdánlivě rozdílnými informacemi pracovat shodným způsobem. Do programů je třeba implementovat velmi složité chování, ale při tom tomu složitost skrýt. Obě tyto charakteristiky se stávají stále více skutečností současných novějších aplikací.

Objektová technologie se pokouší napodobovat způsob skutečného lidského uvažování o informacích a jejich použití. Entity jsou chápány jako objekty mající stav (představovaný aktuálními hodnotami dat) a chování (které může být často ovlivňováno jejich kódem). Oproti relačním tabulkám se v objektech spojují dohromady data a kód. Teoreticky, a někdy i prakticky, je to balíček, který obsahuje hodnoty dat objektu (vlastnosti) a kopii veškerého svého kódu (metody).

Komunikace mezi metodami uvnitř objektu nebo s jinými objekty je zajišťována předáváním zpráv. Objekty téže třídy mohou sdílet společné kopie kódu, a tím snižovat nároky na paměť.

V objektové technologii je složitost dat obsažena v objektu a přístup k datům je realizován prostřednictvím jednoduchého konzistentního rozhranní. Naproti tomu relační technologie používá sice rovněž jednoduché konzistentní rozhranní, ale ukládá data co nejjednodušším způsobem, takže se složitostí dat se musí neustále vypořádávat uživatel nebo programátor.



Obr. 2 – Objektová technologie

Protože objekty modelují složitá data snadno, je výhodnější programovat složité aplikace pomocí objektů. Objektový přístup je výhodnější i pro vkládání dat do databáze a jejich aktualizaci (např. pro transakční zpracování).

Moderní databázové systémy doplňují objektový přístup objektově rozšířeným dotazovacím jazykem SQL.
Nejdůležitější objektové koncepty jsou:

- dědičnost
- vícenásobná dědičnost
- zapouzdření
- polymorfismus

Většina vývojářů, kteří tvoří nové databázové aplikace, dávají přednost objektové technologii, protože vývoj složitých aplikací je mnohem rychlejší a případné pozdější změny mnohem jednodušší. Objektová technologie poskytuje mnoho výhod:

- Objekty podporují bohatší datové struktury, které mnohem výstižněji popisují skutečná data.
- Programování je jednodušší. Je snadné sledovat, co děláte a s čím pracujete.
- Díky objektům lze jednoduchým způsobem propojovat různé technologie a různé aplikace.
- Objektová technologie přirozeně odpovídá jazyku Java, umožňujícímu snadný vývoj webových aplikací a grafickým uživatelským rozhraním.
- Objektová technologie je použita v mnoha nových nástrojích.

Objekty zajišťují dobrou izolaci mezi uživatelským rozhraním a zbytkem aplikace. Pokud bude třeba zavést nový typ uživatelského rozhraní (například web nebo jiný zatím neznámý typ rozhraní), bude možné znovu použít většinu kódu.

1. Databázový systém Caché

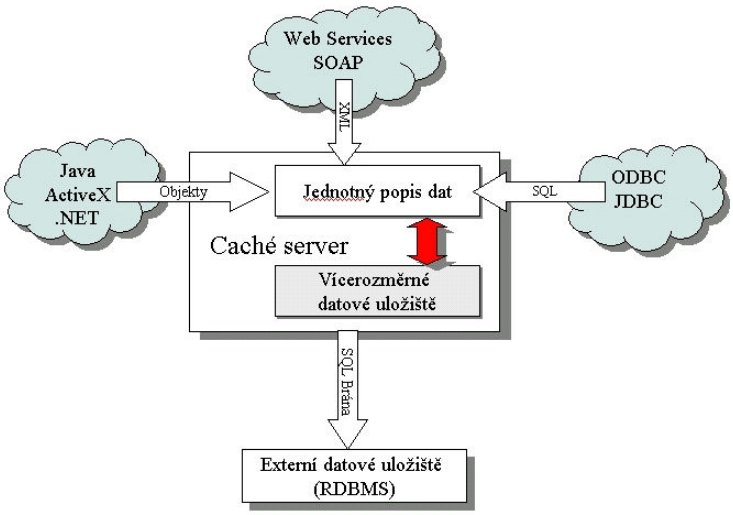
Obsahem mé práce byl návrh a popis vytvoření modulu ambulance, který svým rozsahem pokrývá jednu z důležitých součástí nemocničních informačních systémů.

Z důvodu nasazení v uvedené oblasti, kde se zpracovávají vysoce citlivá data a kde se klade důraz především na spolehlivost a výkon aplikace jsem jako databázový systém zvolil post-relační databázi Caché (čtěte „kašé“). Jde o vyspělý nástroj pro tvorbu komplexních aplikací založených na práci s persistentními (trvale dostupnými) daty. Na rozdíl od klasických relačních databází i čistě objektových databází je Caché díky své architektuře uložení dat schopna pracovat s daty jak z pohledu objektového tak relačního. Kombinací těchto dvou základních přístupů Caché dává vznik samostatné kategorii databází, a to databází postrelačních. Tedy takových, které umí pracovat s daty pomocí jak SQL jazyka tak pomocí objektové syntaxe.

Pro Caché je velmi příznačné, že na rozdíl od např. objektově-relačních databází nedochází ke ztrátě výkonu při přístupu k datům pomocí objektové nadstavby neboť Caché nic takového nemá. Má jen dvě stejně výkonné projekce dat.

1.1 Unifikovaná struktura databáze

Základem modelování komponent aplikací jsou objekty. Objekty jsou organizovány do tříd ve kterých jsou popsány vlastnosti (data) a metody (chování) objektů. Definice tříd jsou uloženy spolu s ostatními daty v jednotném úložišti zvaném Caché Class Dictionary (slovník tříd). Tento slovník tříd je sám o sobě databází a ostatní objekty k němu mohou objektově přistupovat.



Obr. 3 – Unifikovaná struktura databáze Caché

V okamžiku zkompilování definice třídy dojde k vytvoření dvou různých, navzájem synchronizovaných sad kódu, které zajistí optimální přístup k instancím objektů třídy buď pomocí objektového nebo relačního přístupu.

Definice tříd je možno v Caché vytvořit několika různými způsoby:

- „ručně“ pomocí Caché Studia
- “relačně” pomocí DDL jazyka při zavolání DDL příkazu pomocí SQL. Caché automaticky vytvoří na základě definice tabulky definici třídy.
- Pomocí XML, Caché je schopno načíst XML dokument, má-li dokument správnou strukturu, Caché vytvoří definici třídy
- Programově, pomocí objektů. Caché obsahuje systémové třídy pro práci se slovníky tříd.
- Pomocí UML, Caché umí importovat a exportovat datové modely vytvořené v Rational Rose a Microsoft Visual Modelleru.

1.1.1 Přístup k definici uložení dat

Objektový model používaný Caché je v porovnání s objektovými modely jiných programovacích jazyků rozšířen o prvky z relačního prostředí. Lze tedy například definovat indexy, omezení (constrains) a strukturu uložení dat.

Definice fyzické struktury uložení dat objektů je nezávislé na popisu třídy. Vývojáři mají možnost vybrat si přednastavenou strukturu používanou kompilátorem tříd, nebo zvolit vlastní strukturu a ručně ji nadefinovat s ohledem na maximální výkon při dotazech nebo transakcích apod.

1.1.2 Objekty a detaily jejich implementace

Caché disponuje plně vyvinutou objektovou databází pro práci v prostředích náročných na výpočetní výkon a množství zpracovaných transakcí.

Objektový model Caché zahrnuje mj.:

Třídy: to jsou základní kameny objektového programování. Caché umožňuje definovat třídy uchovávající aplikační data a vykonávající aplikační logiku.

Vlastnosti: těmito jsou popisována data ukládaná a zpracovávaná třídami.

Metody: popisují chování objektů a vzájemnou interakci.

Relace: popisují vztahy mezi jednotlivými třídami v aplikaci.

Dědičnost: třídy definované v Caché mohou být odvozeny od jiných tříd a přejímat jejich vlastnosti a metody a další (např. parametry)

Různorodost (**polymorfismus**): třídy mohou modifikovat zděděné vlastnosti a metody.

Řetězení referencí: Caché automaticky při otevření instance třídy otevírá instance všech tříd které jsou ve vztahy k otevřené instanci (reference/relace)

Typy tříd

Objektová technologie implementovaná systémem Caché zahrnuje následující typy tříd:

- Abstraktní třídy
- Registrované třídy
- Persistentní třídy
- Vnořené třídy
- Odvozené třídy
- Datové typy

Nyní k popisu jednotlivých typů tříd:

Abstraktní třídy slouží k popisu obecných tříd, od nichž se v rámci aplikace odvozují třídy představující konkrétní modelované objekty. Abstraktním třídám nelze vytvářet instance objektů.

Registrované třídy implementují plnohodnotné objekty ve smyslu OOP. Vzhledem k tomu, že nepodporují persistenci, využívají se zpravidla pro tvorbu kódu aplikační logiky.

Persistentní třídy slouží k popisu dat, která jsou trvale k dispozici, tedy dat ukládaných na disk nebo jiné záznamové médium. Každá persistentní třída je odvozena od třídy registrované, při kompilaci vytváří kód který zajišťuje komunikaci mezi aplikační vrstvou (objekt) a fyzickou vrstvou (globály). Příkladem persistentní třídy je třída *Pacient*.

Vnořené třídy jsou podobné třídám persistentním ale na rozdíl od nich nemají vlastní úložiště dat. Data obsažená ve vnořených třídách jsou vždy uložena spolu s daty persistentní třídy jež na vnořenou třídu odkazuje. Příkladem vnořené třídy je *Adresa*.

Odvozené třídy jsou obecně třídy odvozené od jedné nebo více uživatelských tříd (*Dědičnost*). Odvozené třídy obecně mohou modifikovat vlastnosti a metody svých nadtříd (*Polymorfismus*).

Datové typy jsou třídy které slouží jako popis dat. Datové typy mohou být libovolně složité. Úlohou datových typů je mj. zajistit správnou konverzi dat mezi zobrazovanou (externí) podobou a jejich uloženou (vnitřní) podobou, dále zajistit validaci vstupních hodnot aj. Zvláštní kategorií tříd jsou CSP stránky, což jsou třídy sloužící pro tvorbu dynamických webových aplikací a Web Services jež implementují webové služby.

Vlastnosti, metody, vnořené dotazy

Vlastnosti tříd mohou být:

- Datový typ
- Jednoduchá reference na jinou třídu
- Jedna strana relace typu: 1-N, rodič-dítě

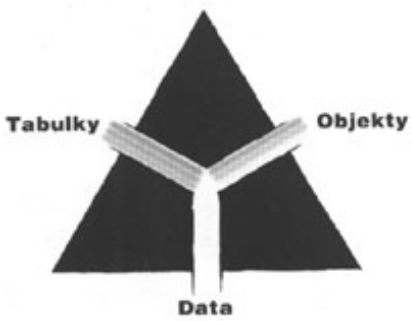
Vlastnosti mohou být privátní, vyžadující neprázdnou hodnotu, indexované, vyžadující unikátní hodnotu, transientní, vypočítané, vícerozměrné. Vlastnost může bát označena jako finální, takovou vlastnost nelze v odvozených třídách modifikovat.

Metody mohou být metodami instance nebo metodami tříd, (pak pro jejich volání není nutno otevírat instanci třídy). Je-li metoda metodou třídy může být zároveň označena jako uložená procedura SQL. Metody také mohou být, podobně jako vlastnosti, privátní a finální.

Třídy mohou definovat též vložené dotazy. Vložené dotazy mohou být definovány buď pomocí jazyka SQL nebo pomocí Caché ObjectScriptu. Výhodou vložených procedur je to, že jsou zkompilovány spolu s definicí třídy a tím se šetří čas při běhu programu. Vložené dotazy dovolují zadávání parametrů. Pomocí parametrů tříd a vlastností lze modifikovat například způsob validace hodnot při ukládání třídy na disk.

Vztah objektů a relačních tabulek

Jak již bylo uvedeno dříve, při kompilaci definice třídy se automaticky vytváří dvě sady kódu. První z nich je pro potřeby objektového přístupu a druhá sada rutin definuje přístup k vlastnostem třídy jako ke sloupcům v tabulce.



Obr. 4 – Vztah objektů a relačních tabulek

Jedním z projevů nezávislosti objektové a relační projekce datového modelu je i možnost přidělit třídě alternativní název jakožto název tabulky. Totéž platí i pro vlastnosti třídy. Důvod je nasnadě. SQL obsahuje klíčová slova, která nemohou být použita jako názvy tabulek nebo sloupců, zatímco svět objektů je daleko svobodnější. Jen pro ilustraci, slovo USER je klíčovým slovem SQL a tedy jej nelze v názvu tabulky či sloupce použít, zatímco v objektovém datovém modelu je poměrně často použito pro název vlastnosti či třídy popisující např. uživatele aplikace.

Každá vlastnost třídy která má jednu hodnotu je automaticky transformována na sloupec relační tabulky. Vlastnost typu pole, které nabývá libovolného počtu hodnot, je transformována na samostatnou tabulku obsahující odkaz na původní tabulku.

Při kompilaci návrhu třídy Caché sama vytvoří kód, který zajistí přístup k fyzické datové vrstvě, programátor se vůbec o tyto vrstvu nemusí zajímat. Nicméně, a tím se Caché výrazně liší od ostatních databázových nástrojů, vývojáři mohou sami definovat způsob uložení dat ve fyzické vrstvě. Díky této otevřenosti Caché je možno provádět optimalizaci / modifikaci uložení v případech kdy původní zvolený způsob ukládání dat není pro náročnost aplikace optimální. Je to velmi zřídka, ale stává se to.

Nicméně s potřebou "ručně" optimalizovat se vývojáři setkají jen u opravdu rozsáhlých databází (GB) provádějících statisíce a více transakcí denně.

1.2 SQL přístup

SQL je v Caché implementováno jako součást unifikované architektury databáze. Skládá se ze dvou částí:

- SQL Procesor a optimalizátor – tvoří jej sada programů, jež mají za úkol analyzovat SQL dotaz a najít nejlepší strategii pro vytvoření cílového kódu.
- Caché SQL Server – toto je sada procesů, jež obstarávají veškerou komunikaci s Caché ODBC a JDBC ovladači. Zároveň obhospodařují seznam často používaných dotazů s cílem docílit maximální rychlosti odezvy. Dotazy, jež jsou načteny v cache, není nutno znovu analyzovat a kompilovat, provádějí se bezprostředně.

1.2.1 Implementace SQL

Caché SQL obsahuje plnou sadu standardních relačních funkcí:

- Možnost definovat tabulky a pohledy pomocí DDL jazyka
- Možnost provádět dotazy nad tabulkami a pohledy pomocí DML

- Podpora transakčního zpracování, včetně příkazů INSERT, UPDATE a DELETE.
- Možnost definovat a používat indexy pro rychlejší provádění dotazů
- Podpora mnoha datových typů včetně možnosti používání vlastních datových typů
- Možnost definování uživatelů a rolí včetně přiřazení práv
- Možnost definovat externí klíče a jiná integritní omezení
- Podpora INSERT, UPDATE a DELETE trigerů
- Možnost definování a provádění uložených procedur
- Možnost vracení dat v různých formátech v závislosti na typu aplikace
- Podpora bitmapových indexů
- Podpora uživatelem definovaných funkcí, atd...

SQL-92 Kompatibilita

Caché implementuje plně základní úroveň standardu SQL-92 s několika málo výjimkami.

Rozšíření

Caché SQL přináší několik zajímavých rozšíření, jež vyplývají z faktu, že Caché je postrelační databází a umožňuje jak objektový, tak relační přístup ke stejným datům.

- Podpora uživatelsky definovaných datových typů
- Podpora objektové syntaxe
- Podpora dědičnosti a sub-classingu
- Možnost definovat strukturu uložení dat za účelem maximalizace výkonnosti.

Následující kód ukazuje jakým způsobem je využita objektová syntaxe rozšiřující SQL (za účelem demonstrace jsme rozšířili definici třídy Osoba o vlastnost Partner jakožto referenci na případného partnera):

```
Select Jmeno, Partner->Jmeno from SQLUser.Osoba where ID=1
```

Tento kód je významově totožný se standardním (méně přehledným) kódem:

```
Select A.Jmeno,B.Jmeno from Osoba as A,Osoba as B where A.ID=1 and A.Partner=B.ID
```

1.2.2 Univerzálnost

JDBC

Caché JDBC driver je kompatibilní na úrovni 4 (čistá Java) a mezi jeho hlavní rysy patří:

- Vysoký výkon
- Kód pouze v Javě
- Podpora UNICODE

ODBC

Caché ODBC je ***nativní*** driver. Není založen na žádné proprietární mezivrstvě. K základním rysům ODBC driveru patří mj.:

- Vysoký výkon
- Přenositelnost
- Podpora UNICODE

1.3 Skriptovací jazyky

To, co činí Caché tak mocným nástrojem pro tvorbu aplikací, je bezesporu možnost psát aplikace pomocí Caché vlastních skriptovacích jazyků. Caché kromě databázového stroje totiž obsahuje též stroj skriptovací, jenž umožňuje programátorům psát libovolné programy pro manipulaci s daty a pro zpracování libovolných transakcí.

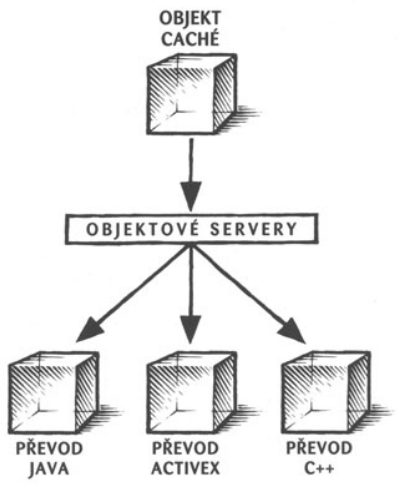
S jistou dávkou zjednodušení lze Caché směle považovat za „programovací jazyk“ stejně jako např. C, Javu, Visual Basic aj. Lze psát i celé aplikace aniž by kód jakkoli přistupoval k datům uloženým v databázi a manipuloval s nimi. Například lze napsat aplikaci, která bude fungovat jako „message broker“, tedy bude zprostředkovávat komunikaci mezi aplikacemi (a databázemi) třetích stran, přijímat, zpracovávat a odesílat zprávy od původce k příjemci.

V současné době je možno v Caché používat dva tyto jazyky a případně je i kombinovat. Historicky starším je jazyk **Caché ObjectScript**, nově je pak k dispozici **Caché Basic**. Oba jazyky jsou rovnocenné, z obou zdrojových kódů vzniká stejný zkompilovaný kód. To, čím se oba jazyky liší, je samozřejmě syntaxe a filosofie. ObjectScript nemá na první pohled zřejmou analogii v jiném programovacím jazyce, zatímco Basic byl vytvořen úmyslně tak, aby co možná nejvíce využíval Visual Basic.

Zájemce o podrobný popis jednotlivých jazyků a jejich syntaxi mohu odkázat na podrobnou a příjemně zpracovanou dokumentaci, která je k dispozici přímo po nainstalování serveru Caché.

1.4 Vývoj aplikací s COM rozhraním

Technologie Microsoft COM je v současné době nejrozšířenější softwarovým komponentovým modelem. Proto jí budeme věnovat pozornost – nejdříve ze všech možných způsobů propojení Caché a produktů třetích stran.



Obr. 5 – Objekty Caché a COM

Komponentový softwarový model (COM) je architektura a podpůrná infrastruktura pro psaní robustních softwarových komponent. Základem je přesně specifikovaný model, který musí tvůrci COM komponent dodržovat, aby jejich produkty byly použitelné ostatními tvůrci softwaru. Technologie COM pracuje s komponentami v binárním tvaru. Tedy vnáší nutnost používat binárně kompatibilní operační systémy (nikoli postačující, ale nutné WINDOWS).

Na druhou stranu obrovským způsobem ulehčuje vývojářům práci při tvorbě nových aplikací, neboť jim stačí použít existujících komponent a dopracovat jen to málo, co je specifické pro jejich aplikaci.

Pod pojmem COM se skrývá řada dílčích technologií a rozšíření původního modelu COM, jako jsou DCOM, COM+, MTS a ActiveX. Zvláště pojem ActiveX bývá občas chybně zaměňován s COM.

1.4.1 COM a Caché

Caché technologie COM plně podporuje a umožňuje tak vývojářům tuto technologii využívajícím psát aplikace, které obsahují propojení do databázového stroje systému Caché.

Základem propojení mezi klientskou aplikací a Caché serverem je knihovna **CacheObject.dll**, která implementuje několik užitečných tříd, přístupných z různých vývojových prostředí (IDE) založených na COM technologii.

Třída Factory

Tato třída je základem objektové komunikace. Má-li daná aplikace komunikovat s Caché třídami, musí nejdříve vytvořit instanci třídy **CacheObject.Factory** a poté pomocí některé z metod třídy **Factory** otevřít spojení se serverem a udržovat jej po dobu komunikace (což nemusí nutně být celá doba provozu aplikace).

Třída ResultSet

Tato třída slouží k provádění dotazů, jež jsou součástí její definice. Lze ji též použít k provádění ad-hoc SQL dotazů (bez použití ODBC).

Třída Syslist

Tato třída slouží k práci s datovými strukturami ve formátu *$List Caché ObjectScriptu*.

Třídy BinaryStream a CharStream

Tyto třídy se používají k manipulaci s binárními a/nebo znakovými datovými proudy. Jsou obdobou Caché tříd *%Library.BinaryStream* a *%Library.CharacterStream*.

Třída ObjInstance

Tato třída reprezentuje klientskou kopii instance Caché třídy v prostředí aplikace.

Komponenta CacheQuery

Tuto komponentu (formulář) je možno použít k vyhledávání primárního klíče instance zadané třídy dle parametrů definovaných spolu s dotazem třídy. Komponenta se volá zpravidla na základě události vyvolané například stiskem tlačítka na formuláři.

Praktické využití výše uvedených tříd je popsáno ve čtvrté kapitole, která je věnována vlastnímu návrhu aplikace a praktické implementaci, ve které jsou tyto třídy použity

1.5 Základní nástroje pro práci s Caché

Po úspěšné instalaci Caché, se na liště vpravo dole objeví „kostka“, která spolu s nabídkou v menu START umožňuje přístup k jednotlivým nástrojům Caché. Z tohoto místa je přístupná a i rozsáhlá dokumentace.

Obr. 6 – Nabídka nástrojů Caché

Při svém návrhu jsem využíval několik důležitých nástrojů, které nyní krátce představím. Jedná se především o Caché Studio a SQL Manager.

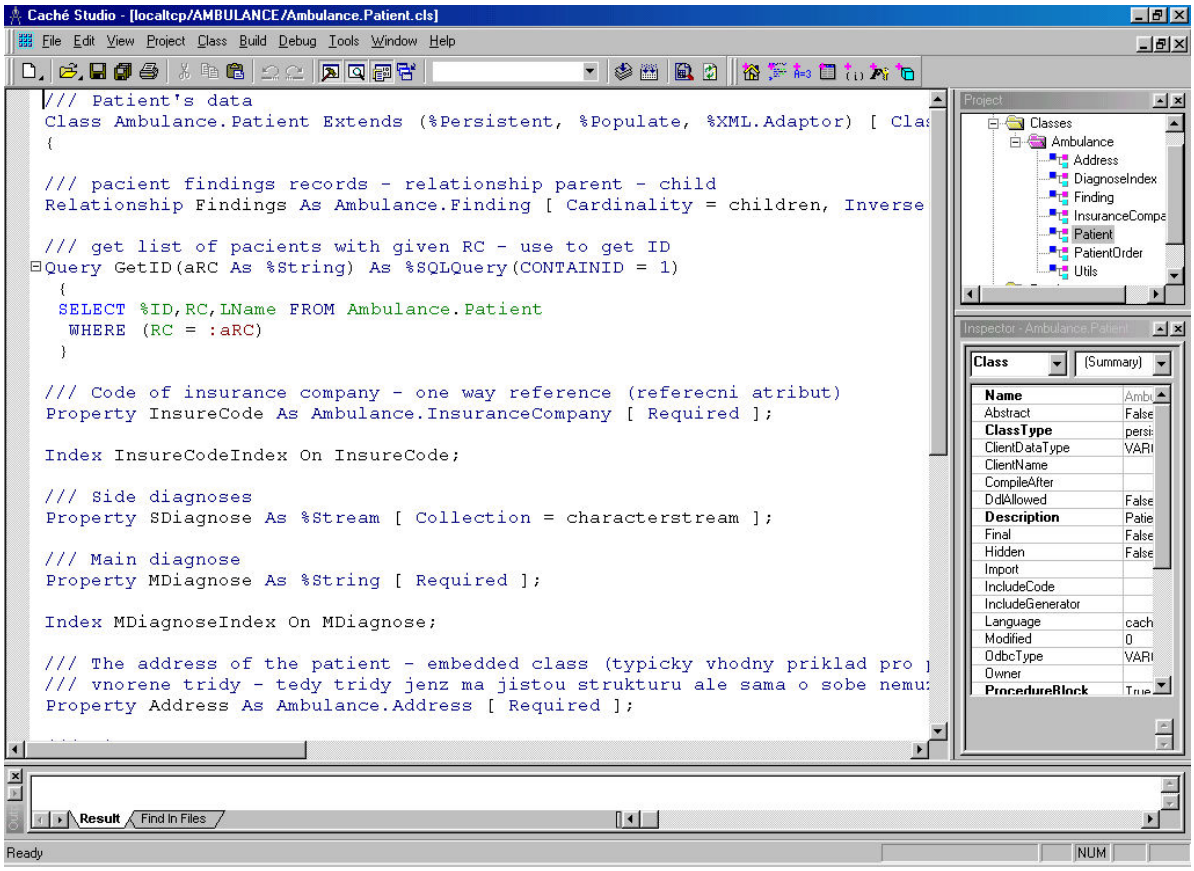
1.5.1 Studio

Nástroj poskytuje příjemné vývojové prostředí, které umožňuje vytváření, úpravu, ladění a kompilaci tříd objektů Caché. Dále zde mohou vývojáři zadávat vlastnosti, psát metody a definovat specializované datové typy. V případě potřeby lze Studio také využít k určení datové struktury pro uložení (jinak ji systém vybere sám) a speciálních charakteristik jazyka SQL, jako jsou triggery SQL a informace pro optimalizaci dotazů.

Návrh datového modelu pomocí Studia

K návrhu složitých datových modelů je vhodné používat některý z profesionálních CASE nástrojů, například firmy Rational Rose, ze kterého je možno model importovat přímo do Caché.

U jednodušších datových modelů je možno začít definici třídy přímo v Caché Studiu. Stejně tak implementace kódu metod tříd vytvořených v CASE nástrojích se provádí v tomto studiu. Při návrhu definice třídy má vývojář možnost použít průvodce pro jednotlivé dílčí úkoly nebo může psát definici přímo pomocí jazyka CDL (Class Definition Language).



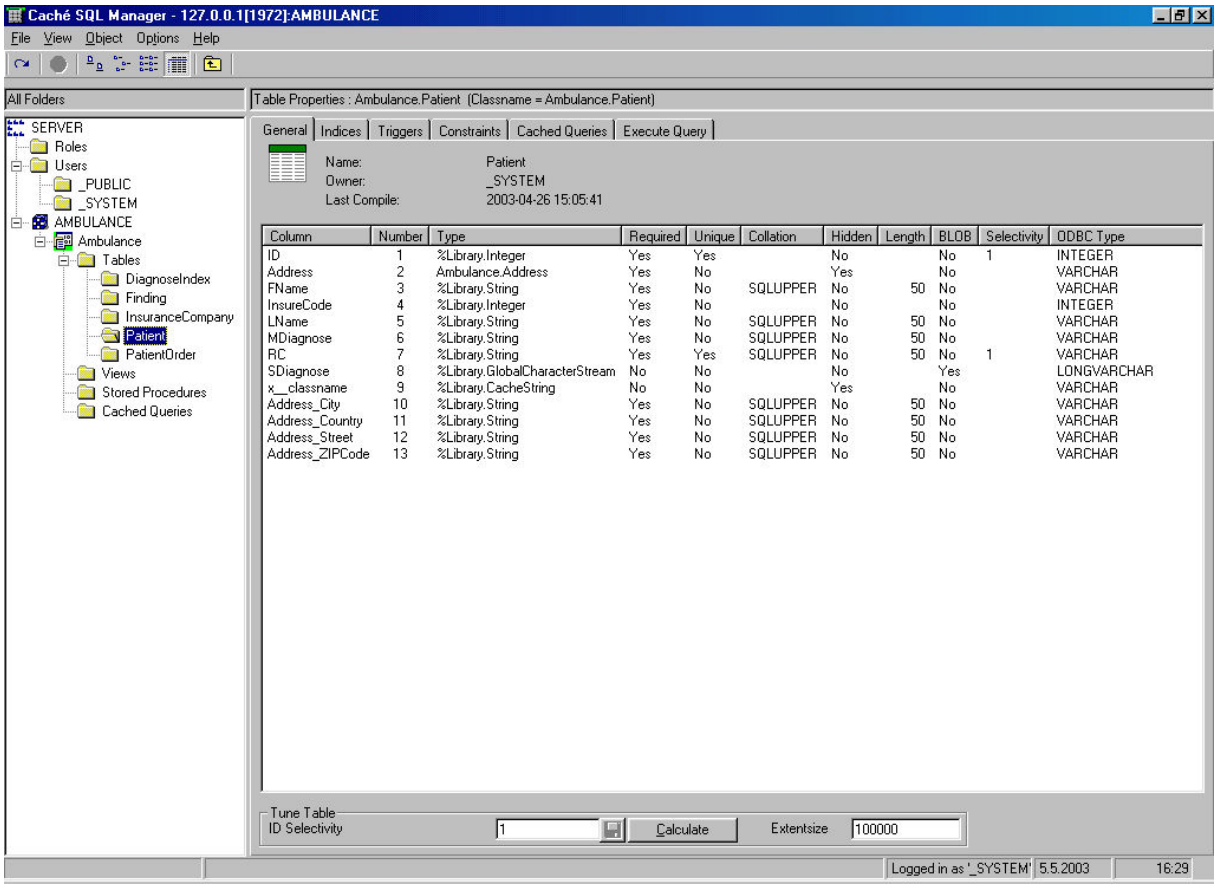
Obr. 7 – Prostředí Caché Studia s praktickou ukázkou definice třídy Patient

1.5.2 SQL Manager

SQL Manager je základní nástroj pro práci s Caché SQL. Hlavními oblastmi použití SQL manageru jsou:

- SQL dotazy a příkazy
- Správa systému - tabulky, pohledy, uživatelé, role, přístupová práva
- Průvodci pro propojování vzdálených tabulek, tj. tabulek definovaných v relačních databázích třetích stran.
- Průvodci pro export/import dat.

Pomocí standardních příkazů SQL (DDL, DML) lze tvořit/rušit tabulky, pohledy atd. a dotazovat se na data. Nedílnou součástí je také možnost definovat uživatele a nastavit přístupová práva k datům.



Obr. 8 – Prostředí nástroje SQL Manager

Pro usnadnění některých složitých operací slouží průvodci, kteří uživatele krok za krokem navigují v procesu. Jedním z průvodců je např. průvodce exportem dat. V několika krocích uživatel zvolí tabulku, z níž se data mají exportovat a formát cílového souboru.

1.6 Uvedení aplikace Caché do provozu a její správa

1.6.1 Hardwarové platformy a operační systémy

Aplikace Caché většinou pracují beze změny na mnoha hardwarových platformách a operačních systémech včetně Windows 95, 98, 2000, NT a XP, všech hlavních platforem UNIX, OpenVMS a Linux.

1.6.2 Konfigurace hardwaru

Caché může být uvedena do provozu na jednovrstvé, dvouvrstvé, třívrstvé nebo peer-to-peer hardwarové konfiguraci bez nutnosti přeprogramování nebo opětné kompilace. Ve velmi rozsáhlých systémech může být více aplikačních serverů a serverů vícerozměrných dat Caché. Obecně ale platí, že daleko nejmenší režie zpracování je tehdy, pokud je aplikační a datový server na stejném počítači.

Klient se obvykle připojuje pouze k jednomu aplikačnímu serveru Caché a tento server zajišťuje, že jsou získána data ze správného datového serveru.

Ve velkých konfiguracích může být nezbytný počítač, který primárně slouží jako datový server a několik počítačů plnících funkci aplikačních serverů. Ve třívrstvé architektuře se

omezením datového serveru na jeden počítač sníží režie spojená s koordinováním více datových serverů.

Jinou běžně používanou architekturou v Caché je architektura peer-to-peer (rovný s rovným), ve které dva nebo více počítačů pracují současně jako aplikační a datové servery. Tato architektura je výhodná v případech, kdy každý počítač pracuje pro skupinu uživatelů, kteří většinou přistupují k datům uloženým na tomto počítači a pouze občas požadují přístup k datům na počítači jiném. Například v nemocnici může být jeden počítač vyhrazen pro aplikaci laboratoře a jiný pro administrativní aplikaci, ale občas potřebují data sdílet.

Caché také podporuje hardwarové clustery („klástry“), ve kterých několik počítačů sdílí přístup ke stejným diskovým jednotkám a koordinuje sdílený a výhradní přístup k blokům disku. Tato architektura poskytuje zvýšenou spolehlivost a větší výkon než jeden počítač, ale správa systému a provoz jsou složité. Podobně jako v síti peer-to-peer se nejvyššího výkonu v tomto prostředí dosáhne, když různé počítače budou zpracovávat odlišné aplikace, což snižuje soupeření počítačů o s tejné bloky disku.

Značně populární jsou zálohovací stínové servery, které představují dobrý způsob zvýšení spolehlivosti. Jejich využitím k vytváření sestav a zpracování dotazů se sníží zatížení primárních serverů.

Tyto principy lze kombinovat a vytvářet tak různé hardwarové architektury.

1.6.3 Provoz Caché s relační databáze

Předností unifikované datové architektury Caché a SQL Gateway je, že aplikace Caché může být překompilována pro spuštění s relační databází. I když výsledný výkon nebude optimální, je tato možnost důležitá pro aplikační partnery, kteří potřebují nabídnout relační řešení s cílem získání nových potenciálních zákazníků.

1.6.4 Instalace aplikace a zavedení změn na pracujícím systému

Vytvořená aplikace se musí nainstalovat později měnit. U většiny technologií to může působit vážný problém, protože může jít o tisíce osobních počítačů a mnoho serverů.

Caché zjednodušuje instalaci softwaru aplikace. Kód jazyka Caché ObjectScript je nutné nainstalovat pouze na jeden datový server. Ostatní počítače získají kopie tohoto kódu pomocí protokolu DCP (Distributed Caché Protocol) a uloží ho do vyrovnávací paměti.

Zavedení změny do rutiny Caché ObjectScript na pracujícím systému je možný pouhým načtením změněného zdrojového kódu na jeden datový server, kde je tento kód trvale umístěn. Zdrojový kód je automaticky zkompilován a aplikační servery jsou upozorněny, že je třeba znovu načíst změněnou rutinu. Taková změna by samozřejmě měla být se zvýšenou opatrností. Proces, který předal ze staré verze řízení jiné rutině, způsobí chybu při pokusu o návrat do změněné rutiny.

1.6.5 Provoz 24 hodin denně

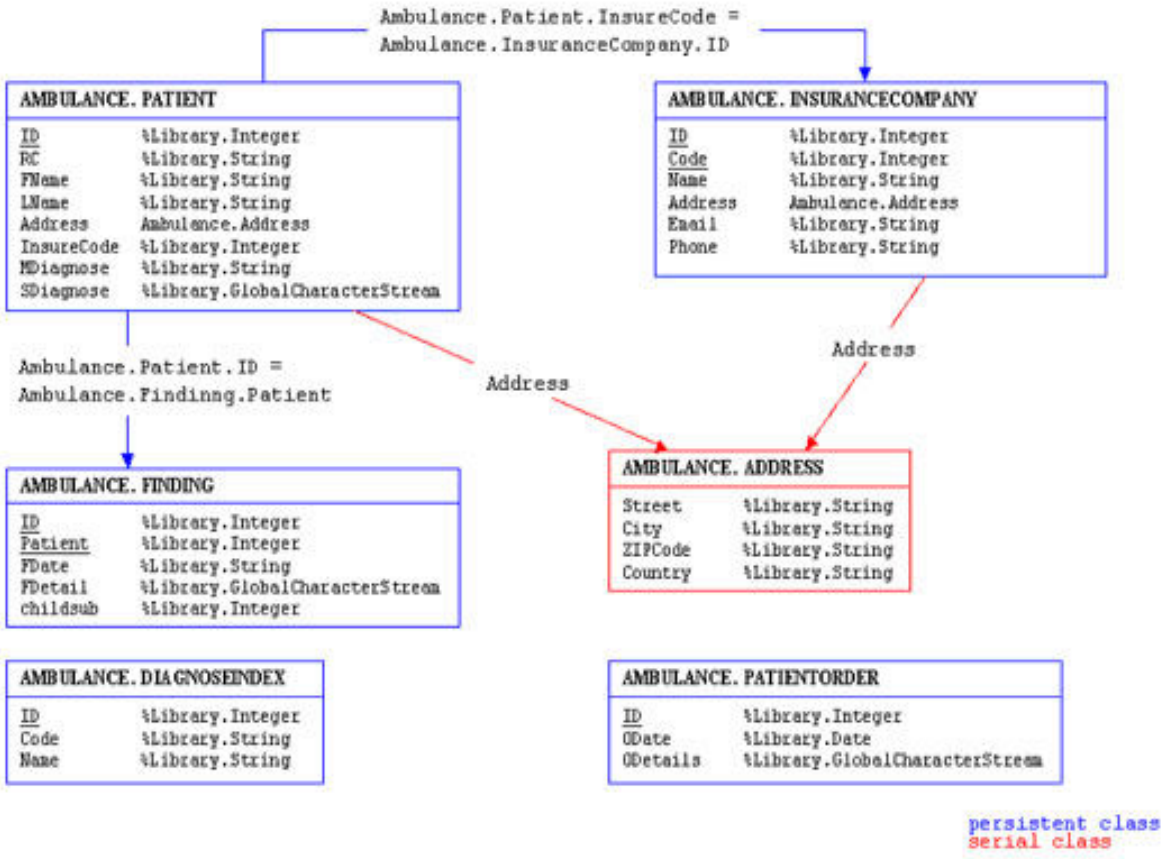
Systém Caché je navržen pro nepřetržitý provoz při současném používání tisíci uživatelů na mnoha místech. Caché pracuje s minimálními nároky na operátora a správu systému. Obsahuje úplnou sadu nástrojů pro správu systému, ke kterým lze přistupovat vzdáleně. Hlavní funkce správy mohou být vykonávány bez obsluhy pomocí skriptů.

2. Objekty Caché při návrhu aplikace

V rámci popisu návrhu aplikace Ambulance se zaměřím především na základní funkce a technologické požadavky aplikace jako je napojení aplikace na databázový server Caché a vlastní objektová komunikace mezi aplikací a serverem.

2.1 Datový model a definice tříd

Vytvořený model aplikace se zaobírá pouze tématem ambulantního provozu, jenž je jednoduchý, není proto ani *datový model* příliš rozsáhlý a komplikovaný. Fyzický datový model je znázorněn na obr. č. 9. Obsahuje šest *entit-objektů* (názvy a definice tříd jsou pojmenovány anglicky).



Obr. 9 – Datový model projektu AMBULANCE

PATIENT (PACIENT)

Persistentní třída PATIENT obsahuje seznam pacientů a uvádím ji především proto, že jde o nejdůležitější entitu modelu. O každém jedinci jsou v databázi uloženy následující údaje: rodné číslo, jméno, příjmení, hlavní diagnóza a seznam vedlejších diagnóz. Adresy a nálezy pacientů jsou řešeny samostatnými třídami napojenými vnořením nebo relací. Následující kód představuje kompletní definici třídy.

Ukázka 1. – Definice třídy Ambulance.Patient

```
/// Informace o pacientech
Class Ambulance.Patient Extends (%Persistent, %Populate, %XML.Adaptor) [ ClassType
= persistent, ProcedureBlock ]
{
/// Nálezy pacienta - relace rodič-potomek
Relationship Findings As Ambulance.Finding [ Cardinality = children, Inverse =
Patient ];
/// Dotaz pro získání ID objektu pacienta podle rodného čísla
Query GetID(aRC As %String) As %SQLQuery(CONTAINID = 1)
{
SELECT %ID,RC,LName FROM Ambulance.Patient
 WHERE (RC = :aRC)
}
/// Kód pojišťovny pacienta formou referenčního atributu do číselníku pojišťoven
Property InsureCode As Ambulance.InsuranceCompany [ Required ];

Index InsureCodeIndex On InsureCode;

/// Vedlejší diagnózy
Property SDiagnose As %Stream [ Collection = characterstream ];

/// Hlavní diagnóza
Property MDiagnose As %String [ Required ];

Index MDiagnoseIndex On MDiagnose;

/// Adresa pacienta
/// vnoření třídy - tedy třídy jenž má jistou strukturu ale
/// sama o sobe nemůže existovat
Property Address As Ambulance.Address [ Required ];

/// Jméno pacienta
Property FName As %String [ Required ];

/// Příjmení pacienta
Property LName As %String [ Required ];

/// Rodné číslo
Property RC As %String [ Required ];

Index LNameIndex On LName;

Index RCIndex On RC [ Unique ];

// dotaz vypisující veškeré infofmace o pacientovy
Query Records() As %SQLQuery(CONTAINID = 1)
{
SELECT %ID,RC,FName,LName,MDiagnose,SDiagnose,InsureCode-
>Code,Address_Street,Address_City,Address_ZIPCode,Address_Country FROM
Ambulance.Patient
 ORDER BY LName
}

}
```

Jsou zde také definovány dva dotazy (Query) *GetID* pro získání čísla objektu pacienta, který se bude otevírat a *Records*, který slouží k výpisu všech záznamů.

Adresa pacienta je řešena formou vnořené třídy ADDRESS, která sama o sobě neexistuje a vytváří se spolu se zakládáním pacienta. Nálezy jsou uloženy v tabulce FINDING a napojení je řešeno relací rodič-potomek. Kód zdravotní pojišťovny je uložen v tabulce INSURANCECOMPANY a odkaz je proveden referenčním atributem.

Kromě výše uvedené třídy a ostatních tříd je v databázové vrstvě definována ještě třída ***UTILS***, která nedefinuje žádnou datovou strukturu, ale pouze metodu Backup, která je programově volána s parametry z formuláře aplikace a vytváří na straně databázového serveru zálohu databáze. Definice třídy UTILS je následující:

Ukázka 2. – Definice pomocné třídy Ambulance.Utils

```
// Třída s definovanu metodou pro vytvoření zálohy
Class Ambulance.Utils Extends (%RegisteredObject, %XML.Adaptor) [ ProcedureBlock ]
{

Method Backup(Type As %String = "F", Desc As %String = "", OutputFile As %String,
LogFile As %String = "", ByRef JournalFile As %String, ByRef Result As %Status = 1)
{
       // najdi název aktuálního žurnálu - důležité pro uložení zálohy
       set JournalFile=$p($zu(78,21),",",2)
       set:LogFile="" LogFile=$zu(12)_"backup.log"
       //
       s ns=$zu(5)
       zn "%SYS"
       set
sc=$$BACKUP^DBACK("",Type,Desc,OutputFile,"N",LogFile,"NOINPUT","N","Y","")
       zn ns
       quit
}
}
```

Takto definované třídy se ve vývojovém prostředí Caché Studia zkompilují a vytvoří tak potřebné datové struktury. Celý projekt s veškerými definicemi tříd lze exportovat do XML nebo CDL souboru a tyto struktury pak kdykoliv importovat a opětovně kompilovat. Tak je tomu i při instalaci této aplikace, kdy se stačí přepnout do správného názvového prostoru, provést import z příslušného souboru s kompilací a datová struktura databáze je připravena pro využití. Datovou strukturu je možné zkontrolovat pomocí nástroje SQL Manager, který nahlíží na strukturu formou tabulek.

2.2 Aplikační vrstva

Vlastní návrh aplikace byl proveden v prostředí *Visual Basic 6.0*, který není pouze programovacím jazykem, ale také integrovaným vývojovým prostředím, ve kterém můžete vyvíjet, spouštět, testovat a odlaďovat svoje aplikace.

V rámci popisu návrhu aplikace Ambulance se zaměřím především na základní funkce napojení aplikace na databázový server Caché a vlastní objektovou komunikaci mezi aplikací a serverem. Uváděné ukázky zdrojového kódu jsou v syntaxi Visual Basicu (VB6), proto čtenáře, kteří tuto syntaxi naznají odkazuji na literaturu [1], [2].

2.2.1 Propojení s Caché serverem

Základem propojení mezi klientskou aplikací a Caché serverem je knihovna *CacheObject.dll*, na kterou je pomocí dialogu References nastaven odkaz, který umožňuje využití tříd pro navázání spojení a komunikaci s Caché.

Pro tuto funkci je speciálně vytvořen programový modul, který je platný v celém projektu pro všechny formuláře. Definuje především důležité globální proměnné jako je *db* (instance třídy CacheObject.Factory) určené pro spojení a komunikaci s Caché. Nejdříve je nutno získat propojovací řetězec a ten předat metodě Connect. Metoda Connect může vracet návratnou hodnotu typu Boolean, kterou lze použít pro kontrolu výsledku operace. Proměnná *rst* (instance třídy ResultSet) určená k provádění dotazů.

Ukázka 3. – Otevření spojení s Caché a globální proměnné

```
' Deklarace globálních proměnných
Public db As CacheObject.Factory
Public rst As CacheObject.ResultSet
Public DocumentForms(10) As New Karta, windex As Integer

' Otevření spojení s Caché serverem
Sub OpenDB()
Dim sdir As String
Set db = CreateObject("CacheObject.Factory")
' Připojení pomocí připojovacího dialogu
'sdir = db.ConnectDlg

If Not db.IsConnected Then
'db.Connect (sdir)
db.Connect ("cn_iptcp:127.0.0.1[1972]:AMBULANCE")
End If
End Sub

' Ukončení spojení se serverem
Sub CloseDB()
Set db = Nothing
End Sub
```

2.2.2 Vyhledávání dat

Vyhledávání dat se provádí spuštěním předdefinovaného vloženého dotazu Caché třídy, tedy zpravidla pomocí SQL. Jakmile je vytvořena instance, lze s touto třídou pracovat stejným způsobem jako například v Caché ObjectScriptu nebo Caché Basicu v rámci Caché prostředí.

Výsledkem hledání je ID instance a poté se otevírá lokální kopie reference na instanci (OREF) *Patient*, opět pomocí proxy třídy **Factory**.

Ukázka 4. – Vyhledávání dat

```
' Hledání ID instance pacienta podle rodného čísla

' Volání Query definované ve třídě Ambulance.Patient
Set rst = db.ResultSet("Ambulance.Patient", "GetID")
rst.Execute currentRC
rst.Next

strID = rst.GetData(1)
' Otevření lokální kopie reference na instanci pacienta
Set Patient = db.OpenId("Ambulance.Patient", strID)

' Synchronizace dat
SyncObjectToScreen
```

2.2.3 Synchronizace dat

Poté, co je nalezeno ID hledané instance a tato otevřena, je provedena synchronizace dat z databáze s formulářem.

Ukázka 5. – Synchronizace dat databáze vs. formulář

```
Sub SyncObjectToScreen()

    txtPrijmeni(0).Text = Patient.LName
    Text1(1).Text = Patient.FName
    RodneCislo.Text = Patient.rc
    Text1(3).Text = Patient.Address.Street
    Text1(4).Text = Patient.Address.City
    Text1(5).Text = Patient.Address.ZIPCode
    Text1(6).Text = Patient.Address.Country

    KodPoj.Text = Patient.InsureCode.Code
    Label10.Caption = Patient.InsureCode.Name


    Text2.Text = Patient.MDiagnose
    Text3.Text = Patient.SDiagnose.Data

End sub
```

Všimněte si že s OREF *Patient* se zachází stejně jako s jakoukoliv třídou v Caché, tedy umí „swizzling“, zřetězení tečkové syntaxe, dědí metody ze svých Caché nadtříd atd.

Dále je možné pro zobrazení dat provést konverzi z vnitřního tvaru do tvaru vnějšího pomocí volání metod *[název_vlastnosti]LogicalToDisplay(...)*. Není to nutné v případě, že vnitřní prezentace hodnoty datového typu je v Caché shodná s její vnější reprezentací.

Samozřejmě, že se používá také opačné synchronizace dat, kdy se data z formuláře synchronizují s daty v databázi.

Ukázka 6. – Synchronizace dat formulář vs. databáze

```
Sub SyncScreenToObject()
Dim obj As Object

    Patient.LName = txtPrijmeni(0).Text
    Patient.FName = Text1(1).Text
    Patient.rc = RodneCislo.Text
    Patient.Address.Street = Text1(3).Text
    Patient.Address.City = Text1(4).Text
    Patient.Address.ZIPCode = Text1(5).Text
    Patient.Address.Country = Text1(6).Text

    Set rst = db.ResultSet("Ambulance.InsuranceCompany", "GetName")
    rst.Execute KodPoj.Text
    rst.Next

    Set obj = db.OpenId("Ambulance.InsuranceCompany", rst.GetData(1))
    Set Patient.InsureCode = obj
    obj.SYS_Close

    Patient.MDiagnose = Text2.Text
    Patient.SDiagnose.Data = Text3.Text
    Finding.FDate = List1.Text
    Finding.FDetail.Data = Text4.TextRTF
    Set Finding.Patient = Patient

End Sub
```

2.2.4 Uložení dat

Při ukládání dat je postup analogický jako při načítání. Nejprve je potřeba provést synchronizaci dat z formuláře s OREF *Patient* a pak se zavolá metoda *sys_Save*, což je totéž jako kdybychom volali metodu *%Save* v rámci prostředí Caché serveru. Vzhledem k tomu, že znak % je vyhrazeným znakem Visual Basicu, nelze jej použít v názvech vlastností a metody a je nutno jej nahradit skupinou znaků „sys_“.

Ukázka 7. – Uložení dat

```
Sub save()
    currentRC = RodneCislo.Text
    Text4.TextRTF = Text4.TextRTF & vbCrLf & vbCrLf

    On Error GoTo actionSaveError
    SyncScreenToObject
    Patient.sys_Save
    Finding.sys_Save
    Kartoteka.validate

Timer1_Timer
Main.StatusBar1.Panels.Item(1).Text = "Ukládání:"
Main.StatusBar1.Panels.Item(2).Visible = True

blModify = False
Exit Sub
```

2.2.5 Ukončení aplikace

Po ukončení práce s instancí třídy (OREF) musíme uvolnit paměť na straně klienta zničením lokální kopie. To se provede nastavením na hodnotu nothing.

Při ukončení aplikace se provede odpojení proxy třídy **Factory** automaticky, je však možno ji odpojit i manuálně pomocí metody *Disconnect*.

Ukázka 8. – Ukončení aplikace

```
Private Sub Form_Unload(Cancel As Integer)
If blModify Then If MsgBox("Uložit kartu?", vbYesNo) = vbYes Then save

Patient.SYS_Close
Finding.SYS_Close
Set Patient = Nothing
Set Finding = Nothing

strID = ""

End Sub
```

Závěrečné zhodnocení

Zhodnocení systému bylo provedeno na postupném popisu vývoje praktickém řešení návrhu ambulantního modulu nemocničního informačního systému pomocí architektury klient/server ve spojení s postrelační databází Caché. Využil jsem výhod objektové technologie v souvislosti s návrhem datové struktury a její reprezentace, včetně dalších technologií a propracovaných COM komponent. Problémy byly vybírány tak, aby korespondovaly s požadavky, které byly stanoveny v úvodu práce.

Nejprve jsem model aplikace vytvořil ve spojení s relační databází Access a MSDE, ke které jsem se připojoval pomocí ODBC spojení. Potřebná data jsem získával pomocí SQL dotazů s využitím objektu *RecordSet* v prostředí Visual Basic. Toto řešení však nebylo vůbec vyhovující. Při větším počtu záznamů v databázi ztrácela aplikace rychlost a počet řádků zdrojového kódu rostl s každým požadavkem na data. Po překlopení aplikace pod databázový systém Caché jsem nejdříve zaznamenal značné snížení počtu řádků zdrojového kódu. Díky přístupu k datům pomocí objektů se vlastní vývoj aplikace výrazně zjednodušil. Dalším významným ukazatelem efektivity řešení byl návrh datové struktury a především stabilita a výkon celého systému. Datový model lze zcela pohodlně upravovat a ladit ve vlastním integrovaném vývojovém prostředí. Po zkompilování definice třídy se změna okamžitě projeví v datové struktuře i s již obsaženými daty.

Na aplikace využívané ve zdravotnictví se klade důraz především na stabilitu, dostupnost a výkon. Mohu jednoznačně uvést, že uvedené řešení těmto požadavkům zcela vyhovuje. Je to především systém Caché, který navrhnuté softwarové řešení zcela posouvá za hranici „obyčejné“ databázové aplikace. Díky přímému spojení se serverem pomocí objektů Caché aplikace vykazuje jak rychlost získávání potřebných dat, tak i nutnou dostupnost a stabilitu. Spolu s propracovaným zálohováním a žurnálováním je zcela ideální pro implementaci v kritických oblastech nasazení. Jde o nově dostupný progresivní nástroj a technologii, která významnou měrou ulehčuje práci softwarového inženýra a umožňuje rychlou tvorbu kvalitních a robustních aplikací.

Literatura

[1] Petroutsos E.: Visual Basic 6 – průvodce programátora., Grada Publishing 1999, ISBN 80-7169-801-6

[2] Petroutsos E.: Visual Basic 6 – průvodce zkušeného programátora., Grada Publishing 1999, ISBN 80-7169-802-4

[3] Kutáč D.: Seriál o Caché., InterSystems B.V. 2002 <http://www.intersystems.cz>

[4] Dokumentace firmy InterSystems: Caché Symposium., 2002 <http://www.intersystems.com>