VYUŽITÍ PLATFORMY .NET V PODNIKOVÉ INTEGRACI

Martin Ota
Ivan Jelínek

ČVUT FEL, Karlovo nám. 13, 121 35 Praha 2, ČR, otam@fel.cvut.cz, jelinek@fel.cvut.cz

Abstrakt
Na Katedře počítačů Elektrotechnické fakulty ČVUT se zabýváme výzkumem IT v prostředí výrobních podniků. Speciálně jsme zaměřeni na CA..-technologie, jejich vzájemné vazby, integraci s podnikovými informačními systémy a ostatním softwarem. Hlavním cílem je zlepšit koloběh CAD-dat. Vzhledem k všudypřítomnosti internetu jsme zvolili jeho mechanizmy jako nosnou platformu komunikačního modelu pro CA..-systémy. V rámci přípravných etap jsme realizovali několik laboratorních experimentů, jejichž podmnožinou bylo zjišťování aspektů internetových aplikací s tlustým a především tenkým klientem v podnikovém prostředí. Vzhledem k nástupu platformy .Net jsme se zaměřili i na její využitelnost. V našem příspěvku jsou diskutovány výsledky těchto experimentů – především problémy aplikací založených na ASP.NET a naopak snadné použití webových služeb. Ty jsme shledali natolik silným a perspektivním nástrojem, využitelným především při aplikační integraci, že jsme se rozhodli uplatnit je i v našem komunikačním modelu. Dále jsme zjistili, že použití technologie ASP.NET v podnikovém prostředí je diskutabilní. Definovali jsme případy, kde je však použití této technologie bezesporně přínosné – distribuované aplikace vyžadující nezávislost na klientovi a aplikace modelu client-server s neznámým vysokým počtem uživatelů. Text rovněž obsahuje zobecněnou úvahu nad tím, kdy a proč je vhodné využít řešení s tenkým klientem a kdy se mu naopak spíše vyvarovat.

1. Úvod

Tento text je zaměřen na část našeho výzkumu, zabývající se průzkumem prostředí IT ve výrobních podnicích. Konkrétně se zabývá CA..-technologiemi, zejména pak CAD-systémy, a vazbami mezi nimi. V této oblasti byl v posledních třech dekádách vykonán intenzivní a široký výzkum. To nás vedlo k provedení určitých přezkoumání a revizí, s následnými návrhy jak využít moderní techniky a technologie při návrhu komplexní infrastruktury IT ve výrobních podnicích. Naším cílem bylo respektovat v této doméně výsledky EAI (Enterprise Application Integration).

Hlavím cílem našeho výzkumu je zlepšit výměnu CAD-dat. Existuje mnoho softwarových systémů, které operují s těmito daty, přičemž nejde zdaleka jen o CA..-systémy, ale i o množství subsystémů, které jsou komponentami komplexního prostředí IT v podniku. Mnoho systémů pomáhá spravovat CAD-data a informace, které jsou v nich obsažené. Jsou to Product Data Management systems (PDM), Product Lifecycle Management sstems (PLM), datové sklady apod., které jsou (nebo by měly být) integrovány do podnikového informačního systému. Mnoho aktivit, jako např. Computer Integrated Manufacturing (CIM) [1] nebo datový formát STEP [2], se snaží řešit problémy CAD-datové výměny. Dnes se však jeví všudypřítomný a internet jako vhodný komunikační kanál i pro systémy CA.. Internet je nyní v tzv. třetí generaci, což znamená, že je využíván i pro interakci neživých uživatelů. Proto jsme se zaměřili na vývoj jednotného komunikačního modelu pro systémy CA.., založeného na internetu [3]. Jsme si vědomi ceny jakékoliv změny ve zmiňovaném softwaru, proto se snažíme zachovávat co možná nejvíce existujícího vybavení. Vzhledem k tomu, že existuje

mnoho vhodných datových formátů – jako např. STEP, IGES, VRML, formáty založené na XML atd. – datové struktury zůstávají beze změny. Náš model definuje nový komunikační mechanizmus – nezávislý na žádné jiné platformě, než je internet – který přináší možnost automatizované výměny těchto datových struktur. Základní vrstva navrhovaného mechanizmu je založena na protokolech TCP/IP, druhá – fakultativní – vrstva na webových službách (XML Web Services). Komunikační model je navržen jako otevřený, s centralizovaným dohledem, realizovaným prostřednictvím Internetového portálu.

První krok našeho výzkumu spočívá v průzkumech a případových studiích stávajícího stavu v reálných podnicích, zejména v aplikační doméně CAD v návrhovém procesu strojního inženýrství. Druhý krok je založen na laboratorních experimentech, protože provádění testů v raných fázích ve skutečném prostředí by bylo neúměrně nákladné. V experimentech jsme testovali několik technických otázek a jejich podmnožina je předkládána v tomto textu. Třetí krok, který bude následovat díky úspěšnému průběhu laboratorních experimentů, je tzv. *action research*, tj. výzkum, který je prováděn v reálném prostředí. Výsledky laboratorních experimentů a především výzkumu v reálném prostředí, budou formulovány a vyhodnoceny – opět ve spolupráci s reálnými podniky – závěrečným průzkumem.

Jedna z podmnožin zmiňované sady experimentů byla věnována využití technologie .Net pro webově orientovanou integraci podnikového softwaru. Dosud jsou hlavním trendem realizace webových řešení systémy založené na Javě a tato technologie je tudíž dobře známá. Proto jsme se rozhodli vyzkoušet technologii .Net, která je přímým soupeřem Javy. Nezabývali jsme se technologií .Net pouze v roli komunikačního kanálu, protože náš projekt zahrnuje i softwarové systémy pro podporu životního cyklu komunikačního mechanizmu. Rovněž jsme chtěli simulovat a testovat možnosti používání komunikační metody ne-CA.. softwarem, včetně klientů pro komunikaci s živými uživateli. Proto jsme se zabývali tzv. tlustými i tenkými klienty a jejich porovnáním, modernizovanou technologii *Active Server Pages* – ASP.NET.

2. Platforma .Net

Platforma Microsoft .Net je softwarová platforma navržená tak, aby poskytla vývojářům vícevrstvou komponentovou architekturu pro vývoj aplikací [4]. Podobá se *Sun Java 2 Platform, Enterprise Edition (J2EE)* [5], avšak aplikační rozhraní .Net není tak těsně svázáno s konkrétním programovacím jazykem – narozdíl od J2EE může být používána napříč mnoha programovacími jazyky. Srdcem platformy .Net je tzv. .Net Framework – běhové a vývojové prostředí, které je: aplikačním běhovým prostředím; sadou objektově orientovaných knihoven, které podporují V/V operace, databázový přístup, komunikační protokoly, řízení bezpečnosti, řízení procesů, vícevláknové programování atd.; jednotnou architekturou pro vývoj desktopových aplikací s grafickým uživatelským rozhraním; nástrojem pro vývoj webových služeb a nástrojem pro vývoj stránek ASP.NET. Základním hostitelským operačním systémem platformy .Net je rodina operačních systémů Microsoft Windows. Počítá se ale s expanzí do ostatních operačních systémů [6].

3. Tlustý versus tenký klient

Platforma .Net umožňuje programátorům vyvíjet dva základní typy internetových aplikací typu client-server – aplikace s tenkým klientem a aplikace s tlustým klientem. Nejedná se sice o specifikum platformy .Net, ale je zde specifická podpora vývoje těchto aplikací.

Logika aplikací typu client-server, které jsou založeny na principu tlustého klienta, je rozdělena do dvou vzdálených modulů. Jeden modul běží na serveru a je společný pro všechny uživatele. Druhý modul je aplikační program, běžící na klientském pořítači. Tento modul je klientem serverového modulu.

Aplikace s tenkým klientem používají standardní klienty, jako např. internetové prohlížeče. Většina logiky je tak soustředěna na serveru a klientský software poskytuje pouze běžné služby, jakými je například vykreslení uživatelského rozhraní, nebo řízení vstupu dat.

Hlavní výhodou modelu s tenkým klientem je vysoká nezávislost na klientovi – uživatelé se mohou připojit k systému, aniž by museli instalovat jakýkoliv specifický software. Mohou tak využívat různé operační systémy, nebo i nepočítačové platformy jako např. mobilní zařízení. Problémy s updatováním klientského softwaru jsou takřka minimální. To je hlavním argumentem současné preference tenkých klientů.

Rostoucí popularita tenkých klientů nás vedla k ověření jejich obecné vhodnosti experimentem. Navrhli jsme zkušební aplikaci v .Net, pracující na simulovaném podnikovém intranetu. Aplikace využívala internetový prohlížeč jako klienta – uživatelské rozhraní tudíž bylo založeno na HTML. Program byl zaměřen na databázový přístup (simulované kusovníky), reprezentaci dat a jednoduché řízení souborů systémů CA.. V prvním kroku byla podniková síť simulována homogenní rychlou sítí (100Mb/s – obr. 1) a v dalších krocích byla síť rozdělena do dvou rychlých podsítí, propojených pomalým spojením (128Kb/s), což není v podnikové praxi nikterak výjimečné (obr. 2, 3 a 4).

3.1 Omezené uživatelské rozhraní

První problém se vyskytl již na počátku vývoje zkušební aplikace. Protože jsme chtěli simulovat reálné prostředí, spolupracovali jsme s uživateli podnikových systémů, kteří komentovali návrh uživatelského rozhraní. Tato spolupráce ukázala, že uživatelské rozhraní založené na HTML je odlišnější než jsme – a především než potenciální uživatelé – očekávali.

Klasická desktopová aplikace je obvykle realizována hlavním oknem s množstvím modálních nebo nemodálních dialogových panelů, tzv. message boxes, dokovacích a jiných oken. Aplikace s tenkým klientem, které využívají prohlížeč HTML jako uživatelské rozhraní, jsou obvykle založeny na jedné nebo více kompaktních HTML stránkách. Fenomén rolování je vždy přítomen. Zkušení uživatelé, kteří používají desktopové aplikace deset, možná patnáct let, nejsou zvyklí používat rozhraní, kde základní ovládací prvky, jako jsou např. tlačítka, rolují společně s informací. Jsou s tímto chováním sžití u webových stránek, avšak u aplikačních programů jej hodnotí výrazně negativně. Omezení napodobenin dialogových panelů, dokovacích oken a záložek (property pages) jsou rovněž bariérou. Toto vše vede jednak k výrazně nižší efektivitě uživatelského rozhraní, tak k enormně vysokému programátorskému úsilí.

Speciálním problémem je ověřování vstupních polí. Jednoduchá validace, jako je test číselného formátu, případně rozsahu, může být realizován JavaScriptem na klientské straně. Mnohdy je ale vyžadována mnohem sofistikovanější validace, např. konfrontace se záznamy v databázi. Mnoho těchto operací tak musí být řešeno na serverové straně. U desktopových aplikací bývá zvykem provádět validaci jakmile je to možné – uživatel je tak ochráněn před vyplňováním nesmyslných položek, které jsou důsledkem dřívějšího špatného vstupu. Jestliže aplikace s tenkým klientem mají poskytovat stejnou službu, musejí obnovovat (tj. znovu

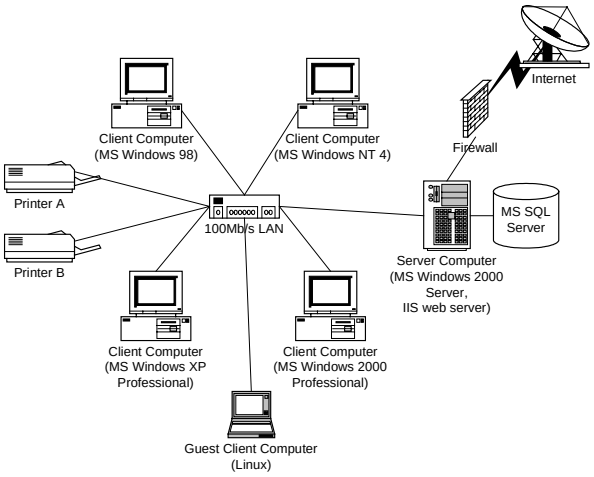
načítat) obsah stránky po vyplnění každého údaje. To však není efektivní a pojí se to i s dalšími problémy (ztráta pozice odrolování, atd.). Takže většina řešení s tenkým klientem kontroluje veškeré vstupy naráz, což není z hlediska uživatele příliš elegantní.

3.2 Problém navigace

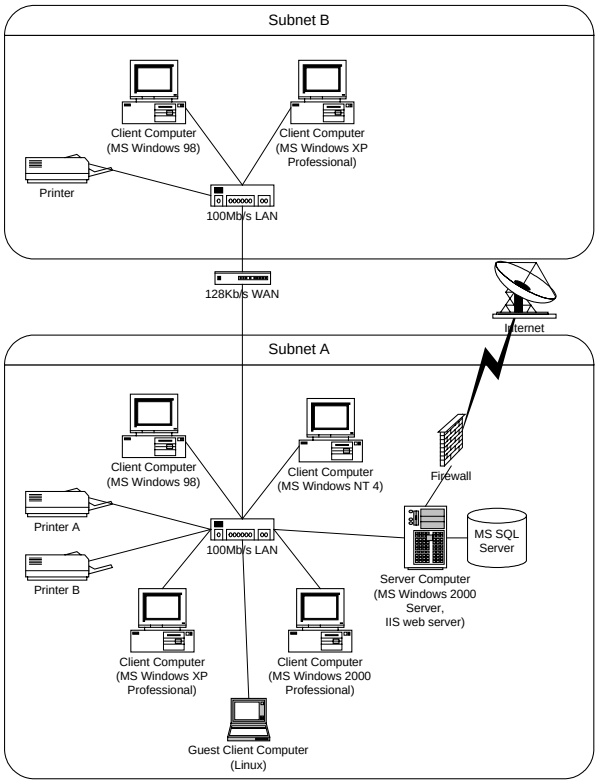
Specifickým problémem je navigace. Klasické aplikace jsou vybaveny nějakým systémem menu a nástrojových lišt. Menu se snadno používají i programují, nástrojové lišty jsou vysoce efektivní, snadno se používají, i realizace je poměrně snadná. HTML v těchto sférách zaostává. Existuje mnoho cest jak sjednat nápravu, jejich společným rysem je ale neúměrné programovací úsilí a problematický výsledek.

Pokud aplikace sestává z více než jedné HTML-stránky, což je běžné, je zde problém se začleněním navigačních prvků. Existují dvě cesty. První spočívá v začlenění kódu navigační struktury do každé stránky. Tato cesta není příliš efektivní – jak z hlediska provozu (což se ukázalo zejména při rozdělení sítě do dvou podsítí), tak z hlediska údržby. Druhá, výrazně efektivnější cesta, je založena na využití tzv. rámů (*frame, frameset*). Jeden rám obsahuje navigační strukturu, ve druhém se vykresluje vše ostatní. Bohužel, ani použití rámů není bezproblémové. Zaprvé, rámy jsou všeobecně považované za zastaralé či nevhodné, dokonce striktní verze HTML (nebo XHTML) toto řešení nedovolí realizovat (kvůli atributu *target*). Zadruhé, jsou popsány problémy s odkazováním z vnějšího prostředí – avšak jednak existují poměrně spolehlivá řešení problému a také se to příliš netýká internetových aplikací. Za zásadní problém, který není všeobecně znám, však experimenty ukázaly špatné zpracovávání rámů v prohlížečích *Internet Explorer*. Jsme si vědomi značné konkrétnosti tohoto zjištění, avšak je důležité mít na zřeteli popularitu a rozšířenost těchto prohlížečů.

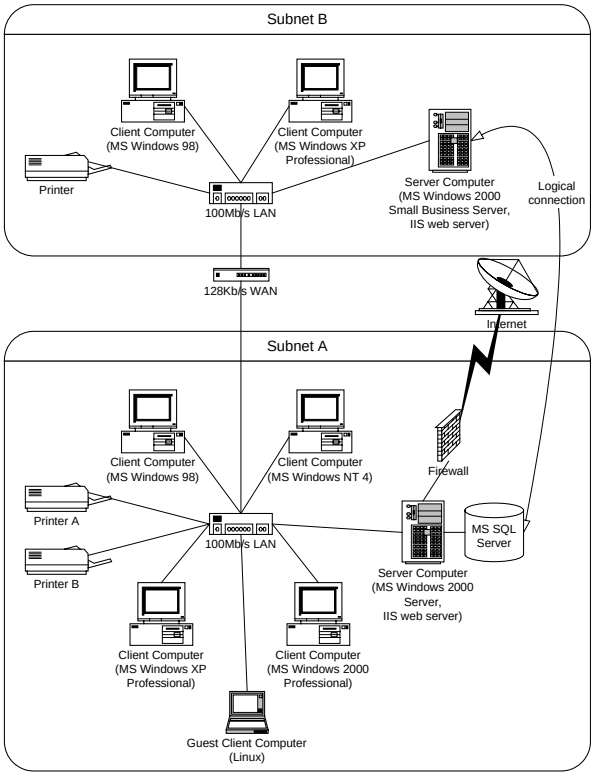
Pokoušeli jsme se problém navigace řešit kompromisním přístupem – něco mezi tenkým a tlustým klientem. Vyvinuli jsme velmi jednoduchou destopovou aplikaci, která sestávala z hlavního okna, které mělo klasické navigační prvky a jeden prvek – ActiveX-komponentu Internet Explorer. Experimenty však ukázaly, že jde – vzhledem k jisté ztrátě nezávislosti a přetrvávání všech ostatních problémů tenkých klientů – o nejhorší řešení.



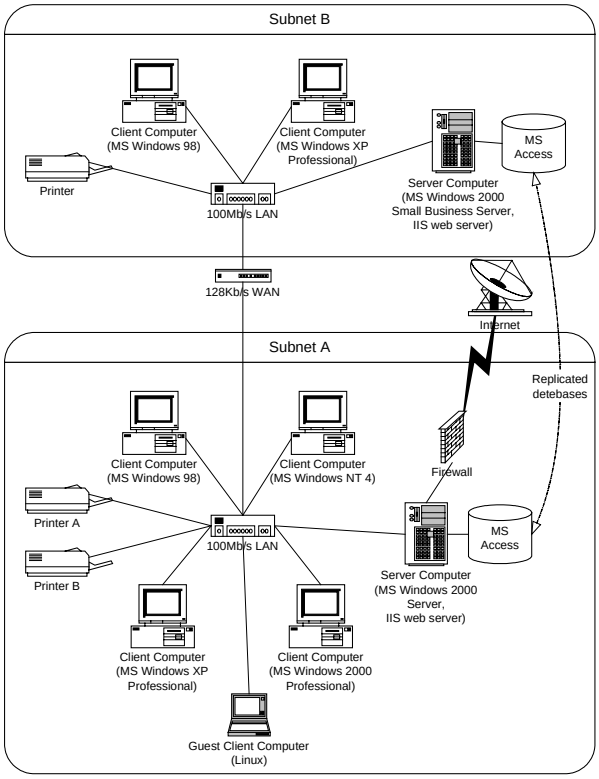
Obr. 1 Homogenní síť



Obr. 2 „Úzké hrdlo“



Obr. 3 Sdílené databáze



Obr. 4 Replikované databáze

3.3 Problém tisku

S tenkými klienty vyvstal problém s tiskem. Pokud se nemá opakovat výše zmíněné fiasko s kombinovaným řešením, je nutné řešení založené na HTML. Avšak tiskové formuláře jsou stejně problematické jako uživatelské rozhraní. Jednoduše proto, že HTML nebylo navrženo pro vytváření formulářů s přesně definovaným vzhledem.

První kategorie problémů se týká stránkování (což částečně řeší druhá verze kaskádových stylů – CSS2), formátování textu (nepříjemná je např. absence tabelačního znaku) atd.

Druhá kategorie se týká vyvolávání tisku a přetrvání nastavení. Jestliže uživatel systému potřebuje tisknout, musí využít mechanizmus prohlížeče. Shledali jsme však, že častou potřebou je tisk více formulářů na několika rozdílných tiskárnách (např. kvůli použití barev nebo formátu papíru). Byť je předem stanovené který formulář přísluší kterému zařízení, není v možnostech tenkého klienta takovou službu poskytnout. Takže uživatel pokaždé volí tiskárnu, formát a orientaci papíru, což je leckdy nepřijatelné.

Poslední kategorie je kontradikce s vývojem HTML – vyvolávání nového okna prohlížeče (čehož se využívá u tiskových sestav) versus striktní verze HTML, resp. XHTML.

3.4 Optimalizace datových toků

Dokud jsme zkušební aplikaci provozovali na rychlé síti, byla výkonnost na klientské straně dostačující. Avšak po zařazení „úzkého hrdla“ do sítě (obr. 2) se aplikace stala téměř nedosažitelnou. Délka HTML-stránek byla okolo 30Kb (průzkumem jsme zjistili, že v případě naší aplikace je to obvyklé). Stránky se simulovanými kusovníky, které byly víceméně jednoobrazovkové, byly přibližně stokilobytové (což záleží i na parametru *ViewState*, který je probírán níže). Tyto stránky se načítaly do 10 sekund, někdy déle (a požadavky na objemnější stránky nejsou výjimečné). Takže jestliže stránka obsahuje tzv. *master-detail* reprezentaci, uživatel je nucen čekat 10 sekund při každé změně výběru v části výpisu zvané *master*. A to je základní princip tohoto problému – kompletní definice uživatelského rozhraní, která je statická, je načítána pokaždé. Naše experimenty ukázaly, že tento problém je nezanedbatelný (uživatelské rozhraní činilo 30 – 100% přeneseného objemu dat). Je třeba zdůraznit, že tlustí klienti tento problém nemají, protože statická definice uživatelského rozhraní stále setrvává na klientském počítači. Datový tok tlustých klientů je méně intenzivní, přenášena jsou pouze dynamická data a vyvážení zátěže v síti je tudíž lepší.

Během experimentů jsme se pokoušeli o optimalizaci tenkého klienta. Zajímavé řešení spočívalo v použití dvou webových serverů – jednoho pro každou podsíť (obr. 3 a 4). Tento přístup pomáhá řešit problém zátěže uživatelským rozhraním, avšak přináší nový problém – jak propojit dvě souběžné serverové aplikace s databází. Řešením může být buď vícenásobné připojení (s minimálně jedním vzdáleným) k jednomu databázovému stroji (obr. 3), nebo dvě replikované databáze (obr. 4). To, které řešení je lepší, záleží na použitém databázovém softwaru a obojí může přinést další specifické problémy. Každopádně existuje řešení jak použít tenkého klienta v takovém prostředí, je však vhodné před volbou architektury udělat důkladnou ekonomickou rozvahu.

4. Aplikace založené na ASP.NET

Platforma .Net poskytuje mocné nástroje pro návrh aplikací s tenkým klientem, souhrnně zvané ASP.NET. Tato technologie, rovněž zvaná ASPX, je následovníkem ASP, avšak je diametrálně odlišná. ASPX, podobně jako ASP, rozšiřují myšlenky HTML. Kód ASP.NET je ve značkovacím jazyce, sestávajícím ze značek HTML a spaciálních značek ASP. Když uživatel vyžádá ASP-stránku, tak je na straně serveru transformována do čistého HTML, tento výsledek je předán uživateli a zobrazen v prohlížeči. Značky ASP a související skripty jsou pravidla pro generování HTML. ASP.NET je zajímavé možností odděleného kódu (*code behind*) a řízením stavu. Oddělení kódu umožňuje programátorům separovat vizuální část kódu (značky HTML a ASP) od části výkonné (jako např. kód v C# [8] nebo VB). Řízení stavu je automatizováno a je snadné držet linii každé jednotlivé uživatelské relace.

4.1 Nezávislost prohlížečů

Když společnost Microsoft uváděla na trh ASP.NET, byla zdůrazňována nezávislost na internetových prohlížečích. Prvním překvapením je přítomnost vlastnosti objektů *DOCUMENT* (tzv. ASP.NET-stránek) *targetSchema*. Tato vlastnost může nabývat následujících hodnot: *Internet Explorer 3.02 / Navigator 3.0*, *Internet Explorer 5.0* a *Navigator 4.0*. Doufali jsme, že jde jen o kompatibilitu se staršími prohlížeči, zejména se třetí generací prohlížeče *Internet Explorer* (IE) a třetí a čtvrtou generací prohlížeče *Netscape Navigator*. Věřili jsme proto, že volba *Internet Explorer 5.0* bude generovat kód použitelný ve všech moderních prohlížečích. Přestože HTML-validátor W3C [7] neoznačil vygenerovanou stránku jako HTML-validní, kód vypadal přijatelně. Avšak naše aplikace se ukázala být na prohlížeči závislá. Testovaná stránka vypadala mírně odlišně v prohlížečích *IE* a *Mozilla* (verze 6 a 1.3). Vyjímaje kosmetických odchylek (zejména vykreslování tabulek je mírně odlišné) se jako kámen úrazu jevil rozměr některých vstupních polí (typu *text area*, *select*). Pokud stránku požaduje *IE*, vše pracuje korektně, avšak v případě *Mozilly* je vypuštěn atribut *style*. Vyzkoušeli jsme všechny hodnoty atributu *targetSchema*, nebyli jsme ale schopni v *Mozille 1.3*, *Netscape Navigatoru 4.77* a *6* obdržet správný výstup. Zkontrolovali jsme i na nastavení generování HTML pro jednotlivé prohlížeče v základním konfiguračním souboru .Net *machine.config*, který se jeví korektně. To, že problém není v prohlížečích, jsme ověřili uložením stránky vygenerované pro *IE* a načtením a korektním zobrazením v ostatních prohlížečích. Pravděpodobně jde o problém IIS a jeho podpory .Net.

4.2 Správa událostí v rámech

Uživatelské rozhraní v HTML umožňuje spravovat určité události, jako např. změny textu, výběru, aktivaci vstupních polí atd. Řízení událostí je ošetřeno skriptováním (v JavaScriptu). Vzhledem k událostnímu modelu na vyšší úrovni poskytuje platforma .Net automatické generování příslušného skriptu, který zajistí volání příslušné části kódu na serverové straně. Pokud je u zdroje události zapnuta volba *AutoPostBack*, je metoda ošetřující událost volána ihned. To pochopitelně vyžaduje okamžité odeslání formuláře, kterým je komunikace se serverovou částí aplikace v .Net realizována, a následným vystavením nové stránky (po ošetření události). Aplikace, sestávající z konstrukce *frameset*, jednoho navigačního rámce a jednoho obsahového, který obsahoval tzv. konstrukci *master-detail*, však vykazovala podivné chování. Dva elementy *select* a příslušné *AutoPostBack*-řešení reprezentovaly tento seznam. Při manipulaci s *master-detail* seznamem aplikace občas končila zacyklením – nebylo možné jej vystopovat mechanizmy ladění, protože aplikace jakéhokoliv mechanizmu ladění chybu potlačila (pravděpodobně rozdílem v časování běhu různých procesů či jejich vláken).

Zajímavé je, že bez rámů (tj. bez obalové struktury a navigačního rámu) vše fungovalo korektně. Nakonec se podařilo extrahovat z aplikace vzorový model – již v čistém HTML (tj. bez vazby na .Net), který havárii též způsoboval. Jde o chybu aplikací rodiny *IE*. Ostatní běžně používané prohlížeče se chovají korektně. Jedná se sice o jednu konkrétní značku prohlížeče, avšak uvážíme-li její procentuální zastoupení mezi běžnými uživateli Internetu, stojíme před velice vážným problémem, který odsuzuje používání rámů v internetových aplikacích, využívajících událostní model.

4.3 Parametr ViewState

S problémem četného vystavování stránek se pojí skrytá pole *ViewState*, která obsahují kódovaný obsah patřičných polí. Tento mechanizmus se snaží udržet informaci o obsahu stránky, je však poměrně značně neefektivní a zatěžuje komunikaci oběma směry. V experimentech osciloval objem dat v těchto polích mezi 50 – 70% celé stránky, průměr byl 68%. Využívání tohoto parametru lze vypnout, programátor však musí při každé události *Page_Load* znovu vyplnit a patřičně nastavit všechna pole, které mechanizmus nevyužívají.

5. Webové služby

Webové služby jsou poměrně novým využitím webu – komunikací stroj-stroj, využívanou k softwarové integraci. Protože zamýšlíme jejich využití pro komunikaci mezi CAD-systémy, zařadili jsme je do experimentální fáze – byť nejde čistě o specifikum .Net. Jsou to webové aplikace, nepodřizují však přenášená data jejich vizualizaci, ale sémantice. Jsou platformně nezávislé, tzn. klient vyvinutý v Javě, provozovaný na Unixu může využívat webových služeb vyvinutých a provozovaných na platformě .Net. Webové služby je doporučeno vytvářet jako bezestavové a co nejvíce nezávislé. Vazba tlustých klientů využívající webové služby jsou tak volnější než u klasických řešení s tlustým klientem.

Zaměřili jsme se na využití webových služeb v podnikovém prostředí. Testy spočívaly v několika oddělených experimentech, přičemž jeden experiment byl zahrnut v dříve zmíněném experimentu s tenkým klientem (obr. 4). Použili jsme dvě jednoduché databáze a synchronizovali je pomocí webové služby. Jde o zajímavou architekturu, protože aplikace koncového uživatele je tenkým klientem serverové aplikace, která je současně tlustým, volně vázaným klientem webové služby.

Veškeré experimenty nás překvapily jak je – s podporou .Net – snadné vyvinout jak serverovou, tak klientskou část. Veškerá netvůrčí činnost je automatizována, bez újmy na funkčnosti a obecnosti. Rovněž spolehlivost zkušebních aplikací byla vynikající. Protože existuje mnoho odlišných cest jak vytvářet webové služby (např. volný Glue nebo Axis), není .Net uzavřen vůči jiným řešením.

Pokoušeli jsme se vytvářet webové služby jako bezestavové. Řízení stavů se však nelze vždy vyvarovat. Potom je velká část této režie na uživateli, bez žádné automatizace ani další podpory.

6. Ostatní aplikace

Probrali jsme dvě inovativní cesty návrhu distribuovaných aplikací, avšak .Net poskytuje i mnoho nástrojů věnovaných vývoji klasických aplikací. Programátoři mohou navrhovat všechny typy aplikací, od konzolových, přes tzv. okénkové, které jsou podporovány

knihovnou WinForms. Tento systém je v porovnání s Microsoft Foundation Classes (MFC) mnohem jednotnější a poskytuje prvky vyšší úrovně a je přístupný pro všechny programovací jazyky. Základní principy programování klasických aplikací zůstávají shodné, pouze jsou zjednodušeny. .Net potírá rozdíly mezi programovacími jazyky, zjednodušuje a automatizuje komponentové programování, rovněž využívání webových služeb a ostatních internetových protokolů.

7. Závěr

Webové služby jsme shledali efektivním nástrojem, který lze využít jako komunikační kanál EAI. Platforma Microsoft .Net je vhodné prostředí pro vývoj a použití této strategie. Výsledkem našich experimentů je doporučení používat webové služby, přednostně v bezestavovém režimu. V souladu s tímto zjištěním posílíme druhou vrstvu našeho komunikačního modelu, založenou na webových službách. V našem případě ponecháváme základní vrstvu na bázi TCP/IP, kvůli specifickému prostředí a širším možnostem využití staršími technologiemi, které mohou být spjaté s konkrétními CA..-systémy. Rovněž pro vývoj desktopových aplikací a aplikací s tlustým klientem lze .Net doporučit, především pro jednoduchost využití webových služeb a obecného přístupu k internetu.

Experimenty ukázaly tenké klienty jako významné řešení, které je však poněkud problematické. Tato řešení jsou nenahraditelná v případě požadavku klientské nezávislosti. Pokud aplikace běží na Internetu a má neznámý a pravděpodobně velký počet uživatelů, zde se rovněž zdá být tenký klient nejlepším řešením. Avšak pokud je aplikace provozována v omezené síti (např. podnikové), je volba tloušťky klienta otázkou. V případě potřeby přesně formátovaných tiskových sestav bude pravděpodobně tlustý klient vhodnější. Zásadně pak nedoporučujeme použití rámů v aplikacích, které jsou na bázi HTML, z důvodu fatálních problémů prohlížeče *Internet Explorer*. V případě zmiňovaného komunikačního modelu budou nabyté zkušenosti využity při vývoji podpůrného internetového portálu.

Literatura:

1. Cheng, W., Yushun, F., Deyun, X. Handbook of Industrial Engineering - 3rd Edition. To be published by John Wiley & Sons. 2003
2. Al-Timimi, K., MacKrell, J. STEP – Towards Open Systems. STEP Fundamentals & Business Benefits. CIMdata Inc., Ann Arbor. 1996
3. Ota, M., Jelínek, I. Information Flow Between CAD and the Rest of an Enterprise. Proceedings of the 17th International Conference on Systems for Automation of Engineering and Research (SAER 2003). Sofia. 2003
4. Microsoft Corporation. Microsoft® .NET Server Solutions for the Enterprise. Microsoft Press, Redmond. 2002
5. Perrone, P., Chaganti, V. Building Java Enterprise Systems with J2EE. Sams, Indianapolis. 2000
6. GNU. DotGNU Portable.NET. http://www.gnu.org/projects/dotgnu/pnet.html, září 2003
7. W3C Consortium. W3C MarkUp Validation Service. http://validator.w3.org/, srpen 2003
8. Gunnerson, E. A Programmer's Introduction to C#. APress, Berkeley. 2000