Česká společnost uživatelů otevřených systémů EurOpen.CZ
Czech Open System Users' Group
www.europen.cz

XXXI. konference

Sborník příspěvků

Jiřičná u Petrovic
21.–24. října 2007

Programový výbor

Rudolf Vladimír, Západočeská univerzita v Plzni
Dostálek Libor, Siemens Praha
Felbáb Jiří, Commerzbank Praha
Sitera Jiří, Západočeská univerzita v Plzni

Sborník příspěvků z XXXI. konference EurOpen.CZ, 21.–24. října 2007
© EurOpen.CZ, Univerzitní 8, 306 14 Plzeň
Plzeň 2007. První vydání.
Editoři: Vladimír Rudolf, Jiří Sitera
Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis, Plzeň
e-mail: servis@vydavatelskyservis.cz

Tisk: TYPOS, Tiskařské závody, a. s.
Podnikatelská 1 160/14, Plzeň

Upozornění:
Všechna práva vyhrazena. Rozmnožování a šíření této publikace jakýmkoliv způsobem bez výslovného písemného svolení vydavatele je trestné.

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-13-6

Obsah

J2EE Eintopf: Základní ingredience a doporučené přílohy

Přemek Brada

E-mail: brada@kiv.zcu.cz

Abstrakt

Technologické základy platformy Java Enterprise Edition prošly od jejího počátku několika proměnami, základní principy pro tvorbu velkých aplikací podle třívrstvé architektury ale zůstávají víceméně stálé. Základní otázka při vývoji J2EE aplikací (zejména v případě vývoje „na zelené louce“) dnes proto nezní až tak „jak“ ale spíše „s čím“: problém nevyzrálé technologie či neznalosti principů je nahrazen problémem orientace ve spleti technologií a aplikačních rámců (frameworků). Cílem tohoto příspěvku je ve zkratce představit základní stavební kameny platformy J2EE, šíři technologií pod touto platformou zahrnutých nebo na ni navazujících, a vývojové platformy související i konkurenční.

1 Úvod

Platforma Java, jejíž začátky spadají do 90. let [4], je v posledních letech známa především pro svojí server-side část oficiálně nazývanou Java EE 5 Platform. Kromě ní jsou samozřejmě k dispozici samostatně také desktopová Java, čili J2SE (Standard Edition), a mobilní J2ME pro aplikace v přenosných zařízeních[1].

V tomto příspěvku se budeme primárně věnovat J2EE ze dvou úhlů pohledu. Jednak bychom chtěli provést jakýsi historický průzkum, v němž provedeme srovnání architektury J2EE aplikací v prvních verzích platformy a dnes. Za druhé chceme představit aspoň v přehledu technologie, které dnes pod „značku“ J2EE spadají oficiálně a ty, které s ní volně korelují. Z obojího bude patrné, že Java na serverové straně představuje dnes komplexní řešení pro velké aplikace, že se i přes různá zakolísání vyvíjí a reaguje na nastupující trendy, a že najít vývojáře s encyklopedickými znalostmi J2EE je úkol pro pohádkového Honzu spíše než pro projektového manažera.

[1] Jistou ironií osudu je, že dnes (zatím) málo rozšířené mobilní a zabudované aplikace byly prapůvodně hlavním cílem pro Javu.

1.1 Základní koncepty

Pro pochopení, jak J2EE aplikace pracují a jaký základ pro ně platforma představuje, nejprve zopakujme základní pojmy. Platforma Java Enterprise Edition [3] je standardizována firmou Sun Microsystems, přičemž na tvorbě standardů (mají formu Java Specification Request dokumentů, JSR) se podílí širší komunita formou Java Community Process (JCP, viz http://www.jcp.org/) aktivit.

Aplikace jsou typicky *vícevrstvé* (oddělená prezentace, aplikační logika, a datová vrstva) což má za cíl zlevnit vývoj a údržbu díky přehlednější architektuře, a umožňuje lepší škálovatelnost a správu. Prezentační vrstva je typicky implementována webovou technologií, kdy uživatelské rozhraní aplikace je generováno na serveru a zobrazováno na tenkém klientu, nejčastěji webovém prohlížeči. Architektura aplikací je typicky *komponentová*, tj. složená z oddělených modulů které je možné (aspoň do jisté míry) vyměňovat. Tento koncept je nicméně v J2EE vykládán poměrně volně (srovnej [5]), takže komponentou je např. jak EJB bean, tak i servlet.

Implementační podoba aplikace ve formě JAR[2] souboru, je nasazena (*deploy*) do *kontejneru*. Ten představuje běhové prostředí, které poskytuje standardizované infrastrukturní služby – řešení závislostí mezi komponentami, sdílení zdrojů, bezpečnost, správu transakcí, jmenné služby, komunikaci komponent i vzdálených klientů s nimi. Perzistenci, zejména napojení na relační databáze, může řešit buď kontejner nebo samostatná technologie, např. EJB Entity Beans. Nejznámějšími J2EE kontejnery jsou Sun Java System Application Server, BEA WebLogic, IBM WebSphere, JBoss, a open source Apache Tomcat (pouze web kontejner) či JOnAS.

Kromě těchto dvou známých věcí je třeba podotknout, že pro funkčnost J2EE aplikací jsou důležité další technologie. Zejména jde o zpracování XML, které se používá v různých oblastech (přenos „dat“, konfigurační a deployment descriptor soubory, mapovací schemata, ...), a o komunikační infrastrukturu (typicky protokoly HTTP a SMTP). Pro úplnost je také možné dodat, že do celkové architektury J2EE aplikací je možno započítat Java aplety běžící v prostředí tenkého klienta.

2 Přehled ingrediencí J2EE

V tomto oddílu popíšeme, spíše výčtovou formou, všechny technologie tvořící dnes platformu Java EE (zabýváme se tedy verzí 5, vydanou v roce 2006). Na strukturu této směsi technologií se podíváme z pohledu vícevrstvé architektury.

[2] Java archive, komprimovaný ZIP formát podle technologické varianty aplikace pojmenovaný WAR, EAR, SAR, RAR, a podobně. Přípony CAR, OAR a BAR jsou zatím volné ;-)

2.1 Datová vrstva

Na úrovni datové či perzistentní vrstvy (v Java tutorialu zvané přesněji vrstva podnikových informačních systémů) poskytuje J2EE několik možností pro přístup aplikace k datové základně. Nízkoúrovňovým základem je JDBC (součást J2SE), které umožňuje přístup k relační databázi pomocí ručně psaných SQL dotazů a procházení získaného výsledu (result set) jako kolekce obecných dat.

Jako „velké“ řešení pro perzistentní vrstvu byly až do předchozí verze J2EE k dispozici entitní Enterprise JavaBean (EJB) komponenty. V Java EE verze 5 jsou nahrazeny standardem JPA (*Java Persistence API*, součást EJB 3.0 výrazně inspirovaná open source ORM frameworkem Hibernate [1]) kdy pro přístup k perzistentním datům je možno použít libovolnou POJO (Plain Old Java Object) třídu označenou anotací – její instance pak reprezentují nejčastěji jednotlivé řádky mapované databázové tabulky nebo kurzoru. Tato implementace perzistentní vrstvy umožňuje také mapování polymorfních hierarchií dědičnosti do relační databáze a použití dynamických dotazů specifikovaných v Java Persistence query language (cosi jako objektové SQL).

Pro enterprise aplikace je klíčová podpora transakcí, kterou zajišťuje *Java Transaction API* (JTA). J2EE kontejnery automaticky podporují transakce na úrovni jednotlivých metod EJB komponent včetně deklarativního označení povinnosti transakce (container-managed transactions). JTA navíc dává k dispozici transakční manažer a třídy, díky nimž je možné do transakce zahrnout různé databáze, databázové servery a transakci rozprostřít v distribuovaném prostředí.

Poslední sada knihoven, které souvisí s datovou vrstvou, souvisí s ukládáním dat v XML formátu. Ve standardní J2SE platformě je k dispozici *Java API for XML Processing* (JAXP) zahrnující SAX, DOM, a XSLT a nově také rozhraní StAX pro sekvenční práci s dokumentem. Enterprise Java na nich staví dále. *Java Architecture for XML Binding* (JAXB) usnadňuje transformaci mezi runtime reprezentací objektů, včetně kontejnerů a stromových struktur, a jejich XML reprezentací definovanou XML schematem – je tedy pomocí něj možné implementovat např. XML-based persistenci resp. serializaci. *Java API for XML Web Services* (JAX-WS) pak tyto transformace používá pro marshalling dat ve zprávách zasílaných mezi webovou službou a klientem.

2.2 Aplikační vrstva

Aplikační vrstva je v J2EE stacku reprezentována oficiálně jedinou technologií, a to Enterprise JavaBean komponentami (EJB), které jsou páteří standardních J2EE aplikací. V tomto příspěvku do ní ale zahrneme také technologii Java Servlets, což zdůvodníme dále.

Aplikační EJB jsou k dispozici již od první verze specifikace, a v principu zůstávají stále stejné (až na způsob implementace ve zdrojovém kódu). Jsou dvo-

jího druhu. *Session beans* implementují aplikační logiku vázanou na konverzaci s klientem přes definované rozhraní – formou relací (stavové beany) nebo volání samostatných metod (bezestavové) – a při jisté míře abstrakce je lze považovat za „chytřejší RMI[3]“. Ve spolupráci s kontejnerem zajišťují pro klienta i tvůrce transparentně škálovatelnost, zabezpečení a pooling, tj. zásobárnu instancí bean pro zvládání nerovnoměrné zátěže a jejich recyklování mezi klientskými požadavky u bezestavových bean. Klient EJB komponenty může být jak vzdálený, typicky v podobě jiné aplikace nebo servletu, tak lokální, zejména jiná komponenta v témže kontejneru.

Asynchronní zpracování požadavků podporují *message-driven beans* tedy komponenty řízené zprávami. Ty jsou napojeny na poskytovatele Java Message Service (JMS) nebo jiné komunikační infrastruktury jako posluchači, a po obdržení zaregistrované zprávy ji zpracují voláním obslužné metody. Stejně jako bezestavové session beans mohou využívat transakční zpracování, clusterování a pooling, a pro přístup k datové vrstvě typicky využívají entitní třídy podle JPA standardu.

Aplikační vrstvu reprezentují také *servlety*, což jsou třídy uložené ve webovém kontejneru které umožňují komunikaci přes protokol HTTP. Tato komunikace je na poměrně nízké úrovni (jednotlivým HTTP metodám odpovídají 1 : 1 servisní metody servletu, které zpracují data požadavku a generují HTML odpovědi) a v posledních letech jsou proto využívány spíše jako implementační základ pro sofistikovanější technologie – JavaServer Faces a podobně. V dobře vytvořené vícevrstvé aplikaci servlety samy nevytvářejí uživatelské rozhraní (což je úkolem prezentační vrstvy, typicky JSP nebo JSF, viz dále) ale koordinují komunikaci s aplikační logikou implementovanou v EJB vrstvě, v jednodušších případech tuto logiku přímo obsahují.

Nepřímo do aplikační vrstvy můžeme zařadit také zabezpečení aplikací. Obecně je v J2EE možno použít jak deklarativní tak i implementační zabezpečení. Při použití prvního způsobu se informace o bezpečnostních rolích, uživatelích a povolených přístupech ukládají do XML deployment descriptor souborů (pro EJB, webové a web service aplikace), takže je možné zabezpečení nastavit při nasazení aplikace dle potřeb konkrétního provozního prostředí.

Implementační zabezpečení, tj. řešení bezpečnosti ve výkonném kódu aplikace, se používá mechanismus anotací a rozhraní *Java Authentication and Authorization Service* (JAAS). Anotace jsou specifické pro jednotlivé typy J2EE technologií, tedy EJB komponenty, webové služby a další. JAAS je součástí standardní J2SE platformy, a díky mechanismu Pluggable Authentication Modules (PAM) umožňuje napojení Java aplikací na různé interní i externí autentikační služby – od jednoduchých hesel, přes LDAP až po Kerberos autentikaci.

[3]Remote Method Invocation, vzdálené volání metod

2.3 Prezentační vrstva

Úkolem prezentační vrstvy je vytvářet uživatelské rozhraní, zprostředkovat komunikaci aplikace s uživateli nebo ostatními systémy. Zejména v případě webových aplikací je při tvorbě uživatelského rozhraní třeba respektovat různá omezení a specifické aspekty, zejména relativně malou vyjadřovací schopnost HTML a velkou variabilitu klientských zařízení. Platforma J2EE nabízí pouze dvě technologie, nepočítáme-li Swing jakožto desktopovou variantu dostupnou ve standardní J2SE platformě.

JavaServer Pages (JSP) je základní technologií, která je založena na obvyklém přístupu zabudování výkonných scriptů (nebo jejich abstraktnějších zástupců) do zdrojového kódu HTML dokumentu. Díky rozšířením a vylepšením, která byla do specifikace JSP v průběhu vývoje přidávána, jsou dnes dosti flexibilním řešením které umožňuje velmi čisté oddělení formátování výstupu od přípravy či zpracování zobrazovaných dat ve výkonné logice – ta je zapouzdřena buď do JavaBean objektů a jejich metod, přístupných v JSP pomocí Expression Language proměnných (mohou odkazovat na různé rozsahy platnosti objektů, od jednotlivého požadavku přes uživatelskou relaci po celou dobu běhu aplikace), nebo do uživatelských značek (tag libraries), které používají XML syntaxi s jmennými prostory ale interně odkazují na obslužné třídy a jejich metody. Standardně je k dispozici knihovna JSTL (JSP Standard Tag Library) pro obecné operace typu větvení dle hodnoty objektu, iterační průchod datovými kontejnery, výpis lokalizovaných textů, atd. JSP předpokládají těsnou spolupráci se servlety, které zajišťují zpracování požadavků a vkládaných dat a celkové řízení aplikace (přesměrování na výslednou JSP stránku).

Novější technologií postavenou na JSP jsou pak *JavaServer Faces* (JSF). Ty používají koncepčně zcela jiný přístup k tvorbě webového rozhraní, a to kompozici dokumentu z prefabrikovaných komponent schopných reagovat na události – v principu velmi podobně, jako je tomu u desktopových GUI vytvářených např. ve Swingu. Komponenty (reprezentované jako uživatelské značky v JSP) mají snadno definovatelné napojení na validátory vstupních dat a pomocí tzv. backing bean tříd (přístupných přes expression language výrazy) komunikují s aplikační logikou. Také navigace v JSF aplikaci je řešena na abstrakčně vyšší úrovni, a to deklarativně formou XML souboru popisujícího stavový model aplikace (výchozí stav = zobrazená stránka čili view, událost = výsledek jejího zpracování v backing bean objektech, přechod = cílové view).

2.4 Integrace

Poslední podstatnou skupinou technologií standardu Java Enterprise Edition jsou technologie komunikační a integrační. Jejich cílem je umožnit vzájemnou komunikaci resp. těsné propojení jednotlivých částí distribuované aplikace nebo aplikací samostatných.

Prvním důležitým předpokladem pro integraci je schopnost najít službu, kterou klient potřebuje. K tomu slouží ověřené *Java Naming and Directory Interface* (JNDI) tj. javovská inkarnace jmenných služeb. V nich je možné do jmenných kontextů registrovat pod logickými jmény (v URI syntaxi) libovolné fyzické Java objekty, a následně je vyhledávat a získat na ně referenci.

Základním prostředkem pro volně vázanou komunikaci je *JavaMail API*, které implementuje SMTP-based zasílání emailových zpráv. *Java Message Service* (JMS) je pak důležitou vysokoúrovňovou komunikační technologií. Slouží ke spolehlivé asynchronní či synchronní komunikaci mezi libovolnými Java objekty; typicky je využívána EJB komponentami. Komunikace může být $1:1$ i $1:N$ (v JMS terminologii tzv. domény), v druhém případě na funguje publish-subscribe mechanismus (posluchač se zaregistruje ke frontě zpráv definující tzv. topic, producent do ní generuje zprávy). V těle zpráv se mohou vyskytovat různé typy obsahu včetně obecného Java objektu. Důležitou vlastností JMS je nezávislost na konkrétní infrastruktuře pro přenos zpráv a tedy konkrétní implementaci JMS API – tu je třeba pouze zaregistrovat jako tzv. poskytovatele, klientská aplikace pak získává reference na příslušné agenty pomocí connection factory objektu.

Pro integraci s „legacy systémy“ slouží *J2EE Connector Architecture* (JCA). Koncepčně je podobná JDBC – klientská aplikace (typicky opět EJB komponenta) může získat spojení na cílový informační systém, spouštět jeho funkce a předávat či získávat z něj data – ovšem navíc dovoluje tzv. inbound komunikaci, tedy interakci iniciovanou informačním systémem a směrovanou na klientskou J2EE aplikaci (typicky pomocí zasílání zpráv). Pro získání spojení na IS slouží tzv. resource adapter což je implementace JCA API pro konkrétní informační systém (analogie s JDBC konektorem). Stará se o překlad volání, transakčních kontextů, autorizačních modelů, zajištění konkurenčního zpracování a také o cachování spojení (connection pooling). Klient JCA používá tzv. Common Client Interface, které umožňuje získat spojení na cílový IS, spouštět jeho funkčnost (interaction) a pracovat s daty vč. parametrů interakce (records, result sets).

Poslední, v dnešní době ale nejpopulárnější, integrační technologií podporovanou J2EE jsou webové služby. Vzhledem k formátu komunikace je tato část J2EE těsně vázána na API pro práci s XML, tj. JAXP a JAX-WS. Základním „nízkoúrovňovým“ web service rozhraním platformy je *SOAP with Attachments API for Java* (SAAJ[4]), které implementuje standardní Simple Object Access Protocol (SOAP [2]) pro komunikaci s webovou službou. Z pohledu autora webové služby je tato implementována jako POJO třída s anotacemi označujícími metody přístupné přes webovou službu. Klient pak přes JNDI nebo jiný registr získá tzv. koncový bod SAAJ (proxy objekt s Java rozhraním cílové webové služby), na kterém volá jeho metody ve standardní Java syntaxi. Na pozadí příslušný J2EE

[4]Neplést zkratku s JAAS uvedeným výše.

kontejner pomocí JAX-WS vytvoří SOAP zprávu, přes SAAJ ji zašle službě, a po získání odpovědi ji díky JAXB převede z XML formátu do POJO objektů.

Java API for XML Registries (JAXR) pak slouží k interakci s registry služeb, a to implementovaných podle obou v dnešní době používaných standardů: ebXML a UDDI.

3 Babylon frameworků a příloh

Zatímco předchozí text představil Java EE platformu jak je definována standardem, tato kapitola se zaobírá možnými dalšími přísadami a frameworky, které je možné použít. Protože jsou určeny pro vývoj velkých, enterprise aplikací ve vícevrstvé architektuře, často jsou také zahrnovány pod „značku“ J2EE.

3.1 Technologie standardizované JCP

První velkou kategorií enterprise technologií jsou „oficiální“ frameworky a API standardizované komunitním procesem a popsané v JSR dokumentech. Těch je v současné době přes 300, toto číslo ale obsahuje i ne-enterprise technologie a standardy posléze zahrnuté do oficiální Java EE platformy. Zde uvedený výčet je tedy víceméně náhodný, daný povědomím autora příspěvku a korelací s jeho tématem.

Z hlediska ukládání dat je užitečné *Java Content Repository API* (JCR) standardizované v JSR 170. Definuje přístupové rozhraní pro přístup do úložišť orientovaných na strukturovaná data a dokumenty, a potažmo jejich virtuální strukturu. Používá se pro sjednocení přístupu v systémech pro správu obsahu (content management, document management).

Zajímavé z hlediska správy aplikací jsou *Java Management Extensions* (JMX, součást J2SE) standardizované v JSR 3 a JSR 255. Umožňují monitorování, správu a konfiguraci libovolných zdrojů. Základem jsou tzv. MBean komponenty, což jsou JavaBeany (případně obohacené o specializovaná rozhraní pro dynamičtější funkčnost) které obalují zdroje. Agentová vrstva tyto komponenty může registrovat a volat na nich přístupné operace – nastavení parametrů zdrojů, periodické monitorování jejich stavu, apod. Kontrolní vrstva pak slouží k implementaci správcovských aplikací, které využívají služeb agentů.

Pro integraci enterprise aplikací na úrovni prezentační vrstvy je k dispozici *Portlet API* – JSR 168 a připravované JSR 286. Umožňuje vytvářet komponenty aplikací, portlety, které mohou zapouzdřit části logiky externích aplikací a prezentovat je ve webovém rozhraní portálu. Ten slouží z jedné strany jako kontejner pro portlety a z druhé jako website pro uživatele. Těm, díky integrovaným službám autorizace, single-sign on, lokalizace a dalším dává k dispozici

nástroj pro skládání uživatelsky konfigurovatelné funkčnosti (kompozitní aplikace) z portletových komponent.

3.2 Open source technologie

Kromě výše uvedených de-iure standardních technologií existuje nepřeberné množství[5] frameworků a knihoven dalších, ať již proprietárních (o kterých se zde nebudeme zmiňovat) či open source. Z nich uvedeme čtyři, které jsou z nějakého důvodu důležité.

Prvním velice rozšířeným frameworkem pro webové aplikace byly *Struts*, projekt Apache Foundation (http://struts.apache.org/). Vznikly v roce 2000 jako reakce na velmi primitivní podporu 3-vrstvé architektury v tehdejší Servlet-JSP technologii. Z dnešního pohledu jde již o mírně zastaralou technologii (i když stále zdokonalovanou), a je zajímavé podotknout že původní autor Struts, Craig McClanahan, byl hlavním architektem JavaServer Faces.

V jistém smyslu podobný osud potkal objektově-relační mapovací (ORM) framework *Hibernate* [1]. Ten se od svého vzniku cca v roce 2003 díky svojí jednoduchosti použití stal natolik populárním, že byl vážnou konkurencí EJB. To vedlo nakonec k přebudování perzistentní vrstvy J2EE a dnešní JPA a související specifikace jsou Hibernatem velmi ovlivněny.

Zcela samostatnou alternativou pro mnohé tvůrce Java aplikací je *Spring framework* (http://www.springframework.org/). Ten v sobě zahrnuje mnohé, ve své době velmi novátorské, principy pro zjednodušení tvorby a zpřehlednění architektury aplikací. Především jde tzv. dependency injection (taktéž zvané Inversion of Control) které vyřešení závislostí mezi komponentami vyjímá z jejich aplikačního kódu a deleguje je na kontejner. Tento princip, mimochodem velmi podobný základní teoretické myšlence výzkumu v oblasti softwarových komponent [5], je nyní převzat mnohými částmi J2EE platformy v implementaci pomocí anotací. Další technologií pocházející původně z čistě výzkumných kořenů, kterou Spring integroval v komerčně použitelné podobě, je aspektově orientované programování pro řešení transakcí, logování a dalších infrastrukturních věcí. Spring dále zahrnuje zjednodušující vrstvy pro práci s JDBC, webovými aplikacemi (zejména kontrolery a určení zobrazovacího view), podporu pro testování a další.

Poměrně nedávnou novinkou je *JBoss Seam* framework (http://www.jboss.com/products/seam). Ten si rychle získává popularitu, což je dáno především tím, že staví na již osvědčených a standardizovaných technologiích – konkrétně JSF a EJB/JPA – které velmi elegantně sešívá (odtud pravděpodobně jméno) do efektivního rámce pro tvorbu vícevrstvých aplikací s webovou prezentační vrstvou. Další důležitou složkou frameworku je těsná integrace tech-

[5] V diplomové práci [6] vedené autorem příspěvku je zmíněno 54 frameworků pro tvorbu webových aplikací, a tento výčet není konečný.

nologií pro definici business procesů (např. jBPM) a řízení aplikací na jejich základě.

4 Závěr

Jak je vidět z předchozího textu, pod nálepkou Java Enterprise Edition se skrývá velmi obsáhlá množina technologií od nízkoúrovňových po velmi komplexní. Za dobu existence této platformy prošly některé její části poměrně důkladným vývojem, který se dá stručně charakterizovat heslem „objevujeme jednoduchost“.

Zejména technologie EJB a webové vrstvy, původně silně strukturované a využívající množství samostatných tříd, rozhraní a XML souborů (tudíž s nepřehlednou architekturou a nutností silné podpory vývojových prostředí) se díky tlaku alternativních řešení a novým možnostem jazyka Java v JDK 5 přetvořila v poměrně flexibilní a pro programátora příjemnou technologii. Dá se předpokládat, že tento trend bude pokračovat i v nyní připravované verzi 6 platformy jejíž vydání je plánováno na rok 2008.

Literatura

[1] *Hibernate – Relational Persistence for Java and .NET.* http://www.hibernate.org/

[2] *Simple Object Access Protocol (SOAP).* W3C Recommendation, Version 1.2, World Wide Web Consortium, 2007.

[3] Sun Microsystems. *Java Platform, Enterprise Edition, version 5.* 2006. http://java.sun.com/javaee/

[4] Byous, J. *Java Technology: The Early Years.* Available on Sun Developer Network, at http://java.sun.com/features/1998/05/birthday.html

[5] Szyperski, C. *Component Software, Second Edition.* ACM Press, Addison-Wesley, 2002.

[6] Kubátová, M. *Aplikační rámce pro webové aplikace.* Plzeň : Západočeská univerzita v Plzni, 2007.

Spring a AOP

Martin Čížek

E-mail: martin@cizek.com

Abstrakt

Na vývoj moderních aplikací jsou v současnosti kladeny poměrně velké nároky. Od jejich designu se očekává oddělení a nezávislost jejich vrstev, flexibilita a možnost snadné rekonfigurace, loose coupling, nezávislost na technologii, možnost výměny klíčových částí na úrovni konfigurace, prototypování, implementace aspektů bez zásahu do business logiky, automatická testovatelnost komponent apod. Spring Framework je open source software, který se snaží pomoci s řešením těchto problémů. Je výjimečný v kombinaci obecnosti, neinvazivnosti a svobody, kterou vývojářům nabízí, při současném poskytnutí dobře dokumentovaných a jednoduchých řešení pro běžné úlohy. Přednáška se dále zabývá tzv. aspect oriented programming – technikami, při nichž se bez zbytečných zásahů do kódu snažíme řešit problémy, pro něž běžný objektový model nenabízí elegantní řešení. Zástupci těchto problémů jsou vynucení bezpečnosti, automatická demarkace transakcí či zaznamenávání přístupů k službám.

1 Úvod

Příspěvek si klade za cíl poskytnout úvod do moderního přístupu k návrhu a tvorbě aplikací s využitím znovupoužitelných komponent, aspektů, loose coupling a unit testování. Takový přístup není pouze interní záležitostí vývojářů, promítá se i do projektového plánování, životního cyklu projektu a v důsledku jej může pocítit i obchodní oddělení a zákazník, zejména díky prototypování pomocí tzv. „mock“ komponent.

Framework Spring pomáhá realizovat tyto přístupy. Poměrně výjimečný je svou univerzálností a neinvazivností, nenutí vývojáře používat žádnou konkrétní technologii ani implementaci speciálních rozhraní tak, jak to známe z J2EE.

Příspěvek nemá ambice být vyčerpávající referencí a konkurovat tisícistránkovým publikacím. Zaměřuje se na letmé seznámení se dvěma ze základních přístupů ve frameworku Spring – Dependency Injection a Aspect Oreinted Programming.

2 Konfigurace objektů – stavíme aplikaci

Každý, kdo někdy navrhoval či vyvíjel nějakou aplikaci, stál před problémem, jak návrh provést. Naivní přístup spočívá v prostém přepsání logiky operací s **konkrétními** entitami do zdrojového kódu. Ukažme si to na příkladu triviální aplikace „Elektronický bavič“. Zadáním je sestavit aplikaci, která bude své uživatele bavit veselými příhodami.

2.1 Jeden způsob a dost

Výpis v této části ukazuje program, který svého uživatele pobaví anekdotou pro lidové vrstvy včetně mládeže a dětí z Cimrmanovy hry Lijavec.

```
package com.cizek.europen.spring;

public class Entertainer {
        public static void main(String[] args) {
            System.out.println(
                    "V~Schönbrunnu, v~Schönbrunnu\n" +
                    "střílel císař na srnu.\n" +
                    "Minul se však cíle,\n" +
                    "zapomněl si brýle.");
        }
}
```

Program zjevně plní svůj účel. Ovšem jen do doby, kdy postačuje anekdota zapsaná ve zdrojovém kódu a jejím konzumentem je uživatel čtoucí standardní výstup programu. Změní-li se kterákoliv z těchto entit, je nutné program celý přepsat.

2.2 Decoupling aneb povolme si opasek

První příklad bere v sobě obsaženou anekdotu a vypisuje jí na standardní výstup. Zamyslíme-li se nad zadáním obecněji, dojdeme k závěru, že úkolem naší aplikace je získat anekdotu ze *zdroje anekdot* a předat ji *konzumentu anekdot*. To vystihuje podstatu programu, jež je nezávislá na konkrétním zdroji anekdot i na konkrétním cíli anekdot.

Vytvoříme tedy rozhraní deklarující obecný zdroj anekdot – `JokeSource` a rozhraní deklarující obecný cíl anekdot – `JokeSink`. Proces přechodu od vazeb na konkrétní entity (těsné vazby, tight coupling) k vazbám na jejich abstrakce (volné vazby, loose coupling) se nazývá *decoupling*. Implementaci pomocí volných vazeb ukazují výpisy níže. Vazby však stále nejsou tak volné, jak by bylo optimální.

```
package com.cizek.europen.spring;

public class EntertainerDecoupled {
    public static void main(String[] args) {
        JokeSource jokeSource = new HardcodedJokeSource();
        JokeSink jokeSink = new StdOutJokeSink();

        jokeSource.setJokeSink(jokeSink);
        jokeSource.pushJoke();
    }
}
```

```
package com.cizek.europen.spring;

public class HardcodedJokeSource implements JokeSource {
    private JokeSink jokeSink;
    public void pushJoke() {
        jokeSink.renderJoke("H2SO5");
    }
    public void setJokeSink(JokeSink jokeSink) {
        this.jokeSink = jokeSink;
    }
}
```

```
package com.cizek.europen.spring;
public class StdOutJokeSink implements JokeSink {
    public void renderJoke(String joke) {
        System.out.println(joke);
    }
}
```

Kromě nesporné výhody možnosti výměny konkrétních entit, s nimiž program pracuje, přináší přístup neocenitelnou výhodu při procesu samotného vývoje. Jistě si lze představit, že zdrojem anekdot bude mnohem komplikovanější třída, která bude příhody extrahovat například z oblíbeného zábavního serveru. Stejně tak cílem anekdot může být zobrazení v grafickém okně, uložení do databáze či rozeslání do mailing listu. Při vývoji aplikace chceme, aby vývojáři vytvářející logiku programu nemuseli čekat na dokončení implementace zdroje a cíle zpráv a stejně tak aby vývojáři zdroje i cíle zpráv nemuseli čekat na výsledek práce ostatních. Loose coupling umožňuje vytvořit nejjedonduší možnou implementaci příslušných rozhraní a všechny týmy mohou pracovat paralelně na funkcionalitách, jež na sobě jinak závisí. Implementace rozhraní může být i „falešná“ (mock). Taková implementace pak provádí požadovanou funkčnost v omezeném rozsahu; typicky to znamená, že pracuje nad pevnou množinou dat – tak jako náš `HardcodedJokeSource` nebo svá volání pouze zaznamenává – tak jako náš `StdOutJokeSink`.

2.3 Nebuďme příliš konkrétní

Příklad v předchozí části má stále neduh „zadrátování“ konkrétní implementace v některých místech logiky. S rozsahem aplikace by počet výskytů konkrétní implementace pochopitelně rostl. Řešením tohoto nešvaru je centralizace výskytu názvu konkrétní implementace. Toho lze dosáhnout použitím návrhového vzoru Factory. Realizaci Factory můžeme vylepšit tím, že budeme konkrétní implementace získávat z externího souboru.

Realizace `EntertainerFactory` pak může být následující.

```
package com.cizek.europen.spring;

import java.io.InputStream;
import java.util.Properties;

public class EntertainerFactory {
    private static EntertainerFactory entertainerFactoryInstance;
    private Properties properties;
    private JokeSource jokeSource;
    private JokeSink jokeSink;

    static {
        entertainerFactoryInstance = new EntertainerFactory();
    }

    protected EntertainerFactory() {
        properties = new Properties();
        try {
            ClassLoader cl = Thread.currentThread().getContextClassLoader();
            InputStream is = cl.getResourceAsStream("entertainer.properties");
            properties.load(is);
            is.close();
            String jokeSourceClass = properties.getProperty("jokeSource.class");
            String jokeSinkClass = properties.getProperty("jokeSink.class");
            jokeSource = (JokeSource)
                         cl.loadClass(jokeSourceClass).newInstance();
            jokeSink = (JokeSink) cl.loadClass(jokeSinkClass).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static EntertainerFactory getInstance() {
        return entertainerFactoryInstance;
    }

    public JokeSource createJokeSource() {
        return jokeSource;
    }

    public JokeSink createJokeSink() {
        return jokeSink;
```

```
    }
}
```

Vzorový konfigurační soubor `entertainer.properties`:

```
jokeSource.class=com.cizek.europen.spring.HardcodedJokeSource
jokeSink.class=com.cizek.europen.spring.StdOutJokeSink
```

Hlavní program pracující s `EntertainerFactory`:

```
package com.cizek.europen.spring;

public class EntertainerConfigurable {
    public static void main(String[] args) {
        EntertainerFactory factory = EntertainerFactory.getInstance();
        JokeSource jokeSource = factory.createJokeSource();
        JokeSink jokeSink = factory.createJokeSink();

        jokeSource.setJokeSink(jokeSink);
        jokeSource.pushJoke();
    }
}
```

2.4 Kdo, kde a s kým

Přechozí příklad stále není dokonalý. Konkrétní implementace objektů jsou externalizovány, ale konfigurace a sestavení objektů je stále zadrátováno do kódu a vyskytuje se při každém použití příslušného objektu. Naším cílem je zcela oddělit definici komponent tak, aby obchodní logika (business logic) nebyla svázána ani s konkrétními implementacemi (toho jsme již dosáhli), ani nemusela znát vzájemné vazby komponent. Požadavek na znovupoužitelnost hlavních i dílčích komponent je samozřejmý.

Pojďme se podívat, jak by taková definice komponent a jejich vazeb mohla vypadat.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <bean id="jokeSource" class="com.cizek.europen.spring.HardcodedJokeSource">
        <property name="jokeSink" ref="jokeSink" />
    </bean>
    <bean id="jokeSink" class="com.cizek.europen.spring.StdOutJokeSink" />
</beans>
```

Aniž bychom formálně popisovali strukturu souboru, je zřejmé, že jsou definovány dvě komponenty `jokeSource` a `jokeSink`. Loose coupling ve vazbách lze ilustrovat tvrzením, že **implementace** `HardcodedJokeSource` má referenci na rozhraní `JokeSink`. Pozn.: fakt, že metoda `setJokeSink()` je i součástí rozhraní

`JokeSource`, berme nyní jako shodu náhod. Nic nevylučuje, aby jiné implementace `JokeSource` a `JokeSink` měly další reference, aniž by se to jakkoliv promítlo do jejich rozhraní `JokeSource` a `JokeSink`.

Asi již není velkým tajemstvím, že poslední výpis je současně definicí komponent pro framework Spring. Ukažme si tedy hlavní program, jímž lze aplikaci otestovat.

```
package com.cizek.europen.spring;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class EntertainerSpringDi {
    public static void main(String[] args) {
        BeanFactory bf = new ClassPathXmlApplicationContext("entertainer.xml");
        JokeSource jokeSource = (JokeSource) bf.getBean("jokeSource");
        jokeSource.pushJoke();
    }
}
```

2.5 Inversion of Control & Dependency Injection

Zbývá si povědět, co vlastně znamenají „buzzwords“ v názvu této části. Oba přístupy jsme již poznali:

Inversion of Control (IoC) je přístup, kdy kód, který využívá určité komponenty, není sám zodpovědný za řízení jejich životního cyklu. Konkrétně v našem příkladu to není ani hlavní program, ani `JokeSource`, kdo se stará o vytvoření a nakonfigurování instance `JokeSink`. Úkol je přenechán tzv. IoC kontejneru, v našem případě frameworku Spring. Řízení je prováděno nezávisle na kódu používajícím řízené objekty, odtud Inversion of Control. Dlužno podotknout, že někdy je IoC považováno za totéž jako Dependency Injection. Na základě této definice však lze do IoC zahrnout další přístupy jako Dependency Lookup, známý z jiných kontejnerů (EJB 2).

Dependency Injection (DI) je přístup, kdy jsou závislosti předávány komponentám kontejnerem. Závislé objekty si tedy reference neobstarávají samy, nýbrž je dostávají od spravujícího kontejneru. Ve Springu je Dependency Injection realizována buď pomocí setterů nebo parametrů konstruktoru.

3 Aspect Oriented Programming

Aspect Oriented Programming (AOP) je přístup adresující určité nedostatky při vývoji reálných aplikací. Většina vývojářů se s těmito nedostatky setká a považuje je za nutné zlo. Pojďme si to ukázat na příkladu.

3.1 Jak neprodat obchodní logiku

Kód na následujícím výpisu je modifikovaný příklad metody pro převod částky z účtu na účet převzatý z [wikipedia-en-aop].

```
void transfer(Account account, long amount) throws Exception {

  if (!getCurrentUser().isInRole(ROLE_TRANSFER)) {
    throw new SecurityException(ROLE_TANSFER);
  }

  Transaction tx = database.newTransaction();
  try {
     if (fromAccount.getBalance() < amount) {
       throw new InsufficientFundsException();
     }
     account.withdraw(amount);
     toAccount.deposit(amount);

     tx.commit();
     systemLog.logOperation(OP_TRANSFER, fromAccount, toAccount, amount);
  }
  catch(Exception e) {
     tx.rollback();
     throw e;
  }
}
```

Jde o poměrně běžný příklad, na první pohled ničím zvláštní. Při dalším vývoji pomyslné bankovní aplikace zjistíme, že některé aspekty uvedené metody se ve stejné nebo obdobné formě opakují. Jsou to části, které do obchodní logiky (business logic) metody `transfer()` logicky nepatří.

Tyto části procházejí celou aplikací. Jsou-li v aplikačím kódu takto zadrátovány, zvyšuje se tak těsnost vazeb (tight coupling) a snižuje flexibilita celého systému. To může vést ke ztrátě konkurenční výhody, proto byla tato sekce nazvána „jak neprodat“.

3.2 Neformální formalizace AOP

Počítačové programy se obecně sestávají z tzv. koncernů (concerns). Koncernem rozumíme jakoukoliv oblast zájmu či záměření v programu. Narozdíl od vlastností (features), které určují chování programu, se koncerny týkají implementace programu.

Oddělení (též zapouzdření) koncernů (Separation of Concerns, SoC) je proces dělení programu na různé koncerny, které se ve funkcinalitě překrývají co nejméně. Všechny programové metodologie pomáhají vývojářům v procesu separace koncernů. Objektové a procedurální jazyky oddělují koncerny do tříd, metod, procedur, funkcí a balíků.

Některé koncerny je však těžké zapouzdřit v rámci existujících možností jazyka. Jako obvyklý příklad se uvádí logování, neboť způsob logování se týká všech částí systému. Logování *prořezává* všechny logované třídy a metody, jde o tzv. *průřezový koncern*.

Cílem AOP je zapouzdřit průřezové koncerny pomocí konstruktů zvaných *aspekty*. Před uvedením příkladů je vhodné vysvětlit základní pojmy AOP.

Přípojný bod (joinpoint) je dobře definované místo v běhu programu, typickými příklady jsou volání metody, inicializace třídy nebo instancování objektu.

Advice je kód, který implementuje dodatečné chování programu a je vykonán v nějakém přípojném bodě. Advices se liší tím, zda jsou vykonány před přípojným bodem, po něm nebo případně okolo.

Pointcut je sada přípojných bodů, která definuje, kdy má být advice vykonána. Jestliže typický přípojný bod je volání metody, typický pointcut je volání všech metod nějaké třídy.

Aspekt je kombinací advice a pointcuts. Tím je definována logika, která bude začleněna v aplikaci i místo, kde bude začleněna.

Proplétání (weaving) je proces vkládání aspektů na vhdoná místa kódu aplikace. Proplétání může být prováděno buď v okamžiku kompilace (statické AOP) nebo v době běhu (dynamické AOP).

3.3 AOP v příkladech – log aktivit

Vynechme umělé příklady základního použití AOP a přejděme rovnou k reálnému užití s podporou frameworku Spring. Následující příklad ukazuje využití AOP ve fiktivní bankovní aplikaci. Poznamenejme, že uvedená konfigurace je aditivní, tedy nejsou potřeba žádné změny již existující konfigurace komponent aplikace.

```
<bean id="transactionLogger"
      class="com.cizek.spring.aop.log.TransactionLoggerImpl">
    <!-- Properties -->
</bean>

<bean id="bankTransactionLogger" class=
         "org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="beanNames">
        <list>
            <value>bankService</value>
        </list>
    </property>
    <property name="interceptorNames">
```

```
        <list>
            <value>bankTransferLogAdvisor</value>
            <value>bankWithdrawLogAdvisor</value>
        </list>
    </property>
</bean>

<bean id="baseTransactionLogInterceptor"
      class="com.cizek.spring.aop.log.TransactionLogInterceptor" abstract="true">
    <property name="transactionLogger" ref="transactionLogger" />
</bean>

<bean id="bankTransferLogAdvisor"
      class="org.springframework.aop.support.DefaultPointcutAdvisor">
    <property name="advice">
        <bean parent="baseTransactionLogInterceptor">
            <property name="transactionType" value="BANK_TRANSFER" />
        </bean>
    </property>
    <property name="pointcut">
        <bean class="org.springframework.aop.support.NameMatchMethodPointcut">
            <property name="mappedName">
                <value>transfer</value>
                <value>transferWithCurrencyConversion</value>
            </property>
        </bean>
    </property>
</bean>
<!-- bankWithdrawLogAdvisor vynechán -->
```

`BeanNameAutoProxyCreator` zaručuje, že všechny vyjmenované komponenty budou při použití vždy zabaleny do tzv. proxies, s jejichž pomocí jsou aspekty vytvořeny. Každé volání komponenty je pak předmětem vyhodnocení pomocí tzv. advisoru, který rozhoduje o použití advice. Např. `bankTransferLogAdvisor` aplikuje advice definovanou komponentou ve vlastnosti `advice` pro metody s názvem `transfer` a `transferWithCurrencyConversion`. Jinými slovy, kdykoliv je volána jedna z těchto metod v komponentě `bankService`, je na volání aplikována advice, jež zaloguje hodnoty, s nimiž byla služba volána a jejich výsledek, ať už skončí úspěchem nebo výjimkou.

Příklad implementace třídy `TransactionLogInterceptor` následuje.

```
package com.cizek.spring.aop.log;

public class TransactionLogInterceptor implements MethodInterceptor {

    private TransactionLogger transactionLogger;
    private TransactionType transactionType;

    /**
     * @see MethodInterceptor#invoke(MethodInvocation)
     */
```

```
    public Object invoke(MethodInvocation invocation) throws Throwable {
        Object result = null;
        TransactionLogEntry entry = new TransactionLogEntry();
        boolean save = false;
        try {
            entry.setMethodName(invocation.getMethod());
            entry.setArguments(invocation.getArguments());
            result = invocation.proceed();
            entry.setSuccess(true);
            entry.setResult(result);
        } catch (Throwable t) {
            entry.setSuccess(false);
            entry.setException(t);
            throw t;
        } finally {
            transactionLogger.log(entry);
        }
        return result;
    }

    //~ Dependency Injection
    @Required
    public void setTransactionLogger(TransactionLogger transactionLogger) {
        this.transactionLogger = transactionLogger;
    }

    @Required
    public void setTransactionType(String transactionType) {
        this.transactionType = TransactionTypeEnum.valueOf(transactionType);
    }

}
```

3.4 AOP v příkladech – automatická demarkace transakcí

Spring 2 poskytuje syntaktické koření, s nímž lze konfiguraci ještě zjednodušit. Následuje poměrně typický příklad automatické demarkace transakcí ve vrstvě služeb aplikace. Na příkladu je též vidět použití jazyka pro definici pointcuts převzatého z AspectJ [wikipedia-en-aspectj], což je novinka Spring Framework 2.0.

```
<aop:config>
    <aop:advisor id="managerTx" advice-ref="txAdvice" pointcut=
         "execution(* com.cizek.europen.spring.aop.srv.*.*(..))" order="1"/>
</aop:config>

<tx:advice id="txAdvice">
    <tx:attributes>
        <tx:method name="get*" read-only="true" rollback-for=
            "com.cizek.europen.spring.aop.srv.RollbackBusinessException"/>
        <tx:method name="*" rollback-for=
            "com.cizek.europen.spring.aop.srv.RollbackBusinessException"/>
```

```
    </tx:attributes>
</tx:advice>
```

Přidáme-li ke konfiguraci aplikace uvedenou část, budou všechna volání metod tříd v balíku `com.cizek.europen.spring.aop.srv` zabalena do transakce. Transakce skončí rollbackem, pokud je vrhnuta jakákoliv „unchecked“ výjimka (tedy potomek `RuntimeException` nebo potomek `Throwable`, který není potomkem `Exception`). Transakce též skončí rollbackem při vržení vyjmenované výjimky `RollbackBusinessException` nebo jejího potomka. V případě úspěšného volání nebo vržení jiné „checked“ výjimky, končí transakce commitem. Za povšimnutí stojí větší flexibilita nastavení reakce na výjimky oproti EJB.

Dále volání metod začínajících `get` vytvářejí read-only transakce. Odpadá tak nutnost všechna tato pravidla implementovat znovu a znovu v každé metodě.

Jako zdroj řízení transakcí lze ve Springu použít mimo jiné JDBC DataSource, Hibernate Transaction Manager a JPA. Konkrétní použitý manažer transakcí nemá vliv na konfiguraci jejich demarkace.

3.5 Nevýhody AOP

AOP má pochopitelně i své nevýhody, mezi ty hlavní lze zařadit následující.

- Velká závislost na pravidlech tvorby aplikace a jmenných konvencích. Nový vývojář musí jmenné konvence nastudovat a držet se jich. Tento fakt ale přináší i užitek, neboť dává reálný argument důležitosti konvencí. Dobře známý z jiných frameworků (Ruby on Rails) je princip „convention over configuration“. Spring také umožňuje jej praktikovat, ačkoliv nevynucuje konkrétní konvence.

- Potíže při debugování. Při debugování kódu volání procházejí proxy, interceptory, kódy advices apod. Přestože výkonostní dopad na běh nebývá výrazný, pro debugování to může být nepříjemné. V praxi bývá výhodné nastavit breakpointy do zdroje a (předpokládaného) cíle volání a aspekty nekrokovat, není-li to účelem.

4 Mýty o Springu

4.1 Mýtus: Spring není Enterprise

Přejděme fakt, že argument o Enterprise řešeních často slýcháme od firem, které se snaží prodat zbytečně drahé produkty. Na adrese [spring-usage] nalezneme seznam některých projektů využívajících Spring Framework. Jmenujme několik projektů, kde je Spring používán v mission-critical nasazení.

- DekaBank – mission-critical obchodovací aplikace,
- French Tax Authority – migrace z EJB 2, Spring slouží 34 milionům plátců daně,
- The European Patent Office – mission-critical aplikace poskytující přístup ke správě duševního vlastnictví pro 25 zemí,
- US National Healthcare Provider System – portál poskutující informace o poplatcích 39 pojišťovacím společnostem ve Spojených státech,
- AAA API – mission critical část projektu AAA Portál České správy sociálního zabezpečení.

4.2 Mýtus: Spring nepoužívají velké firmy

Mezi společnostmi používající Spring Framework jsou Oracle, BEA, eBAY, CERN, Confluence.

4.3 Mýtus: Spring nelze škálovat a nasazovat při potřebě vysoké dostupnosti

- Škálování požadavků na síťové vrstvě je nejčastější řešení (content switche),
- Open Terracotta [terracotta],
- Tangosol Coherence DataGrid [tangosol],
- EJB – Spring lze použít dohromady s EJB s výhodami obou řešení!
- Další řešení podporující clustering a vysokou dostupnost jsou použitelná se Springem
  - Quartz scheduler v clusterovém módu,
  - JBoss TreeCache,
  - integrace s JPA.

5 Shrnutí

Příspěvek se pokusil naladit čtenáře na některé best practices při tvorbě aplikací a ukázat, jak v tomto může pomoci framework Spring. Ukázky byly v jazyce Java, ačkoliv Spring exituje i pro .NET Framework. Pro další studium lze doporučit tutoriály na webu a knihy uvedené v referencích. Zajímavým čtením jsou i informace o motivaci vzniku a historii frameworku, které se do příspěvku nevešly.

Literatura

[spring-web] *Spring Framework.* http://www.springframework.org/

[pro-spring] Harrop, R., Machacek, J. *Pro Spring.* APress, 2005.

[spring-mvc] Seth Ladd with Darren Davison, Steven Devijver and Colin Yates; Edited by Rob Harrop and Keith Donald. *Expert Spring MVC and Web Flow.* APress, 2006.

[wikipedia-en-aop] WikiPedia. *Aspect-oriented programming.* http://en.wikipedia.org/wiki/Aspect-oriented_programming

[wikipedia-en-aspectj] WikiPedia *AspectJ* http://en.wikipedia.org/wiki/AspectJ

[czjug-spring-2] Pichlík, R.: *Prezentace Spring 2.0.* http://http://www.java.cz/detail.do?articleId=3804

[spring-usage] *Spring Real-World Usage.* http://www.springhub.com/component/option,com_weblinks/catid,36/Itemid,56/

[terracotta] Open Terracotta. http://www.terracotta.org/

[tangosol] Tangosol Coherence DataGrid. http://www.tangosol.com/

Od PL/SQL k portletům, aneb třívrstvý STAG

Josef Krupička

E-mail: aragorn@civ.zcu.cz

Úvod

Systém IS/STAG je informační systém pro kompletní evidenci studijní agendy vysoké školy nebo univerzity. Místem vzniku a vývoje systému IS/STAG je Centrum informatizace a výpočetní techniky, Středisko informačního systému na Západočeské univerzitě v Plzni. Aplikace IS/STAG je vyvíjena nad databázovým systémem Oracle a kromě ZČU ji momentálně používá dvanáct dalších vysokých škol v České republice. Ke všem funkcím systému lze přistupovat přes tlustého klienta naprogramovaného v Oracle Forms. K nejčastěji používaným funkcím je umožněn přístup přes Internet pomocí webového prohlížeče.

Toto rozhraní začalo vznikat v roce 1998, což mělo vliv na volbu použitých technologií. V té době bylo PHP neověřenou technologií, Java byla na počátku svého vývoje a technologie Microsoftu nejsou u nás vítány. Výsledné řešení, které se používá dodnes vypadá následovně. Webový server Apache dostane požadavek, který je obsloužen modulem *mod_perl*. Ten podle požadovaného URL určí název uložené procedury a její parametry. Následně otevře spojení do databáze a spustí požadovanou proceduru, která v sobě obsahuje SQL dotazy, volání jiných uložených procedur a generování samotného HTML. Všechna zátěž je tak soustředěna na databázový server. Místo, aby sloužil pouze jako úložiště dat a poskytovatel aplikační logiky, vytváří navíc webové rozhraní. Databázový server tak supluje funkci aplikačního serveru. Pro většinu poskytovaných funkcí je toto řešení dostačující, ne však pro aplikaci *Předzápis*.

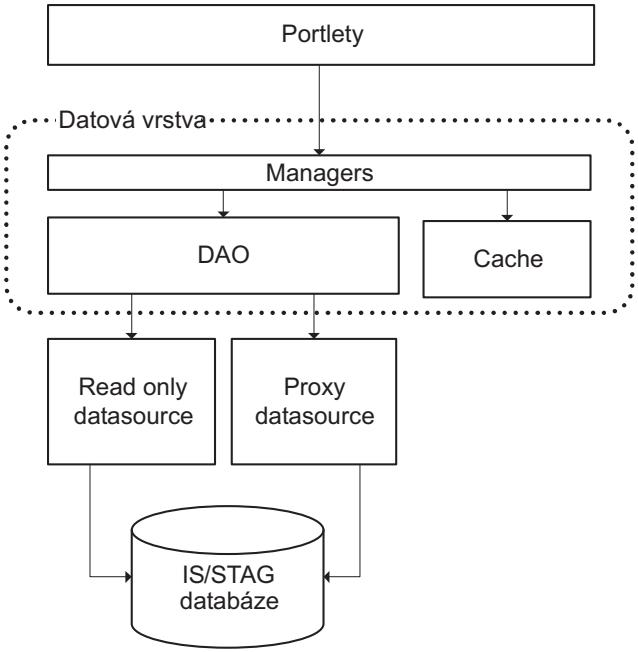
Jednou z nejdůležitějších aplikací IS/STAG je *Předzápis*, který umožňuje studentům zapsat si jaké předměty chtějí studovat příští rok. Největší zájem studentů je samozřejmě během prvních dnů, kdy se každý snaží vytvořit si co nejlepší rozvrh. Počet simultánně pracujících uživatelů v těchto dnech několikanásobně překračuje normální stav. Databáze je pod obrovskou zátěží a není výjimkou, že se databázové servery musejí několikrát restartovat. Další informační systémy, které jsou na těchto databázích závislé, jsou po tyto dny také

v nestabilním stavu. Nepomohlo nasazení clustrování Oracle databází ani omezování počtu aktuálně přihlášených studentů v závislosti na zatížení databáze.

Tento stav byl neudržitelný a bylo rozhodnuto o vytvoření nové verze webového rozhraní. Od roku 2003 na Západočeské univerzitě probíhá nasazování portálu od firmy IBM. Cílovou platformou se tedy stala J2EE.

Architektura aplikační vrstvy

Každá třívrstvá architektura se skládá z klienta, aplikačního serveru a datového zdroje. Klientem je v našem případě webový prohlížeč, roli aplikačního serveru zastupuje libovolný J2EE server s portletovou podporou a datovým zdrojem je databáze Oracle. U typických třívrstvých aplikací je většina bussiness (aplikační) logiky umístěna na aplikačním serveru. IS/STAG má však většinu logiky obsaženou přímo v databázi (uložené procedury, triggery, přístupová práva, ...). Na aplikační server tedy moc složité logiky nezbývá a v podstatě hlavně vytváří webové rozhraní. Obrázek 1 zobrazuje architekturu aplikační vrstvy.



Obr. 1 Architektura aplikační vrstvy

JDBC datasource

JDBC datasource je komponenta, která poskytuje ostatním komponentám otevřené připojení k databázi. Datasource se stará o vytváření nových spojení, connection pooling těchto spojení, nastavuje cache používaných SQL dotazů, odstraňuje neplatná spojení, obnovuje spojení po výpadku databáze, atd. V našem případě používáme dva datasourcy:

- První je používán pro většinu čtecích operací. Do databáze se připojuje pod uživatelem, který má přístup k většině public objektů v databázi.
- Druhý se používá při volání všech zápisových operací a některých čtecích operacích. Do databáze se připojuje pod databázovou identitou uživatele, pro kterého se daná operace provádí (tedy přímo pro uživatele přihlášeného do portálu). Důvodem je aplikační logika obsažená v IS/STAG, která má na úrovni databáze nastaveny tzv. „policies“ – nejnižší vrstvu bezpečnostní politiky, omezení na úrovni řádků databázové tabulky. Použitím nějakého vysoce privilegovaného uživatele (použitý např. pro první datasource) by se obešly autorizační mechanismy databáze, se kterými počítá např. logika obsažená v uložených procedurách. A tento přístup by byl hlavně velice nebezpečný, takovým způsobem se aplikace nejčastěji napadají.

Do portálu se uživatelé hlásí pod jinou identitou než je jejich databázová identita. Portletová aplikace tak nezná údaje nutné k vytvoření připojení do databáze pro přihlášeného uživatele. Oracle poskytuje mechanismus tzv. *proxy spojení*, který umožňuje přihlásit se do databáze pod uživatelem, aniž by aplikace musela znát jeho heslo. Podrobný popis této funkce naleznete např. v [1].

Implementace datasource, který by poskytoval datové vrstvě otevřené proxy spojení, není triviální. K vytvoření proxy spojení je totiž potřeba mít k dispozici nativní spojení (OracleConnection) do databáze, nad kterým se vytvoří nové spojení. Všechny datasource komponenty obalují nativní spojení svou třídou. Pokud chceme použít datasource nadefinovaný přímo v aplikačním serveru, musíme vytvořit mechanismus přenositelný mezi všemi aplikačními servery (v každém aplikačním serveru se nativní spojení získává jiným způsobem). Náš kolega napsal J2EE Resource Adapter [2], který poskytoval tuto funkcionalitu. Jeho nevýhodou bylo pomalé vytváření proxy spojení a z toho plynoucí úzké hrdlo, které se projevilo při zátěžovém testování. Stejně dopadl i pokus použít open source JDBC datasource (DBCP). Získání nativního spojení a vytvoření proxy spojení se opět stalo úzkým hrdlem. Efektivním řešením je využití nativního datasource, který je obsažen přímo v Oracle JDBC ovladači. Vzhledem k tomu, že STAG je zcela závislý na databázi Oracle (nelze provozovat STAG na jiném RDBMS), můžeme si toto řešení dovolit. Propustnost aplikace se s nasazením této komponenty řádově zvětšila.

Nevýhodou používání proxy spojení je fakt, že se pro každé vytvoření otevře nové spojení do databáze. Náš původní předpoklad byl, že se použije již existující spojení, nad kterým se toto spojení vytváří. Při otevření nového spojení se totiž spouští nový process, což je velmi drahá operace. Na úrovni Oracle se dá provést optimalizace v podobě tzv. shared serverů, které slouží jako connection pool. V současné době zvažujeme využití úplně jiného řešení, které by se obešlo bez proxy spojení, ale nastavovalo by specifické uživatelské informace přes tzv. *session context* přímo v databázi. Od tohoto řešení si slibujeme zvětšení propustnosti systému a zmenšení zátěže databázových serverů.

Datová vrstva

Datová vrstva slouží k práci s úložištěm dat, kterým může být např. relační databáze nebo souborový systém. Komponenty v této vrstvě obsahují logiku pro zacházení s vybraným datovým zdrojem. V našem případě je tímto zdrojem relační databáze a tak bylo k implementaci této vrstvy použito klasické JDBC API a návrhový vzor Data Access Object (DAO [3]).

Typické volání jednoho SQL příkazu v JDBC se skládá z několika kroků (získání databázového spojení, nastavení SQL dotazu a jeho parametrů, načtení výsledků dotazu a uzavření všech použitých objektů). Všechny tyto kroky je nutné opakovat pro každý SQL dotaz. Když se k tomu připočte fakt, že SQL dotazy jsou napsány přímo ve zdrojovém kódu, stane se výsledný kód značně nepřehledným a těžko udržovatelným.

V první verzi databázové vrstvy bylo volání JDBC příkazů obaleno bezestavovými EJB komponentami. Pro každý důležitý objekt (student, učitel, předmět, ...) existovala jedna EJB komponenta, která měla výše zmíněné nedostatky: velké množství redundantního kódu a často obrovské SQL dotazy vložené přímo ve zdrojovém kódu. K tomu navíc ještě obecně známé problémy plynoucí z použití EJB verze 2.1 (pomalý vývoj, složité testování, malá přidaná hodnota). V té době sice již skončila práce na novém standardu EJB 3.0, který měl značně usnadnit vývoj, ale pro nás přechod na novou verzi nepřicházel v úvahu. Aplikační server podporoval pouze starou verzi J2EE 1.3 a ani nejnovější verze aplikačního serveru IBM verze 6.1 nemá finální podporu tohoto standardu.

K implementaci nové verze datové vrstvy byl použit framework Spring [4], který je mezi vývojáři J2EE aplikací velmi oblíben, protože značně usnadňuje a urychluje vývoj aplikací. Popis všech vlastností Spring frameworku je mimo rozsah tohoto příspěvku. Popíši zde pouze přínosy přechodu na tento framework:

- Bezproblémové nasazení datové vrstvy na jakýkoliv aplikační server – ve skutečnosti je možné datovou vrstvu jednoduše použít v jakékoliv Java aplikaci (konzolový program, GUI, ...).

- Snadné vytváření automatických testů této vrstvy a z toho plynoucí zvýšení rychlosti vývoje – EJB 2.1 šlo sice také automaticky testovat, ale konfigurace testů byla velmi složitá a testy trvaly dlouho, čímž je porušena jedna ze základních podmínek pro úspěšné využívání automatického testování.

- Pročištění kódu pracujícího s JDBC – spring obsahuje třídy, které umožňují odstranit všechen redundantní kód, zabývající se správou databázových spojení, nastavováním parametrů dotazu a získáváním výsledků. Místo 20 řádků kódu se volání jednoho SQL dotazu omezilo na jeden řádek kódu. Napsali jsme také jednoduchou knihovnu, která umožňuje přesun SQL dotazů ze zdrojového kódu Javy do externích souborů, jejichž obsah se načte při inicializaci datové vrstvy do speciálních proměnných. Tyto dotazy lze pak jednoduše vyvíjet a ladit v oblíbeném databázovém nástroji (Toad, SQL developer, ...).

- Velmi jednoduchá výměna implementace jedné komponenty (nevyhovující datasource, cachovací knihovna, atd.) za jinou implementaci – ta spočívá v drobné úpravě konfiguračního souboru (XML formát) Spring frameworku. Lze tak snadno reagovat na různé případy nasazení naší aplikace. Pokud má cílová univerzita IBM Websphere Portal, nastavíme komponenty, které nejlépe vyhovují tomuto prostředí. Používají-li jiný produkt, ve kterém tyto komponenty nejsou k dispozici, nastavíme generickou implementaci, o které víme, že funguje kdekoliv. To vše bez zásahu do kódu aplikace. Takovou flexibilitu nám EJB rozhodně neposkytují.

Jedním ze zásadních nedostatků první verze databázové vrstvy, byl fakt, že nepoužívala žádnou cache. Řada informací ve STAGu se příliš často nemění a je zcela zbytečné se na ně opakovaně dotazovat databáze. Byla tedy vytvořena další sada tříd (*Managers*), která v sobě obsahuje cachování nejdůležitějších dat. Při výběru vhodné implementace cache jsme se omezili pouze na základní funkce: možnost nastavit maximální počet položek, které je možné do cache uložit a nastavení platnosti jednotlivých záznamů. Websphere Application Server obsahuje distribuovanou cache, která slouží pro ukládání výstupů servletů a portletů, ale lze ji použít také na ukládání jakýchkoliv objektů. Výhodou této implementace je možnost sdílet cache mezi více portletovými aplikacemi. Pro aplikační servery, které neobsahují vlastní implementaci cachování, používáme open source knihovnu *WhirlyCache* [5]. Díky Spring frameworku změna zvoleného cachovacího řešení spočívá v úpravě několika řádek jeho konfiguračního souboru. Pokud nám přestane zvolená implementace cachování vyhovovat je tedy velmi snadné ji nahradit bez jakéhokoliv zásahu do kódu aplikace.

Celou datovou vrstvu je možno bez zásahů do kódu přesunout na jiný aplikační server a volat její služby přes RMI. V našem nasazení tato vrstva zatím běží

v portálovém serveru a portletová vrstva s ní komunikuje přes lokální rozhraní. Výkonnostně nám toto řešení zatím postačuje a nemáme otestováno, zda-li by se komunikace mezi portálovým serverem a datovou vrstvou umístěnou na jiném stroji nestala úzkým hrdlem. Navíc přesunutím této vrstvy na jiný stroj bychom přidali další vazbu do již tak složité portálové infrastruktury.

Portletová vrstva

Poslední vrstva slouží jako prostředník mezi webovým prohlížečem a datovou vrstvou aplikace. Jejím výstupem je primárně HTML. Tato vrstva by měla být implementována podle známého návrhového vzoru MVC (*Model-View-Controller* [6]). V našem případě jsme se rozhodli použít k její implementaci portletové API. Důvodem bylo nasazení portálového řešení, které má sloužit jako integrační prostředí informačních systémů na naší univerzitě.

Většina portálů má svoje proprietární API na psaní portletových aplikací, které však není přenositelné na jiné portálové produkty. První portlety, které poskytovaly základní informace z IS/STAG byly napsány v IBM Portlet API. V té době ještě neexistoval standard JSR 168 [7], který definuje jednotné API pro vývoj portletových aplikací a umožňuje snadnou přenositelnost portletových aplikací mezi různými servery. Naštěstí se tento standard ujal a momentálně ho podporují všechny důležité portálové servery.

JSR 168 rozhodně není dokonalý standard a nám v něm chybí následující funkce:

- Není jasně definovaný způsob komunikace mezi portlety a k posílání zpráv se musí využívat portletové sezení.
- Portlet nemůže ovlivňovat vzhled portálových stránek. Nelze tak z portletu měnit např. obsah navigačního menu.
- Portlety nemohou mít sdílené parametry a musejí si zasílat zprávy, které tyto parametry obsahují.

Tyto nedostatky jsou řešeny v novém standardu JSR 286 [8], který je momentálně (27. 8. 2007) ve stavu *Public review*. Než se tento standard dostane do všech portálových serverů bude trvat minimálně jeden rok. Do té doby ho nelze v našem případě používat a je nutné držet se stávajících možností. I přes výše zmíněné nedostatky nám tento standard posloužil dobře a vývoj aplikací je v něm celkem snadný.

Závěr

Před nasazením nové verze *Předzápisu* byl vytvořen program, který simuloval typické chování studenta při používání této aplikace. Tento program byl použit při zátěžovém testování, během kterého bylo nalezeno několik problematických bodů – úzkých hrdel. Nová verze *Předzápisu* se v ostrém provozu osvědčila. Počet výpadků databáze se oproti minulým létům značně omezil, zlepšila se rychlost odezvy webového rozhraní, došlo ke zvýšení počtu paralelně pracujících uživatelů – tedy k celkovému navýšení propustnosti systému.

J2EE platforma je v dnešní době obrovský „ekosystém“, který vývojářům poskytuje mnoho možností a technologií. Je však snadné se v něm ztratit a vybrat si technologie, které místo usnadnění vývoje kladou zbytečné překážky. Najít správné prostředky (knihovny, vývojové prostředí, ...) je důležitým krokem na cestě k úspěšnému a rychlému vývoji. Nám se, stejně jako spoustě dalších vývojářů, vyplatilo začít používat Spring framework jako náhradu za EJB.

Zdroje

[1] http://download.oracle.com/docs/cd/B19306_01/java.102/b14355/proxya.htm

[2] http://java.sun.com/developer/technicalArticles/J2EE/connectorclient/resourceadapter.html

[3] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

[4] https://whirlycache.dev.java.net/

[5] http://www.springframework.org/

[6] http://java.sun.com/blueprints/patterns/MVC-detailed.html

[7] http://developers.sun.com/portalserver/reference/techart/jsr168/pb_whitepaper.pdf

[8] http://jcp.org/en/jsr/detail?id=286

Linuxový kernel v posledních letech

Jiří Kosina

E-mail: jiri.kosina@suse.com

Klíčová slova: Linux, kernel, development

Abstrakt

Linuxový kernel je jedním z nejpopulárnějších open source projektů, z čehož vyplývá několik specifických vlastností vývoje a vývojového modelu, které u jiných open source projektů nenajdeme (obrovské množství vývojářů, velké množství změněného či nově přidaného kódu za časovou jednotku, atd). Článek se bude zabývat jednak specifiky vývojového modelu jádra, druhak popisem některých nových vlastností, které byly pro kernel v poslední době vyvinuty a které mohou být zajímavé i pro uživatele či administrátora Linuxového systému.

Abstract

The Linux kernel is one of the most popular open-source projects, which implies some specific properties of its development and development model that can't be found among other open-source projects (large number of developers, lots of code lines changed every day, etc). In this article, we describe the specific properties of the Linux kernel development model, and also describe some of the new features that have been developed for the kernel codebase lately, with respect to those features that could potentially be interesting even for an ordinary user or Linux system administrator.

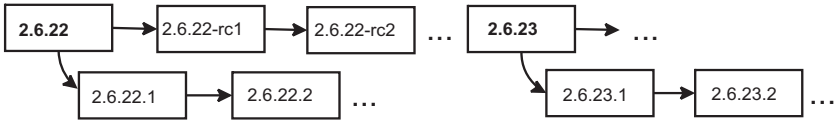
1 Vývojový model

1.1 Verze

V minulosti probíhal vývoj Linuxového kernelu ve dvou paraleních stromech, z nichž jeden nesl označení *stable* a druhý *development*. Vývojová větev byla označena lichým číslem na druhé pozici v číslu verze (např. 2.5.20), sudé číslo značilo stabilní větev (2.4.10).

S příchodem 2.6 kernelu se situace změnila – každý 2.6 kernel je považován za stabilní a nové verze jsou vydávány v pravidelných intervalech dvou až tří měsíců. Vždy po uvolnění nové verze kernelu je vývojářům umožněno začlenit implementaci nových vlastností do jádra (to bylo dřív možné pouze do vývojové větve, stabilní větev pak měla obsahovat pouze opravy chyb a drobná vylepšení).

Navíc došlo k později vytvoření – stable větve, sledující vždy aktuální verzi kernelu a zahrnující jen opravy kritických chyb v dané verzi. Tím je distribucím umožněno snadno zůstávat na jedné konkrétní verzi jádra pro danou distribuci, a mít zaručenu dostupnost oprav kritických chyb. Viz obrázek 1.



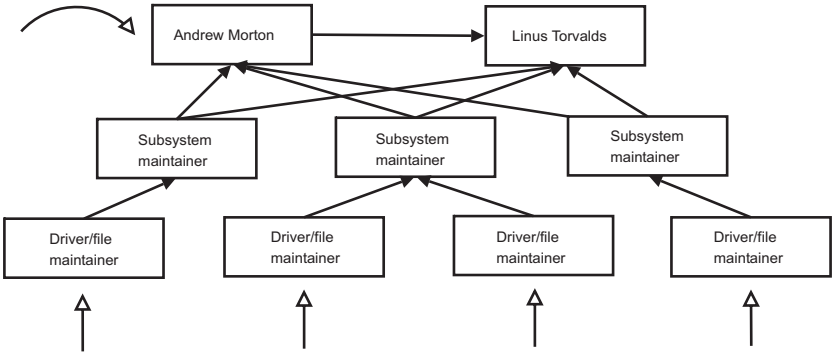
Obr. 1 Číslování verzí

1.2 Hierarchický model

V jistém smyslu však nebylo od rozdělení kernelového kódu na *stable* a *development* upuštěno úplně. Důvod je prostý – existuje příliš mnoho novinek, které jsou tak komplikované, že jim nestačí testování pouze mezi dvěma stabilními verzemi (dva až tři měsíce). Z tohoto důvodu existuje vývojový strom který spravuje Andrew Morton, a který v sobě obsahuje jednak všechen kód, který správci jednotlivých kernelových subsystémů (nezávislých oblastí, jako je síť, vstupní zařízení, memory management, hardwarové architektury, atd) mají připraven pro začlenění do další stabilní verze, tak i velmi experimentální kód, který často potřebuje ještě velmi mnoho testování a ladění. Pro podrobnější znázornění cesty, kterou urazí každý kernelový patch od jeho autora skrz maintainery až do Linusova stromu, viz obrázek 2.

1.3 Rychlost vývoje, source code management

Původně (až do verze 2.5) nepoužívali vývojáři Linuxového kernelu vůbec žádný centrální nástroj pro správu verzí – veškerý vývoj (jak samotná komunikace, tak zasílání patchů) probíhal výhradně za použití e-mailu. Během vývoje verze 2.5 začal Linus Torvalds používat software BitKeeper od firmy BitMover – http://www.bitmover.com. Jednalo se o krok, který velká část vývojářů považovala za poměrně kontroverzní (kvůli licenci), nicméně žádný z vývojářů nebyl nucen BitKeeper používat, Linus stále přijímal patche ve formě e-mailu. Kolem verze 2.6.12 se z různých (především licenčně-politických) důvodů Linus rozhodl ke změně software na správu zdrojových kódů. Bohužel nebylo možné



Obr. 2 Hierarchický vývojový model

najít žádný existující, který by plně vyhovoval velmi distribuovanému vývojovému modelu jádra (každý maintainer chce mít svůj vlastní strom, je nutné mít možnost spojovat některé z vývojových větví vzájemně mezi sebou, nesmí existovat žádný „hlavní“ strom, apod). Během velmi krátké doby Linus Torvalds napsal základní kód SCM s názvem Git – http://git.or.cz/, který velmi rychle začali používat i ostatní správci velkých kernelových subsystémů, a práce se tím velmi zefektivnila. Jako vedlejší efekt je nyní možné získávat triviálně statistiky o zdrojovém kódu jádra. Následující zpracoval Greg Kroah-Hartman. Tabulka 1 a tabulka 2 demonstrují jednak rychlost, jakou je Linuxové jádro opravováno a je přidáváván nový kód, a také ukazuje podíl vývojářů jednotlivých firem na vývoji jádra.

Tabulka 1 Rychlost vývoje

Verze	Řádek	Patchů celkem	Patchů za hodinu	Vývojářů	Firem
2.6.11	6624076	4041	2.44	479	30
2.6.12	6777860	5565	2.15	704	38
2.6.13	6988800	4174	2.38	641	39
2.6.14	7143233	3931	2.69	632	45
2.6.15	7290070	5410	3.31	685	49
2.6.16	7480062	5734	3.10	782	56
2.6.17	7588014	6113	2.80	787	54
2.6.18	7752846	6791	2.98	904	60
2.6.19	7976221	7073	4.09	887	67
2.6.20	8102533	4983	3.05	730	75
2.6.21	8246517	5349	2.75	838	68

Tabulka 2 Podíl firem na vývoji

Firma	Počet změn	Procenta
Nezjištěno	27976	47.3
Red Hat	6106	10.3
Novell	5923	10.0
Linux Foundation	4843	8.2
IBM	3991	6.7
Intel	2244	3.8
SGI	1353	2.3
NetApp	636	1.1
...	...	...

2 Nové vlastnosti

V druhé části článku se pokusíme z tohoto nepřeberného množství změn v jádře vybrat pět takových, které mohou být nějakým způsobem zajímavé nejen pro low-level programátora jaderných modulů či ovladačů zařízení, ale spíše pro uživatele a správce Linuxových systémů. Zároveň bylo cílem vybrat takové, které jsou poměrně nové, takže o nich ještě nevyšlo příliš populárních článků, které by srozumitelně vysvětlovaly principy jejich fungovnání.

2.1 Realtime kernel

Existuje několik různých definic realtime systému. Realtime systémy jsou často používány v prostředích, ve kterých je nutné zaručit takové uspořádání (resp. naplánování) událostí, které vyhovuje předem stanoveným kriteriím (deadlines).

Realtime kernel patch – podrobnější informace viz http://rt.wiki.kernel.org/ – funguje na jiném principu. To je třeba si včas uvědomit, aby se předešlo případnému rozčarování.

Zásadní termín, se kterým kernelový realtime operuje, je *latence*. Latencí rozumíme čas, který je nutný k reakci na dané události (uživatel stiskne klávesu, ovladači jsou doručena do bufferu data, ...). Nízká latence je důležitá především pro desktopové systémy, jelikož zajištuje, že se systém k uživateli chová velmi přívětivě co do interaktivních reakcí. Ještě důležitější je snižování latence pro některé specializované aplikace – práce s audio/video, bankovní operace, atd.

Autoři real-time patche definují dva typy latence – tzv. *scheduling latency* a *interrupt latency*.

První z nich je ovlivněna dlouhými čekacími smyčkami v kódu kernelu (spinlocky) a dlouhým trávením času v místech kde kernel nemůže být přeplánován. Real-time patch řeší dlouhé čekací smyčky na spinlocku záměnou všech kernelových spinlocků na mutex, jejichž implementace nejen že dovoluje preempci (a tedy ani velmi vytížený spinlock nezaměstná plně procesor dokud se zamknutí nepovede) při čekání na spinlock, ale v důsledku toho je také možné přeplánovat i kód který spinlock (resp. mutex) drží, což v současné chvíli není možné ani při plně preemptivním kernelu.

Druhý typ – interrupt latency – je ovlivněna faktem, že kód obsluhy přerušení má v systému výsadní postavení, jelikož nemůže být přeplánován libovolným jiným kernelovým kódem dokud obsluha přerušení nedoběhne, navíc je kvůli deadlockům nutná speciální synchronizace (po dobu obsluhy přerušení jsou všecha ostatní přerušení maskována). Řešení (vzhledem k latenci), které nabízí real-time patch, má základní myšlenku poměrně jednoduchou – kód pro jednotlivé IRQ je přesunut do kernelových vláken (jedno vlákno pro každé IRQ). Po provedení této změny je možné kód obsluhy IRQ jednak přeplánovat, druhak je možné mu dynamicky měnit prioritu, podle toho jak si uživatel/administrátor systému přeje které IRQ prioritizovat (např. může být žádoucí prioritizovat obsluhu přerušení od sítové karty před čímkoliv jiným, apod).

Celý patchset ještě není v kernelu začleněn, pouze některé jeho méně intrusivní části.

2.2 Tickless kernel

Linux je tradičně znatelně horší co do spotřeby energie, než některé jiné konkurenční operační systémy. Jedním z kroků ke zlepšení této situace je právě tickless kernel. Tuto vlastnost je možné při bootu jádra dynamicky zapínat a vypínat na příkazovém řádku pomocí klíčového slova `nohz`.

Základní myšlenka je velmi jednoduchá. Ve starších Linuxových jádrech je konstantou `HZ` určeno, jak často mají chodit přerušení od hardwarového časovače. Obsluha přerušení znamená spustit obslužný kód i tisíckrát za vteřinu. Často ovšem není na daný časový okamžik naplánována žádná událost, a tak obsluha přerušení vyjde naprázdno. Tickless patche se proto při každém tiku hardwarového časovače, zjednodušeně řečeno, podívají, na kdy je naplánovaná nejbližší událost související s časovačem, a požádá hardware o tiknutí až za tuto dobu. Měření ukazují, že na typickém systému se tímto počet tiknutí dramaticky sníží. Po dobu nečinnosti procesoru je možné ho přepnout do nižšího energetického režimu (C-state). Tím dochází k výrazným úsporám energie (a tedy například s tickless kernelem notebooky vydrží výrazně déle běžet na napájení z akumulátoru). Tickless patche se poprvé objevily v jádře 2.6.21, do přiměřeně odladěného stavu byly upraveny až ve verzi 2.6.22.

2.3 SLUB allocator

Uživatel či administrátor do žádného blízkého kontaktu s kernelovým alokátorem paměti pravděpodobně nikdy nepřijde, nicmněně přesto ho v tomto článku zmíníme, jelikož algoritmy použité v alokátoru kernelové paměti mají přimý vliv na výkon celého systému.

V kernelu existuje již tradičně alokátor SLAB. Tento alokátor udržuje fronty pro jednotlivé velikosti alokací, kvůli snadnějšímu cachování objektů dané velikosti. Na druhé straně management těchto front přidává netriviální požadavky na potřebnou pamět – jednoduše řečeno čím více paměti kernel má, tím (řádově) více je třeba jí obětovat pro alokátor. Christopher Lameter, kernelový vývojář z SGI k tomu říká:

> Fronty SLAB objektů jsou na každém uzlu a procesoru. Fronta alien cache má dokonce pole front, které obsahuje frontu pro každý procesor na každém uzlu. U velkých systémů může počet front a objektů, které může být potřeba v těchto frontách uožit mohou zachytit, růst exponenciálně. Na našich systémech s 1k uzlů a/nebo procesorů máme několik gigabytů spotřebovaných jen referencemi na objekty těchto front. A to nepočítám ty, které by mohly být ve frontách. Mám obavy, že by těmi frontami jednou mohla být zabrána celá paměť stroje.

Nový alokátor – SLUB – odstraňuje tento (a další) problém SLAB alokátoru. Velmi dobře škáluje na velkých systémech (však ho také vyvíjí u SGI), a výrazne na nich šetří pamětí. Konkrétní detaily implementace jsou již nad rámec tohoto článku, a proto se jimi nebudeme dále zabývat.

V aktuálních verzích jádra je možné při kompilaci zvolit, který z alokátorů se má použít.

2.4 Virtualizace

O virtualizaci toho bylo již řečno a napsáno mnoho, nicmněně by nebylo vhodné ji zde, vzhledem k aktuální popularitě, oponenout.

Vytrvalým bojovníkem na poli Linuxové virtualizace je již dlouhou dobu Xen, viz http://www.xensource.com/. Začlenění Xenu do hlavního kódu jádra bylo dlouhou dobu odmítáno, především kvůli příliš intruzivním změnám v kernelových interfacech, které byly navíc řešeny s velmi malým ohledem na obecnost, a tedy případná další virtualizační řešení by musela implementovat veškeré potenciálně sdílené věci znova.

Dříve než vývojáři Xenu stihli tyto problémy opravit, bylo do Linuxového jádra začleněno jiné virtualizační řešení – KVM od firmy Quamrnet – viz http://web1.qumranet.com/.

Mnozí předvídali po tomto kroku Xenu zkázu, protože dvě virtualizační řešení v kernelu vypadala velmi nepravděpodobně. Nicméně KVM má pro některé uživatele jednu podstatnou nevýhodu – vyžaduje procesory s podporou virtualizace, což Xen nevyžaduje. Nakonec se podařilo Xen upravit tak, aby správně využíval obecnou vrstvu v kernelu pro virtualizaci – `paravirt_ops` – a tudíž mohl být také začleněn do kernelového stromu.

Vzniknutí této vrstvy velmi podstatně zjednodušilo psaní paravirtualizačních řešení. Jednou z takových velmi jednoduchých implementací virtualizace je Lguest hypervisor, který je začleněn do aktuálního Linusova stromu. Hlavním cílem bylo ukázat, jak jednoduché je napsat hypervisor za pomocí této obecné vrstvy (cca 5 000 řádek kódu), nicméně výsledek sám o sobě není špatný a je možné, že bude postupně docházet k dalšímu rozšiřování.

2.5 Asynchronní syscally

Asynchronní I/O je vleklý problém z mnoha důvodů – implementace vyžaduje poměrně sofistikovaný stavový automat, je obtížné naprogramovat systém bez race conditions a zároveň rozumný co do výkonu, atd.

Asynchronní syscally umožnují aplikačním programátorům zkonstruovat malé jednoduché „programy“ které jsou předem definovanou formou předány kernelu, který je následně provádí (asynchronně), bez nutnosti vracet řízení zpět do userspace.

Základní myšlenka je uštření režie, která je potřebná při předávání řízení mezi kernelem a userspace v průběhu každého systémového volání. Navíc je možné nechat kód v kernelu vykonávat asynchronně.

Původní implementace (v kernelu není stále podpora začleněna) vykazovala na velmi jednoduchém stress testu (open() + read() + close() na 1 000 náhodných souborů z disku) zlepšení o cca $\frac{1}{3}$ co do času potřebného na dokončení operace.

Desktop a jádro Linuxu

Jan Kasprzak

E-mail: kas@fi.muni.cz

Klíčová slova: Linux, desktop, SysFS, `freedesktop.org`, hal, d-bus, Avahi, zeroconf, suspend

Abstrakt

Operační systém UNIX nebyl původně navrhován jako desktopový systém. Linux, který do značné míry přebírá API i další vlastnosti UNIXu, dnes ale najdeme na široké škále systémů – od mobilních telefonů přes desktopy až po superpočítače z TOP 500. V přednášce se zaměříme právě na desktopové nasazení Linuxu, a to zejména na technologie, kterými se Linux liší od jiných (i UNIXu podobných) systémů nebo které jsou jinak blízko jádru samotnému.

Abstract

The UNIX operating system has not originally been designed as a desktop system. Linux, which more or less copies the API and other properties of UNIX, can currently be found on various devices ranging from cell phones to desktops to the supercomputers of the TOP 500 list. In this paper, we will focus on Linux installed on desktops and especially on the technologies which make it different from other (even UNIX-like) systems as well as those that are somehow close to the kernel itself.

1 Desktop? Je to vůbec zajímavé?

V současné době je Linux pravděpodobně nejflexibilnější operační systém. Jádro Linuxu přeložené z jedněch a těch samých zdrojových textů, s tím stejným aplikačním rozhraním, najdeme na hardwaru od mobilních telefonů[1] až po masivně paralelní superpočítače ze seznamu TOP 500[2]. Každá z těchto oblastí nasazení

[1] Například http://www.openmoko.org.

[2] Podle http://www.top500.org/stats/list/29/osfam běží pod Linuxem 77,8 % všech superpočítačů z aktuálního TOP 500.

Linuxu má svá specifika: od vývoje speciálních souborových systémů pro nerotační média (například flash paměti) až po snahu o maximální paralelizaci různých částí jádra včetně úplného vyloučení zamykání[3]. Řada z těchto snah má nečekaný přesah z původní oblasti nasazení do jiné třídy počítačů.

Z hlediska operačního systému lze za nejzajímavější oblast nasazení považovat právě desktopové počítače. Malá zapouzdřená zařízení často zpracovávají jen jednu jednoúčelovou aplikaci, a od operačního systému kromě správy hardwaru a možná komunikace po síti vyžadují jen jedno – aby se hlavní aplikaci pokud možno co nejméně pletl do cesty. Podobně je tomu někdy i u velkých počítačů, často zpracovávajících jediný program – databázový stroj (který je sám malým operačním systémem včetně správy paměti a diskových bufferů).

Naproti tomu na desktopovém počítači typicky běží desítky až stovky nejrůznějších procesů s velmi odlišnými nároky na systém (multimediální přehrávače, webové prohlížeče, správci souborů, prohlížeče a konvertory dokumentů, atd.). Desktopové počítače také aktivně komunikují s největší škálou zařízení: od blokových zařízení, připojených přes nejrůznější sběrnice velmi odlišných rychlostí a vlastností (IDE, SATA, SCSI/SAS/FC, USB, IEEE 1394) přes hardwarově akcelerované grafické a zvukové karty, až po síťová zařízení (od Ethernetu přes PPP a Wi-Fi až po Bluetooth), která ještě ke všemu mohou v systému dynamicky vznikat a zanikat. Linus Torvalds říká[4]:

> *What makes the desktop so interesting is in fact that it shows more varied usage than any other niche.*

V tomto článku se zaměříme na to, co jádro Linuxu poskytuje desktopovým aplikacím navíc oproti jiným UNIXovým systémům. Dále pak na nástroje, které primárně vznikly pod Linuxem (i když mnohdy jsou přenositelné i na jiné systémy), a mají za cíl zlepšit propojení desktopového prostředí se zbytkem operačního systému.

2 Správa zařízení v Linuxu 2.6

2.1 Exkurze do historie

UNIX již tradičně rozlišuje dvě hlavní třídy zařízení: bloková zařízení (například disky; vyznačují se možností přímého/náhodného přístupu a obvyke bloky pevné délky) a znaková zařízení (pásky, tiskárny, terminály a podobně). Zařízení je charakterizováno svým *speciálním souborem* v adresáři `/dev`, a ten zase svým *hlavním a vedlejším číslem*:

[3] Například metoda Read-Copy-Update: http://lse.sourceforge.net/locking/rcupdate.html
[4] http://lkml.org/lkml/2007/7/28/106

```
$ ls -l /dev/sdb
brw-r----- 1 root disk 8, 16 Aug 23 17:05 /dev/sdb
```

Zde jde o blokové zařízení (`b` na začátku výpisu) `/dev/sdb` s hlavním číslem 8 a vedlejším číslem 16. Hlavní číslo se používá pro adresaci ovladače (driveru) uvnitř jádra a vedlejší číslo pro rozlišení mezi zařízeními, spravovanými tímtéž ovladačem (zde ovladač SCSI disků).

Pamětníci jistě vzpomenou, jak fungovala správa zařízení na UNIXu v7 a systémech z něj odvozených: jádro mělo v sobě zakompilovaná dvě statická pole – tabulky `bdevsw[]` a `cdevsw[]`. V těchto polích (indexovaných hlavním číslem zařízení) měl každý ovladač uvedeno, která jeho funkce – dnes bychom objektově řekli *metoda* – se má zavolat při pokusu o otevření zařízení, čtení, zápisu a dalších operacích.

2.2 Kde je problém?

Výše uvedený přístup je do značné míry problematický:

- Jak velké má být `dev_t`? Norma POSIX říká, že hlavní a vedlejší číslo by se mělo dohromady vejít do jednoho numerického typu. UNIX používal 8 bitů na hlavní a 8 bitů na vedlejší číslo zařízení. To ovšem ne vždy dostačuje.

- Co všechno mít v `/dev`? Tradičně UNIX měl v adresáři `/dev` speciální soubory pro většinu možných zařízení, i když na daném počítači se některé typy zařízení jistě nevyskytnou (starší distribuce Linuxu měly například poctivě vytvořené soubory `/dev/xd*` pro MFM/RLL disky, které byly naposledy ve větší míře k vidění někde na IBM PC-XT).

- Jak vlastně zařízení adresovat? Linux obvykle používal označení podle pořadí detekce: `/dev/sda` byl první SCSI disk, `/dev/sdb` druhý atd. Když pak zmizel nebo byl přidán disk někam doprostřed, označení části disků se posunulo. Naproti tomu Solaris používá jména typu `/dev/dsk/c0t7d3s0` podle topologie (číslo SCSI sběrnice, SCSI ID, SCSI LUN, číslo oblasti). Máme-li ale v systému třeba jen jeden disk a přepojíme jej na jiný řadič, začne se najednou jmenovat jinak. Na desktopových systémech je to ještě horší: co když vezmeme IDE disk (na Linuxu dosud známý jako `/dev/hdc`), zabalíme jej do USB nebo IEEE 1394 rámečku a připojíme jej k počítači? Je to ten stejný disk a měl by se tedy jmenovat stejně?

- Dynamicky vznikající zařízení. Tato zařízení byla do jisté míry i na starších UNIXech – vzpomeňme na pseudoterminály a adresář `/dev` plný souborů jako `ttypN`, `ptypN`, `ttyqN`, atd. Jak se má jmenovat nově zapojený USB flash disk nebo Bluetooth klávesnice? Jaká má mít přístupová práva?

2.3 DevFS – slepá cesta

V Linuxu řady 2.4 se objevil virtuální souborový systém DevFS, jehož idea byla pravděpodobně převzata ze Solarisu. Ovladače samy registrovaly v DevFS speciální soubory podle toho, která zařízení zrovna viděly v systému. Systém DevFS se ovšem v reálných distribucích příliš neuchytil (s čestnou výjimkou Gentoo Linuxu), protože přinášel řadu problémů, mezi nimi tyto:

- Politika uvnitř jádra. DevFS vnucoval uživateli svoje pojmenování zařízení (u disků bylo podobné tomu na Solarisu) a nebyla možnost toto změnit.
- Okrajové stavy. DevFS s sebou nesl spoustu *race conditions* mezi otevíráním zařízení a paralelním rušením jeho ovladače z jádra.
- Přístupová práva: speciální soubor v DevFS prostě vznikl, aniž by správce systému jakkoli mohl ovlivnit, s jakými přístupovými právy se vytvoří. Přístupová práva navíc nebyla perzistentní mezi restarty systému.

2.4 První část řešení – SysFS

V Linuxu 2.6 byl DevFS prohlášen za zastaralý a skupina programátorů v čele s Gregem Kroah-Hartmannem začala pracovat na alternativě. Výsledkem byl nový model správy zařízení (všech – nejen blokových a znakových, ale také ovladačů sběrnic, síťových zařízení a dalších). Zařízení jsou zde evidována podle topologie (připojení na sběrnici), třídy (disk, vstupní zařízení, atd.) ovladačů, hlavních a vedlejších čísel a dalších atributů.

Datové struktury nového modelu jsou navíc přehledně zpřístupněny v novém virtuálním souborovém systému SysFS, obvykle připojeném na adresář `/sys`. Kromě základních atributů jako je topologie obsahuje SysFS i možnosti provádět některé akce pomocí čtení nebo zápisu hodnot z/do příslušných virtuálních souborů:

```
# echo 0 0 0 > /sys/class/scsi_host/host0/scan
# echo 1 > /sys/block/sdb/device/delete
# echo 45 > /sys/devices/platform/w83627hf.656/temp2_max
# cat /sys/class/input/input0/name
Power Button (FF)
# cat /sys/bus/pci/devices/0000:00:0f.1/vendor
0x1106
```

Správce systému tedy v SysFS má kompletní inventář hardwaru včetně typů zařízení, topologie, atributů jako výrobce, výrobní číslo a podobně. Navíc většina ovladačů byla upravena tak, že své hlavní číslo registrují dynamicky, a tedy na jednom systému může koexistovat daleko více zařízení, než by mohlo při statickém přidělování hlavních čísel centrální autoritou.

2.5 Druhá část řešení – hotplug

Jeden z problémů evidence zařízení je, že na desktopových systémech zařízení dynamicky vznikají a zanikají. Jádro přirozeně aktuální stav stále zobrazuje uvnitř SysFS, nicméně by bylo dobré nemuset pořád sledovat změny a být o nich informován až podle potřeby.

Jádro Linuxu 2.6 poskytuje dva mechanismy: starší `hotplug` funguje tak, že jádro při vzniku nebo zániku zařízení spustí externí program `/sbin/hotplug`, kterému předá informace o příslušném zařízení.

Pro větší frekvenci vzniku a zániku zařízení jádro poskytuje sofistikovanější notifikační mechanismus. Jde o frontu událostí, postavenou nad virtuálními sockety rodiny `AF_NETLINK`.

2.6 Vše pak propojí `udev`

Nyní už je vidět, že jádro dává správci systému nejen potřebné informace o aktuálním stavu zařízení v systému, ale i notifikační mechanismy, informující o změně. S těmito informacemi je už možné implementovat správu adresáře `/dev` v uživatelském prostoru.

Jádro ovšem zůstává zpětně kompatibilní se statickým `/dev`: nejpoužívanější zařízení mají i nadále přidělené své hlavní číslo staticky, a tak vytvoříme-li si ručně (nebo z distribuce) v `/dev` všechny potřebné soubory, může systém i nadále fungovat postaru.

Jedním ze systémů, využívajících dva výše uvedené mechanismy pro dynamickou správu adresáře `/dev`, je balík `udev`. S jeho pomocí můžeme mít adresář `/dev` buďto jako dříve přímo na disku, anebo jej přesunout na ramdisk (souborový systém TmpFS) a vytvářet speciální soubory dynamicky, vycházejíce z prázdného adresáře `/dev`.

Systém `udev` má dvě fáze činnosti: ta první, nazývaná *coldplug* inventarizuje stávající zařízení a vytváří jim speciální soubory v `/dev`. Ve druhé fázi pak již démon `udevd` zpracovává *hotplug* události od jádra a vytváří, resp. ruší speciální soubory nově objevivších se, resp. právě odpojených zařízení.

Podstatným přínosem systému `udev` je, že politiku (pojmenování zařízení, přístupová práva a podobně) přenáší do uživatelského prostoru do svých konfigurací. Například: líbí se mi pojmenování disků ze Solarisu, ale chci, aby se myš v mém laptopu jmenovala vždy `/dev/mouse` bez ohledu na to, jakou myš použiji a do kterého USB portu na zadní straně laptopu se zrovna poslepu trefím.

Systém `udev` dokonce umožňuje i správci systému nastavit reakce na vznik zařízení typu vyvolání externího skriptu nebo notifikace přes D-Bus (viz dále).

2.7 Příklady konfigurace udev

Chování udev lze nastavovat pomocí sad pravidel, umístěných do souborů v adresáři /etc/udev/rules.d:

```
# PS/2 mys
SYSFS{protocol}=="ImExPS/2", \
   KERNEL=="mouse*", \
   SYMLINK+="input/mouse_first"
# USB mys
SYSFS{name}=="Logitech USB-PS/2 Optical Mouse", \
   KERNEL=="mouse*", \
   SYMLINK+="input/mouse_second"
# PS/2 klavesnice
SYSFS{name}=="AT Translated Set 2 keyboard", \
   SYSFS{phys}=="isa0060/serio*/input*", \
   KERNEL=="event*", \
   SYMLINK+="input/keyboard_first"
# USB klavesnice
KERNEL=="event*", \
   SYSFS{../name}=="HID 1267:0103", \
   SYSFS{bInterfaceNumber}=="00", \
   SYMLINK+="input/event31"
```

Příklad konfigurace vstupních zařízení pro víceuživatelský desktop: PS/2 klávesnice a myš jsou zpřístupněny jako /dev/input/keyboard_first a mouse_first, USB klávesnice a myš jako event31 a mouse_second.

```
KERNEL=="ttyUSB*", \
   ATTRS{product}=="Papouch TMU Thermometer", \
   ATTRS{serial}=="PPQ3NTMG", \
   SYMLINK+="tmu0"
```

Zpřístupní USB teploměr daného sériového čísla (viditelný dosud jako USB sériový port) pod jménem /dev/tmu0.

```
KERNEL=="sd*1", \
   SYSFS{model}=="G3               ", \
   SYSFS{vendor}=="M-System", \
   RUN+="/usr/local/sbin/zpravy-to-palm"
```

Zkopíruje po připojení handheldu Palm v režimu USB mass storage do jeho paměti aktuální OGG soubor s poslední zpravodajskou relací oblíbené rozhlasové stanice.

2.8 Psaní pravidel pro udev

Asi nejjednodušší cestou jak zjistit, podle čeho vlastně lze v pravidlech pro udev vybírat a vyhledávat naše zařízení, je zařízení připojit, podívat se pod jakým jménem je aktuálně viditelné v /dev a spustit následující příkaz:

```
# udevinfo -a -p ‘udevinfo -q path -n /dev/mojezařízení‘
```

Upozornění: ve výše uvedeném příkazu se jedná o zpětné apostrofy.

Z vypsaných atributů pak vybrat ty, které považuji za charakteristické pro dané zařízení a můj záměr: někdy mě například zařízení zajímá podle svého sériového čísla nebo identifikace výrobce (ať už je zapojeno kamkoli a přes kteroukoli sběrnici), někdy naopak podle místa kam je zapojeno nebo podle pořadí objevení v systému, a tak podobně.

Podrobnější informace o psaní pravidel pro udev lze najít v článku *Writing* udev *rules*[5].

2.9 udev a disky

Systém udev má jednu specialitu pro disky: pro většinu scénářů použití má připravené symbolické linky v adresáři /dev/disk:

- /dev/disk/by-id/scsi-SATA_WDC_WD2500JB-32WD-WMAEP1078446 je pojmenování podle výrobce a sériového čísla.
- /dev/disk/by-path/pci-0000:00:0f.1-scsi-0:0:0:0 – podle topologie nadřazených sběrnic.
- /dev/disk/by-uuid/7056dad9-8837-485d-bdbc-88225b76c678 – pojmenování diskové oblasti podle UUID souborového systému na ní. Zůstává stejné i po dump(8) a restore(8) například po havárii disku.

3 HAL a evidence zařízení

Dalším typicky desktopovým problémem je evidence dostupných zařízení, jejich vlastností, schopností a atributů. Například program pro archivaci fotografií řeší úkol „najdi na tomto počítači připojený fotoaparát“, ať už je připojený jako USB nebo IEEE 1394 mass storage zařízení (aktuálně přimontované nebo ještě nepřipojené) nebo jako USB zařízení ovládané nějakým proprietárním protokolem (například fotoaparáty Canon).

Podobně program pro vypalování CD nebo DVD chce zjistit nejen jaké CD a DVD mechaniky v systému jsou, ale také potřebuje vědět, které z nich umí

[5] http://reactivated.net/writing_udev_rules.html

samy nahlásit, že v nich je médium, a které naopak je třeba periodicky po několika vteřinách na přítomnost média dotazovat.

A desktopové prostředí potřebuje vědět, jak jsou na tomto konkrétním laptopu pojmenovány všechny rozšiřující klávesy a jaké vysílají scan kódy.

Řešením tohoto problému pro Linux na desktopu je HAL[6], jeden ze standardů `freedesktop.org`. Je původně psán pro Linux, ale postupně se jeho hardwarově specifická strana implementuje i pro jiné systémy, jako je FreeBSD a Solaris.

Středem systému HAL je démon `hald`. Běží na pozadí a eviduje zařízení, která v systému jsou. Mimo jiné je také napojen na mechanismus `udev`.

Dále má HAL sady skriptů pro neobvyklá zařízení (například zamykání dvířek některých méně obvyklých CD mechanik) a databázi zvláštností jednotlivých zařízení.

Výpis databáze HAL lze získat například programem `lshal(1)`. Vyhledávání je možné pomocí programů `hal-find-by-capability(1)` a `hal-find-by-property(1)`. Bližší informaci o již nalezeném zařízení lze zjistit programem `hal-get-property(1)`:

```
$ hal-find-by-capability --capability storage.cdrom
/org/freedesktop/Hal/devices/storage_model_DVDRAM_GSA_4160B
$ hal-get-property --udi \
   /org/freedesktop/Hal/devices/storage_model_DVDRAM_GSA_4160B \
   --key storage.cdrom.write_speeds
7056 5645 4234 2822 1411 706
$
```

Dalším využitím HAL je správa přístupových práv k zařízením: HAL může evidovat přihlášené uživatele a povolit například přístup ke zvukové kartě těm uživatelům, jejichž sezení je u systémové konzoly. Další možností je namapování sady zařízení na konkrétní sezení: například informace o tom, která zvuková karta přísluší uživateli pracujícímu v X11 na AGP grafické kartě, a které jinému uživateli, sedícího u monitoru od PCI grafické karty. HAL v tomto případě používá modifikaci přístupových práv přes Access Control Listy.

4 Zasílání zpráv v desktopových prostředích

UNIXový model systému, kde je jádro striktně oddělené od aplikací, a kde X server je jen jedna z mnoha aplikací, přináší jisté problémy pro uživatelskou příjemnost desktopového systému. Pamětníci možná vzpomenou na podobný problém ve starších verzích grafických nadstaveb DOSu, kdy při chybě diskety byl uživatel přepnut zpět do textového režimu a zobrazena obligátní výzva

```
[A]bort, [R]etry, or [F]ail?
```

[6] http://hal.freedesktop.org/

UNIX má podobný problém: jádro je „příliš daleko“ od desktopových aplikací a nemá jak jim sdělit informace typu hardwarové chyby nebo i jen vzniku nového zařízení (na což by desktopové prostředí mohlo reagovat například vytvořením nové ikony na ploše uživatele).

Iniciativa `freedesktop.org` vytvořila pro tyto účely nový standard, komunikační rozhraní D-Bus[7], tedy Desktop Bus.

D-Bus je systém zasílání strukturovaných zpráv (ve formátu XML) mezi procesy. Obvyklé použití je v rámci jednoho systému, ale D-Bus podporuje i komunikaci nad TCP/IP. Komunikace může být buďto jeden k jednomu (běžné zasílání zpráv) nebo jeden k mnoha (publikuj/přihlas se k odběru). D-Bus může sloužit k aplikačním oznámením o změnách v systému. Tyto informace by jinak aplikace musely složitě získávat periodickým dotazováním nebo by v horším případě nebyly dostupné vůbec. Dalším využitím D-Bus je objektový přístup, kdy přes D-Bus lze vytvářet určité objekty a komunikovat s nimi zasíláním zpráv. Příkladem takového objektu může být třeba textový editor. Do jisté míry je technologie D-Bus podobná dalším systémům pro strukturované objektové zasílání zpráv (CORBA, COM, DCop a další)[8].

V běžném systému obvykle najdeme dvě nezávislé instance D-Bus:

- Systémová sběrnice. Je spuštěna po startu systému a běží po celou dobu existence systému. Sem posílají zprávy aplikace, nezávislé na uživatelově sezení: například zpráva `udev` o vzniku nového zařízení nebo zpráva HAL o připojení dalšího svazku. Desktopová prostředí pak mohou tyto zprávy zpracovávat a například podle typu zařízení zobrazit ikonku nového svazku na ploše nebo spustit stahování fotek z fotoaparátu.

- Uživatelská sběrnice. Vzniká při startu desktopového prostředí a každé sezení má tuto sběrnici samostatnou. Slouží pro zasílání zpráv v rámci sezení (změna globálního nastavení, například typ a velikost fontu; objektové použití, například vyvolání nového okna textového editoru).

Dalšími uživateli D-Bus jsou například následující projekty:

- `libnotify`[9] – systém zobrazování krátkodobých informativních hlášení uživateli desktopového sezení (viz též `notify-send(1)`).

- Galago[10] – knihovna a démon pro sledování prezence uživatele. Využívají IM a VoIP klienti pro globální sledování, je-li uživatel zrovna u počítače, je-li aktivní (nebo například zaneprázdněný VoIP hovorem), atd.

[7] http://dbus.freedesktop.org/

[8] Srovnání podrobněji ve FAQ: http://dbus.freedesktop.org/doc/dbus-faq.html

[9] http://trac.galago-project.org/wiki/DesktopNotifications

[10] http://www.galago-project.org/

5 Víceuživatelský desktop

Jednou z vlastností, kde se pozitivně projevuje modularita UNIXových systémů, je víceuživatelský desktop. UNIX je již od základu systém víceuživatelský, s individuálními nastaveními (například jazykovými) pro každého uživatele. Proto myšlenka na rozšíření tohoto pohledu o více uživatelů, současně pracujících *v grafickém sezení* na tomtéž počítači je jen dalším přirozeným krokem v řadě.

5.1 Více grafických karet

Víceuživatelský desktop s sebou nese několik problematických oblastí. tou největší je asi vzájemná kompatibilita více grafických karet v jednom počítači. Všechny grafické karty standardu VGA totiž mapují svoje VGA registry a sdílenou paměť do adresního prostoru mezi 640 KB a 1 MB. Při startu systému je BIOSem takto nainicializována jen primární grafická karta, což je v pořádku. Při startu druhého X serveru (pro sekundární grafickou kartu) se ale může stát, že vinou ovladače sekundární grafiky nebo i samotného hardwaru začne být i druhá grafická karta mapovaná do VGA oblasti a dojde ke konfliktu. Naštěstí většina nových grafických karet již tento problém nemá, a kromě textového režimou se obejde zcela bez VGA registrů a VGA sdílené paměti.

X server X.org se tradičně chová tak, že po startu projde všechny sběrnice počítače, a zakáže všechny grafické karty kromě té, se kterou právě pracuje. Toto je pro víceuživatelské prostředí nežádoucí. Podobně je třeba vyřešit, který z několika paralelně běžících X serverů se bude starat o přepínání virtuálních konzol. Příslušné přepínače X serveru se jmenují `-isolateDevice`, `-sharevts` a `-novtswitch`. Po zjištění PCI adres jednotlivých grafických karet (použijte příkaz `lspci(8)`) může konfigurace display manageru GDM vypadat například takto:

```
[servers]
0=Prvni
1=Druhy

[server-Prvni]
name=Prvni X server
command=/usr/bin/Xorg vt7 -layout ATiLayout \
    -isolateDevice PCI:01:00:0
flexible=true

[server-Druhy]
name=Druhy X server
command=/usr/bin/Xorg vt7 -layout RivaLayout \
```

```
    -isolateDevice PCI:00:19:0 \
    -sharevts -novtswitch
flexible=true
```

Podobně v konfiguraci X serveru (`xorg.conf`) musí mít každá grafická karta vyznačenou svoji PCI adresu:

```
Section "Device"
    Identifier  "ATI"
    Driver      "radeon"
    BusID       "PCI:1:0:0"
EndSection

Section "Device"
    Identifier  "Riva"
    Driver      "nv"
    BusID       "PCI:0:19:0"
EndSection
```

5.2 Více vstupních zařízení

Současné počítače stále ještě mají vstup pro klávesnici a myš PS/2 portem. Nejjednodušší řešení vstupních zařízení pro víceuživatelský desktop tedy je mít jedno pracovní místo ovládané z PS/2 klávesnice a myši a druhé místo z USB klávesnice a myši. Jen je potřeba jednotlivá zařízení od sebe rozpoznat i v případě odlišného pořadí objevování zařízení. V kapitole o systému `udev` jsme si naznačili, jak se tohle dělá.

Linux poskytuje pro každé vstupní zařízení tzv. *event interface*, speciální soubor `/dev/input/eventN`. Každá vstupní událost (například stisk nebo uvolnění klávesy, pohyb myši) je jádrem reportovaná jako balik dat pevné délky, který lze z event interface přečíst. Grafický subsystém X.org obsahuje ovladač `evdev`, který umí právě tyto události rozpoznávat a přeposílat X klientům. Pro jednoduchost nepoužijeme pro PS/2 klávesnici vestavěný driver X.org, ale i zde použijeme event interface. Konfigurace by v souboru `xorg.conf` by mohla vypadat nějak takto:

```
Section "InputDevice"
        Identifier  "PS2Keyboard"
        Driver      "evdev"
        Option      "XkbModel" "evdev"
        Option      "Name" "AT Translated Set 2 keyboard"
#       Option      "Device" "/dev/input/event31"
EndSection
```

Ve výše uvedeném je skryta jedna neobratnost ovladače `evdev` – ovladač se snaží rozpoznávat zařízení na sběrnicích sám (podobně jako to dělá `udev`). Proto zde nemáme cestu ke speciálnímu souboru, ale textové jméno (stejné je v atributech zařízení v SysFS).

Pro myši je situace ještě jednodušší: Linux poskytuje pro každou myš kromě event interface také zařízení `/dev/input/mouseN`, které používá protokol kompatibilní s myší IMPS/2. Příslušná část `xorg.conf` bude následující:

```
Section "InputDevice"
        Identifier  "PS2Mouse"
        Driver      "mouse"
        Option      "Protocol" "IMPS/2"
        Option      "Device" "/dev/input/mouse_first"
        Option      "Buttons" "5"
        Option      "ZAxisMapping" "4 5"
EndSection
```

5.3 Dokončení konfigurace

Nyní zbývá už jen v `xorg.conf` definovat příslušné sekce pro monitory, spojit je spolu se sekcemi `Device` do sekcí `Display` (zde se nastavují věci jako podporovaná rozlišení a bitové hloubky) a to vše na závěr spojit se vstupními zařízeními do popisu rozložení obrazovek, tedy sekce `ServerLayout`:

```
Section "ServerLayout"
        Identifier     "ATiLayout"
        Screen      0  "ATI+LGLCD" 0 0
        InputDevice    "PS2Mouse" "CorePointer"
        InputDevice    "PS2Keyboard" "CoreKeyboard"
EndSection

Section "ServerLayout"
        Identifier     "RivaLayout"
        Screen      0  "Riva+PhilipsLCD" 0 0
        InputDevice    "USBMouse" "CorePointer"
        InputDevice    "USBKeyboard" "CoreKeyboard"
EndSection
```

6 Správa síťových rozhraní

Připojení k síti přináší na desktopových a zejména mobilních systémech řadu problémů, které jsme z UNIXových serverů dosud neznali. Například konfigu-

race sítě by měla být co nejvíc automatická, nanejvýš s částečnou asistencí právě přihlášeného uživatele (ne nutně superuživatele). Uživatel by měl mít možnost vybrat si z několika slyšitelných WiFi sítí, nejlépe tak, aby zároveň viděl například i informaci o síle signálu jednotlivých dostupných sítí. Podobně konfigurace VPN, šifrovaného přístupu a podobně.

Informace o aktuálním stavu síťového připojení a jeho vlastnostech by ale měla také být k dispozici i ostatním uživatelovým aplikacím: webový prohlížeč se například může na základě typu připojení rozhodnout nestahovat obrázky, uživatel může chtít automaticky spouštět svého oblíbeného klienta pro sdílení multimediálních dat jen v určité síti, a tak podobně.

Aplikace která řeší výše uvedené problémy pod Linuxem se jmenuje NetworkManager[11]. Jedná se o systémového démona, který přes HAL zjišťuje informace o síťových zařízeních a přes D-Bus komunikuje s uživatelským appletem a dalšími aplikacemi, které se o aktuální stav síťového připojení zajímají.

7 Uspávání a hibernace

Pro přenosné počítače a další mobilní zařízení je klíčová schopnost ušetřit napájení tím, že se počítač do jisté míry deaktivuje. Obvykle rozeznáváme dva druhy deaktivace: v prvním (*suspend*) je vypnuto vše kromě paměti RAM počítače a několika málo dalších komponent. Druhý (*hibernate*) znamená úplně vypnutý počítač. Pro pozdější obnovení stavu je u hibernace uložen obsah paměti RAM a další informace na disk počítače.

7.1 Suspend

Suspend (uspání do RAM) je složitější ze dvou způsobů uspávání. Vyžaduje součinnost firmwaru počítače (ACPI) a nese s sebou další problémy, zejména při opětovné aktivaci dříve vypnutých zařízení. Asi nejproblematičtější bývají grafické karty, protože ty za normálních okolností bývají po zapnutí počítače inicializovány přímo BIOSem. Proces probouzení z RAM proto může vyžadovat některé úpravy podle typu hardwaru a chování BIOSu[12].

7.2 Hibernace

Hibernace je jednodušší. Funguje zjednodušeně řečeno tak, že se vytvoří obraz paměti systému, uloží na disk (do odkládacího prostoru), a při obnovení se nabootuje docela obyčejné jádro, které běžným způsobem nainicializuje svá zařízení, ale pak místo startu uživatelského prostoru načte a obnoví obraz paměti z odkládacího prostoru.

[11] http://www.gnome.org/projects/NetworkManager/
[12] http://people.freedesktop.org/~hughsient/quirk/quirk-suspend-try.html

7.3 Balík `pm-utils`

Aktuální vývoj ve správě hibernace a suspendu je balík `pm-utils`[13]. Sestává se z hlavního skriptu `pm-action`, volaného pomocí symlinku jako `pm-suspend` nebo `pm-hibernate`. Tento skript spouští jednotlivé úlohy, které je třeba provést před uspáním počítače (pro hibernaci například aktivace stejného jádra v boot loaderu, jako je právě běžící jádro, atd.). A podobně má i skripty pro obnovení provozu po probuzení.

Program `pm-action` je volán obvykle z HAL na pokyn některého desktopového appletu (`gnome-power-manager` nebo `kpowersave`).

8 Sdílení zvukového subsystému

Problematika zvukového hardwaru je z hlediska jádra dostatečně zvládnuta, jádro zpřístupňuje vlastnosti zvukového hardwaru do uživatelského prostoru. Problém ovšem je v přístupu více aplikací ke zvukové kartě. Ne všechny zvukové karty podporují hardwarové mixování více zvukových proudů, a tak se může stát, že pokud zrovna přehráváte video, nikdo se nedovolá na vašeho VoIP klienta, protože neuslyšíte vyzvánění.

Situaci se snaží řešit systém PulseAudio[14], dříve známý pod názvem PolypAudio. Jde o přenesení zprávy zvukového zařízení do samostatného démona a úpravu aplikací (aplikačních knihoven) tak, aby místo přímého přístupu k zařízení komunikovaly s tímto démonem. Podobný přístup známe z projektů EsounD a aRts (oba jsou dnes již nevyvíjené).

PulseAudio je reimplementací aplikačního rozhraní EsounD s tím, že obslužný démon má modulární architekturu a daleko více možností. Kromě EsounD protokolu poskytuje i nativní protokol včetně práce nad TCP/IP. Takže například je možno výstup zvukové aplikace posílat po síti na jiný počítač.

PulseAudio řeší věci jako softwarové mixování, ale také nastavení hlasitosti: pro každý proud zvukových dat se eviduje název (daný aplikací nebo právě přehrávanými daty) a k názvu se eviduje naposledy použitá hlasitost. Takže můžete mít jinou hlasitost pro VoIP klienta a jinou pro přehrávač hudby, aniž byste museli měnit nasavení hlasitosti pokaždé, když dotelefonujete.

Systém PulseAudio dokonce umí definovat akce pro jednotlivé proudy zvukových dat, takže například je možno ztišit přehrávač hudby při příchodu VoIP hovoru a podobně. Systém umožňuje funkci i bez trvale spuštěného démona tak, že instanci démona spouští ta aplikace, která poprvé v rámci sezení chce používat zvukové zařízení. Uživatel navíc může existující proudy dat přesměrovávat na různá zařízení nebo i po síti.

[13] http://en.opensuse.org/Pm-utils
[14] http://www.pulseaudio.org/

PulseAudio je asi nejvíc přenositelnou komponentou ze všech v tomto textu zmiňovaných. Funguje kromě Linuxu například i na Solarisu, FreeBSD a dokonce existuje i nativní (ne Cygwin) port pro Windows.

9 Informace o zdrojích v síti

Jednou z často vyžadovaných vlastností desktopových systémů je, aby takovéto systémy fungovaly bez centrální evicence nebo centrálních serverů. Aby se dokázaly domluvit dva počítače jen na základě toho, že jsou na stejné síti.

Jedním z protokolů pro takovéto bezkonfigurační (zeroconf) komunikace je multicast-DNS (mDNS) a DNS service discovery (DNS-SD). Oba tyto protokoly jsou na desktopech pod Linuxem implementovány balíkem Avahi[15].

Pomocí Avahi mohou aplikace vidět okolní počítače jménem i bez DNS a dokonce i bez centrální autoritou přidělené IP adresy. Avahi ale také poskytuje informace o službách na daných počítačích běžících, takže aplikace mohou přes Avahi znát podobné aplikace na jiných strojích v síti. Avahi komunikuje s aplikacemi přes D-Bus, aplikace mohou publikovat své vlastní zdroje nebo se dotazovat na cizí zdroje. Například VoIP klient Ekiga může tímto způsobem vidět připojená SIP nebo H.323 zařízení ve stejné síti. Podobně lze vidět například seznam SSH serverů nebo veřejných datových archívů.

Zajímavým využitím Avahi je malá aplikace `gnome-user-share`. Umožňuje uživatelům velmi jednoduchým způsobem vystavovat některé soubory pro stažení na jiné počítače: uživatel soubory umístí do svého adresáře `~/Public`, a spustí tuto aplikaci. `gnome-user-share` pak má na starosti spuštění kopie Apache s minimální konfigurací na některém neprivilegovaném portu a publikování informací o nově vzniklém zdroji dat přes Avahi. Ostatní uživatelé mohou ihned vidět nový adresář, ve kterém jim jsou nabízena data. Samotné `gnome-user-share` je pěkný příklad modularity UNIXu: aplikace má asi 80 KB včetně několika katalogů zpráv. Většina činností je realizována externími nástroji: Apache a Avahi.

Systém Avahi běží kromě Linuxu i pod *BSD systémy a Solarisem.

10 Je Linux připravený na desktop?

Cílem tohoto příspěvku bylo ukázat, kam dospěla v posledních letech podpora desktopových prostředí ze strany jádra Linuxu a některých dalších systémových programů. Přestože v některých oblastech jako je dostupnost specifikací hardwaru grafických karet nebo bezdrátových síťových karet situace stále není optimální, postupně i zde dochází ke zlepšení.

[15] http://avahi.org/

Na druhou stranu v mnoha jiných aspektech přináší modularita UNIXu spolu s orientací části vývojářů Linuxu na desktopové systémy již mnohé zajímavé výsledky.

Projekt OpenSolaris aneb co se uvařilo u Sunů

Martin Man

E-mail: mman@martinman.net

Abstrakt

Operační systém Solaris byl pro mnohé z nás prvním UNIXem na kterém jsme se učili pracovat v devadesátých letech minulého století. I přes to že dal celému světu spoustu technologií, mnohé již v minulém století jako otevřené standardy, jeho slibný vývoj byl téměř zlikvidován samotnou firmou Sun, která se rozhodla že jej přestane vyvíjet na platformě Intel, v té době již masově rozšířené, a která nebyla schopna přilákat dostatek mladých inženýrů, v té době se věnujících zejména vývoji stále populárnějšího Linuxu. Zdá se, že vše se v dobré obrátilo, a drtivá většina zdrojového kódu Solarisu 10 byla na jaře roku 2005 uvolněna pod licencí CDDL veřejnosti jako opensource. Co toto uvolnění přineslo? Jak se Sun vypořádal s OpenSolarisem a opensource? Kde se dějí opravdové inovace? Proč se o OpenSolarisu tolik mluví?

Projekt OpenSolaris

Rozhodnutí uvolnit zdrojový kód Solarisu jako opensource padlo uvnitř Sunu někdy v roce 2003. Nebylo to rozhodnutí jednoduché. Pro mladší generaci odrostlou na Linuxu a jeho otevřeném vývoji může být zajímavé i to, že přesvědčovat bylo třeba jak kruhy manažerské a víceprezidentské, tak kruhy inženýrské. Víceprezidenti chtěli vidět zisk a inženýři si chtěli uchránit své dlouhá léta zdokonalované know-how na které byli právem hrdí.

Následovaly celé dva roky práce, během nichž se procházely postupně všechny zdrojové soubory a zjišťovalo se pod jakou licencí je Sun získal a jestli je možné je uvolnit. Výsledkem této snahy bylo spuštění portálu http://opensolaris.org [OSOL] ke kterému došlo přesně 16. června 2005. Převážná část zdrojového kódu Solarisu 10 byla uvolněna pod licencí CDDL [CDDL] veřejnosti.

Volba jakou licenci použít nebyla též jednoduchá. Nábožensky fanatičtí zastánci Free Software Foundation [FSF] a licence GPL [GPL] chtěli, a pravděpodobně stále chtějí aby byl OpenSolaris k dispozici pod GPL. Pragmatický Sun

spolu se svým syndromem NIH – „not invented here“ zvolil licenci založenou na Mozilla Public License a nazval ji CDDL – Common Development and Distribution License [CDDL]. Volba licence nebyla v pravdě nejvhodnější, CDDL licence samotná je v pořádku, nicméně Sun tímto krokem získal v očích (určité) části veřejnosti obraz firmy, která to (opět) s opensource nemyslí vážně.

Uvolňování zdrojového kódu probíhalo a stále ještě stále probíhá ve fázích. Jednotlivé komponenty, které je možné uvolnit pod licencí CDDL se uvnitř Sunu musí nejprve připravit pro distribuci, což většinou znamená přidat do všech zdrojových souborů licenci a celek upravit tak aby se dal zkompilovat bez speciálních Sunovských nástrojů.

Proč by mne to mělo zajímat

Mnoho lidí se domnívá, že ve světě plném Linuxových distribucí určených pro nejrůznější účely, a ve světě další spousty alternativních otevřených operačních systémů není místo pro další otevřený UNIXový systém. Skutečnost je taková, že OpenSolaris se rozhodně nemusí za své vlastnosti stydět a že má světu co nabídnout:

- ZFS [ZFS] – souborový systém, který je snem každého administrátora který měl kdy co do činění s hardwarovými RAID poli nebo Linuxovými systémy LVM a EVMS. Sun sám používá pro ZFS marketingový slogan: *ZFS: the last word in filesystems.*

- DTrace [DTRC] – dynamická instrumentace kernelu a uživatelských aplikací. Naprosto nepostradatelný nástroj pro každého vývojáře a administrátora. Nová generace nástrojů jako `strace(1)` nebo `truss(1)`. DTrace byl již úspěšně portován na FreeBSD a MacOS/X.

- Enterprise class kernel – OpenSolaris je stavěný pro silné serverové stroje a poskytuje na nich naprosto unikátní vlastnosti. FMA – fault management architecture, výměnu hardwarových komponent za běhu (včetně CPU a paměti), a další.

- Zones a BrandZ [ZONES] – nástupce příkazu chroot(8) který dokáže implementovat mimo jiné syscall API Linuxu 2.4. Velmi efektivní a jednoduchá virtualizace podobná Xenu.

Kromě technologických inovací může být zajímavý i samotný vývojový model OpenSolarisu a jeho klíčové vlastnosti:

- binární kompatibilita – Sun klade maximální důraz na zachování zpětné binární kompatibility, tato vlastnost byla samozřejmě přenesena i do Open-

Solarisu. Mimo jiné zajišťuje, že ovladače zařízení od třetích stran budou fungovat i do budoucna bez nutnosti je rekompilovat.

- stabilita API – každá funkce API operačního systému má garantovanou úroveň stability tak aby vývojáři věděli na co se spolehnout a na co raději ne.

- kontrolované přidávání nových vlastností – než je možné začlenit novou vlastnost do jádra, je potřeba detailně zdokumentovat jaký vliv to bude mít na zbytek systému, jaké nové příkazy a API budou k dispozici. To vede k relativně kontrolovanému a dokumentovanému růstu celého OpenSolarisu.

OpenSolaris na druhé straně částečně zaostává na straně desktopu. Většina současných desktopových inovací je vyvíjena komunitou na Linuxových systémech a je závislá na některých vlastnostech Linuxového jádra. Ty musí být později portovány na OpenSolaris a ačkoli není jednoduché tímto způsobem udržet vývoj s Linuxem, již nyní je možné provozovat na OpenSolarisu například 3D desktop založený na Compiz Fusion [COMPIZ].

Komunita versus Sun Microsystems

Komunita lidí pracujících na jakémkoli otevřeném softwaru má v současné době určitá očekávání. Jakým způsobem tato očekávání Sun s projektem OpenSolaris naplňuje má velmi velký vliv na to jak bude komunita Sun vnímat a jak moc bude ochotna přijmout projekt za vlastní a věnovat mu svůj čas.

- Dostupnost zdrojového kódu: nestačí pouze vystavit `.tar.gz` archivy na web. Je důležité mít efektivní a otevřený systém pro správu verzí, který umožní všem zájemcům stejný pohled na změny kódu a jeho případné modifikace. Projekt OpenSolaris se po dlouhém výběru rozhodl pro nástroj Mercurial [HG] a v současné době dochází k migraci zdrojových kódů do nových repozitářů z původního Sunem používaného nástroje TeamWare [TW].

- Otevřené závislosti pro kompilaci kódu: OpenSolaris jde momentálně zkompilovat pouze na oficiální distribuci OpenSolarisu od Sunu [SXDE]. K úspěšné kompilaci je nutné mít několik komponent dostupných pouze v binární podobě.

- Otevřený systém pro správu chyb: Sun používá pro správu chyb interní komerční software na nějž byl napojen systém http://bugs.opensolaris.org. Nejedná se o cílový stav a v současné době se pracuje na otevřeném systému

pro správu chyb OpenSolarisu, který by bylo možné efektivně napojit na interní Sunovský systém.

- Rozhodovací pravomoce: Nestačí pouze „házet zdrojáky přes zeď“ jak někteří vtipně komentují vývoj některých částí OpenSolarisu. Komunita chce mít pocit vlastnictví kódu a možnost rozhodovat o jeho budoucnosti. Díky licenci CDDL a díky celému projektu OpenSolaris se toto naštěstí děje a komunita vývojářů a uživatelů, jejíž aktivní majorita jsou v současné době hlavně zaměstnanci Sunu, si právem vydobývá své místo na slunci a daří se jí velmi efektivně ovlivňovat rozhodnutí korporátního Sunu.

Opět je nutné podotknout, že ne všechny vývojové týmy Sunu jsou zvyklé pracovat na otevřených systémech a ne vždy dodržují pravidla, která by komunita ráda viděla. Celkově to vede k určitému napětí mezi uživateli a elitářskými vývojáři. Toto napětí doufejme postupem času vymizí spolu s tím jak se jednotlivé součásti OpenSolarisu naučí být opravdu „otevřené“.

Současnost a budoucnost

V současné době je projekt OpenSolaris velmi živý. V Emailových konferencích se žhavě diskutují problematické body OpenSolarisu jakými v současnosti jsou zejména:

- Package Management: V době kdy každá linuxová distibuce má svůj klon `apt-get(1)` Ian Murdock (zakladatel projektu Debian/GNU Linux) s dalšími týmy pracuje na projektu Indiana, který mimo jiné přinese nástupce `apt-get(1)` pro OpenSolaris.

- Správa verzí Mercurial: veřejný repozitář zdrojových kódů spravovaný systémem Mercurial bude oficiálním a jediným repozitářem kódu ze kterého Sun bude brát zdrojové kódy které se stanou oficiálním Solarisem.

- Otevřený systém správy chyb: Probíhá zjišťování stavu a výběr kandidáta pro oficiální nástroj pro správu chyb OpenSolarisu.

Samotný OpenSolaris je pouze jádrem operačního systému a sadou základních nástrojů. Vytvořit z něj distribuci ve smyslu Linuxového systému je možné mnoha způsoby a v současné době je k dispozici několik variant OpenSolarisu, jejichž hlavní představitelé jsou:

- Solaris Express Developer Edition [SXDE] – distribuce vytvořená Sunem založená na OpenSolarisu a obsahující standartně nástroje jako SunStudio a NetBeans.

- BeleniX [BELNX] – komunitní live-cd distribuce pro snadné spuštění OpenSolarisu z USB sticku nebo CD.
- Nexenta [NEXENTA] – port uživatelského prostředí a aplikací Ubuntu a Debian/GNU Linuxu na OpenSolaris jádro.

Jak se zapojit do projektu OpenSolaris

Kromě celé řady emailových konferencí a skupin uživatelů OpenSolarisu po celém světě je možné zapojit se do vývoje jedné z distribucí OpenSolarisu.

Česká skupina uživatelů OpenSolarisu – [CZOSUG] pořádá pravidelně setkání a přednášky na půdě MFF UK a ČVUT v Praze a nepravidelně pak též na univerzitách v Plzni a Brně.

Na stráknách http://opensolaris.org je možné objednat si *OpenSolaris Starter Kit*, což je sada dvou DVD obsahující instalační a live CD všech v současnosti dostupných distribucí OpenSolarisu.

Reference

[OSOL] http://opensolaris.org

[CDDL] http://en.wikipedia.org/wiki/Common_Development_and_Distribution_License

[FSF] http://www.fsf.org

[GPL] http://en.wikipedia.org/wiki/GNU_General_Public_License

[CZOSUG] http://cz.opensolaris.org

[ZFS] http://en.wikipedia.org/wiki/ZFS

[DTRC] http://en.wikipedia.org/wiki/DTrace

[ZONES] http://www.opensolaris.org/os/community/zones/

[COMPIZ] http://www.compiz-fusion.org/

[HG] http://selenic.com/mercurial

[TW] http://docs.sun.com/source/806-3573/intro.html

[BELNX] http://www.belenix.org

[SXDE] http://developers.sun.com/sxde/

[NEXENTA] http://nexenta.org

FreeBSD

Tomáš Kraus

E-mail: tomas.kraus@sun.com

FreeBSD a možnosti jeho uplatnění

FreeBSD se velmi dobře uplatní všude, kde je potřeba provozovat spolehlivé a dobře zabezpečené aplikace a síťové služby. Jedná se zejména o software, který je dostupný v rámci Open Source:

- DNS/DHCP
- SMTP+IMAP
- WWW server (Apache, PHP)
- Directory server (Open LDAP)
- Windows server (Samba)
- Aplikační server (Tomcat, jBoss)
- Databázový server (MySQL, PostgreSQL)

Bohužel, komerční dodavatelé software tento operační systém víceméně ignorují a je téměř nemožné ho využít pro velké enterprise systémy. Přitom v porovnání například s komerčně mnohem úspěšnějším Linuxem se FreeBSD jeví jako stabilnější a lépe zabezpečené.

Naštěstí se Sun Miocrosystems jako jeden z mála rozhodl dát k dispozici Javu a Glassfish jako Open Source a snad proto dojde k zlepšení alespoň v oblasti Javy a aplikačních serverů.

Nevýhodou FreeBSD již mnoho let zůstáva absence administračních nástrojů pro méně zkušené uživatele. Stále je to systém pro úzkou skupinu odborníků. Jisté rezervy se skrývají i v modelu distribuce – vlastní systém je oddělený od aplikačního software a použitý package management rozhodně nepatří k těm nejlepším.

Pro nezkušeného uživatele je FreeBSD téměř nepoužitelný i jako desktop systém. Na druhé straně při potřebných znalostech není problém z FreeBSD udělat desktop, který je SW vybavením srovnatelý s Linuxem. Většina Open Source desktop aplikací (včetně např. Open Office, Gimpu, Firefoxu, atd.) zde funguje a k jejich zprovoznění stačí nainstalovat potřebné balíky.

Využití ve firmě či jiné organizaci

V rámci malé i velké firmy (či jiné organizace) může FreeBSD sloužit pro provoz výše uvedených služeb. Z praxe uvedu několik příkladů:

- ICZ, a. s. – dlouhou dobu zde bězely následující služby na FreeBSD:
  - SSH access point z internetu
  - SMTP/IMAP server
  - DNS
- Česká televize – několik let provozovala http://www.czech-tv.cz na FreeBSD
- MVČR – http://www.mvcr.cz

Před dvěma lety jsem společně se dvěma kolegy založil malou firmu, která se specializuje na vývoj a provozování různých internetových služeb (WWW, e-mail, atd). Tyto služby rovněž provozujeme na FreeBSD.

Abychom mohli jednotlivé služby provozovat na společném hardware a operačním systému při rozumné míře zabezpečení, museli jsme každnou službu uzavřít do vlastního zabezpečeného prostoru – jailu.

Pro zabezpečení dat mimo jiné využíváme SW RAID subsystémy vinum a geom.

Jail

Jail představuje virtuální prostor v operačním systému, z kterého je zamezen přístup k ostatním částem systému. Tento prostor má k dispocici pouze jedinou IP adresu.

Jail lze přirovnat například k Java sandboxu, případně k zónám v Solaris 10. Na rozdíl od výrazně propracovanější implementaci zón v Solarisu zde nelze řídit přidělování některých systémových prostředků (např. CPU) a inicializaci a konfiguraci jailu je nutné provést ručně.

Jail se obvykle používá pro:

1. Uzavření jediné aplikace (často běžící s nadstandardními právy), které se takto zamezí v přístupu k ostatním částem ststému
2. Vytvoření virtuálního systému pro provoz více služeb a aplikací, které je potřeba oddělit od ostatních částí systému.

Konfigurace

1. Instalace kopie OS
2. Vytvoření devfs a procfc
3. Přidělení vlastní IP adresy
4. Konfigurace parametrů jail_* v /etc/rc.conf

Instalace kopie OS Běžný postup buildu a instalace OS z CVS:

- `make buildworld`
- `make installworld DESTDIR=<jail_root>`
- `mergemaster`

Pro potřeby jailu obvykla stačí minimální konfigurace:

```
/etc/make.conf
CFLAGS=-O -pipe
COPTFLAGS=-O -pipe

NO_ACPI=        true    # do not build acpiconf(8) and related programs
NO_BOOT=        true    # do not build boot blocks and loader
NO_BLUETOOTH=   true    # do not build Bluetooth related stuff
NO_FORTRAN=     true    # do not build g77 and related libraries
NO_GDB=         true    # do not build GDB
NO_GPIB=        true    # do not build GPIB support
NO_I4B=         true    # do not build isdn4bsd package
NO_IPFILTER=    true    # do not build IP Filter package
NO_PF=          true    # do not build PF firewall package
NO_AUTHPF=      true    # do not build and install authpf (setuid/gid)
NO_KERBEROS=    true    # do not build and install Kerberos 5 (KTH Heimdal)
NO_LPR=         true    # do not build lpr and related programs
NO_MAILWRAPPER=true     # do not build the mailwrapper(8) MTA selector
NO_MODULES=     true    # do not build modules with the kernel
NO_NETCAT=      true    # do not build netcat
NO_NIS=         true    # do not build NIS support and related programs
NO_SENDMAIL=    true    # do not build sendmail and related programs
NO_SHAREDOCS=   true    # do not build the 4.4BSD legacy docs
NO_USB=         true    # do not build usbd(8) and related programs
NO_VINUM=       true    # do not build Vinum utilities
NO_ATM=         true    # do not build ATM related programs and libraries
#NO_CRYPT=       true    # do not build any crypto code
NO_GAMES=       true    # do not build games (games/ subdir)
NO_INFO=        true    # do not make or install info files
NO_MAN=         true    # do not build manual pages
NO_PROFILE=     true    # Avoid compiling profiled libraries
```

```
# BIND OPTIONS
NO_BIND=                true    # Do not build any part of BIND
NO_BIND_DNSSEC=         true    # Do not build dnssec-keygen, dnssec-signzone
NO_BIND_ETC=            true    # Do not install files to /etc/namedb
NO_BIND_LIBS_LWRES=     true    # Do not install the lwres library
NO_BIND_MTREE=          true    # Do not run mtree to create chroot directories
NO_BIND_NAMED=          true    # Do not build named, rndc, lwresd, etc.
```

Vytvoření devfs a procfs

- `mount -t devfs <jail_root>/dev`
- `mount -t procfs proc <jail_root>/proc`

Lze nastavit pomocí konfiguračních parametrů v /etc/rc.conf:

- `jail_<název>_devfs_enable="YES"`
- `jail_<název>_procfs_enable="YES"`

Přidělení vlastní IP adresy Z bezpečnostních důvodů se doporučuje využívat loopback interface.

```
ifconfig lo0 inet alias 127.0.0.2 255.255.255.255}
```

Lze nastavit pomocí konfiguračních parametrů v /etc/rc.conf:

```
ifconfig\_lo0\_alias0="inet 127.0.0.2 netmask 255.255.255.255"
```

Konfigurace parametrů v /etc/rc.conf

```
jail_enable="YES"
jail_set_hostname_allow="NO"
jail_sysvipc_allow="YES"
jail_list="httpd"
jail_httpd_hostname="www.localdomain"
jail_httpd_ip="127.0.0.2"
jail_httpd_rootdir="/jail/httpd"
jail_httpd_devfs_enable="YES"
```

Přesměrování TCP/IP na privátní adresu Příchozí TCP/IP spojení je nutno přesměrovat na IP adresu a port loopback interface:

```
rdr on fxp0 proto tcp from any to 123.123.123/32 port  80 -> 127.0.0.2 port 80
rdr on fxp0 proto tcp from any to 123.123.123/32 port 443 -> 127.0.0.2 port 443
```

A případně povolit příchozí HTTP/HTTPS požadavky v IP filtru.

```
pass in quick on fxp0 inet proto tcp from any to 127.0.0.2 port 80
pass in quick on fxp0 inet proto tcp from any to 127.0.0.2 port 443
```

Pravidla jsou ve formátu pf. Jejich podoba samozřejmě závisí na použitém IP filtru.

Příklad použití

Oddělení služeb bežících na společném OS:

- IMAP serveru (veřejná IP)
- WWW serveru (veřejná IP)
- Relační databáze
  - MySQL (lokální IP – 127.0.0.2)
  - PostgreSQL (lokální IP – "127.0.0.3)

Každá aplikace má vlastní jail. Minimální kopie OS je sdílená na readonly FS.

Ostatní části jsou mountované jako read/write:

- /etc
- /tmp
- /var
- /home
- /root
- /usr/local
- /usr/X11R6
- /usr/ports/distfiles

```
/etc/fstab
/jail/template/mroot    /jail/httpd     nullfs  ro      0       0
/jail/template/mroot    /jail/mysql     nullfs  ro      0       0
/jail/template/mroot    /jail/pgsql     nullfs  ro      0       0
/jail/template/mroot    /jail/smtp      nullfs  ro      0       0

/jail/template/httpd    /jail/httpd/s   nullfs  rw      0       0
/data/www               /jail/httpd/www nullfs  rw      0       0
/jail/template/mysql    /jail/mysql/s   nullfs  rw      0       0
/data/mysql             /jail/mysql/db  nullfs  rw      0       0
/jail/template/pgsql    /jail/pgsql/s   nullfs  rw      0       0
/data/pgsql             /jail/pgsql/db  nullfs  rw      0       0
/jail/template/smtp     /jail/smtp/s    nullfs  rw      0       0
```

Všechny read/write adresáře jsou ve společném prostoru `/jail/<název>/s`. Jejich struktura je do společné readonly části namapována pomocí symbolických linků.

```
>ls -l /jail/template/mroot | grep lrwx
lrwxr-xr-x   1 root  wheel     5 Jul 14 21:48 etc@ -> s/etc
lrwxr-xr-x   1 root  wheel     6 Jul 14 21:48 home@ -> s/home
lrwxr-xr-x   1 root  wheel     6 Jul 14 21:48 root@ -> s/root
lrwxr-xr-x   1 root  wheel    11 Jul 14 21:29 sys@ -> usr/src/sys
lrwxr-xr-x   1 root  wheel     5 Jul 14 21:48 tmp@ -> s/tmp
lrwxr-xr-x   1 root  wheel     5 Jul 14 21:48 var@ -> s/var

>ls -l /jail/template/mroot/usr | grep lrwx
lrwxr-xr-x   1 root  wheel    14 Jul 14 21:48 X11R6@ -> ../s/usr-X11R6
lrwxr-xr-x   1 root  wheel    14 Jul 14 21:48 local@ -> ../s/usr-local

>ls -l /jail/template/mroot/usr/ports | grep lrwx
lrwxr-xr-x   1 root  wheel    17 Jul 14 21:48 distfiles@ -> ../../s/distfiles

/etc/rc.conf
ifconfig_lo0_alias0="inet 127.0.0.2 netmask 255.255.255.255"
ifconfig_lo0_alias1="inet 127.0.0.3 netmask 255.255.255.255"

jail_enable="YES"
jail_set_hostname_allow="NO"
jail_sysvipc_allow="YES"
jail_list="smtp mysql pgsql httpd"
jail_mysql_hostname="mysql.localdomain"
jail_mysql_ip="127.0.0.2"
jail_mysql_rootdir="/jail/mysql"
jail_mysql_devfs_enable="YES"
jail_pgsql_hostname="pgsql.localdomain"
jail_pgsql_ip="127.0.0.3"
jail_pgsql_rootdir="/jail/pgsql"
jail_pgsql_devfs_enable="YES"
jail_httpd_hostname="www.example.org"
jail_httpd_ip="123.123.123.123"
jail_httpd_rootdir="/jail/httpd"
jail_httpd_devfs_enable="YES"
jail_smtp_hostname="mail.example.org"
jail_smtp_ip="123.123.123.123"
jail_smtp_rootdir="/jail/smtp"
jail_smtp_devfs_enable="YES"
```

SW RAID pomocí vinum a geom

Za zmínku stojí zejména starší vinum a jeho nástupce geom. Ve FreeBSD existují i další, například implementace zařízení ar (ATA RAID), které implementuje RAID 0, 1 a 0+1 pro ATA/SATA disky.

Logical volume manager Vinum

Až do verze 5.X se jednalo o nejlepší dostupnou implementaci SW RAIDu, která byla inspirována Veritas volume managerem. Ve verzi 6.X došlo k jeho nahrazení

novou implementací gvinum v rámci GEOM frameworku. Ta má ale ke kvalitám původní implementace ještě daleko.

Vinum zvládá běžné typy RAIDu:

- RAID-0 (Strip)
- RAID-1 (Mirror)
- RAID-5

A samozřejmě také spojení (concatenation) několika (různě velkých) částí disků.

Jednotlivé typy lze samozřejmě kombinovat (např. RAID 0+1).

GEOM framework

GEOM framework byl přidán do FreeBSD verze 5. Umožňuje snadné vytváření implementací různých diskových operací, např.:

- RAID
- concatenation
- vzdálený přístup k zařízením
- šifrování dat

Jednotlivé implementace lze libovolně kombinovat a vytvořit například RAID-5 pole na několika různých zašifrovaných partitions, nebo naopak jeden zašifrovaný prostor na RAID-5 poli.

Implementace GEOM frameworku:

- gconcat – concatenation
- gstripe – RAID-0
- gmirror – RAID-1
- graid3 – RAID-3
- graid5 – RAID-5 (ve vývoji, není zatím v distribuci)
- gvinum – implementace vinum (RAID 0, 1, 5, concatenation)
- gbde – Geom Based Disk Encryption
- gshsec – Shared Secret Devices
- ggatec – přístup ke vzdáleným zařízením (nahrazuje např. NFS)

GEOM nabízí určitě mnohem více možností, než například LVM v Linuxu. Na rozdíl od Linuxu ale není k dispozici žádný nástroj pro snadnou konfiguraci. Správu disků je nutno dělat ručně pomocí command-line utilit.

IRC Servery

Od roku 1994 provozujeme v České republice službu IRC (Internet Relay Chat). Po počátečních experimentech na Sun SparcStation 1/SunOS, HP 9000/HP-UX a Intel P166/Linux jsme přešli na FreeBSD 3-STABLE a od toho okamžiku jsme u tohoto OS zustali.

Požadavky na IRC server:

- Provoz 24/7 365 dnů v roce (uptime serveru řádově stovky dnů)
- Velké síťové zatížení
- tisíce otevřených socketů
- stálý datový tok řádově stovky kbps
- nárazově jednotky až desítky Mbps (při rekonfiguraci sítě)
- Vysoký stupeň zabezpečení

FreeBSD se v tomto ohledu projevilo jako velmi dobrá volba:

- Dobře snáší DoS útoky hrubou silou.
- Do dnešního dne jsme nezaznamenali úspěšný útok hackerů.
- Systém bez problémů snáší dlouhodobý provoz (uptime až 500 dnů)

Servery pro žáky ZŠ

Ve dvou základních školách jsme zajišťovali provoz infrastruktury pro výuku. Pracovní stanice byly PC s Windows 2000/XP. Ostatní služby běžely na dvojici serverů s FreeBSD 5.X.

Požadavky na provoz:

- Základní síťové služby:
  - Interní DHCP
  - Interní DNS
  - SMTP/IMAP
  - WWW Server
  - Firewall + NAT
- Náhrada nespolehlivého Windows serveru

- Minimální pořizovací a provozní náklady
- Vysoká spolehlivost
- Minimální nároky na údržbu

Služby jsme rozdělili na dvojici serverů s OS FreeBSD 5-STABLE. První server zajišťoval základní síťové služby DNS + DHCP a sloužil zároveň jako firewall s podporou NAT.

Druhý server sloužil jako Primary Domain Controler pro windows doménu, dále jako file server pro pracovní stanice a zároveň jako WWW server pro intranet.

Server pro síťové služby

- DHCP: ISC DHCP server 3.0.X
- DNS: BIND z FreeBSD distribuce
- SMTP gateway: Postfix 2.2.X (pouze oddělení intranetu)
- FW+NAT: ipf + ipnat z FreeBSD distribuce
- Uvažovali jsme i o http proxy (Squid 2.6)

Server měl 2 síťové karty pro internet a intranet. Provoz na internetovém rozhraní byl omezen pravidly IPF.

Na internetovém rozhraní byla aktivní pouze dvojice služeb:

- SMTP na portu 25
- SSH na portu 22 omezená na cca 5 IP adres (vyžadovala autentizaci DSA klíčem)

Celá vnitřní síť pak byla pomocí NAT mapována na IP adresu internetového rozhraní.

Server zároveň zajišťoval přidělování IP adres (DHCP) a DNS pro intranet.

Intranet Server

- Samba 2.2.X jako PDC
- WWW: Apache 2.0.X + PHP 5.2.X
- Postfix 2.2.X + Cyrus IMAP 2.2.X

Server nahradil původní Windows 2000 server ve funkci PDC a file serveru. Zároveň jsme umožnili žákům vyšších ročníků experimentovat s HTML a PHP při návrhu vlastních webových stránek.

Diskový prostor pro fileserver byl vytvořen jako SW RAID 5 pole pomocí logical volume manageru Vinum.

Správa těchto serverů zabrala řádově jednotky hodin měsíčně a spočívala pouze v pravidelné aktualizaci operačního systému a instalovaných aplikací. Pouze na začátku školního roku bylo nutno aktualizovat uživatelské účty pro příchozí a odchozí žáky.

Servery i pracovní stanice jsme bez problémů dokázali provozovat za $\frac{1}{2}$ ceny, kterou si účtovali předešlí provozovatelé v rámci projektu Internet do škol.

Uživatelské prostředí v prodejní síti Baťa, a. s.

Norbert Volf

E-mail: volf@bata.cz

Úvod

V tomto příspěvku se budu snažit popsat implementaci uživatelského prostředí pomocí GNU/Linuxu, která proběhla v naší společnosti v roce 2004.

Od roku 2002 probíhalo v naší společnosti hledání nového pokladního systému, který slouží k přenosu informací o prodejích z jednotlivých obchodních míst na centrálu do účetního a komerčního systému. Ve výběrovém řízení zvítězil program, který běží v prostředí operačního systému GNU/Linux. Tento výběr pak samozřejmě ovlivnil uživatelské prostředí, které používají naši pracovníci na jednotlivých prodejnách – to byl hlavní důvod nasazení GNU/Linuxu jako uživatelského prostředí. Dodavatel původně doporučoval řešení s desktopem na operačním systému Windows 2000 s tím, že k rozhraní pokladního systému se uživatelé budou připojovat pomocí X serveru. To by znamenalo, že budeme muset provozovat dva různé operační systémy a vyčleňovat další počítač pro potřeby lokálního prodejního serveru. Proto jsme se rozhodli pro použití pracovního prostředí v prostředí operačního systému GNU/Linux.

Každá naše prodejna je počítačová síť se svojí vlastní adresou. V této síti jsou obvykle dvě pokladny a jeden počítač, kterému říkáme prodejní server a který slouží jako aplikační a databázový server pro pokladny, ze kterého jsou odesílány prodejní a skladové informace na centrální servery, a současně jako osobní počítač pro pracovníky na prodejně. A právě o uživatelském prostředí na tomto osobním počítači naleznete informace v tomto příspěvku.

Typickou instalaci prodejny máme nasazenou na 150 místech, které tvoří spolu s centrálou jednu počítačovou síť.

Původní uživatelské prostředí

Naše původní uživatelské prostředí na prodejním serveru běželo pod operačním systémem Windows 2000. Používaly se na něm především programy pro kancelář, to znamená Microsoft Office a programy pro základní správu pokladního systému napsané ve FoxPro. Ke správě se používal program PC Anywhere.

Používaný hardware

Používáme počítače od firmy Fujitsu-Siemens, a to jak na pokladnách, tak na prodejním serveru. Tyto počítačové sestavy jsou uzpůsobeny tak, že v nich lze vyměňovat komponenty bez použití jakéhokoliv nástroje. Na prodejních serverech jsou standardně použity tyto konfigurace:

1. Pentium III, 256 MB paměti, 2× disk 40 GB PATA
2. Pentium IV, 512 MB paměti, 2× disk 80GB PATA

Každý prodejní server má dva disky z důvodu použití softwarového RAID 1 pole.

Programové vybavení

Aktuálně používané uživatelské prostředí běží pod OS GNU/Linux, konkrétně to je distribuce RedHat ve verzi 7.3. Tato stará verze distribuce je použita na základě požadavku dodavatele pokladního systému. To pro nás znamená z dnešního pohledu jisté omezení, ale v této věci jsme vázáni dodavatelem pokladního systému. To nás omezuje ve výběru programů, které lze v pracovním prostředí použít. Aktuálně provozujeme následující programy pro práci v uživatelském prostředí a jeho správu:

- Programy pro kancelářskou práci
  - Okenní manažer QVWM (podobný Windows 9x) – tento okenní manažer je velmi jednoduchý jak svým nastavením, tak svými možnostmi. Nicméně naprosto dostačoval našim záměrům
  - OpenOffice.org – kancelářská činnost, kvůli kompatibilitě používány soubory typu XLS a DOC jako implicitní
  - Acrobat Reader – pro prohlížení PDF dokumentů (pro náhled postscriptu používáme gv)
  - Vlastní klient pokladního systému – pro správu pokladen a skladu prodejny.
  - Mozilla Suite – webový prohlížeč/poštovní klient
  - Nautilus – jako souborový manažer
- Programy pro správu desktopu
  - x0rfbserver – jako VNC server, ke kterému se připojujeme k aktuálnímu sezení X serveru, se kterým uživatel pracuje

  - OpenSSH server – pro většinu úkonů na desktopu nám stačí připojení přes SSH relaci a práce v shellu

- Další programy
  - OpenVPN – pro vytvoření virtuální privátní sítě
  - Xorg server – který umožňuje vlastní běh grafického uživatelského prostředí
  - PostgreSQL server – jako databázový backend pro pokladní systém

Uživatelské prostředí

Principy, na kterých jsme postavili uživatelské prostředí, a prostředky, jakými jsme toho dosáhli, byly:

- **Administrátorské zásahy musí být možno v maximální možné míře dělat vzdáleně.** Tento požadavek je legitimní hlavně z hlediska geografické vzdálenosti a množství našich prodejních míst. Právě unixové systémy se vzdálenou správou principiálně počítají a je pro ně přirozená.

- **Uživatel nesmí mít přístup k ničemu, co by mohlo ovlivnit stabilitu operačního systému, konektivitu pokladního serveru, běh databázového serveru a běh aplikačního serveru pokladního systému.** K tomuto jsme nepotřebovali žádné zvláštní opatření. Uživatelé na prodejně se přihlašují s právy normálního uživatele operačního systému a nejsou zařazeni do skupin, které by jim umožnili připojit jakékoliv výměnné médium nebo instalovat do systému balíček či program.

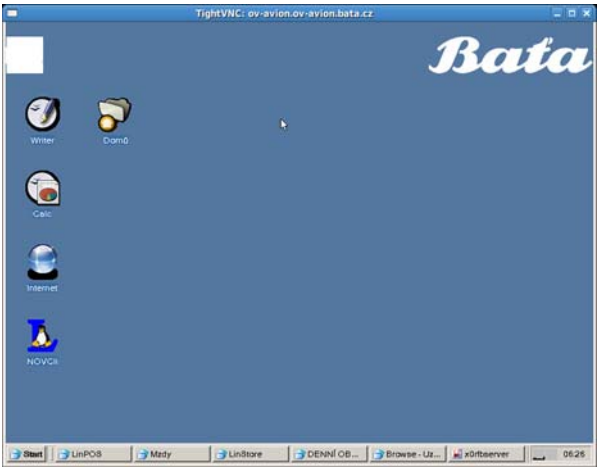
- **Uživatelé nebudou provádět žádná nastavení v operačním systému.** Vše jsme směřovali k vzdálené správě a vše nastavujeme vzdáleně.

- **Uživatelé si nebudou nastavovat ani vlastní uživatelské prostředí.** Nastavení uživatelského prostředí je uloženo v souborech, jejichž práva jsou nastavena tak, aby je uživatel mohl jenom číst (pozice a ikony spouštěčů na ploše, nastavení menu, nastavení pozadí) a jsou uloženy mimo jejich domovské adresáře.

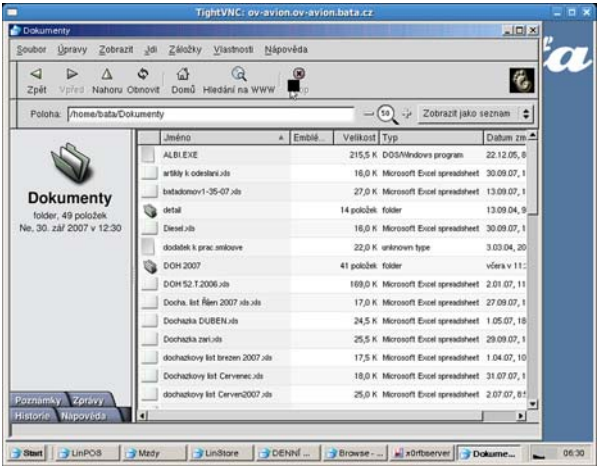
Tato opatření nejsou dokonalá. Neřešili jsme blokací nastavení programů OpenOffice.org a Mozilla Suite. Následně jsme z toho měli problémy s nastavením prohlížeče Mozilla, kde si uživatelé mění nastavení, které nejsou schopni opravit, a podobné problémy máme s OpenOffice.org. To mě utvrzuje v pro nás správné cestě minimalizace možnosti nastavení uživatelského prostředí ze strany uživatele v našich podmínkách.

Screenshoty

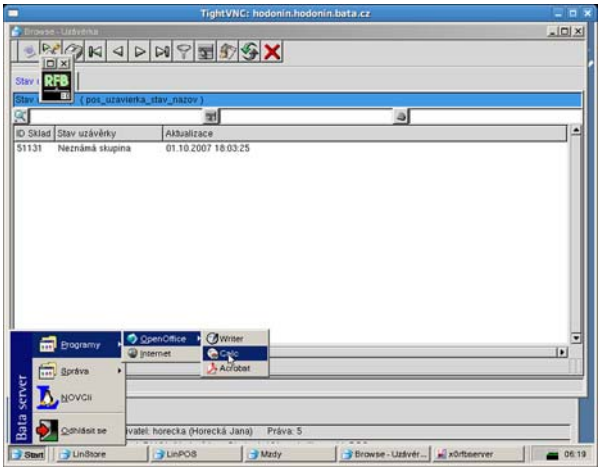
Pro náhled, jak vlastně naše uživatelské prostředí vypadá, zde uvádím několik málo screenshotů, získaných z aktuálně běžících prodejních serverů pomocí VNC, ve dvou náhodně vybraných prodejních místech.



Obr. 1 Uživatelské prostředí s náhledem tapety a minimalizovanými okny



Obr. 2 Souborový manažer Nautilus



Obr. 3 Otevřené menu – v pozadí je otevřený klient pokladního systému a v levém horním rohu okno, které signalizuje aktivní VNC připojení

Instalace

Vlastní instalace probíhala ve třech vlnách podle jednotlivých zemí (Česká republika, Slovensko a Polsko). První vlna v České republice trvala nejdéle, tři měsíce a ještě v jejím průběhu jsme, kromě jiného, dolaďovali uživatelské prostředí. Slovensko a Polsko pak těžily z této zkušenosti a jejich instalace již byly o poznání jednodušší.

Pro vlastní instalaci byl vytvořen instalační program, který nainstaloval systém, nastavil síť, rozdělil disky, vytvořil RAID pole a inicializoval databázi pokladního systému a nakopíroval konfiguraci uživatelského prostředí.

Instalace probíhala současně na několika prodejnách a byla prováděna externí firmou, jejíž pracovníci sice byli vyškoleni, ale nebyli to lidé s hlubokými znalostmi operačního systému GNU/Linux a tudíž i skript pro inicializaci byl napsán jako TUI aplikace.

Vlastní instalace jedné prodejny trvala pouze několik málo hodin, pak následovalo mnohem delší školení personálu.

Servisní zásahy

Servisní zásahy probíhají průběžně. Lze je rozdělit do dvou skupin.

Problémy, které se týkají softwaru a hardwaru. Celá prodejna je navržena tak, aby výpadek co nejvíce hardwarových nebo softwarových komponent neznemožnil prodej na pokladnách a to se týká i prodejního serveru, který obsahuje softwarový RAID. Běžně se řeší tyto problémy:

- Výpadek jednoho disku – pokud je to disk, na kterém není zavaděč OS, tak stačí na prodejnu poslat nový disk (poštou) a navést uživatele k jeho výměně. Pokud je to disk, na kterém je zavaděč, necháme systém běžet, dokud nepřijede technik, který disk vymění a zavede do MBR prvního disku zavaděč OS.

- Další HW poruchy, výpadek zdroje, výpadek paměti způsobí nefunkčnost prodejního serveru. Nicméně pokladny dál prodávají dokud nepřijede technik. V tomto stavu může prodejna fungovat i několik dnů.

Softwarové problémy řešíme téměř vždy vzdáleně. K vzdálenému servisnímu zásahu používáme SSH relaci, pomocí které se připojíme k prodejnímu serveru. Případně pokud potřebujeme interakci uživatele, používáme protokol VNC a v případě potřeby spuštění nějakého programu, který je napsán jako X client (typický klient od pokladního systému), použijeme X11 forwarding přes SSH protokol. Typické softwarové problémy prodejního serveru:

- Poškození PostgreSQL databáze při výpadku napájení nebo tvrdém vypnutí uživatelem

- Mnohonásobné spuštění OpenOffice nebo prohlížeče

- Zapomenutá hesla

- Nefunkční X server po výměně zařízení

Technické shrnutí

To, jestli jsou dané body výhodou nebo nevýhodou, nechávám na laskavém čtenáři těchto řádků.

- Vzdálená správa – s běžně dostupnými prostředky pro vzdálenou správu na GNU/Linuxu jsme dosáhli plné vzdálené vlády nad OS bez nutnosti spolupráce s uživatelem. Jsme schopni spravovat OS, uživatelovu relaci v interakci s ním i vzdáleně spouštět grafické programy bez jeho interakce. Toho bychom v původním prostředí Windows 2000 nebyli schopni vůbec a po případném přechodu na Windows XP velmi omezeně.

- Jednoduchost prostředí – desktop je jednoduchý s velkou mírou odolnosti proti uživatelským zásahům a nemožností měnit nastavení vlastního prostředí.

- Ochrana proti škodlivému software – na GNU/Linuxu prakticky neexistuje škodlivý software. Tímto jsme se zbavili poměrně velkého břemene nestabilního chování systému v důsledku škodlivého software.

- Znalosti uživatelů – uživatelé si až několik týdnů zvykali na nové prostředí. Museli začít pracovat v neznámém prostředí a mnoho postupů, na které uvykli, přestalo fungovat.

- Kompatibilita – došlo k problémům s kompatibilitou dokumentů generovaných především v programu Microsoft Word, to vedlo k širšímu použití formátu PDF.

- Postupné ladění – uživatelské prostředí na GNU/Linuxu v době, kdy vznikala naše instalace, mělo mnoho drobných vad, které jsme museli řešit svépomocí (vícenásobné spouštění prohlížeče, neobnovení locales Mozilly po jejím pádu, pády OpenOffice ve větší míře než u Microsoft Office).

- Orientace na web – přenesení velké váhy dalších aplikací do prostředí intranetu, protože jsme nemohli použít programy napsané pro MS Windows.

- Otevřenost – u drtivé většiny používaných programů nejsme závislí na konkrétním dodavateli (kromě vlastního pokladního systému). Máme možnost řešit problémy kreativně na základě hlubší znalosti OS, ke kterému máme naprosto svobodný přístup.

Šetříme

- Při instalaci byly zakoupené licence Windows 2000 a Microsoft Office 2000 a o ty jsme přišli. Šetříme až nákupem nových strojů v rozsahu cca 30 ks ročně, na kterých již není zakoupen ani operační systém Windows ani sada Microsoft Office.

- O 100 prodejních serverů se stará jeden člověk, který má navíc na starosti správu všech centrálních serverů a VPN. Tady je nevýhodou široká škála požadovaných znalostí správce.

- Jednoduchá funkční implementace VPN na aplikační úrovni pomocí OpenVPN, která nám vyřešila palčivý problém bezúdržbové cenově dostupné VPN.

Budoucí vývoj

Postupným příchodem novějšího hardware s větším množstvím paměti, výkonnějšími procesory a portací pokladního systémy na distribuci CentOS se otevřela cesta pro novou verzi desktopu. Zkoušeli jsme další okenní manažery a nakonec jsme postavili novou verzi desktopu na prostředí GNOME, které nám se svou ideou konfigurace vyhovovalo nejvíce.

Shrnutí

Instalace pokladního systému na GNU/Linux byla v Čechách a na Moravě dokončena v roce 2004. Máme tři roky zkušeností z ostrého provozu. Požadavky na funkční uživatelské prostředí, na kterém poběží současně i databáze a aplikační server pro pokladní systém, byla splněna. Uživatelé se dokázali poměrně rychle v novém prostředí zorientovat a od konce roku 2004 jsme dosáhli, co se týká uživatelského prostředí, rutinního provozu, který zvládá jeden člověk. I když se někomu prostředí může zdát spartánské, tak je funkční a pro naše potřeby plně dostačující. Naše zkušenost říká, že postavit uživatelské prostředí na GNU/Linuxu pro běžné uživatele v naší společnosti je možné.

Nasazení linuxových serverů a desktopů v podmínkách Úřadu městského obvodu Ostrava-jih

Josef Perzyna

E-mail: josef.perzyna@ovajih.cz

Abstrakt

Příspěvek pojednává o nasazení, správě a rozvoji informační infrastruktury úřadu, který používá GNU/Linux jako uživatelské rozhraní na cca 250 desktopech a s výjimkou jednoho na všech serverech. Příspěvek obsahuje pojednání o původních důvodech nasazení, dále popisuje vlastní realizaci nasazení a způsob jakým je celá tato infrastruktura spravována.

Historie

Počítačová síť ÚMOb Ostrava-jih je budována od roku 1992, kdy úřad disponoval pouze několika personálními počítači bez síťového propojení.

V roce 1993 byla koncepčně vytipována a vybrána platforma operačního systému, databázová struktura a kancelářský systém. Vzhledem k předpokládané rozsáhlosti informačního systému (velkému počtu koncových stanic a velkému objemu zpracovávaných a uchovávaných dat) byl v součinnosti s Magistrátem města Ostravy (dále MMO) a s přihlédnutím k vzájemné kompatibilitě mezi úřady, policií, hasiči a záchranou službou (IZS – integrovaný záchranný systém) vybrán operační systém SCO Unix, databázový systém Informix a kancelářský systém Uniplex. Byl zprovozněn server k němuž bylo v první etapě připojeno 20 terminálových pracovišť, 8 stanic a několik tiskáren. Kabelový rozvod byl proveden strukturovanou kabeláží a byl vytvořen zvláštní rozvod elektrické sítě pouze pro připojení výpočetní techniky.

V následujících letech se postupně zvyšoval počet koncových stanic, terminálů a tiskáren. Byly vytvářeny aplikace pro jednotlivé agendy jak dodavatelsky v koordinaci s MMO, tak vlastními zdroji. Byly pořizovány produkty vyšších verzí, byl několikrát vyměněn server za rychlejší a modernější a počítačová

síť se neustále rozšiřovala a modernizovala. Na serveru běžel operační systém Unix SCO OpenServer v. 5.05, databázový stroj Informix 4GL v. 7.30 a Informix DS v. 7.20, kancelářský balík UNIPLEX v. 8.10, fulltextová databáze Byllbase (ASPI, rady a zastupitelstva, vyhlášky) a program na údržbu živnostenských dat MAGIC. Z původních znakových terminálů jsme postupně asi do roku 1996 přecházeli na bezdiskové stanice bootujících z disket (OS DOS + terminálový emulátor). Důvodem nasazení bezdiskových stanic s emulátorem byl tisk, který byl u znakových terminálů omezen pouze na tiskárnu připojenou k terminálu. Bezdiskové stanice s emulátorem měly svou IP adresu a tiskový server, což umožnilo efektivnější tisk v rámci celé sítě. Mezi léty 1996–2000 jsme pořizovali kromě bezdiskových stanic také PC s OS Windows 9x. Jediným důvodem jejich nasazení byla jejich schopnost pracovat v grafickém prostředí na Internetu. Zaměstnanci pracující na znakových terminálech neměli přístup k Internetu, protože pouze zlomek webových stránek umožňovalo stejně jako dnes rozumné zobrazování obsahu pouze v textu bez grafiky. Spojení bylo pomocí proxy serveru a asynchronní komutované linky, přístupy na Internet se logovaly a následně i kontrolovaly. Zaměstnanci vykazující větší aktivity museli své počínání řádně zdůvodnit. S e-maily mohli pracovat všichni zaměstnanci. U znakových terminálů se používal poštovní klient pine a mohu potvrdit, že byl bez problémů. Kancelářský balík Uniplex umožňoval konverzi svých textových souborů do formátu RTF a také naopak konverzi z RTF do Uniplexu, což přispělo ke komunikaci s úřady, kde se masově nasazoval Microsoft Office. Jelikož byl tehdy formát RTF rozumným a dobře popsaným, nebyl problém při převodech. Bohužel ho později Microsoft značně deformoval a dnes máme s tímto formátem spíše problémy. Největší problémy tak byly s negramotností uživatelů Windows, kterým dělaly problém své texty, které posílali našim uživatelům uložit právě do formátu RTF. Těžištěm veškerých dat i aplikací však byly stále unixové servery, na kterých se i z Windows přistupovalo prostřednictvím terminálových emulátorů, protože měli na desktopech nainstalován pouze operační systém a zmíněný emulátor.

Rok 2000 byl zlomový. Tehdy jsme začali experimentovat s linuxovým desktopem (SuSE Linux 6.4), na kterém byl v té době mimo jiné dodáván ještě kancelářský balík StarOffice. Tyto desktopy se do stávající struktury integrovali daleko lépe než PC s OS Windows. Největším problémem v počátcích však byla kompatibilita s formáty Microsoft Office a neschopnost SW firem vyvíjet aplikace nezávislé na platformě. Obecně se v té době tvořily aplikace pouze pro PC s OS Windows, protože pro serverové aplikace nebyl trh. Je si ale také třeba uvědomit, že obecních úřadů naší velikosti není v republice mnoho. Problém s nedostatkem aplikací pro Linux jsme začali řešit pomocí Citrix MetaFrame serveru s Win2000, na kterém mohou uživatelé pracovat prostřednictvím ICA klienta, který samozřejmě existuje i pro linuxové desktopy. I když nám bylo zprvu vytýkáno, že toto řešení je nákladnější než pořízení PC s Windows, praxe ukázala,

že to byl krok správným směrem a Citrix se rozšířil i na úřadech používajících výhradně platformu Windows. Dobře napsané aplikace nemají s Citrixem problém. Aplikace psané pro Windows jsme nuceni používat na základě Statutu města Ostravy, který jasně definuje používaný SW pro účetní program, personalistiku, mzdy, sociální agendu atd. Jestliže si vybíráme aplikace pouze pro náš úřad, musí být jednoznačně multiplatformní.

Informační systém „Starosta“ fy Merit Group, a. s., byl v minulosti stěžejním pro chod úřadu. Dnes se již tato firma dodávkou SW pro obecní úřady nezabývá. Z nabízených modulů jsme používali:

- Ohlašovna (registry pobytu, volby, referenda)
- Adresa (registry územní identifikace)
- ISEN (registr nemovitostí)
- Registr organizací (dle evidence ČSÚ)
- Účetnictví (účet. deník, hl. kniha, sestavy, KDF, KOF, rozpočet
- Pokuty (finanční agenda pokut)
- Poplatky (ze psů, z automatů, . . . , ostatní příjmy)
- HIM, DHIM
- Majetek
- Investiční akce
- Stavební úřad
- Sociální dávky
- Přestupky
- RKTISK (obecný tiskový program)
- GEO (převody dat z katastrálního úřadu)

Kancelářský systém Uniplex nám sloužil dlouhá léta nejen jako textový editor a tabulkový procesor, ale využívali jsme hlavně v něm integrovanou databázi (Informix SE). V této databázi se vedla veškerá pomocná data, která se zpracovávala v tzv. generátorech sestav. Příkladem námi vyvinutých databází je:

- Doručovací kniha a podací deníky odborů
- Organizace voleb (volební komise, materiální zabezpečení atd.)

- Evidence stravenek
- Evidence skladu kancelářského a ostatního materiálu
- Pořadník na byty včetně evidence žadatelů
- Směny bytů
- Registr ochrany dětí a mládeže
- Usnesení rad a zastupitelstev, vyhlášky
- Program na zpracování referátníků
- Kartotéka řemeslnických firem
- Skladové hospodářství krytů CO
- Program na zpracování, tvorbu a hlídání rozpočtu
- Ekonomické rozbory rozpočtu, evidence faktur
- Program pro výměnu dat s Úřadem práce
- Program pro předávání dat na složenky typu „H“
- Evidence záruk prací
- Evidence pohledávek úřadu
- Evidence docházky
- Zpracování výběrových řízení atd.

Tyto databáze byly postupně nahrazovány různými systémy a dnes již žádnou z nich nevyužíváme.

Jak jsem uvedl výše, integrace linuxových desktopů do sítě byla až nápadně jednoduchá. Stávající unixový server, na kterém měli uživatelé data např. z Uniplexu sloužil zároveň jako file server pro všechny linuxové desktopy. Uživateli se po přihlášení přes NIS automaticky namountoval přes NFS jeho domovský adresář na unixovém serveru. Tímto se desktopy chovaly téměř jako grafické terminály, jen s tím rozdílem, že aplikace se spouštěly na desktopu a data byla k dispozici přes NFS na serveru. Uživatelé rovněž uvítali možnost zpracovávání textových výstupů z Informixu (IS Starosta) ve StarOffice, hlavně jim to pomáhalo při tvorbě tabulek a grafů.

Geografický informační systém byl na Úřadě městského obvodu Ostrava-jih využíván pouze dvěma odbory a to odborem komunální správy a odborem výstavby a životního prostředí. Na odboru výstavby a životního prostředí byl umístěn server Hewlett Packard (OS HP Unix), který tvořil základ GISu. Na tomto

serveru byla umístěna geografická data, licence programu ArcView a aplikace napsané pro tento program. Popisná data byla získávána z UNIXového serveru (Informix). Dalším prvkem architektury GIS byl PC s OS Linux (distribuce SlackWare). S tímto počítačem disponoval odbor komunální správy. Na IT oddělení byl umístěn počítač, který pracoval pod OS Windows 95, a kde byl nainstalován program ArcWiev. Připojení k serveru HP bylo možno přes rozhraní X Window i na linuxových desktopech. Tisk dokumentů všech výše uvedených stanic byl umožněn pomocí tiskárny připojené k serveru HP, a to ve formátu A4. Tisk větších map (až A1) byl možný na plotter umístěný na IT oddělení.

S postupem let se téměř pravidelně instalovaly na desktopech nové verze SuSE Linuxu. U této distribuce jsme zůstali dodnes, pouze StarOffice jsme nahradili za OpenOffice.org. Samozřejmě přibývaly také nové servery (file server, backup server, mail server, proxy server, různé aplikační a dohledové servery atd.).

Současnost

V současné době disponuje ÚMOb Ostrava-jih rozsáhlou sítí s více jak 300 PC (k datu tvorby tohoto textu 254× Linux, 61× Windows) a 12× servery. Původní informační systém „Starosta“ pod DB Informix byl před několika lety nahrazen informačním systémem „Radnice“ fy VERA, spol. s r. o., pod DB Oracle. Veškeré databázové i aplikační servery tohoto systému jsou fyzicky umístěny v prostorách MMO a spojení mezi aplikačními servery a desktopy zprostředkovává Genero Desktop Client prostřednictvím metropolitní sítě. Tento IS je vedle OpenOffice.org, Firefoxem a Thunderbirdem nejpoužívanější aplikací, protože jak je ze seznamu níže patrno, jedná se o komplexní informační systém pro obecní úřady.

Námi používané moduly IS Radnice:

- Evidence obyvatel – základní registr obyvatel
- Adresy – registr adres, územní identifikace
- Ohlašovna
- Matrika
- Volby
- Příjmová účtárna
- Příjmová pokladna
- Faktury vydané

- Banka, pošta – příjmy
- Evidence majetku
- Psi – evidence psů
- Pozemky – poplatky z nájmu pozemku
- Automaty – poplatky za provoz hracích automatů
- Integrovaný systém evidence nemovitostí
- Registr smluv
- Registr organizací v Ostravě
- Přestupky
- Stavební úřad

Dalším rozsáhlým systémem je ekonomický systém „Ginis“ fy Gordic, spol. s r. o. I když se mohly využívat další moduly v IS „Radnice“, byl tento systém určen pro všechny obvody města Ostravy Statutem města Ostravy. Z hlediska účetnictví se jedná o velmi kvalitní ekonomický systém, ale pro systémové řešení v podmínkách města se mi jeví jako nevhodný. Na jedné straně máme registr obyvatel a ekonomických subjektů v IS „Radnice“ z čehož vyplývá, že veškeré poplatky, pokuty nebo příjmy jako takové jsou evidovány v tomto systému. Výdaje úřadů se účtují v systému „Ginis“, a protože účetnictví by mělo být jednotným celkem, veškeré příjmy se exportují z IS „Radnice“ do systému „Ginis“. Kdyby šlo pouze o export dat, tak by bylo vše v pořádku, ale je zde nutná spolupráce mezi administrátory obou systémů, protože číselníky se bohužel neexportují, takže každou chvíli řešíme problémy se zbloudilými položkami. Zde zafungovalo to, co je myslím velikou brzdou rozvoje a nasazování informačních systémů v podmínkách státní správy a samosprávy a to je, že o nasazení rozhodují nekompetentní úředníci, kteří nevidí širší souvislosti v rámci celého systému, drží se svých „osahaných“ programků a odsuzují vše jiné, aniž by to byli schopni vyzkoušet a na základě zkušeností a ve spolupráci s IT pracovníky kvalifikovaně rozhodnout. Tento systém je psán výhradně pro OS Windows, takže díky tomu jsme byli nuceni nasadit Citrix MetaFrame.

Námi používané moduly IS Ginis:

- POK – pokladna výdajová
- PPD – příprava pokladních dokladů
- KDF – kniha došlých faktur

- POU – kniha poukazů
- PRE – kniha převodních poukazů
- UCR – rozpočtové a účetní výstupy
- BUC – komunikace s bankou
- ADE – administrace ekonomická
- ADP – administrace předkontací
- ADR – administrace rozvrhu
- INU – manipulace s daty
- ROZ – rozpočet
- UCT – pořizovač účetních dokladů

Podobným způsobem jako IS Ginis byl nasazen i modul „Sociální agenda“ informačního systému „CityWare“ dodavatele Geovap, spol. s r. o. Rovněž modul „Sociální agenda“ by mohl být využíván ve stávajícím systému „Radnice“, ale referentky sociálních odborů městských obvodů přesvědčila hlavně vizáž přednášejícího. Jedná se tradičně o systém pouze na platformě Windows, takže jsme začali uvažovat o virtuálních serverech (VMware) s Citrixem. Některé referentky sociálních odborů (55) musí v rámci zákona o hmotné nouzi používat systém dodaný Ministerstvem práce a sociálních věcí „OK Nouze“, který vytvořila firma OKsystem, s. r. o. Samotná DB i aplikační servery jsou na ministerstvu, přístup je přes browser a zabezpečenou přímou linku. Dalo by se říct, že aplikace by mohla být multiplatformní, ale není. Není ji ani možné provozovat na Citrixu, protože je záměrně psaná pro desktopy s OS Windows (mapování čtečky čárových kódů a chystané čipové karty pro autentizaci). Mnoho starostů obcí snad potěšilo, že PC, tiskárny a kopírky byly zapůjčeny ministerstvem, ale vzpomněl jsem si na jinou akci v minulosti s názvem INDOŠ a jako daňového poplatníka mě hrubě uráží způsob jakým se plýtvá při přerozdělování veřejných financí.

Systém pro evidenci jednání rad a zastupitelstva jsme přesunuli z dříve používaného Uniplexu na samostatný systém AiP Safe III, dodavatele AiP Safe, s. r. o. Tento systém pracuje s DB Oracle a aplikační server je Apache s aplikací Jakarta Tomcat. Je to kvalitní open-source řešení pro Java platformu. V současné době se jedná o nejkomplexnější volně šiřitelné řešení pro serverové aplikace.

Několik let jsme čekali na nasazení spisové služby, protože MMO plánoval její nasazení pro všechny městské obvody v rámci projektu eSMO (elektronické Statutární město Ostrava) a nedoporučoval nákup aplikací na jednotlivých úřadech.

Řešení je podobné jako u systému pro evidenci jednání rad a zastupitelstva. Dodavatelem řešení je Aplis.cz, a. s., a v těchto dnech probíhá školení uživatelů. Do ostrého provozu bude nasazen v listopadu 2007.

Servery GISu jsou již několik let fyzicky přesunuty do prostor MMO, kde se jim věnuje celé IT oddělení. Data je již možno zobrazovat pomocí browserů, takže není ani potřeba vysoké průchodnosti metropolitní sítě. Podařilo se nám dokonce umravnit tvůrce aplikací tak, aby nebyly problémy se zobrazováním dat v jiných prohlížečích než na své operační systémy dodává Microsoft.

Intranet a webové stránky úřadu (http://www.ovajih.cz) jsou tvořeny v aplikaci OpenIntranet. Aplikace představuje v první řadě systém pro rychlou, snadnou a pohodlnou správu obsahu intranetu a pro publikování na internetu. V principu se tedy řadí mezi takzvané CMS (Content Management Systems – systémy pro správu obsahu). Díky své modularitě a otevřenosti však tyto v mnohém překonává (moduly Evidence zaměstnanců, Evidence budov, Organizační struktura, Evidence hardware, Kontrola docházky, Evidence stravenek, InfoMail atd.). Spolutvůrcem tohoto systému je náš webmaster, takže je zde zaručena jistá pružnost v případě nově vzniklých požadavků na vlastnosti nebo funkčnosti systému.

Klíčové vlastnosti aplikace OpenIntranet:

- nezávislost na platformě, systém je použitelný v prostředí MS Windows, Unix-GNU/Linux, MacOS
- plně modulární systém s možností snadného rozšíření
- možnost zcela volně definovat strukturu intranetu podle vlastních potřeb
- integrovaný schvalovací mechanismus pro publikaci obsahu
- správa obsahu je realizována prostřednictvím integrovaného WYSIWYG HTML editoru
- rychlá, snadná a pohodlná správa obsahu (texty, obrázky, odkazy)
- intuitivní a nenáročná obsluha v prostředí WWW prohlížeče
- aktualizaci obsahu může provádět kdokoliv se základními znalostmi práce s internetovým prohlížečem a textovým editorem
- úspora času a finančních prostředků při aktualizaci a tvorbě obsahu

Přístup k internetu je řešen prostřednictvím metropolitní sítě stejně jako přístupy na servery MMO. Jelikož máme i dislokovaná pracoviště, přístupy na naši síť a potažmo i na internet a servery MMO řešíme díky přímé viditelnosti šifrovaným wi-fi spojením (Canopy v pásmu 5 GHz).

Instalace nových distribucí se dějí po síti a neustále vylepšujeme její způsob. V praxi to vypadá tak, že se nová verze nainstaluje na jedno PC, odzkouší se

(cca 14 dní), vytvoří se image disku, která se uloží na server a potom se přes thinstation nainstalují všechny PC. Instalace jednoho PC trvá cca 10 až 15 minut. Jestliže se později vyskytne nějaký problém, řeší se to opravnou instalací pomocí skriptů. Veškeré konfigurační soubory uživatelů jsou v jejich domovských adresářích na serveru, takže po instalaci nové distribuce nemusíme uživatele obcházet a neustále konfigurovat již jednou nakonfigurované aplikace. Nové verze programů s tím zatím neměly větší problémy. Jedinou věcí kterou konfigurujeme na desktopu je tiskárna, ale i tady máme do budoucna řešení, jak to obejít.

Na všech desktopech včetně desktopů s OS Windows je nainstalován OpenOffice.org, protože tvorbu všech příloh např. pro aplikaci „Evidence jednání" nebo dokumentů obecně máme podmíněno formátem ODF, případně PDF, do kterého se dají dokumenty z OpenOffice.org exportovat.

V současnosti již také nepoužíváme pro autentizaci uživatelů NIS, ale pořízením file serveru s větší diskovou kapacitou (RAID, hotswap) jsme přešli z SCO Unix serveru na Linux a zároveň jsme také přešli na autorizaci prostřednictvím Open LDAPu, samozřejmě přes ssh. Mountování domovských adresářů přes NFS používáme i nadále. Pro zaměstnance sociálních odborů nucené pracovat na strojích dodaných MPSV jsme zprovoznili na file serveru Sambu, takže se do sítě hlásí také přes Open LDAP a jejich domovský adresář je stejně jako u uživatelů pracujících na linuxových desktopech na linuxovém file serveru. Obecně známou výhodou je nezávislost na desktopu a odpadá také starost uživatelů o zálohování svých dat. Zálohování uživatelských dat probíhá automaticky každou noc na backup server, který se jednou týdně zálohuje na USB disky. Zálohy systémových dat se provádí také automaticky, ale s delším intervalem. Z bezpečnostních důvodů je backup server umístěn v jiné budově než file server a zálohované USB disky se vozí také zcela mimo areál úřadu, kde jsou uschovány v trezoru.

Budoucnost

Nejbližší budoucností v rámci úřadu Ostrava-jih je přeorganizování serverové struktury. Pomalu nastává čas, kdy většina z 12 serverů dosluhuje a místo nákupu nových samostatných strojů zvažujeme virtualizaci serverů. Základem by byly 3 fyzické servery a na každém z nich bude nakonfigurováno několik serverů virtuálních. Na každém fyzickém serveru bude nainstalován VMware ESX Server samozřejmě s doplňkem Virtual SMP, který virtuálním strojům umožňuje přímo přistupovat k více reálným procesorům. K těmto třem strojům plánujeme nákup dvou diskových polí, každé s kapacitou cca 9 TB. Jelikož jsme nikde nenašli informace o náročnosti provozu OpenOffice.org a dalších námi používaných aplikací na aplikačním serveru, budeme zkoušet a monitorovat zátěž s několika uživateli. Na základě výsledků se rozhodneme, zda nepřejít z PC na grafické terminály. Na centrálním serveru by byla uložena data a běžely by zde všechny aplikace, za-

tímco terminály by byly použity pouze jako zobrazovací jednotky. Celá síť se tak redukuje na správu jednoho (virtuálního) počítače. Tato technologie nazývaná také technologií tenkých klientů přináší:

- úsporu provozních nákladů o 80 % a pořizovacích nákladů o 40 %

Toho je dosaženo zejména díky:

- nižší spotřebě elektrické energie a snížení tepelného vyzařování
- jako terminály lze použít i staré (vlastní či bazarové) počítače
- značnému zjednodušení správy celé sítě (péče o jeden virtuální server)
- úspoře prostoru (terminály mohou být pouze malé skříňky)
- vyššímu zabezpečení sítě
- snížení hluku na pracovišti
- zvýšení pohody na pracovišti

Virtualizace v oblasti serverů přináší podle nejrůznějších benchmarků a studií hned několik výhod, tou hlavní je lepší využití hardware, který je k dispozici. Průměrná hodnota zvýšení využití serveru při nasazení virtualizačního řešení, tedy běhu několika virtuálních serverů na jednom hardware, je dle dostupných zdrojů 15%, ale v některých případech může dosáhnout až 80% hodnoty.

Druhou výhodou na serverech je vyšší dostupnost a jejich případné snadné nahrazení nebo upgrade, který proběhne jen přidělením většího dílu dostupných prostředků, například operační paměti nebo procesorů. Oddělení jednotlivých virtuálních strojů od sebe zvyšuje bezpečnost a stabilitu systémů, komunikace mezi jednotlivými operačními systémy probíhá prostřednictvím virtuální počítačové sítě, kterou lze monitorovat a chránit firewally.

Dalším krokem bude ve spolupráci s MMO zprovoznění Identity a Access Manageru, protože jsou uživatelé díky aplikačním serverům umístěným na MMO nuceni používat účty a hesla v několika systémech. U nás se uživatel přihlásí do sítě a následně se hlásí pod jiným jménem a heslem do aplikačních serverů. Jde sice „pouze“ o dva účty, ale Identity a Access Manager značně zjednoduší a zpřehlední správcům databází nebo nadřízeným uživatele jeho aktivity.

Závěr

S potěšením mohu konstatovat, že i při nárůstu nových aplikací jsou problémy na desktopech i serverech s OS Linux čím dál tím menší. Trh s aplikačními servery nezávislých na platformě utěšeně roste a pevně věřím, že při současném

trendu více využívat takové aplikace nebude taková potřeba instalovat na desktopy OS Windows alespoň ve státní správě a samosprávě, kde by se dalo při trochu dobré vůle ušetřit nemalé finanční prostředky již dnes. Také mě velmi těší tlak EU na standardizaci datových formátů, kdy si úředníci uvědomili, že i v budoucnu bude potřeba číst data vytvořená před mnoha lety. Bez jasně definovaných pravidel to však možné nebude, což podle mne povede k zániku proprietárních formátů. Současně se tak trochu obávám iniciativy Microsoftu o vytvoření vlastního otevřeného formátu, který nás ale zřejmě nemine. Doufám, že to bude opravdu otevřený formát bez binárních vsuvek, a že bude použitelný i pro jiné aplikace než Microsoft. Ke skepsi mě vedou zkušenosti s formátem RTF, který si také upravili spíše pro sebe.

Potřebujeme standardizované formáty pro kancelářské dokumenty?

Jiří Kosek

E-mail: jirka@kosek.cz

Klíčová slova: XML, ODF, OOXML, formáty dokumentů, standardizace

Abstrakt

Po dlouhá léta většina kancelářských aplikací ukládala svá data do proprietárních binárních formátů. Nyní se dvěma nejpoužívanějšími formáty stávají ODF a OOXML, které jsou oba otevřené a založené na XML. První z nich je ISO normou a druhý pravděpodobně také brzy bude. Má smysl, aby vedle sebe existovaly dva tak podobné formáty? Jak tato situace vznikla. Může standardizace formátů zaručit bezproblémovou výměnu dokumentů? Nad historií, současností a budoucností otevřených formátů dokumentů se ve své přednášce zamyslí Jiří Kosek, který se mimo jiné podílí na vývoji formátu DocBook a zpracovával připomínky ČR k návrhu normy OOXML.

V době, kdy jsou všechny počítače propojené pomocí Internetu, nám připadá zcela samozřejmé, že si s ostatními vyměňujeme elektronické dokumenty a můžeme s nimi dále pracovat. Ale i přesto, že informační technologie se vyvíjejí rychle, stále se občas stane, že některé dokumenty si neotevřeme nebo po jejich otevření vidíme něco jiného, než jejich původní autor.

Tyto problémy jsou způsobeny dnešním prostředím, které je heterogenní – uživatelé používají různé operační systémy, na nich běží různé aplikace, ... V počítačovém středověku byla situace o poznání jednodušší – nikdo ani neočekával, že by byl schopen pracovat s dokumenty pořízenými na počítačích od jiného výrobce, někdy to dokonce neplatilo pro různé verze počítačů od jednoho výrobce. V unifikovaném prostředí šlo vše hladce, akorát byla fakticky znemožněna výměna dokumentů, což byl z pohledu uživatel zásadní nedostatek.

1 Historické ohlédnutí za vývojem formátu souborů

Již od konce 60. let se proto hledaly způsoby, jak zajistit výměnu a sdílení dokumentů v heterogenním prostředí. Vznikaly formáty jako TEX a troff, které se oprostili od platformové závislosti, ale přitom zachovávaly formátování i po přenosu na jiné zařízení. Ještě dále šly obecné značkovací jazyky, ze kterých se později vyvinuly jazyky SGML a XML. Ty se snažily dokument zcela oprostit od popisu formátování a zachycovat pouze význam a strukturu informace. Později vznikly formáty PostScript a PDF, které dokázaly věrně zachytit vzhled dokumentu, nebyly však vhodné pro další editaci.

S masivním nástupem osobních počítačů v 80. letech však začalo být jasné, že masy uživatelů potřebují něco jiného – pohodlný textový editor. Textový editor z principu musí dovolit uživateli provádět všechna obvyklá zvěrstva jako nahodilé změny písma, zarovnání a barvy. Paradigma uživatelského rozhraní textového editoru tak bylo téměř neslučitelné s více strukturovanými formáty jako SGML/XML nebo TeX. Formáty PostScript a PDF se zase příliš nehodí pro editovatelné dokumenty. Každý textový editor tak přicházel s vlastním formátem pro ukládání dat. Při výměně dokumentů bylo potřeba spoléhat na to, že všichni používají stejný produkt, nebo mají alespoň příslušné importní/exportní filtry.

V České republice jsme si na začátku 90. let užívali luxusu „monopolního“ formátu T602. Osobně považuji textový editor T602 v mnoha ohledech za dosud nepřekonaný – zejména z toho důvodu, že příliš nepřevyšoval schopnosti většiny běžných uživatelů.

Vývoj se však nezastavil, přišla grafická uživatelská rozhraní a s nimi další plejáda editorů – MS Word, AmiPro, WordPro, WordPerfect, StarOffice a další. Jediným formátem, kterému v té době rozuměli snad všechny editory, byl formát RTF (Rich Text Format). Tento formát původně vymyslela firma DEC, ale od roku 1990 ho převzala společnost Microsoft a postupně do něj doplňovala nové funkce tak, jak se vyvíjel MS Word. Poslední verze formátu RTF 1.9 byla vydána v lednu 2007.

Formát RTF fungoval docela dobře, trpěl dvěma zásadnějšími problémy. Základem RTF je prostý text. Delší dokumenty s obrázky tak byly po uložení do RTF obludně velké, protože binární data se musela překódovat do textově bezpečné podoby a tím se nafoukla.

Druhým problémem byla nevzdělanost uživatelů. V druhé půli 90. let získal Microsoft na trhu textových editorů téměř monopolní postavení. Hodně mu v tom pomohli samotní uživatelé, kteří dokumenty ukládali do proprietárního binárního formátu DOC, místo formátu RTF. Protože v té době bylo možné formát DOC analyzovat v podstatě jen metodou reverzního inženýrství jeho podpora v ostatních aplikacích nebyla úplně dokonalá. Kdo chtěl mít možnost bez pro-

blémů pracovat s dokumenty DOC, tak si raději pořídil MS Office. Masivnější používaní RTF by přitom pro trh textových editorů bylo jistě zdravější.

Díky tomuto mechanismu se postavení MS Office na trhu dále upevňovalo. Podobná situace zavládla i na poli ostatních kancelářských aplikací jako tabulkových kalkulátorů a prezentačních nástrojů. Trojice formátů DOC/XLS/PPT se tak stala „de facto“ standardy pro výměnu kancelářských aplikací.

V uvozovkách by však mělo být spíše slovo „standardy“. Z uživatelského hlediska (pokud byl uživatel vlastníkem MS Office) byly formáty DOC/XSL/PPT v zásadě bezproblémové.[1] Tyto formáty však byly problém pro vývojáře třetích stran. Dokumentace k těmto binárním formátům nebyla dlouhou dobu dostupná, a po jejím uvolnění licence nedovolovala využití dokumentace při tvorbě kancelářských aplikací (tedy produktů konkurujících MS Office).

Složité binární formáty nepřály ani aplikacím, které měly například automaticky generovat různé reporty. Spolehlivou cestou bylo jen skriptovat samotný MS Office, což bylo náročné jak finančně (licence), tak na strojový čas. S tímto stavem nebyli vývojáři aplikací spokojení a proto Microsoft začal postupně od roku 2000 přidávat do jednotlivých produktů MS Office alternativní možnost ukládání/načítání dat v XML. Tyto formáty však nikdy nebyly nastaveny jako výchozí, a proto nedošlo k jejich rozšíření mezi běžnými uživateli. Nicméně velké uplatnění našly v místech, kdy bylo potřeba automatizovat práci s kancelářskými dokumenty.

2 Éra otevřených a standardizovaných formátů přichází

Přelom milénia byl však ve znamení renesance otevřených formátů. Rozvoj webu a potřeba komunikace dali vzniknout jazyku XML a vytváření výměnných formátů založených na XML se tak postupně stalo nejen v mnoha případech výhodné, ale i módní. Ke starším formátům pocházejícím ještě z dob SGML jako DocBook (počítačová dokumentace), TEI (literatura) a leteckým a vojenským standardům pro dokumentaci se tak začaly přidávat desítky nových formátů. Pro komunikaci mezi aplikacemi se začínají masově používat webové služby, které jsou postaveny na výměně zpráv XML. XML se tak pevně zabydlelo v oblasti výměny strukturovaných dat (kde nahradilo mnohem nákladnější EDI) a strukturovaných dokumentů (zastoupené například formáty DocBook, DITA a TEI).

Mezi hlavními výhody XML bývá uváděna nezávislost na dodavateli a dlouhá životnost dokumentů. Nicméně obojí jsou vlastnosti, které nesouvisí přímo s XML, ale samotná podstata XML vede k tomu, že od něj odvozené formáty tyto výhody nabízejí. Popularita XML znovu přivedla na výsluní standardizaci

[1] Pomíjím teď některé drobné problémy s kompatibilitou mezi verzemi MS Office.

datových formátů. Standardizace přitom není nic jiného než proces, během kterého se zainteresované strany dohodnou, jak něco dělat jednotným způsobem. Výsledek nemusí být vždy technicky nejlepší řešení, ale to vyváží výhody dosažené interoperability. Navíc většina dnešních standardů je vytvářena jako tzv. otevřené. Znamená to, že jsou veřejně dostupné a že pro jejich implementaci není potřeba platit žádné licenční poplatky.

Pro obecné kancelářské dokumenty však žádný takový formát dlouhou dobu neexistoval. Až v roce 2000 kancelářský balík StarOffice začíná pro ukládání svých dat používat formát XML. Formát je postupně vylepšován a přejímá jej i open-source odnož OpenOffice.org. Na konci roku 2002 je pak v rámci standardizační organizace OASIS použit tento formát jako základ budoucího formátu ODF (Open Document Format). Na vývoji se podílí hlavně firma Sun Microsystems. Microsoft se práce na standardu neúčastní. A není to ani překvapivé – těžko si představit, že by firma s majoritním podílem na trhu akceptovala funkčně zaostalý[2] formát aplikace s minoritním podílem na trhu. Takhle svět nefunguje.

V květnu 2005 se ODF stává standardem OASIS a na žádost EU je také postoupeno ke schválení jako ISO norma. Nutno podotknout, že to vše se děje poměrně bez zájmu odborné veřejnosti, protože formát je podporován aplikacemi, které mají pouze zanedbatelný podíl na trhu. Navíc přijetí ODF jako ISO normy je poměrně rozporuplný počin – formát má mnoho nedodělků, které mají být odstraněny až v dalších verzích. Nicméně rychlost procesů uvnitř OASIS a ISO je rozdílná, takže například v současné době máme formáty ODF vlastně dva – ISO ODF 1.0 a OASIS ODF 1.1. Probíhají přitom už i práce na novější verzi ODF 1.2. Jak, kdy a zda vůbec budou novější verze přijaty i jako ISO norma není v tuto chvíli vůbec jasné. Pro uživatele tak vznikla poněkud nepřehledná situace. Osobně si myslím, že se mělo počkat nejméně na ODF 1.2 a až to případně přijmout jako ISO normu.

Přijetí ODF jako normy ISO mělo pozitivní efekty. Rozhýbalo ledy i softwarového giganta Microsoft. Těžko říci, zda větší vliv na to měly doporučení EU, mediální tlak propagátorů open-source, příklon některých vlád k řešením postaveným na mezinárodních normách nebo se Microsoft sám změnil – nejspíše asi kombinace všech těchto faktorů.

Microsoft tak vylepšil XML formáty předchozích verzí MS Office a formát s názvem Office Open XML (OOXML) zasílá na konci roku 2005 do standardizační organizace ECMA, která jej po úpravách o rok později přijímá jako svůj standard. Poté je OOXML zasláno do ISO pro přijetí jako ISO norma. Tento proces ještě neskončil, ale je pravděpodobné, že v únoru 2008 se i OOXML stane mezinárodní normou.

[2]První verze formátu ODF nepodporovala ani veškerou funkčnost OpenOffice.org, který však nabízí méně funkcí než MS Office, které je nutné nějak promítnout do formátu souborů. ODF například nedefinuje syntaxi pro zápis vzorců v tabulkové kalkulátoru a prezentace nemůže obsahovat tabulky.

Formáty OOXML a ODF jsou si v mnohém podobné, a může se tak zdát zbytečné, že existují oba dva. Nesmíme však zapomenout, že jsme teprve na začátku cesty vedoucí k případnému vytvoření jednotného formátu. Do začátku je potřeba mít konkurenci, která požene vývoj kupředu. Během posledního roku vzniklo tolik nástrojů pro konverzi mezi ODF a OOXML a pro práci s těmito formáty jako nikdy předtím. Během schvalování OOXML na půdě ISO bude specifikace formátu vylepšena. Vytvářejí se studie, které oba formáty porovnávají a hledají vlastnosti, které nemají v druhém formátu ekvivalent.

Bohužel jsme dnes svědky toho, že standardizace se stala magickým zaklínadlem a v případě kancelářských formátů se dokonce bitva konkurentů dostala do podoby, kdy se soupeří o to, kdo dřív dostane razítko ISO. Formáty a produkty, které je podporují, by přitom měli soutěžit na volném trhu svými schopnostmi a užitečností pro uživatele. U obou formátů by přitom stačilo, kdyby si zachovaly statut standardů OASIS, resp. ECMA a o případné mezinárodní standardizaci by se mělo uvažovat až s několikaletým odstupem na základě praktických zkušeností s oběma formáty.

3 Samotná standardizace nestačí

To, že je dnes nějaký formát schválen jako norma je automaticky považováno za dostatečné pro dosažení interoperability. To je však naprosto mylná představa.

Kancelářské formáty dovolují vkládat do dokumentů objekty, jejichž formát není nijak omezen. Pokud tedy chceme definovat nějaké prostředí pro spolehlivou výměnu dokumentů, nestačí říci „bude se používat ODF nebo OOXML“. Musíme ještě přesně definovat jaké objekty mohou být do dokumentů vložené – například povolit jen obrázky v široce akceptovaných formátech.

Dalším problémem jsou fonty – každý uživatel může mít na svém počítači nainstalovány jiné fonty, dokument se tak nemusí vždy zobrazit stejně. Chceme-li dosáhnout věrnějšího zobrazení dokumentu po jeho přenesení na jiný systém je potřeba definovat i množinu fontů, které je dovoleno používat.

Standardizované kancelářské formáty umožňují sdílení dokumentů mezi několika různými aplikacemi od různých dodavatelů. To je dobře a je to vlastně stav, kterého by mělo být dosaženo. Uživatelé si však musí na toto heterogenní prostředí zvyknout. Ani ODF ani OOXML nedefinuje přesné algoritmy pro řádkový a stránkový zlom, nejsou definovány jednotné sady písem a jejich metrik, každá aplikace používá vlastní algoritmus pro dělení slov – ve výsledku tak bude stejný dokument v různých aplikacích zobrazen s drobnými odlišnostmi. Na to uživatelé odchování na softwarové monokultuře nejsou zvyklí.

Velké firmy nebo veřejná správa by si tak měli vytvářet dodatečné směrnice, které obecné standardy v takovýchto detailech dospecifikují. Bez toho nebude nikdy bezproblémová výměna dokumentů zaručena.

Příkladem toho, jak by se to dělat nemělo je doporučení,[3] které vydalo Ministerstvo informatiky těsně před svým zrušením.

> ***Doporučení k aplikaci § 4 odst. 2 zákona č. 106/1999 Sb., o svobodném přístupu k informacím, ve znění pozdějších předpisů***
>
> *V § 4 odst. 2 je stanovena povinnost v případě zveřejňování informace v elektronické podobě zveřejnit tuto informaci i ve formátu, jehož specifikace je volně dostupná a použití uživatelem není omezováno.*
>
> *V těchto případech Ministerstvo informatiky doporučuje využívat následující formáty:*
>
> - *Pro dokumenty jako jsou texty (.odt), tabulky a grafy (.ods) a prezentace (.odp):*
>
> *Open Document Format for Office Applications (OpenDocument) v1.0*
> - *Pro dokumenty, které jsou zpřístupněny přes internet/intranet nebo použitím webového prohlížeče:*
>
> *The Extensible HyperText Markup Language (XHTML v1.0)*
>
> *HyperText Markup Language (HTML v4.01)*
> - *Pro dokumenty obsahující pouze textové údaje, bez potřeby formátovacích funkcí:*
>
> *Plain text format (.txt – např. u e-mailových zpráv, kdy formátování není důležité)*
> - *Pro úřední dokumenty, které není potřeba dále upravovat/měnit:*
>
> *Portable Document Format/A-1 (ISO 19005-1:2005)*
>
> *Portable Document Format (PDF Reference v1.5 a vyšší)*

Jaké jsou problémy tohoto doporučení.

1. Preferování formátu ODF, který nemá dostatečnou podporu v aplikacích nainstalovaných u uživatelů. I když je samotný software podporují ODF zdarma, jeho instalace a proškolení uživatelů by stálo nemalé prostředky. Nestojím o to, aby se kvůli tomu zvyšovaly daně.

   Navíc současná verze formátu ODF nedefinuje standard pro zápis vzorců v tabulkovém kalkulátoru a je tudíž pro tabulky prakticky nepoužitelná. Prezentace v případě veřejné správy asi není potřeba publikovat v editovatelné podobě, takže v tomto konkrétním případě by bylo ODF používáno

[3] http://www.micr.cz/scripts/detail.php?id=3888

pouze pro ukládání textových dokumentů, pro které lze stejně dobře použít RTF s mnohem širší aplikační podporou bez nutnosti instalování nových produktů na straně uživatelů.

2. Není specifikováno, jaké vložené objekty se mohou vyskytovat uvnitř ODF a HTML dokumentů.

Ministerstvo to jistě myslelo dobře, ale podobné dokumenty odtržené od reality ve výsledku pomalou cestu k používání otevřených formátů spíše zbrzdí. Jak by doporučení mělo vypadat v dnešní době:

- *Pro zveřejňování dokumentů primárně upřednostňovat formát HTML, který je nejpřístupnější.*
- *Pokud povaha dokumentu není vhodná pro publikování v HTML (např. mapa), pak použít formát PDF. Formát PDF se může použít i jako alternativní doplnění HTML verze, pokud se jedná o dokument primárně určený pro tisk.*
- *Jiný formát než HTML nebo PDF je povolen pouze pro dokumenty, u kterých se předpokládá jejich následné upravování (šablony, formuláře apod.). Takový dokument musí být zveřejněn nejméně ve dvou z následujících formátů:*
  - *ODF*
  - *OOXML*
  - *RTF*
  - *DOC/XSL/PPT*

  Dokumenty nesmí obsahovat vložené binární objekty. Mohou obsahovat pouze obrázky v některém z formátů GIF, PNG, JPEG.

Takové doporučení by umožnilo bez nutnosti instalování nového software publikovat dokumenty, které se již dnes přečte kdokoliv. Za pár let by se doporučení mohlo upravit a v seznamu dovolených formátů ponechat jen ODF a OOXML a možná do něj přidat nějakého nástupce, který do té doby vznikne a převezme dobré myšlenky obou formátů.

4 Závěr

Konečně jsme se dočkali situace, kdy všechny hlavní kancelářské balíky jako svůj výchozí formát pro ukládání používají formát, který je otevřený, dobře zdokumentovaný a založený na XML. To je velká výhra pro uživatele i vývojáře.

Uživatel má mnohem větší šance, že dokument otevře i za deset let než tomu bylo v případě binárních formátů. Vývojáři se bude s kancelářskými dokumenty mnohem lépe pracovat v jeho aplikacích, protože může využívat obrovské množství nástrojů pro práci s XML, nemusí se trápit s binárním a nedokonale zdokumentovaným formátem. Ale ani samotné rozšíření formátů ODF a OOXML samo nevyřeší všechny problémy spojené s výměnou dokumentů.

Pod pokličkou ODF a OOXML

Jiří Kosek

E-mail: jirka@kosek.cz

Klíčová slova: XML, ODF, OOXML

Abstrakt

Formáty ODF a OOXML mají mnoho společného i rozdílného. Přednáška posluchače seznámí se základními principy těchto moderních formátů dokumentů a ukáže jejich podobnosti i odlišnosti. Součástí přednášky budou i praktické ukázky toho, jak snadno lze s těmito formáty pracovat díky tomu, že jsou otevřené a založené na rozšířených technologiích jako XML a ZIP.

Před pár lety většina kancelářských aplikací používala pro ukládání dokumentů proprietární binární formáty. Dnes je situace zcela odlišná – pro ukládání se používají standardizované a otevřené formáty Office Open XML (OOXML) a Open Document Format (ODF). První z nich je přitom výchozím formátem pro ukládání v MS Office 2007. ODF je dnes používáno hned několika kancelářskými balíky. Mezi ty nejznámější patří například OpenOffice.org a jeho deriváty, Google Docs a KOffice. Je tak jen otázkou času, kdy tyto formáty ve větším měřítku nahradí dnes dominující binární formáty DOC/XLS/PPT, a vývojáři aplikací budou stát před ukolem zpracování a generování dokumentů v těchto nových formátech.

1 Základní struktura souborů

Z hlediska vývojářů je dobrou zprávou, že formáty jsou založené na XML, takže je lze číst mnohem jednodušeji než starší binární formáty. Formát XML však není použit přímo. Kancelářské dokumenty typicky obsahují i vložené objekty jako třeba obrázky. Ty by se pro účely přímého vložení do XML musely bezpečně překódovat (např. metodou base64) a výsledkem by bylo uložení dokumentu do souboru o značné velikosti. Oba formáty proto od sebe oddělují samotný

obsah dokumentu (ten je reprezentován pomocí XML) a další binární objekty jako obrázky, na které z obsahu dokumentu vedou jen odkazy. Běžný uživatel by však s dokumentem roztroušeným do několika souborů nezvládl práci, a proto se všechny soubory reprezentující jeden dokument uloží do jednoho archivu ZIP.

To má hned několik výhod. Z pohledu uživatele je tak dokument reprezentován jedním souborem, takže se nestane, že při posílání emailem se jaksi zapomene poslat část dokumentu. Archiv ZIPu však neslouží jen jako obálka, ale automaticky také dojde ke komprimaci dat. Pro obrázky to nemá smysl, ale podstatně se sníží velikost částí uložených v XML. V mnoha případech tak dokumenty uložené v OOXML/ODF zaberou méně místa než ekvivalenty uložené do starších binárních formátů.

Chceme-li se podívat, jak jsou formáty navrženy, není nic snazšího než je otevřít jako archiv a podívat se dovnitř. Přejmenujeme-li původní příponu jako `.docx` nebo `.odt` na `.zip`, můžeme pro průzkum vnitřností využít zcela standardní nástroje.

Pojďme se nejprve podívat na formát ODF. Nejdůležitější součástí archivu je soubor `content.xml`, ve kterém je samotný obsah dokumentu. Další soubory obsahují metadata (`meta.xml`), informace o stylech (`styles.xml`) a nastavení aplikace (`settings.xml`). Odkazy na vložené objekty (obrázky) jsou identifikovány přímo pomocí relativní cesty k umístění objektu v archivu.

Formát OOXML používá obecnější mechanismus pro organizaci jednotlivých částí, ze kterých se dokument skládá, nazvaný Open Packaging Conventions (OPC). Obecná struktura OPC může být teoreticky namapována na různé další formáty, v praxi však dnes existuje a používá se jen jeden způsob mapování na archiv ZIP. V praxi je tedy i OOXML dokument obyčejný archiv ZIP obsahující několik souborů.

Formát OPC je nepřímý, proto musíme nejprve v archivu najít soubor `_rels/.rels` a v něm si zjistit, kde se v archivu nachází samotný soubor s tělem dokumentu, kde metainformace apod. Odkazy z dokumentu na vnořené objekty se opět nezapisují přímo, ale pomocí identifikátorů jejichž mapování na skutečné soubory je definováno v samostatném mapovacím souboru. Je na to potřeba dát pozor a číst umístění z těchto mapovacích souborů a nespoléhat se na jedno konkrétní umístění a pojmenování, které používá třeba MS Office.

2 Architektura dokumentů

Veškeré informace o dokumentu jsou v obou formátech ukládány v podobě XML. Protože kancelářské dokumenty obsahují různé druhy dat, nepoužívá se jedna monolitická sada elementů, ale několik specifických definovaných v oddělených jmenných prostorech.

Formát ODF tak definuje jmenné prostory pro celý dokument, pro metainformace, pro konfiguraci aplikace, pro textový obsah, pro tabulky, pro kresby, pro prezentace, pro 3D grafické objekty, pro animace, pro grafy, pro formuláře, pro skripty, pro definici stylů a pro popis formátu čísel. Kromě toho si ODF vypůjčilo některé elementy a atributy z jiných standardů jako XSL-FO, SVG a SMIL – zapisuje je však ve vlastních jmenných prostorech. V několika věcech ODF kompletně spoléhá na existující jazyky – konkrétně se jedná o popis metadat podle DublinCore, tvorbu odkazů pomocí XLinku, zápis matematických vzorců pomocí MathML a pro složitější formuláře je možné použít XForms.

OOXML používá rovněž několik jmenných prostorů. Základem jsou značkovací slovníky specifické pro jednotlivé typy kancelářských dokumentů – WordprocessingML, SpreadsheetML a PresentationML. Společně s nimi pak lze používat další elementy jazyků VML nebo DrawingML pro popis grafických objektů, OMath pro zápis matematických vzorců, DublinCore pro metadata a existuje i speciální sada elementů pro popis bibliografických údajů.

OOXML bývá někdy vytýkáno, že nestaví už na existujících standardech, ale většinu věcí si definuje po svém. Ve srovnání s formátem ODF můžeme však tuto výtku brát vážně snad jen v případě matematických vzorců. Standardy jako XSL-FO, SVG a SMIL nejsou do ODF převzaty jako celek, ale jsou účelově vybrány jen jejich části. Navíc na úrovni XML používají převzaté elementy a atributy změněný jmenný prostor. Při implementaci podpory ODF tak nejde snadno použít již existující kód, který dané jazyky podporuje. Formát pak obsahuje i mnoho případů, kdy se relativně související věci nastavují pomocí atributů z různých jmenných prostorů, protože některé vlastnosti byly převzaty ze SVG a jiné zase z XSL-FO.

Výsledkem je, že současná dokumentace k OOXML má až obludné rozměry (přes 6 000 stran), ale je v ní vše, co potřebujeme. Specifikace ODF je kratší, ale je plná odkazů na další specifikace, které však někdy odkazují ještě na další specifikace. Otázkou tak je, který z těchto dvou přístupů je lepší.

3 Základy reprezentace textu

Na detailní popis možností obou formátů nemáme samozřejmě prostor, pojďme se tedy alespoň podívat, jak by se v nich reprezentoval následující odstavec textu, který bychom přímo napsali v editoru.

Příliš **žluťoučký kůň** *úpěl ďábelské ódy.*[1]

Do formátu ODF se tento odstavec uloží jako:

```
<text:p text:style-name="Standard">Příliš
  <text:span text:style-name="T1">žluťoučký kůň</text:span>
```

[1]Dodejme, že slovo „ódy“ je kromě kurzívy zobrazeno i červenou barvou, která není při černobílém tisku patrná.

```
  <text:span text:style-name="T3">úpěl ďábelské</text:span>
  <text:span text:style-name="T4">ódy</text:span>
  <text:span text:style-name="T3">.</text:span>
</text:p>
```

Odstavec textu je uvnitř elementu `text:p` a změny formátování jsou obaleny elementem `text:span`. Vidíme, že pro ručně nastavené formátování se automaticky vytvoří styly, jejichž definice je na jiném místě dokumentu. Například definice stylu T4 pro červené slovo kurzívou je následující:

```
<office:automatic-styles>
  ...
  <style:style style:name="T4" style:family="text">
    <style:text-properties
      fo:color="#ff0000"
      fo:font-style="italic"
      fo:font-weight="normal"
      style:font-style-asian="italic"
      style:font-weight-asian="normal"
      style:font-style-complex="italic"
      style:font-weight-complex="normal"/>
  </style:style>
  ...
</office:automatic-styles>
```

Formát OOXML používá odlišný přístup. Odstavec je uložen v elementu `w:p`. Uvnitř odstavce však není přímo text, ale podelementy `w:r` s jednotlivými částmi textu. Každá část textu přitom má svůj textový obsah uvnitř elementu `w:t` a formátovací vlastnosti v `w:rPr`:

```
<w:p>
  <w:r>
    <w:t xml:space="preserve">Příliš </w:t>
  </w:r>
  <w:r>
    <w:rPr>
      <w:b/>
    </w:rPr>
    <w:t>žluťoučký kůň</w:t>
  </w:r>
  <w:r>
    <w:rPr>
      <w:b/>
    </w:rPr>
    <w:t xml:space="preserve"> </w:t>
  </w:r>
  <w:r>
    <w:rPr>
      <w:i/>
    </w:rPr>
    <w:t xml:space="preserve">úpěl ďábelské </w:t>
```

```
  </w:r>
  <w:r>
    <w:rPr>
      <w:i/>
      <w:color w:val="FF0000"/>
    </w:rPr>
    <w:t>ódy</w:t>
  </w:r>
  <w:r>
    <w:rPr>
      <w:i/>
    </w:rPr>
    <w:t>.</w:t>
  </w:r>
</w:p>
```

Elementy jako `w:i`, `w:b` a `w:color` pak slouží k nastavení vlastností částí textu. Styly se tak nevytváří automaticky jako v případě ODF, ale formátování může být uloženo přímo u textu. Samozřejmě pokud uživatel použije styly, jejich definice je i v OOXML oddělená od samotného textu.

Na první pohled vypadá reprezentace použitá ve formátu ODF elegantněji, má však i své problémy. Například není možné generovat ODF proudovým způsobem, protože všechny automatické styly musíme znát ještě před tím, než začneme generovat samotný obsah dokumentu. V OOXML však můžeme formátování vkládat průběžně do obsahu dokumentu.

4 Programová podpora pro práci s formáty

Vzhledem k tomu, že formáty ODF a OOXML jsou postaveny na rozšířených technologiích ZIP a XML, můžeme s nimi celkem rozumně pracovat jen za použití obecných knihoven pro práci s XML a formátem ZIP. Oba dva formáty lze velice úspěšně zpracovávat i pomocí XSLT transformace. Následující příklad je ukázka transformace v XSLT 2.0, která vygeneruje jednoduchou osnovu z libovolného ODF dokumentu používajícího nadpisy.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0"
                xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
                xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
                xmlns:dc="http://purl.org/dc/elements/1.1/">

  <!-- Parametr pro předání adresy dokumentu, který chceme zpracovávat -->
  <xsl:param name="url">file:OpenDocument-v1.0ed2-cs1.odt</xsl:param>

  <!-- Proměnné zastupující jednotlivé soubory uvnitř ODF souboru -->
  <!-- Schéma jar: umožňuje transparentní přístup k archivum ZIP/JAR -->
  <xsl:variable name="content"
                select="doc(concat('jar:', $url, '!/content.xml'))"/>
```

```
  <xsl:variable name="meta"
                select="doc(concat('jar:', $url, '!/meta.xml'))"/>

  <!-- Šablona, kterou transformace začíná -->
  <xsl:template name="outline">
    <html>
      <head>
        <title>Osnova dokumentu
         <xsl:value-of select="$meta/office:document-meta/office:meta/dc:title"/>
        </title>
      </head>
      <body>
        <!-- Průchod všemi nadpisy -->
        <xsl:for-each select=
             "$content/office:document-content/office:body/office:text/text:h">
          <!-- Vygenerování odpovídajícího HTML nadpisu - h1, h2, ... -->
          <xsl:element name="h{@text:outline-level}">
            <xsl:value-of select="."/>
          </xsl:element>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

V mnoha případech však chceme být od detailů formátů odstíněni. Pak je vhodnější využít nějakou knihovnu, která nabízí vysokoúrovňové rozhraní pro práci s dokumenty. Pro formát ODF je takovou knihovnou ODF Toolkit,[2] který je i základem balíku OpenOffice.org. Microsoft nabízí pro práci s OOXML vlastní SDK[3]. S rostoucí oblibou formátů se objevují další open-source i komerční knihovny.

Zajímavé bude sledovat, zda se objeví knihovny, které budou nabízet stejné API pro práci s oběma formáty – to by vývojářům umožnilo bez přidané práce ve svých aplikacích podporovat oba formáty. Už dnes je možné alespoň textové části formátů ODF a OOXML generovat z jedné předlohy při využití jazyka XSL-FO a vhodného formátovacího engine jako třeba XFC.[4]

5 Závěr

Nové formáty ODF a OOXML jsou velkým krokem kupředu. Pro uživatele znamenají větší jistoty při dlouhodobé archivaci dokumentů. Pro vývojáře zase formáty nabízejí velice snadnou možnost čtení i generování díky tomu, že jsou

[2] http://http://odftoolkit.openoffice.org/

[3] http://www.microsoft.com/downloads/details.aspx?FamilyId=AD0B72FB-4A1D-4C52-BDB5-7DD7E816D046

[4] http://http://www.xmlmind.com/foconverter/

založeny na XML. Práci vývojářů v budoucnu jistě usnadní i mnoho knihoven a dalších nástrojů, které se budou snažit o maximální odstínění od vnitřností formátů.

Reference

[1] ECMA-376. *Office Open XML File Formats.* ECMA International, 2006. http://www.ecma-international.org/publications/standards/Ecma-376.htm

[2] ISO/IEC 26300:2006. *Information technology – Open Document Format for Office Applications (OpenDocument) v1.0.* ISO, 2006. http://standards.iso.org/ittf/PubliclyAvailableStandards/c043485_ISO_IEC_26300_2006(E).zip

[3] *Open Document Format for Office Applications (OpenDocument) v1.1.* OASIS, 2007. http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.pdf

[4] http://openxmldeveloper.org/ – web s informacemi o zpracování OOXML

[5] http://opendocument.xml.org/ – komunitní web věnovaný formátu ODF

Generování dokumentů pro tabulkové kalkulátory

Štěpán Bechynský

E-mail: Stepan.Bechynsky@microsoft.com

Požadavek klienta, se kterým jsem se setkal snad na všech projektech, byl export do tabulkového kalkulátoru, konkrétně do aplikace Excel. Pokud se jednalo o „těžkého“ klienta, tak byl k dispozici i nainstalovaný Excel a šlo využít přímo objektů API Excelu. U webových aplikací, nebo u aplikací běžících na jiných OS než Windows, byla situace o něco složitější. U jednoduchých exportů se využívala vlastnost aplikace Excel, která bez zbytečného ptaní poslušně otevře soubor s příponou `.xls`. A to i v případě, že obsah souboru není ve formátu Excel. Tímto způsobem bylo možné podstrčit textový soubor oddělený tabulátory nebo tabulku v HTML i s formátováním, a uživatel nic nepoznal. Tuto „berličku“ však nelze využít v případě, pokud chce zákazník vzorce, grafy, kontingenční tabulky, podmíněné formátování a další funkce.

Office Open XML

Poslední verze Microsoft Office 2007, tedy i Excel 2007, ukládá soubory v novém otevřeném formátu, který byl v roce 2006 standardizován společností ECMA pod číslem ECMA-376. Formát Office Open XML (OOXML) je postaven na XML a ZIP kompresi, proto je možné využít jej na libovolné platformě. OOXML řeší i další problém, o kterém jsem se zatím nezmínil. Pokud se pro generování dokumentů dříve používal přímo Excel, bylo občas potřeba pro různé verze aplikace Excel (2000, XP, 2003) udržovat i různé verze kódu pro generování. Spolu s uvedením Microsoft Office 2007 však byla uvedena i aplikace Compatibility Pack (CP), která převádí OOXML do starších binárních formátů Microsoft Office a naopak. Tím pádem dokument OOXML otevřete i ve starších verzích Microsoft Office a můžete ho považovat za univerzální formát pro Microsoft Office. CP je samostatná aplikace nezávislá na Microsoft Office, proto ji lze využít i ve spolupráci s OpenOffice.org.

Generování dokumentu od „nuly“

Pokud se rozhodnete pro tuto variantu, tak musíte nastudovat specifikaci OOXML pro příslušnou část funkcionality, kterou potřebujete. Tato možnost je rozhodně nejnáročnější na znalosti vývojáře, ale na druhou stranu poskytuje největší volnost v generování dokumentů. Pokud se rozhodnete generovat dokumenty od „nuly“, musíte řešit v podstatě dvě věci – uložení XML do správné struktury (Open Packaging Conventions, OPC) a vlastní obsah XML. Uložení XML do struktury OPC vám při použití .NET Framework 3.0 zjednoduší knihovna *System.IO.Packaging* a při použití jazyka JAVA pomůže knihovna ze stránek http://www.openxml4j.org/. Na vás už pak je jen generování XML s obsahem. Pro .NET Framework 3.0 ještě existuje Open XML SDK, které obsahuje další knihovny pro manipulaci s obsahem OPC. Ty poskytují vyšší abstrakci než *System.IO.Packaging*, ale neoprostí vás od znalosti struktury XML souborů s obsahem dokumentu.

Úprava existujících dokumentů

Úprava existujících dokumentů bude pravděpodobně nejčastější variantou převodu informací do formátu OOXML. Princip je velmi jednoduchý. Pomocí aplikace Excel 2007 si vytvoříte vzorový dokument a v něm pak budete prostřednictvím vaší aplikace měnit jen potřebné části. To je usnadněno způsobem ukládání informací uvnitř ZIP archivu. Tedy co typ informace, to separátní soubor. Např. každý list je uložen ve zvláštním souboru, stejně tak formátování, texty atd. Díky tomu manipulujete jen s nezbytně nutnými daty a snižujete tím riziko, že poškodíte části dokumentu, které jste vůbec změnit nechtěli. Současně spotřebujete méně systémových zdrojů počítače, což je u serveru obzvláště důležité.

Využití hotových knihoven

Pokud vytváříte dokumenty, které nepotřebují speciální vlastnosti, je velmi pravděpodobné, že najdete hotovou knihovnu, která splní vaše požadavky. V současné době jsou dispozici převážně takové knihovny, které umožňují generovat dokumenty pro tabulkové kalkulátory. To je dáno zejména popularitou a možnostmi aplikace Microsoft Excel. Hotové knihovny vás úplně odstiňují od vnitřní struktury OOXML dokumentů. V praktické ukázce si předvedeme použití dvou knihoven. Pro platformu .NET to bude knihovna *ExcelPackage*. Po vytvoření objektu *ExcelPackage* je k dispozici rozhraní velmi podobné objektovému modelu aplikace *Excel*, které je známé především vývojářům ve VBA.

```
...
...
using (ExcelPackage xlPackage =
          new ExcelPackage(@"c:\temp\produkty.xlsx"))
{
    ExcelWorksheet worksheet =
          xlPackage.Workbook.Worksheets.Add("Products");
    cmd.Connection.Open();
    SqlDataReader rst = cmd.ExecuteReader();
    int r = 1;
    while (rst.Read())
    {
        for (int c = 0; c < rst.FieldCount; c++)
        {
            worksheet.Cell(r, c+1).Value = rst[c].ToString();
        }
        r++;
    }
    rst.Close();
    cmd.Connection.Close();
    xlPackage.Save();
}
...
...
```

Vývojářům v PHP je zase určena knihovna *PHP Excel 2007 classes*. U této knihovny není bez zajímavosti, že se na vývoji podílí český PHP guru, pan Jakub Vrána. Jak může vypadat generování dokumentu pro tabulkový kalkulátor pomocí PHP, ukazuje následující příklad:

```
<?php
include "PHPExcel.php";
include "PHPExcel/Writer/Excel2007.php";

$date = "2007-09-01";

$quotations = array();
$result = mysql_query("Ď");
while ($row = mysql_fetch_assoc($result)) {
    $quotations[$row["mark"]][] = $row;
}
mysql_free_result($result);
```

```
$excel = new PHPExcel;
$excel->getProperties()->setTitle("Akcie $date");
$excel->removeSheetByIndex(0);

$columns = array("name" => "A", "d_quotation_buy" => "B",
"d_quotation_sale" => "C", "v_quotation_buy" => "D",
"v_quotation_sale" => "E");
foreach ($quotations as $mark => $rows) {
    $sheet = $excel->createSheet();
    $sheet->setTitle($mark);
    $sheet->getColumnDimension('A')->setWidth(30);
    $sheet->setCellValue("A1", "Směnárna");
    $sheet->setCellValue("B1", "Devizy nákup");
    $sheet->setCellValue("C1", "Devizy prodej");
    $sheet->setCellValue("D1", "Valuty nákup");
    $sheet->setCellValue("E1", "Valuty prodej");
    foreach ($rows as $i => $row) {
        foreach ($columns as $key => $sl) {
            $sheet->setCellValue($sl . ($i + 2), $row[$key]);
        }
    }
 }

$writer = new PHPExcel_Writer_Excel2007($excel);
$writer->save("$date.xlsx");
?>
```

Zdroj: http://php.vrana.cz

Obě knihovny lze stáhnout, a to včetně zdrojových kódů, ze serveru Codeplex (http://www.codeplex.com).

Závěr

Díky použitému způsobu ukládání informací ve formátu Office Open XML je manipulace s obsahem dokumentů velmi snadná. Podpora ZIP komprese a analýzy XML je dostupná na všech platformách a pro všechny programovací jazyky. Vývojář tak není vázán na jedinou platformu a má mnohem více možností jak pracovat s formátem pro kancelářské balíky.

Technologie moderních procesorů

Zdeněk Salvet

E-mail: salvet@ics.muni.cz

Klíčová slova: SSE, Hyper-Threading, multicore, power saving, NUMA, MCE

Abstrakt

Tento příspěvek se snaží stručně vysvětlit nejdůležitější nové technologie, které se objevily v procesorech IBM PC-kompatibilních počítačů v posledních letech a jejich dopad na administrátora Linuxového systému.

Abstract

This article tries to explain most important new technologies that appeared in processors of IBM PC compatible computers in last several years and their importance for Linux-based system administrator.

V posledních letech došlo k poměrně dramatickému vývoji v oblasti architektury PC-kompatibilních počítačů a procesorů v nich užívaných (stejných deset let jako uběhlo od uvedení Pentia s MMX technologií trval předtím vývoj od procesoru Intel 386 k Pentium Pro, prvnímu intelskému procesoru s prováděním instrukcí mimo pořadí), objevilo se mnoho nových vlastností a funkcí, některé specifické pro PC, některé odvozené ze starších velkých systémů (mainframy, velké SMP stroje SGI a IBM a podobně). Tento příspěvek si klade za cíl shrnout základní informace o nejdůležitějších z nich a připomenout tak čtenáři, co úžasného má pravděpodobně k dispozici v útrobách svého osobního počítače nebo serveru nebo co, kromě různých kvantititativních hledisek, zvažovat při nákupu a konfiguraci nového stroje.

1 Ještě více instrukcí

x86 zůstává asi jedinou rodinou procesorů, která se drží architektury CISC. Neplatí to stoprocentně, používání jednodušších instrukcí je v nových implementacích obvykle z pohledu výkonu výhodnější a vnitřní mikroarchitektura procesorů se přibližuje RISC (a snaží se využívat zřetězené a superskalární zpracování instrukcí), na druhou stranu ale pokračuje neustálé rozšiřování instrukčních sad novými instrukcemi. Nové bloky instrukcí byly ze začátku navrhovány především

pro urychlení výpočtů ve specifických aplikačních oblastech jako je zpracování videa nebo 3D zobrazení, později se přidala i obecněji použitelná rozšíření.

První rozšíření typu SIMD (jednou instrukcí se aplikuje stejná operace na více datových položek) nazvané *MMX* bylo uvedeno Intelem v roce 1997. Poskytuje 8 nových vektorových registrů (bohužel sdílených s registry x87 FPU), které se logicky mohou dělit na dvě 32bitová až 8 osmibitových čísel, a nad nimi různé celočíselné operace, které se dají výhodně využít zejména v aplikacích pro digitální zpracování signálů (DSP). Firma AMD reagovala dalším rozšířením *3DNow!*, které k MMX přidalo SIMD operace v pohyblivé desetinné čárce (dvě FP čísla v jednom registru) a přednačítání z paměti (prefetch). Později bylo ještě doplněno dalšími instrukcemi pod názvem *Enhanced 3DNow!*, ale nebylo přejato Intelem, takže je podporováno jen procesory AMD a některými menšími výrobci (VIA, IDT).

Nejvýznamnějšími rozšířeními byly *Streaming SIMD Extensions (SSE)* (poprvé v Pentiu III), které definovaly 8 nových 128bitových registrů XMM (nesdílených s x87 FPU) a 70 single-precision (32b) FP operací na nich a *SSE2*, přidávající double-precision (64bitové) vektorové operace a rozšiřující MMX operace na XMM registry. Tato dvě rozšíření se dají realisticky využít i bez detailních znalostí jejich instrukcí nebo předpřipravených knihoven, protože jsou velmi obecná a relativně dobře je už zvládají některé kompilátory, samozřejmě zejména ty od Intelu.

Vývoj pokračoval méně významným *SSE3* a patrně bude pokračovat již ohlášenými *SSE4* (další vektorová primitiva, podpora zpracování řetězců, od Intelu) a *SSE5* (AMD).

2 Dva v jednom

Jedním ze způsobů, jak zvýšit reálně využitý paralelismus při vykonávání instrukcí v superskalárním procesoru, je provádění instrukcí z více vláken nebo procesů zároveň – další vlákna mohou využít zdroje, které první vlákno nechává nevyužité, například kvůli čekání na čtení dat z paměti. Existuje více variant tohoto přístupu (např. střídání po cyklech, přepínání při při čekání s velkou latencí) , Intel přišel v Pentiu 4 s implementací umožňující procesoru provádět více vláken i v rámci jednoho taktu (Simultaneous multithreading – SMT), nazvanou *Hyper-Threading Technology (HTT)*. Funguje tak, že zdvojuje zdroje, které uchovávají zvnějšku viditelný stav procesoru (registry), ale ne jednotky pro provádění. Dopad na celkový výkon je hodně aplikačně závislý, někdy může dojít i k jeho snížení (vlákna soutěží o omezené zdroje jako je cache), v příznivých případech se udává až 30% nárůst. Navenek se pak jeden fyzický procesor tváří jako dva logické, což dovoluje i operačnímu systému, který umí pracovat s více procesory, ale nezná HTT, využít jejích výhod. V případě SMP systému je ale velmi vhodné mít v operačním systému zapnutou podporu pro HTT („SMT (Hy-

perthreading) scheduler support“ v Linuxu), aby plánovač procesů neplánoval nevhodně více procesů na jeden fyzický procesor, zatímco jiný zahálí.

Jinou cestou, jak využívat paralelismu vláken v aplikacích a celých systémech a zároveň výsledků vývoje v výrobě procesorů (menší prostorové nároky, lepší zvládnutí produkce tepla procesory), je umístění více procesorových obvodů (jádro, core) do jednoho integrovaného obvodu nebo alespoň společného pouzdra. Výsledek je podobný klasickému multiprocesorovému systému, za ideálních podmínek zdvojnásobení počtu jader (v oblasti PC jsou dnes dostupné až čtyřjádrové procesory) může vést ke zdvojnásobení výkonu, ale je třeba stále počítat s tím, že jádra mezi sebou sdílejí některé zdroje, přinejmenším systémovou sběrnici, případně cache apod. Podobně jako u HyperThreadingu je pro optimální využití strojů s více fyzickými procesory radno aktivovat podpůrné funkce plánovače procesů operačního systému.

3 Někdo to rád pomalejší

Je mnoho důvodů k úsporám energie – snížení množství produkovaného tepla a hluku u stolních počítačů, prodloužení doby běhu na baterii u přenosných, úspory nákladů na energii a chlazení u velkých instalací. Moderní procesory nabízejí k ovlivnění spotřeby elektrické energie i v době, kdy musí něco provádět a nemohou být úplně zastaveny (např. instrukcí HLT u starší procesorů nebo MWAIT u nových čipů Intelu, případně přechodem celého systému do spícího režimu) několik technik, které bývají souborně označovány zajímavými názvy jako PowerNow!, SpeedStep, PowerSaver. Vesměs zahrnují možnost přepínání mezi různými taktovacími frekvencemi a napájecím napětím (napětí potřebné k provozu procesoru závisí na frekvenci), novější procesory Intel Core umí vypínat některé obvody, které nejsou aktuálně využívané, ve druhé generaci procesorů Pentium-M je dokonce možné omezovat velikost používané L2 cache. Pro kontrolu a ochranu proti přehřátí mají procesory zabudována teplotní čidla, v krajních případech se některé dokáží automaticky zpomalit nebo zastavit, aby se předešlo poškození.

Linuxové jádro nabízí k řízení taktovací frekvence procesoru interface modulu *cpufreq*, který umožňuje nastavení maximální a minimální frekvence a schématu pro automatické nastavování („performance“, „powersave“, ...), pokud nenecháme aktualizaci nastavení přímo na uživateli nebo user-space programu (např. *powersaved*), což dovoluje implementaci složitějších schémat a jednodušší provázání akcí se zbytkem systému. Vše samozřejmě probíhá v mezích schopností jednotlivých podporovaných typů procesorů.

4 Není paměť jako paměť

Od generace Pentia Pro a procesorů jemu konkurujících je možné nastavovat způsob přístupu procesoru k různým oblastem adresního prostoru prostřednic-

tvím speciálních registrů procesoru – *Memory Type Range Register (MTRR).* Kromě používání vyrovnávací paměti (režim write-back nebo write-through) lze nastavit také režim spojování zápisů (write-combining). Když je tento režim povolen, procesor může spojit několik menších zápisů do dané oblasti v jeden větší přenos na vnější sběrnici procesoru a PCI/AGP, což může výrazně zvýšit efektivní rychlost přenosu snížením režie transakcí na sběrnicích. Často bývá tento režim užitečný pro operace v ovladačích videokaret, proto je vhodné zpřístupnit MTRR registry X serveru příslušnou volbou v konfiguraci jádra (prohlížet nebo manuálně nastavovat MTRR je pak možné pomocí speciálního souboru `/proc/mtrr`).

5 Není paměť jako paměť 2

S procesory AMD Opteron přišla do světa běžných PC architektura ve které již není veškerá paměť soustředěna na jednom místě, jednom řadiči paměti (v SMP systému s Opterony může mít každý procesor lokální pamět, pro kterou má zabudouván řadič paměti), a přístup ke všem jejím oblastem již není stejně rychlý (rychlost přístupu závisí na topologii propojení mezi procesory a rychlosti linek) – *Non-Uniform Memory Architecture (NUMA).* U takovéto architektury je obvykly velmi užitečné, když jádro systému, případně i aplikace ve spolupráci s jádrem, dokáže minimalizovat komunikaci procesoru se vzdálenými oblastmi paměti tak, že alokuje datatové oblasti pokud možno poblíž procesoru, ze kterého budou určitě nebo pravděpodobně používány. Hodně práce bylo v tomto směru uděláno i v jádře Linuxu, v případě potřeby vyždímání maximálního výkonu ze stroje ale nezaškodí zamyslet se nad rozdělením paměťových modulů do dostupných pozic na základní desce (pro standadrdní systém bývá nejlepší rovnoměrné rozdělení na jednotlivé paměťové kanály) a možností přiřazení konkrétních procesorů a bloků paměti aplikacím nebo subsystémům pomocí rozhraní *cpusets.*

6 Když je něco špatně

Moderní procesory disponují také také interním subsystémem, který umožňuje detekovat a zaznamenávat vnitřní chyby ve fungování procesoru nebo chyby na procesor navazujících kompoment (sběrnice, paměti), tzv. *Machine Check Architecture (MCA).* Dříve byly takové subsystémy součástí designu strojů třídy mainframů, ve světě PC-kompatibilním se objevily s Pentiem. Protože obvykle v případě problémů (a problémy s procesorem bývají většinou důležité) chceme znát co nejvíce informací o příčinách, je dobré mít zapnutu podporu pro ohlašování událostí od MCA uživateli, v Linuxu viz konfigurační volby jádra „Machine Check Exception“, „Intel MCE features“ a související. V případě problémů se zapnutou podporou MCE (nestandardní stroj) je možné ji vypnout i při zavádění jádra volbou „nomce“.

CMT (Chip Multithreading) architektura procesorů

Tomáš Okrouhlík

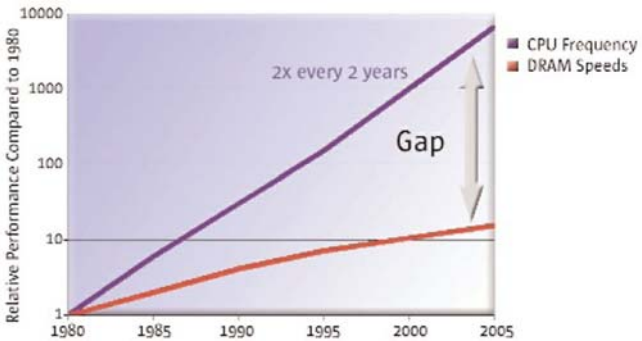
E-mail: Tomas.Okrouhlik@Sun.com

Abstrakt

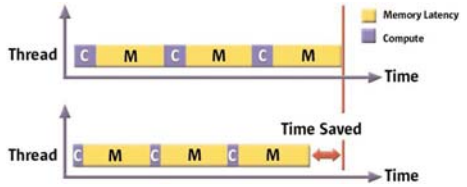
Představení jiného pojetí procesorové architektury – openSPARC. Jedná se o procesory UltraSPARC T1 a UltraSPARC T2 založené hlavně na velké míře paralelizmu více jader (až 8) a více vláken (až 64) na jeden chip. Optimalizace OS a aplikací na tento typ architektury. Výhody a nevýhody daného řešení.

Úvod

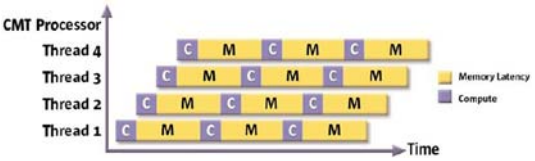
Během posledních dvaceti let se vývojáři procesorů snažili zvyšovat výkon procesorů hlavně za pomocí zvyšování jejich frekvence a dalších technologií jako jsou například vícečetné instrukce, odhadování větvení a mnoho dalších obecně se těmto typům optimalizace říká ILP (Instruction Level Parallelism). Výsledkem jsou procesory, které mají opravdu značný výkon v jedno vláknových operacích a jako boční efekt mají tyto procesory problémy s odvodem tepla a velikou spotřebu elektrické energie. Mnohem vážnějším nedostatkem se však ukazuje rozdíl mezi rychlosti procesorů a rychlostí pamětí. Neboli procesory dnes stráví mnoho času čekáním na data z paměti. Navíc se ukazuje, že se situace v tomto směru nebude zlepšovat, ba naopak rozdíl rychlosti cpu a rychlostí pamětí se bude nadále zvětšovat.



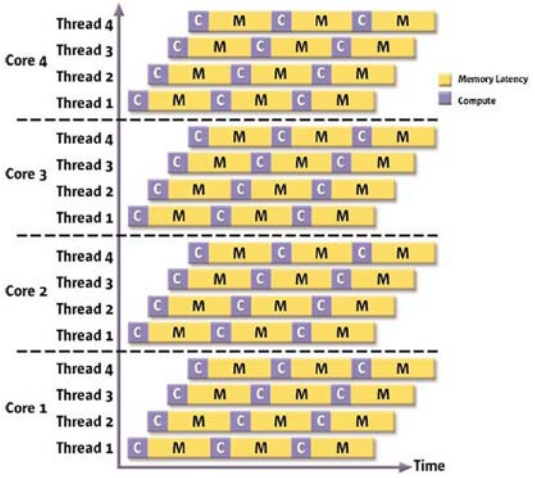
Díky tomuto efektu rozvýrajících se nůžek se ukázalo, že další snaha o zvýšení výkonu procesorů pomocí vylepšování instrukčního paralelizmu ILP přináší jen malé výsledný efekt viz následující obrázek, kde se ukazuje jen malá úspora v potřebném čase pro spočtení dané úlohy.



Proto přišli vývojáři s řešením pokusit se zvýšit výkon procesoru pomocí paralelizace více vláken (TLP = Thread Level Parallelism). Začali proto vyvíjet procesory, které mohou v rámci jednoho jádra zpracovávat více vláken. Někdy se těmto procesorům také říká CMT (Chip Multy Thread cpu).



Zároveň začali vývojáři také navrhovat procesory s více jádry na jednom chipu. Sun Microsystem obě tyto koncepce zahrnul do svého projetu procesorů UltraSPARC T1 Niagara. Mluvíme tady o procesorech, které mají více jader a každé jádro může vykonávat více vláken.



V případě procesorů UltraSPARC T1 Niagara se jedná o osm jader a každé může vykonávat až čtyři paralelní vlákna. Dohromady tak procesor UltraSPARC T1 Niagara bude zpracovávat paralelně až 32 nezávislých vláken a pro operační systém se bude jevit jako 32 procesorový server.

Historie SPARC

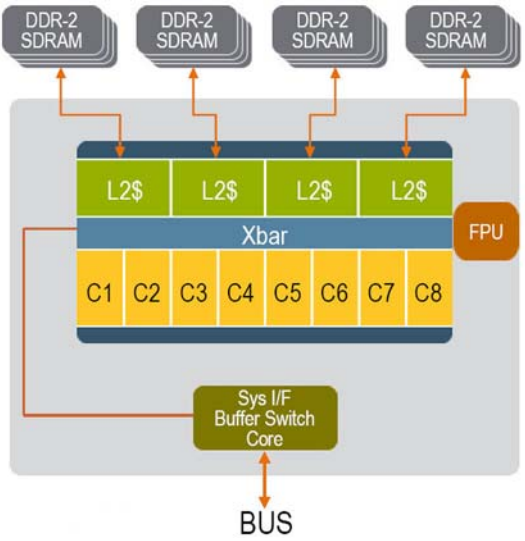
Architektura procesorů SPARC (Scalable Processor ARChitecture) se řadí do rodiny RISC (reduced instruction set computer) procesorů. Jde o standard, který se zrodil v rámci asociace „SPARC International“ v roce 1986 (SPARC V7 Version 7). Dnes je aktuální devátá verze SPARC V9 z roku 1993, kterou splňují produkty od firem Sun Microsystems a Fujitsu. Firma Sun Microsystems později (1995) uvedla vlastní rozšíření standardu tzv. UltraSPARC. Na začátku roku 2006 pak bylo v rámci projektu openSPARC uveřejněno a otevřeně poskytnuto v rámci OpenSource komunity pod pod SCSL licencí (http://www.opensparc.net) její poslední rozšíření. V rámci toho uvolnění se zde představuje poslední architektura CMT procesoru UltraSPARC T1 Niagara. V roce 2007 je pak představena a v rámci projektu openSPARC publikována architektura dalšího procesoru UltraSPARC T2 (Niagara2). Oba zmíněné procesory rozšiřují předcházející standard SPARC V9 o novou implementaci „sun4v“.

To v praxi znamená, že na procesorech splňující standard SPARC V9 běží stejný operační systém Solaris SPARC a lze na něm provozovat veškeré aplikace vyvíjené pro tyto standardy. Zpětná kompatibilita je tak teoreticky od procesorů UltraSPARC, prakticky však od UltraSPARC II tj. rok 1997. Který jiný OS toto dokáže, bez úprav kernelu, nebo distribuce?

UltraSPARC T1

Procesor má 8 jader, každé jádro obsahuje 64 bit sparc pipeline běžící na 1,2 GHz včetně 16 KB instrukční cache a 8 KB datové cache. Každé jádro obsahuje plánovač, který zajišťuje vykonávaní vždy toho aktuálního vlákna. Vlákna v rámci jednoho jádra jsou v tzv. „thread group“. Všechny jádra sdílejí 3 MB L2 cache. Tato cache logicky rozčleněná na tzv. banky, které jsou namapované na memory kontroléry. Aby nedocházelo ke konfliktům mezi velkým počtem vláken a L2 cache je tato cache dvanácti cestně asociativní. Interconnect mezi jednotlivými jádry a L2 cache je pomocí crossbaru, který poskytuje propustnost až 134 GB/s. Propustnost mezi L2 cache a externími DDR moduly je cca 23 GB/s. Externí paměť je tvořená 144 bitovými DDR2 moduly na 333 MHz a její celková velikost může mít kapacitu až 128 GB (při dnešních 4 GB DIMM modulech je to 64 GB) Procesory jsou vyráběné 90 nm technologií. procesor má jen jednu FPU (floating-poin unit).

Blokové schéma procesoru Ultra SPARC T1



Výhody

- Výkon serveru s jedním procesorem Ultra SPARC T1 s 8 jádry na 1,2 GHz je srovnatelný jako výkon 4× Ultra SPARC IV+ na 1,5 GHz, nebo jako 12× Ultra SPARC III na 1,2 GHz.
- Vysoký výkon u paralelizovatelnýcgh a škálovatelných aplikací. Java aplikační servery, databáze.
- Maskování latence paměti díky paralelnímu běhu několika vláken. Z toho dokáže profitovat například Oracle databáze, která navíc velmi dobře škáluje na všech 32 vláknech a při zpracovávání velkého množství transakcí není brzděna latencí paměti.

Nevýhody

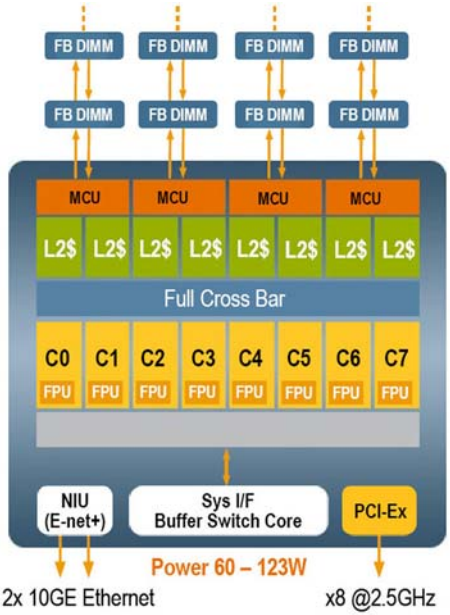
- Malý výkon u single treded aplikací.
- Malý výkon pro operace s plovoucí čárkou.
- Úzké hrdlo na i/o operacích směřujících na ethernet.

UltraSPARC T2

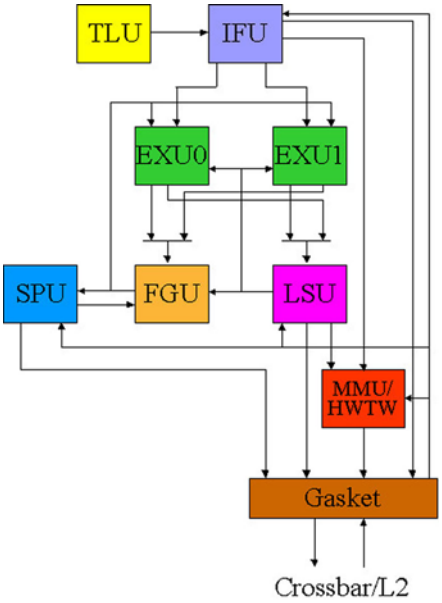
Nový procesor má stejně jako jeho předchůdce 8 jader a jeho hlavní architektonické vylepšení jsou:

- Zvětšil se počet vláken na jádro, z 4 na 8 což vedlo k cca dvojnásobné propustnosti oproti T1.
- Zvětšil se počet EXU (Integer Execution Units) jednotek z jedné na jádro na dvě na jádro. To vedlo k dalšímu zlepšení propustnosti.
- Vylepšila se práce s L1 a L2 cache, to vedlo ke zlepšení single threaded výkonu o cca 40 %.
- Zlepšil se výkon operací s plovoucí čárkou FP, došlo k cca desetinásobnému zlepšení oproti procesoru T1, kde byla jedna FPU (Floating Point Unit) jednotka na cpu, u nového procesoru T2 je jedna FPG (Floating Point and Graphics Unit) na jádro. Vlákna běžící FP operace neblokují ostatní operace.
- Přímo na úroveň cpu bylo přidáno rozhraní s 10 Gb Ethernetem (2× line), Layer 1, 2, 3, 4 protocol. stack, s výkonem cca 30 M paketů/s na každé rozhraní.
- Bylo použito rychlých a vysoce propustných „Fully Buffered“ pamětí FB-DIMM.
- Zlepšila se kryptovací jednotka (podpora více algoritmů).
- Přidal se instrukční L1 cache buffer.
- Rozšířila se asociovatelnost L1 Instrukční cache z 4 na 8.
- Zvětšila se velikost DTLB (Data Translation Look-aside Buffer) z 64 záznamů/jádro na 128 záznamů/jádro.
- Rozšířila se L2 cache z 3 MB na 4 MB.
- Zvětšila se asociovatelnost L2 cache z 12 na 16.
- Zvětšil se počet banků pro L2 cache ze 4 na 8 (to zvětšilo výkon L2 cache o 15 %).
- Procesory jsou vyráběné 65 nm technologií.

Blokové schéma procesoru UltraSPARC T2



Blokové schéma jednoho jádra procesoru UltraSPARC T2



IFU – Instruction Fetch Unit
16 KB I$, 32 B lines, 8-way SA
64-entry fully-associative ITLB

EXU0/1 – Integer Execution Units
4 vlákna sdílejí jednu jednotku
8 register windows/thread
160 IRF záznamů/vlákno (entries/thread)

LSU – Load/Store Unit
8 vláken se dělí o jeden LSU
8 KB D$, 16B lines, 4-way SA
128-entry fully-associative DTLB

FGU – Floating-Point/Graphics Unit
8 vláken se dělí o jeden FGU
32 FRF záznamů/vlákno (entries/thread)

SPU – Stream Processing Unit
Neboli kryptograficky koprocesor, který se skládá ze dvou nezávislých jednotek
Modular Arithmetic Unit, for RSA, binary and integer polynomial elliptic curve (ECC)
Cipher/Hash Unit for RC4, DES/3DES, AES-128/192/256, MD5, SHA-1, SHA-256
navržen tak aby „stihl“ kryptovat load od obou 10 Gb/s eth. rozhraní

TLU – Trap Logic Unit
Udržuje informace o stavu mašiny, řídí výjimky (exceptions) a přerušení (interrupts)

MMU – Memory Management Unit
Hardware tablewalk (HWTW)
8 KB, 64 KB, 4 MB, 256 MB pages

Výhody

- 2× více vláken.
- 2× větší propustnost. v Integer operacích.
- 8× až 10× větší propustnost v FP operacích.
- Až 1,4× větší výkon u integer jedno vláknových operací.
- Až 5× větší výkon u FP jedno vláknových operací.

- Až 2× více paměti.
- Až 3× větší propustnost paměti.
- Až 2× větší propustnost na I/O (PCI-E, HD, USB, 4× 1 Gb/s ethernet).
- Až 6× větší propustnost na Ethernetu (2× 10 Gb/s).
- Kryptovací koprocesor cca 60 M paketů za sekundu.

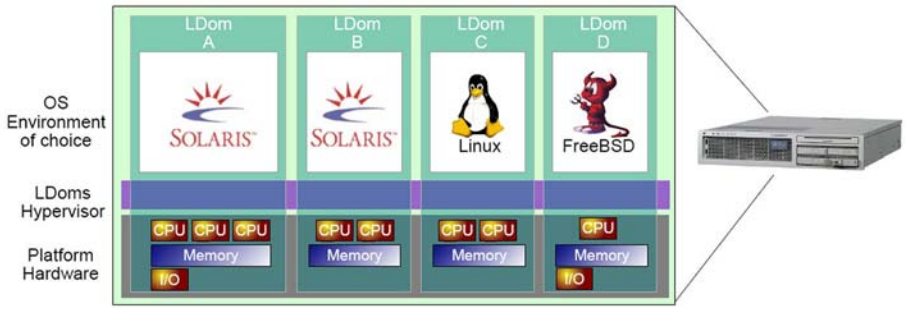
Nevýhody

- Slabý výkon pro pro single threded aplikace.

Porovnání parametrů procesorů Ultra SPARC T1 a Ultra SPARC T2 je v tabulce 1.

Virtualizace

Další velmi důležitou vlastností nových procesorů je jejich přínos na poli virtualizace. Lze totiž vytvářet tzv. logické domény LDOM (Logical Domain). Do těchto domén lze následně instalovat samostatné instance operačního systému. Dnes tak existují na tyto procesory portace OS FreeBSD a linuxu Ubuntu, Gentoo a dalších. Granularita pro vytváření logických domén je u tohoto modelu velmi vysoká tj. minimum pro jednu logickou doménu je jedno vlákno. Neboli na procesoru UltraSPARC T1 mohu vytvořit 32 logických domén a na procesoru UltraSPARC T2 jich mohu vytvořit 64.



Standard OpenSPARC totiž rozšiřuje zaběhnutý exekuční model standardu SPARC V9 použity v architektuře „sun4u“ o vrstvu hypervizoru.

Tabulka 1

	Ultra SPARC T1	Ultra SPARC T2
kódový název	Niagara	Niagara 2
počet jader	4, 6, 8	4, 6, 8
počet vláken na jádro	4	8
max. počet vláken (virtuálních cpu)	32	64
frekvence	1,0/1,2/1,4 GHz	1,4 GHz
počet EXU (Integer Execution Units)	8 (1/core)	16 (2/core)
počet FPU (Floating Point Unit)	1 (1/cpu)	8 (1/core)
počet MAU (Modular Arithmetic Module) SSL	8 (1/core)	8 (1/core)
L1 cache		
Instrukční cache na jádro/asociovatelnost	16 KB/4×	16 KB/8×
Datová cache na jádro/asociovatelnost	8 KB/4×	8 KB/4×
ITLB (Instruction Translation Look-aside Buffer)	64/core	64/core
DTLB (Data Translation Look-aside Buffer)	64/core	128/core
L2 cache		
velikost L2 cache	3 MB	4 MB
asociovatelnost L2 cache	12×	16×
počet banků L2 cache	4	8
crossbar		
propustnost read/write	134,4 GB/s	280 GB/s
propustnost pin	n/a	400 GB/s
externí paměť		
typ externí paměti	DDR2	FBDIMM
frekvence ext. paměti	333 MHz	667 MHz
počet a typ kanálů k ext. paměti	4× single channel 144b	4 dual channel, 168 b northbound, 120 b southbound
propustnost k ext. paměti	23,9 GB/s	63 GB/s
počet dimm modulů	16	16
velikosti paměťových modulů	512 MB, 1 GB, 2 GB, 4 GB	1 GB, 2 GB, 4 GB, 8 GB
maximální velikost paměti	64 GB	128 GB
I/O periferie procesoru		
Network interface unit	n/a	2× 3,125 Gb/s
i/o bus	128 b@250 MHz (3,2 GB/s)	256 b@350 MHz (11,2 GB/s)
teoretická max. propustnost cpu – i/o	3,2 GB/s	12 GB/s
efektivní propustnost cpu – i/o (implementace i/o rozhraní)	2,5 GB/s	4 GB/s
Fyzikální parametry		
Příkon cpu	< 80 W	84 W
počet tranzistorů	300 M	503 M
plocha die	378 mm^2	342 mm^2

Schéma architektury sun4u známé z procesorů UltraSPARC I, II, IIi, III, III+, IIIi, IV, IV+.

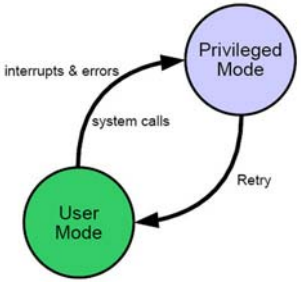


Schéma architektury sun4v známé z procesorů UltraSPARC T1, T2.

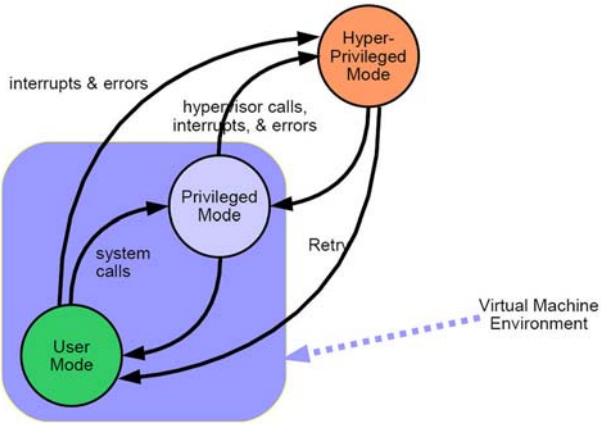
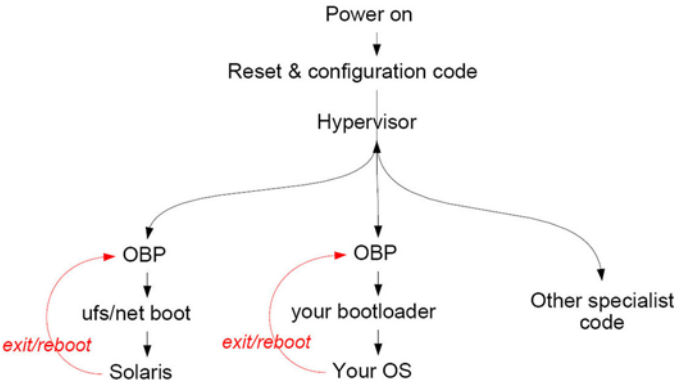


Schéma procesu bootování u logických domén.



Závěr

Tyto procesory rozhodně nejsou pro univerzální použití, nicméně je mnoho aplikací a typu použití, kde tyto procesory mají své uplatnění. Následující tabulka dobře napovídá, že procesor UltraSPARC T1 byl doménou zejména webových a java aplikačních serverů a procesor UltraSPARC T2 toto ještě rozšiřuje i do oblasti kryptování a TPC-C aplikací. Ve spojení s možností virtualizace tak dostáváme velmi zajímavý nástroj a je jen na nás jak ho použijeme.

Test	***Aplikační úloha***	***ILP Instrukční paralelizmus***	***TLP Vláknový paralelizmus***	***Objem zpracovávaných dat***	***Datová závislost***
Web99	Web server	Nízký	Vysoký	Velký	Malá
JBB	Java J2EE aplikační server	Nízký	Vysoký	Velký	Střední
TPC-C	Transakční aplikace	Nízký	Vysoký	Velký	Velká
SAP-2T	ERP aplikace	Střední	Vysoký	Střední	Střední
SAP-3T	ERP aplikace	Nízký	Vysoký	Velký	Velká
TPC-H	Aplikace pro podporu rozhodování	Vysoký	Vysoký	Velký	Střední

Literatura

[1] http://en.wikipedia.org/wiki/Microprocessor

[2] http://en.wikipedia.org/wiki/SPARC

[3] http://en.wikipedia.org/wiki/UltraSPARC_T1

[4] http://en.wikipedia.org/wiki/UltraSPARC_T2

[5] http://www.opensparc.net/publications

[6] http://www.sun.com/processors/niagara/M45_MPFNiagara2_reprint.pdf