Posudek oponenta bakalářské práce

Autor/autorka práce: **Vojtěch Kinkor**
Název práce: **Využívání .NET assembly z neřízeného C++**

„1. Prozkoumejte stávající možnosti využívání funkcionality z .NET assembly v neřízeném C++ kódu na různých platformách (Windows, Linux, Mac, aj.) a proveďte jejich srovnání."

Vedle platformy MS Windows student dále zmínil ještě existenci projektu Mono, který lze použít na platformách Linux a Mac. Možnost použití Wine, popř. NET Core, není vůbec zmíněna. Pro platformu Windows student zmínil techniky C++ interop (dříve nazývané „It Just Works"), P/Invoke, COM API a hostování CLR. U projektu Mono zmínil možnost hostování. Po pročtení (dle mého názoru málo) citovaných zdrojů se ukazuje, že student provedl pouze kompilaci různých článků, které našel na Internetu. Ale už si nezkusil jednotlivé techniky sám naprogramovat a řádně vyzkoušet – což se projevilo na kvalitě dalšího textu. Zadání blíže ovšem nespecifikuje, jak dalece má student „prozkoumávat a srovnávat".

„2. Po dohodě s vedoucím práce navrhněte a implementujte nástroj, který pro zadanou .NET třídu (resp. assembly) automaticky vytvoří C++ wrapper sloužící pro transparentní volání řízeného kódu zadané .NET třídy z neřízeného C++ kódu".

Student se při vypracování tohoto úkolu omezil na VC++ za použití C++ interop. Jenomže, C++ interop není nic jiného než syntaktický cukr VC++, který vkládá tzv. thunks do výsledného, tzv. mixed-mode kódu. Tyto thunks zaobalují volání P/Invoke na předem známé adresy. Ve výsledku tak VC++ umožňuje programátorovi běžným, naprosto transparentním přístupem volat řízený .NET kód z neřízeného C++ (studentova práce naopak tuto transparentnost skrývá). Přičemž C++ interop poskytuje jak typovou kontrolu, tak i optimální přenos parametrů, aby co nejvíce redukoval režii marshallingu. Tato teorie v prvním bodě zadání naprosto chybí. Výsledkem je pak to, že student dále zaobaluje tento syntaktický cukr, aniž by vůbec v práci jakkoliv osvětlil, jaký to má vůbec praktický smysl.

Student sice hojně cituje články z MS Developer Network, ale např. článek „Performance Considerations for Interop (C++)" (https://msdn.microsoft.com/en-us/library/ky8kkddw.aspx) chybí. V tomto článku je výslovně uvedeno, že se má z výkonnostních důvodů omezit počet přechodů mezi řízeným a nativním kódem. Správný postup je tedy vytvořit #pragma managed funkci, která bude pracovat s .NET kódem, a kteroužto funkci pak bude volat neřízený C++ kód. Namísto toho student vytvořil generátor wrapperu .NET třídy, který uskutečňuje dva přechody mezi řízeným a nativním kódem pro každé volání metody objektu .NET třídy, a k tomu se ještě přidává nemalá režie studentova marshallingu.

Bez řešení wrapperu jako multiplatformního nástroje mi celá práce vůbec nedává smysl. Celé automatické generování C++/CLI mostu v prostředí VC++ nepřináší žádné benefity – jenom přidává prostor pro další chyby, nedodělky a režii. Student do problematiky zjevně nepronikl. Např. na straně 14 student uvádí, že při použití C++ interop je nutné celou aplikaci překompilovat jako C++/CLI, což vůbec není pravda! Jako C++/CLI stačí zkompilovat pouze #pragma managed funkce, které volají řízený kód. Výsledkem je pak mixed-mode kód. V práci chybí teorie ohledně hostování CLR v procesu aplikace – nějak se CIL kód přece vykonávat musí.

„3. Funkčnost programového vybavení ověřte na několika jednoduchých interop (z C++ do .NET) příkladech. Proveďte zhodnocení časové režie volání".

Student provedl dva microbenchmarky, ze kterých vyšlo čisté použití C# jako nejvýkonnější. Ve skutečnosti se pouze jedná o potvrzení již vytknutých nedostatků celé práce. Konkrétně, první microbenchmark při práci s řetězci ukázal, že mj. student pro marshalling vůbec nepoužil specializované funkce. Student alespoň poukázal na to, že výsledky byly zatíženy marshallingem řetězců. Druhý benchmark pak potvrzuje, že si studentova práce vynucuje mnoho přechodů mezi řízeným

a nativním kódem. Už i student takové množství přechodů identifikoval jako problém. Tak proč tedy stále kriticky nezrevidoval svou práci?

Výsledky COM nejsou dostatečně zdůvodněny. Early-binding volání metody COMu z VC++ (což v testu dělal) je ve VC++ ekvivalentem volání metody instance potomka __interface, které má mít správně __stdcall konvenci volání na IA-32 a vždy návratový typ HRESULT. Režie volání .COM se tak musí skrývat uvnitř volané funkce, respektive COM Callable Wrapperu.

Student závěrem microbenchmarku tvrdí, že „C# nabízí nedosažitelně lepší výkon [sic]". Ve skutečnosti ale hovoří o vykonání testů bez přechodů mezi řízeným a nativním kódem, čehož by dosáhl použitím čistého C++ interop ve shodě s tím, jak se má správně používat – viz již odkázaný text „Performance Considerations for Interop (C++)".

„4. Vyzkoušejte nasazení řešení na reálný problém a proveďte zhodnocení tohoto nasazení".

Student otestoval svůj C++/CLI most na .NET knihovně DotNetZip. V příslušné kapitole uvádí, že bylo nutné celý nástroj značně upravit, aby bylo možné s knihovnou DotNetZip pracovat. Z příkladu na přiloženém CD vyplynulo, že se mu podařilo zkomprimovat jeden soubor. Dále tvrdí, že svůj nástroj otestoval ještě na dvou knihovnách, ale jmenuje pouze jednu - Math.Net. Ačkoliv na jednu stranu tvrdí, že je jeho nástroj v praxi použitelný, v poznámce zároveň uvádí, že jeho nástroj nedokáže správně vygenerovat všechny metody právě pro knihovnu Math.NET.

Student měl vyzkoušet výsledné řešení na reálném problému. Žádný reálný problém však neřešil, pouze otestoval, že dokáže zavolat metody z dané knihovny. Kdyby namísto toho řešil reálný problém s ohledem na výpočetní výkon, přišel by na to, že přidávání další, zbytečné režie ke stávajícímu C++ interop v prostředí VC++ k ničemu rozumnému nevede. A to by mu zároveň potvrdilo, že by bylo přínosnější věnovat se multiplatformnímu P/Invoke wrapperu – jak už ho k tomu ostatně vede první bod zadání bakalářské práce.

Práce s literaturou

Naprostá většina zdrojů jsou odkazy na on-line články (včetně Wikipedie), které ukazují fragmenty kódu. Celá práce vypadá, jakoby bez hlubšího přemýšlení či proniknutí do dané problematiky zcela nekriticky přebral celý postup ze zdroje [11], odkud jsou mimochodem i příklady s Yahoo. Knihy jako jsou např. C++/CLI in Action (ISBN 978-1932394818) nebo Microsoft Visual C++/CLI Step by Step (Step by Step Developer) (ISBN 978-0735675179) v seznamu literatury nejsou. Přičemž zejména první jmenovaná má na straně 148 kapitolu věnovanou přímo tématu s názvem „Accessing a managed library from native code", a to v rámci kapitoly „Introduction to mixed-mode programming".

Splnění zadání

Jelikož body zadání výslovně nepožadují vytvoření multiplatformního řešení, nezbývá mi než konstatovat, že po technické stránce student vytvořil wrapper, který dokáže alespoň pro vybrané .NET assembly zavolat jejich metody. Pouze z tohoto důvodu navrhuji hodnocení známkou **dobře** a práci doporučuji k obhajobě.

V Plzni 29. 7. 2015

Ing. Tomáš Koutný, Ph.D.