Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Informační technologie
Studijní obor: B2646 – Informační technologie

Systém automatického sestavování a testů pro simulátor proudění Flow123d

Continuous integration system for the flow simulator Flow123d

Bakalářská práce

Autor: Michal Nekvasil
Vedoucí práce: Mgr. Jan Březina, Ph.D.

V Jičíně dne 15. května 2011

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon

č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum 17.května 2011

Podpis

Poděkování

Chtěl bych poděkovat všem, kteří mi pomohli a podporovali mě při psaní této bakalářské práce. Hlavně bych chtěl poděkovat vedoucímu mé práce panu Mgr. Janu Březinovi, Ph.D. za jeho rady a časté konzultace, bez nichž bych se při tvorbě neobešel. Dále samozřejmě své rodině, bez jejichž podpory by tato práce nemohla vzniknout.

Abstrakt

Tato práce se zabývá průběžnou integrací, její teorií a základními principy proč a k čemu je tento přístup výhodou v řízení softwarových projektů. Konkrétně se zabývá spouštěním automatických překladů a sestavení simulátoru proudění Flow123d během jeho vývoje. Tato sestavení jsou dále testována a tím je kontrolována správnost dané vývojové verze. Program Flow123d slouží k výpočtům proudění tekutiny v puklinovém prostředí. Je vyvíjen ústavem NTI.

Průběžná integrace je realizovaná pomocí nástroje Bitten, který běží jako plugin systému Trac. Trac je webová aplikace k řízení vývoje softwarových projektů a odhalování chyb v nich. V práci je popsán princip, funkce i nastavení nástroje Bitten. Průběžná integrace je zprovozněna na třech rozdílných strojích. Na stroji s operačním systémem Linux, na clusterovém stroji, kde se provádějí výpočty paralelně na více procesorech, konkrétně se jedná o cluster Hydra na TUL. A také na stroji s operačním systémem Windows. V práci je uveden popis zprovoznění na všech třech strojích.

Protože je program Flow123d napsán pro operační systém Linux je k překladu pod operačním systémem Windows využit nástroj Cygwin, který umožňuje v prostředí Windows emulovat chování UNIXových systémů. Systém dále využívá skriptů napsaných pro příkazovou řádku Bash a také nástroje Make. Ve skriptech je řešena problematika spouštění více úloh, prezentace výsledků výpočtů testovacích úloh a jejich porovnání s referenčními, čímž se provádí testování daného sestavení. Výsledky všech sestavení a výpočtů se zároveň prezentují na web, pomocí již zmiňovaného programu Bitten.

V poslední části se zmiňuje automaticky generovaná dokumentace pomocí nástroje Doxygen. Tato dokumentace je zároveň ve formě HTML prezentovaná na webové stránce.

Klíčová slova: průběžná integrace, automatické testy, vývoj software, překlad programů, verzování softwaru

Abstract

Following work is about continuous integration, its theory and basic principles why to use it when developing software products and which advantages it has. More specifically it is about performing automatic builds of a flow simulator Flow123d during its development. Builds are being tested to check their correctness. The Flow123d is simulator Darcy flow in fractured porous media using mixed-hybrid formulation. It is being developed by NTI.

Building and testing is done by Bitten utility, which is a plugin to Trac. Trac is an open source, web-based project management and bug-tracking tool. In this work there are explained Bitten principles, functions and installation. Builds and testing is done on three different systems, on a Linux based system, as well as on computer cluster where computation is running in parallel on more processors. More specific this is done on cluster Hydra, which is administrated by NTI department on TUL. Builds and tests are also done on Windows based system. In this work is described how it is done.

The Flow123d is written for Linux based system, so building on Windows is done by Cygwin utility, which emulates Linux like environment on Windows. Performing builds and tests itself is done by system of scripts, written for Bash and also by using the Make utility. Scripts are solving problems about running more computations at once, presenting tests results and comparing it with referential results, which is a way how to test the build. Results are being presented on web by Bitten utility.

In the last part I mention automatically generated documentation, it's done by Doxygen. This documentation is automatically presented on website.

Keywords: continuous integration, automatic tests, software development, compiling projects, software versioning

Obsah

Seznam obrázků

Seznam symbolů, zkratek a termínů

Flow123d – program pro simulaci proudění tekutin v puklinovém prostředí. Vyvíjen v ústavu NTI na fakultě FM Technické univerzity v Liberci

FM – zkratka pro Fakultu mechatroniky, informatiky a mezioborových studií na Technické univerzitě v Liberci

TUL – zkratka pro Technickou univerzitu v Liberci

NTI – zkratka pro Ústav nových technologií a aplikované informatiky

SVN – zkratka pro program Subversion. Jedná se o systém pro správu a verzování zdrojových kódů a je založen na systému centrálního repozitáře

XML – obecný značkovací jazyk pro popis dat

HTML – značkovací jazyk, základní jazyk pro tvorbu webových stránek

1 Úvod

Tématem mé bakalářské práce je systém automatického sestavování a testů pro simulátor proudění Flow123d. Práce na tomto programu probíhá v rámci širšího týmu lidí v rámci akademického prostředí, jednalo se tedy o poměrně zajímavou zkušenost s vývojem nekomerčního softwaru.

Z názvu práce plyne, že jejím cílem je navrhnout a zprovoznit automatické sestavování programu Flow123d. Automatické sestavování, v literatuře většinou označované jako průběžná integrace, je automatizovaný systém určený k integraci softwarových projektů průběžně během vývoje. Pojmem sestavování nebo integrace se rozumí spojení více částí kódu do fungujícího celku a zkompilování zdrojových kódů do spustitelné podoby. Tento systém využívá uložení zdrojových kódů na jednom centrálním serveru označovaném jako repozitář. Podstatou je myšlenka, že pokaždé, když dojde v repozitáři ke změně, je pak celý program, případně část programu, která byla změněna, znovu sestavena. Výhodou je například možnost poměrně snadno dohledat chyby.

Pokud se nevyskytne chyba během sestavování, nemusí to nutně znamenat správnost programu. Je tedy dále potřeba toto sestavení testovat určitým systémem testů, které by ideálně měly proběhnout hned po sestavení nové verze programu. Důležité je samozřejmě navrhnout správně testy, co mají testovat a jejich vypovídací hodnotu.

Při vývoji softwaru je nutné brát zřetel, na jaké platformě bude software spuštěn. Program Flow123d je vyvíjen pro POSIXové operační systémy, jako například Linux. V rámci přenositelnosti je ovšem uživateli nabídnuta i možnost přeložit program pod operačním systémem Windows. Při sestavování je tedy využito více strojů s různými platformami. A nejedná se pouze o jiné operační systémy. Vzhledem k účelu systému, tedy výpočtům proudění tekutin v puklinovém prostředí, je příhodné také řešit sestavování a běh programu na systémech s více procesory. Konkrétně se v této práci jedná o cluster Hydra. Tyto systémy mají jisté specifické vlastnosti a je na ně potřeba brát zřetel při vývoji.

Práce je organizovaná následovně. Na začátku práce se krátce zmíním o problematice průběžného integrování. Hlavně o myšlence a účelu této metodiky. Také zároveň o nárocích na programové vybavení a návyky lidí. Pokusím se shrnout výhody a zhodnotit, proč využít průběžného sestavování. Tímto se zabývá zejména kapitola 2.

Kapitola 3 se týká samotného programu Flow123d. Protože tato práce je zejména o sestavování a testování tohoto programu, tak v této části je uvedeno, jak se program Flow123d překládá, co je k tomu potřeba a případné možnosti překladu.

Na toto navazuje část o systému skriptů, které jsem musel vytvořit, aby bylo možné provádět sestavení a integraci na různých strojích v různém prostředí. A to

pokud možno co nejjednodušeji, vlastně vytvořit jakousi robustní nástavbu na spuštění výpočtů a testů. O tomto pojednává celá kapitola 4.

V kapitole 5 se blíže zaměříme na praktické nasazení systému průběžného sestavování programu Flow123d, k tomu je využit nástroj Bitten. Zmíním se o jeho nainstalování, nastavení a samozřejmě jeho zprovoznění na jednotlivých strojích, které byly vybrány jako testovací pro vývoj programu Flow123d. A také o souvisejících věcech jako výstup výsledků a možnosti zpracování. Hlavně tedy prezentace výsledků na webu. V této části je také uvedeno praktické nasazení tohoto systému.

V kapitole 6 se již pouze krátce zmíním o automaticky generované dokumentaci, která je vytvářena pomocí aplikace Doxygen. Výstup z této aplikace je ve formě HTML automaticky prezentován také na webu.

2 Průběžná integrace

Při vývoji softwaru je k dispozici mnoho praktik, které se dají použít k řízení vývoje projektu. Jednou z velice užitečných je automatické sestavování a testování, nazývané průběžné sestavování nebo integrace. Průběžná integrace navazuje na metodologii extrémního programování, i když extrémní programování má daleko přísnější nároky, zejména na testování.

Extrémní programování je agilní metodologie vývoje softwaru, která předepisuje určité činnosti všem účastníkům vývojového procesu. Jedná se o tradiční činnosti, které jsou však dovedeny do extrému. Díky tomu by mělo být extrémní programování schopné se lépe přizpůsobit měnícím se požadavkům zákazníků a dodávat software vyšší kvality. Extrémní programování využívá několika základních principů. Zdrojový kód se neustále reviduje. Zároveň se využívá takzvaného párového programování, tedy u jednoho počítače sedí dva lidé, čímž se můžou myšlenkově doplňovat, a zároveň se tím zabraňuje takzvané profesionální slepotě, tedy to že programátor nevidí chyby ve svém kódu, i když jsou poměrně zřetelné, ale jiný člověk to vidí hned. Dalším důležitým principem je neustálé testování kódu, kdy programátoři vytvářejí pro své třídy jednotkové testy, které neustále během celého vývoje testují, zda se během vývoje nepoškodila funkcionalita, zdrojový kód testů může leckdy přesáhnout velikost vlastního výkonného kódu. Dále také programování co nejjednodušších kusů, programujeme jen co je nezbytné, tedy nejjednodušší program, který bude fungovat.

Podle [1] je důležitým principem neustálá integrace projektu. Spojováním všech částí do funkčního celku se neustále kontroluje správnost všech provedených změn, a zároveň pomáhá najít a odstranit případné chyby. Existují přístupy, kde integrace tvoří samostatnou část vývoje software. Integrace se poté provádí v samostatné fázi, může trvat i měsíce a stát velké množství financí. Proto je na místě provádět integraci

postupně, během vývoje. Investice do zprovoznění tohoto systému jsou poměrně malé, naopak výsledek většinou ušetří velké množství času i financí.

2.1 Úvod do průběžného integrování

Průběžné sestavování, v angličtině „continuous integration“, je postup jak vyvíjet software, kde členové týmu poměrně frekventovaně sestavují, neboli integrují, svoji práci. Obvykle každá osoba spouští sestavení každý den, aby ověřila správnost své části. Obvykle tedy probíhá více integrací denně. Každá integrace je poté ověřena automatickým testem, který by měl odhalit chyby v integraci tak rychle, jak jen to bude možné.

Práce na projektu by mohla probíhat teoreticky takto. Je mi přiřazen nějaký úkol, udělat například několika hodinovou práci. Nejdříve si stáhnu aktuální verzi ze systému pro správu a verzování zdrojových kódů. Verzování znamená v tomto případě uchovávat informace o všech provedených změnách v kódu nebo obecně u jakékoliv digitální informace. Systém si vlastně ukládá informace o změnách, poté je snadné se vrátit k poslední funkční verzi, zjistit co způsobuje nefunkčnost a opravit chybu. Stažená verze se nazývá pracovní kopie. Na této kopii provedu své úpravy. Jakmile jsem hotov se svojí prací, provedu sestavení na svém stroji. Vlastně sestavím pracovní kopii, což obnáší zkompilování a slinkování do spustitelného souboru a provedení všech testů. Pouze pokud vše proběhne bez chyb, mohu považovat vše za správně provedené.

Pokud tedy vše proběhne u mne na stroji v pořádku, mohu teprve potom provést nahrání svých změn, takzvaný „commit“, do centrálního repozitáře. Mohlo se stát, že během mé práce mohl někdo jiný provést nahrání svých změn, které já ve své pracovní verzi nemám. Nejdříve provedu aktualizování verze na mém stroji a opět provedu sestavení, pokud mé změny kolidují s těmi, které provedl někdo, kdo provedl nahrání na repozitář přede mnou, objeví se chyba ať již v samotném sestavení nebo v následujících testech. V tomto případě je mojí odpovědností toto opravit a opakovat do té doby než se mi podaří sestavit svoji pracovní verzi, která bude kompatibilní s verzí v repozitáři. Teprve poté mohu změny do repozitáře nahrát.

Práce tímto nicméně nekončí. Provedené změny mohu považovat za správné a funkční pouze v případě, že proběhne i sestavení na referenčním integračním stroji bez chyb. Sestavení na integračním stroji lze samozřejmě spustit ručně, ideálně je však spouštěno automaticky.

Pokud se kolize objeví mezi dvěma vývojáři, je to obvykle odchyceno ve chvíli, kdy druhý vývojář chce nahrát své změny na repozitář. Pokud ne, sestavení by mělo selhat. V každém případě je chyba rychle zjištěna. V tu chvíli je nejdůležitější chybu opravit a zajistit znovu správné sestavení. Pokud se dodrží výše zmíněné principy, nemělo by se stávat, že sestavení proběhne špatně. I přesto se to samozřejmě může stát. Cílem by poté mělo být opravit chyby jak nejrychleji je to možné.

Výsledkem je, že je zde stále stabilní kus softwaru, který správně pracuje a obsahuje minimum chyb. Což je velice výhodné, protože každý člen týmu může dál pracovat a ví, že dělá práci na stabilním kusu softwaru, který se nebude příliš lišit od další verze, kterou může někdo během práce nahrát na repozitář. Tedy i integrace různých částí by neměla být velký problém.

2.2 Praktické nasazení průběžného sestavování

Text výše je pouze shrnutí a princip, jak by průběžné sestavování mělo fungovat v každodenním životě. Zajistit aby toto všechno v pořádku fungovalo, je samozřejmě daleko složitější. V další části se tedy budu zabývat tím, jak předchozí principy prakticky provádět.

Základem nasazení průběžného integrování je mít nástroj k verzování softwaru. Nástrojů existuje velké množství, například Mercurial, Git nebo Subversion. V rámci této práce se budeme zabývat pouze nástrojem Subversion [2]. Ten je k dispozici pod licencí, která umožňuje bezplatné komerční využití a k dispozici jsou i zdrojové kódy. Navíc lze provozovat na více operačních systémech, včetně *Windows* a UNIXových systémech. Je založen na systému jednoho centrálního repozitáře.

V repozitáři by mělo být vše, co daný program k instalaci vyžaduje. Testovací skripty, konfigurační soubory, databázová schémata, instalační skripty i knihovny třetích stran. Základní myšlenkou je totiž to, že pokud přijdu k novému stroji, měl bych být schopen stáhnout si z repozitáře aktuální verzi a nainstalovat jí bez větších problémů. Největší výhodou je ale to, že pokud na kódu pracuje více lidí, je stále uložen na jednom místě, kde jsou i k dispozici údaje o všech změnách. Značně to pomáhá odstranit situace, kdy si více lidí stále dokola přepisuje kód. Existuje totiž pouze jedna referenční podoba kódu uložená na repozitáři, do které se změny nahrávají. To, zda spolu provedené změny od více lidí nekolidují, se poté dá kontrolovat právě například pomocí průběžné integrace.

Další užitečnou vlastností nástrojů ke správě verzí je možnost vytvářet více vývojových větví projektu. To umožňuje udržovat mimo hlavní vývojovou větev další větve, kde je možné testovat experimentální postupy, zkoušet jiné možnosti, atd., bez narušování hlavní větve.

Provést sestavení programu je často otázka více než jednoho kroku. Nejedná se většinou pouze o přeložení zdrojových kódů do spustitelné podoby, ale také o různé přesuny souborů, nastavování databází, registrů a dalších věcí. Protože se tato činnost v rámci průběžného integrování provádí i vícekrát denně, je vhodné vytvořit systém, který toto automatizuje a umožní spouštění ideálně jedním příkazem. Prostředí pro automatické sestavení je běžná věc. V UNIXovém světě je k dispozici systém Make.

Sestavení tradičně znamená kompilování, slinkování a další věci potřebné k tomu, aby se dal program spustit. I když se dá program spustit, neznamená to, že dělá to co má. Moderní staticky typované jazyky mají silnou typovou kontrolu, a tím dokážou hned při překladu upozornit na celou řadu chyb, ale pouze syntaktických. Sémantické chyby je těžší odhalit, jednou z cest je testování.

Existuje více způsobů, jak testovat výsledný kód. Lze testovat na úrovni jednotlivých tříd. Princip je takový, že k dané třídě je napsána další část kódu, která testuje, zda tato třída na definované vstupy poskytuje správné výstupy. Testování probíhá tak, že programátor určí, na jaké hodnoty vstupů přísluší jaké hodnoty výstupů. Při testování se tot zkontroluje. Druhou možností je testovat až přeložený a sestavený program. Pomocí testovacích úloh se zjistí, zda program vše provede správně. Tento systém testování je využit při vývoji Flow123d a je popsán v této práci.

2.2.1 Časté nahrávání změn

Základem průběžné integrace je, že se nahrávání změn do repozitáře provádí častokrát denně. Chybou naopak je, pokud vývojář nahrává velké kusy kódu najednou, například týdenní práci. Toto odporuje celé myšlence průběžného integrování. V průběžném integrování je důležitá právě komunikace. Pokud nahraji své změny do repozitáře, každý další vývojář může tyto změny hned vidět, případně jim přizpůsobit své změny. Usnadní mu to také hledání případných chyb zabraňujících integraci jeho změn, protože důvody, proč integrace nejde, se nachází jen v malé části kódu, a ne v celotýdenní práci.

Základem by tedy mělo být nahrávání změn do repozitáře co nejčastěji, klidně i vícekrát denně. Rozdělením práce do více kousků usnadňuje hledání chyb, protože existuje méně míst, kde by se chyba mohla ukrývat.

Denním nahráváním do repozitáře získává tým také testovaná sestavení programu. To by mělo znamenat, že hlavní vývojová linie zůstává stabilní a bez chyb. Jak již bylo řečeno, může se samozřejmě stát, že sestavení selže. Příčin může být hned několik, lidský faktor, odlišnost integračního stroje od stroje, na kterém vývojář provedl sestavení svých změn před nahráním do repozitáře, atd. Nutné je ovšem rychle chyby odstranit a mít opět správné sestavení.

Tedy po každém nahrání do repozitáře by mělo proběhnout sestavení na integračním stroji. A pouze pokud toto proběhne v pořádku, lze říct, že nahrání změn proběhlo v pořádku. Za toto je zodpovědný vývojář, který nahrání provedl. Tudíž po nahrání musí sledovat sestavování hlavní vývojové linie, a pokud toto selže, musí provést opravu.

Existují dva základní postupy, jak tohoto dosáhnout. Spouštět sestavení ručně, nebo použít server pro průběžnou integraci.

Spouštět sestavování ručně je jednoduché. Je to vlastně stejné jako sestavení, které vývojář provádí u sebe na stroji. Připojí se na integrační stroj, stáhne nejnovější verzi s jeho změnami a spustí sestavení. Sleduje tento proces, a pokud proběhne v pořádku, je jeho práce hotová.

Server pro průběžnou integraci naopak stále monitoruje repozitář. Pokaždé, když dojde ke změně v repozitáři, automaticky provede stažení této nové verze, provede sestavení, včetně případných testů, a poté informuje o výsledcích tohoto sestavení. Například emailem. Pokud například ve firmě probíhá sestavení každý večer, nejedná se o průběžnou integraci, smyslem průběžné integrace je sestavit program hned po nahrání změn. Ovšem v tomto případě se toto provede jen jednou za den a to už mohlo být nahráno více změn od různých vývojářů. Tedy je opět těžší případnou chybu objevit.

Důležitá je rychlá odezva. Pokud bude trvat chybné sestavení delší dobu, je možné, že na tomto základě budou vyvíjet i další vývojáři a jejich verze už proto může obsahovat chybu, která se bude distribuovat dál. Oprava by tedy měla být co nejrychlejší.

2.2.2 Testování na stroji s optimálním nastavením

Cílem testování je odhalit možné problémy, které by mohly vyvstat během instalování na koncový systém. Největší měrou do toho zasahuje prostředí, ve kterém koncová verze softwaru bude běžet. Je tedy dobré testovat sestavování na co největším množství možných platforem, abychom vyloučili možné chyby.

Z toho důvodu je snahou nastavit podmínky na testovacím systému tak, aby co nejvíce odpovídal podmínkám na koncovém systému uživatele. Například použít stejnou databázi ve stejné verzi, stejnou verzi operačního systému, použít stejné knihovny jako na koncovém, i když je systém nevyužívá, použít stejné IP adresy a porty, spouštět na stejném hardwaru, a tak dále.

Samozřejmě pokud píšeme desktopové aplikace, je prakticky nemožné testovat na všech možných verzích, se všemi knihovnami, které různí lidé používají. Nebo je dané prostředí příliš drahé napodobit. I přesto by mělo být cílem napodobit testovací podmínky jak je to jen možné.

V poslední době se čím dál více rozmáhá virtualizace jako prostředek k simulování více prostředí. Virtualizované prostředí mohou být uložena s odpovídajícím nastavením, a na jednom fyzickém stroji lze mít větší množství prostředí.

Moderní nástroje průběžné integrace umožňují různé možnosti prezentace výsledků. Poměrně častá je prezentace výsledků pomocí webových aplikací. Přístup na web má dnes snad každý vývojář. Pomocí například červených a zelených indikátorů se

rozliší chybné, respektive správné sestavení. Dále je například možné také zobrazit, jak dlouho v řadě se v sestavení objevují chyby.

I v případě, že se sestavování spouští ručně, je důležité prezentovat výsledky. Například jednoduchým ukládáním textového výstupu do souboru a jeho distribucí dále. Toto je důležité proto, že pomocí těchto výsledků se vyhledávají chyby. Z informací o průběhu sestavení lze určit, kde se chyba může nacházet.

Servery pro průběžné integrování mohou samozřejmě zobrazovat mnohem víc informací, než jen zda bylo sestavení úspěšné či ne. Například zobrazovat, jaká verze se překládá, kdo a jaké změny nahrál a uchovávat historii všech změn a výsledků sestavení. Každý tedy může vidět, jak probíhá vývoj, co se právě provádí a jaké změny se udály. Vedoucí projektu toho také mohou využít ke sledování toho, co kdo zrovna dělá a mít povědomí o průběhu vývoje.

Výhodou používání prezentace přes web je také to, že i ti, kteří se nachází i na druhé straně Země, mohou stále sledovat, co se děje. Obecně je samozřejmě lepší, pokud lidé aktivně pracují na jednom místě, ale často jsou v týmu i lidé, kteří jsou vzdálení, ale stejně potřebují mít možnost sledovat vývoj projektu.

2.2.3 Shrnutí výhod průběžné integrace

Jednou z největších výhod je snížení rizika chyb a dlouhé integrace. Jak bylo řečeno na začátku, dříve byla integrace samostatnou částí vývoje, a protože se integrace neprováděla průběžně, nikdo nevěděl, jak dlouho bude trvat odstranit všechny problémy, které mohou vyvstat. Dokonce ani jak daleko se v procesu integrace zašlo.

Průběžná integrace kompletně a elegantně odstraňuje tento problém. Už se dále nejedná o dlouhý proces integrace, vše probíhá postupně a postupně se také odstraňují vzniklé problémy. Tím je odstraněno slepé místo a vždy víme, kde se nacházíme, a co funguje a co ne.

Chyby v softwaru jsou nepříjemné a dokážou značně znepříjemnit vývoj a zničit časový plán. Navíc pokud se chyby objeví v již hotovém programu, často to může odradit zákazníky a poškodit reputaci firmy.

Průběžná integrace neodstraňuje tyto chyby, ale značně ulehčuje jejich nalezení a odstranění, zvláště pokud je správně nastaven systém testování každého sestavení. Pokud je chyba rychle objevena, je mnohem snazší jí odstranit. V průběžném integrování je chyba rychle objevena již z principu, protože se postupuje po malých kouskách a lze snadno dohledat, která verze ještě danou chybu nevykazovala. Navíc, protože chyba vznikla v nedávno vytvořeném kódu, je ještě tento kód v podvědomí vývojářů, což opět snižuje náročnost opravy. Dalším usnadněním je snadné porovnání předchozí a nynější verze, k čemuž například v programu pro správu verzí Subversion existuje jednoduchý nástroj.

Celkový výsledek tedy je, že by se měla poměrně značně zkrátit doba vývoje softwaru, stejně jako by se měl zmenšit počet neobjevených chyb. Toto je samozřejmě z velké části závislé na úrovni vytvořených testů. Ovšem v dnešní době, kdy je k dispozici řada vyspělých nástrojů pro jejich tvorbu, již není takový problém vytvořit bez větších problémů testy, které budou podávat správné výsledky.

3 Simulátor proudění Flow123d

V této části práce bych se krátce zmínil o simulátoru proudění Flow123d. Cílem práce bylo automatizovat sestavování tohoto programu. V úvodu bych se zmínil, co je a k čemu je vlastně určen program Flow123d, jaké jsou požadavky při instalaci a také popis, jak tento program nainstalovat.

Simulátor proudění Flow123d je software pro simulaci proudění vody, rozpuštěných látek v heterogenním porézním a puklinovém prostředí. Zejména je vhodný pro simulaci podzemních procesů v žulovém masivu. Program je schopen popsat explicitně procesy v 3D prostředí, 2D prasklinách a 1D kanálech a převody mezi těmito prostory. Předpokládá nasycení média popsaného Darcyho zákonem. Pro diskretizaci využívá metody konečných prvků. Program je realizován v jazyce C/C++ a pro výpočty v lineární algebře využívá knihovnu PETSC. Tedy před samotnou instalací programu Flow123d je nutné nainstalovat knihovnu PETSC, bez tohoto nelze program Flow123d nainstalovat. Instalace knihovny PETSC probíhá přes UNIXovou příkazovou řádku Shell, takže je nutné mít ji k dispozici, ať už se v rámci UNIXových systému jedná o standardní součást nebo zda je emulovaná pomocí speciálních programů, například *Cygwin*, o němž se dále také zmíním. Překlad probíhá pomocí nástroje Make, je nutné mít tento nástroj nainstalován.

Nástroj Make je určen k automatizaci překladu zdrojových kódů do binární podoby. Programátor definuje závislosti mezi soubory. Tyto definice se zapisují do souboru *makefile* v každém adresáři. Podle tohoto souboru se poté určí postup překladu. Jsou zde také určená pravidla, která lze volat. V každém pravidle může být určeno, co se má vykonat.

3.1 Instalace knihovny PETSC

PETSC [3] je knihovna určená k paralelnímu řešení lineárních a nelineárních soustav rovnic. PETSC je volně k dispozici ke stažení na internetu. Dostupná je pod licencí GPL ve verzi 2.

Nyní již k samotné instalaci. Knihovna PETSC je dostupná pro více operačních systémů včetně *Windows*, ale vzhledem k tomu, že program Flow123d je napsán pro

operační systém *Linux*, předpokládejme instalaci pomocí příkazové řádky Shell v tomto operačním systému, například Bash. Je také potřeba mít nainstalovaný interpret jazyka Python, protože knihovna využívá pro překlad konfigurační skript napsaný v tomto jazyce.

Předpokládejme, že máme stažený balíček PETSC, umístěný pro jednoduchost například v adresáři */home/petsc/*. K tomuto adresáři je potřeba nastavit cestu do proměnné PETSC_DIR a rovnou ji exportovat, aby byla dostupná pro spuštěné skripty. Lze to učinit ručně, ale nejjednodušší způsob je v adresáři PETSC použít příkaz

```
/home/petsc > export PETSC_DIR=$PWD
```

Protože je možné nakonfigurovat knihovnu PETSC mnoha způsoby, je dále potřeba nastavit ještě proměnnou PETSC_ARCH. Tím je umožněno mít více různých kompilací knihovny na jednom místě a nastavováním PETSC_ARCH využívat vybranou verzi. Nastavíme tedy proměnou PETSC_ARCH například takto

```
/home/petsc > export PETSC_ARCH=<název>
```

PETSC umožňuje využívat řadu externích balíčků, které jsou využity k výpočtům. Které balíčky budou staženy a používány se určí pomocí nastavení konfigurace. K tomu je určen výše zmíněný konfigurační skript napsaný v jazyce Python, jedná se o skript umístěný v adresáři *config/configure.py*. Spustíme tento skript s požadovanými parametry. Program Flow123d využívá externí balíčky MPICH, Parmetis a BLAS LAPACK. Spuštění konfigurace by tedy mohlo vypadat například takto

```
/home/petsc > ./config/configure.py --with-debugging=1 --with-64-bit-
                        pointers=0 --with-shared
          --CFLAGS_O=-O3 --FFLAGS_O=-O3 --download-mpich=yes
          --download-parmetis=yes --download-f-blas-lapack=1
                              --with-x=0
```

Tímto se automaticky provede stažení a nainstalování těchto balíčků. Toto je pouze jedna z možností nastavení. Každý systém se může chovat jinak, vzhledem ke konfiguraci a nastavení. Pokud se během konfigurace vyskytne chyba, je potřeba obrátit se na dokumentaci knihovny PETSC. Nicméně tyto balíčky jsou nutné.

Po proběhnutí tohoto skriptu už zbývá přeložit knihovnu pomocí nástroje Make. Stačí tedy spustit překlad příkazem

```
/home/petsc > make all
```

Knihovna je po správném proběhnutí nainstalovaná. Pro kontrolu je možné ještě spustit test kompilace pomocí příkazu

```
/home/petsc > make test
```

Pokud test proběhne v pořádku, měla by být knihovna nainstalovaná správně.

3.2 Sestavení programu Flow123d

Jak již bylo řečeno v úvodu, předpokladem k průběžné integraci je potřeba mít možnost spustit překlad jedním příkazem. Flow123d jakožto aplikace napsaná pro operační systém *Linux* toto umožňuje. Využívá k tomu nástroj Make. Ovšem nejdříve je potřeba nastavit konfigurační soubor.

Předpokládejme opět stažení aktuální verze Flow123d z repozitáře. Například do adresáře */home/flow123d*. Konfigurační soubor *makefile.in* nastavuje řadu proměnných nutných ke správnému překladu. Pokud je správně nainstalovaná knihovna PETSC, je nutné prakticky pouze nastavit cestu k této knihovně. V adresáři se nachází soubor *makefile.in.template,* toto je šablona, jak vytvořit *makefile.in*. *Makefile.in.template* obsahuje následující řádky, které jsou potřeba upravit

```
PETSC_DIR=
```

Zde doplníme cestu k adresáři PETSc.

```
PETSC_ARCH=
```

A sem zadáme jméno kompilace knihovny PETSc, kterou chceme použít. Jedná se o název, který jsme si zvolili při kompilaci.

Důležitou proměnou, kterou je možné nastavit je proměnná CFLAGS. Tato proměnná určuje, zda bude překlad optimalizován či nikoliv.

Dále je zde proměnná USE_NODEBUG. Tato proměnná naopak určuje, zda se provede překlad s ladícím výstupem. Tato možnost se používá pokud vytváříme produkční verzi.

Na začátku bylo řečeno, že průběžná integrace by měla probíhat na různých strojích. Aby se daly tyto stroje odlišit, v *makefile.in* je ještě jedna proměnná. Jedná se o proměnnou MACHINE. Proměnná MACHINE označuje jméno stroje, na kterém se překlad spouští a není nutná pro správný překlad. Blíže je princip popsán v kapitole 4.2.

Pokud je tedy vše v *makefile.in* nastaveno tak, jak má být, zbývá jen pomocí nástroje Make spustit překlad příkazem

```
/home/flow123d > make all
```

Pokud vše proběhne bez chyby, je program Flow123d přeložen. V adresáři *bin* se vytvořil spustitelný soubor *flow123d.*

4 Systém skriptů pro spouštění

Cílem průběžná integrace je neustálé sestavování a testování vyvíjeného softwaru, v ideálním případě ihned po každém nahrání změn do repozitáře. To znamená spouštění sestavení i mnohokrát denně. Zároveň, pokud nepočítáme s manuálním spouštěním, je více než vhodné umožnit spouštění těchto úloh ideálně jedním příkazem.

Tedy v následující části bude popsáno, jak tyto skripty spouštět a jaký je jejich princip a realizace. Nejdříve se zmíním o pomocných pravidlech pro překlad programu Flow123d.V předchozí části jsem sice uvedl, jak program Flow123d přeložit, ale tento systém nebyl dostatečně robustní, aby umožnil aplikovat metody průběžného integrování a bylo ho tedy potřeba vylepšit. Dále potom detailně popíšu skripty, které jsou určené k testování výsledného sestavení.

Nyní už přímo k praktickému použití. Průběžná integrace se skládá z několika fází. Nejdříve je potřeba stáhnout aktuální verzi z repozitáře, konkrétně v našem případě se v rámci vývoje programu Flow123d jedná o verzovací systém Subversion. Tento program umožňuje stažení aktuální verze jedním příkazem. Následující kroky již vyžadují vytvoření skriptů, které umožní je spouštět jednoduše, jedním příkazem.

4.1 Pomocná pravidla při překladu

Systém pro překládání programu byl již hotov, ale nebyl dostatečně vyhovující. Nevýhodou byla nemožnost různých nastavení pro různé systémy. Bylo tedy potřeba vytvořit jakýsi systém, který by pomocí určitých pravidel vytvářel jednotné prostředí pro spouštění výpočtů na různých systémech. Ideálně samozřejmě automaticky hned při překladu.

Úvodem ještě uvedu, jak se standardně spouští paralelní výpočty. Program Flow123d využívá knihovnu PETSC, a ta obsahuje knihovnu MPI. Toto je knihovna, která definuje specifikaci pro podporu paralelního řešení výpočetních problémů v počítačových clusterech. Jedná se o systém zasílání zpráv mezi jednotlivými uzly. V systému se může vyskytovat více implementací knihovny MPI. Aby bylo zajištěno, že k výpočtu bude použita správná knihovna, ta která je součástí PETSC, je při překladu Flow123d vytvořen odkaz na tuto knihovnu. Stačí jen zavolat tento odkaz na spustitelný soubor se správnými parametry a výpočet proběhne paralelně. Syntaxe je následující

```
./mpiexec -np <NPROCS> <EXECUTABLE> <PARAMETERS OF EXECUBALE>
```

Skript `mpiexec` je automaticky vytvořený odkaz na MPI v knihovně PETSC, který se vytvoří při překladu program Flow123d. Parametr `-np` značí počet procesorů, za ním by mělo následovat `<NPROCS>`, hodnotou je číslo, tedy kolik procesorů chceme pro výpočet použít. Dalším parametrem je `<EXECUTABLE>`, jeho hodnotou je cesta

k volanému spustitelnému souboru, který chceme spustit paralelně. Dále už pouze `<PARAMETERS OF EXECUTABLE>`, hodnotami jsou parametry příslušející k `<EXECUTABLE>`.

Vzhledem k tomu, že lze tuto knihovnu volat i s parametrem `-np 1`, tedy že bude výpočet probíhat jen na jednom procesoru, je výpočet zpravidla vždy volán přes knihovnu MPI, i když je počet procesorů roven jedné.

Na stejném principu funguje také vytváření odkazu *current_flow*. Odkaz *current_flow* v sobě uchovává informaci, jak spouštět výpočet na aktuálním stroji. Je také automaticky vytvářen při překladu. Tím je zajištěna možnost odlišit jednotlivé stroje a zajistit jednotné prostředí pro spouštění

Aby se odlišili jednotlivé stroje, je zavedena již zmiňovaná proměnná MACHINE. Ta určuje, na jakém stroji se nyní daný překlad provádí a podle toho se určí, jak se budou úlohy spouštět. Vše je opět realizováno ještě v systému Make při překladu. Při spuštění překladu je automaticky vytvořen odkaz na skript, kterým bude použit ke spouštění výpočtů. Soubor *current_flow* je umístěn v adresáři *bin*. Existují tři možnosti, které se mohou vyskytnout.

V prvním případě je možné, že proměnná MACHINE není nastavena. Poté lze tedy předpokládat, že by měl být použit základní způsob spouštění. Základním způsobem je spouštění přímo přes volání výše zmíněného odkazu `mpiexec` na knihovnu MPI. Toto je popsáno v kapitole 4.2.2.

Druhou možností je, že proměnná MACHINE je nastavena, poté se do vytvářeného odkazu umístí cesta k souboru, který je pojmenován MACHINE_flow.sh a je umístěn v adresáři *bin*. MACHINE nahrazeno jménem stroje, které je nastaveno v souboru *makefile.in* při překladu. Ovšem pouze za předpokladu, že tento soubor existuje. Tímto je umožněno vytvářet vlastní možnosti spouštění úloh. Do daného skriptu se dá napsat prakticky cokoliv, co se má během spouštění vykonat, ať už se jedná o různé výpisy do konzole nebo je možné zde například určit, s jakými parametry bude spustitelný soubor volán. Možností je velice mnoho. V našem případě je tohoto konkrétně využito při spouštění přes program pro dávkovací zpracování úloh. Toto spouštění má své specifické nároky. Popsáno v kapitole 5.

A nakonec třetí možnost. A to případ, kdy je sice nastavena proměnná MACHINE, ale daný skript není nalezen. Poté je do odkazu přiřazena cesta k základnímu skriptu na spouštění.

4.2 Skripty pro spouštění testů

V rámci teoretického úvodu k metodice průběžného integrování se zmiňovala nutnost testovat program po každém úspěšném sestavení. Programu Flow123d je testován pomocí referenčních úloh. Testování probíhá tak, že na aktuálně přeloženém

programu se provede výpočet úlohy a výsledky se otestují porovnáním s referenčními, které jsou k dispozici ke každému testu.

Testovací úlohy se nachází v adresáři *tests*, kde má každá testovací úloha vlastní adresář. Spouštění testů obstarává skript *run_test.sh*, který je také umístěn v adresáři *tests*. Tento skript by měl být volán z adresáře dané testovací úlohy a provést kompletní otestování správnosti výsledků získaných výpočtem testovací úlohy na přeloženém programu Flow123d. Pro výpočet úloh využívá skript *run_flow.sh*. Tento skript spouští výpočet dané testovací úlohy se zadanými parametry, tedy hlavně jméno konfiguračního souboru, použitého pro výpočet, a také počet procesorů, na kterých se má výpočet provádět. Po provedení výpočtu provede skript *run_test.sh* porovnání získaných výsledků. Pro porovnání využívá skript *run_check.sh*, který provádí porovnání dvou zadaných adresářů, přesněji numericky porovná jednotlivé soubory v celé adresářové větvi zadaného adresáře. Výsledky porovnání, stejně jako výsledky výpočtů poté umístí do adresáře *Results* v daném testu, pro pozdější zpracování, a pokud se v nějakém místě objeví chyba, skončí s chybovým hlášením. Samotné spouštění testů probíhá přes systém Make.

4.2.1 Skript run_test.sh

Skript run_test.sh je vstupním skriptem pro volání jednotlivých testů. Nepředpokládá se volání tohoto skriptu jinak, než přes systém Make a proto se parametry zadávají pozičně, a ne pomocí identifikátoru. Základní syntaxe volání tohoto skriptu je následující

```
./run_test.sh  <INI_FILE>  <NPROCS>[<FLOW_PARAMS>]
```

Skript předpokládá, že bude volán z adresáře daného testu. Parametrem `INI_FILE` se předává seznam konfiguračních souborů, se kterými má být daný test spuštěn. Jednotlivé položky musí být relativní nebo absolutní cestou k existujícímu konfiguračnímu souboru. Druhý parametr `NPROCS` udává počet procesorů, použitých k výpočtu úlohy. Obdobně jako u předchozího parametru, i zde je možné zadat více položek formou seznamu. Volitelně je možné uvést další parametr `FLOW_PARAMS`. Tento parametr může být libovolně dlouhý a bude předán přímo spustitelnému souboru Flow123d.

Skript poté postupně v cyklech spouští pro každou zadanou položku seznamu `INI_FILE` a `NPROC` výpočet testu se zadanými parametry. Ke spuštění výpočtu se volá skript *run_flow.sh* popsaný dále. Následuje smyčka, která čeká, na dokončení výpočtů, a zároveň kontroluje, zda se vůbec začala úloha provádět. Jedná se o nejzajímavější část tohoto skriptu. Úloha může být zpracovávána přes dávkový systém. Zpracování přes dávkový systém znamená, že je úloha zařazena do fronty, ve které čeká na spuštění do té doby, než je možné ji zpracovat, například kvůli zatížení systému. Bylo tedy nutné vybudovat mechanismus, který by umožňoval kontrolovat, zda byl již

výpočet ukončen. Nejenom kvůli tomu zda již lze vypočtená data zkontrolovat porovnáním s referenčními, ale také proto, že existuje možnost, že by se v daném adresáři mohlo spustit výpočtů více zároveň, což by nejspíše vedlo k chybným výsledkům. V současné verzi k tomuto nemá program Flow123d zabudované žádné vnitřní mechanismy. Do budoucna se s implementací této ochrany počítá, ale protože ještě není, použili jsme jinou alternativu. A to zamykání adresáře, kde probíhá výpočet, pomocí jednoduchého vytvoření prázdného souboru *lock*. Tento soubor se vytvoří vždy před spuštěním výpočtu v programu Flow123d, a smaže se až ve chvíli, kdy je výpočet dokončen.

Standardní výstup je přesměrováván do souboru *out*. Tak je možné sledovat, zda byla úloha vůbec spuštěna. Pokud tento soubor existuje, úloha byla spuštěna. V kombinaci s předchozím zamykáním adresáře, je tedy možné vytvořit poměrně jednoduchý a funkční mechanismus kontroly průběhu. Ve skriptu je umístěna čekací smyčka, která kontroluje, zda je adresář uzamknut, tedy zda je zde soubor *lock*. Pokud je adresář uzamknut, ale neexistuje zde vytvořený soubor *out* se standardním výstupem, potom skript ve smyčce čeká, a každých třicet vteřin kontroluje, zda stále soubor *out* neexistuje. Takto čeká a kontroluje maximálně pět minut. Pokud ani potom není soubor *out* vytvořen, předpokládáme, že se někde objevila chyba při spouštění výpočtu a skript ukončíme s chybovým hlášením.

V opačném případě soubor *out* byl vytvořen, skript pokračuje dál. Nyní již víme, že výpočet byl nejspíš spuštěn, ale je potřeba čekat na jeho ukončení. Jak již bylo řečeno výše, po ukončení se smaže soubor *lock*. V další smyčce tedy čekáme na smazání tohoto souboru. Opět každých třicet vteřin kontrolujeme, zda byl soubor smazán a pokud ano, skript může pokračovat dál. Takto čekáme maximálně dvacet minut, předpokladem je, že výpočet nebude trvat déle než těchto dvacet minut. Pokud ale do té doby není soubor *lock* smazán, opět předpokládáme, že se někde stala chyba a skript ukončíme s chybovým hlášením.

Pokud ale vše proběhne bez problémů, skript pokračuje dál a získané výsledky umístí do adresáře *Results*/<INI_NAME>.<NPROCS>, kde INI_NAME je jméno konfiguračního souboru získaného z INI_FILE a NPROCS odpovídá počtu procesorů, na kterých byl daný výpočet spuštěn. Zde jsou výsledky k dispozici pro další zpracování hned v dalším kroku, ale i později. Skript ovšem předpokládá, že je v konfiguračním souboru nastaveno, že výstup se ukládá do adresáře *output*.

Získané hodnoty porovnáme s hodnotami referenčními pomocí skriptu *run_check.sh*, který je popsán v části 4.2.3

4.2.2 Skript run_flow.sh

Spouštění úloh probíhá pomocí skriptu *run_flow.sh*. Je navrhnut dostatečně robustně, aby se pomocí něho dal spouštět výpočet i samostatně. Což byl také jeden z důvodů, proč není přímo součástí *run_test.sh*. Skript také využívá identifikace parametrů, nejedná se tedy už o poziční parametry, jako v případě skriptu *run_test.sh*. Základní syntaxe vypadá takto

```
./run_flow.sh –s <INI_FILE> [–np <NPROCS>][–q <TIME>][–m <MACHINE>]
```

Parametr `–s` je povinný. Udává jméno konfiguračního ini souboru úlohy, kterou chceme spočítat. Ať již pouze jako název, poté se musí nacházet ve stejném adresáři, nebo jako relativní nebo absolutní cesta k tomuto souboru. Další parametry jsou nepovinné. Parametr `–np` udává počet procesorů použitých k výpočtu Pokud není zadán, je použit jeden procesor. Parametr `–q` umožňuje zadat maximální čekací doba úlohy, která je zařazena do fronty pro výpočet, pokud se úloha zpracovává pomocí fronty úloh. Toto je specifický problém výpočtů na rozsáhlejších strojí jako například clusterech. Posledním je parametr `–m`. Zadáním jeho hodnoty `MACHINE` lze určit, který skript má být použit pro výpočet úlohy. Pokud není zadán, je automaticky použit základní skript, který je uveden v souboru *current_flow*, který je umístěn v adresáři *bin*. Princip tohoto byl popsán již dříve. Důležité ale je, že lze tímto parametrem obejít základní nastavení pro výpočty, které bylo nastaveno během překladu a použít jiný skript. Skript předpokládá pojmenování skriptu MACHINE_flow.sh a jeho umístění v adresáři *bin*. Pokud zadaný skript neexistuje, použije se opět základní a hláškou vypsanou na standardní výstup je uživatel upozorněn, že se používá standardní skript pro výpočty.

Skript *current_flow* standardně obsahuje odkaz na skript *generic_flow.sh*. Vysvětlíme na něm princip spouštění úloh v *run_flow.sh*. Skript *generic_flow.sh* vypadá takto

```
#!/bin/bash
touch lock
"$MPI_RUN" -np "$NPROC" "$EXECUTABLE" -s "$INI" $FLOW_PARAMS
2>err 1>out
rm lock
```

První řádek určuje, že k provedení následujících řádků souboru bude použit program Bash, jedná se o jeden z typů příkazové řádky Shell. Na druhém řádku se vytváří soubor *lock*, tento soubor slouží k zamykání adresáře, ve kterém probíhá výpočet, důvody a principy tohoto systému jsou popsány v další části, konkrétně v části o skriptu *run_test.sh*. Na dalším řádku už jsou použity proměnné, které jsou exportované ze skriptu *run_flow.sh*. Exportovanými proměnnými jsou `MPI_RUN`, tato proměnná v sobě obsahuje právě cestu ke knihovně MPI, přes kterou jsou úlohy spouštěny, prakticky se tedy jedná o cestu k odkazu `mpiexec`, který byl popsán v kapitole 4.1. Dále `NPROC` značící počet procesorů použitých k výpočtu, `EXECUTABLE` což je spustitelný soubor,

který chceme spustit. Exportovaná proměnná `INI` představuje jméno konfiguračního souboru a `FLOW_PARAMS`, což jsou volitelné parametry výpočtu. Pomocí těchto proměnných, které jsou získány z *run_flow.sh* je proveden výpočet.

Předpokládá se sestavování jak na strojích s operačním systémem *Linux*, tak i na strojích se systémem *Windows*. Systém *Windows* ukládá spustitelné soubory se specifickými příponami `.exe`. Systém *Linux* ukládá spustitelné soubory po překladu bez přípony. Podle jména souboru tedy určíme, jaký spustitelný soubor `EXECUTABLE` budeme volat k provedení výpočtů. Pokud neexistuje ani jedna varianta je skript ukončen s chybovým hlášením.

Výpočet tedy probíhá v aktuálním adresáři a zde se také uloží výsledky. Jak a kam se uloží přesně, je záležitostí programu Flow123d a také záleží na nastavení v konfiguračním souboru dané úlohy.

4.2.3 Skript run_check.sh

Tento skript je určen k porovnání vypočítaných výsledků. Je tedy volán v rámci skriptu *run_test.sh* pro porovnání vypočtených výsledků. Ale funkce není omezena jen na striktní použití v rámci *run_test.sh*. Stejně jako předchozí skript *run_flow.sh* může být použit samostatně. Jeho funkcí je porovnat obsah dvou zadaných adresářů. Syntaxe volání skriptu je následující

```
./run_check.sh <DIR_1> <DIR_2>
```

Vstupem do skriptu jsou cesty ke dvěma adresářům DIR_1 a DIR_2, ať již relativně nebo absolutně. Skript zjistí obsah prvně zadaného adresáře a postupně prochází jednotlivé nalezené položky. Pokud zjistí, že se jedná o adresář, rekurzivně zavolá sám sebe a prohledá tento adresář a porovná jeho obsah. Pokud ale objeví soubor, provede jeho porovnání se stejně pojmenovaným souborem, který se nachází v druhém adresáři. Pokud tento soubor v druhém adresáři není, je zobrazena chybová hláška s informací, který soubor chybí. Skript tímto neskončí kontrolu adresářů. Pokračuje dál, pouze si pamatuje, že se chyba vyskytla a po porovnání zbytku souborů skončí s chybovým hlášením.

Porovnání souborů je realizováno pomocí skriptu *ndiff.pl*. Jedná se o skript napsaný v programovacím jazyce Perl. Byl napsán panem Janem Březinou. Skript slouží pro numerické porovnání dvou souborů. Tedy postupně prochází soubor a data numericky porovnává. Důležitou vlastností je, že je možné zadat jak absolutní přípustnou odchylku, tak také relativní odchylku, tato se vyhodnocuje v závislosti na řádu čísla. Na konci je uvedeno, kolik odchylek větších než přípustné hodnoty se v daných souborech vyskytovalo a také jaká byla objevena maximální absolutní i relativní odchylka. Zároveň pokud je objevena větší odchylka, než je nastavená přípustná hodnota, je na konci ukončen skript s chybovým hlášením.

4.2.4 Volání skriptů pomocí nástroje Make

Skript run_test.sh bylo nutné jednotně spouštět. K tomu bylo využito možností nástroje Make. Tento nástroj je původně určen k automatizaci překladu zdrojových kódů do binárních souborů. Ovšem definuje ve svých souborech *makefile* pravidla, ve kterých lze i volat příkazy z příkazové řádky Shell, lze tedy i volat skripty, přesunovat se v adresářích, ale hlavně také definovat vlastní proměnné.

V adresáři *tests* se nachází složky se všemi dostupnými testy. V tomto adresáři je v souboru *makefile* definováno jak spouštět testy. Následující pravidlo spustí test, který se jmenuje `01_steady_flow_123d`

```
testbase:
	make -C 01_steady_flow_123d test
```

Spuštění probíhá tak, že se zavolá pravidlo `test`, které je definované v souboru *makefile*, který je umístěn v adresáři tohoto testu. V adresáři *tests/01_steady_flow_123d* je umístěn *makefile*, obsahující následující řádky

```
INI_FILES="./flow.ini"
NPROC="1 2"
FLOW_PARAMS=" -ksp_atol 1.0e-10 -ksp_rtol 1.0e-10 -
ksp_monitor -i    input"

test:
	bash ../run_test.sh ${INI_FILES} ${NPROC}
	${FLOW_PARAMS}
```

Proměnnou `INI_FILES` si definujeme konfigurační soubory, se kterými chceme daný test spouštět. Pomocí proměnné `NPROC` zadáme počet procesorů použitých k výpočtu. Na konec je možné ještě volitelně zadat parametry, které budou předány přímo programu Flow123d pomocí proměnné `FLOW_PARAMS`. Pravidlo `test` po zavolání provede spuštění skriptu *run_test.sh* v aktuálním adresáři se zadanými hodnotami.

Je zde také pravidlo pro spouštění testu zadáním prvních písmen z názvu. Stačí zadat počáteční písmena s koncovkou .tst. Pravidlo vypadá následovně

```
%.tst:
	BASE=$*;\
	dir="`pwd`/`echo $${BASE}*`";\
	echo $${dir};\
	if test -d "$${dir}";\
	then make -C "$${dir}" test;\
	else echo "missing test directory $${dir}";\
	exit 1;\
	fi
```

Podle zadaných znaků se v aktuálním adresáři pokoušíme najít odpovídající složku, jejíž název začíná stejnými písmeny. To je provedeno pomocí rozšíření znaků

pomocí hvězdičky, která má význam jakéhokoliv počtu jakýchkoliv znaků. Pomocí podmínky *if* testujeme, zda je nalezena shoda s existujícím adresářem. Pokud ano, opět zavoláme pravidlo *test*, které je v nalezeném adresáři definováno v souboru *makefile*.

Spouštět testy je možné i z hlavního adresáře. I v *makefile* v tomto adresáři existují pravidla, která se odkazují na pravidla v *makefile* adresáře *tests*.

5 Nástroje Bitten a Trac

Při vývoji simulátoru proudění Flow123d je využito nástrojů Subversion, Trac a Bitten. Jedná se o větší projekt, na kterém spolupracuje více lidí. Navíc existuje více paralelních větví. Proto se využívá možností verzovacího nástroje Subversion.

Program Trac je nástroj pro řízení vývoje software. Jeho základní vlastností je, že umí spolupracovat s nástrojem Subversion a přehledně prezentovat obsah repozitáře, včetně informací o provedených změnách u jednotlivých souborů. Nástroj Bitten je pluginem do nástroje Trac. Jeho funkcí je monitorovat repozitář, který je řízen pomocí nástroje Subversion a je zobrazován pomocí nástroje Trac. Tento nástroj poté spouští prováděni překladu a testů. Zároveň prezentuje získané výsledky na webovém rozhraní.

Celý systém je poté realizován na třech různých strojích, na stroji s *Linux*, na systému umožňujícím paralelní výpočty na více procesorech a také na stroji s *Windows*.

5.1 Trac

Trac je open source nástroj pro řízení vývoje projektů a také nástroj pro pomoc při hledání chyb. Jedná se o webovou aplikaci napsanou v programovacím jazyce Python. Jednou z vlastností je, že poskytuje přehledné grafické webové rozhraní pro různé verzovací systémy, v našem případě hlavně tedy pro nástroj *Subversion*, ale i pro další. Obsahuje v sobě přehledně graficky zobrazenou strukturu repozitáře a umožňuje její interaktivní procházení. Zobrazuje kdo jaké změny provedl. Umožňuje vytvářet takzvané tikety, což jsou vlastně sepsané úkoly, co je potřeba udělat, případně jakou chybu opravit. Tyto tikety je možné přesně přiřadit jednotlivým lidem, čímž je možné rozdělovat práci. Také je zde možné sepsat plán vývoje a cíle pro další verze. Součástí je i wiki pro prezentování dokumentace a informací o projektu.

Funkce je možné rozšířit pomocí přídavných pluginů. V našem případě se jedná zejména o plugin Bitten, který slouží k průběžné integraci.

Nástroj Trac je nainstalován a běží na stroji Dev.

5.2 Bitten

Bitten [4] je plugin do nástroje Trac. Jeho účelem je umožnit spouštět překlad a testy. Skládá se ze dvou částí, z řídící části Bitten master a z klientské části Bitten slave. Řídící část úzce spolupracuje s nástrojem Trac. Připojují se na něj klienti, kterým posílá, jaké sestavení a testy mají provést.

Bitten se instaluje na server, kde se již nachází a běží nástroj Trac. Na tomto stroji musí být nainstalován Bitten master, tedy řídící část programu Bitten. Ten úzce spolupracuje s nástrojem Trac a připojují se na něj klienti, na kterých běží Bitten slave. Bitten slave je klientská část programu Bitten, může být nainstalována na více strojích a úkolem je se po spuštění připojit na Bitten master, identifikovat se a pokud se zde nachází nějaké sestavení k testu pro daný stroj, je klientovi ve formě XML skriptu zasláno, klient poté začne u sebe provádět jednotlivé kroky XML skriptu a o jejich výsledku zpětně informuje Bitten master, který pomocí nástroje Trac prezentuje tyto výsledky na webovém rozhraní.

Zapisování XML skriptů a nastavení, na jakém stroji se má daný skript spouštět, probíhá přes webové rozraní v nástroji Trac. Zde je umožněno po přihlášení uživateli s právy na změnu Bitten skriptů tyto skripty vytvářet a měnit. To zároveň umožňuje ihned validovat syntaktickou správnost tohoto skriptu, případně ho odmítnout jako nevalidní.

5.2.1 Instalace

Pokud chceme nástroj Bitten využívat, musí být na stroji, kde je umístěn program Trac, nainstalován Bitten master, tedy již zmiňovaná řídící část programu Bitten. To umožní vytvářet XML skripty a posílat je klientům, kteří se připojí a zažádají o ně.

Vzhledem k tomu, že Bitten je plugin nástroje Trac, je nutné zvolit i verzi programu Bitten podle nástroje Trac. Pokud je využívána verze Trac 0.11, tak je zaručena kompatibilita s verzí Bitten 0.6 ve všech revizích. Naopak pokud je verze Trac 0.12, je poté nutné pro správnou funkcionalitu nainstalovat Bitten ve vývojové verzi 0.7.

Instalace programu Bitten je realizována pomocí skriptu napsaného v jazyce Python. Stačí tedy pouze stáhnout aktuální verzi z domovské stránky a spustit příslušný skript. Předpokládáme-li umístěni například v adresáři /home/bitten/ tak takto

```
/home/bitten/> python setup.py install

/home/bitten/> python setup.py test
```

První příkaz Bitten nainstaluje, druhý ověří správnost instalace.

Zbývá jen povolit Bitten v Tracu. K tomu je potřeba modifikovat konfigurační soubor nástroje Trac, přidat následující řádky

```
[components]
bitten.* = enabled
```

A nastavit uživatelům práva k vytváření XML skriptů a spouštění sestavování pomocí nástroje Bitten. Nastavit vybraným uživatelům `[yourname]` práva k vytváření XML skriptů, právo `BUILD_ADMIN` a anonymním uživatelům umožnit sledovat výsledky sestavení, právo `BUILD_VIEW`

```
$ trac-admin /path/to/projenv permission add [yourname] BUILD_ADMIN
$ trac-admin /path/to/projenv permission add anonymous BUILD_VIEW
```

Po povolení Bittenu se v grafickém rozhraní nástroje Trac objeví záložka „Build status“, po kliknutí na ní se zobrazí aktuální stav provádění všech aktivních XML skriptů pro všechny zadané testovací stroje.

Na testovacích strojích, kde server Trac neběží, stačí nainstalovat pouze klientskou část. Tato část se nazývá Bitten slave. Bitten slave pouze žádá o XML skript s úkoly Bitten master, snaží se je provést a o výsledku zpětně informuje Bitten master. Instalace se spouští obdobně jako instalace Bitten masteru, pouze je zde přidán parametr *–without-master*. Takto

```
/home/bitten/> python setup.py -without-master install
```

Instalace toho klienta nevyžaduje mít na daném stroji nainstalován Trac. Jediné co vyžaduje je mít nainstalován interpret jazyka Python. Pokud je na stroji nainstalován klient, je poté možné pomocí příkazu *bitten-slave* spouštět provádění Bitten skriptů na aktuálním stroji. Například takto

```
bitten-slave https://dev.nti.tul.cz/trac/flow123d/builds
```

Příkaz provede spojení s daným Trac serverem, v tomto případě se jedná o server *dev.nti.tul.cz/trac/flow123d*, odešle své údaje, hlavně tedy své jméno, o jehož použití se zmíním dál, a pokud se pro daný stroj nachází na serveru sestavení k testu, které by měl vykonat, pošle se mu ve formě XML skript, kde jsou zadané úkoly, které se mají provést.

5.2.2 Konfigurace

Pokud máme Bitten správně nainstalován a povolen, je potřeba pomocí grafického rozhraní v nástroji Trac nakonfigurovat, jaké činnosti se mají provádět. K tomu je využito skriptů ve formátu XML.

Nástroj Bitten umožňuje vytvořit více těchto skriptů s různým nastavením a popisem. Navíc umožňuje spouštět jednu konfiguraci na více strojích, toho je dosaženo

tím, že je každému skriptu přiřazeno, na jakých strojích se má spouštět. Jinak řečeno pro jakou platformu je určen. K tomu je využito vlastností, které o sobě pošle klient, když se připojí na server. Mezi vlastnostmi, které o sobě posílá, je například jméno serveru, IP adresa serveru, typ nebo verze operačního systému. K danému skriptu se poté dá přes webové rozhraní přiřadit více platforem, na kterých se bude spouštět, a budou se lišit právě těmito vlastnostmi, které je specifikují. Například pro klienta s operačním systémem *Linux*, bude mít vlastnost *family*, tedy k jaké rodině operačních systémů patří, hodnotu „posix“. Podle této masky se poté vybere skript určený pro daný stroj. Jeden skript může být určen pro více platforem. Výstup je poté oddělen pro různé platformy.

Formát XML skriptů má poměrně pevnou strukturu, která musí být v rámci pravidel XML validní. Základní syntaxe je, že celý skript je uzavřen v XML značkách <build>, takto

```
<build>
</build>
```

Ve značce `<build>` je nutné ještě definovat, odkud se má brát význam dalších použitých značek. Bitten umožňuje volat příkazy nástrojů Shell, Java, Python, PHP, Subversion, atd. Proto je nutné v této první značce uvést URL cestu ke jmennému prostoru daného příkazu. Tím je umožněno volat příkazy příslušející k této značce. Pro příklad zde uvedu, jak by měla vypadat tato značka v případě používání příkazů Shell v těle skriptu. Shell se označuje příkazem sh

```
<build xmlns:sh=http://bitten.edgewall.org/tools/sh>
```

V našem případě budeme využívat pouze možnosti Shell a Subversion.

V těle značky `<build>` se nacházejí jednotlivé kroky, která se mají během průběhu skriptu vykonat. Každý krok je možné umístit do vlastní párové značky `<step> </step>`. Ve výpise se poté jednotlivé kroky odlišují a je tímto možné rozdělit sestavení do více částí a tím zpřehlednit kontrolu vykonání celého sestavení a testů. Logické je, že prvním krokem většinou bývá provedení stažení aktuální verze z repozitáře. Pro ukázku, tento krok může vypadat takto

```
<step id="SVN checkout">
      <svn:checkout
      url="https://dev.nti.tul.cz/repos/flow123d/"
      path="${path}" revision="${revision}"/>
</step>
```

Argument `id` značí jméno daného kroku, který se poté bude zobrazovat ve výpise pod tímto jménem. V těle značky `<svn:checkout …>` se poté provede samotné stažení aktuální verze z repozitáře Subversion, argumenty jsou cesta k repozitáři, tedy kde se repozitář nachází, většinou URL adresa. Dále je možné zadat bližší cestu v repozitáři, je totiž možné, že se zde nachází více vývojových větví nebo projektů a my v daném případě chceme stáhnout pouze určitou část. Toto je zde reprezentováno proměnnou

${path}, jedná se o proměnnou prostředí Bitten a je automaticky nastavována podle toho, jak je daný skript v grafickém prostředí vytvořen, nebo-li jaký mu je nastaven adresář v repozitáři, který má stáhnout. Můžeme zvolit, jakou vývojovou větev chceme z repozitáře stáhnout a provést na ní činnosti dané XML skriptem. Bitten si zároveň spojí tento XML skript s daným adresářem v repozitáři a bude provádět sestavení pouze v případě, že došlo v tomto adresáři k nějaké změně. A také ještě jakou revizi, tedy jakou verzi zdrojových kódů, chceme z repozitáře stáhnout. Zde je opět nejlepší zadat proměnnou, kterou si samo nastavuje prostředí Bitten, a to ${revision}. Bitten si sám určí podle nastavení, jakou revizi má stáhnout.

Po stažení aktuální verzi z repozitáře, provedeme překlad programu. Jak již bylo řečeno předtím, konfigurace překladu se nastavuje pomocí souboru *makefile.in. N*áš překlad bude spouštět na různých strojích, tedy s různou konfigurací, je nutné nastavit hodnoty v souboru *makefile.in* tak, aby odpovídaly danému aktuálnímu stroji. Provedení je vysvětleno na příkladu. Před samotným spuštěním překladu se nakopíruje správný *makefile.in*, tímto krokem

```
<step id="Copying makefile.in">
                <sh:exec executable="cp"
 args="config/${name}.opt.mk.in makefile.in"/>
</step>
```

V těle kroku se provede jednoduché vykopírování daného *makefile.in* pomocí příkazu *cp* nástroje Shell. V adresáři programu Flow123d se nachází složka *config.* V ní jsou uloženy všechny konfigurační soubory pro překlad pro jednotlivé stroje. Jméno každého souboru se skládá ze jména stroje, pro který je určen, dále zda se jedná o konfiguraci pro produkční nebo testovací verzi a nakonec už jen z koncovky *mk.in*. Například pokud by se jednalo o konfigurační soubor určený pro stroj se jménem Hydra a bude určen pro produkční verzi, bude název tohoto souboru vypadat *hydra.opt.mk.in*.

Zbývá tedy jen určit, o který stroj se zrovna jedná. Klient, který se připojí na Bitten server odešle své údaje, mimo jiné své jméno. Bitten si tuto informaci uloží do proměnné ${name}, která je poté přístupná uvnitř skriptu. V příkladu se pokoušíme zkopírovat z adresáře *config* soubor, který se jmenuje *${name}.dbg.mk.in*, kde *${name}* nahradí Bitten jménem stroje, ze kterého je skript volán. Pokusí se daný soubor zkopírovat a uložit ho jako *makefile.in.* Pokud se mu toto nepovede nahlásí chybu. Ta poté bude zobrazena na výstupu Bittenu.

Máme staženou aktuální verzi programu Flow123d z repozitáře, nastaven správný konfigurační soubor pro překlad *makefile.in*, provedeme sestavení programu. Zavoláním příkazu *make all*. V syntaxi Bittenu takto

```
<step id="Make all">
      <c:make target="all" file="makefile"/>
</step>
```

Pokud je program sestaven, je dalším krokem, otestovat toto sestavení. K tomu využijeme systém skriptů na spouštění testů. Spouštění probíhá přes systém Make. Obdobně jako jsme spustili sestavení, můžeme spustit i všechny testy. Spuštění testu provedeme pro test, jehož jméno začíná `01` tímto krokem

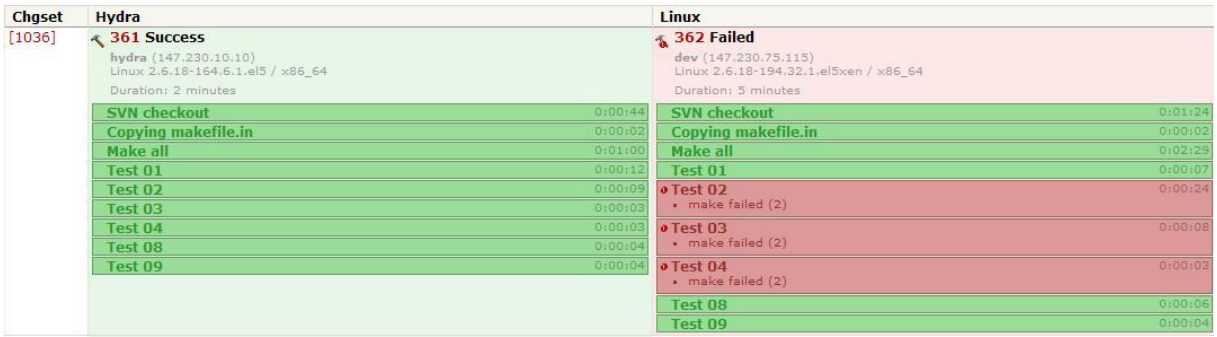
```
<step id="Test 01">
      <c:make target="01.tst" file="makefile"/>
</step>
```

Obdobně můžeme nechat provést i další testy.

Obr. 1 – Souhrný výstup s platformama

Na obrázku 1 je vidět, jak takový výstup poté vypadá. Je zde vidět pro jakou platformu a také, pro jakou vývojovou větev sestavení proběhlo. Na obrázku se jedná konkrétně o vývojovou větev „trunk - optimalized“ a platformy Hydra a Linux. Výsledek sestavení zda byly všechny kroky sestavení provedeny správně, je poté zobrazeno barvou. Zelenou úspěšné sestavení, červenou neúspěšné.

Každý krok je poté možno detailněji zobrazit. Viz obrázek 2



Obr. 2 – Zobrazení jednotlivých kroků

Zde už je poté vidět výsledek daného sestavení podrobněji. Jsou zde zobrazeny jednotlivé kroky. U každého je vidět, zda proběhl správně nebo se vyskytla chyba. Je zde vidět i doba trvání jednotlivých kroků.

Každý jednotlivý krok lze ještě více otevřít a podívat se, co bylo ve výstupu provedeno, včetně výstupu do konzole.

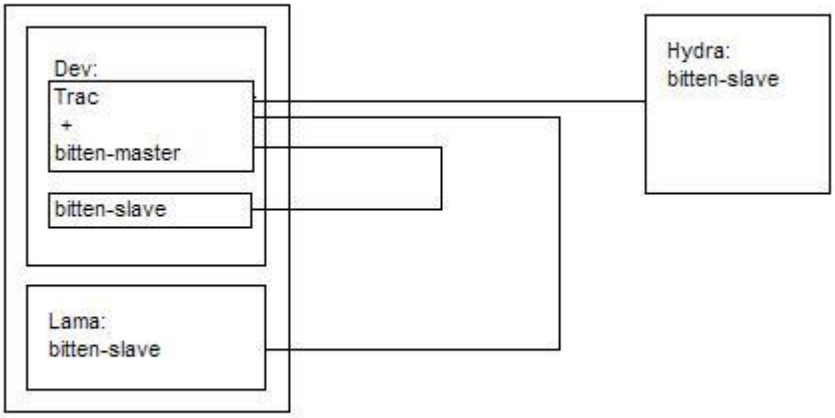
Obr. 3 – Detailní výstup jednotlivého kroku

Na obrázku 3 je vidět detailní výstup daného kroku. Obsahuje vše, co by mohlo být prospěšné v případě hledání chyby. Zde se jedná konkrétně o krok Test 02, ve kterém se provede stejně pojmenovaný test. Je zde vidět kompletní výstup všech skriptů, *run_test.sh*, *run_flow.sh* i *run_check.sh*. Včetně výstupu porovnávacího skriptu *ndiff.pl,* o kterém byla řeč dříve.

5.2.3 Praktické nasazení systému

Pro vývoj programu Flow123d bylo nutné testovat sestavení na třech rozdílných strojích, na systému s *Linux*, na systému umožňujícím paralelní výpočty na více procesorech a také na stroji s *Windows*.

Nástroj Bitten byl zprovozněn na třech rozdílných strojích. Jedná se o server Dev, na kterém běží operační systém *Linux*, dále cluster Hydra, který je vybrán jako zástupce rozsáhlejších paralelních výpočetních systémů a využívá dávkové zpracovávání úloh a nakonec server Lama, kde běží operační systém *Windows*.



Obr. 4 – Schéma propojení Bittenu

Na obrázku 4 je vidět propojení jednotlivých serverů. Server Dev a Lama se nachází na jednom fyzickém stroji a a jsou virtualizované. Server Hydra je samostatný stroj. Na serveru Dev je nainstalován nástroj Trac pro řízení vývoje projektu. Je zde nahrán plugin Bitten master, a protože se jedná o instalaci celého Bittenu do nástroje Trac. Klient je nainstalovaný na všech serverech, Dev, Hydra a Lama. Po spuštění klienta na daném stroji, se tento klient připojí na Bitten master, který běží na serveru Dev, pokud je zde nějaké sestavení k testování pro tento stroj tak je klientovi poslán ve formě XML skriptu, klient se jej pokusí vykonat a o výsledku zpětně informuje Bitten master, který poté pomocí nástroje Trac prezentuje výsledky této práce na webovém rozhraní.

Jako stroj, na kterém toto bude realizováno sestavování pod operačnímsystémem Linux, byl vybrán server Dev. Na tomto serveru se mimo jiné nachází také server pro verzování Subversion a také systém pro správu požadavků a chyb Trac. Protože se jedná o standardní stroj, spouštění výpočtů probíhá standardním způsobem, tedy přes skript *current_flow.sh,* jak bylo popsáno v předchozích částech. Cesta k tomuto skriptu je automaticky uložena do *current_flow* při překladu.

Sestavování na clusteru Hydra. Clusterem se rozumí systém více seskupených počítačů, které se na venek tváří jako jeden systém. Dohromady mohou být propojeny například pomocí ethernetové sítě. Cílem je získat vysoký výpočetní výkon spojením menších jednotek s malým výpočetním výkonem, které dohromady stojí méně než by stál jeden superpočítač o srovnatelném výkonu.

Spouštění úloh na tomto clusteru probíhá přes frontu úloh. V praxi to znamená, že pokud chceme provést výpočet nějaké úlohy, předáme tuto úlohu dávkovacímu systému, u clusteru Hydra je použit *Sun Grid Engine*, který danou úlohu zařadí do fronty a podle vytížení systému ji předá ke zpracování.

Zde bylo potřeba vytvořit speciální skript pro spouštění výpočtů a využít proměnnou MACHINE. Pokud je sestavení spouštěno na tomto systému, je při překladu do souboru *current_flow,* ve kterém je automaticky uloženo, který skript použít pro spouštění, uložena cesta k souboru *hydra_flow.sh* pomocí poměnné MACHINE. Samotný skript *hydra_flow.sh* obsahuje postup, jak úlohy na stroji Hydra spouštět. Skript musí vytvořit další skript, který v sobě bude obsahovat parametry pro dávkovací systém, který má úlohu spustit. A samozřejmě musí obsahovat co vlastně spustit pomocí exportovaných proměnných a voláním odkazu na knihovnu MPI, obdobně jako v kapitole 4.2.2. Vytvořený soubor se poté předá dávkovacímu programu a ten danou úlohu provede podle vytížení clusteru.

Poslední platformou, na které bylo sestavení vhodné testovat, je platforma s operačním systémem *Windows*. Využit pro praktické nasazení je server Lama, který běží virtualizovaně a s operačním systémem *Windows*.

Program Flow123d byl napsán pro operační systém *Linux,* proto je k překladu potřeba emulovat toto prostředí. K emulování je použit nástroj *Cygwin* [5]. *Cygwin* je kolekce volně šiřitelných programů, které emulují v operačním systému *Microsoft Windows* chování UNIXových systémů. Samotný *Cygwin* se skládá ze dvou částí. První částí je DLL knihovna zajišťující kompatibilní POSIXové volání v *API Win32*. Dále obsahuje řadu softwarových nástrojů, které zajišťují chování a funkce napodobující UNIXové prostředí. Mimo jiné například *GCC* překladač pro vývoj aplikací a mnoho dalších programů z UNIXu. Umožňuje také instalovat démony jako standardní služby *Windows*. Obsahuje také UNIXovou příkazovou řádku. Přes tu budeme provádět konfiguraci a samotný překlad, protože obsahuje již zmíněný kompilátor *GCC*.

V příkazové řádce tohoto programu se provede standardní postup pro překlad knihoven PETSC i programu Flow123d. Stejný je i postup instalace klienta systému Bitten.

6 Automaticky generovaná dokumentace Doxygen

Součástí programu Flow123d je i možnost generovat dokumentaci ze zdrojových kódů. K tomu je využit nástroj *Doxygen*, který pomocí komentářů a popisu jednotlivých tříd, skriptů, funkcí a dalších částí kódu vygeneruje dokumentaci. Zároveň je možné volitelně zobrazovat vztahy a relace mezi třídami. Navíc je možné využít více možností výstupu, například ve formátu HTML.

Pokud probíhá sestavení na testovacím stroji Dev, kde probíhá překlad programu pod operačním systémem *Linux*, je jedním z kroků, který je vykonán v rámci Bitten

skriptu, také vygenerování dokumentace. Pro generování dokumentace je, stejně jako například pro spouštění testu, použit nástroj Make. Příkazem

```
/home/flow123d > make online-doc
```

Generování této dokumentace je tímto příkazem provedeno. Tento systém byl již, obdobně jako systém pro spouštění překladu, hotov a nebylo to účelem mé práce.

V rámci mé práce jsem pouze prezentoval tuto dokumentaci na webové stránce. Na serveru Dev se nachází také webový server *Apache*. V rámci kroku v Bitten skriptu, během něhož se provede generování této dokumentace, se také vykopíruje tato dokumentace do složky, kde je volně přístupná v rámci webu. Tento krok vypadá takto

```
<step id="Make doxygen doc">
      <c:make target="online-doc" file="makefile"/>
      <sh:exec executable="cp" args="-rf doc/doxy/
      /home/nekvasil/public_html/1.6.5/"/>
</step>
```

Odkaz je umístěn na stránce Trac, kde je zařazen mezi dokumentaci a každý si ji může projít.

7 Závěr

Cílem mé práce bylo vytvořit systém automatického sestavování pro simulátor proudění Flow123d. Systémem automatického sestavování se vlastně rozumí používání principů průběžné integrace. Tato metodika značně usnadňuje vývoj software, protože sjednocuje vývoj ve více lidech a usnadňuje vyhledávání případných chyb.

Průběžná integrace se skládá z několika částí. Základem je uchovávání zdrojových kódů na jednom centralizovaném místě, v našem případě se jednalo o repozitář spravovaný nástrojem Subversion. Na tomto místě se nachází kompletní zdrojové kódy a knihovny, včetně knihoven třetích stran. Jediné, co nenaplňuje kompletně principy průběžné integrace je fakt, že program Flow123d vyžaduje ke svému překladu knihovnu PETSC. Tato knihovna není součástí repozitáře a je nutné ji na daném stroji ručně nainstalovat, pokud zde chceme používat simulátor Flow123d. Ovšem pokud je knihovna nainstalovaná, je možné daný stroj zprovoznit jako integrační a automaticky spouštět sestavování.

V rámci průběžného integrování je důležité testování daného sestavení. Testování je prováděno vypočítáním testovacích úloh a porovnáním získaných výsledků s referenčními. Byl vytvořen systém skriptů, který toto testování provádí. Část skriptů lze použít i samostatně, jako příklad uveďme skript *run_flow.sh*, který je dobře

použitelný ke spouštění výpočtů a značně usnadňuje spouštění hlavně na systémech s dávkovým zpracováním úloh.

Vlastní praktické nasazení průběžné integrace je realizováno pomocí nástroje Bitten. Systém monitoruje repozitář a po nahrání změn provede sestavení na integračních strojích, zde provede překlad a testování sestaveného programu a výsledky prezentuje na webovém rozhraní. Automatické sestavování probíhá na třech rozdílných strojích, na stroji se systémem Linux, na stroji, který umožňuje paralelizovat výpočty na větší množství procesorů, tedy na clusteru, a také na stroji s operačním systémem Windows.

Celý systém je poměrně modulárně navrhnut. Přidání dalších testů obnáší pouze vytvoření jednoho *makefile* souboru. Zajištěno je také spouštění výpočtů na různých strojích. Toho je docíleno spouštěním výpočtů přes specifické skripty pro daný systém. Stačí vytvořit krátký skript, který provede spuštění daným způsobem.

Poslední věcí, na kterou bych chtěl upozornit, je vytvoření jednotkových testů pomocí nástroje *CppUnit*. Tento bod jsme po konzultaci s vedoucím práce označili jako volitelný a dohodli jsme se, že se touto částí nebudu zabývat. Ovšem systém je připraven na budoucí implementaci těchto testů, protože nástroj Bitten umožňuje prezentovat tyto výsledky přes své webové rozhraní.

8 Seznam použité literatury

[1] FOWLER, Martin. *Continuous Integration*. [online]. 2006, 2, [cit. 2011-04-27]. Dostupný z WWW: <http://martinfowler.com/articles/continuousIntegration.html>.

[2] Apache Subversion. *Subversion Documentation.* [online]. 2011 [cit. 2011-04-27]. Dostupné z WWW: <http://subversion.apache.org/docs/>.

[3] PETSc*. PETSc: Documentation*. [online]. 2011 [cit. 2011-04-27]. Dostupné z WWW: <http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html>.

[4] Edgewall*. Bitten Documentation* [online]. 2003-2010 [cit. 2011-04-27]. Dostupné z WWW: <http://bitten.edgewall.org/>.

[5] Cygwin. *Cygwin Documentation.* [online]. 2011 [cit. 2011-04-27]. Dostupné z WWW: <http://cygwin.com/docs.html>.