Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Informatika a výpočetní technika

Počítačová grafika

Prostředí pro modulární vizualizaci dat (MVE)

Vypracoval: Michal Roušal

Vedoucí diplomové práce: Prof. Ing. Václav Skala, Csc.

Rok a místo vydání: Plzeň, 2000

Abstract

The interpretation of various data resulting from simulation and real world is an important of a disparate range of sciences. The field of data visualisation seeks to address this problem by providing appropriate analysis via graphical representations.

Various software tools now exist that provide suitable environments for a range of interpretational tasks. Modular Visualisation Environments are one from them. These provide a visual programming environment in which the user can manipulate data and create various graphical representations.

A major disadvantages of visualisation via standard MVEs are, that the systems are not easy to use, too complex, huge and very expensive.

To address these problems, a new system is presented. This system builds upon recognised benefits of MVEs, simplifying the visualisation, module creation process and introducing a framework that allows complex data (volumetric data in our case, for example) to be easily investigated.

OBSAH

1 Úvod

V současné době, kdy jsou lidé stále více a více zahlcováni obrovským množstvím informací se dostává do popředí problém, jak tato data co nejvíce přiblížit obyčejnému člověku. Jednou a podle mého názoru v současnosti nejpoužívanější metodou je vizualizace dat pomocí počítače. S tímto souvisí otázka nalezení nejvhodnějšího nástroje, pomocí kterého by bylo možno zobrazovat co nejširší spektrum různých datových typů, tak aby proces vizualizace (návrh vizualizačního procesu včetně samotného „výpočtu“ potřebného pro zobrazení požadovaných dat) nebyl časově náročný a byl zvládnutelný i pro nepříliš zkušeného uživatele. Naopak zkušenému uživateli by tento nástroj neměl klást žádné překážky v rychlé a efektivní práci.

Dalším významným problémem, z pohledu tvůrců vizualizačního software, ať již z akademické nebo komerční sféry, který se poměrně často vyskytuje a s předchozí problematikou na první pohled příliš nesouvisí je, jak spojovat práci jednotlivých členů vývojového týmu dohromady, bez větších komplikací. Pokud si představíme skupinu programátorů, kteří navíc používají různé programovací jazyky a vývojové systémy, může se jednat o značně komplikovanou problematiku.

V této diplomové práci chci prezentovat vlastní systém, který byl navržen tak, aby řešil, nebo se alespoň pokoušel řešit, výše zmíněné problémy a popíši některé charakteristické rysy, které se této problematiky týkají.

Tato práce byla podporována projektem MŠMT ČR - projekt VS 97 155.

2 Existující metody pro komplexní vizualizaci dat

Během vývoje v oblasti počítačové grafiky a vizualizace dat, ať již v oblasti hardware či software, se metody řešení tohoto problému rozdělily do tří hlavních kategorií.

Grafické knihovny

Poskytují poměrně značnou flexibilitu a dávají uživatelům mnoho možností jak kontroloval nebo pozměňovat výslednou formu vizualizace. Do této skupiny je možno zařadit jak nízkoúrovňové knihovny, které pracují již dále přímo s hardware počítače např. OpenGL , tak grafické knihovny, které jsou orientovány na specifický problém nebo poskytují celou škálu různých funkcí, jako například Unaris nebo NAG Graphical Supplement. Hlavní nevýhodou řešení vizualizace pomocí grafických knihoven je to, že je nejprve potřeba poměrně hodně času a trpělivosti, než se uživatelé naučí s danou knihovnou pracovat a ani poté, není implementace vybraného algoritmu zcela triviální.

Specializované systémy s omezenou sadou modulů (Turnkey systems)

Jedná se systémy, které jsou většinou určeny pro daný konkrétní problém. Například systémy pro filtrování a zpracování obrazu, grafické informační systémy. Používání těchto systémů je velice jednoduché a uživatelé nemají většinou žádné problémy naučit se s nimi pracovat. Pro problematiku k níž jsou určeny poskytují velmi dobré výsledky, ale pokud uživatel zrovna potřebuje něco trochu jiného, co v takovém systému není implementováno, musí se poohlédnout jinde.

Prostředí pro modulární vizualizaci dat (MVEs)

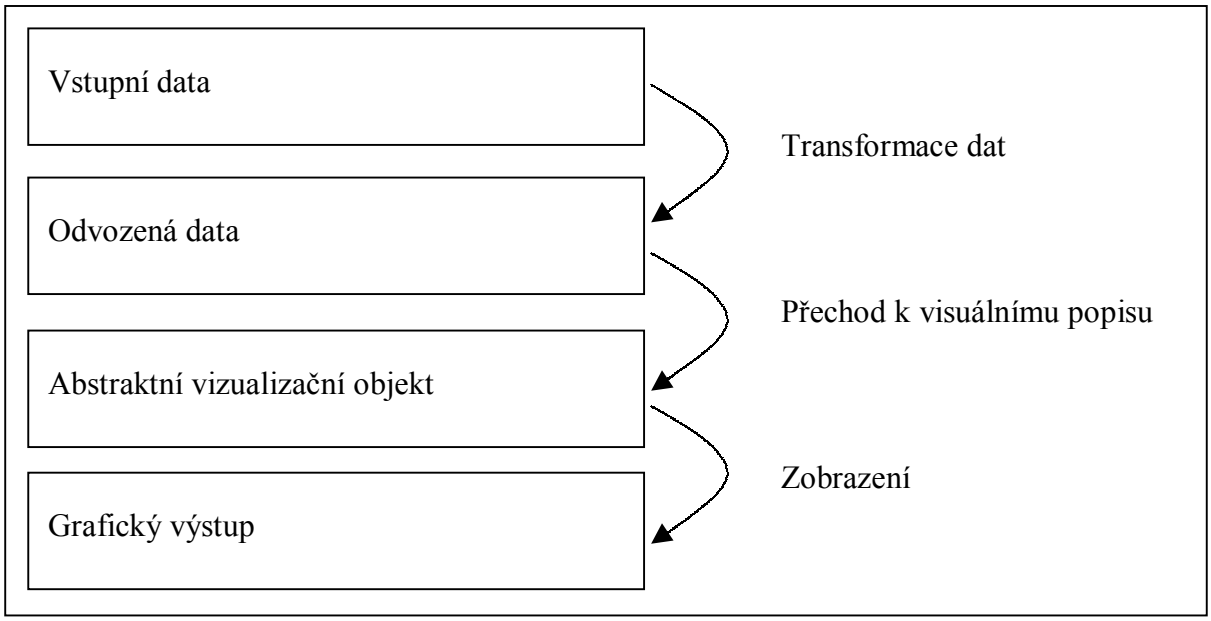
Systémy pro modulární vizualizaci dat (dále jen MVEs – Modular Visualisation Environmets) spadají někam mezi dva výše zmíněné grafické vizualizační systémy. Uživatel získá s takovýmto systémem určitou sadu modulů, které mohou být určeny pro konkrétní problém, nebo být obecného charakteru a pokrývat širší oblast. Pospojováním těchto modulů dohromady může uživatel vizuálně vytvořit vlastní aplikaci (síť modulů) pro vizualizaci svých dat. Moduly je možno si jednoduše představit jako funkce ve vyšším programovacím

jazyce. Tyto systémy také dovolují uživateli, který má nějaké specifické požadavky, napsat (naprogramovat) si vlastní moduly, jenž je poté možno přidat k již existujícím modulům a dále je používat.

3 Co je to MVE

Jak je popsáno výše MVE (Modular Vizualization Environments) jsou specifickým druhem systémů pro komplexní vizualizaci dat. Jejich specifikem je to, že se snaží být co možná nejotevřenější, tj. poskytují uživatelům ze všech možných nástrojů pro vizualizaci dat nejvíce možností co se týká variability použití a případné rozšiřitelnosti. Tyto systémy dávají uživateli ucelené prostředí pro vývoj vlastních aplikací pomocí, k tomu určenému, návrhovému systému.

MVE systémy používají data-flow model založený na konceptu vizualizačního pipeline. Schéma tohoto postupu je znázorněno na obrázku 3.1. Data procházejí sérií transformací až k produkci finálního zobrazení.

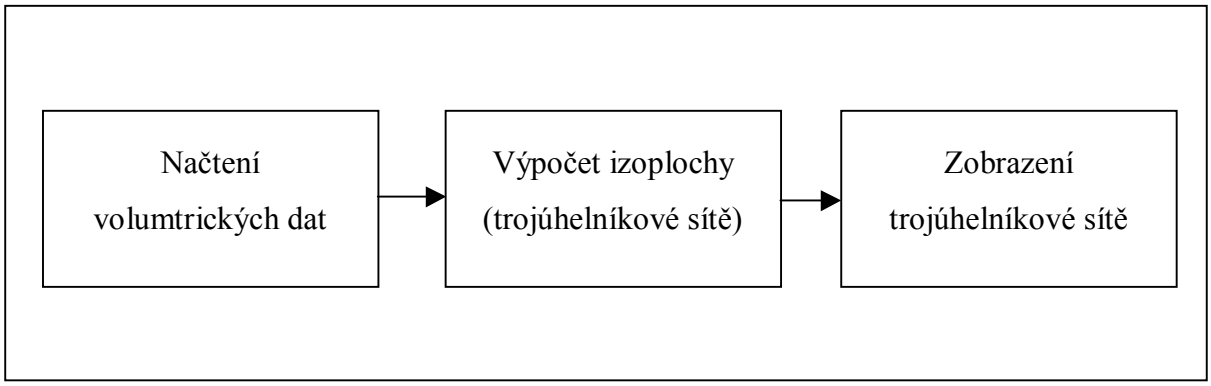


Obrázek 3.1

Jak vyplývá z pojmenování „Modular ...“ základní složkou takovéhoto systému jsou moduly, které reprezentují základní programovou jednotku aplikace navržené pomocí MVEs. Postup zpracovávání dat se popisuje ve formě obrazu, do něhož se umístí symboly algoritmů

(moduly). Celý proces vytváření vizualizační aplikace sestává ze tří hlavních částí. Nejprve je potřeba vybrat ty moduly, které budou pro řešení daného problému potřeba a umístit je na pracovní plochu. Poté je nutné propojit vstupy a výstupy modulů tak, aby byl zaručen správný tok dat mezi jednotlivými moduly. Nakonec je potřeba provést uživatelské nastavení jednotlivých modulů (zadat potřebné parametry pro jednotlivé algoritmy).

Například chceme zobrazit objekt s danou hustotou z volumetrických dat, která jsou uložena na disku jako soubor. Návrh vizualizačního schématu bude vypadat asi takto (viz obrázek 3.2). Z dané množiny dostupných modulů vybereme moduly pro Načtení volumetrických dat (Modul 1), Generování izoplochy (trojúhelníkové sítě) z volumetrických dat (Modul 2) a Zobrazovač trojúhelníkové sítě (Modul 3). Propojíme moduly tak, že výstup z Modulu 1 půjde na vstup Modulu 2 a výstup Modulu 2 propojíme na vstup Modulu 3. V nastavení modulů zadáme pro Modul 1 jméno souboru pro načtení a pro Modul 2 hustotu dat (objektu) pro která se má vygenerovat izoplocha. Po spuštění takto navržené sítě modulů se na obrazovce objeví okno zobrazující vygenerovanou trojúhelníkovou síť.



Obrázek 3.2

3. 1 Výhody a nevýhody MVE systémů

Jako první je potřeba uvést obrovskou variabilitu těchto systémů, kde pouhým propojením několika modulů může uživatel získat funkční aplikaci zcela podle svých představ. Je však potřeba podotknout, že se co týká navržené sítě modulů, nejedná se o novou, zcela samostatnou aplikaci, ale pouze popis propojení a nastavení modulů, k jehož spuštění je vždy potřeba nějaká další část, kterou si je možno například představit jako interpret interního jazyka daného systému.Výstupem většiny existujících systémů je opravdu jakýsi script v jejich interním jazyce. To dává pokročilému uživateli další možnosti využití těchto systémů

včetně možnosti provádět návrh sítě modulů pomocí vlastní aplikace, jejímž výstupem bude právě tento scriptovací jazyk, nebo přímo ročně zasahovat do výstupních souborů.

K hlavním přednostem modulárních vizualizačních systémů je zajisté potřeba připočíst možnost rozšiřování o vlastní moduly, případně i vlastní datové typy. Existuje velké množství sofwarových firem nebo i univerzitních pracovišť, které se zabývají vývojem specifických modulů pro tyto systémy a nabízejí je k dispozici.

Přestože otevřené modulární vizualizační systémy jsou navrženy tak, aby eliminovaly nevýhody jak grafických knihoven, tak i specializovaných systémů s omezenou množinou modulů, nejedná se stále ještě o zcela dokonalé systémy.

To co se na první pohled může jevit jako velká výhoda (většina existujících systémů má za sebou již pár let vývoje a jsou na relativně velmi vysoké úrovni co se programové realizace týká) se může ze strany obyčejného nepříliš zkušeného uživatele stát velkou překážkou k využití takovéhoto systému.

Tyto systémy jsou velmi rozsáhlé a také značně náročné na hardwarové vybavení počítače (počítačů), který má sloužit k jejich provozování. Díky velkému rozsahu funkcí a možností nastavení takového systému se s ním musí každý uživatel nejprve důkladně seznámit a naučit se s ním pracovat. Značná složitost často odradí nepříliš zkušené uživatele, kterým se nevyplatí strávit dlouhou dobu učením se systému, který potřebují používat pouze v omezeném rozsahu. Přístup k takovému systému pro nezkušeného uživatele může být stejně složitý, jako když se člověk bez dobré znalosti programování pokouší pro řešení problému vizualizace dat využívat grafické knihovny.

Další nevýhodou těchto systémů je jejich vysoká finanční náročnost (tj. jejich cena), i když je potřeba podotknout, že v této oblasti došlo v poslední době k velkému posunu a MVE systémy začínají být dostupné i pro obyčejné uživatele a ne jen pro velké podniky a instituce, které mají prostředky k jejich zakoupení, případně dlouhodobému obnovováni licence.

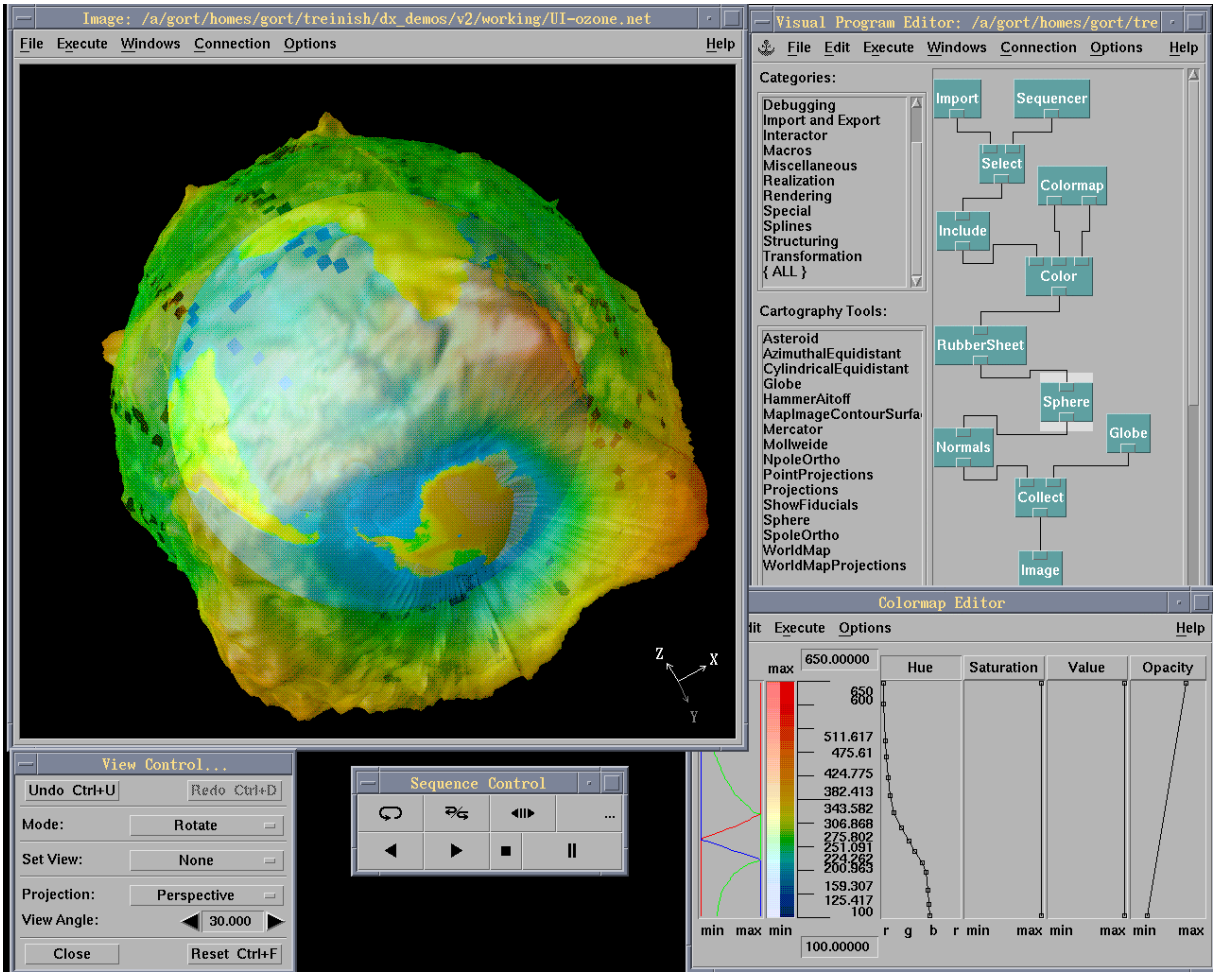
3. 2 Existující systémy

V této oblasti již existuje celá řada úspěšných komerčních systémů, z nichž některé bych zde rád stručně zmínil. Většinou se jedná o produkty velkých softwarových firem a jsou výsledkem dlouhodobého vývoje v této oblasti a práce celých vývojových týmů. Většina z nich obsahuje celou řadu podpůrných nástrojů jako např. *Module Generators* pro tvorbu

modulů atd. V současnosti tyto společnosti svádí souboj o to, který ze systémů se stane standardem a převládne na trhu.

IBM Vizualization Data Explorer, IBM

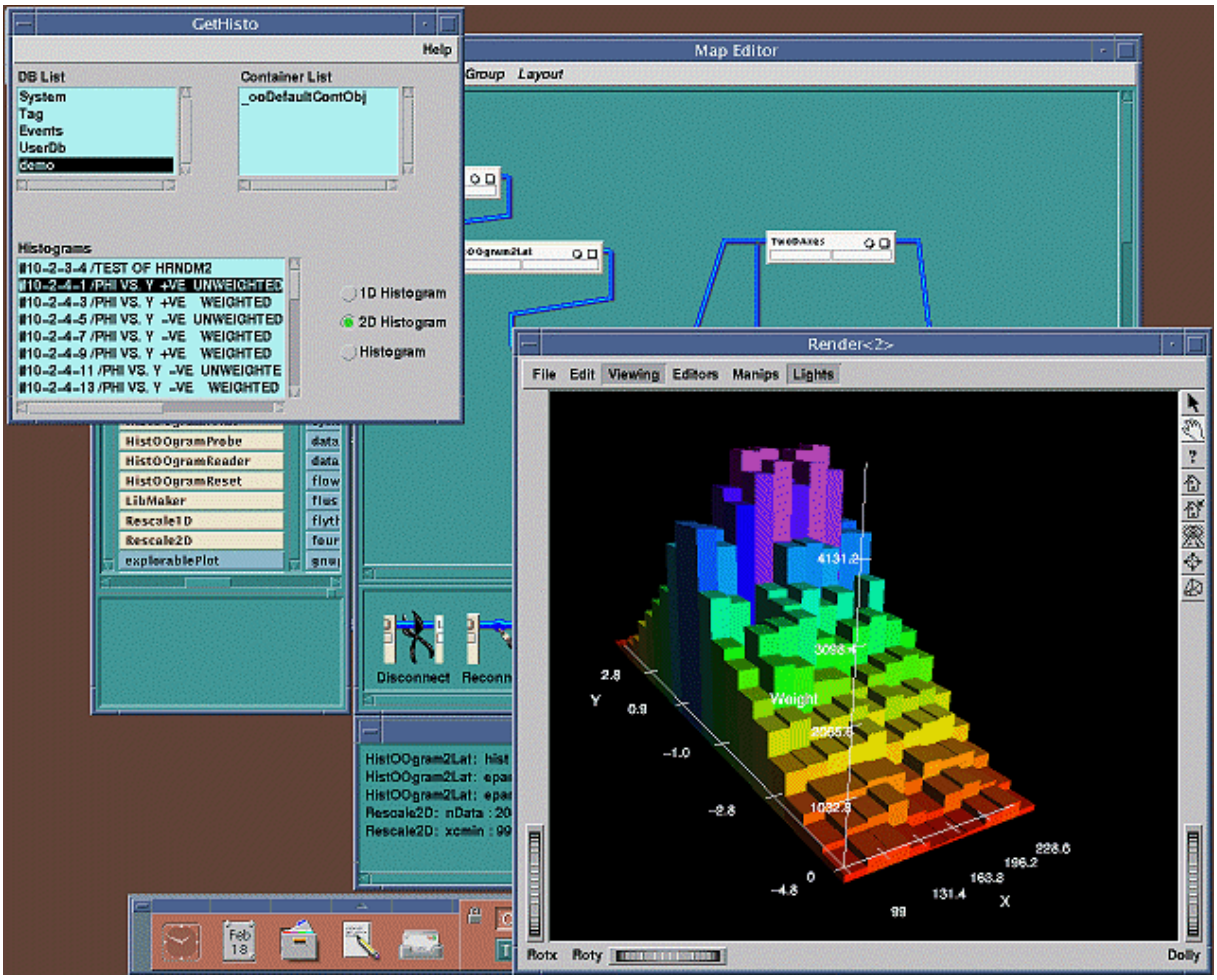
Rozsáhlý systém, který pokrývá celé spektrum problémů. V současnosti přešla firma IBM k jiné obchodní strategii a pro větší rozšíření tohoto systému je nyní celé jádro systému na internetu[1] volně k dispozici včetně zdrojových souborů což mu, dle mého názoru, dává poměrně značnou výhodu.



[1] http://www.research.ibm.com/dx/

IRIS Explorer[2], SGI

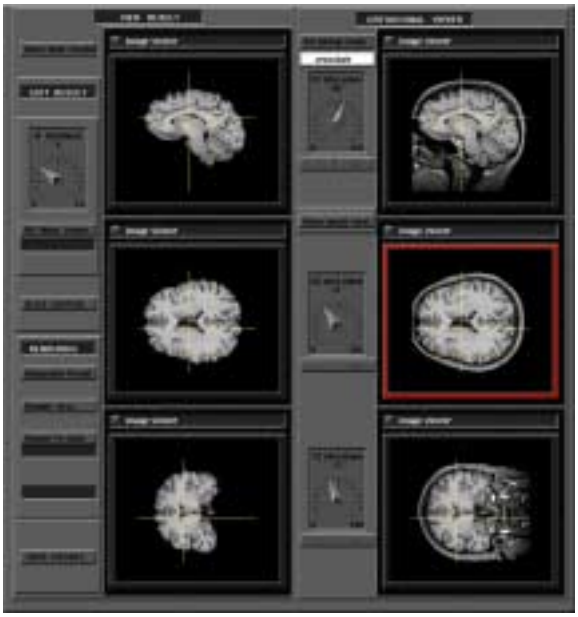
Obdobný software, tentokrát od SGI. Tento systém umí pracovat v distribuovaném prostředí, tj. pokud běží více serverů od IRIS Exploreru na různých počítačích, je možno v navržené síti spouštět i vzdálené moduly běžící na jiném počítači. IRIS Explorer také obsahuje poměrně propracovaný a značně automatizovaný sytém tvorby nových modulů. S tímto systémem mám osobní zkušenost a i přes některé drobné nedostatky (dle mého názoru) se jedná o velice kvalitní a propracovaný systém.

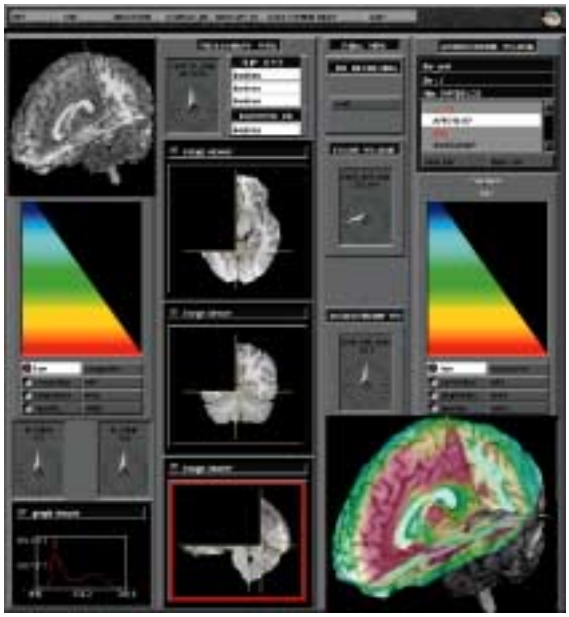


[2] http://www.nag.co.uk/Welcome_IEC.html

AVS modular vizualization environment[3], Advanced Vizual Systems Inc.

Znovu MVE systém s celou řadou možností, ale tentokrát již ne tolik rozšířený, alespoň podle mých informací. Poměrně silnou pozici má však tento systém v oblasti lékařských aplikací a to jak co se týká zpracování 2D obrazu, tak i zobrazování a extrakce izoploch a informací z volumetrických dat. Systém také umožňuje běh v distribuovaném prostředí a spuštění modulů na superpočítačích nebo masivně paralelních strojích.



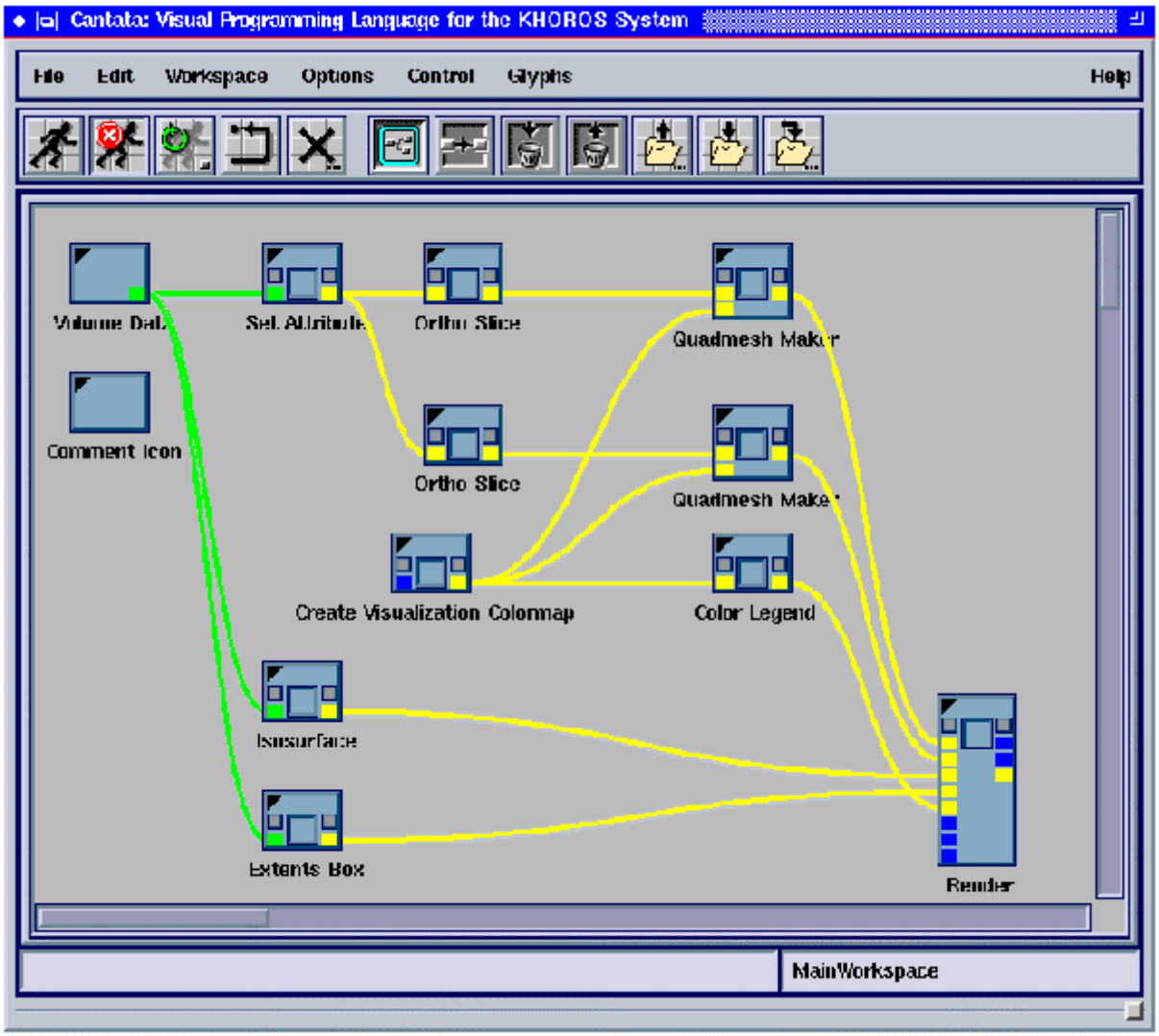


Khoros[4], Khoral research Inc.

Modulární vizualizační systém zaměřený především na zpracování signálu a to jak zvukového, tak i obrazového. Systém obsahuje celou řadu filtrů a dalších možností pro zpracování tohoto typu dat.

[3] http://www.avs.com/products/AVS5/avs5.htm

[4] http://www.khoral.com/



Další systémy:

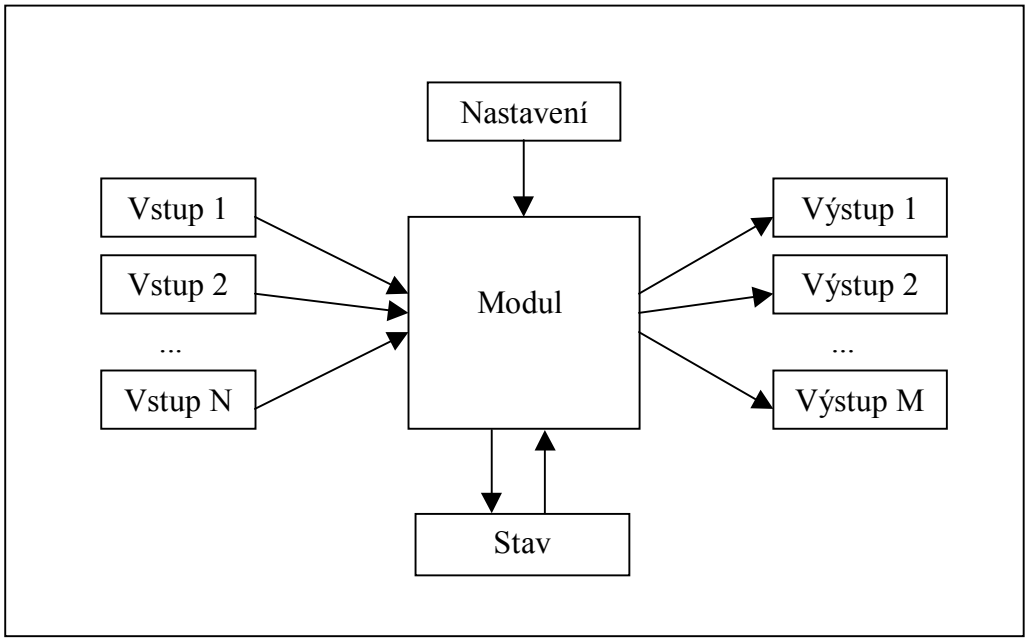
Amber[5], IFS Ltd.

[5] http://www.ifsys.demon.co.uk/html/amber.html

4 Moduly

Modul si můžeme představit jako funkci nebo objekt, který má nějaké vstupy a výstupy (obrázek 4.1). Pro MVEs jsou moduly základními stavebními prvky, pomocí nichž buduje uživatel vlastní aplikaci.

Každý modul má možnost nechat si zadat některá pro něj specifická data od uživatele, ať již pomocí dialogu nebo jinou cestou a uchovávat svůj stav mezi jednotlivými spuštěními. Moduly mívají své ovládací prvky, sloužící k nastavování pracovních parametrů. Parametrem bývá nejčastěji skalár (celé nebo reálné číslo) nebo textový řetězec.



Obrázek 4.1

Základní moduly u libovolného systému lze jednoduše rozdělit do několika kategorií podle činnosti, kterou tyto moduly mají vykonávat.

vstupní moduly – slouží pro načítání dat ze souborů různých formátů, nebo generování dat podle daného algoritmu (např. generování trojúhelníkové sítě pro danou funkci dvou proměnných),

filtry – zpracovávají data, ale nemění jejich sémantický význam (například filtrací obrazu mediánovým filtrem se změní číselné hodnoty, ale obraz zůstává obrazem),

mapovací a výpočetní moduly – mění formu dat z jedné reprezentace do jiné (například modul pro generování trojúhelníkové sítě z volumetrických dat má na vstupu trojrozměrné pole hodnot a jeho výstupem je geometrický popis povrchu vzniklého tělesa),

řídící moduly – slouží k vytváření cyklů, reakcím na vnější podněty a k větvení výpočtu,

výstupní a zobrazovací moduly – jsou posledním článkem výpočetního řetězce a slouží buď k ukládání výstupních dat do souborů, nebo pro zobrazení finální informace ve formě grafů, dvoj- či trojrozměrného obrazu (rendering) a pro grafickou interakci s uživatelem.

4. 1 Problémy, které je potřeba řešit

Při výběru způsobu realizace modulů je potřeba mít na zřeteli několik zásadních předpokladů. Jedná se o následující podmínky, které byly buď dány pro daný systém a nebo vyplývají ze specifikace a určení vyvíjeného systému jako vizuálního modulárního návrhového nástroje.

Více různých jazyků a vývojových nástrojů pro tvorbu modulů

Vzhledem k předpokladu, že moduly do prezentovaného systému MVE bude vyvíjet poměrně různorodá množina uživatelů, nelze zaručit použití (znalost) jednotného jazyka nebo vývojového nástroje. Jelikož je cílem zpřístupnit systém pokud možno všem, kteří o to budou mít zájem, je potřeba navrhnout realizaci modulů tak, aby uživatelé mohly moduly vytvářet pomocí nástroje na nějž jsou zvyklí a nemuseli se složitě učit pracovat s něčím novým.

Easy to create (pro programování modulů nesmí být potřeba žádné speciální znalosti)

Vytváření modulů musí být co možná nejjednodušší. Uživatel by neměl být nucen učit se něco nového, pokud potřebuje vytvořit pouze modul pod již existující systém. Přepracování již existujících algoritmů nebo aplikací do podoby modulů by mělo být pokud

možno bez problémů co se implementačních záležitostí týká. Tuto problematiku řeší výše popsané komerční systémy pomocí tzv. *ModuleGenerators*, což jsou samostatné programy, které jsou schopny k dodané funkci v C nebo jazyce Fortran automaticky vygenerovat celý interface modulu. Některé tyto systému obsahují vlastní kompilátory a knihovny pro podporované programovací jazyky, pomocí nichž jsou poté moduly generovány. Vytvoření takovéhoto generátoru modulů je však z programátorského hlediska velice náročné a vyžaduje vysokou úroveň znalostí v oblasti kompilátorů a softwarového inženýrství.

Platforma MS Windows

V tomto případě se nejedná o nijak silné omezení, jelikož v prostředí MS Windows jsou v současnosti dostupné všechny funkce jako např. pod systémy typu Unix. Všechny výše popsané komerční MVE systémy jsou v současnosti dostupné jak pro platformu MS Windows, tak pro platformu Unix.

4. 2 Možné způsoby realizace

Procesy, objekty nebo množina funkcí

Otázka reprezentace modulů je poměrně důležitá, jelikož se přímo dotýká veškeré další práce na tomto druhu systémů. Obecně lze modul popsat jako proces (vlastně jedna funkce, která může samozřejmě volat další funkce) s nějakými vstupy a výstupy, jako objekt, kde pro jednotlivé funkce modulu jsou definovány metody, a nebo jako množinu funkcí, kde jednotlivým činnostem odpovídají samostatné funkce.

Využití realizace modulu pomocí procesu je poměrně komplikované a neposkytuje příliš mnoho možností pro uživatelsky přívětivé nastavování parametrů modulu, případně vyšší stupeň kontroly výpočtu ze strany řídícího sytému. Celý modul by vypadal jako černá skřínka s nějakými vstupy a výstupy.

Objektová reprezentace modulu ze vzhledem k charakteru systému MVE přímo nabízí. Využití objektů by jednoznačně odpovídalo výše popsanému charakteru modulů (samostatná jednotka poskytující svému okolí k dispozici vybrané metody). Bohužel

nekompatibilita objektů v různých programovacích nástrojích (VC++, Delphi, Builder) zabraňuje použití klasických objektů z OOP pro reprezentaci modulů. Co se týká objektové reprezentace, připadá ještě v úvahu využití objektově orientovaných standardů CORBA nebo DCOM, kde jsou jednotlivé objekty popsány pomocí speciálního IDL (interface definition language).

Další variantou je popis činnosti modulu pomocí množiny funkcí, jedná se o obdobné řešení jako popis pomocí objektu, až na to, že jednotlivé funkce nejsou vázány uvnitř jedné entity (objektu v OOP), ale jsou zcela samostatné. Toto řešení je možno implementovat ve všech potřebných vývojových prostředcích. Je pouze potřeba jednoznačně specifikovat, které funkce náleží ke kterému modulu.

Pro nedistribuovaný (lokální) systém

Samostatné programy

Moduly lze realizovat jako samostatné programy a data mezi nimi předávat pomocí pevně definovaného rozhraní. Komunikace mezi jednotlivými moduly může probíhat například na úrovní *socketů (schránek)* nebo *pipes (rour)*. Toto řešení by předpokládalo implementaci funkcí pro realizaci datového přenosu do všech modulů i s tím, že by se musel vybrat takový druh komunikace mezi moduly, který by bylo možno použít pro všechny požadované vývojové nástroje. Tuto variantu by bylo možno použít i pro distribuovaný systém, samozřejmě za předpokladu zvolení takového způsobu komunikace mezi jednotlivými moduly, který by distribuci výpočtu umožňoval.

Dynamicky připojovatelné knihovny funkcí

Toto řešení předpokládá využití dynamicky připojovatelných knihoven funkcí pro reprezentaci modulů. Většina operačních systémů v současnosti podporuje možnost dynamického připojování částí kódu za běhu programu. V tomto případě se vytváří nejtěsnější vazba mezi moduly a samotným řídícím systémem výpočtu. Jednotlivé moduly se při spuštění stávají vlastně součástí hlavního programu celého systému.

Také poskytuje optimální vlastnosti pro lokální implementaci, jelikož umožňuje, z pohledu řídícího systému, zcela nezávislý přenos dat mezi moduly (řídící systém nemusí nic vědět o tom jaká je struktura dat, která se předávají), kdy je možné mezi moduly předávat pouze ukazatele na data, jelikož se ze strany operačního systému jedná vždy pouze jeden proces a veškerá paměť je přístupná všem modulům. Nevýhodou tohoto přístupu je obtížnější ladění modulů, jelikož chyba při běhu jednoho modulu se z vnějšku jeví jako chyba celého systému MVE.

Distribuované

PVM, MPI, ...

PVM (Paralel Virtual Machine) nebo MPI (Message passing interface) jsou systémy (knihovny) navržené pro jednodušší vytváření distribuovaných aplikací. Tyto systému poskytují uživateli sadu funkcí a programů, s jejichž pomocí lze programovat distribuované aplikace, aniž by musel programátor řešit nízko-úrovňové problémy, jako je realizace mezi-procesní komunikace nebo přenos dat po síti. Moduly by bylo možno realizovat jako paralelní procesy nad některým z těchto standardů. Problémem je dostupnost potřebných knihoven funkcí pro všechny potřebné vývojové nástroje a jazyky a potřeba znalosti použitého standardu u všech autorů modulů.

CORBA

Common Object Request Broker Architecture (CORBA) je specifikace objektově orientovaného distribuovaného heterogenního prostředí, navržená skupinou OMG. Součástí této specifikace je Object Requst Broker zajišťující transparentní volání vzdálených objektů, definice Common Services a Common Facilities.

Základní funkcí architektury CORBA je podpora jazykově neutrálního transparentního použití distribuovaných *objektů*. Klient může používat dostupné objekty bez ohledu na to, v jakém jazyce jsou implementovány, kde a na jakém počítači běží a jakým komunikačním

protokolem jsou dostupné. Objektem se přitom rozumí identifikovatelná, zapouzdřená entita, která poskytuje nějaké *služby*. Klient přistupuje k těmto službám zasíláním *požadavků*. Forma generování požadavků závisí na jazyce, ve kterém je klient napsán. V objektově orientovaném jazyce to může být volání metod zástupného objektu, zatímco v klasických programovacích jazycích volání vygenerovaných funkcí (*stubs*). Plná implementace prostředí CORBA se skládá z prostředí klientského, z prostředí na straně serveru a objektově orientovaných služeb.

Tento standard je v současnosti dostupný pro všechny potřebné programovací nástroje. Problémem zůstává nutná znalost alespoň jeho základů u všech programátorů píšících moduly a značný nárůst složitosti celého systému pro řízení výpočtu a jeho vazby na editor pro návrh sítí modulů. Každopádně se však jedná v současné době o nejvhodnější platformu pro přechod k distribuovanému systému. Další výhodou je poměrně jednoduchá možnost realizace rozhraní pro WWW.

COM/DCOM

Distributed Common Object Model je distribuovaná verze architektury Common Object Model firmy Microsoft. Jde o alternativu specifikace CORBA, která se snaží řešit stejné problémy, ale zcela odlišnými prostředky. Bohužel ale nejde o objektově orientovanou architekturu ale pouze o architekturu na objektech založenou. DCOM je jádrem Network OLE a prvků ActiveX (což jsou ostatně synonyma) a je součástí Windows NT 4.0 a Windows 95.

Dále je to MTS (Microsoft Transaction Server), který doplňuje DCOM o další funkce a vytváří prostředí pro existenci distribuovaných objektů. Umožňuje definovat thread model objektů, transakce a práva uživatelů k objektům. Zajímavé také je, že bude přímou součástí Windows NT 5.0.

Možnosti využití tohoto standardu pro vývoj MVE jsou obdobné jako pro standard CORBA, pouze s tím rozdílem, že zde se jedná o nativní technologii firmy Microsoft a tím je zajištěna stálá podpora pod operačním systémem MS Windows. Zda se jedná o výhodu či nevýhodu již necháme na uvážení čtenáře.

4. 3 Zvolené řešení

V tomto oddílu se pokusím zdůvodnit proč a jaké z výše popsaných možných řešení jsem zvolil pro reprezentaci modulů v novém systému v závislosti na daných předpokladech a dalších důvodech.

Moduly jsem se rozhodl reprezentovat pomocí množiny funkcí (každá funkce modulu reprezentuje jednu jeho výkonnou část), které jsou uloženy v dynamicky připojovatelné knihovně (DLL). Pro tento postup jsem se rozhodl z následujících důvodů.

Vzhledem k tomu, že systém je vyvíjen především pro práci s velkými objemy dat (např. volumetrická data), se nevyplatí řešit problematiku distribuovaného systému, kde je nutno přenášet tato data po síti. Časová náročnost přenosu po dostupné síti by nejspíše zcela znehodnotila časovou úsporu získanou distribucí výpočtu. Navíc by byla implementace distribuovaného systému značně náročnější a to nejen z pohledu jeho vývoje, ale také pro autory jednotlivých modulů.

Pro možné autory modulů se jedná o nejvhodnější variantu, jelikož se nemusejí učit pracovat se žádnými novými prostředky nebo knihovnami a snadno mohou využít a jednoduše pozměnit již existující programy. Všechny existující (nám známé) programové nástroje pro platformu Microsoft Windows podporují možnost vytváření dynamicky připojovatelných knihoven (DLL).

Jak již bylo zmíněno výše, poměrně vhodné řešení, kde by byly pro reprezentaci modulů použity klasické objekty z OOP, nelze použít, jelikož interní reprezentace objektů pro různé vývojové nástroje není kompatibilní.

4. 4 Konkrétní popis řešení

Modul jednoznačně charakterizuje jméno DLL knihovny v níž se modul nachází a jméno samotného modulu. Každý modul je reprezentován pomocí 6 funkcí, které představují metody, které jsou od modulu vyžadovány.

Všechny funkce mají pevně daná pravidla pro jejich pojmenování, jedná se o jméno konkrétního typu funkce, k němuž je jako předpona přidáno jméno modulu (viz Tabulka 4.1).

Přehled funkcí obecného modulu
ModuleName_MAIN_MODULE_FUNC – Hlavní výkonná funkce modulu, která reprezentuje samotnou činnost modulu.
ModuleName_SETUP_FUNC – Funkce pomocí které je umožněno uživatelské nastavení modulu uvnitř návrhového systému (editoru). V této funkci může autor modulu zobrazit libovolný dialog pomocí něhož mohou být nastaveny všechny parametry modulu, které nejsou nebo nemohou být součástí vstupních dat modulu.
ModuleName_FREE_DATA – Funkce pro uvolnění datových struktur, které jsou vráceny jako výstup hlavní funkce modulu (*ModuleName_MAIN_MODULE_FUNC*).
ModuleName_FREE_SETUP_DATA – Funkce pro uvolnění dat reprezentujících uživatelské nastavení modulu. Tato data byla alokována ve funkci (*ModuleName_SETUP_FUNC*)
ModuleName_FREE_STATE – Funkce pro uvolnění možných stavových dat modulu.
ModuleName_HELP_FUNC – Tato funkce je nepovinná a umožňuje autorovi modulu předat podrobnější informace o modulu (textový řetězec např. jednoduchá nápověda), tyto je pak možno zobrazit pomocí návrhového systému.

Tabulka 4.1

Pro konkrétní uživatelský modul je potřeba nahradit řetězec *ModuleName*, skutečným jménem modulu. Pojmenování modulu je ponecháno zcela na vůli jeho autora, jméno by však mělo být jednoslovné a vyjadřovat funkci modulu.

Funkce pro uvolňování paměti jsou nutné, jelikož při alokaci paměti ve funkci, která se nachází v dynamicky připojené DLL knihovně, platí, že paměť se alokuje v segmentu, kam byla namapována DLL knihovna a nikoli v hromadě hlavního programu. Z hlavního programu realizujícího řízení výpočtu tuto paměť není možno uvolnit a tak musí být uvolněna funkcí z té samé DLL knihovny, jako byla alokována. Navíc runtime nemůže znát strukturu dat, kterou předává mezi moduly (předávají se jen ukazatele na data, v případě strukturovaných dat vázaných přes další ukazatele nelze jednoduše paměť uvolnit).

V současnosti jsou v systému rozlišovány čtyři základní typy modulů viz tabulka 4.2. Toto dělení má pro samotný systém praktický význam pouze v případě rendereru, se kterým se v některých momentech pracuje jinak, než s ostatními moduly. Jinak se jedná většinou o informaci pro uživatele.

Typy modulů	
Loader	– Modul sloužící k načítání případně generování vstupních dat.
ComputingModule	– *Obecně jakýkoliv vnitřní modul systému*
Saver	– *Modul pro ukládání výstupních dat do souboru na disk*
Renderer	– *Modul pro zobrazování*

Tabulka 4.2

A následující datové typy (tabulka 4.3)

Typy dat	
Undefined	– Pro možnost realizace typově nezávislého vstupu/výstupu
Points	– Množina bodů
Triangles	– Trojúhelníková síť
Tetrahedras	– Tetrahedronová síť
Image	– Obrázek
Volume	– Volumetrická data
Freq	– Frekvenční data
COSFreq	– Další typ frekvenčních dat

Tabulka 4.3

Jelikož systém je navržen tak, aby byl co nejvíce otevřený, přidání nových typů modulů, případně nových typu dat, není žádným problémem. Data obsažená v tabulkách 4.2 a 4.3 jsou uložena v podobě textových konfiguračních souborů a jejich pozměnění je velice jednoduché.

5 Spouštění navržené sítě modů – Runtime

Pokud již máme moduly a pomocí nějakého návrhového systému (editoru) je z těchto modulů vytvořeno vizualizační schéma, nastává problém jako tuto síť modulů spustit. Tuto problematiku by měla řešit programová část, kterou budeme nazývat Runtime. Runtime musí podle popisu vizualizačního schématu, který vytvoří uživatel pomocí návrhového systému, zajistit volání jednotlivých modulů ve správném pořadí, synchronizaci spouštění modulů a přenos dat mezi moduly tak, aby se navržená síť chovala z pohledu uživatele jako jedna aplikace.

5. 1 Problémy, které je potřeba řešit

Jak je patrné, runtime je součástí nižší úrovně celého sytému, stejně jako jednotlivé moduly, zatímco návrhovou část je možno označit za úroveň vyšší (tj. bližší uživateli). Z toho vyplývá, že, co se implementace týká, musí být realizace runtime poměrně úzce svázána se samotnými moduly (způsobem implementace modulů). Tudíž je potřeba řešit často obdobné nebo přímo navazující problémy, jako v případě realizace samotných modulů. V této kapitole se proto budu zabývat pouze specifickými problémy, které se týkají této části prezentovaného systému.

Synchronizace běhu modulů

Při spuštění navržené sítě modulů je potřeba vyvolávat jednotlivé moduly, tak aby celý výpočet proběhl správně. Nejprve je potřeba spustit moduly, které nemají žádný vstup z jiného modulu, tedy moduly pro načítání nebo generování dat, poté výpočetní moduly ve správném pořadí a nakonec výstupní nebo zobrazovací moduly.

Předávání a nastavování parametrů modulů

Tento problém částečně souvisí se způsobem nastavování modulů. Obecně je potřeba uvažovat, že po uživatelském nastavení modulu je potřeba toto někde uchovat a při spuštění navrženého vizualizačního schématu znovu modulu zpřístupnit. Nastavení modulu (data reprezentující nastavení) nelze vázat přímo s modulem, ale je potřeba je připojit k objektu, který reprezentuje modul při návrhu, případně při ukládání navržené sítě modulů do souboru. Další otázkou je zda parametry modulů uchovávat jako binární data bez jakékoliv podobnější informace o nich, a nebo definovat možné datové typy, které by bylo možno pro nastavení modulů použít.

Předávání dat mezi moduly

Řešení tohoto problému se přímo odvíjí od zvolené metody pro reprezentaci modulů, přesněji od toho, zda se má jednat o systém distribuovaný nebo lokální. V případě distribuovaného systému se jedná o složitou problematiku s více méně rozdílnou implementací, podle zvolené metody (CORBA, PVM, ...). Bez použití nějaké nadstavby nad sítovou vrstvou je nutné provádět výměnu dat např. pres *sockety*. Výše zmíněné standardy pro vývoj distribuovaných aplikací tuto problematiky částečně řeší sami, ale ani tak není realizace přenosu velkých objemů dat po počítačové síti nijak jednoduchá. Pokud se jedná o nedistribuovaný systém je vše již značně jednodušší. Pro lokální systém je možno pro výměnu dat mezi moduly použít např. sdílenou paměť.

Uvolňování nepotřebných dat

Během výpočtu je potřeba uvolňovat nepotřebná data, možností jak toho dosáhnout je zajisté mnoho, ale jedná se především o dva rozdílné přístupy. Data může vlastnit (modul po výpočtu předá data systému) a uvolňovat přímo runtime, a nebo je alokace a dealokace dat součástí modulu. Pokud má data uvolňovat runtime, je nutno definovat datové typy, které je a bude možno používat, přímo v runtime, jelikož pokud chceme uvolnit složitější data, musíme znát jejich strukturu. Toto řešení omezuje možnost snadné rozšiřitelnosti systému o moduly, které pracují s jinými datovými typy, než co jsou řídícímu systému známi, ale je vhodnější pro

distribuované systémy. Pokud jsou funkce pro dealokaci dat umístěny přímo v modulech a jedná se o nedistribuovaný systém, je možno vytvořit runtime zcela nezávislý na přenášených datových typech (datové struktury mezi dvěma různými navazujícími moduly musí být samozřejmě identické).

5. 2 Možné způsoby realizace

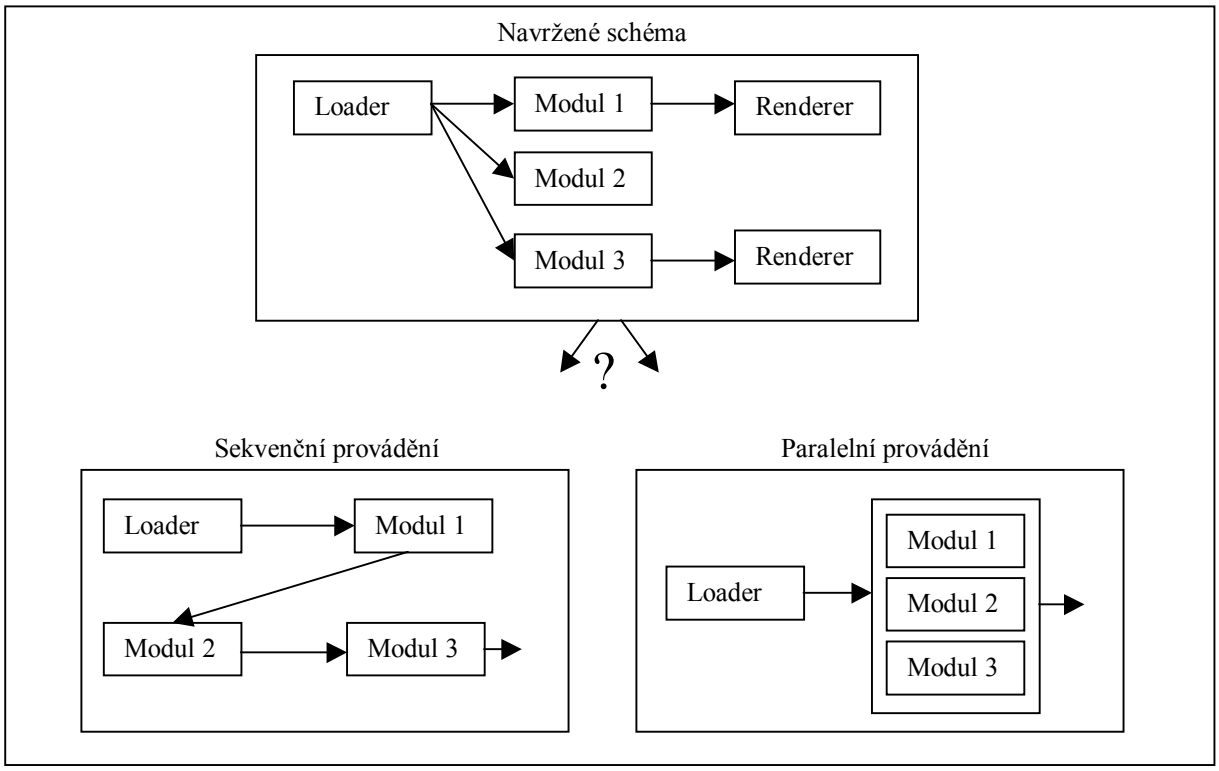
Interní reprezentace modulů

Jelikož jsou moduly reprezentovány pomocí množiny funkcí, nabízí se otázka, jak je reprezentovat uvnitř samotného systému. V úvahu připadají pouze dvě možnosti. Je možno pracovat s jednotlivými funkcemi modulů tak jak jsou, beze změny charakteru popisu modulů. Toto řešení by však bylo velice nepřehledné. Vylepšit je ho možno použitím speciálních seznamů a tabulek funkcí pro jednotlivé moduly, ale i tak by bylo ladění, případně pozdější realizace určitých změn v takovémto programu poměrně náročné. Druhou možností pro interní reprezentaci modulů je přechod od popisu pomocí funkcí, k popisu pomocí objektů. Zde již odpadá problém s různými programovacími jazyky pro vývoj modulů, jelikož reprezentace funkcí uvnitř dynamicky připojovatelných knihoven je transparentní. Z hlediska přehlednosti a rozšiřitelnosti je objektový charakter modulů pro řídící systém jednoznačně nejvýhodnější.

Paralelní nebo sekvenční spouštění modulů

Pro spouštění modulů mohou také existovat dva různé modely. Navržené schéma modulů lze spouštět tak, že jednotlivé moduly jsou spouštěny postupně (jeden po druhém). Samozřejmě ve správném pořadí, které je dané tokem dat mezi moduly. Nebo je možno spustit ty moduly, u kterých je to možné paralelně (z hlediska toku dat musí být tyto moduly nezávislé) (obrázek 5.1). Paralelní běh modulů vyžaduje, aby jejich program v hlavní funkci (*..._MAIN_MODULE_FUNC*) byl reentrantní, tj. musí být možno v případě potřeby zavolat tuto funkci znovu nebo i vícekrát, před tím než první skončí.

Na jednoprocesorovém nedistribuovaném systému není v rychlosti provádění výpočtu žádný rozdíl (teoreticky, prakticky je možno drobného urychlení dosáhnout), v případě víceprocesorového nebo dokonce distribuovaného sytému je však již paralelní běh modulů jednoznačně časově výhodnější. Samozřejmostí je, že implementace paralelního zpracování je o něco náročnější.



Obrázek 5.1

Pro paralelní běh modulů použít vlákna nebo jiné specializované knihovny

Pro vývoj paralelních aplikací existuje v současnosti celá řada možností. Nejprve bych se rád zmínil o vláknech (*threads*). Vlákna je v současnosti možno použít na všech moderních operačních systémech. Podpora vláken je většinou přímou součásti operačního systému, a proto takto navržené programy mohou pracovat velmi efektivně a jejich vývoj není nikterak náročný. *Multithread* programování předpokládá dělení programu na samostatné rutiny, nazývané vlákna, tak, aby tyto mohly běžet *konkurentně* v rámci jednoho procesu. To je v tomto případě snadno splnitelné, za podmínky, že hlavní funkce (*..._MAIN_MODULE_FUNC*) všech modulů budou napsány reentrantně. Specializované

knihovny pro vývoj paralelních aplikací jsou založeny právě na vláknech nebo využívají paralelně běžící procesy se sdílenou pamětí. Většina těchto knihoven pouze vytváří jakousi nadstavbu, v níž poskytuje větší uživatelský komfort pro používání např. vláken. Jelikož podpora paralelismu je ve většinou realizována pomocí sady speciálních funkcí, jedná se často pouze o objektovou reprezentaci využívající právě tyto funkce.

Distribuovaný systém nebo lokální s možností využití více procesorů

Zde se jedná o obdobnou, ne však zcela stejnou problematiku jako v případě modulů. Jak bylo již popsáno, moduly sami o sobě distribuované nejsou. Přesto by však bylo možno realizovat celý systém jako distribuovaný. Bylo by však nutné, aby runtime znal všechny používané datové typy, aby bylo možno všechna data přenášet mezi jednotlivými moduly. Nebo použít nějaký systém, který řeší sám jak distribuci výpočtu, tak i přenos dat mezi jednotlivými počítači. Takovýto systém již, podle mých informací, existuje, ale jeho pořízení je velmi finančně náročné.

5. 3 Zvolené řešení

Všechny moduly jsou uvnitř systému reprezentovány pomocí objektů, kde jednotlivé funkce náležící k modulů jsou mapovány na metody těchto objektů. Také pro celý runtime jsem se rozhodl použít objektově orientovanou kompozici, aby byla zachována co možná nejvyšší přehlednost a možnost rozšiřitelnost této části systému.

Pro realizaci této části (runtime) jsem se prozatím rozhodl, že se nebudu pokoušet o řešení v oblasti distribuovaných z výše popsaných důvodu, avšak tuto oblast bych do budoucna nerad uzavíral. Celý systém je navržen tak, aby jej bylo možno rozšířit o nové funkce (např. právě o přechod k distribuovanému sytému) co nejjednodušeji.

Systém umožňuje jak sekvenční, tak i paralelní spouštění navržených sítí modulů, které se nechá jednoduše přepínat pomocí uživatelského rozhraní. Vzhledem k tomu, že obě dvě možná řešení mají dle mého názoru své výhody, jsou v systému implementována obě. Sekvenční spouštění modulů je vhodné pro ladění modulů (je lépe vidět, kde nastala chyba),

případně pro práci na počítači s jedním procesorem. Pro zpracování výpočetně náročných sítí a pokud je k dispozici víceprocesorový systém, je vhodné používat paralelní model spouštění navrženého schématu.

Data, která během výpočtu již nejsou potřebná, mohou být okamžitě uvolňována, což by mělo umožňovat snadnější využití systému i na počítačích s menším množstvím paměti. Zde je nutno zmínit jednu drobnou nevýhodu. Vzhledem ke koncepci modulu (snaha o co nejjednodušší tvorbu modulu, pro autory modulů) je implementována pouze jedna funkce pro uvolňování výstupních dat modulu, tudíž pokud má modul více výstupů, nelze uvolňovat pouze ta data, která již nejsou potřeba, ale je potřeba počkat, až bude práce se všemi výstupními daty modulu ukončena a pak teprve je možno uvolnit všechna výstupní data modulu najednou.

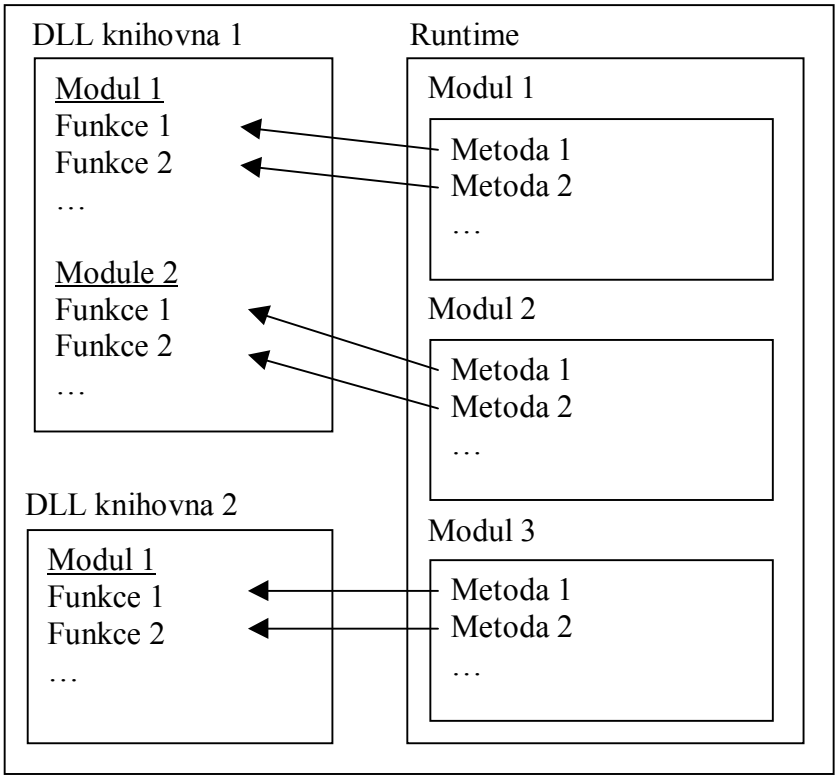
5. 4 Stručný nástin objektového návrhu a schéma spouštění modulů

Jak jsem se již zmínil, celý systém je objektově navržen. V této kapitole chci popsat, jak z tohoto pohledu vypadá runtime část a také podle jakého algoritmu jsou spuštěny moduly z uživatelem navrženého schématu. Podle následujícího popisu by mělo být jednoduché pochopit principy konstrukce a práce celého nově vytvořeného systému MVE.

Jelikož jsou moduly reprezentovány pomocí objektů, je potřeba vytvořit vazbu mezi těmito objekty a vlastními moduly, které jsou uloženy jako funkce v DLL knihovně. Tento problém je vyřešen pomocí namapování těchto funkcí na příslušné metody objektů (obrázek 5.2).

Celá runtime část je z globálního pohledu reprezentována pouze jedním objektem, který spravuje celý proces řízení výpočtu a předávání dat mezi moduly. Tento řídící objekt si je možno představit jako jakéhosi správce všech modulů (*ModuleManager*), které se nacházejí v daném okamžiku v systému.

V objektu správce modulů je také uložen seznam referencí na všechny moduly, které se nacházejí v aktuálním návrhu vizualizačního schématu (obrázek 5.3). Na této úrovni dochází také k dělení na sekvenční a paralelní část. Pro každou z obou variant existuje speciální správce modulů, který obsahuje metody jednou pro sekvenční a podruhé pro paralelní spouštění navrženého schématu.

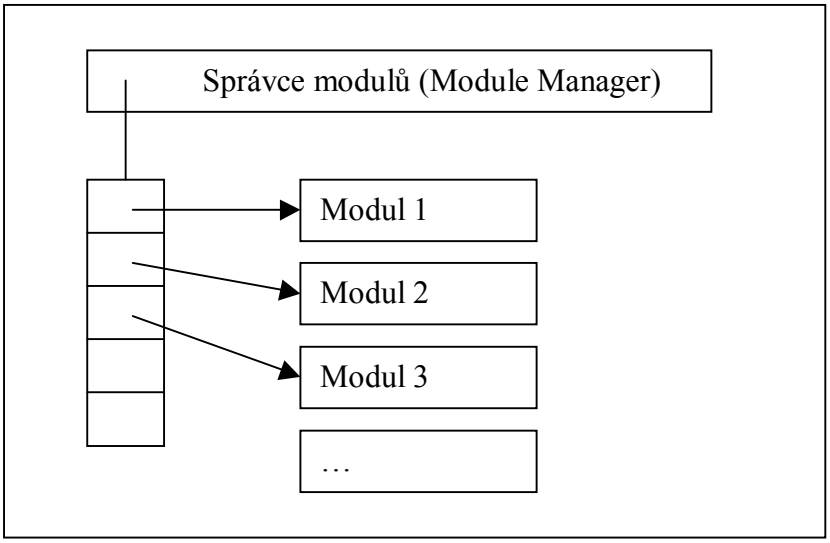


Obrázek 5.2

Toto řešení je výhodné, protože je možno oba správce modulů za běhu programu jednoduše zaměňovat (pomocí uživatelského nastavení systému) a tak dává uživateli jednoduché řešení jak si vybrat pro něj nejvhodnější způsob spouštění modulů. Také neuzavírá možnost případného dalšího rozšíření systému o jiné implementace řízení výpočtu (např. možnost práce s moduly distribuovanými na síti). Při ukládání navrženého schématu se samozřejmě také uloží informace o tom, jaký algoritmus pro spouštění navrženého schématu se má použít.

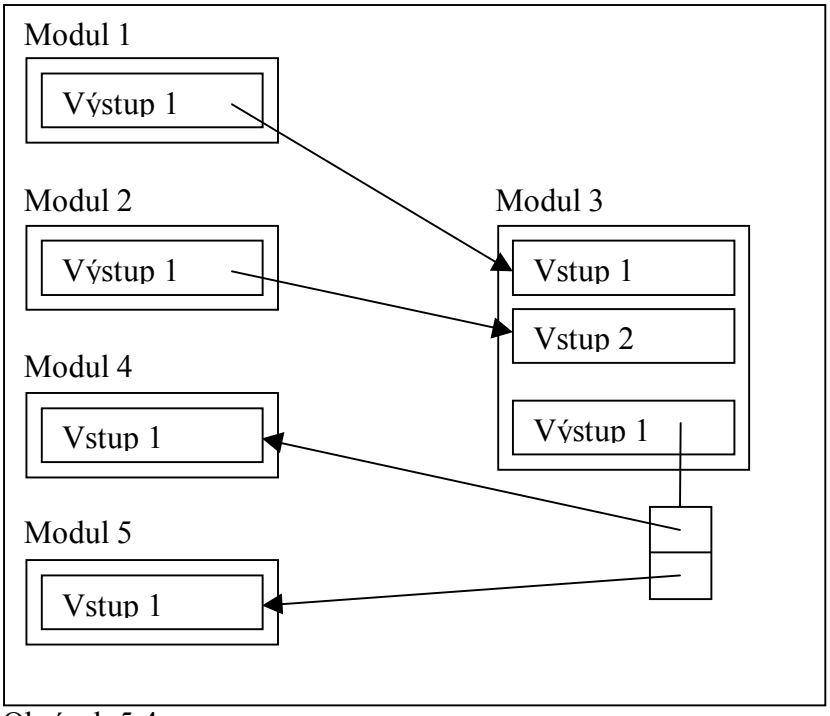
V runtime vrstvě je také uloženo uživatelské nastavení modulů. Toto nastavení je sice realizováno pomocí uživatelem definované funkce (nějakého dialogu) v návrhovém systému, ale po ukončení této funkce přebere data s nastavením modulu (musí se vždy jednat o konzistentní blok dat) řídící systém a při spuštění vizualizačního schématu je znovu předá jednotlivým modulů.

Pomocí tohoto postupu je nyní možno uchovávat všechna nastavení jednotlivých modulů společně s informací popisující moduly a jejich vzájemné propojení.



Obrázek 5.3

Informace o vazbách (propojení modulů vytvořené pomocí návrhového systému) je uložena na úrovni modulů a realizována pomocí objektů popisujících vstup a výstup modulu (obrázek 5.4). Vazby jsou vytvářeny na úrovni návrhového systému, avšak informace o nich je uložena také v nižší úrovni tj. runtime.



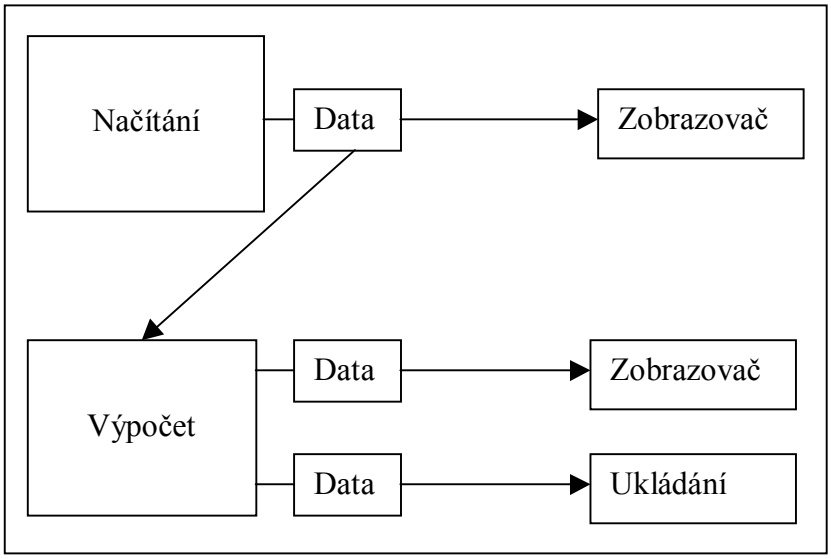
Obrázek 5.4

Přenos dat mezi moduly

Vzhledem k charakteru systému je přenos dat realizován poměrně jednoduše, jedná se vlastně pouze o předávání ukazatelů na data. Ke každému výstupu libovolného modulu je možno navázat právě jeden datový objekt (obrázek 5.5). Z teoretického hlediska by se jevilo jako výhodnější, použít pro každý výstup modulu seznam dat, ale pro praktické použití, obzvláště v případě velkých objemů dat, nemá toto řešení až takový význam.

Mohlo by totiž jednoduše docházet k zahlcení systému moduly, které by produkovaly na svém výstupu data rychleji, než je ostatní moduly stačí zpracovávat. Z dosavadních zkušeností je patrné, že jedna datová schránka navázaná k výstupu modulu, pro drtivou většinu případů, bohatě postačuje.

Data jsou vázána vždy k modulu, který je vygeneroval, jelikož k němu patří jak logicky, tak i fyzicky (vzhledem k implementaci DLL knihoven pod operačním systémem MS Windows).



Obrázek 5.5

Algoritmus spouštění modulů

Algoritmus spouštění modulů je poměrně jednoduchý a v některých částech je stejný pro sekvenční i paralelní část. Celý postup se skládá ze dvou hlavních kroků.

Nejprve jsou vybrány moduly, které je možno spustit, tj. moduly, které buď nemají žádný datový vstup (v prvním kroku hned po spuštění navrženého schématu), nebo mají již na všech svých vstupech připravena data. Modul musí mít také volné datové schránky na všech svých datových výstupech (schránka musí být prázdná, nebo k ní náležící data musí být označena jako zpracovaná). Tyto moduly jsou zařazeny do speciálního seznamu.

Druhá část algoritmu se již pro sekvenční a paralelní část liší. Při sekvenčním spouštění jsou moduly ze seznamu vybírány postupně (jedná se o frontu typu FIFO) a jsou spouštěny, po ukončení každého modulu se testuje zda s vygenerováním nových dat není možno spustit nějaký další modul, pokud ano, je tento zařazen to výše zmíněného seznamu.

Při paralelním spouštění jsou všechny moduly ze seznamu modulů, připravených ke spuštění, spuštěny najednou pomocí vláken (viz programová dokumentace nebo přílohy „Vlákna a MS Windows“ na CD. Následně systém čeká na ukončení libovolného modulu a poté znovu najde spustitelné moduly a všechny je spustí.

Krok dvě se pro obě metody neustále opakuje, dokud se v systému nacházejí moduly, které je možno spustit. Ukončení výpočtu (běhu navrženého schématu) nastane v okamžiku, kdy již neběží žádný modul (vyjma zobrazovacích modulů) a žádný další modul již není možno spustit.

6 Návrh sítě modulů – Editor

Postup zpracovávání dat se popisuje ve formě obrazu, do něhož se umístí symboly algoritmů (moduly). K tomuto má sloužit speciální editor, či návrhový program pro vytváření sítí modulů viz obrázky z již existujících komerčních systémů v kapitole 3.2.

Tento program musí umožňovat přidávání jednotlivých modulů na pracovní plochu a napojování vstupů a výstupů modulů. S tím souvisí nutnost uchovávat informace o modulech, které jsou pro návrh použitelné. Návrhový systém by měl také umožňovat přímé spouštění navržené sítě modulů, aby si uživatel mohl ihned ověřit a zkontrolovat správnost výstupů, či zareagovat na případné chyby a nedostatky.

6. 1 Problémy, které je potřeba řešit

Grafický systém pro editor

Zde se jedná o otázku, pomocí jakého grafického systému (grafické knihovny) realizovat grafickou reprezentaci modulů v návrhovém systému. Je potřeba zvolit takové řešení, aby implementace nebyla z programátorského hlediska příliš náročná a z pohledu uživatele bylo celé prostředí přehledné a snadno pochopitelné. Další vlastností musí být dostatečná rychlost vybrané grafické knihovny, i když pokud vezmeme v úvahu výkonnost dnešních počítačů, tak se tento problém posouvá do pozadí.

Uživatelsky přívětivé prostředí pro začátečníky

Proces návrhu sítě modulů musí být co možná nejjednodušší a pokud možno interaktivní. Nezkušený uživatel by měl být po krátkém zaškolení schopen se systémem sám pracovat, vkládat na pracovní plochu existující moduly a jejich spojováním vytvářet jednoduchá výpočetní schémata. Významy všech funkcí systému by měly být pokud možno jasně zřetelné z jejich grafické reprezentace.

Co nejméně omezení pro pokročilé uživatele

Přestože systém má být snadno použitelný pro začátečníky, musí také dávat dostatek prostoru pro pokročilé uživatele. Proces návrhu modulární aplikace by měl být co možná nejpřímočařejší bez často zbytečných pokusů o nápovědu v místech, kde není vůbec potřeba. Systém by také měl umožňovat snadno měnit uživatelské nastavení samotného editoru, případně jednoduché ovládání pomocí klávesových zkratek.

Přehlednost návrhu i pro složité sítě modulů

V případě systémů, které obsahují velký počet modulů jejichž funkce jsou silně specializované (více se podobají vyššímu programovacímu jazyku) může docházet k tomu, že práce na složitějším vizualizačním schématu se stane značně nepřehlednou vzhledem k velkému počtu modulů a vazeb mezi moduly, které se v návrhu vyskytují. Proto je potřeba vyvarovat se přílišnému rozdrobování jednotlivých modulů.

Grafická reprezentace modulů

Jednotlivé moduly je na pracovní ploše editoru potřeba zobrazovat pomocí vhodné grafické reprezentace. Z grafické reprezentace modulu by mělo být jasné o jaký modul se jedná, jaké, případně jakého datového typu, jsou jeho vstupy a výstupy. Přímo pomocí objektů reprezentujících moduly by mělo být možno provést nastavení modulu (např. vyvolání dialogu pro nastavení modulu). S předchozím blokem také souvisí otázka množství informací, které je potřeba mít u modulu zobrazeny. Čím více informací o modulu je zobrazeno na pracovní ploše, tím musí být také větší grafický objekt reprezentující modul a tím méně modulů je poté možno na pracovní ploše editoru zobrazit.

Realizace uživatelského nastavení modulu uvnitř editoru

Uživatelské nastavení modulů je nutno provádět v rámci návrhového systému a toto nastavení by mělo být možno uchovat společně s uložením informace o modulech, které jsou ve vizualizačním schématu a napojení jejich vstupů a výstupů. Nastavení modulu je pravděpodobně nejlépe provést pomocí speciálního dialogu, který je bud přímo součástí modulu a nebo je dynamicky generován návrhovým systémem podle parametrů modulu.

Realizace a vizualizace napojování modulů

Napojování datových vstupů a výstupů modulů by mělo být pokud možno interaktivní a přehledné. Tuto problematiku řeší skoro každý z existujících systému vlastním způsobem od jednoduchého využití funkce drag&drop (přetažení vstupu/výstupu jednoho modulu nad vstup/výstup druhého modulu) až po napojování modulů pomocí speciálního menu (viz IRIS Explorer).

6. 2 Možné způsoby realizace

Jako základní grafický systém pro návrhový systém použít WinAPI nebo OpenGL

Co se týká implementace návrhového systému připadají v úvahu dvě možné varianty, a to využití WinApi, obecně libovolné grafické prostředí, které je součástí cílového OS, nebo OpenGL, tj. grafickou knihovnu, která je nezávislá na platformě (operačním systému) pro něž je výsledný systém určen.

Pro zobrazení modulů použít nebo nepoužít standardní objekty WinAPI nebo vlastní

Při použití knihovních funkcí operačního systému je možno pracovní plochu editoru (tj. grafickou reprezentaci všech modulů a vazeb mezi nimi) realizovat dvěma rozdílnými

přístupy. A to využít prostředků, které daný OS poskytuje a vlastně poskládat všechny potřebné grafické objekty z dostupných součástí (tlačítka, listbox, ...), které mají již implementovánu vrstvu pro interakci s uživatelem, a nebo navrhnout a implementovat vlastní grafické objekty.

Ukládaní informací o modulech do speciální samostatné databáze

Jak je z filosofie tohoto systému patrné, návrhový systém musí uchovávat informace (jméno a popis modulu, počet a datové typy vstupů a výstupů) o dostupných modulech. Z nich pak má uživatel možnost vybírat moduly pro svůj vlastní návrh. Tuto problematiku je možno řešit registrací modulů ve speciálním datovém souboru, případně externí databázi do níž editor poté přistupuje pomocí definovaného rozhraní. Výhodou tohoto řešení je transparentní uložení informace o modulech, tyto informace je poté možno dále jednoduše zpracovávat nebo třídit podle vybraných kritérií. Tento přístup je vhodný v případě rozsáhlých sad modulů se specifickými požadavky na přístup k informacím o modulech. Nevýhodou je oddělené uložení informace, což má za následek, že v případě ztráty nebo neexistence datového záznamu (souboru,...) o modulu je celá výkonná část modulu nepoužitelná, jelikož systém neví, jak s ní zacházet.

Svázání informace o modulu přímo s modulem (tj. s výkonnou částí modulu)

Druhou variantou výše popsaného přístupu je možnost uložení informace o modulu společně s výkonnou částí modulů (přímo do souboru s kódem modulů, realizace např. pomocí speciální informační funkce). V tomto případě je náročnější získávání a zpracovávání této informace, ale výhodou je právě ona těsná vazba mezi samotným modulem a datovým záznamem o něm. Toto řešení je vhodné pro systémy, které jsou stále ve vývoji, tj. moduly se často mění a neustálé obnovování databáze s informacemi by bylo příliš náročné. Odpadá také problém ztráty (neznalosti) informace o modulu, která je takto vždy k dispozici.

Ukládání navržené sítě modulů do souboru v podobě scriptu nebo binárně.

Otázka způsobu ukládání navržené sítě modulů je poměrně důležitá. Zde jsou možné vlastně pouze dva zcela odlišné přístupy. První je možnost ukládání celého návrhu pomocí speciálního scriptovacího jazyka, tato varianta je velmi výhodná z pohledu koncového uživatele, jemuž dává možnost ručně zasahovat do již hotových schémat a případně měnit některé parametry, bez nutnosti spouštění celého návrhového systému. Implementace ukládání pomocí ať již existujícího nebo zcela nového scriptovacího jazyka je velmi náročná (je potřeba řešit syntaktickou a sémantickou analýzu vybraného jazyka). Binární forma ukládání navržené sítě modulů se liší především právě nemožností explicitní editace výstupních dat návrhového systému. Implementace tohoto přístupu je však mnohem jednodušší.

6. 3 Zvolené řešení

Editor považuji za velice důležitou součást celého systému, jelikož reprezentuje rozhraní mezi samotným MVE na jedné a uživatelem na druhé straně. A vlastně rozhoduje o tom, jaký si na systém udělá každý uživatel první (často rozhodující) názor. Proto je editor navržen tak, aby působil pokud možno jednoduše a nekomplikovaně, ale zároveň aby poskytoval uživateli všechny možnosti a funkce, které jsou potřeba pro rychlý a efektivní proces návrhu vizualizačního schématu.

Aby byl samotný proces navrhování (rozložení a pospojování modulů) co možná nejjednodušší a přímočarý, je mnoho činností v editoru realizováno pomocí funkce drag&drop (táhni a pusť) pomocí myši (k ovládání systému je myš nutná). Na pracovní ploše jsou pro úsporu místa a zpřehlednění návrhu zobrazovány jen nejnutnější informace, ale například stručné popisy modulů, případně jejich vstupů a výstupů, je možno zobrazovat pomocí ToolTipů (malé okénko s textovou informací, které se zobrazí pokud přidržíme kurzor myši nad daným místem na obrazovce).

Proces nastavování parametrů runtime (např. zda se má navržené schéma spouštět paralelně nebo sekvenčně) je v editoru realizován pomocí speciálních dialogů. Stejně tak je řešeno přidávání nových knihoven s moduly do systému.

Pro implementaci celé vizuální části návrhového systémů jsem se rozhodoval mezi OpenGL a využitím funkcí WinAPI. Nakonec jsem se rozhodl použít WinAPI, protože má všechny potřebné vlastnosti a je snadno použitelné, výhody OpenGL (např. nezávislost implementace na operačním systému) by nebylo možno stejně většinou využít a proces vývoje editoru by se stal značně složitějším.

Uložení a získání informací o dostupných modulech

Aby bylo možno pomocí editoru navrhovat vizualizační schémata, je potřeba mu nejprve předat informaci o tam, jaké moduly jsou právě k dispozici. Jak již bylo popsáno, tuto problematiku lze řešit dvěma různými způsoby. Pro tento systém je využita varianta s ukládáním informace o modulech přímo do knihovny obsahující samotné moduly, což je v tomto případě výhodnější (existuje malé množství modulů a systém je stále ve vývoji, to má za následek časté změny v oblasti modulů, a takto odpadá nutnost neustálého aktualizování databáze modulů). Vybrané řešení je realizováno pomocí speciální funkce(í) (viz tabulka 6.1), která musí být v každé DLL knihovně obsahující moduly. Tato funkce vrací speciální strukturu obsahující popis všech modulů (jméno a typ modulu, počet a typy jednotlivých datových vstupů a výstupů (viz tabulka 4.2 a tabulka 4.3)), které jsou v dané DLL knihovně obsaženy. Tvůrce modulů musí zajistit, aby tato informace byla korektní, jinak samozřejmě takový modul nemůže pracovat správně. Podle této informace je také vytvářena grafická reprezentace modulu na pracovní ploše editoru.

Přehled funkcí pro získání informací o modulech uložených v DLL knihovně
Get_Modules – Tato funkce je v DLL knihovně pouze jedna a slouží k tomu, aby se editor po zaregistrování této DLL knihovny dozvěděl jaké moduly jsou v ní obsaženy.
Free_DLL_Descr – Tato funkce slouží pouze pro uvolnění dat, která alokovala funkce *Get_Modules.*

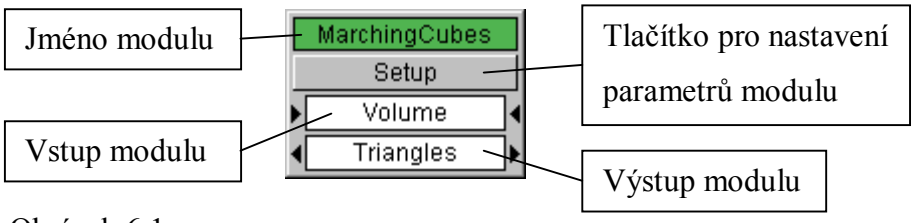
Tabulka 6.1

Grafická reprezentace modulů

Pro grafickou reprezentaci modulů jsem zvolil takové řešení, které umožňuje zobrazit všechny potřebné informace v přehledné a kompaktní podobě (obrázek 6.1), tak aby bylo na první pohled vidět vše, co je potřeba. V tomto případě se jedná o jméno modulu, které by mělo jasně vyjadřovat jeho funkci, počet a typ jednotlivých vstupů a výstupů modulů (typ dat je potřeba vidět, aby bylo jasné, které vstupy a výstupy je možno spojit). Poslední součástí obrázku reprezentujícího libovolný modul je tlačítko umožňující vyvolání uživatelem definované funkce pro nastavení parametrů modulu. Obraz reprezentující modul je řešen tak, aby bylo pokud možno každému hned jasné o co se jedná.

Přestože jsou všechny grafické objekty (tlačítko, nadpis, seznam řetězců) na obrázku nově vytvořené, snažil jsem se, aby se podobaly standardním objektům z MS Windows a tak na uživatele působily známým dojmem. Každopádně by neměl být žádný problém, v případě potřeby, grafickou reprezentaci jednotlivých částí či dokonce celého modulu jednoduše změnit.

Velikost jednoho grafického objektu modulu jsem se snažil zvolit tak, aby se na pracovní plochu editoru vešlo, při rozumném rozlišení, dostatek modulů pro středně komplikovanou síť, při zachování maxima zobrazené informace.



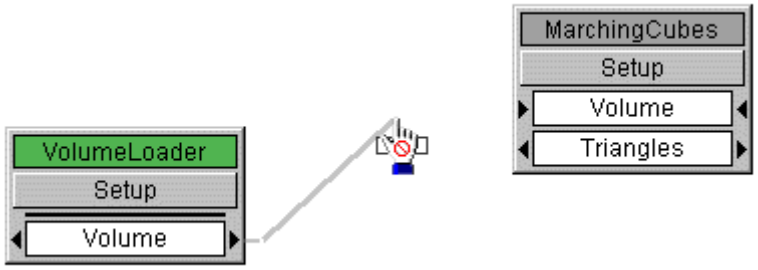
Obrázek 6.1

Proces napojování modulů

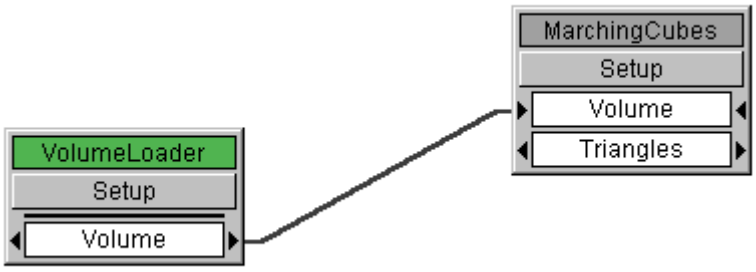
V procesu napojování modulů se jedná o spojování datových výstupů jedněch modulů na datové vstupy druhých. Tato operace je implementována do systému tak, aby byla maximálně jednoduchá a rychle proveditelná. Je zde využita funkce drag&drop, pro napojení dvou modulů stačí chytit pomocí myši vstup/výstup jednoho modulu a přetáhnout ho na výstup/vstup druhého modulu (obrázek 6.2 a obrázek 6.3). Systém je v tomto ohledu

interaktivní a nedovolí uživateli provést chybné napojení modulů, tj. nelze napojit výstup a vstup dvou různých datových typů, stejně tak nelze napojit více výstupů na jeden vstup, samozřejmě je možné jeden výstup napojit na více vstupů.

Zde je také nutné zmínit se o jedné věci, kterou tento systém neřeší, a to je možnost navržení chybného výpočetního schématu. Jedná se o to, že v editoru je možno vytvořit z modulů smyčku, která samozřejmě nemůže pracovat správně, u složitějšího návrhu může uživatel vytvořit smyčku třeba i náhodou nebo drobnou chybou. Takovéto schéma samozřejmě nebude funkční (jsou očekávána data, která zatím nemohla být vygenerována). Při spuštění takového návrhu proběhne výpočet až k začátku smyčky, kde se zastaví. Každopádně bych rád přidal do některé z následujících verzí systému možnost detekce výskytu smyček již při procesu návrhu sítě modulů.



Obrázek 6.2



Obrázek 6.3

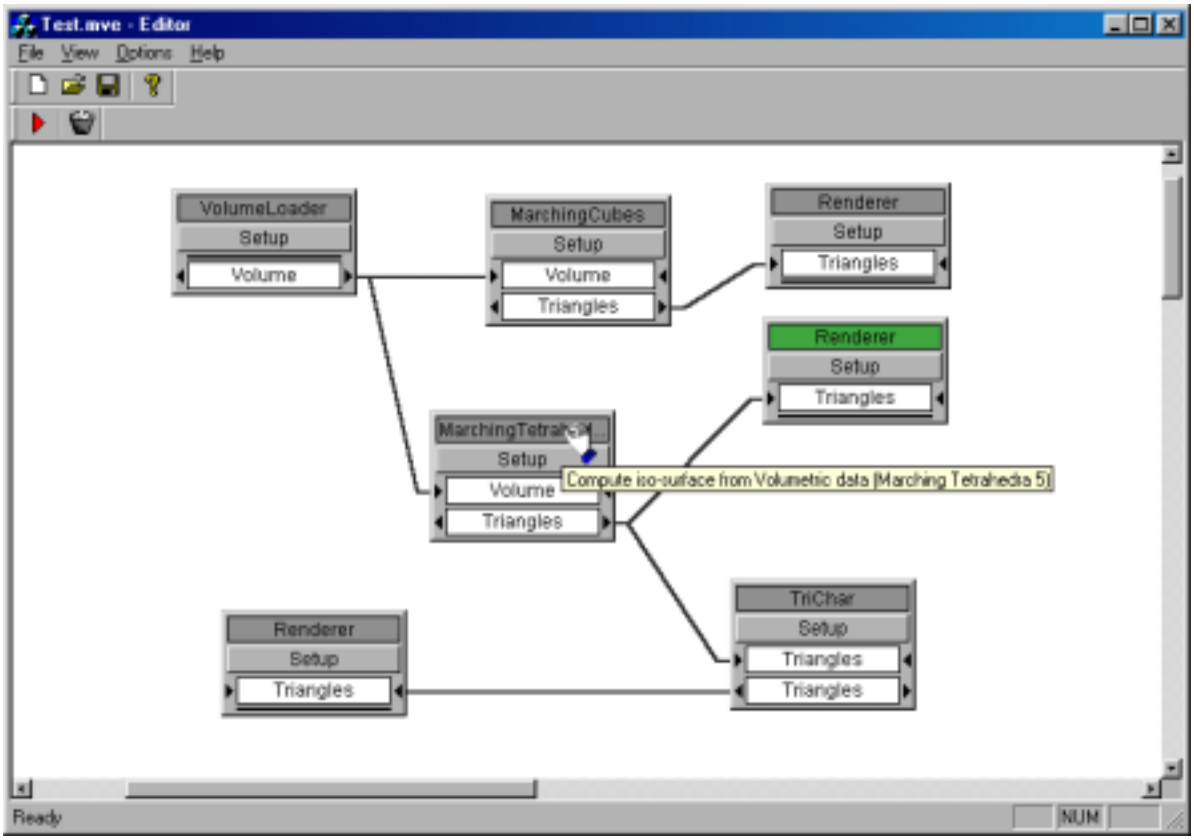
Ukládání navrženého vizualizačního schématu

Pro ukládání navržené sítě modulů jsem zvolil binární formu. Ukládání pomocí speciálního *scriptu* by mělo z uživatelského hlediska určitě přednosti, avšak jak je popsáno

výše, jeho implementace je nesrovnatelně náročnější. Vzhledem k použití dvou vrstev systému (editor, runtime) je umožněno ukládat navržené schéma buď jako kompletní návrh, tudíž včetně veškeré informace o rozložení modulů na ploše editoru, a nebo uložit pouze data z nižší vrstvi (runtime), které obsahují všechny potřebné informace pro spuštění celého schématu. Tento postup při řešení dává možnost spouštět navržené sítě modulů pomocí speciálního samostatného programu např. přímo z příkazové řádky. Na tomto programu prozatím pouze pracuji a tak není součástí této diplomové práce.

Zobrazování dodatečných informací o modulech

I přesto, že v obrazu modulu na pracovní ploše je oproti ostatním systémům zobrazeno poměrně hodně informací, nelze zobrazit vše potřebné. Například jméno modulu nemůže vždy úplně vystihnout účel modulu a stejně tak pouhé jméno datového typu na vstupu, či výstupu modulu nedává uživateli příliš mnoho informace o co konkrétně se jedná. Proto editor umožňuje zobrazení další informace o modulu pomocí ToolTips (viz text výše a obrázek 6.4). Jako dodatečnou informaci, zobrazitelnou pomocí této metody, je možno přidat k modulu jeho stručný popis a taktéž popis ke každému vstupu, či výstupu. K modulu je také možno přidružit jednoduchou nápovědu pomocí nepovinné funkce *ModuleName_HELP_FUNC*, text vrácený touto funkcí je možno jednoduše zobrazit v rámci editoru (viz uživatelská dokumentace k editoru).



Obrázek 6.4

7 Do budoucna

Byl bych nerad, kdyby obhajobou této práce skončil vývoj výše popsaného systému. Samozřejmě je stále co vylepšovat a doplňovat. V tomto oddílu bych se rád zmínil alespoň o některých možnostech, které připadají v úvahu pro následující období.

Přechod k distribuovanému systému

Celý systém je navržen tak, aby bylo možno pokud to situace dovolí (čas a nutné vybavení) přejít do oblasti distribuovaných systémů. Jedná se o dvě možná řešení a to přechod k plně distribuovanému systému, kde by přenos dat byl úplně nebo částečně realizován již na úrovni modulů (např. realizace modulů nad standardem CORBA a využití *marshalingu* pro přenos dat). Toto řešení by bylo pravděpodobně efektivnější a poskytovalo by větší možnosti, na druhou stranu by kladlo značné nároky na znalosti autorů jednotlivých modulů. Druhá

varianta by předpokládala realizaci modulů v podobném duchu jako doposud, avšak řídící sytém by musel být schopen sám transportovat všechna potřebná data po síti. Možnosti takového systému by byli částečně omezené, ale vývoj modulů by byl podstatně jednodušší než v předchozím případě.

Možnost ovládání pres WWW ?

Za použití standardu CORBA (JAVA) pro vývoj distribuovaného systému by bylo možno realizovat uživatelské rozhraní na úrovni internetových technologií, např. implementovat návrhový systém jako *aplet*, který by komunikoval s moduly a případně je řídil na úrovni distribuovaných objektů architektury CORBA. Realizace takového rozhraní obecně nepotřebuje objektově orientovanou distribuovanou architekturu a tak by bylo nejspíše možné vytvořit rozhraní pro WWW i s jednodušším systémem, každopádně by se však muselo jednat o systém distribuovaný.

Zlepšení interakce s uživatelem během procesu výpočtu.

Systém by bylo vhodné doplnit o moduly pro řízení výpočtu (podmínky, smyčky ...) a doladit implementaci v této části, která není dokonalá, jelikož prozatím nic takového nebylo potřeba využívat. Systém by také mohl umožňovat výstup podrobných informací o průběhu spouštění navržené sítě modulů.

V současném stádiu vývoje systému není možno interaktivně měnit nastavení modulu a zároveň automaticky obnovovat zobrazená data. Modul je potřeba nejprve nastavit a poté spustit. Do budoucna bych chtěl implementovat jednoduchý interface (pravděpodobně za využití systému zpráv pod MS Windows), který by umožňoval volnější komunikaci mezi samotným systémem a jednotlivými moduly.

Druhou variantou, ale poměrně implementačně náročnou, by bylo vytvoření podobného Generátoru modulů, jaký je obsažen ve většině komerčních systémů a dialogy pro nastavování parametrů modulů poté vytvářet automaticky.

8 Závěr a zhodnocení

Výše popsaný systém splňuje všechny požadavky zadání. Každému musí být jasné, že co se možností a celkové propracovanosti systému týká, nemůže se tento systém měřit s komerčními produkty v oblasti MVEs. Jeho předností by však měla být jednoduchost a flexibilita. Tento MVE systém by měl být použitelný pro libovolnou úlohu využívající modulární dekompozice, tudíž nejen v oblasti počítačové grafiky, pro niž byl vyvíjen.

Proces návrhu a implementace jednotlivých algoritmů, jako modulů, je jednoduchý a dle mého názoru zvládnutelný (za pomoci uživatelské dokumentace) i pro nepříliš zkušené uživatele. Stejně tak návrh sítě modulů pomocí editoru je poměrně jednoduchý a přímočarý. Pro práci na víceprocesorových strojích dává tento MVE možnost paralelního spouštění navržených vizualizačních schémat.

Co se týká požadavku na vytvoření obecné sady modulů pro ověření funkčnosti systému, nebylo nutno je vytvářet, jelikož v té době již existovaly moduly od jiných studentů či pracovníků ZČU. Tak jsem mohl pro ověření funkčnosti použít právě tyto moduly a věnovat více času vývoji samotného systému.

9 Použitá literatura

[1] Stead,G.A.: Simplifying the Visualization Process, The University of Leeds, 1995

[2] Skala,V.&others: Computer Graphics and Visualization in Parallel and Distributed Environment, Technical Report, Computer Sci.Dept., Univ.of West Bohemia, Plzen, 1999

[3] Chen,M., Kaufmann,A.E., Yagel,R.(Ed.): Volume Graphics, Springer Verlag, 2000

[4] Skala,V.(Ed.): WSCG'99 Int.Conf. on Computer Graphics, Visualization and Interactive Digital Media'99, Plzen, 1999

[5] Jiří Žára, Bedřich Beneš, Petr Felkel: Moderní počítačová grafika, Computer Press, Praha, 1998

10 Přílohy

Programátorská dokumentace k modulů, popis tvorby modulů a uživatelské dokumentace k editoru jsou v samostatné příloze.

10. 1 Testy

Pro otestování systému jak při sekvenčním, tak také při paralelním běhu jsem využil moduly pro práci s volumetrickými daty a trojúhelníkovou sítí od Marka Krejzi. Testy byly prováděny na počítači DELL 2x PIII 500 Mhz, 1GB RAM.

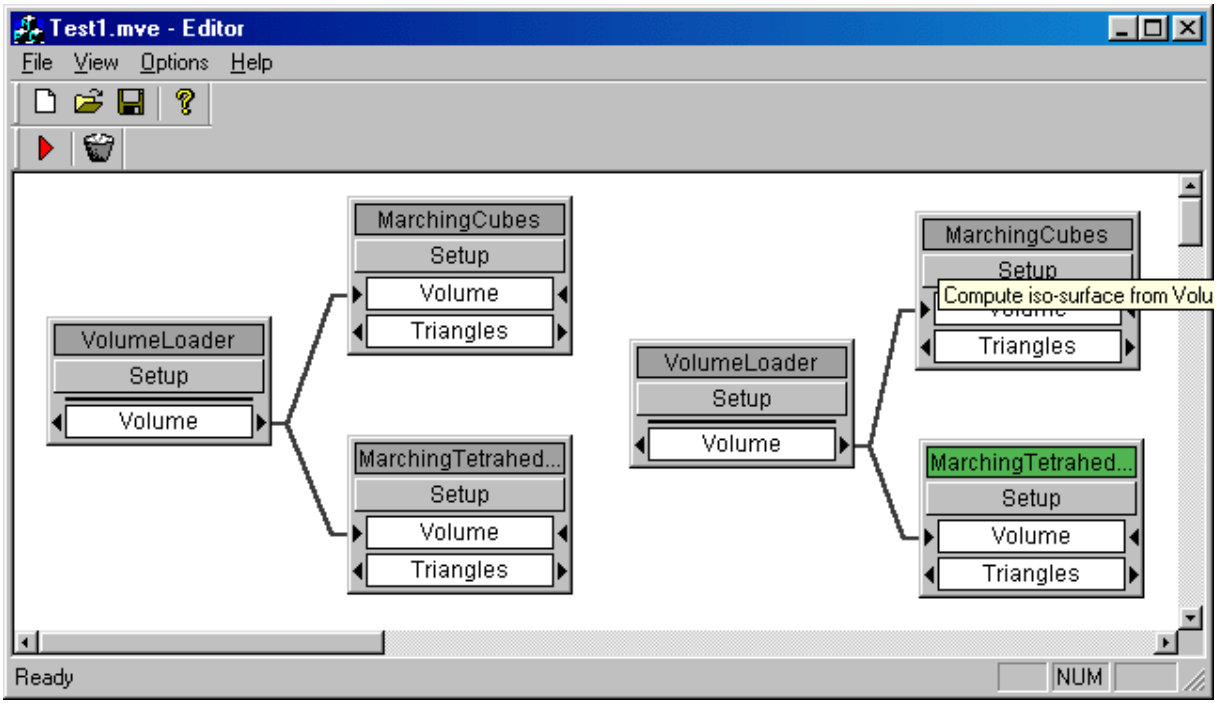
Sekvenční běh	**Paralelní běh**
14:726	11:828
14:703	11:797
14:705	11:844

Tabulka 10.1 (Byla zpracovávána volumetrická data syn_64 a CTHead (obrázek 10.1))

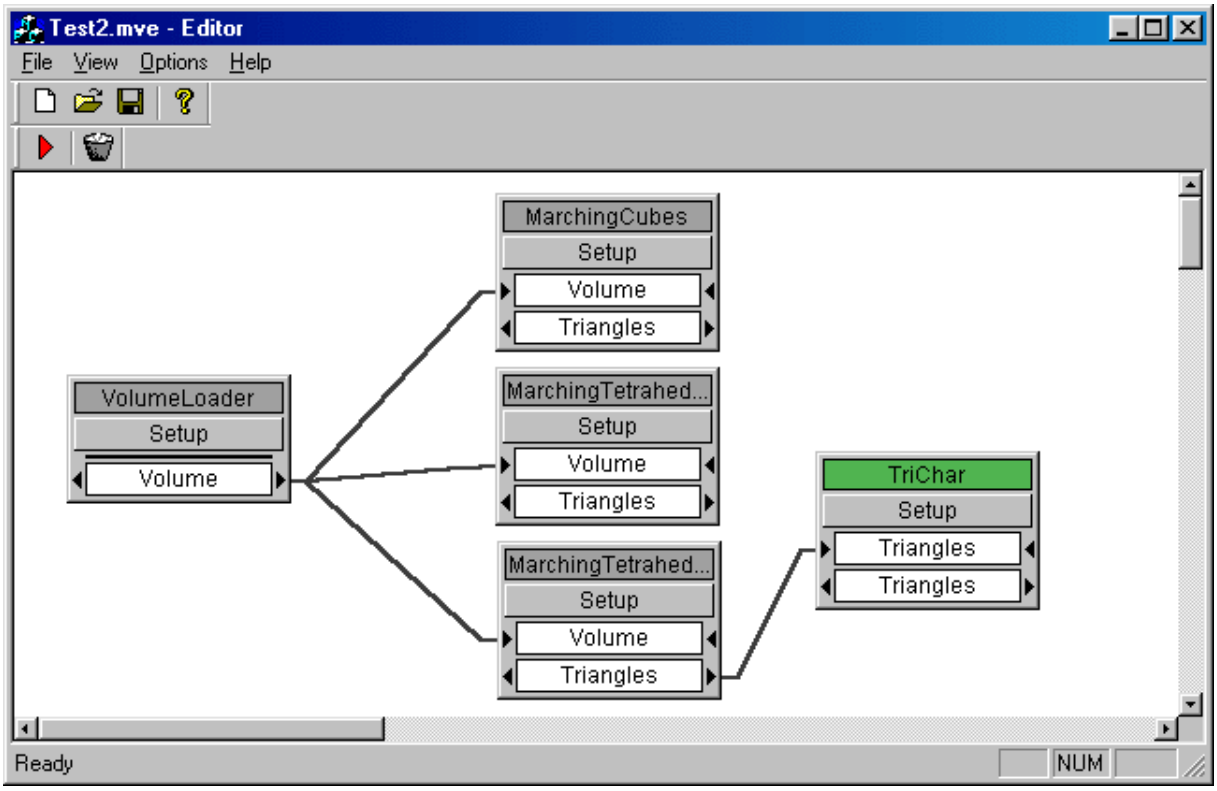
Sekvenční běh	**Paralelní běh**
2:532	1:906
2:453	1:938
2:500	1:937

Tabulka (Byla zpracovávána volumetrická data syn_64 (obrázek 10.2))

10. 2 Obrázky



Obrázek 10.1



Obrázek 10.2