Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Grafické rozhraní pro volně šiřitelné astrofyzikální numerické kódy

Bakalářská práce

Zdeněk Melkes

Školitel: RNDr. Petr Jelínek, Ph.D.

České Budějovice 2011

Anotace

Tématem této bakalářské práce je vytvoření vhodného grafického rozhraní (GUI) mezi uživatelem a volně šiřitelným programem pro astrofyzikální výpočty, pro jeho jednodušší a pohodlnější ovládání. Jde o zhotovení takového rozhraní, kde uživatel zvolí vstupní textové soubory, zadá parametry a spustí výpočet. Úvodní část nás seznamuje s použitím numerických kódu v astrofyzice a problematice magnetohydrodynamických rovnic (MHD). Následuje část věnována programu Athena a grafickému uživatelskému rozhraní, kde je popsána instalace programu, jeho struktura, ovládání a rozvržení jednotlivých komponent, včetně příkladu spuštění testovacích úloh.

Abstract

The topic of this bachelor's thesis is creating a suitable graphical interface (GUI) between a user and a freeware program for astrophysical computation, for its simpler and more comfortable use. My goal is to create such an interface where a user selects input text files, sets parameters and starts computation. The prefatory part introduces numeric codes used in astrophysics and the issue of magnethydrodynamics equations (MHD). The following part is about the program Athena and a graphical user interface. There is a description of installation of the program, its structure, control, layout of individual components, including a sample of the start test function.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 15.12.2011 ……..

Zdeněk Melkes

Poděkování

Rád bych zde poděkoval vedoucímu své bakalářské práce RNDr. Petru Jelínkovi, Ph.D. za cenné rady a připomínky při zpracování mé práce.

Obsah

1 Astrofyzikální numerické kódy

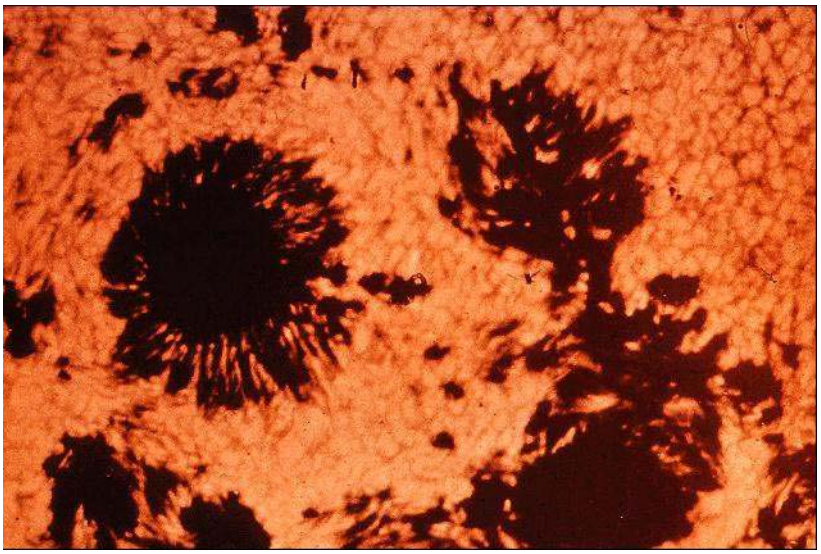
Řešení fyzikálních problémů na velkorozměrových kosmologických škálách nebo v mikrosvětě vyžaduje enormní množství výpočtů. Zpracování rozsáhlých souborů astronomických observačních dat klade velké nároky na výpočetní výkon, paralelizaci výpočtů MPI a počítačové implementace numerických metod, statistického zpracování a fyzikálních simulací.

V dnešní době existuje velké množství numerických kódů pro různé astrofyzikální simulace. Vytvoření programu pro řešení těchto úloh není jednoduchou záležitostí, ať již z časového hlediska, nebo ze strany znalostí řešení daného problému, či programování. Z těchto důvodu postupně vznikly specializované weby, které jsou otevřeným společenstvím pro uživatele a vývojáře astrofyzikálních simulačních kódů. Pro příklad zde uvádím:

Athena	SW určen pro výpočet MHD založen na Godunovových algoritmech [6]
Enzo	SW využívající metodu AMR na bázi hybridních kódu pro kosmologické struktury
Flash	modulární, přizpůsobivý, paralelní simulační kód, který je schopný zvládnout obecné problémy stlačitelného proudění v astrofyzikálních prostředích [14]
Nirvana	univerzální C kód pro astrofyzikální výzkum, který numericky integruje 2D/3D rovnice závislé na čase, nerelativistické, stlačitelné (compresssible) magnetohydrodynamiky na kartézské, válcové a sférické sítě.
Ramses	paralelní adaptivní AMR kód pro astrofyzikální dynamiku kapalin
VAC	software, na kterém je založen další kód používaný ve sluneční fyzice skupinou R. Erdélyiho ze Sheffieldu UK (SAC - Sheffield Advanced Code)
Zeus	různé numerické kódy pro astrofyzikální dynamiku plynů ve 2D a 3D

2 Magnetohydrodynamika (MHD)

Podle odhadů je více jak 99 % standardní hmoty v kosmu ve formě plazmatu. Hvězdy včetně našeho Slunce jsou také z převážné části tvořeny plazmatem. Vzájemné působení plazmatu a magnetického pole Slunce nám dovoluje sledovat některé zajímavé jevy, jako jsou, sluneční erupce, sluneční vítr, koronární smyčky.



Obrázek 1: Sluneční skvrny.

Pochopení dějů, které probíhají na Slunci nám umožňuje nejen zjistit vliv Slunce na naší sluneční soustavu, ale zároveň pochopit jevy, které probíhají ve vzdáleném vesmíru. Jednou z metod používaných k popisu a modelování chování sluneční atmosféry je magnetohydrodynamika (dále jen MHD). Tato disciplína studuje pohyb elektricky vodivého prostředí v magnetickém poli. Jedná se o syntézu hydrodynamiky a teorie elektromagnetického pole. Magnetické pole mění pohyb vodivé tekutiny (nabitých částic) a ten zpětně ovlivňuje výsledné magnetické pole. Sluneční MHD se snaží popsat jednotným a co nejpřesnějším způsobem vzájemné interakce mezi slunečním magnetickým polem, plazmatem a sluneční atmosférou, aby se dala ve výpočtech použít jako kontinuální médium. Základy MHD byly položeny švédským fyzikem Hannesem Alfvénem, který popsal třídy vln MHD, a za tyto a další práce v oboru sluneční fyziky obdržel v roce 1970 Nobelovu cenu. Mezi jeho hlavními objevy patří především rovnice, které popisují chování plazmatu v podmínkách sluneční atmosféry.

Obrázek 2: Hannes Alfvén.

2.1 Teorie MHD dynama

Pro generování magnetického pole v rotujících hvězdách nebo planetách musí být splněno několik podmínek. Prostředí musí být:

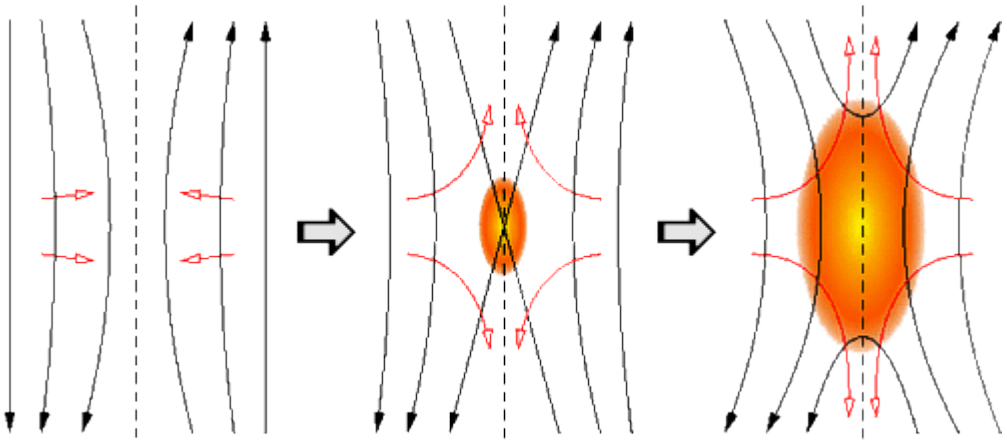
- vodivé
- tekuté (aby se mohla uplatnit diferenciální rotace, kdy se vrstvy v různých sférách a rovnoběžných vrstvách otáčí různou rychlostí)
- musí docházet k radiálnímu proudění mezi vnitřními a vnějšími vrstvami

Při splnění těchto podmínek dochází ve vzestupných a sestupných proudech v důsledku Coriolisovy síly k vytváření vírů. Jelikož jde o vodivé prostředí, jsou pohyby částic spojeny se vznikem elektrických proudů a magnetických polí.

Realizaci magnetohydrodynamického dynama ještě podstatně ovlivňují dvě další okolnosti. Jednou je fluktuace v hustotách a rychlostech a druhou samoorganizační schopnosti magnetických polí. Malé a náhodně indukované víry se mohou silově ovlivňovat interakcí dipólových magnetických polí a projevuje se tendence ke spojování do útvarů s větším uspořádáním a menší celkovou energií.

2.2 Magnetická rekonekce

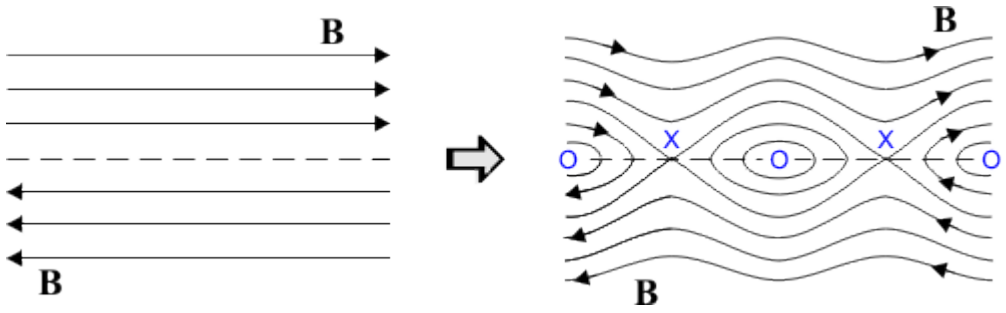
Magnetická rekonekce je přepojení magnetických silokřivek, při kterém dojde k prudké změně dosavadní topologie do jiné, energeticky výhodnější podoby (magnetický zkrat). Při tom dojde k uvolnění energie, jež zahřeje okolní plazmu. Někdy natolik, že plazma zazáří i v RTG oboru.



Obrázek 3: Rekonekce, přepojení magnetických silokřivek.

Pokud má plazma konečnou vodivost, může dojít k transformaci mezi magnetickou, tepelnou a kinetickou složkou energie. Přispívají k tomu proudy tekoucí v plazmatu a následný ohmický ohřev. Představme si, že v plazmatu se k sobě přiblíží dvě oblasti magnetického pole s opačně orientovanými silokřivkami. V této oblasti vzniká tzv. difúzní region. Ten je charakteristický velmi nízkou hodnotou magnetického pole. Právě zde dojde ke změně topologie magnetických silokřivek, jejich přepojení do nové konfigurace s nižší energií. Při přepojení tečou v difúzním regionu velké elektrické proudy, které zahřívají plazma. Energie magnetického pole je transformována do tepelné energie plazmatu. Horké plazma nadbytečnou energii intenzivně vyzařuje do okolí. Makroskopický pohyb plazmatu je při přepojení také ovlivněn. Před přepojením se plazma pohybuje kolmo na silokřivky směrem do difúzního regionu (červené šipky na obrázku). Po přepojení je plazma vytlačována ve směru původní orientace magnetických silokřivek. Podle tvaru magnetických silokřivek se střed difúzního regionu, ve kterém je nulové pole, někdy nazývá neutrální bod typu X. V třírozměrné situaci tvoří hodnoty nulového magnetického pole celou křivku.

V některých situacích způsobí nestability opakované přepojení magnetických silokřivek s periodicky se opakujícími body nulového pole tvaru X a O.



Obrázek 4: Opakované přepojení magnetických silokřivek.

Po přepojení magnetických silokřivek vznikají plazmoidy (plazmová oblaka) a výtrysky (oblasti plazmatu, které odnášejí část energie magnetického pole transformované do energie kinetické a tepelné).

2.3 Ideální a resistivní MHD

Nejjednodušší forma MHD (ideální) předpokládá, že tekutina má tak málo odporu, že ji lze považovat za ideální vodič. V ideální MHD Lenzův zákon říká, že tekutina je v jistém smyslu vázána na magnetické siločáry. Ideální MHD je použitelná pouze pro omezenou dobu a oblast určité velikosti. Resistivní MHD popisují magnetizované tekutiny s konečnou elektronovou difuzivitou $\eta \neq 0$.

2.4 Rovnice ideálního MHD

Rovnice kontinuity:

$$\frac{\partial \rho}{\partial t} + \nabla.[\rho \mathbf{v}] = 0 \qquad (1)$$

Pohybová rovnice:

$$\rho \frac{\partial v}{\partial t} + \rho(v.\nabla)v = -\nabla p + \frac{1}{\mu_0}(\nabla x B)xB \qquad (2)$$

Indukční rovnice:

$$\frac{\partial \mathbf{B}}{\partial t} - \nabla \mathbf{x}(\mathbf{v}\mathbf{x}\mathbf{B}) = 0 \quad (3)$$

Rovnice pro energii:

$$\frac{\partial U}{\partial t} + \nabla S = 0 \quad (4)$$

Maxewllova rovnice pro divergenci magnetického pole:

$$\nabla . \mathbf{B} = 0 \quad (5)$$

Celková energie je vyjádřená jako součet tlakové, kinetické a magnetické rovnice:

$$U = \frac{p}{\gamma - 1} + \frac{\rho}{2} v^2 + \frac{B}{2\mu_0} \quad (6)$$

Rovnice pro tok:

$$S = \left(U + p + \frac{B^2}{2\mu_0}\right) . v - (v . B) \frac{B}{\mu_0} \quad (7)$$

kde B = magnetické pole, p = tlak, u = rychlost proudění, ρ = hustota plynu.

3 Athena a GUI pro zadání a spuštění výpočtu

3.1 Základní popis

Athena je software pro numerické výpočty na rovnoměrně rozložené bodové síti pro řešení astrofyzikálních magnetohydrodynamických rovnic (MHD) v astrofyzice. Tento SW byl především vyvinut pro studium mezihvězdné hmoty, tvorbu hvězd, výpočty dynamiky plynů.

Aktuální verze (4.1), implementuje algoritmy pro:

- stlačitelné hydrodynamiky (compressible hydrodynamics) a MHD v 1D, 2D, 3D
- speciální relativistické hydrodynamiky a MHD
- rovnice ideálního plynu s libovolnou hodnotou γ (včetně $\gamma = 1$, isotermická stavová rovnice)
- libovolný počet pasivních skalárů, které jsou přenášeny tokem
- vlastní gravitaci nebo statický gravitační potenciál
- odporovou, ambipolární difůzi a Hallův jev
- Navier-Stokes a anizotropní (Braginskii) viskozitu
- izotropní a anizotropní vedení tepla
- opticky úzké radiační chlazení

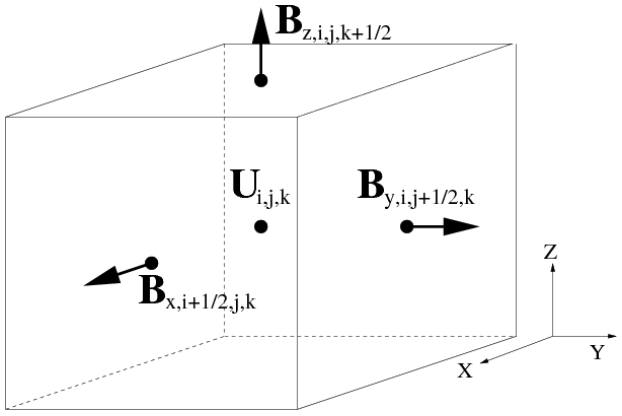
Kromě toho Athena nabízí postoupení mřížky a paralelizační možnosti:

- v pravoúhlých nebo válcových souřadnicích
- statické (fixní) zjemnění
- nastavení výchozích podmínek a orbitální přenesení algoritmu pro MHD
- paralelizace pomocí doménového rozdělení a MPI

Potenciál je také pro numerické algoritmy číselných toků vypočtených pomocí aproximačních nebo lineárních Riemanovských řešitelů.

SW Atheny je vyvíjen od roku 2000 pod vedením J. Stona a P. Teubena (Princeton University). Vývoj je podporován v rámci programu NSF ITR (Information Technology Research for National Priorites). Kód byl vyvinut za použití vývojových nástrojů GNU a přísného dodržování ANSI standardů, díky čemuž je možné jej nakonfigurovat, zkompilovat a spustit jakékoliv platformě, která tyto standardy podporuje. Pro umožnění výpočtu (SMR - Static, not Adaptive, Mesh Refinement) pomocí paralelního zpracování více

procesory MPI jsou výpočetní úkoly v Atheně uspořádány do hierarchie mřížky a domén. Kompletní hierarchie, včetně všech domén a sítí v kalkulaci, se jmenuje Mesh. Doména obsahuje pole výpočetních bloků, které po integraci informací o souřadnicích vytvoří souvislý prostor. Mřížky (Grid) jsou nejmenší oblastí domény a obsahují libovolný počet buněk. Jednotlivé svazky buněk jsou průměrovány, ukládány a aktualizovány.



Obrázek 5: Struktura výpočetní domény.

Na obrázku 5 je vidět umístění magnetických polí na čelních stranách buněk a uložení proměnných uvnitř buněk. Hlavní mřížka obsahuje struktury, magnetické pole umístěné na přední části buňky a informaci o souřadnicích.

Struktura mřížky (Gridu) v kódu:

```
typedef struct Grid_s{
  ConsS ***U;                /* conserved variables */
#ifdef MHD
  Real ***B1i,***B2i,***B3i; /* interface magnetic fields */
#endif /* MHD */
#ifdef SELF_GRAVITY
  Real ***Phi, ***Phi_old;   /* gravitational potential */
  Real ***x1MassFlux;        /*x1 mass flux for source term
                             correction*/
  Real ***x2MassFlux         /* x2 mass flux for source term
                             correction */
```

```
  Real ***x3MassFlux;       /* x3 mass flux for source term
                               correction */
#endif /* GRAVITY */
  Real MinX[3];             /* min(x) in each dir on this
                               Grid [0,1,2]=[x1,x2,x3] */
  Real MaxX[3];             /* max(x) in each dir on this
                               Grid [0,1,2]=[x1,x2,x3] */
  Real dx1,dx2,dx3;         /* cell size on this Grid */
  Real time, dt;            /* current time and timestep  */
  int is,ie;                /* start/end cell index in x1
                               direction */
  int js,je;                /* start/end cell index in x2
                               direction */
  int ks,ke;                /* start/end cell index in x3
                               direction */
  int Nx[3];                /* # of zones in each dir on
                               Grid [0,1,2]=[x1,x2,x3] */
  int Disp[3]               /* i,j,k displacements of Grid
                               from origin [0,1,2]=[i,j,k] */

  int rx1_id, lx1_id;       /* ID of Grid to R/L in x1-dir
                               (default=-1; no Grid) */
  int rx2_id, lx2_id;       /* ID of Grid to R/L in x2-dir
                               (default=-1; no Grid) */
  int rx3_id, lx3_id;       /* ID of Grid to R/L in x3-dir
                               (default=-1; no Grid) */

#ifdef CYLINDRICAL
  Real *r,*ri;              /*cylindrical scaling factors */
#endif                      /* CYLINDRICAL */
}GridS;
```

3.2 Konzole

Athena je v podstatě konzolový počítačový program, který nevyužívá grafické rozhraní. Příkazový řádek (konzole) představuje uživatelské rozhraní, ve kterém uživatel s programy nebo operačním systémem komunikuje zapisováním příkazů do příkazového řádku. Na rozdíl od textového rozhraní a grafického uživatelského rozhraní nevyužívá myš ani menu a nedovede pracovat s celou plochou obrazovky (terminálu). Dřívější verze operačních systému využívaly jenom konzolové aplikace. V unixových systémech je to tradiční způsob ovládání. Dostupný je přímo v textovém uživatelském rozhraní, kde je spuštěn shell, který vytváří příkazový řádek. V grafickém prostředí je shell spouštěn uvnitř speciálních programů, které emulují textový terminál (xterm, Gnome terminal, Konsole, …).

3.2.1 Formát příkazu v shellu

Příkazy zadávané v příkazovém řádku mají formální tvar:

```
prikaz          [prepinace]          [parametry]
```

S přesměrováním pak například:

```
prikaz [prepinace][parametry]<[vstupniSoubor]>[vystupniSoubor]
```

3.2.2 Výhody a nevýhody příkazového řádku

Výhody

Hlavními důvody jsou automatizace a jednoduchost rozhraní. Pomocí skriptů, což jsou textové popisné soubory, můžeme snadno a rychle sestavit posloupnosti příkazů, které zajistí řešení nějakého problému, aniž na to musíme programovat velké množství složitých jednoúčelových programů. Operace se vždy vykonávají stejným příkazem upravitelným různými přepínači. Příkazový řádek může být použit jako skriptovací programovací jazyk a vykonávat operace v dávkách bez zásahu uživatele.

Nevýhody příkazového řádku

Uživatel si musí osvojit znalosti příkazů a různých znakových řetězců, což trvá o mnoho déle a vyžaduje o mnoho více úsilí než osvojení práce s GUI. GUI nabízí různé možnosti řešení určitého problému a informuje o nich uživatele. Konzole nabízí výhradně prázdný promt (prompt /výzva/ indikuje, že příkazový řádek je připraven zpracovat další

příkaz) a předpokládá, že uživatel zná přinejmenším potřebnou syntaxi. Příkazový řádek vyžaduje, aby si uživatel uvědomil, co přesně má dělat a nikoli pouze rozpoznat, co mu systém nabízí. Příkazový řádek má strmější křivku učení než GUI - více času a úsilí musí být vynaloženo učení základům a až poté je možné vykonávat základní úlohy. Proto je příkazový řádek nevhodný pro začátečníky.

3.3 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (anglicky Graphical User Interface, známe pod zkratkou GUI) je uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků. Na monitoru počítače jsou zobrazena okna, v nichž program znázorňuje svůj výstup. Uživatel používá klávesnici, myš a grafické vstupní prvky jako jsou menu, tlačítka, posuvníky, formuláře a podobně. Nevýhody konzolového programu uvedené v předchozí kapitole v podstatě brání využití Atheny méně zkušeným uživatelům OS Linux. Z tohoto důvodu byl vytvořen GUI pro Athenu, kde vstupní parametry jsou ve formě textových souborů a uživatel zadá přes toto rozhraní pouze příslušné parametry a spustí výpočet. Nicméně pro správné používání je zapotřebí pochopit, jakým způsobem program Athena pracuje, jak zpracovává vstupní data a kam se ukládají výstupní soubory. GUI je vytvořeno v programovacím jazyce JAVA s využitím grafické knihovny Swing. Díky tomuto rozhraní mohou program k výpočtům využívat i ti, kteří nemají zkušenosti s prací OS Linux pomocí příkazové řádky.

3.4 Požadavky na OS a výběr distribuce

Jak již bylo uvedeno v základním popisu, program byl napsán za použití GNU a dodržování ANSI standardů. Lze tedy jej spustit v zásadě na jakékoliv Linuxové distribuci. V některých distribucích se občas vyskytne problém s kompilací aplikace a poté je nutné provést odladění. Pro instalaci SW Athena a vývoj grafického uživatelského prostředí pro zadávání, úpravu vstupních parametru a spuštění výpočtu byl zvolen OS linux distribuci Ubuntu 10.10CZ.

4 Instalace programu Athena

4.1 Získání programu

Program Athena aktuální verze 4.1 je volně ke stažení z internetu na stránkách Princetonské univerzity. Před jejím použitím je doporučeno prostudovat dokumentaci v příslušné části.

4.2 Instalace programu

Po stažení souboru `athena4.1.tar` jej pomocí příkazů:

```
gunzip athena4.1.tar.gz
tar xf athena4.1.tar
```

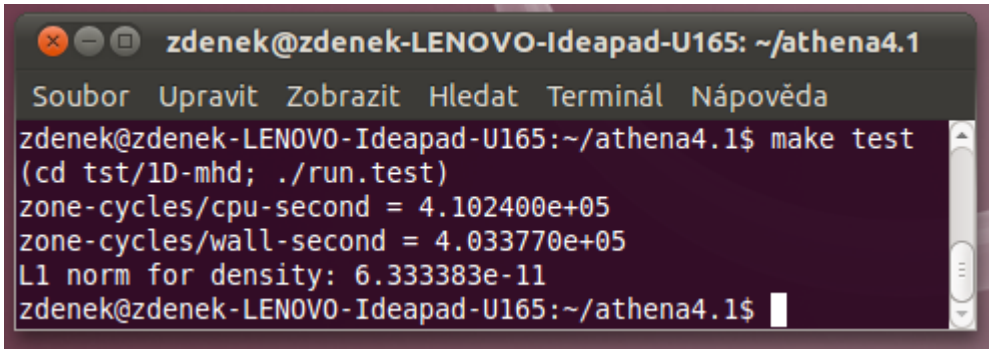
rozbalíme do adresáře na počítači, kde máme nastavena příslušná přístupová práva. Po provedení příkazu vstoupíme do adresáře */athena4.1,* spustíme konfigurační skript:

```
autoconf
```

Nyní máme program nainstalován. Pro kontrolu, že instalace proběhla v pořádku, provedeme test pomocí příkazů:

```
./configure
make all
make test
```

`make test` spustí konvergenční test pro lineární vlnu na mřížce 512 zón. Po úspěšném testu vypadá výpis v obrazovce terminálu následovně:



Obrázek 6: Výstupní okno terminálu po úspěšném testu.

Pro úspěšnou kompilaci bylo nutné do OS Linux distribuce Ubuntu 10.10CZ doinstalovat interpret příkazů **csh**. Tabulka 1 ukazuje adresářovou strukturu programu Athena 4.1 v jednotlivých krocích.

Tabulka 1: Adresářová stuktura programu Athena 4.1.

Po rozbalení	Po konfiguraci	Po kompilaci	Popis
/autom4te.cache	/autom4te.cache	/autom4te.cache	Odhad hodnot pro systém - závislé proměnné a vytváří Makefile.
/doc	/doc	/doc	Příklad skriptu PBS pro spuštění Athény na seskupení s MPI
/src	/src	/src	Zdrojový kód, a soubory include
/tst	/tst	/tst	Testovací vstupní soubory pro řešení problémů
/vis	/vis	/vis	Vizualizační nástroje a skripty
-	-	/bin	Spustitelný soubor „athena“
configure	configure	configure	
configure.ac	configure.ac	configure.ac	
Install-sh	Install-sh	Install-sh	
Makefile.in	Makefile.in	Makefile.in	
Makeoption.in	Makeoption.in	Makeoption.in	
-	config.log	config.log	
-	config.statut	config.statut	
-	Makefile	Makefile	

5 Vstupní soubory

Před spuštěním výpočtu musí být nastaven vstupní soubor, v němž jsou definovány parametry programu. Obvykle jsou tyto vstupní soubory pojmenovány textovým řetězcem ve formátu `athinput.pojmenovani_problemu`, kde část `pojmenovani_problemu` je identifikátor. Zpravidla má stejné označení jako název souboru vytvořený generátorem úloh ve složce *./athena/src/prob,* který je použit pro inicializaci dat. Velké množství vstupních souborů pro rozdílné úlohy je umístěno ve složce *./athena/tst*. Parametry vstupních dat, jsou v souboru seskupovány do bloků. Např.:

```
<job>
problem_id = Strat  # Problem ID: basename of output filename
maxout = 5          # Output blocks number from 1-> maxout
num_domains = 1     # Number of Domains in Mesh
```

Názvy jednotlivých bloků musí být vždy uvedeny v hranatých závorkách na samostatném řádku. Prázdné řádky nad a pod názvy bloků nejsou povinné, slouží pouze pro přehlednost. Pod názvem každého bloku je seznam parametrů se syntaxí:

```
parametr_jmeno =    hodnota   # komentar
```

Prázdné místo po `parametr_jméno`, po `=` a před # je ignorováno. Všechno za znakem #, včetně tohoto znaku, je také ignorováno. Hodnota pro každý parametr musí být uvedena na zvláštní řádce. Maximální počet znaků na řádku je 256. Názvy bloků i parametrů jsou citlivé na velikost písmen. Vstupní soubor je čten programem zvaným „Parser“ vytvořeným pro program Athena *./athena/src/par.c* .Vstupní parametry lze také měnit pomocí příkazové řádky. Například pokud chceme změnit rozlišení mřížky v „*2D Orszag-Tang vortex test“*, stačí spustit Athenu pomocí příkazu:

```
athena -i ../tst/2D-mhd/athinput.orszag-tang domain1/Nx1=128
domain1/Nx2=128
```

Podobně změnu formátu výstupu:

```
Athena -i ../tst/2D-mhd/athinput.orszag-tang
output2/out_fmt=vtk
```

Má to stejný účinek jako editace `<output2>` bloku ve vstupním souboru.

5.1 Blok komentáře

Parametry uvedené v tomto bloku nejsou použity při výpočtu v samotném programu Athena, ale slouží pro zpřehlednění. Využití nacházejí v nově vytvořeném GUI, který využívá hodnotu `config` pro vytvoření spouštěcího skriptu konfigurace při kompilaci.

```
<comment>
problem   = Brio & Wu shock tube
author    = M. Brio & C. C. Wu
journal   = J. Comp. Phys. 75, 400-422 (1988)
config    = --with-problem=shkset1d
```

5.2 Pracovní blok

Parametry v tomto bloku slouží k řízení úloh, které Athena provádí a jsou použity programem *main.c*

```
<job>
problem_id = Strat  # Problem ID: basename of output filename
maxout     =  5     # Output blocks number from 1-> maxout
num_domains =  1    # Number of Domains in Mesh
```

`problem id` - řetězec použitý jako jméno výstupního souboru.
`maxout` - maximální počet bloků, které jsou načteny ze vstupního souboru.
`num_domains` - určuje, které mřížky domény budou přečteny ze vstupního souboru.

5.3 Výstupní blok

Parametry tohoto bloku přímo ovlivňují charakter dat na výstupu. Například výpis programu, binární soubor, obrázek atd. Výstupních bloků může být libovolný počet a ve vstupním souboru mohou být uvedeny v libovolném pořadí.

```
<output1>
out_fmt   = hst          # History data dump
dt        = 62.831853    # time increment between outputs
```

```
<output2>
out_fmt   = rst          # Restart dump
dt        = 6.2831853e3
<output3>
out_fmt   = bin          # Binary data dump
dt        = 628.31853    # time increment between outputs

<output4>
out_fmt   = ppm          # ppm image dump
out       = dVy          # output variable
id        = dVy          # id added to file names
usr_expr_flag   = 1      # indicates dVy is user defined
quantity
palette   = jh_colors    # color palette for images
dt        = 62.831853    # time step between output of deltaV3
dmin      = -0.0006      # min value for imaging delta V3
dmax      = 0.0006       # max value for imaging delta V3
x2        = 0.0          # slice in X-Z plane at x2=0.0
```

out - proměnné výstupu. Povolené hodnoty jsou cons, prim, d, M1, M2, M3, E, B1c, B2c, B3c, ME, V1, V2, V3, P, S, cs2.

out_fmt - Výstupní formát např. bin, hst, tab, rst, vtk, pdf, pgm, ppm.

dat_fmt -Volitelné pole pro formátování výstupního řetězce. Nastavuje tabulátorový výstup do souboru.

dt - čas mezi výstupy.

time - čas pro další výstup. Pokud není nastaven, bude výchozí nastaven s inicializací.

id - textový řetězec přidaný k výstupu.

dmin/dmax - max/min aplikováno pro výstup (vhodné pro obrázky).

palette - paleta barev pro obrázky.

x1, x2, x3 - rozsahy os pro směry x1, x2,x3, přes které jsou data průměrována.

usr_expr_flag - nastavení 1 jestliže uživatelem definovaný výraz se používá pro výpočet výstupní veličiny.

level - úroveň sítě pro výstup. Základní úroveň je 0. Nastavení na hodnotu -1 je výstup tvořen ve všech úrovních. Výchozí hodnota je -1.
Domain - index domény na každé úrovni mřížky výstupu. Počítání začíná na 0.

5.4 Časový blok

Parametry v tomto bloku jsou používány pro definování časových hodnot (např. ukončení výpočtu). Jsou používány programem *„main.c“*

```
<time>
cour_no   =    0.4       #
nlim      =    100000    # Limit cyklů
tlim      =    6.2832e4  # časový limit musí být menší než 1
pro 1D a 2D a menší než 0.5 pro 2D VL nebo 3D
```

cor_no - udává dimenzi výpočtu, musí být menší než 1.0 pro 1D a 2D, pro 3D musí být menší než 0.5.
nlim - maximální počet cyklů hlavní smyčky před zastavením. Přednastavena hodnota -1 a zastaví se za čas uvedený v `tlim`.
tlim - doba, za jakou se zastaví integrace v jednotkách definovaných v řešené úloze.

5.5 Blok domén

Parametry v tomto bloku nastavují vlastnosti domény a MPI. Rovněž pak, jakým způsobem je každá doména rozložena do mřížky. Příklad bloku domén:

```
<domain1>
Level           = 0 # refinement level this Domain (root=0)
Nx1       = 8       # Number of zones in X-direction
x1min     = -0.5    # minimum value of X
x1max     = 0.5     # maximum value of X
bc_ix1    = 4       # boundary condition flag for inner-I (X1)
bc_ox1    = 4       # boundary condition flag for outer-I (X1)
NGrid_x1  = 1       # with MPI, number of Grids in X1
coordinate
AutoWithNProc  = 0  # set to Nproc for auto domain
```

```
decomposition

Nx2       = 64      # Number of zones in Y-direction
x2min     = -4.0    # minimum value of Y
x2max     = 4.0     # maximum value of Y
bc_ix2    = 4       # boundary condition flag for inner-J (X2)
bc_ox2    = 4       # boundary condition flag for outer-J (X2)
NGrid_x2  = 1       # with MPI, number of Grids in X2
                    coordinate
AutoWithNProc  = 0  # set to Nproc for auto domain
                    decomposition

Nx3       = 96      # Number of zones in X3-direction
x3min     = -6.0    # minimum value of X3
x3max     = 6.0     # maximum value of X3
bc_ix3    = 4       # boundary condition flag for inner-K (X3)
bc_ox3    = 4       # boundary condition flag for outer-K (X3)
NGrid_x3  = 1       # with MPI, number of Grids in X3
                    coordinate
AutoWithNProc = 0   # set to Nproc for auto domain
                    decomposition
<domain2>
Level     = 1       # refinement level this Domain (root=0)
Nx1       = 16      # Number of zones in X1-direction
Nx2       = 128     # Number of zones in X2-direction
Nx3       = 96      # Number of zones in X3-direction
Idin      = 0       # i-displacement measured in cells of this
                    level
jesp      = 0       # j-displacement measured in cells of this
                     level
kDisp     = 48      # k-displacement measured in cells of this
                     level
AutoWithNProc= 0    # set to Nproc for auto domain
                    Decomposition
```

```
NGrid_x1  = 1       # with MPI, number of Grids in X1
                    coordinate
NGrid_x2  = 1       # with MPI, number of Grids in X2
                    coordinate
NGrid_x3  = 1       # with MPI, number of Grids in X3
                    coordinate
```

level - stanovení úrovně domény. Musí existovat pouze jeden kořen (level=0) domény bloku a nemusí to být první.

Nx1, Nx2, Nx3 - počet mřížek v osách x1-, x2-, a x3 - směr v každé doméně.

x1min, x2min, x3min - x1-, x2-, x3 - souřadnice levého okraje první buňky.

x1max, x2max, x3max - x1-, x2-, x3 - souřadnice pravého okraje poslední buňky.

bc_ix1, bc_ox1 - příznak pro okrajové podmínky.

iDisp, jDisp, kDisp - offset původu domény od vzniku kořene.

NGrid_x1, NGrid_x2, NGrid_x3 - počet bloků MPI.

AutoWithNProc - nastavení počtu procesorů požadované pro tuto doménu.

5.6 Problém blok

Údaje z této části jsou používány generátorem úloh a jsou závislé na typu řešené úlohy. Hodnoty z tohoto bloku, které jsou společné pro všechny vstupní soubory, jsou:

gamma - poměr specifické teploty použité ve stavové rovnici.

iso_csound - izotermická rychlost zvuku, pouze pro izotermické děje.

5.7 Logovací blok

Parametry v tomto volitelném výstupním bloku jsou určeny pro nastavení standardního `stdout` nebo chybového výstupu `stderr`.

file_open - nastavení výstupu ze `stderr` do souboru se jménem problému `problem_id.err` a z výstupu stdout do souboru `problem _id.out`

flush - počet cyklů mezi vyprázdnění vyrovnávacích pamětí.

lazy - při nastavení 0 vynutí otevření .err a .out souboru, jestliže file_open=1 a to i v případě, že žádný výstup nebyl generován. Pokud **lazy** není rovno 0, jsou tyto soubory

otevřeny pouze v případě, že výstup má být generován. Tato funkce je užitečná pro paralelní úlohy.

`out_level, err_level` - normální a chybový výstup základního procesu.

`child_out_level, child_err_level` - normální a chybový výstup odvozených procesu (potomků).

6 Konfigurace, kompilace, spuštění programu Athena

6.1 Konfigurace

Konfigurační skript slouží k zapínání nebo vypínání funkcí programu a k výběru volitelných balíků funkcí. Základní balíky fyzikálních funkcí obsahují hydrodynamické nebo magnetohydrodynamické úlohy, adiabatickou nebo isotermickou stavovou rovnici a další algoritmické funkce. Další funkcí skriptu je nastavení kompilátoru a linkeru tak, aby reflektoval aktuální stav systému.

Syntaxe:

```
./configure    [--enable - funkce]      [--disable - funkce]
[-- with-balik_funkcí =volba]
```

Pro názornost je třeba uvést příklad nastavení konfigurace pro modelování izotermické hydrodynamické rázové vlny inicializované pomocí Roeova magnetického toku přes interpolaci třetího řádu s jednoduchou přesností:

```
configure  --with-problem=shkset1d  --with-eos=isothermal  --
with-gas=hydro --with-order=3
```

6.2 Kompilace

Po dokončení konfigurace je nutné program ještě zkompilovat pomocí příkazu `make`. V tom samém adresáři *./athena4.1* použijeme skript:

```
make all
```

Dojde k automatickému vytvoření adresáře *./athena4.1/bin* , ve kterém je umístěn spustitelný soubor, k němuž se spustí příkaz `make` v adresáři ./athena4.1/src/, jenž zkompiluje a linkuje kód, jak ukazuje tabulka 2.

Tabulka 2: Příkaz *make* na příkazové řádce.

Volba	**Poznámka**
all	Vytvoří adresář */athena4.1/bin* a zkompiluje do spustitelného stavu
compile	Kompiluje kód
clean	Odstraní soubory s příponou *.o* z adresáře */athena4.1/src*
help	Vyvolá nápovědu
test	Spustí test instalace

6.3 Spuštění programu

Po úpravě vstupních souborů, konfiguraci a úspěšné kompilaci (v adresáři */athena4.1/bin* je spustitelný soubor) můžeme provést spuštění programu pomocí příkazu:

```
./athena  –i  [vstupni_soubor].
```

Pokud použijeme pro spuštění následující příkaz:

```
./athena  –i  [vstupni_soubor]  -d  [vystupni_adresar]
```

dojde k vytvoření adresáře s názvem *výstupní_adresář,* ve kterém budou umístěny data zapsaná do výstupních souborů. Postup při spuštění úlohy pomocí příkazového řádku, za předpokladu, že je program korektně nainstalován:

1) `make clean`
2) `./configure [--enable - funkce] [--disable - funkce]`
   `[-- with-balik_funkcí =volba]`
3) `make all`
4) `cd bin`
5) `./athena  –i  [vstupni_soubor] -d` `[výstupni_adresář]`

V kroku č. 1 dojde k odstranění souborů s příponou `*.o` z adresáře */athena4.1/src*, zadáním příkazu na řádce č. 2 provedeme konfiguraci programu pro daný typ úlohy, dalším krokem č. 3 dojde ke kompilaci programu. V bodě č. 4 pouze vstoupíme do adresáře */bin*, který je umístěn v programu Athena *./athena4.1/bin.* Zde se nachází spustitelný soubor *./athena*. Posledním krokem č. 5 spustíme běh programu a v parametrech mu předáme cestu k vstupním souborům s daty a případně necháme vytvořit adresář, do kterého se nám zapíší soubory s výsledky výpočtu.

7 Výstupní soubory

Výstup dat je nastaven ve vstupních v blocích `<out>.` Počet výstupních souborů není omezen. Výstupní soubory jsou pojmenovány následovně:

`basename-id#.dumpid.outid.type`

`basename` - zděděno z názvu souboru `<job>/problem_id`

`id#` - číslo procesoru při paralelizaci.

`dumpid` - je int naplněný 0.

`outid` - je specifikován v <output> bloku.

`type` - určuje výstupní formát (bin, tab, hst, vtk, rst, pdf, pgm, ppm).

Soubory s výsledky výpočtů najdeme v adresáři ***./athena/bin***

Krátký popis souborových formátů:

History Dumps - (typ=hst) formátovaná tabulka integrovaných hodnot v závislosti na parametru `<output>`, kde každá hodnota je napsána na samostatném řádku, a další parametry. Data jsou přidávána do souboru pokaždé, když je tato funkce volána.

Binární výstup - (typ=bin) neformátovaný zápis závislých hodnot přes všechny aktivní zóny. Pokud je příkazem `configure` aktivována volba DX bude vytvořena hlavička OpenDX souboru a vytvořen `soubor .dx.`

Tabulátorový výstup - (typ=tab) formátovaná tabulka všech závislých proměnných ze všech zón. Vytvoří nový soubor v časovém intervalu `<output>/dt` .

Ppm výstup - typ=ppm dvourozměrné obrázky.

Pgm výstup - (typ=pgm) černobílé obrázky ve formátu pgm. Škála, orientace a průměrování použité pro vytváření jednotlivých obrázku jsou stejné jako u formátu ppm. Generováno funkcí `output_pgm.c`.

Distribuční funkce pravděpodobnosti - (typ=pdf) výstup z vybraných proměnných je vytvářen pomocí funkce `output_pdf.c`.

Obrázek FITS (Flexible Image Transport System) - (typ=fits) to samé jako obrázky ppm, ale ve formátu FITS je vytvářen pomocí funkce `output_fits.c`.

Obrázky VTK (Virtualization Toolkit) - (typ=vtk) podobné binárnímu výstupu, ale ve formátu VTK. Užitečné pro 3D simulace. Je vytvářen pomocí funkce `output_vtk.c`.

Výpis pro restart výpočtu (Restart dumps) - (typ=rst). Vytvoří binární výstup všech proměnných (pokud je to nutné i s dvojitou přesností), které se poté dají použít při restartování simulace. Pro restart simulace je nutný vstupní soubor s hodnotami v ASCII.

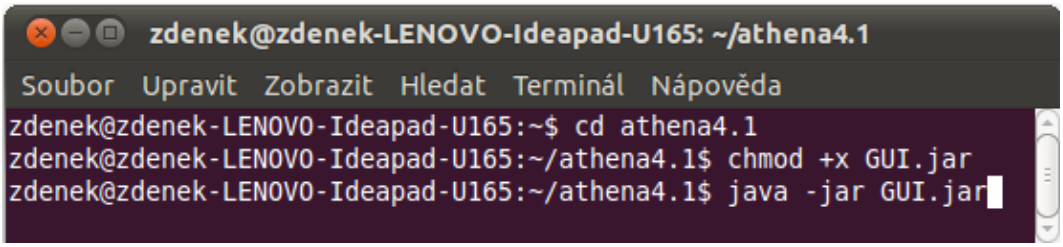
8 Instalace grafického uživatelského prostředí

Grafické uživatelské rozhraní pro program Athena je vytvořeno v programovacím jazyku JAVA. Najdeme jej v archivním souboru `GUI.jar`. Tento soubor umístíme do kořenového adresáře programu Athena a doporučujeme provést také kontrolu spuštění emulátorového okna Xterm v systému příkazem: `xterm` z příkazové řádky. Následně provedeme nastavení přístupových práv k souboru, resp. označíme tento soubor jako spustitelný, pomocí příkazu:

```
chmod +x GUI.jar
```

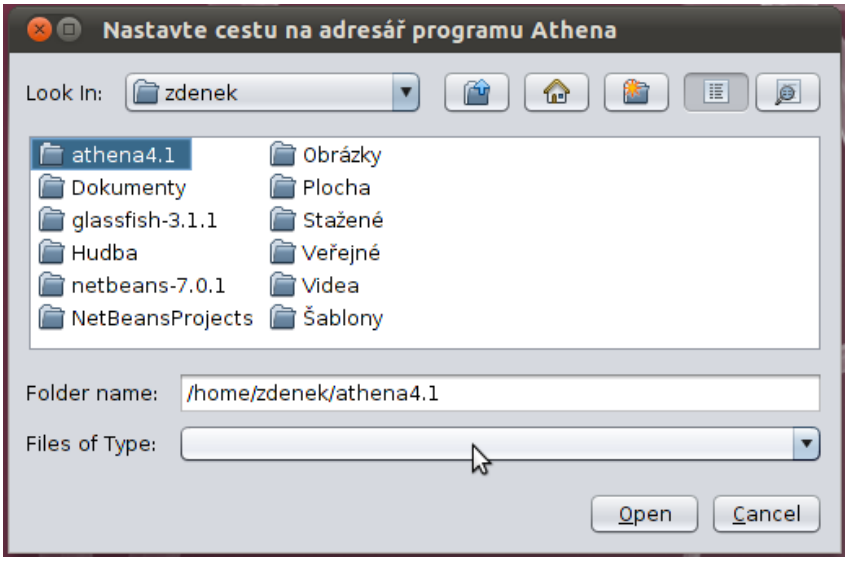
Nyní je všechno připraveno pro spuštění, které provedeme pomocí příkazu:

```
java -jar GUI.jar
```



Obrázek 7: Instalace GUI z příkazového řádku.

Pouze při prvním spuštění se objeví dialogové okno, ve kterém nastavíme cestu ke kořenovému adresáři Atheny.

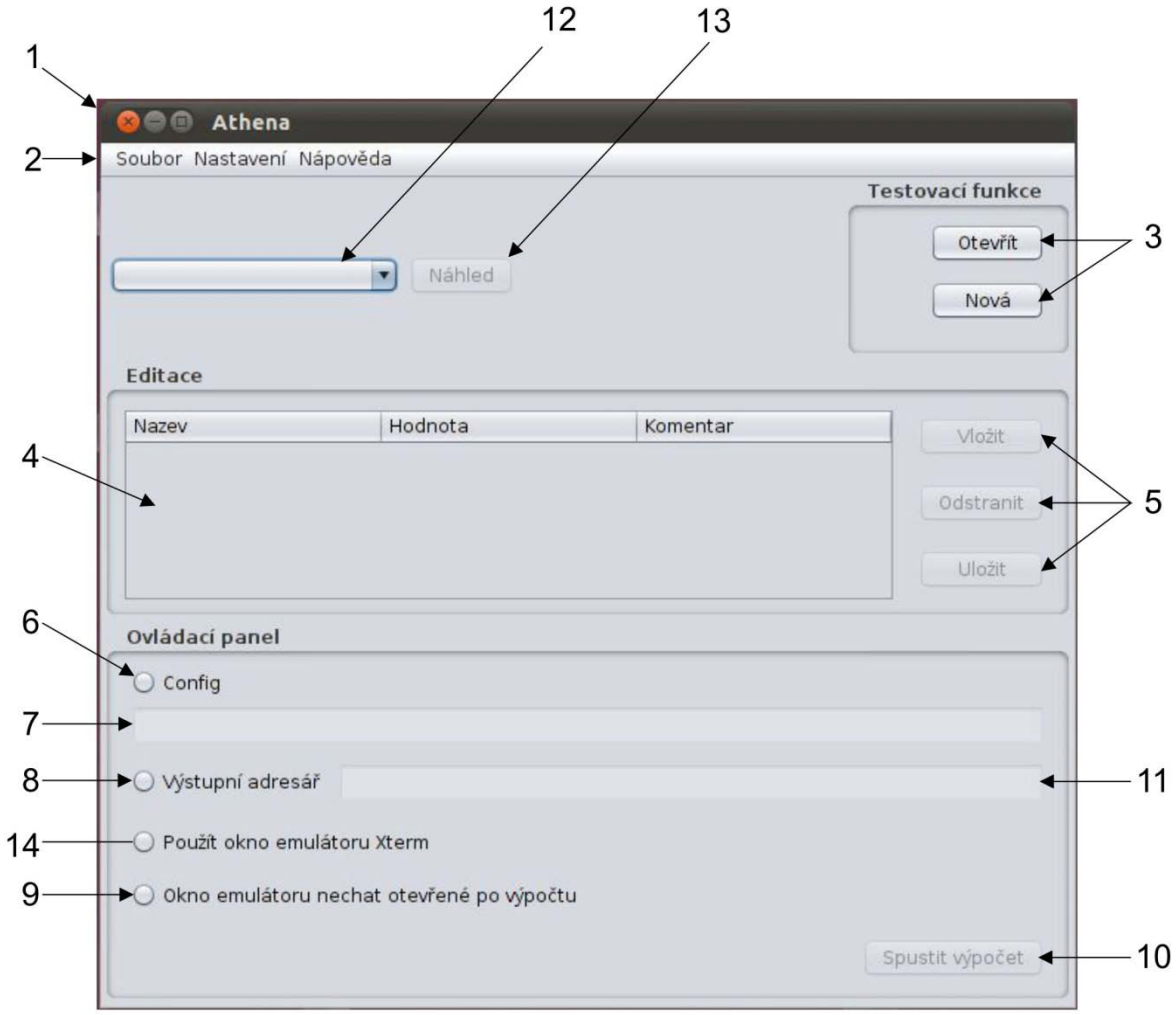


Obrázek 8: Nastavení cesty do kořenového adresáře programu Athena.

Po provedení volby se dostaneme k základní obrazovce GUI pro zadávání vstupních parametrů a spuštění výpočtu.

9 Základní popis a rozvržení komponent

Po spuštění GUI se objeví základní obrazovka programu. Aplikační okno se skládá z následujících částí, které jsou popsány níže:



Obrázek 9: Aplikační okno.

1) tlačítka k ovládání okna (zavřít, zmenšit, zvětšit na celou obrazovku)
2) řádek nabídek s jednotlivými názvy volitelných nabídek
3) tlačítka k otevření stávající testovací funkce z adresáře *./athena/src/prob* nebo vytvoření nové testovací funkce.
4) tabulka, ve které se zobrazí data vstupního souboru

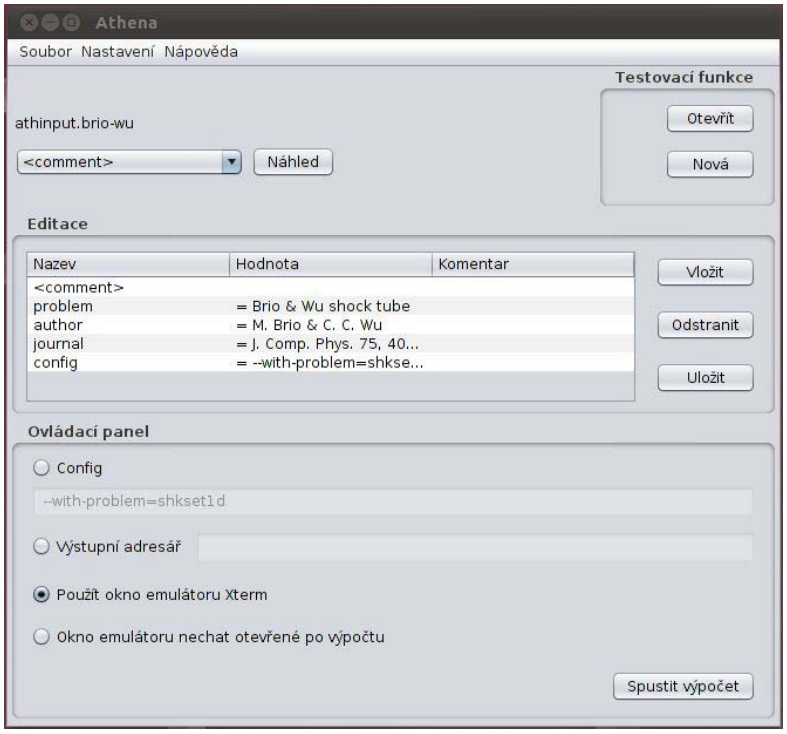
5) tlačítka k úpravě vstupních dat a uložení vstupního souboru
6) přepínač (zaškrtávací políčko) k umožnění editace konfiguračního příkazu pro program Athena
7) textové pole pro změnu konfiguračního skriptu
8) přepínač pro umožnění volby vytvoření a pojmenovaní adresáře pro výstupní data
9) přepínač pro okno externího emulátoru Xterm
10) tlačítko pro spuštění výpočtu v programu Athena
11) textové pole pro pojmenování výstupního adresáře
12) roletové menu pro výběr jednotlivých bloků ze souboru vstupních dat
13) tlačítko pro otevření celkového náhledu na vstupní data
14) přepínač pro zapnutí nebo vypnutí okna emulátoru Xterm

10 Načtení souborů, přístup ke konfiguračním souborům

V případě, že jsme nenastavili cestu programu GUI do kořenového adresáře Atheny, můžeme učinit výběr v *Menu* položky *Nastavení*.

10.1 Načtení souboru se vstupními daty a jejich úprava

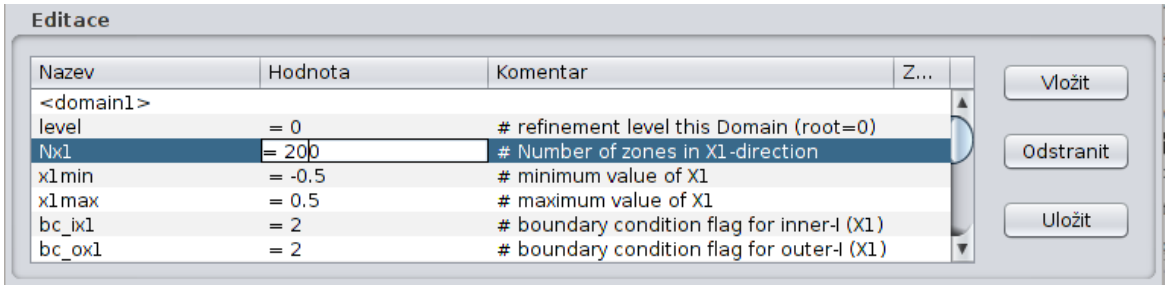
Soubory se vstupními daty se nacházejí v adresáři programu Athena */athena/tst.* Pokud máme nastavenou cestu do kořenového adresáře Atheny, stačí nám pro otevření dialogového okna výběru vstupních souborů stisknout klávesy *CTRL+O*, nebo přes *Menu* v nabídkové liště základního okna *Soubor – Otevřít.* V dialogovém okně pro výběr souboru je přednastaven filtr typu souboru na soubory začínající jménem `athinput._jmen_souboru`. Jedná se totiž o zaběhnutou konvenci uživatelů Atheny. Samozřejmě můžeme tento filtr zrušit změnou typu vstupních souborů v roletovém menu *file – dialogu*. Po výběru vstupních dat, dojde k jejich načtení do aplikačního okna (Obrázek 10).



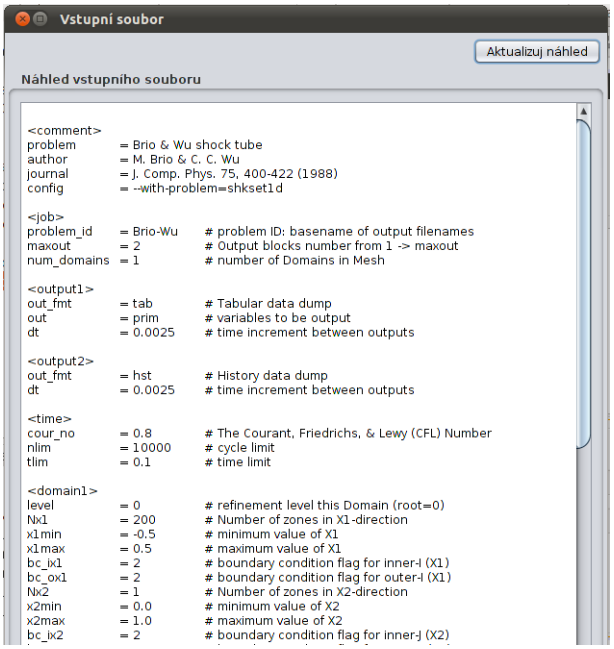
Obrázek 10: Vstupní data načtená do aplikačního okna.

Jméno načteného souboru vidíme nad roletovým menu výběru jednotlivých bloků vstupního souboru (12). Data příslušných bloků jsou načtena v tabulce (4), zároveň dochází ke

zpřístupnění tlačítek pro vkládání a odstraňování řádků dat v příslušném bloku a pro uložení souboru vstupních dat, dále pro spuštění výpočtu (10) a zobrazení náhledu celého vstupního souboru (13). Pokud byla ve vstupních datech informace o nastavení konfiguračního souboru, jsou tyto údaje načteny do textového pole (7). Volbou přepínače (6) je můžeme libovolně měnit. Volba přepínače (8) nám umožní vytvořit a pojmenovat adresář pro výstup výstupních dat z programu Athena. Tento adresář je umístěn */athena/bin/nový_adresář_pro danou_ulohu*. Přepínač č. 9 slouží k vytvoření požadavku, ponechání otevření okna emulátoru, ve kterém můžeme sledovat proběh konfigurace Atheny, kompilace souborů i průběh provádění výpočtu, otevřeného po skončení procesu. Úprava parametrů vstupních dat se realizuje dvojklikem levým tlačítkem myši na příslušnou buňku a zadáním příslušné hodnoty v tabulce (4). Celý vstupní soubor si můžeme prohlédnout v okně náhledu po stisku tlačítka *Náhled* (13).



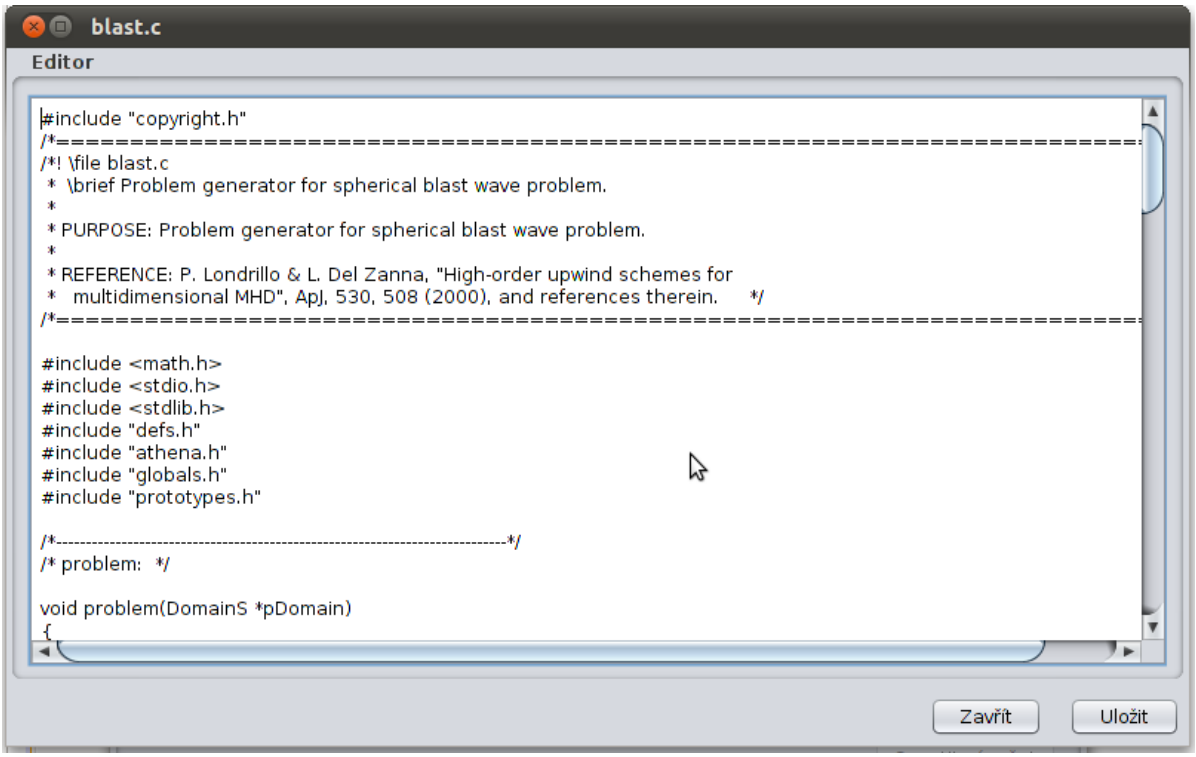
Obrázek 11: Editace dat vstupního souboru.



Obrázek 12: Okno náhledu vstupního souboru.

10.2 Testovací funkce

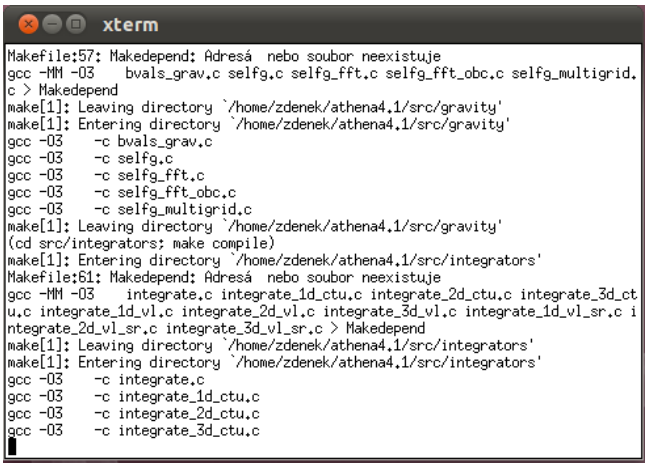
V programu Athena si každý řešitel úloh může vytvořit nebo upravit vlastní testovací úlohu. Děje se tak stiskem jednoho z tlačítek (3) v části testovací funkce. Po stisknutí příslušného tlačítka se buď vytvoří nový prázdný soubor, nebo si přes *file–dialogové* okno otevřeme k úpravám již existující úlohu. Testovací úlohy se nacházejí v podadresáři Atheny */athena/src/prob*. Po provedení úprav, či vytvoření nové testovací úlohy stiskneme tlačítko uložit.



Obrázek 13: Editor testovacích úloh.

10.3 Spuštění výpočtu

Samotné spuštění konfigurace, kompilace a spuštění výpočtu obstará skript *start.sh*, který se vytvoří po stisku tlačítka (10) *Spustit výpočet*. Automaticky se mu přidělí práva spustitelného souboru a dojde k jeho spuštění bez jakéhokoliv dalšího zásahu uživatele. Průběh procesu můžeme sledovat v emulátoru Xterm. V případě, že nemáme unixový emulátor Xterm nainstalovaný v systému, musíme zrušit volbu přepínače (14) „*Použít okno emulátoru Xterm*“, nebo jej doinstalovat. Aplikační okno Javy čeká na dokončení výsledku procesu výpočtu v Atheně. Soubory s výstupními daty se nachází v adresáři *./athena/bin/*



Obrázek 14: Xterm.

10.4 Unixový emulátor Xterm

Xterm je standardní unixový emulátor terminálu pro X window systém. Uživatel může mít současně spuštěno mnoho různých instancí xtermu na jedné obrazovce, z nichž některé poskytují nezávislý vstup / výstup z procesů běžících v něm [15].

10.5 Vizualizace výstupu

Vytvořené grafické uživatelské prostředí se stará pouze o vstupní soubory a automatické spuštění výpočtu. Pro vizualizaci dat je třeba se rozhodnout, který vizualizační nástroj nejlépe vyhovuje našim potřebám.

Supermongo

Supermongo je SW balík pro vytváření 1D obrazových souborů typu SM 11. Jedná se o jednoduchou makrofunkci SM, která dokáže číst tabulkový vstup a zpracovat ho do obrazového formátu je umístěna v adresáři */athena/vis/sim*.

Procedury IDL (Interactive Data Language)

IDL (Interactive Data Language) je výkonný a profesionální externí program, jenž může číst jak binární, tak VTK soubory a vytvářet z datových souborů obrázky. Procedury pro tento program jsou uloženy v adresáři *athena /vis/idl*. Pokud je nutno tyto procedury spouštět

musíme být balík IDL v systému nainstalován. Z adresáře *athena 3.1/bin* je nutno zadat následující příkaz a z binárního souboru se mohou vytvořit 1D obrazové soubory. Vizualizaci datových souborů spustíme tímto příkazem:

```
idl
IDL> .run ../vis/idl/pltath.pro
IDL> nine_plot,'Brio-Wu.0040.bin',1
```

Formát OpenDX

Balík OpenDX umí číst binární soubory z programu Athena tak, že přidá do hlavičky souborů příponu `*.dx`. Před použitím formátu musíme samozřejmě program zkonfigurovat s parametrem `-dx` na příkazové řádce. A samozřejmě do systému musí být nainstalován příslušný OpenDX balíček.

VTK (Visual Tool Kit)

Formát VTK, který program Athena 3.1 vytvoří, je již možno přímo číst programem VisIT. Pro zpracování datových souborů vytvořených paralelizací v síti MPI je v adresáři /*athena/vis/vtk* připravený „C“ kód, který spojuje vytvořené soubory do jednoho. Tento kód je velmi užitečný pro paralelizované zpracovávání většího množství dat na výpočetně výkonných mainframových serverech či clusterech.

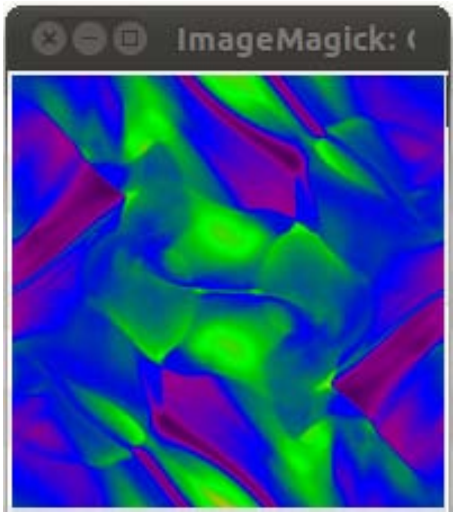
2D Animace

Athena umí vytvářet 2D obrázky, které mohou být zobrazeny přímo nebo snadno animovány. Například, pokud chceme, aby byla vytvořena série obrázků s proměnou ve formátu *ppm*, spojíme je a zobrazíme postupně přímo v programu ImageMagick tímto příkazem: `animate *.ppm`

11 Příklad spuštění Atheny přes GUI

11.1 Spuštění 2D testovací úlohy

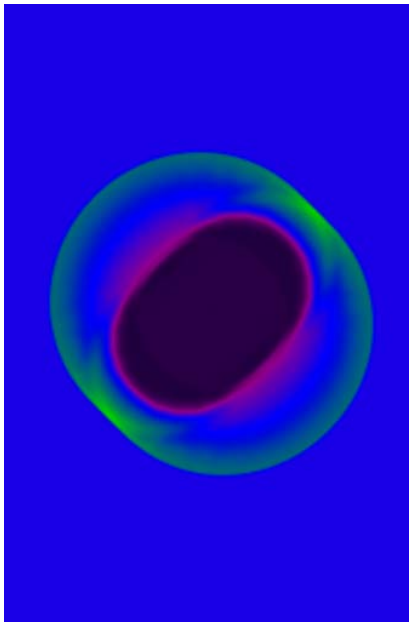
Spuštění testovací úlohy přes GUI je velice jednoduché. Když spustíme program, otevře se nám základní okno, v nabídce *Menu* vybereme *Otevřít*, v souborovém dialogovém okně zvolíme adresář `2D-mhd` a v něm soubor - např. `athinput.orszag-tang`. Dojde k načtení vstupních dat do aplikace. Na řádku pro nastavení konfigurace (7) uvidíme, že došlo zároveň k načtení příkazu pro konfiguraci. Stačí tedy pouze stisknout tlačítko (10) *Spusť výpočet.* Průběh nastavení konfigurace, kompilaci i výpočet můžeme sledovat v okně emulátoru Xterm. Po skončení výpočtu se okno emulátoru automaticky uzavře a v základním okně GUI se objeví nápis „Výpočet dokončen“. V adresáři */athena/bin* si můžeme prohlédnout výstupní data. Program vytvořil 250 obrázků tlaku plynu `OrszagTang.*.P.ppm`, a hustoty plynu `OrszagTang.*.d.ppm`, dále soubor `OrszagTang.hst` a asi 100 binárních souborů `OrszagTang.bin`. Např. Animaci rozložení tlaku plynu si můžeme spustit pomocí programu ImageMagick zadáním příkazu z příkazové řádky v adresáři */athena/bin*: `animate *.P.ppm`.



Obrázek 15: Rozložení tlaku plynu test „Orzsag-tang vortex".

Jak je patrné, postup ovládání je velmi jednoduchý. V případě, že otevřeme například soubor `athinput.blast_B1` v adresáři `2D-mhd` a spustíme výpočet, můžeme po jeho

skončení nalézt výstupní data rovněž v adresáři */athena /bin*. Program vytvořil 100 souborů `Blast_B1.*.d.ppm`, dále soubor `Blast_B1.hst` a 4 binární soubory `Blast_B1.*.bin`. Pokud zadáme příkaz: `animate *.d.ppm`, můžeme sledovat animaci průběhu tlakové vlny.



Obrázek 16: Tlaková vlna test „Blast wave".

12 GUI popis jednotlivých komponent

12.1 Grafické uživatelské rozhraní

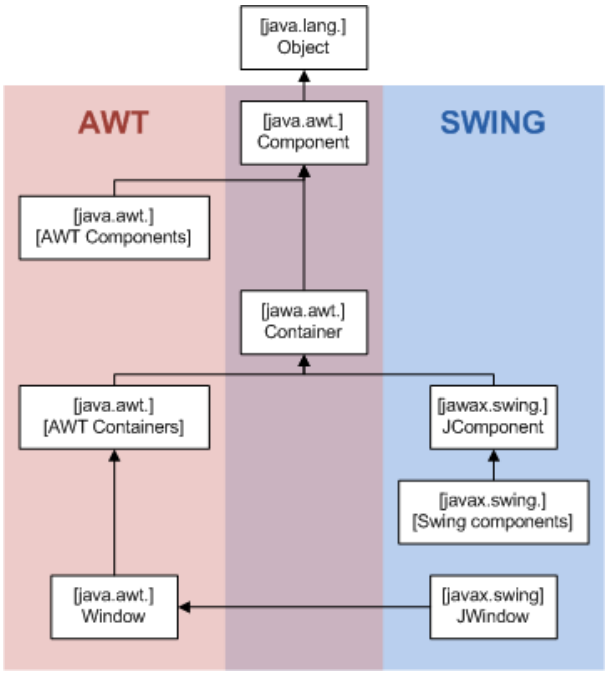
Grafické uživatelské rozhraní (Graphical User Interface, známe pod zkratkou GUI) je rozhraní mezi uživatelem a počítačem, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků. Na monitoru počítače jsou zobrazena okna, v nichž programy zobrazují svůj výstup. Uživatel používá klávesnici, myš a grafické vstupní prvky jako jsou menu, ikony, tlačítka, posuvníky, formuláře a podobně. Neustále docházelo a stále dochází k vývoji rozhraní v komunikaci mezi uživatelem a počítačem. Od původního rozhraní pomocí příkazového řádku (*Command Line Interface - CLI*) se přecházelo přes textové celoobrazovkové rozhraní (*Text- Mode User Interface - TUI*) k rozhraní plně grafickému (*Graphical User Interface - GUI*).

12.2 JAVA

Java je jedním z nejpoužívanějších programovacích jazyků na světě, na němž je mimo jiné oceňována jednoduchá přenositelnost zdrojových kódů na libovolnou platformu. Jedná se o objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems a představila 23. května 1995. Pro účely tvorby grafického uživatelského rozhraní nám Java nabízí hned dvě grafická prostředí - starší AWT a moderní s názvem Swing.

AWT (je obsažené v balíku `java.awt` – 14 rozhraní, 77 tříd a 11 podbalíků se 141 třídami) se objevilo jako první z javovských prostředí a v současné době již není samo o sobě ve větší míře využíváno. Nicméně znalost postupů, které se při tvorbě v AWT používají, nám podstatně usnadní orientaci při používání balíku Swing.

Swing (balík `javax.Swing` – 20 rozhraní, 98 tříd, 15 podbalíků se 363 třídami) představuje nástupce dnes již zastaralého AWT. Toto prostředí podstatně rozšiřuje možnosti komponent AWT (přičemž je možné k základní práci používat známé posluchače z AWT) a navíc přidává i řadu speciálních komponent, které nebyly v AWT obsaženy (například JTable, JTree atd.). Hierarchie tříd grafických komponent Swingu je založena na rodičích z AWT, jak je patrné z obrázku níže.



Obrázek 17: Hierarchie tříd.

12.3 Tvorba hlavního okna aplikace

Abychom měli vůbec možnost na obrazovce vidět nějaké komponenty, musíme vytvořit základní okno. Nejpoužívanějším způsobem tvorby okna z balíku Swing je použití těch to příkazů:

```
import javax.swing.*;
public class ZakladniOkno extends JFrame {
  public ZakladniOkno() {
  // zde si nastavíme název okna
  this.setTitle("Athena"); }
public static void main(String[] args){
     // zde vytvoříme instanci naší třídy ZakladniOkno
     ZakladniOkno frame = new ZakladniOkno();
     // zde nastavíme velikost okna podle
     // velikosti komponent na něm umístěných
     frame.pack();
     // zobrazení okna frame.setvisible(true) ;
     // nastavení operace pro zavření okna
     frame.setDefaultCloseOperation (EXIT_ON_CLOSE);}}
```

Jak je vidět ve zdrojovém kódu, v metodě `main()` si vytváříme instanci „sebe sama“, pomocí konstruktoru `ZakladniOkno frame = new ZakladniOkno();`. Dále nastavíme velikost podle komponent, které zde umístíme. Následuje zobrazení okna a nastavení akce, jež zabezpečí uzavření okna.

12.4 Rozmisťování komponent a práce s panely

V předchozí části je popsán naprostý základ při tvorbě GUI, kterým je především způsob tvorby hlavního okna naší aplikace. Žádná ze základních komponent nemůže být zobrazena samostatně. Umístění jednotlivých komponent v kontejnerové komponentě má na starosti tzv. *layout manager*. Pro vytvoření uživatelsky příjemného GUI je právě výběr správných manažerů zcela klíčový.

12.4.1 Kontejnery

V prostředí Javy se komponenty třídy `java.awt.Container` (kontejner) a její potomci používají pro seskupování dalších komponent. Kontejner slouží jako schránka pro jiné komponenty, včetně dalších kontejnerů. Vzájemným vnořováním komponent a kontejnerů tedy je definována hierarchická struktura prvků v okně. Nejvyšším kontejnerem v této hierarchii je rámec `java.awt.Frame`, který reprezentuje plochu okna.

12.4.2 Správci rozvržení

Správci rozvržení (*layout managers*) jsou třídy používané k řízení velikosti a umístění jednotlivých komponent v kontejneru. Starají se zpravidla o zjištění optimální velikosti kontejneru pomocí metod `getMinimumSize()`, `getMaximumSize()` a `getPreferedSize()`. Použití správců rozvržení je v Javě usnadněno programování, neboť nemusíme zadávat přesné souřadnice jednotlivých prvků. Zadáme pouze hrubý tvar okna a správce rozvržení se sám stará o optimální rozmístění komponent i o jejich vhodné přeskupení při změně jeho velikosti. Všechny *layout managery* mají tu zvláštnost, že téměř nikdy nemusíme volat jejich metody. Stačí pouze vytvořit vhodný manager pomocí konstruktoru a pak jen použít metodu `add()` pro umístění komponenty.

12.4.2.1 FlowLayout

Jedná se o nejjednodušší layout manager. Vkládané komponenty rozmisťuje v pořadí vložení do řádky a nemění jejich velikost. Pokud se již nevejdou na řádek, pokračuje na další řádce.

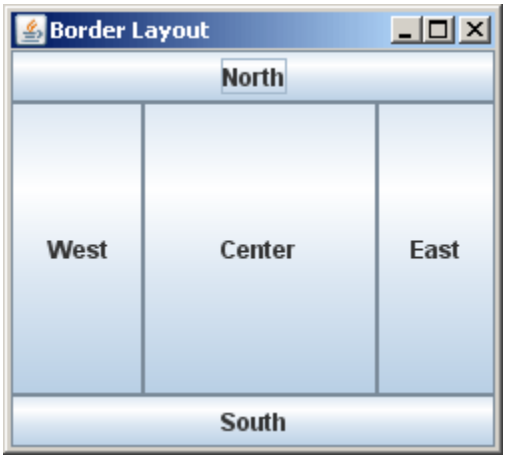
12.4.2.2 GridLayout

GridLayout je jednoduchý layout manager, který vkládané komponenty rozmisťuje v pořadí vložení do předefinované mřížky. K dispozici jsou tři konstruktory:

```
GridLayout()
GridLayout(int radky, int sloupce)
GridLayout(int radky, int sloupce, int hmezera, int vmezera)
```

12.4.2.3 BorderLayout

BorderLayout přestavuje layout manager, do kterého můžeme umístit maximálně pět komponent. Umístění nezáleží na vložení, ale je přesně definováno. Vkládané komponenty zvětšuje tak, že severní a jižní komponentu roztáhne do maximální šířky, ale ponechá jejich výšku. Západní a východní komponenty roztáhne do maximální výšky, ale ponechá jejich šířku. Zbylý prostor vyplní středová komponenta.



Obrázek 18: BorderLayout.

12.4.2.4 CardLayout

V CardLayout jsou komponenty uspořádané do bloků a v jednu chvíli je viditelný pouze jeden blok.

12.4.2.5 GridBagLayout

GridBagLayout umožňuje využít strukturu tabulky, přičemž, každá komponenta může zabírat i více řádek nebo sloupců. K těmto možnostem se přidává schopnost roztažitelnosti vybraných řádek nebo sloupců.

12.4.2.6 FreeDesign

FreeDesign umožňuje umístění komponent na absolutní pozice.

12.5 Základní Komponenty Swing

12.5.1 jLabel

Komponenta slouží k umístění textu, obrázku, případně obojího, který je nezávislý na jakékoliv další komponentě. Pokud nemá přednastavenou barvu pozadí a neobsahuje žádný text, není vidět. Text zobrazený na jLabel lze měnit pomocí `void setText(String text)` a číst pomocí `String getText().`

12.5.2 jTextField

Komponenta představuje editační políčko, do něhož mohou být zadávány jednořádkové údaje z klávesnice. Potvrzení údajů je provedeno stiskem klávesy Enter. Zadávaný text může být delší než šířka JTextFieldu. Komponenta se používá, pokud potřebujeme od uživatele získat vstupní údaje nutné pro běh programu. K dispozici jsou následující konstruktory:

```
JTextField();
JTextField(String text);
```

První konstruktor vytvoří prázdný text field, druhý konstruktor textfield se zadaným textem. Přehled často používaných metod komponenty JTextField. Stisk klávesy <*Enter*> způsobí vygenerování události `ActionEvent`, v implementaci rozhraní `ActionListener` musíme naprogramovat metodu `void actionPerformed`().

12.5.3 jTextArea

Třída `javax.swing.JTextArea` reprezentuje víceřádkové textové pole. Do této komponenty můžeme zapsat libovolný počet řádek textu. Aby bylo možné v tomto poli scrolovat, je nutné JTextArea vložit do komponenty `javax.swing.JScrollPane`. JTextArea je velmi podobná textovému poli.

12.5.4 jButton

JButton neboli tlačítko představuje jednu z nejpoužívanějších komponent, zpravidla slouží ke spouštění nějaké akce. K dispozici jsou dva základní konstruktory:

```
JButton();
JButton(String text);
```

Při stisku myší generuje událost `ActionEvent`. Objekt, který zachycuje tuto událost, musí implementovat rozhraní `ActionListener` s metodou `void actionPerformed()`.

12.5.5 jRadioButton

Tato komponenta je vlastně dvoustavový přepínač a se svým tvarem podobá JCheckBoxu, tlačítko je však kulaté. Od check boxů se odlišuje způsobem výběru, v případě skupiny radio buttonů může být v jednom okamžiku vybráno pouze jedno tlačítko. Radiobuttony mohou být používány samostatně nebo sdružovány do skupiny prostřednictvím komponenty JButtonGroup. Pokud umístíme radio buttony na formulář, aniž je sdružíme do skupiny, mohou být vybírány nezávisle na sobě (nevědí vzájemně o svém stavu). To, že je sdružíme do skupiny, způsobí, že vybraný radiobutton informuje ostatní radio buttony skupiny, takže již nemohou být následně vybrány. Užitečný konstruktor je `public JRadioButton(String text, boolean selected),` jenž vytvoří přepínač s popisem a nastaví na stav zapnuto (true) nebo vypnuto (false).

12.5.6 jComboBox

Komponenta JComboBox představuje rozbalovací seznam. Seznam je tvořen jednotlivými položkami setříděných podle předem známého klíče. Viditelná je pouze jedna řádka. Po kliknutí na šipku se seznam rozbalí a umožní zvolit kliknutím některou z položek seznamu. Použití je podobné jako v případě radio buttonů, komponenta je vhodná pro výběr z většího množství variant. V takovém případě při použití radio buttonů vznikají okna příliš velkých rozměrů, na něž se pak již nevejdou další komponenty. Combo boxy mohou mít poměrně odlišný vzhled i způsob práce s nimi. Jejich položky mohou být editovatelné popř. lze položky přidávat i za běhu programu. Položky combo boxu jsou číslovány, lze k nim přistupovat za použití indexu; první položka má index 0.

12.5.7 jTable

Komponenta zobrazuje tabulku, často se používá pro zobrazování výsledků výpočtů, výpisu seznamů či zadávání vstupních dat. Data v tabulce mohou, ale nemusí být, editovatelná. S tabulkou se zpravidla pracuje za použití dvojice modelů. První model je používán pro sloupce, druhý model pro celou tabulku. Model sloupce ovlivňuje způsob

zobrazení dat ve sloupci či možnosti jejich editace, model tabulky disponuje řadou metod pro práci s vlastní tabulkou. K dispozici je několik konstruktorů:

```
JTable ();
JTable (int rows,int columns);
JTable (TableModel t);
JTable(Object[][] data, Object[] column_names)
JTable(Vector row_data, Vector column_names)
```

12.5.8 Modely v Javě

V GUI Javy je často využívaným postupem práce s modelem. Model představuje implementaci nějakého rozhraní jinou (např. grafickou) třídou. Přes toto rozhraní přistupuje grafická třída ke svým metodám. Použití modelů umožňuje důsledně oddělit grafický návrh aplikace od funkční části. Modely se používají nejen pro práci s grafikou, setkáme se s nimi i při manipulacích s běžnými komponentami. Koncept rozhraní v Javě je tvořen třemi skupinami rozhraní:

- základní rozhraní
- abstraktní implementace rozhraní
- výchozí implementace rozhraní

Základní rozhraní pro práci s modely představují např. `TableModel`, `ListModel`. Abstraktní implementace (implementace rozhraní abstraktní třídou) používáme v případech, kdy chceme předefinovávat funkcionalitu rozhraní (např. *`AbstractTableModel`*), výchozí implementaci rozhraní (implementace rozhraní neabstraktní třídou) v případech, kdy funkcionalitu rozhraní neměníme, např. *`DefaultTableModel`*). Implementace modelu. Chceme - li vytvořit vlastní model tabulky, použijeme abstraktní třídu *`AbstractTableModel`*. Astraktní metody třídy je nutné předefinovat, můžeme tak ovlivnit jejich funkčnost podle potřeby. V tomto případě se to týká zejména následující trojice metod: `getRowCount()`, `getColumnCount()` a `getValueAt()`. Dále je vhodné předefinovat metou `getColumnName()` vracející jméno sloupce. Chceme - li tabulku následně editovat, musíme přetížit metody *`setValueAt(int row,int column)`* a *`isCellEditable(bool status)`*. Při změně dat je nutné tuto informaci poskytnout všem zaregistrovaným posluchačům.

12.6 Objekt

Objekt je datový prvek, který je vytvořen podle vzoru třídy. Objekt je vlastně instance třídy. Pro vytvoření objektu musíme provést dvě akce:

- deklarovat referenční proměnnou, která v sobě bude uchovávat referenci na skutečný objekt
- pomocí **new** nechat vytvořit v paměti objekt a získanou referenci přiřadit do proměnné

Například voláním příkazu:

```
Data data = new Data();
```

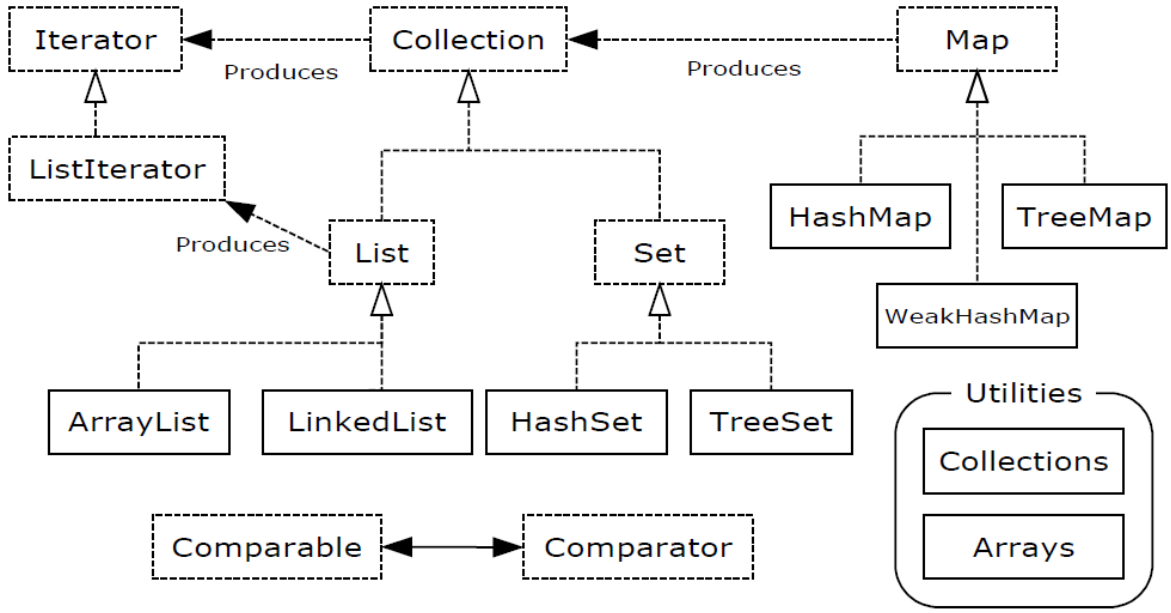
Vytvoříme referenční proměnnou, který ukazuje na objekt `data`, třídy `Data`, obsahující 10 metod (viz níže):

```
public class Data {
 protected String nazev;
 protected String data;
 protected String komentar;
 protected String znak;
public Data(String nazev, String data, String komentar, String
znak) {
         this.nazev = nazev;
        this.data = data;
        this.komentar = komentar;
        this.znak = znak;    }
  public void setNnazev (String name) {
      nazev = name;}
  public String getNazev () {
    return nazev;}
  public void setData (String name) {
      data = name;}
  public String getData () {
    return data;}
  public void setKomentar (String name) {
      komentar = name;}
  public String getKomentar () {
    return komentar;}
  public void setZnak (String name) {
```

```
      znak = name;}
  public String getZnak () {
    return znak;}
   public String getVse () {
return nazev+"\t  " +data+"\t  "+komentar+ "\t "+znak+"\n";}
}
```

12.7 Kolekce (kontejnery)

Při ukládání objektů do dynamických datových struktur (často se pro ně používá pojem kontejner) nemusíme dopředu znát počet vkládaných objektů. Jednotlivé typy kontejnerů umožňují složitější operace s daty. Na následujícím obrázku vidíme zjednodušenou strukturu pro nové třídy.



Obrázek 19: Vazby mezi třídami balíku java.util.

Na první pohled vypadá tato struktura velmi složitě, ale po podrobnějším zkoumání zjistíme, že jde vlastně jen o dva druhy tříd – kolekce (Collection) a mapy (Map). Rozhraní Collection poskytuje základní metody pro tvorbu a použití seznamů (List) a množin (Set). Seznamy ukládají instance do lineární struktury s opakováním prvků. Obvykle se používá třída ArrayList.

12.8 Práce se soubory

Nástroje pro práci se soubory, vstupními a výstupními proudy jsou v Javě součástí knihovny, balíku `java.io`, jenž obsahuje dvě základní třídy – vstupní proud InputStream (z toho můžeme v programu číst data) a výstupní proud OutputStream (do toho naopak data zapisujeme). Tento balíček obsahuje množství dalších proudů, které po těchto dvou dědí a používají se pro nějaký specifický účel. Většinou je z názvu třídy patrné, zda se jedná o vstupní (DataInputStream, FileInputStream, BufferedInputStream) nebo výstupní třídu (DataOutputStream, FileOutputStream, BufferedOutputStream). Základem je třída `java.io.File`. Používá se pro soubory, adresáře, linky i soubory identifikované UNC jmény *(\\pocitac\adresar).*

Příklad pro čtení textového souboru:

```
try {                          // příprava pro postupné
načítání obsahu souboru
BufferedReader in = new BufferedReader(new
FileReader("file.txt"));
                        // načítání souboru po řádcích
String str;
while ((str = in.readLine()) != null) {
                        // nyní můžeme řádek načtený v
                    proměnné str zpracovat
System.out.println(str);  }
in.close();
} catch (IOException e) {
System.out.println("Chyba při zpracování souboru.");}
}
```

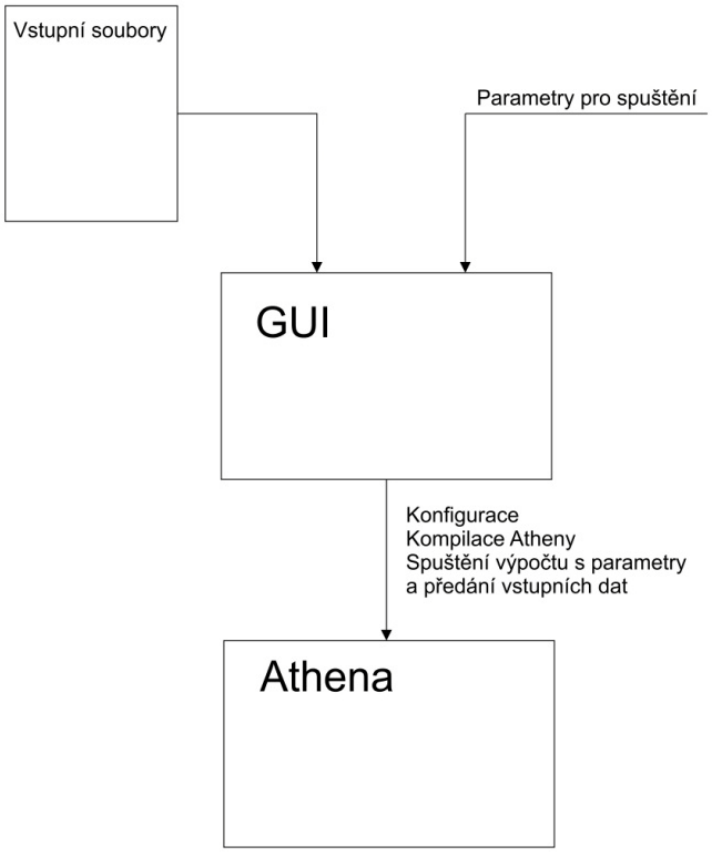
12.9 Spuštění jiné aplikace v samostatném procesu

Ke spuštění programů v jiných programovacích jazycích nabízí JAVA hned několik metod. Můžeme využít přetíženou metodu `exec()` z třídy `Runtime` a výstup zachytit použitím metody `getInputStream()` třídy `process`.

13 Struktura programu

13.1 Základní požadavky na GUI

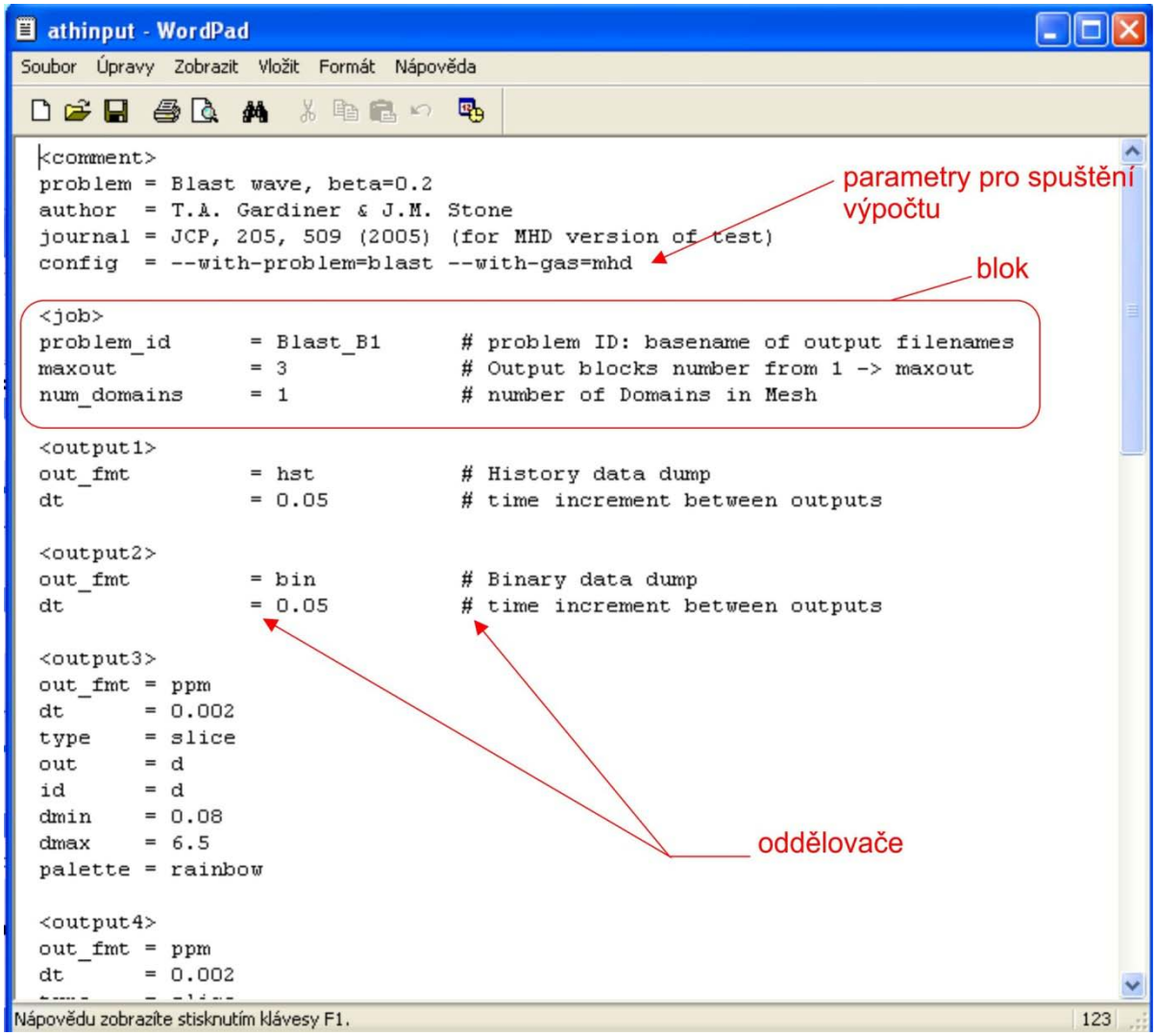
Na následujícím obrázku je znázorněna základní představa, vyplývající se zadání, kdy uživatel po spuštění GUI provede výběr textového souboru se vstupními daty, zadá parametry a spustí výpočet v konzolovém programu Athena.



Obrázek 20: Základní rozbor úlohy.

Protože by bylo značně nepohodlné upravovat soubory se vstupními daty v jiném programu, přidáme možnost editace těchto souborů v grafickém uživatelském rozhraní v jednotlivých blocích, dále možnost tvorby nových, případně úpravu stávajících testovacích funkcí, které se nacházejí v adresáři */athena4.1/src/prob*. Pokud bude vstupní soubor uvádět parametr `config,` v nepovinném bloku `<comment>`, přednastavíme tento parametr v příkazu pro spuštění výpočtu s možností volby změny tohoto parametru pro uživatele.

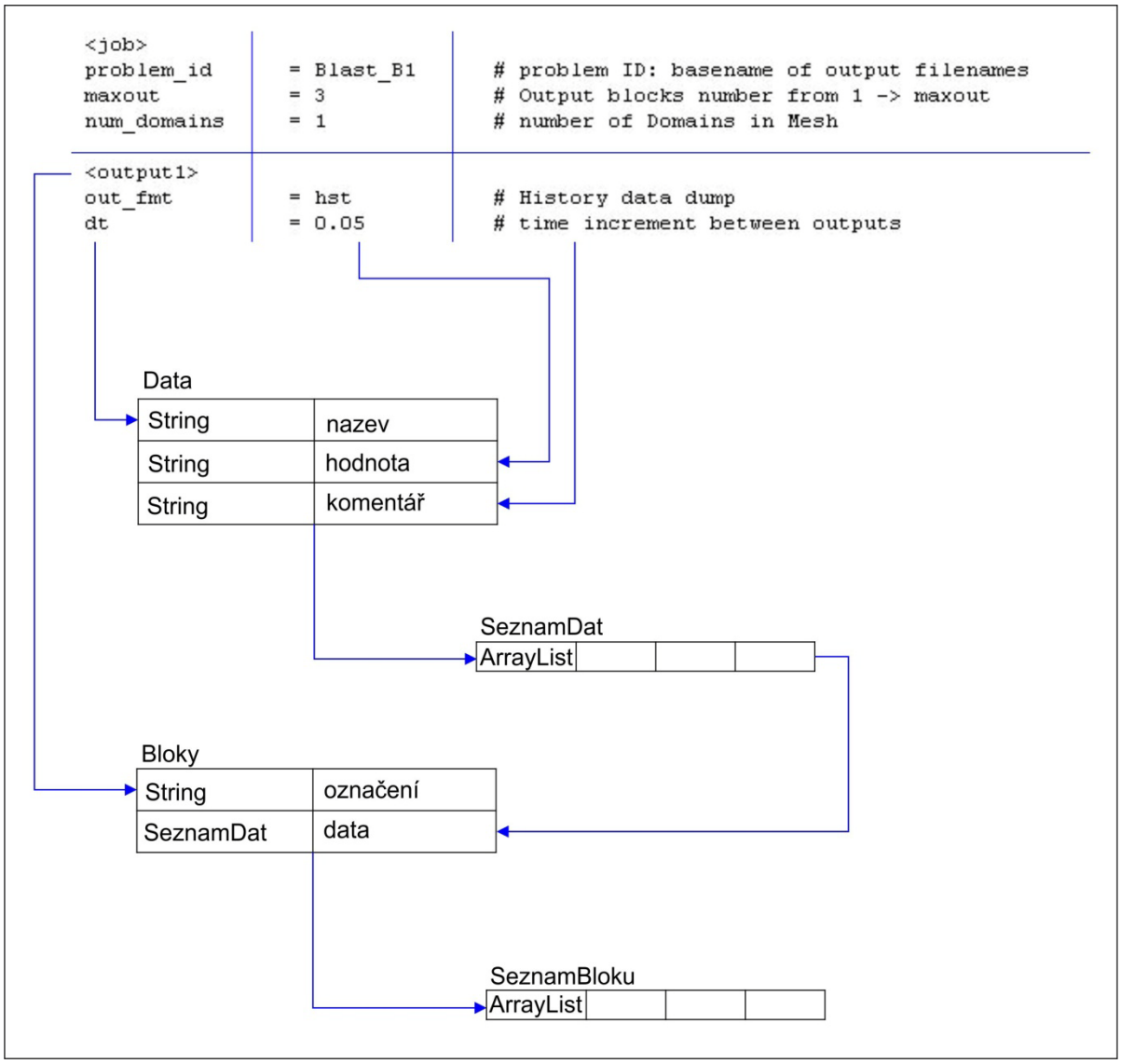
13.2 Rozbor vstupních textových souborů



Obrázek 21: Soubor se vstupními daty.

Vstupní textové soubory se vstupními daty jsou určitým způsobem členěny. Parametry vstupních dat jsou seskupovány do bloků, které musí být na samostatném řádku a jsou umístěny v závorkách `<nazev_bloku>`. Znaky = a # jsou použity jako oddělovače. V **nepovinné** části blok `<comment>` se vyskytuje řádek config, na kterém jsou zpravidla uvedeny parametry pro spuštění výpočtu programu Athena.

Vstupní soubor si rozdělíme do jednotlivých bloků, které jsou umístěny v seznamu (ArrayList) bloků třídy `SeznamBloku` (obrázek č. 20). Každý blok (`třída Bloky`) se skládá z označení (String) a objektu `SeznamDat` (ArrayList). V seznamu dat `(SeznamDat)` jsou uloženy jednotlivé řádky, rozdělené do skupin řetězců třídy `Data` podle jejich využití (název, hodnota, komentář). Vše je názorně ukázáno na obrázku č. 19.

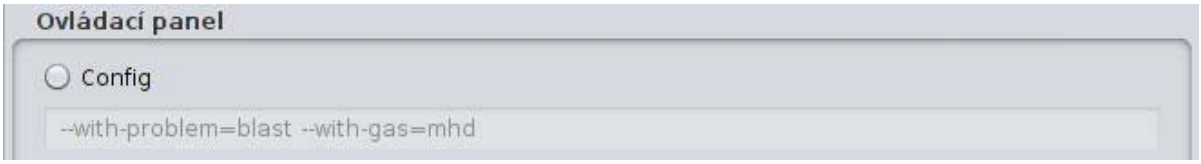


Obrázek 22: Rozklad vstupního souboru do seznamu ArrayList.

Vzhledem ke skutečnosti, že znak = , který je použit jako oddělovač, se vyskytuje i v místech, kde z hlediska našeho rozboru funkci oddělovače neplní, ať již je to např. v bloku `<comment>` nebo kdekoliv v komentářích, máme k analýze vstupního souboru metodu `rozbor()` třídy `AnalyzujZarad`, která vrací objekt `SeznamBloku`. Využíváme skutečnosti, že znak = má funkci oddělovače pouze v prvním výskytu na řádku vstupního souboru. Jeho pozici zjistíme pomocí metody `indexOf()` třídy String, která vrací pozici znaku `int`. Ten samý postup využíváme i pro znak #.

13.3 Zjištění parametru *config* ve vstupním souboru

Při načtení vstupního textového souboru zavoláme metodu `vrat()`, třídy *`VratPolozku`*. Tato metoda nám vrátí textový řetězec s konfiguračními údaji, pokud se ve vstupním souboru vyskytuje blok `<comment>` a v něm položka *`config`* a zapíše parametry pro konfiguraci do textového pole `jTextField1`třídy `ZakladniOkno`.



Obrázek 23: Vloženi konfiguračních parametru z textového vstupního souboru.

13.4 Zobrazení a editace dat vstupního souboru

Data ze vstupního souboru po blocích zobrazujeme v tabulce třídy *`JTable`*, která je komponentou grafického uživatelského rozhraní z knihovny Swing ze základního balíčku `javax.swing`. Pro zhotovení tabulky jsme vytvořili třídu *`Model`*, která popisuje, co má být obsahem jednotlivých buněk. Tato třída je potomkem třídy `AbstractTableModel`. Při změně ve výběru bloku v roletovém menu dojde k načtení dat příslušného bloku do Modelu a následně do Tabulky, pomocí metod `fire<typZmeny>`, kde si uživatel může editovat jednotlivé položky, přidávat, nebo ubírat řádky tabulky. Tato změna se zapisuje do příslušných částí v seznamu bloku. Náhled celého souboru si můžeme prohlédnout v dialogovém okně po stisku tlačítka *Náhled* umístěného vpravo od roletového menu výběru jednotlivých bloků.

Tabulka 3: Metody *fire*, třídy AbstractTableModel.

Metoda	Změna
fireTableCellUpdated	Aktualizace zadané buňky
fireTableRowsUpdated	Aktualizace uvedených řádků
fireTableDataChanged	Aktualizace tabulky (pouze data)
fireTableRowsInserted	Vložení nového řádku
fireTableRowsDeleted	Odstraní existující řádek
fireTableStructureChanged	Zruší tabulku jak data, tak strukturu

13.5 Vytvoření spouštěcího skriptu a spuštění výpočtu

Po stisknutí tlačítka *Spustit výpočet* se zavolá metoda `pripravenSpustit()`, která vytvoří v adresáři */athena4.1* soubor `start.sh` , ve kterém je napsán příslušný spouštěcí skript:

```
#!/bin/sh
cd [cesta_do_adresáře_programu_athena]
make clean
./configure  [doplni_parametry_pro_konfiguraci]
make all
cd bin
./athena -i [soubor_se_vstupnimi_daty] [vystupni_adresar]
```

Po té dojde přes metodu `opravneni()` třídy *Spust* k přidělení povolení spustit soubor jako program `chmod +x cesta_k_souboru/start.sh.` Následně dojde k zavolání metody `start()` , rovněž třídy *Spust* ke spuštění tohoto skriptu `start.sh` v dialogovém okně emulátoru Xterm, kde může uživatel sledovat průběh práce programu Athena. GUI čeká na dokončení procesu výpočtu a při úspěšném skončení vypíše text: *„Výpočet ukončen“*.

13.6 Editor pro testovací funkce programu Athena

Pro vytvoření nového souboru testovacích funkcí nebo editaci stávajícího slouží dialogové okno, které vyvoláme stiskem tlačítka v části *Testovací funkce* hlavního okna. Po provedení volby se nám otevře dialogové nové okno, v němž můžeme v komponentě `jTextArea1` vytvořit nebo upravit stávající testovací soubor. Otevření stávajícího souboru probíhá přes *file dialog*, kde provedeme volbu souboru. Pro čtení souboru nejprve vytvoříme instanci čtenáře `FileReader`. Do komponety třídy JTextArea ji načteme pomocí metody `read()` třídy JTextComponent.

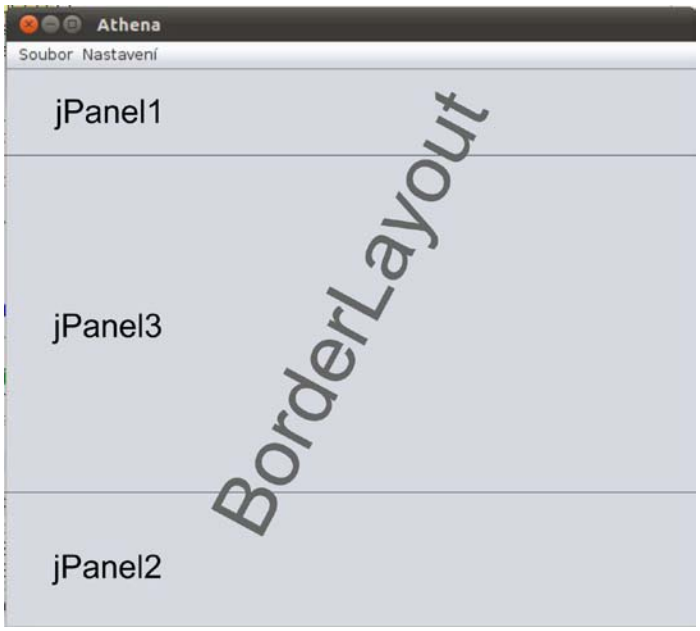
```
jTextArea1.read(new FileReader(file.getAbsolutePath()),null);
```

13.7 Konfigurační soubor pro GUI

Při první spuštění grafického uživatelského prostředí se nám nejprve objeví dialogové okno s žádostí o nastavení cesty do kořenového adresáře programu Athena. Po jeho zadání se dojde k vytvoření souboru se jménem `j_conf_athena`, ve kterém je tato cesta zapsána. GUI vždy při spuštění testuje, zda tento soubor existuje, a pokud tomu tak není, dojde znovu k objevení dialogového okna. V případě, že soubor s cestou existuje, dojde k automatickému načtení do proměnné typu `String athena_path`. Tuto hodnotu můžeme změnit výběrem nabídky *Nastavení – Předvolby* v Menu.

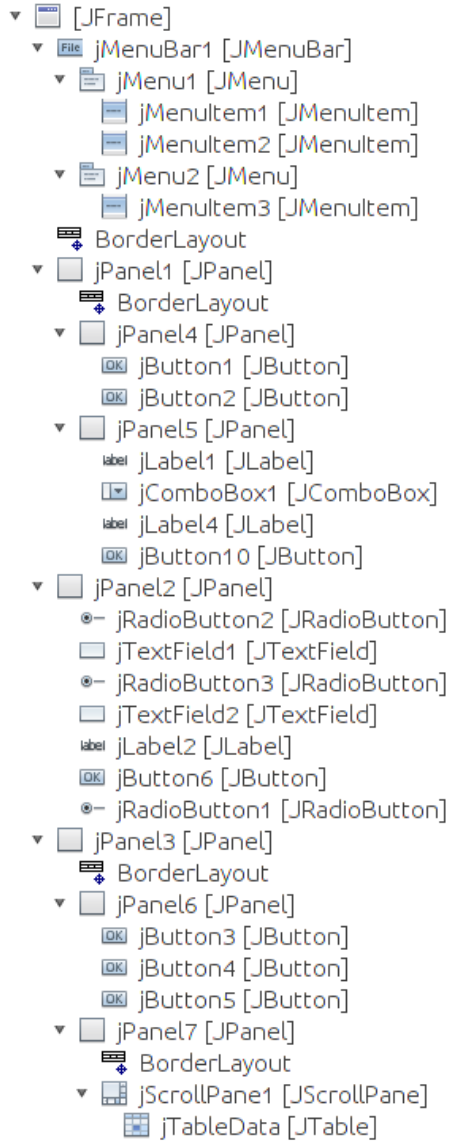
13.8 Způsob rozvržení aplikačního okna

Kontejnerové třídy pojmou celou řadu komponent, které je možné v okně rozmístit různými způsoby. Aby nebyl každý kontejner zatížen mnoha různými druhy výpočtů, rozvržení výpočty provádí za kontejner třída, jež se nazývá *správce rozvržení (layout manager).* Základní typy správců rozvržení jsou popsány v kapitole 11. Hlavním prvkem GUI je okno, ve kterém jsou kontejnery, a v nich základní prvky umístěny následujícím způsobem. Do okna je umístěn správce z rozvržení s BorderLayout. V horní části NORTH je umístěn jPanel1, uprostřed CENTER – jPanel2 a v dolní části SOUTH – jPanel3, jak můžeme vidět na obrázku 22.



Obrázek 24: Rozvržení kontejneru v základním okně.

Celková skladba uspořádání je zobrazena na následujícím obrázku ve stromové struktuře. V kontejneru jPanel1 je opět zvolen správce rozvržení BorderLayout, do něhož jsou vloženy *jPanel4* na pozici EAST s tlačítky *jButton1* a *jButton2* a *jPanel5* na pozici CENTER s roletovým menu `jComboBox1`, tlačítkem *jButton10* a textovými komponenty `jLabel1` a `jLabel4`. V kontejneru `jPanel2` je použit `FreeDesing` a jednotlivé základní komponenty jsou umístěny v absolutních pozicích. V prostřední části `jPanel3` opět využívá správce rozvržení *BorderLayout* , do něhož jsou umístěny panely `jPanel6` s tlačítky na pozici EAST jPanel7 na pozici CENTER, do kterého je umístěna komponenta `jScrollPane1` v níž se nachází tabulka `jTableData`.



Obrázek 25: Stromové zobrazení uspořádání komponent v základním okně.

14 Závěr

Program Athena je užitečným a silným nástrojem pro řešení astrofyzikálních úloh. Jeho předností je především možnost řešení problémů pomocí tvorby nových testovacích funkcí.

Cílem této práce bylo vytvoření grafického uživatelské prostředí, jenž umožní uživateli výběr textového souboru se vstupními daty, zadání parametrů a spuštění výpočtu. Pro splnění tohoto cíle byl důkladně prostudován konzolový program Athena, včetně způsobu zadávání dat, nastavení konfigurace, kompilace a spuštění výpočtu. Vytvořené grafické uživatelské prostředí umožňuje provádět úpravu souborů se vstupními daty a změnu parametru konfigurace. V dialogovém okně lze vytvářet nebo upravovat testovací funkce. V případě, že si uživatel vybere ke spuštění některou z testovacích úloh, která zahrnuje údaj o potřebné konfiguraci ve vstupním souboru, stačí mu vstupní soubor otevřít a jedním stiskem tlačítka spustit výpočet. Díky GUI mohou s tímto programem pracovat i méně zkušení uživatelé systému Linux. Nicméně pro správné používání je zapotřebí alespoň minimálních znalostí, jakým způsobem jsou zadávána vstupní data a kam se ukládají výstupní soubory.

15 Seznam obrázků

16 Seznam tabulek

17 Seznam literatury

[01] Chung, T. J.: Computational Fluid Dynamics.Cambridge University Press, New York, 2002.

[02] Priest, E. R.: Solar Magnetohydrodynamics, D. Reidel Publishing Company, London England, 1982.

[03] Jelínek, P. - Karlický, M.: Magnetoacoustic waves in diagnostics of the flare current sheets, Astronomy and Astrophysics, in print, 2011.

[04] Jelínek, P. – Bárta, M., MHD Simulations in Plasma Physics, in e-Proc. Technical Computing Prague 2006, Praha, 6 stran, 2006.

[05] Kulhánek, P.: Úvod do teorie plazmatu, AGA (Aldebaran Group for Astrophysics), 2011.

[06] Athena 3D Documentation: https://trac.princeton.edu/Athena/

[07] Siever, E.: Linux v kostce, Coputer Press, Praha, 1999.

[08] Petrlík, L.: Jemný úvod do systému UNIX, nakladatelství Koop, České Budějovice, 2001.

[09] Shah, S.: Administrace systému Linux, Grada Publishing, Praha, 2002.

[10] Darwin, F. I.: Java, kuchařka programátora, Computer Press, Brno, 2006.

[11] Java SE Technical Documentation: http://docs.oracle.com/javase/

[12] Herout, P.: Java, grafické uživatelské prostředí a čeština, nakladatelství Koop, České Budějovice, 2001.

[13] Herout, P.: Java bohatství knihoven, nakladatelství Koop, České Budějovice, 2003.

[14] http://flash.uchicago.edu/website/home/

[15] Emulátor Xterm: http://invisible-island.net/xterm/xterm.html

18 Příloha CD

Soubory

01 Anotace.docx

02 Anotace.pdf

03 Grafické rozhraní pro volně šiřitelné astrofyzikální numerické kódy.docx

04 Grafické rozhraní pro volně šiřitelné astrofyzikální numerické kódy.pdf

Adresáře

05 Programy

- athena4.1.tar.gz (verze k 15.12.2011)
- GUI.jar

06 Elektronická dokumentace javadoc

07 NetBeansProjects