Introdução à Programação de Aplicações Científicas em Java

Escola de Verão do Laboratório Associado de Computação e Matemática Aplicada do INPE

Rafael Santos

Sobre o curso

- Dividido em quatro partes:
  1. **Tecnologia, Linguagem e Orientação a Objetos.**
  2. *API*s comuns.
  3. Programação com Interfaces Gráficas.
  4. APIs para Processamento Científico (aplicações diversas).
- Cada parte tem aproximadamente 3 horas.
- O curso é somente teórico, sem laboratório.
  - Exemplos, *links*, etc. serão disponibilizados em http://www.lac.inpe.br/~rafael.santos
- Muitos exemplos formatados como *how-to*.

Porquê Java?

Porquê Java?

- Simplicidade (sim!)
- Portabilidade (diferentes sistemas operacionais).
- Aplicabilidade (*desktop*, web).
- Padrão “aberto”, licenciamento.
- Flexibilidade (APIs).

- Onde sua aplicação/biblioteca/ API estará em 2, 5, 10 anos?

A Tecnologia Java

Tecnologia Java

- Todo mundo já ouviu falar!
- Descrição em duas frases:

> *Java technology opens up a wealth of exciting possibilities for consumers. It enables just about any application -- including games, tools, and information programs and services -- to run on just about any computer or device. From desktop PCs to mobile handheld devices and cell phones, Java technology today is just about everywhere.*

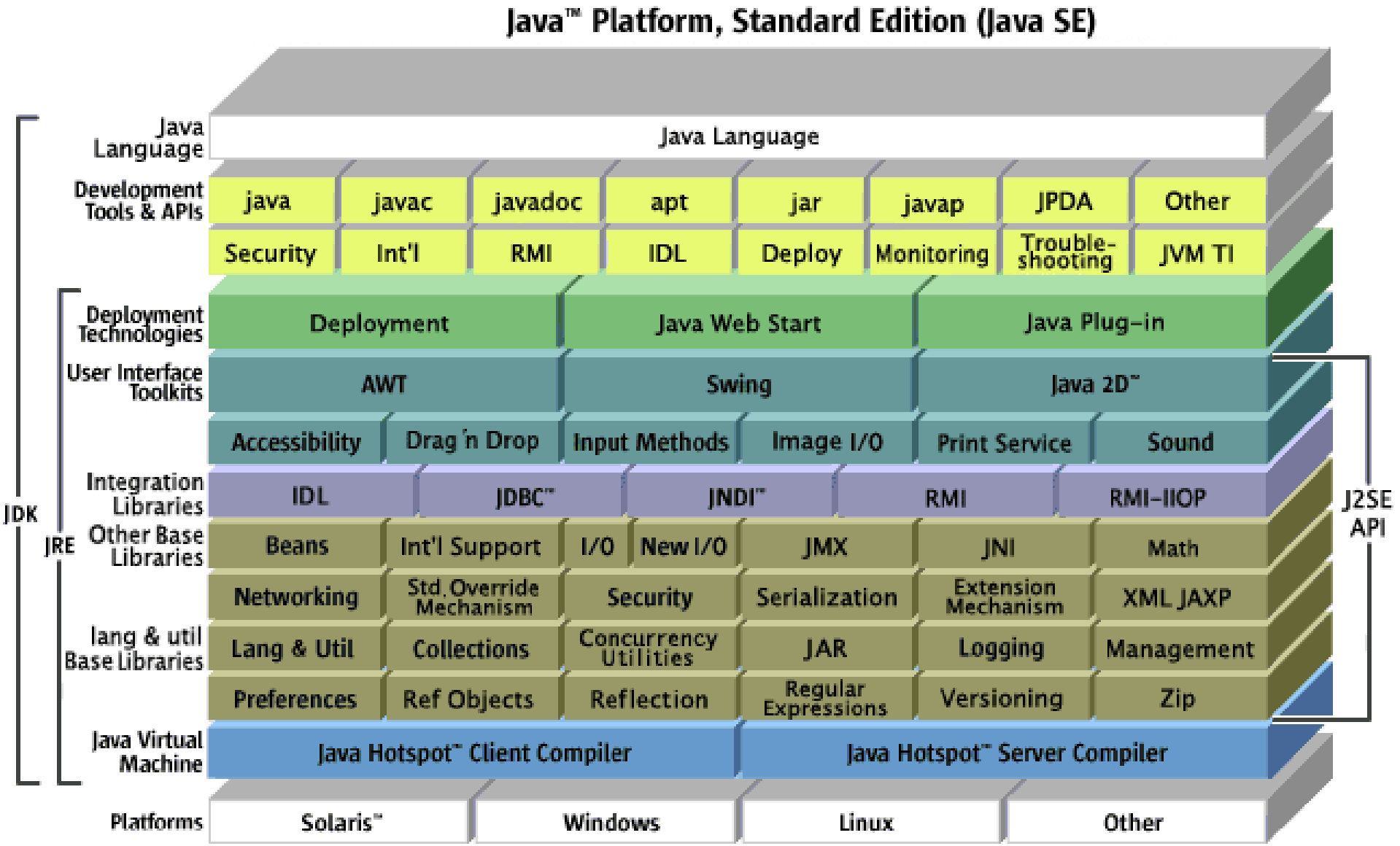
Veja

- http://www.java.com/en/download/faq/whatis_java.xml
- http://www.java.com/en/index.jsp

Tecnologia Java

- Código Java pode ser executado em *containers* ou máquinas virtuais:
  - Em navegadores, como *applets* com interfaces gráficas ricas;
  - Em desktops, como aplicações completas, possivelmente distribuídas via Web (*Java Web Start*);
  - Em servidores, como servidores de aplicação (JSP/*Servlets*);
  - Em dispositivos portáteis como celulares, computadores de mão e outros.
- A base é a mesma (linguagem Java), algumas APIs são **diferentes** para os *containers*.
  - J2SE, J2ME, J2EE (recentemente JSE, JME, JEE).

Tecnologia Java



Linguagem Java

- Simples, orientada a objetos.
- Herdou muitos conceitos de C, C++, outras.
- Código compilado para *bytecodes*, interpretado por uma máquina virtual.
  - *Bytecodes* compatíveis entre sistemas operacionais*.
  - Base compatível entre máquinas virtuais.
  - APIs dependem da finalidade, mas código de negócio é portátil!
- Otimização de *bytecodes* melhora a performance.

Showcase

Showcase

- Quem disse que Java...
  - Só serve para *applets*, e Flash é melhor?
  - Só serve para aplicações em servidores com interfaces em HTML?
  - Não pode ser usada para criar aplicações com interfaces gráficas ricas?
  - Não serve para aplicações do “mundo real”?

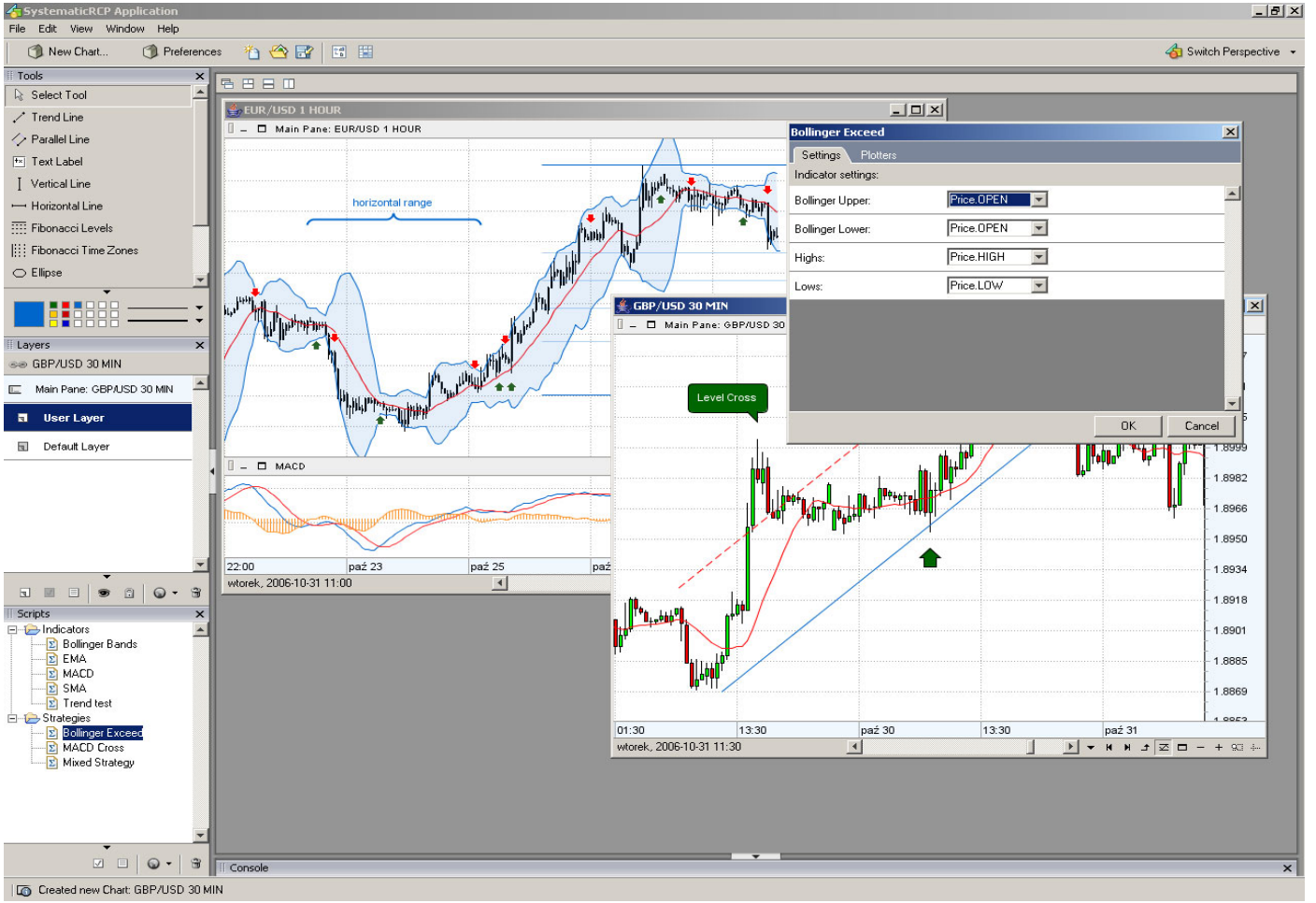
- Lembrete 1: muitas das aplicações mostradas são comerciais/proprietárias, então devemos considerar *investimento* na sua criação.
- Lembrete 2: Praticamente só JSE.

Showcase: Pessoal

- Aerith (https://aerith.dev.java.net/): *Mashup* para criação de *slideshows* georeferenciados.

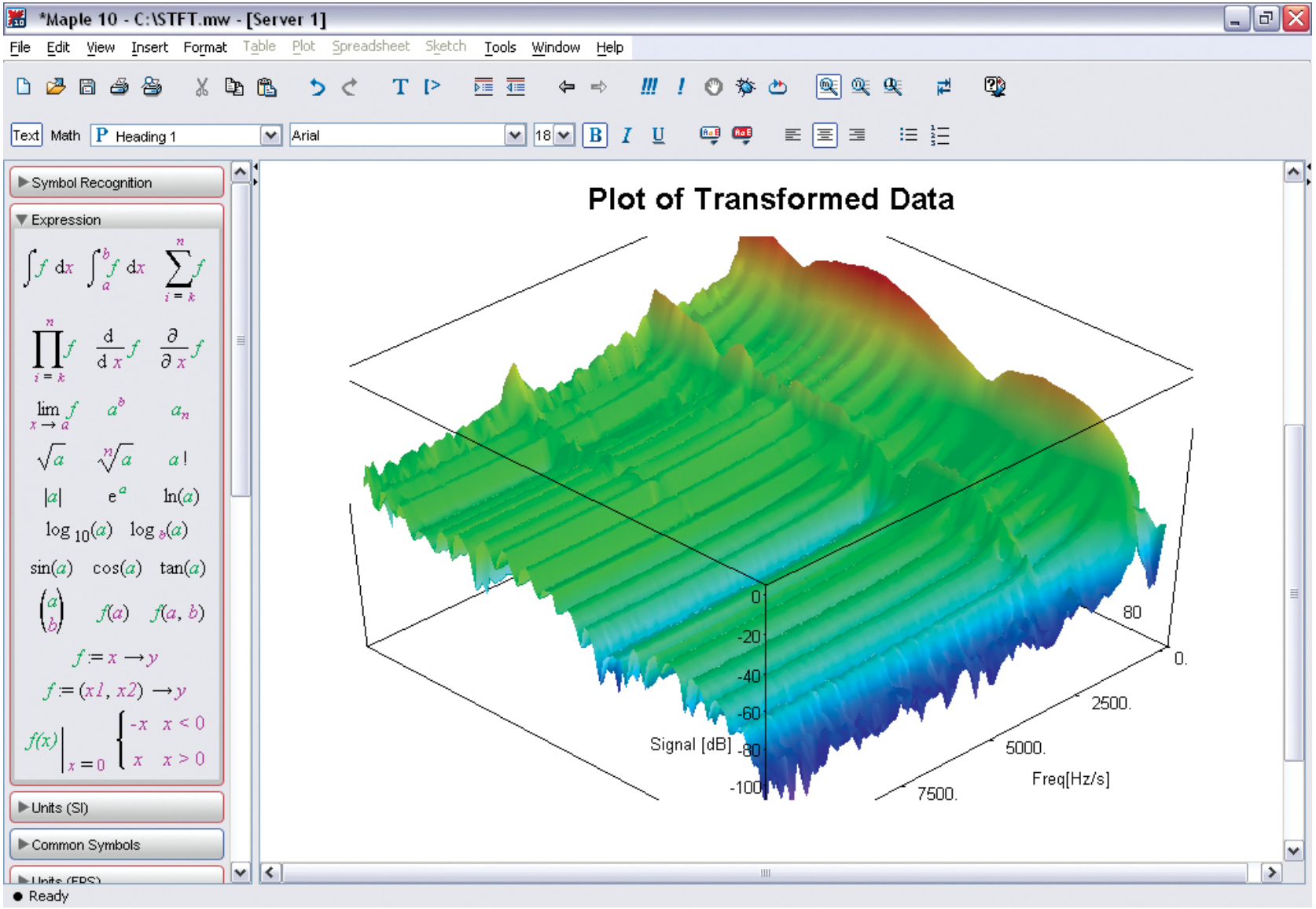
Showcase: Matemática/Estatística

- *Integrated Trading Enviroment* (http://www.efixpuls.pl/cotton2/get/sites/ite.html): visualização e análise de indicadores econômicos.



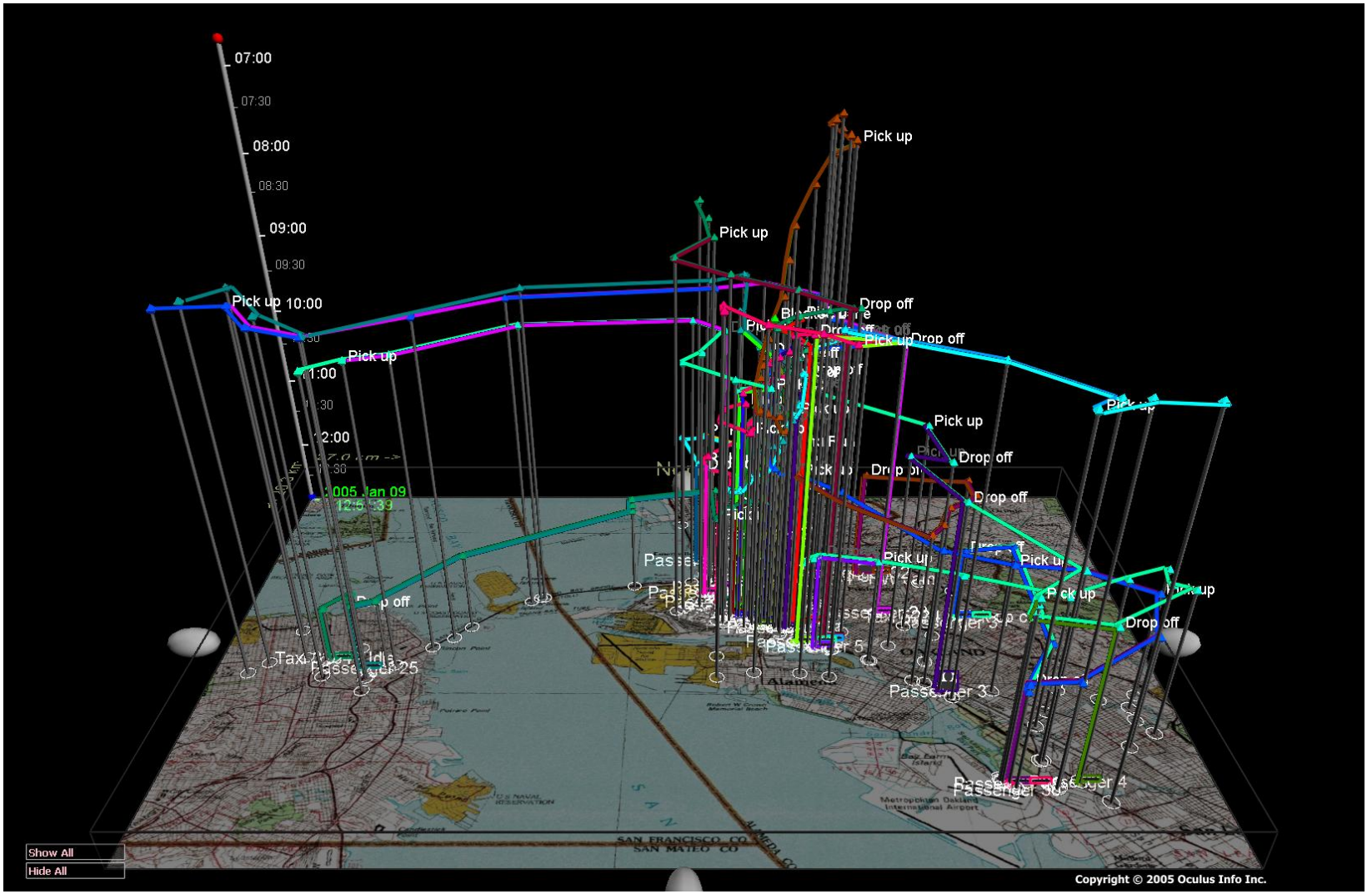
Showcase: Matemática/Visualização

- A interface do Maple (http://www.maplesoft.com/) está sendo portada para Java.



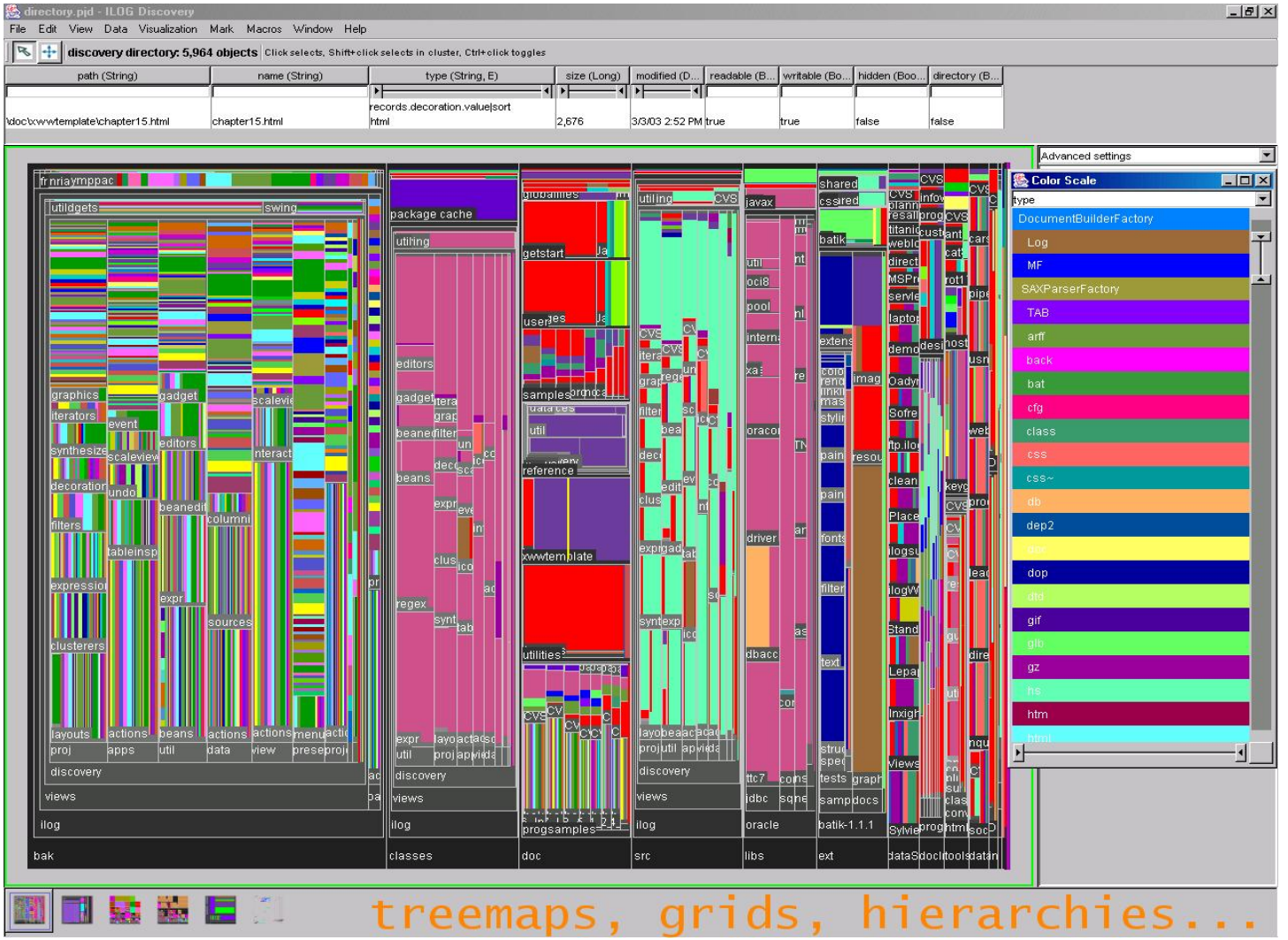
Showcase: Visualização

- Oculus (http://www.oculusinfo.com): visualização de eventos no espaço e no tempo.



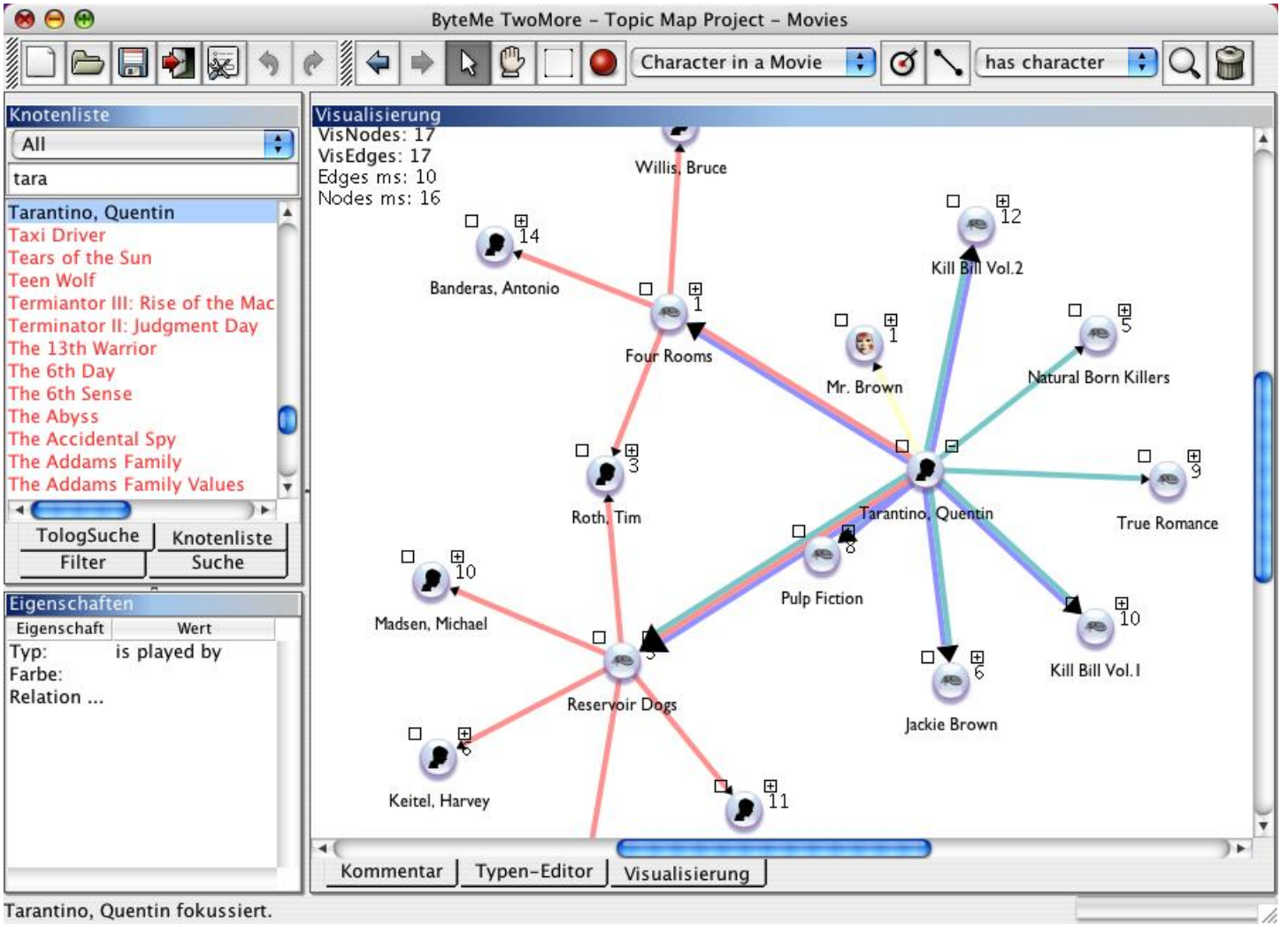
Showcase: Visualização

- *ILOG Discovery Preview* (http://www2.ilog.com/preview/Discovery): visualização científica/análise visual de dados.



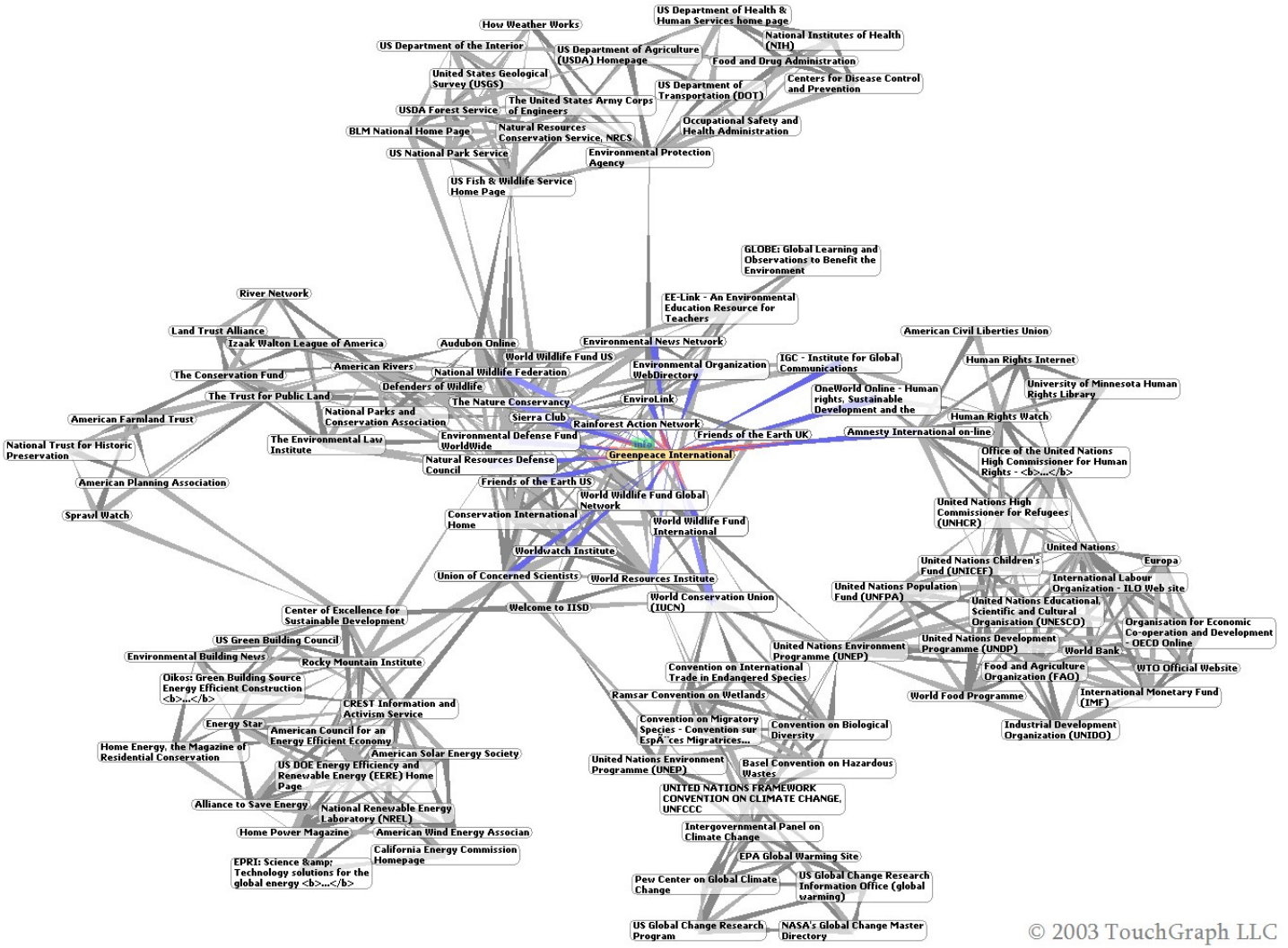
Showcase: Visualização

- *TwoMore* (http://www.pi.informatik.tu-darmstadt.de/se2004/byteme): visualização de redes semânticas.



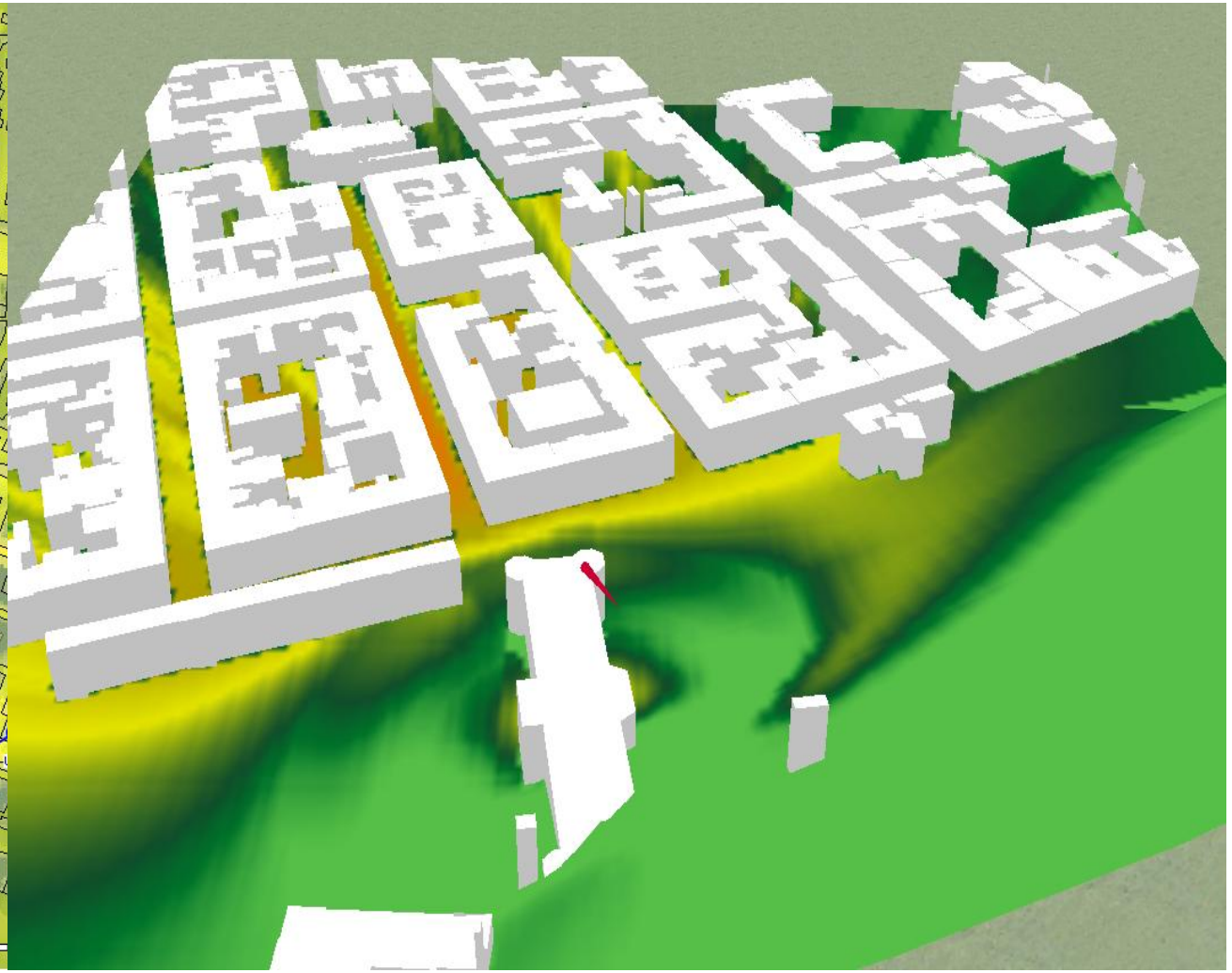
Showcase: Visualização

- *TouchGraph* (http://www.touchgraph.com/index.html): visualização de redes semânticas/de conhecimento.



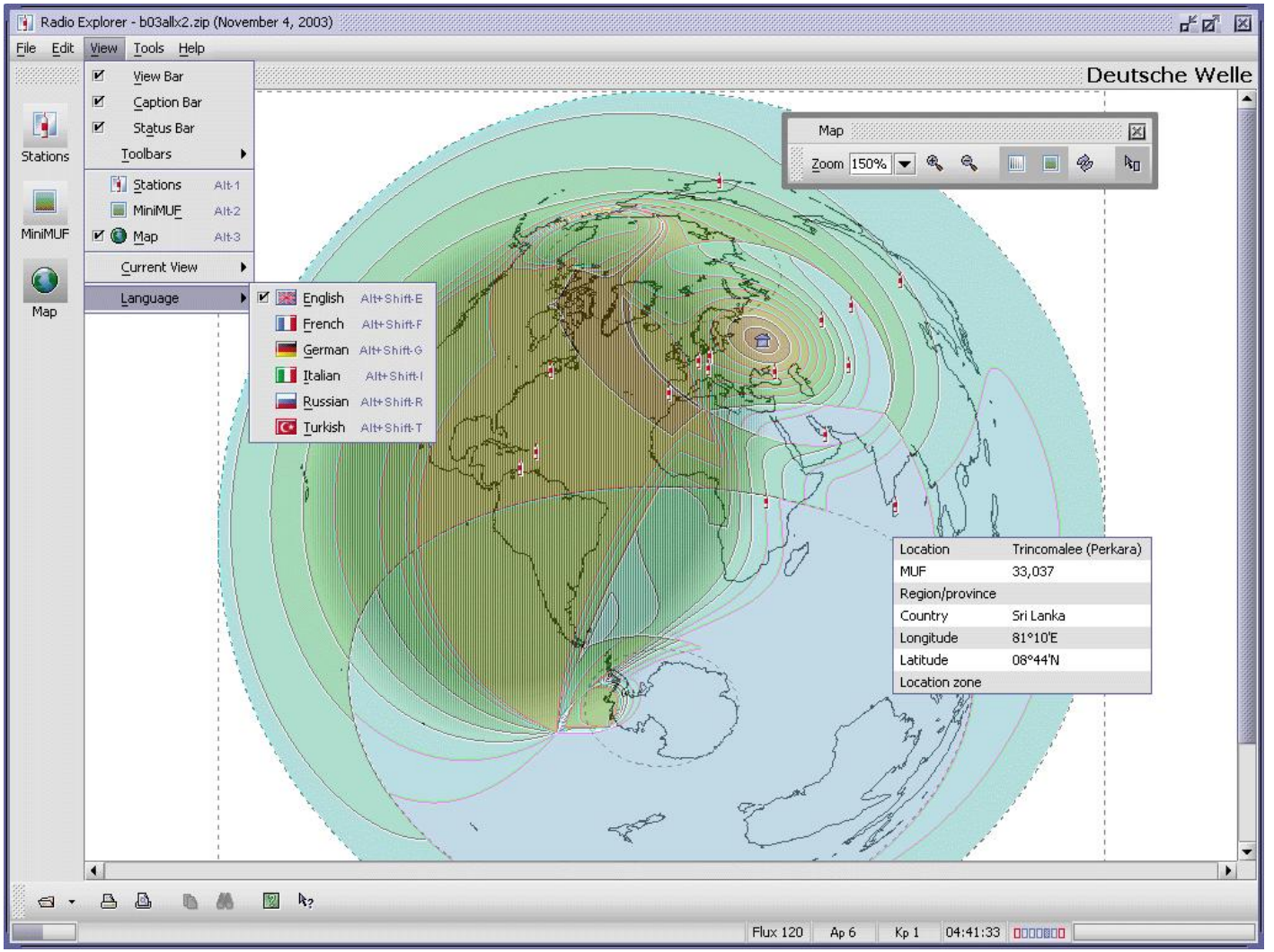
Showcase: Visualização/Modelagem

- *NIRView* (http://www.nirview.com): visualização e exploração em 2D/3D de campos de força de estações de telecomunicações.



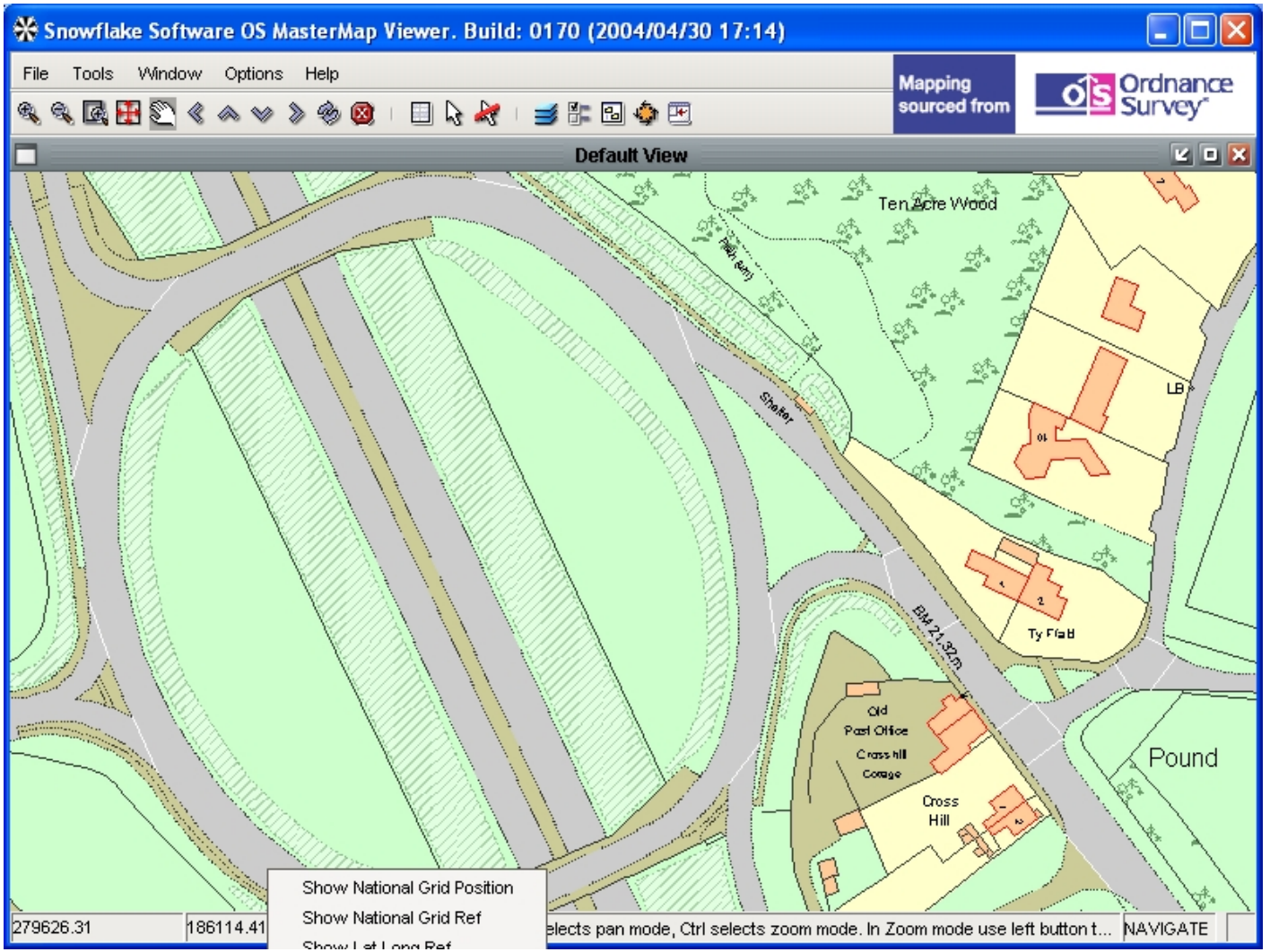
Showcase: Visualização/Modelagem

- *Radio Explorer* (http://www.radioexplorer.com.ru/en): exibe/visualiza horários de transmissão internacionais de rádio.



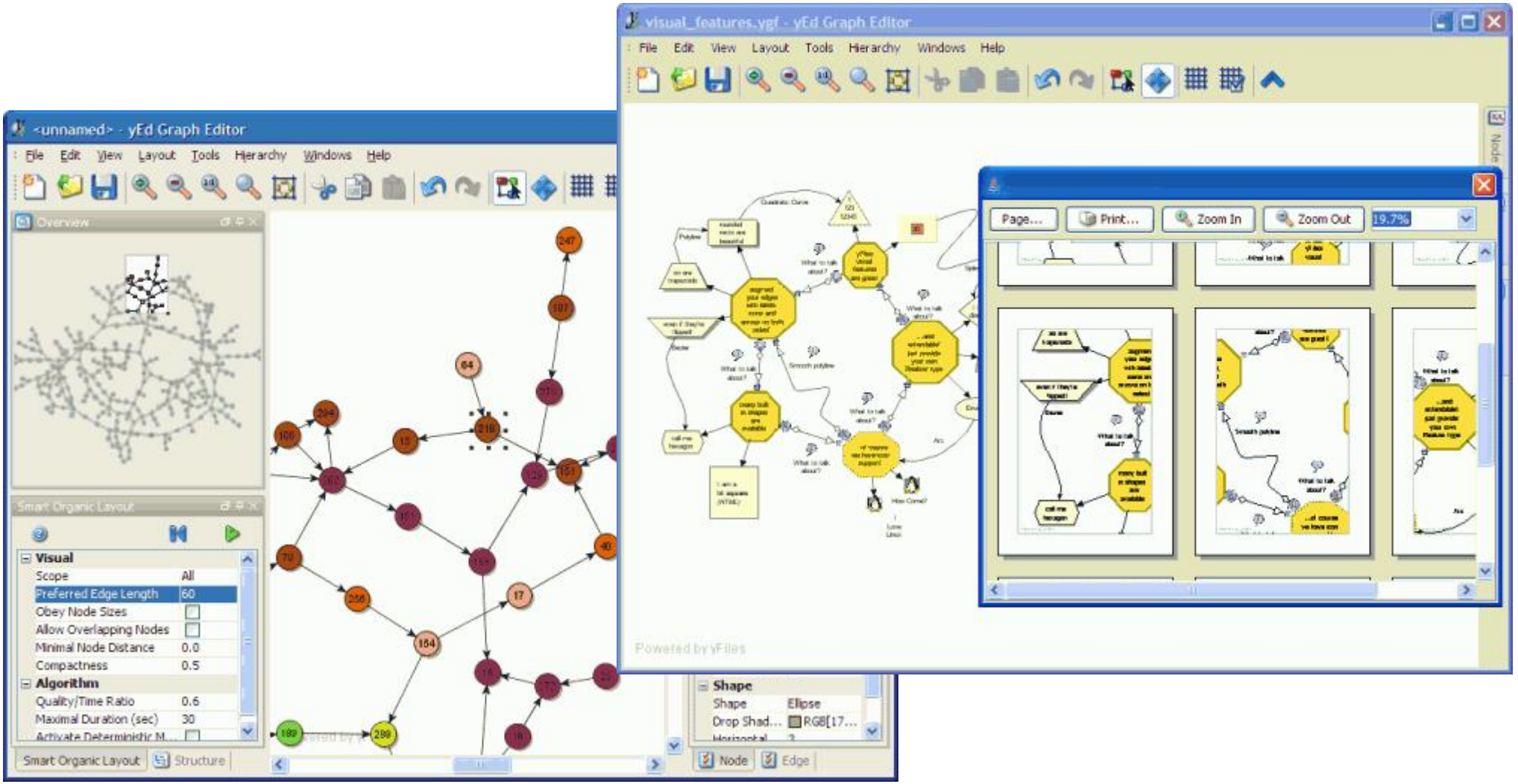
Showcase: GIS/Visualização

- *OS MasterMap Viewer* (http://www.snowflakesoft.co.uk/products/viewer): exibe/visualiza alguns tipos de arquivos de topologia/GML.



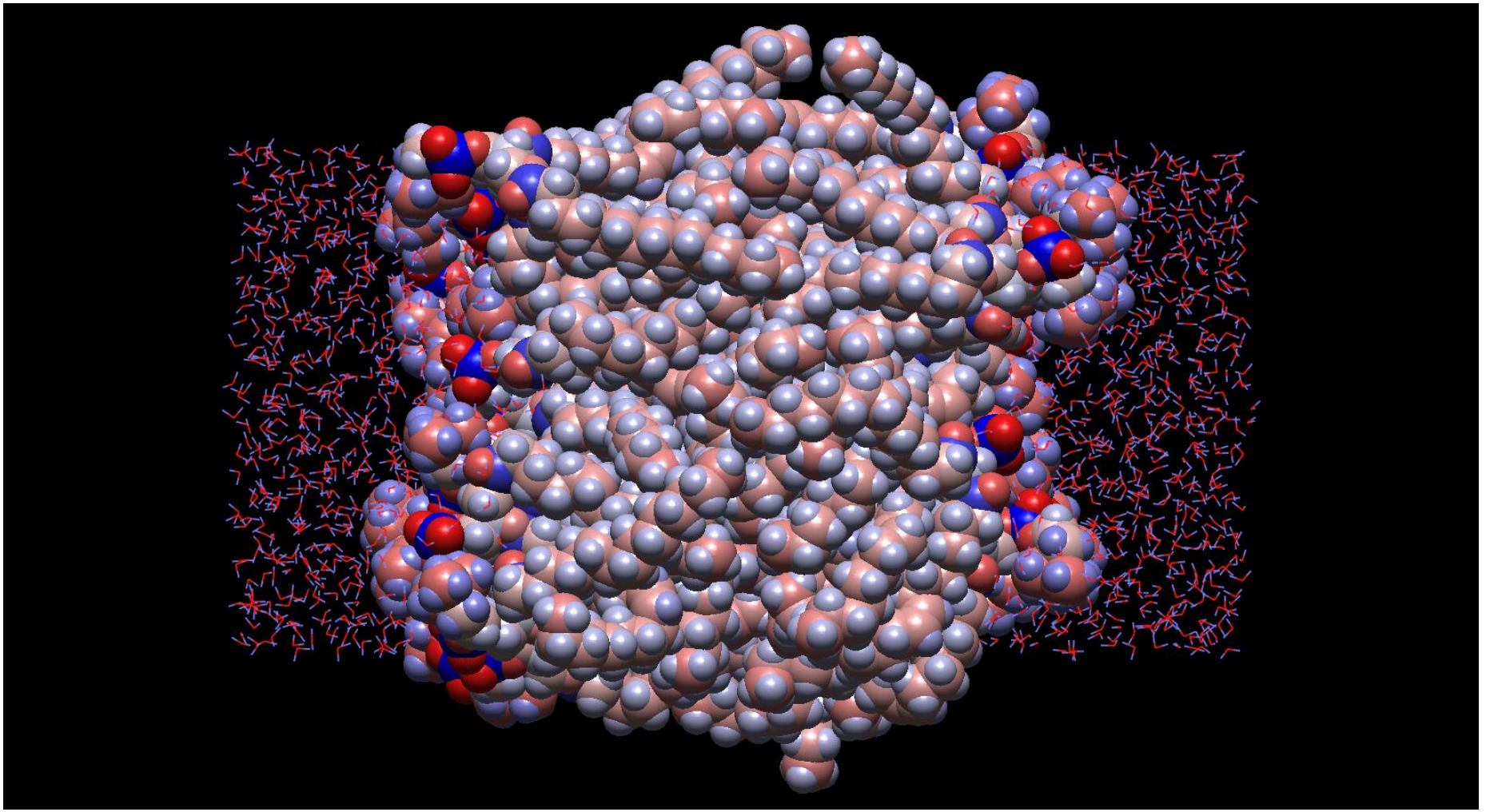
Showcase: Visualização

- *yEd* (http://http://www.yworks.com/en/products_yed_about.htm): visualização de vários tipos de dados com estrutura de grafos.



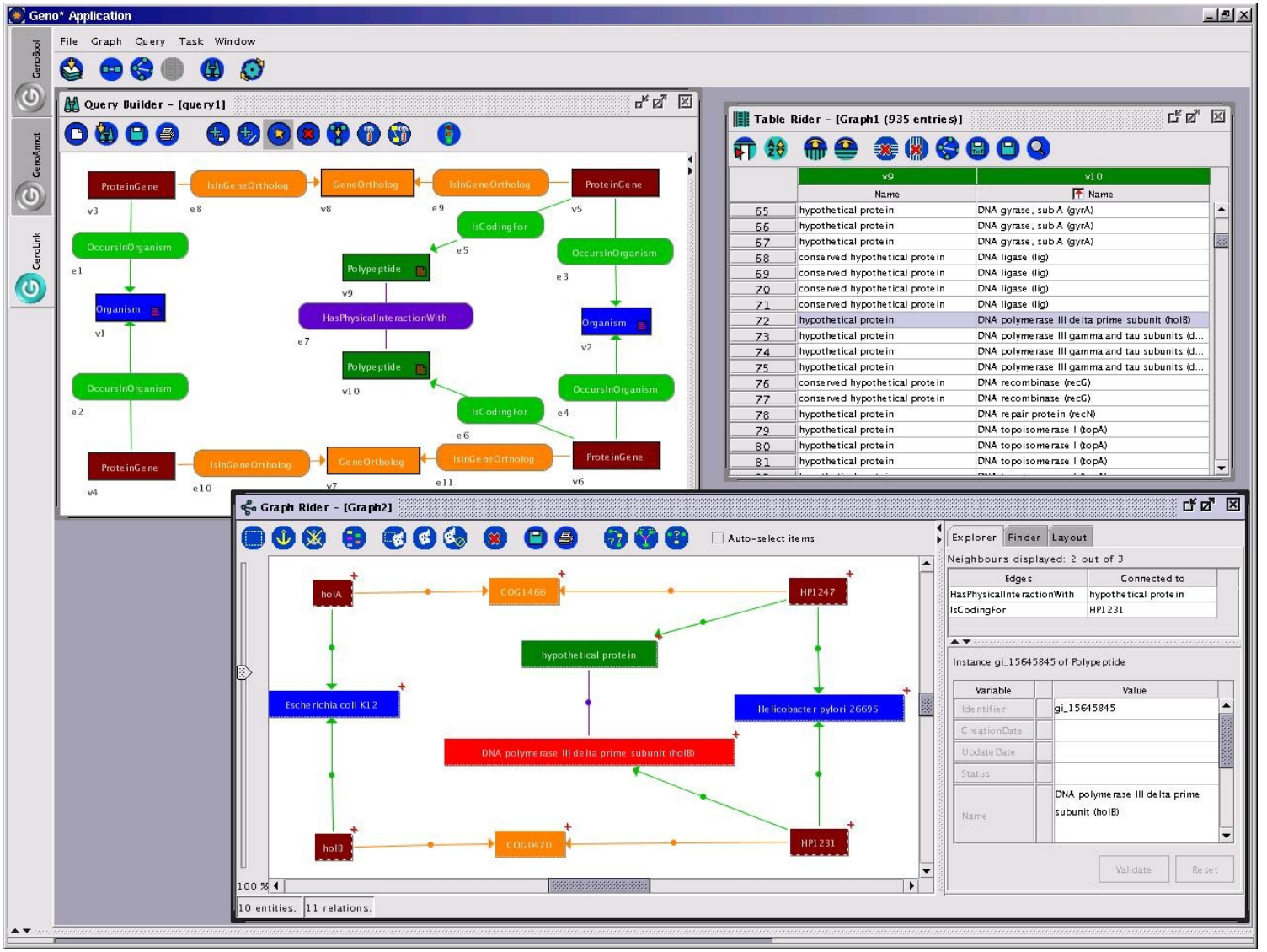
Showcase: Visualização/Modelagem

- *Force Field Explorer* (http://dasher.wustl.edu/ffe): visualização e exploração para química/biologia/nanotecnologia (pacote Tinker em Fortran)



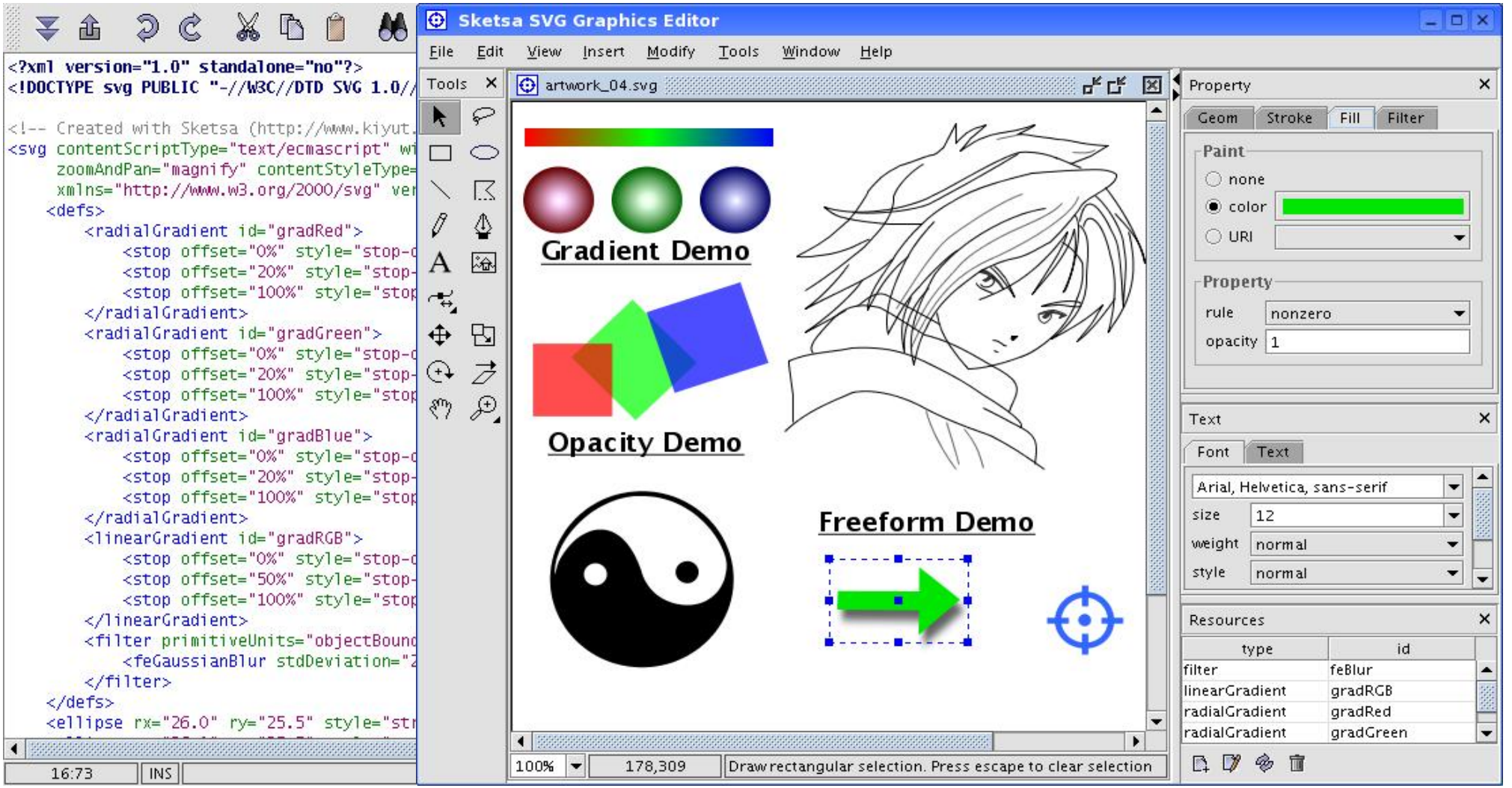
Showcase: Bioinformática

- *Genostar/GenoAnnot* (http://www.genostar.org): anotação de seqüências.



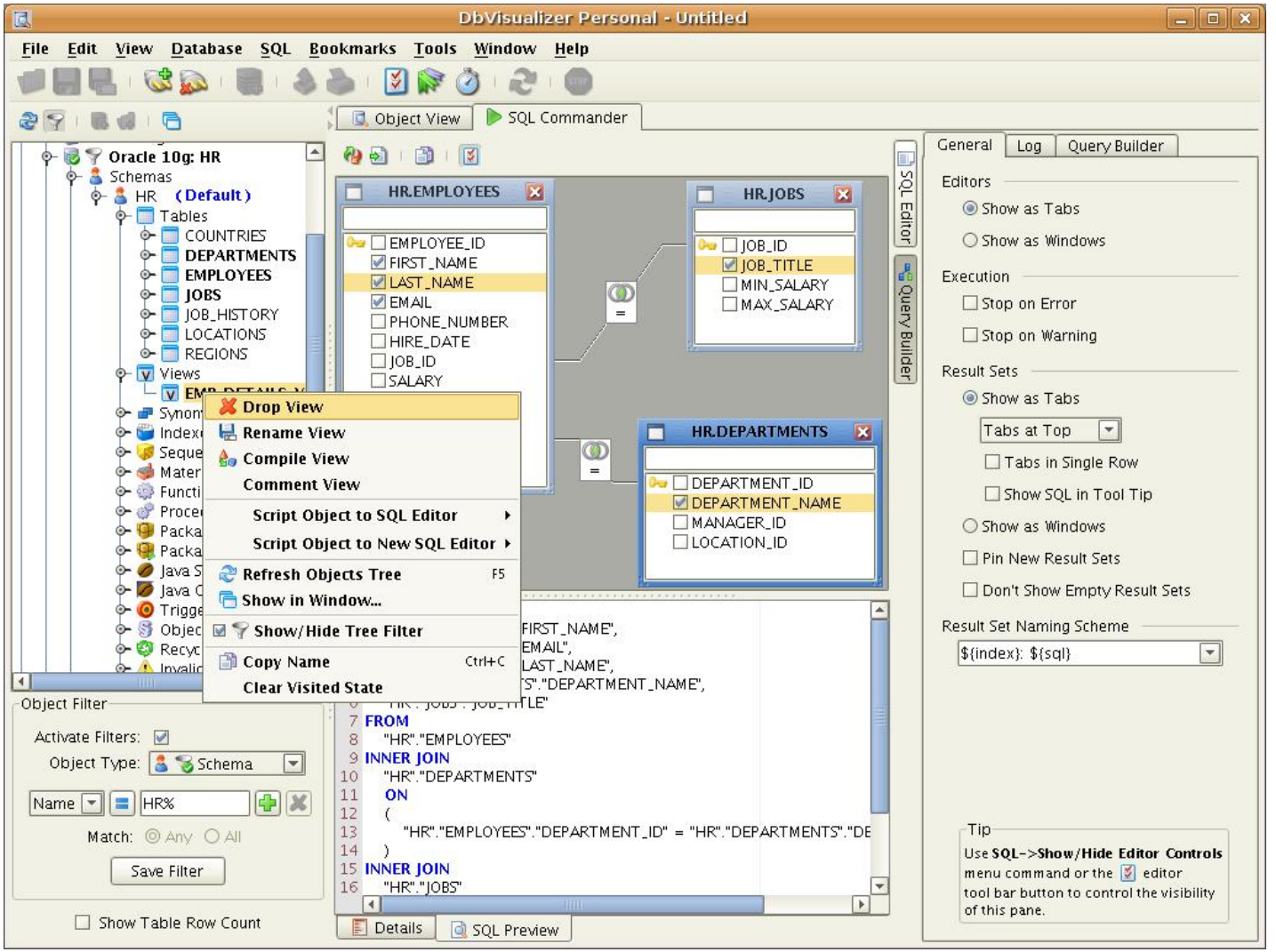
Showcase: Desenvolvimento

- *Sketsa SVG Editor* (http://www.kiyut.com/products/sketsa/index.html): Edição de arquivos SVG.



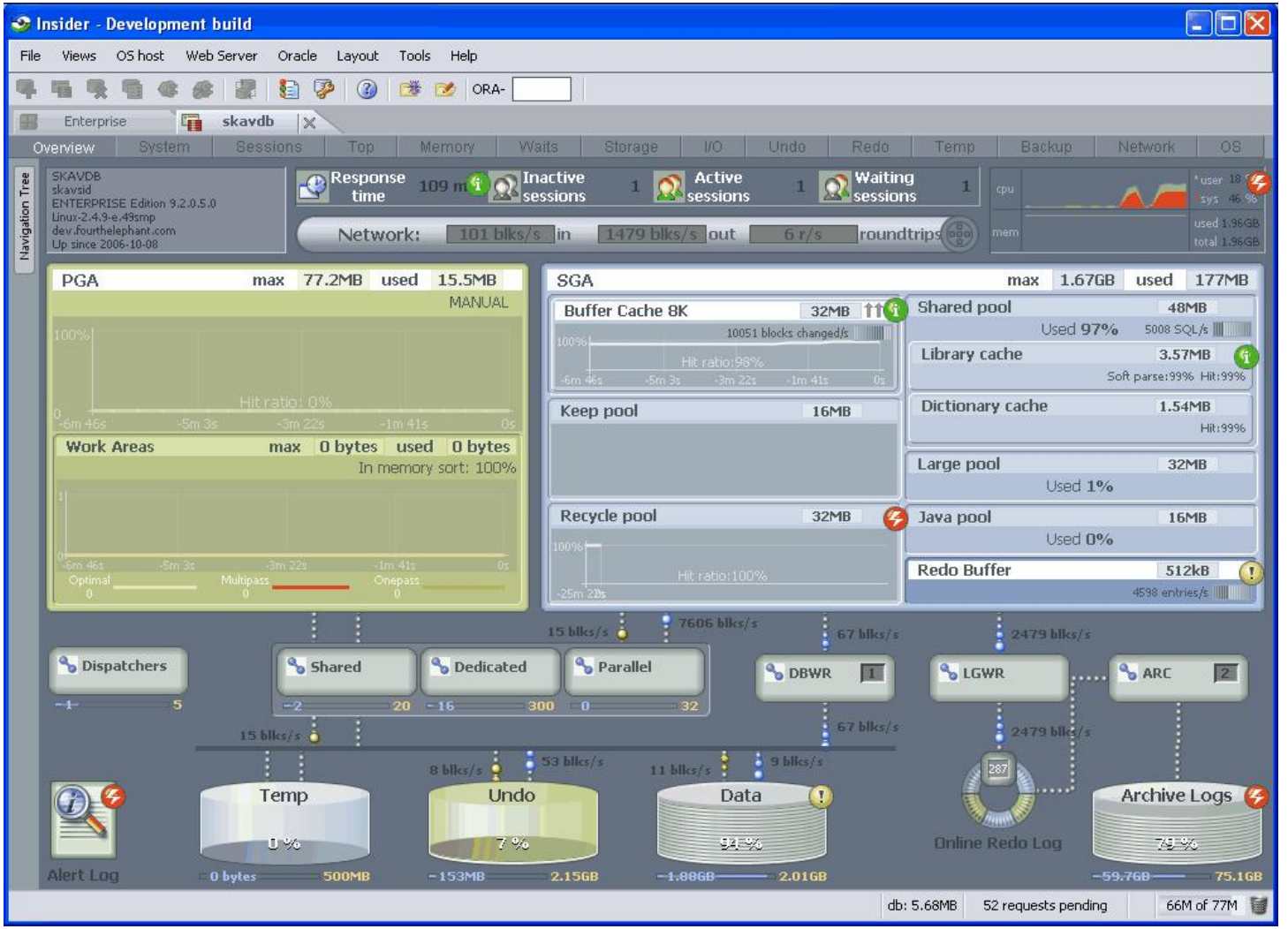
Showcase: Desenvolvimento/Sistemas

- DBVisualizer (http://www.dbvis.com/products/dbvis/index.html): criação e visualização de bancos de dados.



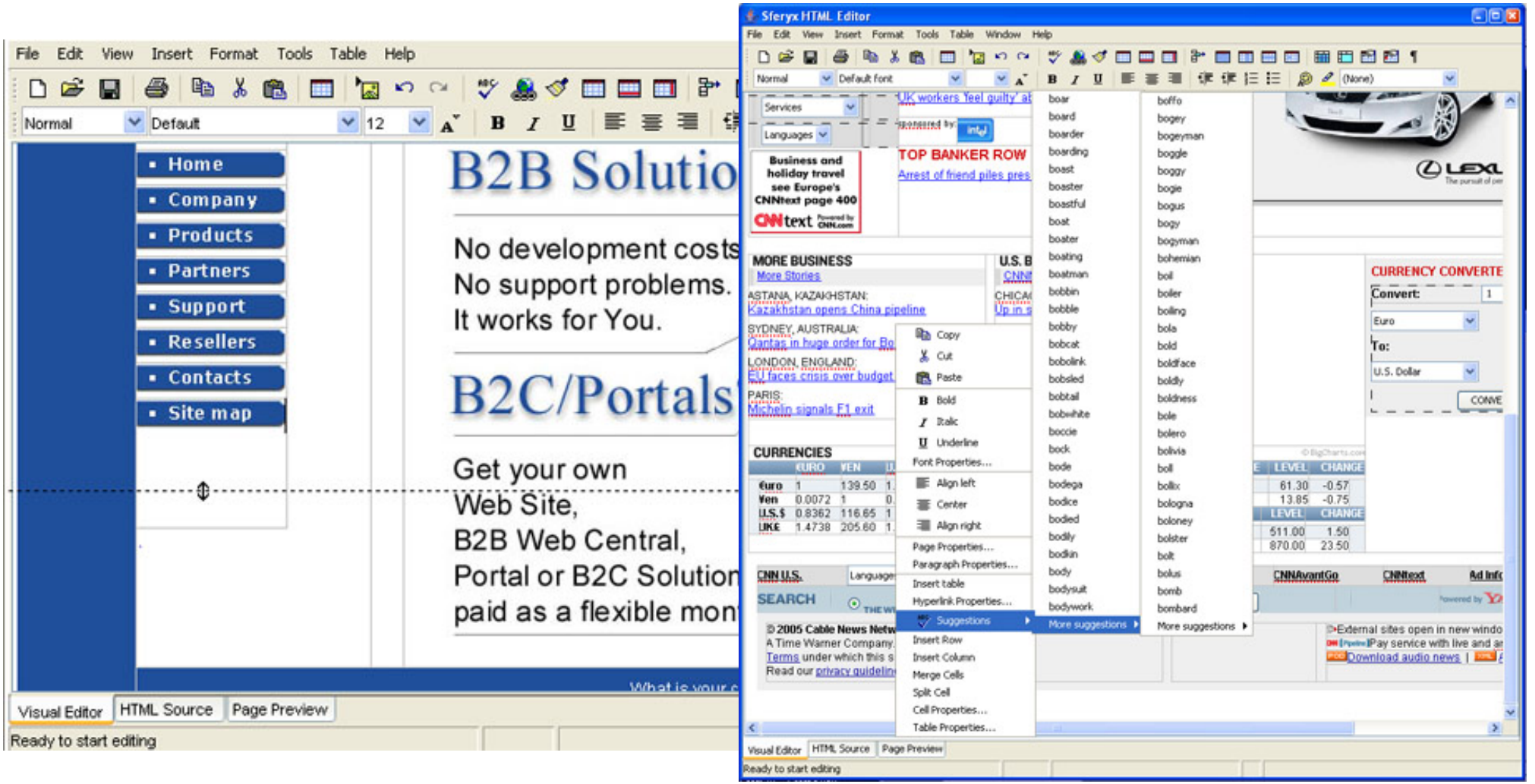
Showcase: Desenvolvimento/Sistemas

- *Insider for Oracle* (http://www.fourthelephant.com/insider.html): monitoramento e gerenciamento de bancos de dados Oracle em tempo real.



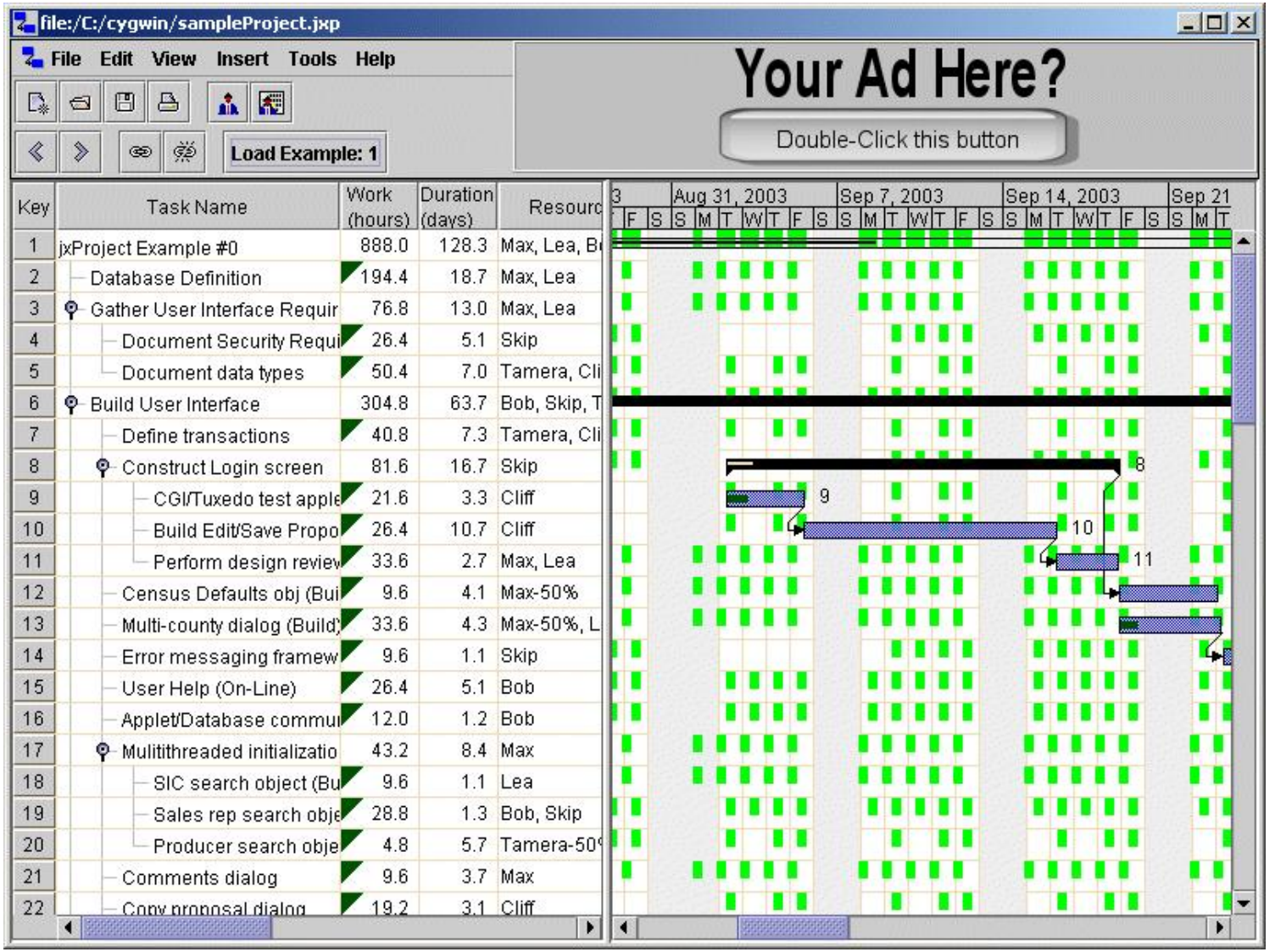
Showcase: Desenvolvimento/Sistemas

- *Sferyx JSyndrome HTMLEditor Applet Edition* (http://www.sferyx.com/index.htm): edição de HTML.



Showcase: Gerenciamento/Projetos

- *jxProject* (http://www.jxproject.com/index.shtml): gerenciamento de projetos e tarefas.



Showcase: Jogos

- *Wurm Online* (http://www.wurmonline.com): *Massively Multiplayer Online Role Playing Game*.

Showcase: Jogos

- *Puzzle Pirates* (http://www.puzzlepirates.com): *Massively Multiplayer Online Role Playing Game*.

Ferramentas para Desenvolvimento

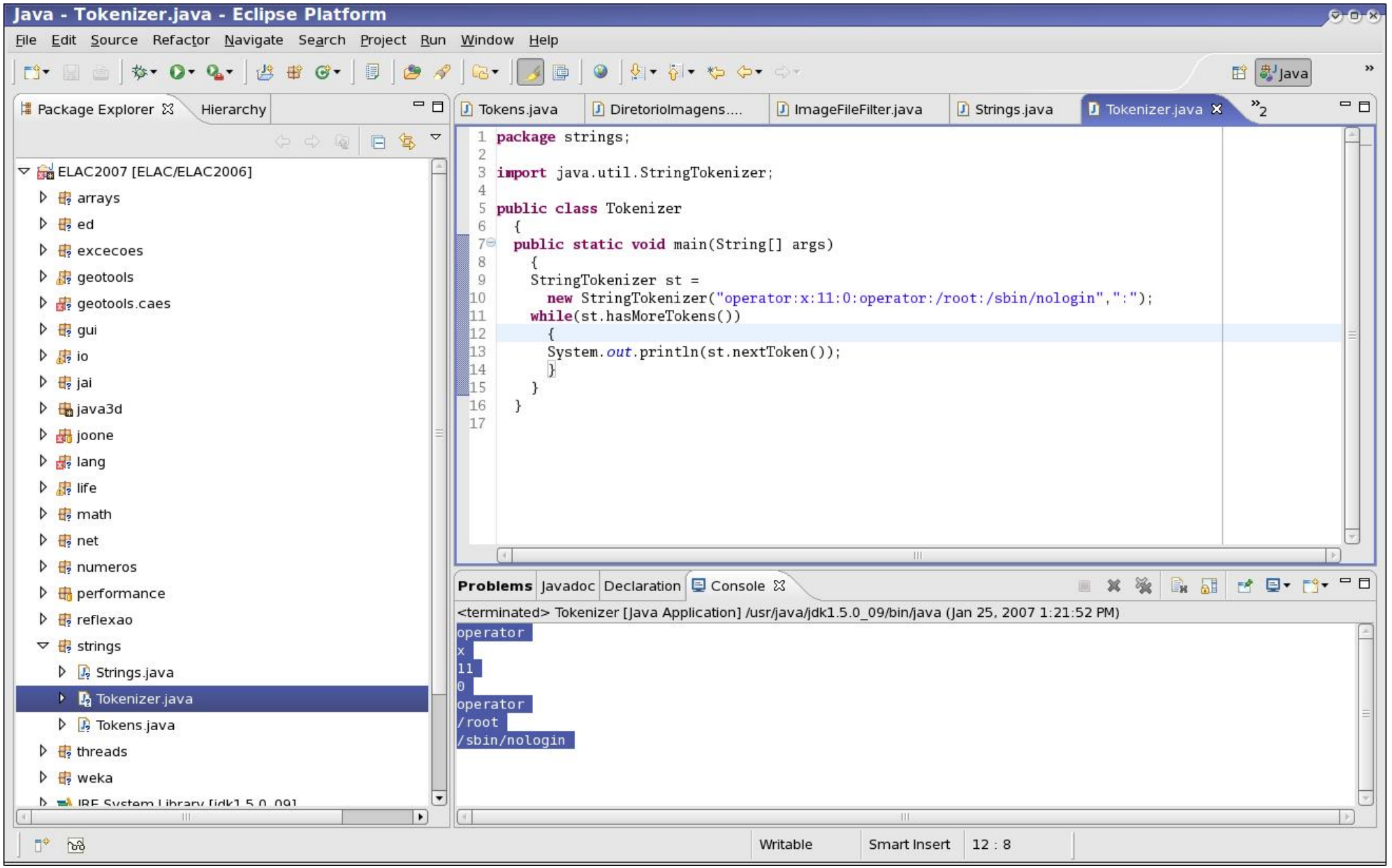
JDK

- JDK (*Java Development Kit*), pode ser baixado do site da Sun (java.sun.com)
- Têm ferramentas de linha de comando para a criação e execução de aplicações em Java.
  - Suficiente para criação de qualquer aplicação *desktop* de Java!
- JRE (*Java Runtime Environment*): somente execução de aplicações.
- Pode ser instalado juntamente com NetBeans (IDE).

Eclipse

- Eclipse: IDE gratuita, flexível (http://www.eclipse.org/downloads/)
  - Evitar download automático, para desenvolvimento de aplicações em Java somente são necessários dois arquivos: *Platform Runtime Binary* e *JDT Runtime Binary*.
  - Instalação: basta descompactar os arquivos.
- Outros módulos podem ser adicionados via Web, a partir do próprio Eclipse.

Eclipse


Java - Tokenizer.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
ELAC2007 [ELAC/ELAC2006]
arrays
ed
excecoes
geotools
geotools.caes
gui
io
jai
java3d
joone
lang
life
math
net
numeros
performance
reflexao
strings
Strings.java
Tokenizer.java
Tokens.java
threads
weka
Tokens.java
DiretorioImagens....
ImageFileFilter.java
Strings.java
Tokenizer.java
1 package strings;
2
3 import java.util.StringTokenizer;
4
5 public class Tokenizer
6 {
7 public static void main(String[] args)
8 {
9 StringTokenizer st =
10 new StringTokenizer("operator:x:11:0:operator:/root:/sbin/nologin",":");
11 while(st.hasMoreTokens())
12 {
13 System.out.println(st.nextToken());
14 }
15 }
16 }
17
Problems
Javadoc
Declaration
Console
<terminated> Tokenizer [Java Application] /usr/java/jdk1.5.0_09/bin/java (Jan 25, 2007 1:21:52 PM)
operator
x
11
0
operator
/root
/sbin/nologin
Writable
Smart Insert
12 : 8

Eclipse

- Procedimento padrão para desenvolvimento:

0. Criar um projeto (**File**, **New**, **Project**, **Java**, **Java Project**, escolhemos nome e localização, configuramos compatibilidade e versões).
1. Criar, no projeto, classes em pacotes.
2. Desenvolver: editar, compilar, testar.
3. Opcionalmente, exportar pacote como .jar (**File**, **Export**, **Java**, **JAR file**).

Eclipse: JARs

- Algumas APIs são distribuídas como um arquivo JAR.
- Selecione o projeto em **Package Explorer**, clicar com botão direito e selecionar **Properties**, **Java Build Path**, aba **Libraries**, botão **Add External JARs**.
- Selecionar o JAR e clicar em **OK** até fechar diálogos.

Eclipse: Bibliotecas

- Algumas APIs são distribuídas em *vários* arquivos JAR separados (ex. Geotools).
- Para simplificar inclusão em projetos, podemos criar *Bibliotecas* (conjuntos de JARs).
  - A partir de **Package Explorer**, clicar com botão direito e selecionar **Properties**, **Java Build Path**, aba **Libraries**, botão **Add Library**.
  - Selecionar **User Library** e **Next**.
  - Clicar em **User Libraries** e **New**.
  - Entre um nome para a biblioteca e clique **OK**.
  - Clique em **Add JARs** e adicione os JARs relevantes. Clique em **OK** e **Finish** para fechar as várias janelas.

Subversion

- *Version Control System*: permite que vários desenvolvedores compartilhem código:
  - Em um servidor para compartilhamento (usando Apache e WebDAV).
  - Com *versioning* (arquivos, diretórios, metadados têm versões associadas).
  - Com diferentes tipos de autorização (para diferentes perfis de usuários).
  - Possibilitando *branching* de código.
- Criado para ser uma versão melhor de CVS.
- Para este curso, nos interessa mais como repositório de código.

Subversion

- Operações:
  - Criação de repositórios no servidor (com usuários e permissões).
  - Criação de projetos no Eclipse (com *plug-in* Subclipse).
  - *Commit* de projetos no servidor.
  - *Check-out* de projetos do servidor.
- No momento não estou realmente preocupado com *versioning* e *branching*.
  - Maior interesse é ter a capacidade de colocar código em um repositório global.
  - Repositório pode ser *read-only* para a comunidade.

Subversion

- Criando um servidor Subversion:
  - Usar Linux, Apache, mod_dav_svn (sugestão: instalar como pacotes compilados).
  - Criar um diretório-base (por exemplo, /home/svn/repos) e mapear com uma diretiva <Location> no Apache.

```
<Location /svn/repos>
DAV svn
SVNParentPath /home/svn/repos
# Política de acessos por diretorio
AuthzSVNAccessFile /home/svn/users/svnauthz.conf
Satisfy Any
Require valid-user
# Como autenticaremos?
AuthType Basic
AuthName "LAC Subversion Repository"
AuthUserFile /home/svn/users/passwords
</Location>
```

Subversion

- Criando um servidor Subversion:
  - Criar um repositório, por exemplo, **ELAC**.

```
svnadmin create /home/svn/repos/ELAC
chown -R apache.apache /home/svn/repos/ELAC
```

  - Criar um usuário com permissões.

```
htpasswd -b /home/svn/users/passwords rafael ********
```

  - Adicionar o usuário em /home/svn/users/svnauthz.conf

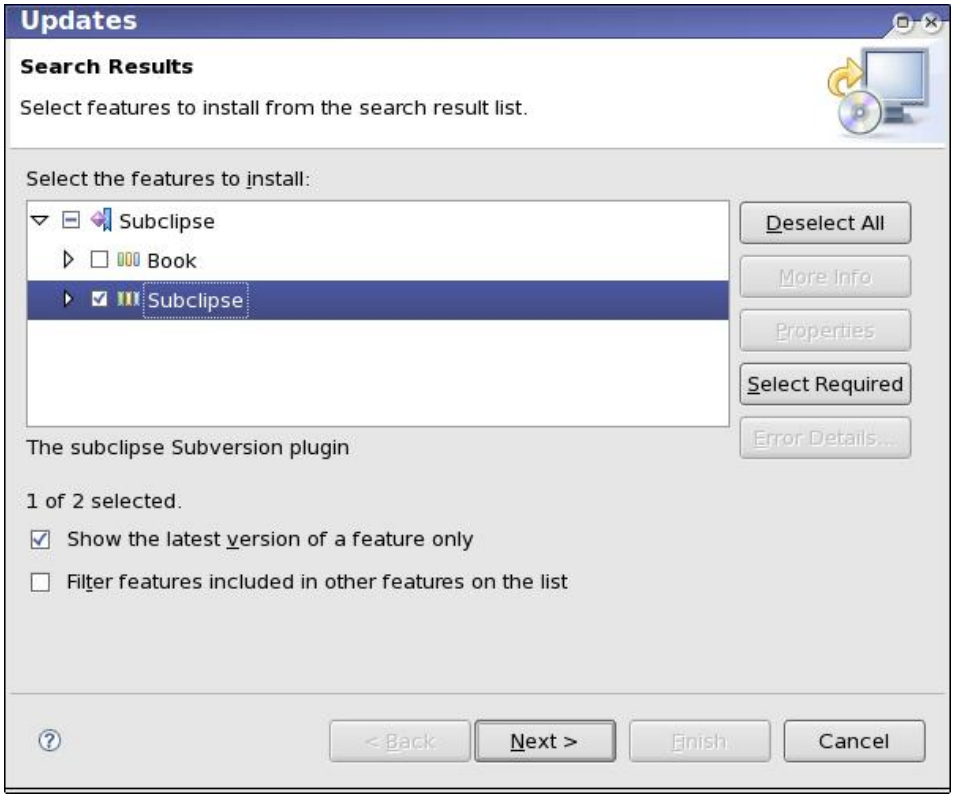
```
[/]
* = r
[ELAC:/]
rafael = rw
* = r
```

Subversion

- Testando...

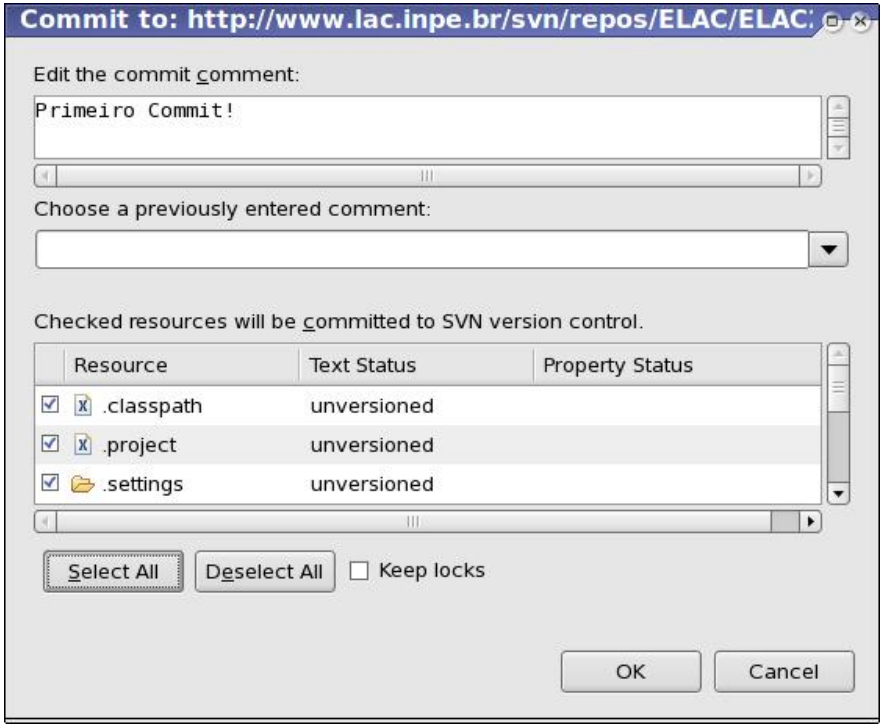
Subversion/Eclipse

- Instalando Subclipse no Eclipse:
  - **Help/Software Updates/Find and Install**.
  - **Search for new features to install** / **Next**.
  - **New Remote Site**: entrar **Subclipse** e **http://subclipse.tigris.org/update**.
  - Marcar **Subclipse** e clicar em **Finish**.
  - Selecionar **Subclipse/Subclipse** / **Next**.
  - Aceitar os termos da licença / **Next** / **Finish** / **Install All**.
  - Reinicializar o Eclipse.



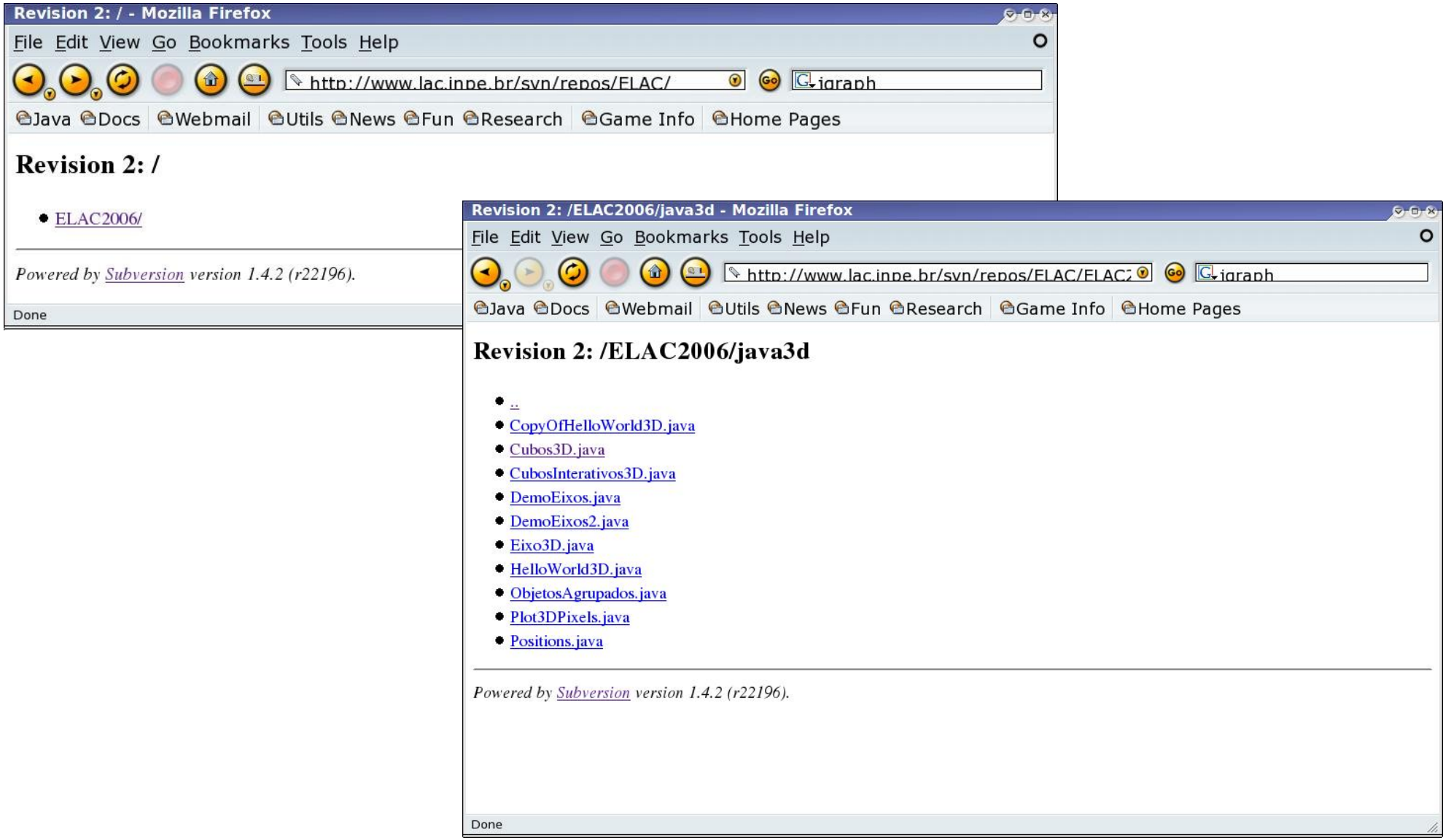
Subversion/Eclipse

- Associando projetos a repositórios (criação e *commit*):
  - Selecionar um projeto (com código/arquivos associados).
  - Clicar no projeto com botão esquerdo, selecionar **Team** / **Share Project**.
  - Escolher **SVN** / **Next**.
  - Como URL, colocar a URL associada ao *subversion* (inclusive com repositório) / **Finish**.
  - Pede autorização.
  - Pede comentário para primeiro *commit*. Clique também em **Select All** e OK.



Subversion/Eclipse

- Verificando...



Subversion/Eclipse

- Associando projetos a repositórios (*checkout*):
  - Em uma outra instalação do Eclipse (com Subclipse instalado), selecionar **File** / **Import** / **Other** / **Checkout Projects from SVN** / **Next**.
  - Selecionar local do repositório (ou criar novo) / **Next**.
  - Selecionar o nome do projeto / **Next**.
  - Escolher o nome do projeto local / **Finish**.
- O projeto pode ser compilado, modificado, etc. localmente.
- Só podemos fazer um *commit* do projeto se tivermos permissões e senhas.
  - Neste caso, clicar no projeto com botão direito, **Team** / **Commit** e seguir instruções.
- Bibliotecas externas ***devem*** estar disponíveis no cliente.
  - Podemos fazer algumas ser parte do pacote...

Brevíssimo Curso de Orientação a Objetos

Princípios básicos de orientação a objetos

- **Encapsulamento**
- Atributos e funções relevantes a um domínio ou problema são *encapsulados* em uma classe de forma que:
  - Somente atributos e funções relevantes sejam representados;
  - Uma interface seja criada para que usuários/desenvolvedores tenham acesso somente a funções e atributos que podem ser acessados diretamente.

Princípios básicos de orientação a objetos

- **Herança**
- Uma classe pode ser descrita de forma incremental usando classes já existentes.

Princípios básicos de orientação a objetos

- **Polimorfismo**
- Classes que tem relação de herança podem ser processadas de forma igual e transparente (com algumas condições).
- Relação *é-um-tipo-de*.

Princípios básicos de orientação a objetos

- **Classes** são descritas em Java para representar modelos ou conceitos.
- **Objetos** ou **Instâncias** são criados ou instanciados a partir das classes.
  - Criação pela palavra-chave new.
  - **Referências** são usadas para acesso aos objetos.
- Uma classe pode ser usada para criar muitos objetos.
  - Os atributos de cada objeto serão independentes*

Princípios básicos de orientação a objetos

- **Classes** contém campos ou atributos e métodos.
- **Atributos** são usados para armazenar estado (valores) relacionados com as instâncias das classes.
- **Métodos** são operações que podem ser realizadas com estes atributos.
  - **Construtores** são métodos declarados de forma especial, serão os primeiros* a ser executados quando instâncias forem criadas.
- Em princípio, ocultamos os atributos mas permitimos a operação via métodos.

Encapsulamento: Um Exemplo Simples

- Conta bancária:
  - Queremos armazenar os dados da conta.
  - Queremos ser capazes de efetuar operações com os dados da conta (alguns podem ser modificados, outros não, alguns ainda somente sob certas condições)...
- Não é científico, mas exemplifica bem as idéias.

Encapsulamento: Um Exemplo Simples

```
class ContaBancaria1
  {
  String nomeCorrentista;
  double saldo;

  ContaBancaria1(String n, double s)
    {
    nomeCorrentista = n;
    saldo = s;
    }

  double getSaldo()      {    return saldo;    }

  String getNome()       {    return nomeCorrentista;     }

  void deposita(double quantia)
    {
    saldo = saldo + quantia;
    }

  void retira(double quantia)
    {
    if (quantia < saldo) saldo = saldo - quantia;
    }
  }
```

Encapsulamento: Um Exemplo Simples

```
class UsaContaBancaria1
  {
  public static void main(String[] args)
    {
    ContaBancaria1 fred = new ContaBancaria1("Fred",1000);
    ContaBancaria1 richard = new ContaBancaria1("Richard",2000);
    richard.retira(500);
    fred.deposita(500); // ok
    richard.saldo = 1000000; // oops
    }
  }
```

Encapsulamento: Um Exemplo Simples

```
public class ContaBancaria2
  {
  private String nomeCorrentista;
  private double saldo;

  public ContaBancaria2(String n, double s)
    {
    nomeCorrentista = n;
    saldo = s;
    }

  public double getSaldo()  { return saldo; }

  public String getNome()   { return nomeCorrentista; }

  public void deposita(double quantia)
    {
    saldo = saldo + quantia;
    }

  public void retira(double quantia)
    {
    if (quantia < saldo) saldo = saldo - quantia;
    }
  }
```

Encapsulamento: Um Exemplo Simples

```
class UsaContaBancaria2
  {
  public static void main(String[] args)
    {
    ContaBancaria2 fred = new ContaBancaria2("Fred",1000);
    ContaBancaria2 richard = new ContaBancaria2("Richard",2000);
    richard.retira(500);
    fred.deposita(500); // ok
    richard.saldo = 1000000; // haha
    }
  }
```

Princípios básicos de orientação a objetos

- Classes podem ser organizadas em pacotes.
- Existem regras de visibilidade que indicam que atributos e métodos são visíveis...
  - Entre classes do mesmo pacote.
  - Entre classes de diferentes pacotes.
- Dentro de uma classe, todos os métodos e atributos são visíveis.
- Classes podem, evidentemente, usar instâncias de outras classes (e até dela mesma!)

- **Tudo** são classes (exceto atributos de tipos nativos).
- **Nada** é declarado independente (fora) de uma classe.
- Estrutura geral:
  1. Declaração de pacotes;
  2. Importação de classes externas;
  3. Declaração da classe e atributos (ex. extensão).
  4. Declaração de campos e métodos.
- Boas práticas:
  - Cada classe em um arquivo com nome igual e extensão .java;
  - Atributos declarados antes dos métodos;
  - Indentação e comentários!

Um exemplo de classe

```
package applets;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

public class Circulo
   {
   protected Color cor;
   protected int x,y;
   protected int raio;

   public Circulo(Color c,int x,int y,int r)
      {
      cor = c;
      this.x = x; this.y = y; raio = r;
      }

   public void draw(Graphics2D g)
      {
      g.setColor(cor);
      Ellipse2D.Float circ = new Ellipse2D.Float(x-raio,y-raio,raio*2,raio*2);
      g.draw(circ);
      }

   }
```

Um exemplo de classe

- No exemplo anterior vimos...
  - Declaração de pacote.
  - Importação de classes externas necessárias.
  - Declaração da classe.
  - Atributos e classe com declarações de visibilidade.
  - Construtor.
  - Métodos.

Um exemplo de classe com herança

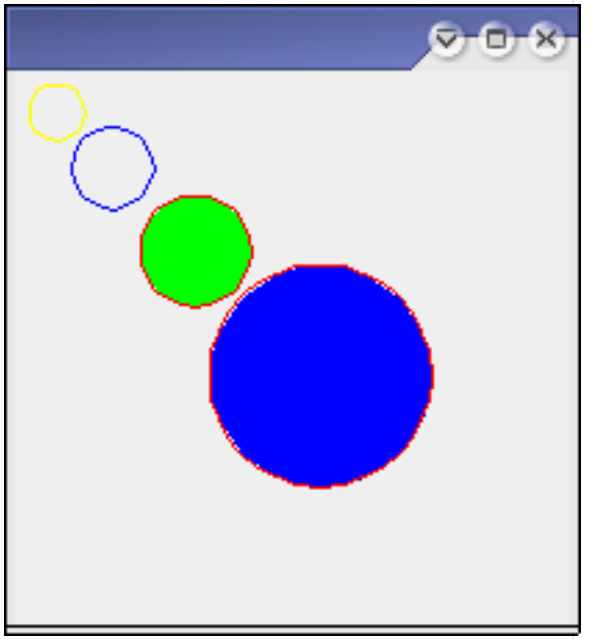
```
package applets;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

public class CirculoPreenchido extends Circulo
   {
   protected Color corP;

   public CirculoPreenchido(Color c,Color p,int x,int y,int r)
      {
      super(c,x,y,r);
      corP = p;
      }

   public void draw(Graphics2D g)
      {
      g.setColor(corP);
      Ellipse2D.Float circ = new Ellipse2D.Float(x-raio,y-raio,raio*2,raio*2);
      g.fill(circ);
      super.draw(g);
      }

   }
```

Um exemplo de classe com herança

- No exemplo anterior vimos...
  - Declaração de classe que herda de outra.
  - Execução de construtor ancestral (*super*).
  - Acesso a campos herdados (diretamente).
  - Acesso a métodos herdados (*super*).
  - Sobrecarga de métodos (métodos com mesmo nome nas duas classes).

Exemplo mais complexo

```
package applets;

import java.awt.*;
import javax.swing.*;

public class TestaCirculo extends JComponent
  {

  public Dimension getPreferredSize()
    {
    return new Dimension(200,200);
    }

  public void paintComponent(Graphics g)
    {
    Graphics2D g2d = (Graphics2D)g;
    Circulo x = new Circulo(Color.YELLOW,15,15,10); x.draw(g2d);
    new Circulo(Color.BLUE,35,35,15).draw(g2d);
    CirculoPreenchido y = new CirculoPreenchido(Color.RED,Color.GREEN,65,65,20); y.draw(g2d);
    Circulo z = new CirculoPreenchido(Color.RED,Color.BLUE,110,110,40); z.draw(g2d);
    }

  public static void main(String[] args)
    {
    JFrame f = new JFrame();
    f.add(new TestaCirculo());
    f.setVisible(true);
    f.pack();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
  }
```



Exemplo mais complexo

- No exemplo anterior vimos...
  - Mais herança.
  - Métodos “mágicos” (getPreferredSize, paintComponent) sobreescritos.
  - Método “mágico” main.
  - Instâncias anônimas!
  - Polimorfismo (exemplo bem simples).

Finalizando

Referências

- *Showcase*: a maioria de java.sun.com/products/jfc/tsc/sightings
- JDK: java.sun.com
- Eclipse: www.eclipse.org
- Subversion: svnbook.red-bean.com/, dicas de instalação em www.ferdychristant.com/blog/articles/DOMM-6NFJ6J

Perguntas?