



INVISIP, IST-2000-29640

# INVISIP

## IST-2000-29640

### Information Visualisation for Site Planning

#### WP 4: Analyser

#### D 4.3: Evaluation Report of Demonstrator

#### Version 2

Deliverable Type	Technical Report
Dissemination level	CO
Date	19.12.2003
Due date	31.12.2003
Version	V1.0
Authors	Marangon, Piccazzo, Albertoni, Bertone, Podolak
Institution	DAPP
WP coordinator	DAPP

**Table of content:**

<b>1</b>	<b>Lists of Tables and Figures</b>	<b>4</b>
1.1	List of Figures	4
<b>2</b>	<b>Introduction</b>	<b>6</b>
2.1	Control Unit	7
2.2	Visual Data Mining	9
2.2.1	A VDMTOOL TO ANALYSE METADATA	9
2.2.2	LAND PRICE PREDICTION MODULE (DM FOR DATA MINING)	10
2.3	Configurator	11
<b>3</b>	<b>The Control Unit</b>	<b>12</b>
3.1	The layout of INVISIP site	12
3.2	The client side	15
3.2.1	WORKFLOW AND SESSION	17
3.2.2	CUSTOMIZING THE WORKFLOW	21
3.2.3	IMPLEMENTATION TECHNOLOGIES	26
3.2.4	ACCESSING JAVA FROM JAVASCRIPT	28
3.2.5	ACCESSING JAVASCRIPT FROM JAVA	29
3.2.6	CONNECTING CONTROL UNIT TO A COMPONENT DISPLAYES IN THE SITE_PLANNING_FRAME	30
3.2.7	CONTROL UNIT COMMANDS	33
3.3	The server side	39
<b>4</b>	<b>Visual Data Mining</b>	<b>42</b>
4.1	VDM-module	42
4.1.1	THE ARCHITECTURE OF THE VDMTOOL	46
4.1.2	AN APPLICATION EXAMPLE	48
4.2	Land Price Prediction	50
4.2.1	THE DECISION TREE METHODOLOGY	51
<b>5</b>	<b>Configurator</b>	<b>55</b>
5.1	The Configurator module	55

<b>6</b>	<b>Summary / Conclusion</b>	<b>58</b>
<b>7</b>	<b>Appendix 1</b>	<b>59</b>
7.1	The new VDM Module Design	59
7.2	New VDM module classes	59
<b>8</b>	<b>Appendix 2</b>	<b>61</b>
8.1	Case study	61
<b>9</b>	<b>References</b>	<b>64</b>
9.1	Literature	64

# 1 Lists of Tables and Figures

## 1.1 List of Figures

1	Figure 1 - Framework of INVISIP .....	6
2	Figure 2 – Data flows during the site planning analysis.....	9
3	Figure 3 - Analyser interface.....	12
4	Figure 4 – The frames in the HomePage of the INVISIP website .....	13
5	Figure 5 – INVISIP website when Control_Frame is hidden.....	14
6	Figure 6 – The login page .....	15
7	Figure 7 – Planning process page.....	16
8	Figure 8 – The session menu.....	16
9	Figure 9 – List of planning process.....	18
10	Figure 10 – Generic workflow .....	19
11	Figure 11 – The tree representing the workflow.....	20
12	Figure 12 – The open dialog.....	21
13	Figure 13 – The „Save as“ dialog .....	21
14	Figure 14 - Customizing the workflow .....	22
15	Figure 15 – Delete a task.....	23
16	Figure 16 – The user read „Search for documents“ .....	23
17	Figure 17 – The user modifies the name of the node .....	24
18	Figure 18 – The user changes the name of the node.....	24
19	Figure 19 – The description of a node.....	25
20	Figure 20 – The user selects „Change description“ .....	25
21	Figure 21 – Window for the insertion of description .....	26
22	Figure 22 – The library.....	26
23	Figure 23 – Connections among JavaScript, Applet and Plugin .....	29
24	Figure 24 – actionSCHEMA.xsd.....	31
25	Figure 25 – Sample of action.....	31
26	Figure 26 – outputSCHEMA.xsd .....	32
27	Figure 27 – Call an action from Site_Planning_Frame .....	33

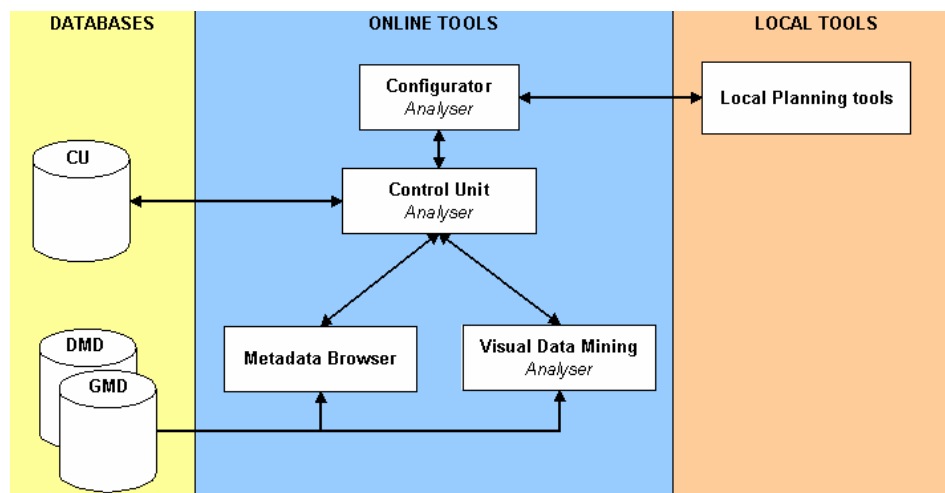
28	Figure 28 – The client and the server sides of the Control Unit.....	40
29	Figure 29 - The histogram visualisation .....	42
30	Figure 30 - A pie chart .....	43
31	Figure 31 - The Table visualisation .....	44
32	Figure 32 - A Parallel Coordinate Diagram .....	45
33	Figure 33 - The VDMcontrol panel .....	47
34	Figure 34 - A screen shot of what the tool looks like .....	49
35	Figure 35 - The screen shot of the tool after a selection task.....	50
36	Figure 36 - The screen shot of Land Price Prediction Tool .....	51
37	Figure 37 – Land Price Prediction Decision Tree.....	52
38	Figure 38 – The Configurator.....	55
39	Figure 39 – The menu of the Configurator .....	56
40	Figure 40 – The popup menu .....	56
41	Figure 41 – The input dialog for the new description. ....	57

## 2 Introduction

Within INVISIP project, people are involved to the site planning process. Municipal authorities, departments, planning offices, data suppliers and citizens use INVISIP website in order to improve search and analysis of necessary data that can be, for example, documents, laws and maps.

The Analyser is a set of tools that enables users to navigate in information spaces, to select data or links to data sources and to determine relationships between them.

The following picture shows the components of the Analyser.



**Figure 1 - Framework of INVISIP**

In Work Package 4 the Analyser is created. It is organized in the Control Unit, the Visual Data Mining and the Configurator

DAPP, CNR-IMA, KTH, UoC, tim GmbH/Thales GmbH and Inregia, are involved to Work Package 4 (leader DAPP).

In the present deliverable the demonstrator of the Analyser is described so far:

- Control Unit (Chapter 3);

- Visual Data Mining (Chapter 4);
- Configurator (Chapter 5).

## 2.1 Control Unit

The main goal of INVISIP is to follow the user during the site planning process. Site planning process consists of the following tasks:

1. Search data;
2. Analyse data and select a set of them;
3. Use data as input of Instruments for the Site planning.

Task 1 and task 2 can be repeated in order to refine the search of data.

Data needed during the site planning are laws, reports, maps, spatial data and non-spatial data.

In INVISIP project, the users can select data using tools as Metadata Browser and Visual Data Mining. These tools show the metadata (or the description of data) using different visualizations that enforce different aspects of the same data. The analysis provided by Metadata Browser and Visual Data Mining can be combined in order to select the best sets of data concerning the site planning process.

This is the way the Control Unit coordinates INVISIP tools: every time the user selects data using an INVISIP tool, the Control Unit stores them internally. When the user opens a tool during the analysis, the Control Unit pass the data selected to the new tool. For example, let us suppose the user is looking for data concerning the "Municipality of Genoa". The application he needs is the Metadata Browser. At the end of his research, he selects three data: now the metadata browser is storing data in the Control Unit. The next step of the planning process of the user is the Visual Data Mining tool: when the Control Unit start the Visual Data Mining tool, it gives the data selected in the Metadata Browser.

During the analysis, the Control Unit shows to the user the list of the selected data.

The Control Unit gives to the user the workflows that is the list of tasks they have to complete during the site planning process. The workflow is a “template” of a planning process defined in INVISIP project.

During the planning process, the user adds data to the workflow so we call it “session”.

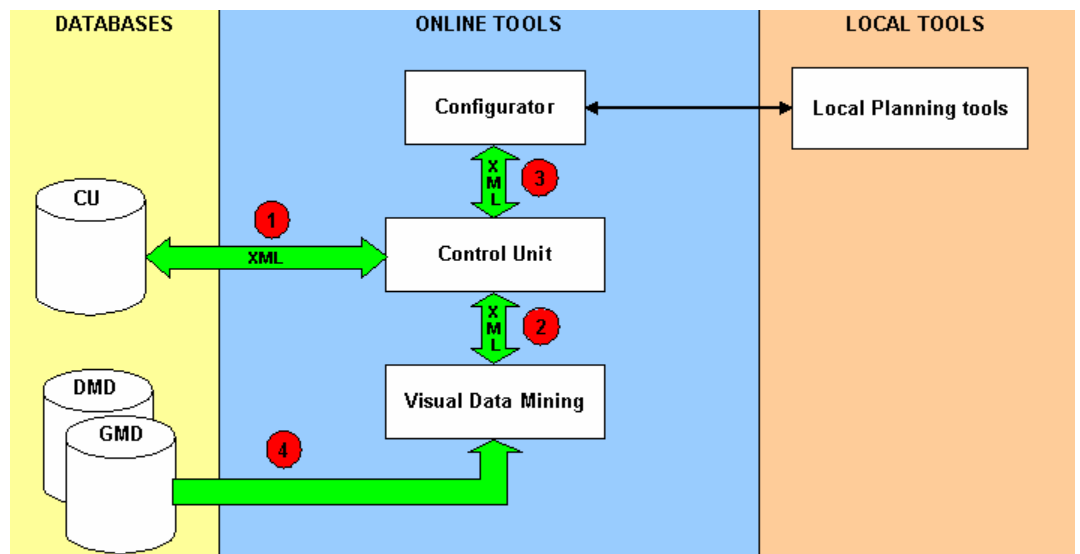
At any moment the user can save, stop and repeat the session and the Control Unit updates the state of it. When the user find data on the web, he download them and use them as input for local instruments that are application for the site planning.

At the end, the Control Unit follows the user during the integration of data where the user is invited to describe the results of the local instruments.

The Figure 2 depicts the data flows during the site planning analysis. Every flow has a specific meaning.

- The flow 1 is composed by the workflow and the session. Data are send and received in XML.
- Flows 2 and 3 are the data flows between the Control Unit and the tools. Data are send and received in XML.
- Flows 4 is the data the Visual Data Mining read in the databases containing metadata about Geodata (GMD) and Document (DMD)





**Figure 2 – Data flows during the site planning analysis**

## 2.2 Visual Data Mining

The Visual data mining module is the part of analyser component that aims to provide tools which facilitate the decision making process for all involved parties.

Two kinds of tools are provided within the VDM module:

- a VDM tool to analyse metadata contained in the repository;
- a DM application to analyse geographical data for Land Price prediction .

In the following, a brief description of the tool goal is provided.

### 2.2.1 A VDM tool to analyse metadata

VDM tool can be used to support users in the data selection task. A basic problem in the data selection task is the search for actual and expressive geo-data and their analysis. Deciding which data are the most suitable to face with a site planning problem is a hard task since geographical data are characterised by many features and often user does not know all the data requirement.

The problem can be defined as finding the optimal subset of all the existing geographical datasets for the particular site. The optimisation criteria for this subset are initially set by the user, e.g. finding the most up-to-date datasets, finding the datasets with the appropriate resolution, etc. The user formulates these suitability criteria according to his specific discipline (traffic analyses, environmental aspects, etc.). In automatic data mining these criteria are usually expressed as logical statements, connected by either conjunctions or disjunctions. In the case of visual data mining however, the user opens visualisations based on these criteria. Using different types of visualisations, the user can display one or more metadata attributes. He recognises patterns in the open visualisations and bases his choice on a subset of items that he is interested in. His input is the desired selection of subsets.

VDM tool leads user to acquire new knowledge about data that are available in the repository. This knowledge consists of relationships, similarities among data, and it helps to come to an arrangement between data, which he is looking for and what is available.

### **2.2.2 Land Price Prediction module (DM for Data Mining)**

The land price prediction (LPP) is supposed to be a tool that enables an investor to check land prices in a region he plans his development. Prior to selecting a region, the investor might want to check where the immobilities prices are the most competitive, and suited to his/her needs. Usually these informations may only be found by looking through advertisements or querying Internet databases, which would not give a general overview. At the same time, these advertisements are usually local, and only give information as for the active moment. On the other hand by utilising artificial intelligence methods it is possible to build models that span a whole region. With the LPP tool the prospective investor is able to select his preferences as to what type of real property he/she is interested in, where it should be

situated, as well as the availability of media, e.g. gas, electricity, etc. As a reply he would get a most probable price per a square meter of such a property.

In the LPP tools several mathematical models built were based on two sources of information: a set of accomplished sale prices in one of regions and, in a separate model, from advertisements coming from an Internet advertisement service. The models are augmented with a 'wizard' that helps a user with selecting proper attribute values for a set of most common real estate transactions.

### **2.3 Configurator**

The Configurator is the component of the Analysis that follows the user during the analysis on his local computer.

The Configurator is the application that connects the INVISIP project to specific tools for planning as worksheets or CAD systems. In fact he Configurator links the “online” analysis to the “off-line analysis”. The first is the analysis the user does using the Visual Data Mining and the Metadata Browser, the second is the analysis of data using the applications installed on local machine.

The Configurator helps the user to “configure” the analysis. Let us suppose the user selects data using the INVISIP tools. When he has the data , he begins the analysis that consists in a list of steps that he has to complete. It gives the possibility of configuring the list of steps and save it. Every analysis can be saved and loaded locally or on the database of INVISIP project.

At the end of the off-line analysis, the user can improve INVISIP database giving the description of the results obtained.

### 3 The Control Unit

#### 3.1 The layout of INVISIP site

The Control Unit is composed by two parts. One of this runs on the browser of the user and one of it is on the server. The Control Unit was developed by DAPP as a component of the Analyser.

The user interface of the analyser, where the Control Unit is displayed, is composed by a Website and by local site planning tools. The home page of INVISIP website is available at <http://bordeaux.igd.fhg.de:8080/cu/HomePage.html>

The website interface is depicted in Figure 3.

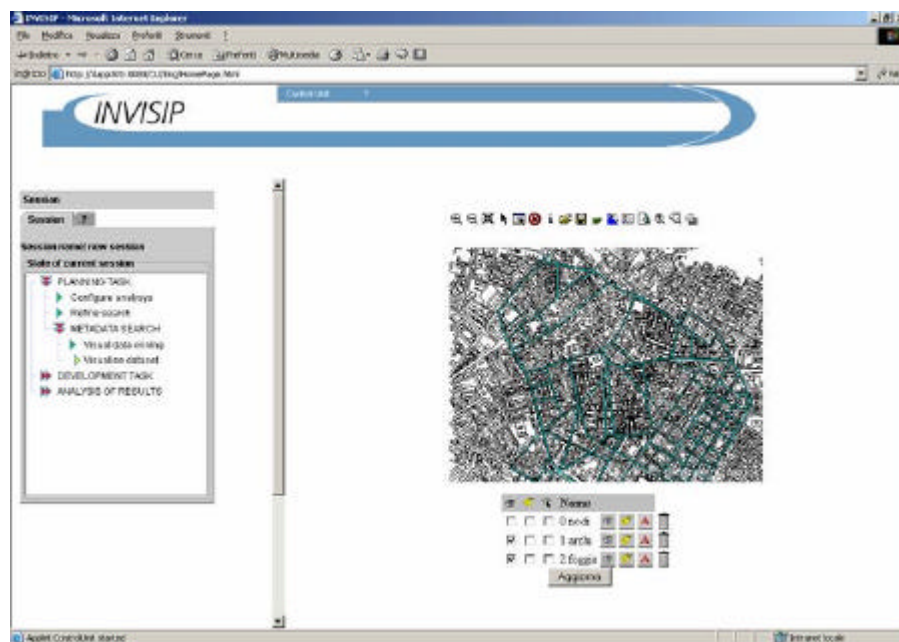
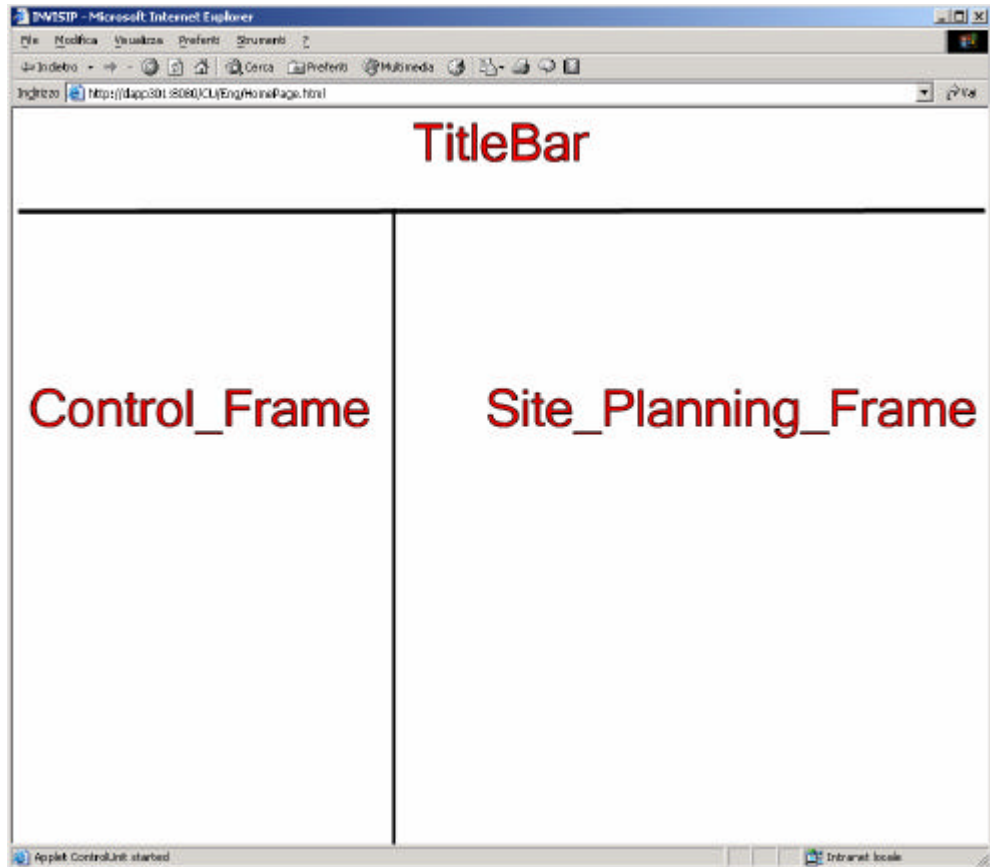


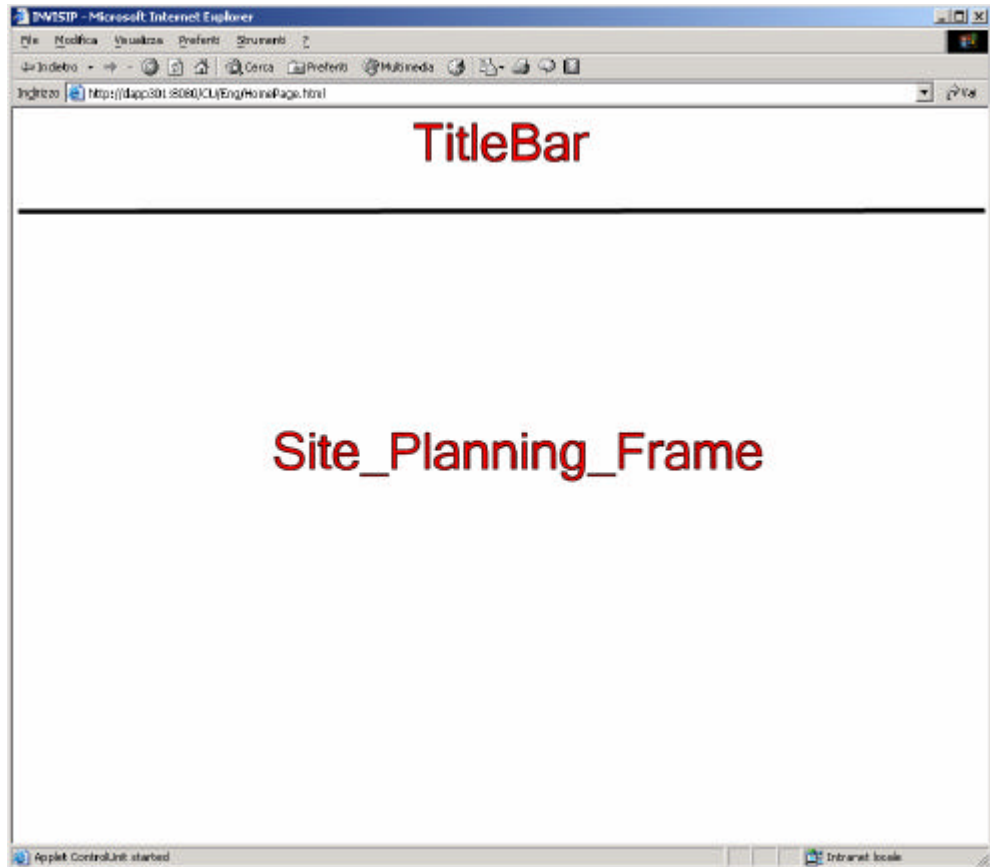
Figure 3 - Analyser interface

INVISIP homepage is divided into 3 rectangular cells (frames) named TitleBar, Control\_Frame and Site\_Planning\_Frame. Figure 4 depicts the three frames.



**Figure 4 – The frames in the HomePage of the INVISIP website**

TitleBar contains the INVISIP logo and a menu always displayed. Main characteristic of the TitleBar is to give to other frames global functions. The most important functions are *Show* and *Hide*: they set the visibility of the Control\_Frame. When the Control\_Frame is hidden, the Site\_Planning\_Frame is large as the window of the browser (Figure 5).



**Figure 5 – INVISIP website when Control\_Frame is hidden**

Site\_Planning\_Frame shows the Metadata Browser tool, the Visual Data Mining tool and all other components involved in INVISIP project except the Control Unit.

Control\_Frame contains the Control Unit applet that connects client to server, co-ordinates client applications and provides session management. Control Unit can receive commands from all the components displayed in the Site\_Planning\_Frame. Let's name these commands "action": they are instructions that add flexibility and functionality to the client. For example we can move a frame to a specific location or we can store the client data analysis to the server. All components displayed in the

Site\_Planning\_Frame can call the actions of the Contro\_Unit even though it is hidden.

Local planning tools have different interfaces. These tools execute specific analysis for the site planning and they need input data (for example maps and data about traffic). Every time the user find data using INVISIP website, he downloads them and use them as input for local planning tools.

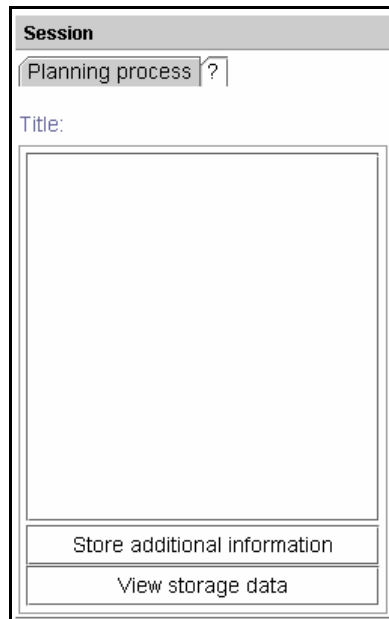
### 3.2 The client side

The application of the Control Unit shows two page: the first is the “login” page (Figure 6) and the second is dedicated to the “Planning process” (Figure 7).



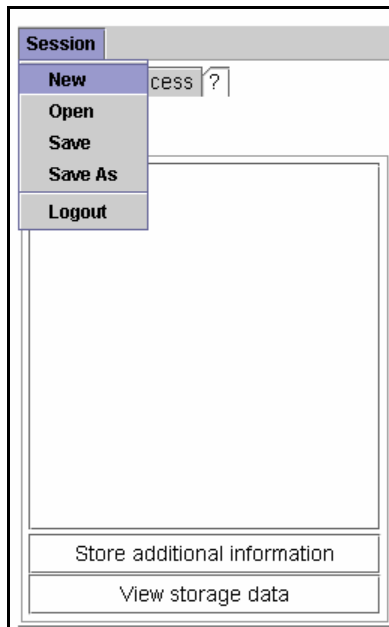
**Figure 6 – The login page**

In the login page the user has to write his login and he can change the language of the interface of the Control Unit. During the INVISIP project the Control Unit can be viewed in Italian, German, English, Polish and Swedish languages. The login page is hidden when the user inserts his login.



**Figure 7 – Planning process page**

The “Planning process” page is shown always and shows the session of the user. When the user is logged on the site, the Control Unit shows the menu “session” (Figure 8).



**Figure 8 – The session menu**



The pull-down menu "session" offer the user different possibilities for handling his session:

- **New** : opens the list of the workflows and the user can start a new session with no data stored.
- **Open** : opens an already processed session.
- **Save**: stores session.
- **Save as** : stores the opened session and gives the user the possibility to define a name for the opened session.
- **Logout** : closes the opened session and logs you out of the system.

The button **Store additional information**" gives the possibility to store any free-text documents to the session you are working in.

The button **"View storage data"** shows to the user the list of data selected and stored in the open session.

### 3.2.1 Workflow and Session

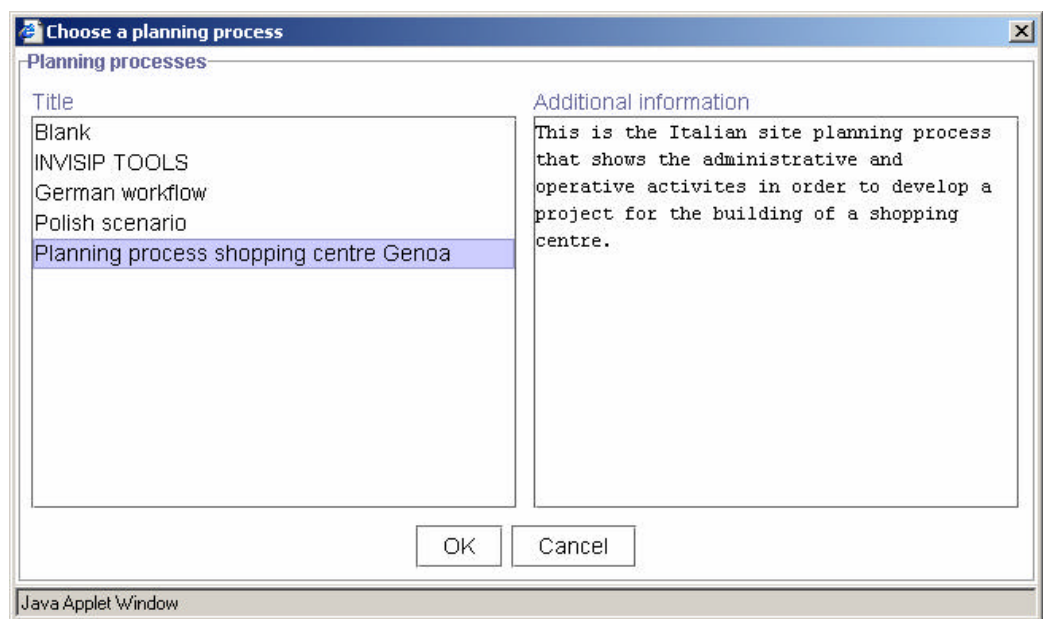
The INVISIP workflow is a template that is an abstract model of the site planning analysis. The necessity of define a workflow comes from the fact that all analyses have to perform a list of steps that the user must complete in order to terminate the site planning analysis.

When the user selects "New" from the "Session" menu, the dialog "Choose planning process" is shown (Figure 9): on the left side of the window, the user reads the title of the planning process and on the right, he reads its description. When the user selects a planning process and press the "OK" button, the workflow is downloaded in the Control Unit.

Five planning processes (or workflows) has been developed during the INVISIP projects:

1. **Blank**: it is a blank planning process;
2. **INVISIP TOOLS**: is the planning process that shows all the tools developed in INVISIP project;

3. **German Workflow.** is the workflows customized for german municipalities
4. **Polish scenario:** is the planning process customized for polish municipalities
5. **Planning process shopping centre Genoa:** is the Italian site planning that shows the administrative and operative activities in order to develop a project for the building of a shopping centre.

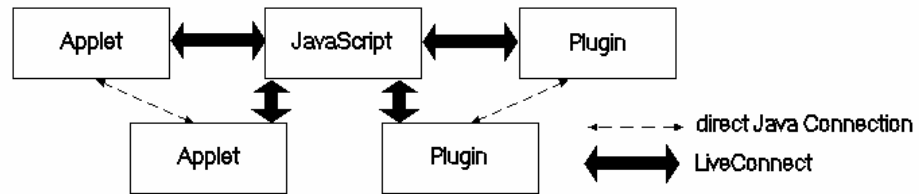


**Figure 9 – List of planning process**

In the site planning analysis, the workflow is a list of operations that the user have to do in order to complete the analysis successfully. It may say that a workflow is composed by a set of atomic operations (steps) grouped in task. Each task can be recursively structured as set of tasks and each step can be necessary or optional.

In the Control Unit applet, a workflow is depicted as a hierarchical structure named tree. This data structure consists of a logically arranged set of nodes. Each tree contains one or more root node, which serves as that tree's top-most node. Any node can have any arbitrary number of child

(descendant) nodes. In this way each descendant node is the root of a sub-tree. A node that has no descendants is called a leaf node. Using the tree to represent the workflow, a root node is a task and a leaf node is a step.



**Figure 23 – Connections among JavaScript, Applet and Plugin**

represents a generic workflow organised in one task (“FEASIBILITY STUDY”) and a list of subtasks. The task “GEODATA” is the only task that is composed by steps and a sub-task: the metadata search.

TASK	STEP	
<b>FEASIBILITY STUDY</b>		
	Search for documents	
	Integrate geodata	
	Local planning	
	Search for geodata	
	View geodata	
	Analyse geodata found	
	<b>TASK</b>	<b>STEP</b>
	<b>Geodata</b>	
		German WEBGIS
	Geo Library	

**Figure 10 – Generic workflow**

Figure 11 shows the workflow in the Control Unit. All tasks and steps are represented with an icon and their name. A task is depicted as a double arrow while a step is figured as a single arrow. The colour of the images gives information about the state of the step or task. A generic step can be only in one state of these states:

- to be done : when the user do not execute this step. This step is depicted by a red arrow (as “Geo Library”);
- on going: when the user is executing this step. During the execution of this step, we have an animation of yellow arrows (as “Analyse geodata found”);
- executed: when the user completes this steps. This state is associated to a green arrow (as “Search for documents”).

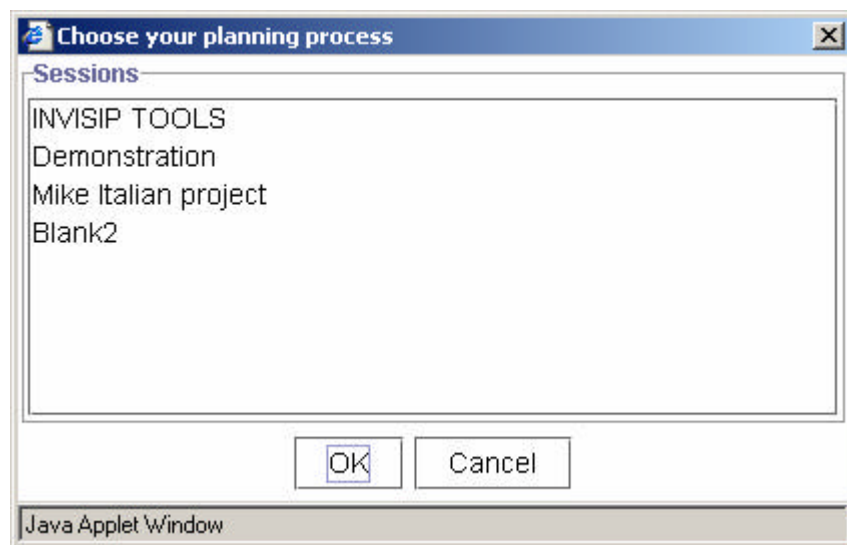
While the icons of the steps can have one of three colours, the double arrow of the tasks can be only red and green. This means that the user has to complete the task or just completed it. Of course, a task is completed (that is it has two green arrows) only when all its steps and subtasks are executed. As depicted in Figure 11, only the “DEVELOPMENT TASK” is completed.



**Figure 11 – The tree representing the workflow**

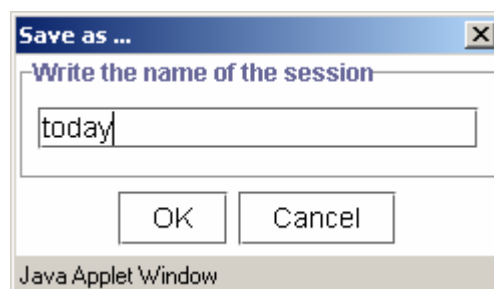
The double arrows of a task can be oriented horizontally or vertically (as “Geodata”): in the first case, the user has “opened” the root node while in the other case he has “closed” the task.

When the user selects “**Open**” in the session menu, the window depicted in Figure 12 is shown. The open dialog lists all the sessions the user stored previously. When the user selects a session in the “open” dialog and click the “OK” button, the session selected is loaded in the Control Unit and he can modify it.



**Figure 12 – The open dialog**

The “Save as” command in the session menu shows the “Save as” dialog (Figure 13) where the user has to insert the title of his planning process.



**Figure 13 – The „Save as“ dialog**

### 3.2.2 Customizing the workflow

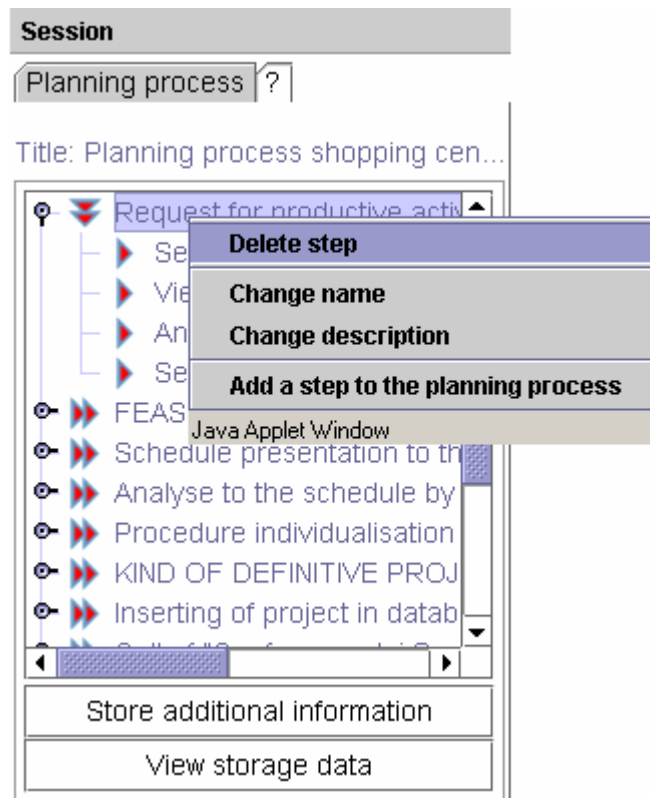
The site planning analysis is a sequence of tasks. During INVISIP project, we define different workflows that the user can download and complete. But the main problem of site planning analysis is that users have different

interests. For example, during the same site planning analysis, one person can study the laws of a country and another the traffic. Due to this fact, during INVISIP project we develop modular workflows.

Modular workflow means that users can customize the sequence of tasks of a site planning analysis in these ways: adding new nodes, deleting nodes and changing the order of the node.

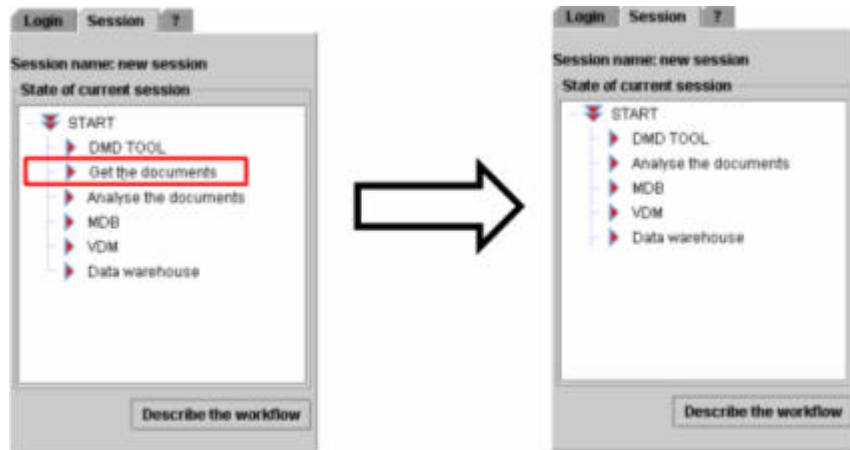
The user can start his site planning analysis using preconfigured workflows or a blank workflow and then he can modify it. During the development of INVISIP project, we decide to customize the workflow in order to increase user's interests.

Figure 14 shows the popup menu with the list of operation the user can do on a task of the workflow: delete the selected node, changing its interface (the name and/or description) or add a new step copying it from the library.



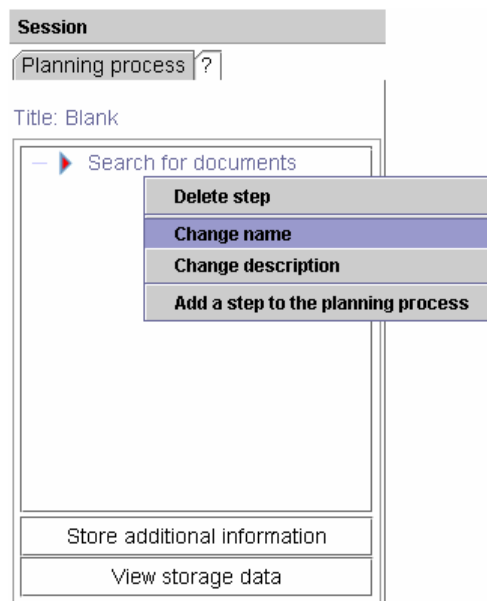
**Figure 14 - Customizing the workflow**

For example Figure 15 shows the workflow before and after the user deletes the task “Get the documents”.



**Figure 15 – Delete a task**

The Control Unit gives the possibility to change the name of the steps. Let us suppose that the user wants to rename the step “Search for documents” in “Search for legal documents”. He has to select “Change name” from the popup menu (Figure 16), then he has to modify the name of the step (Figure 17).

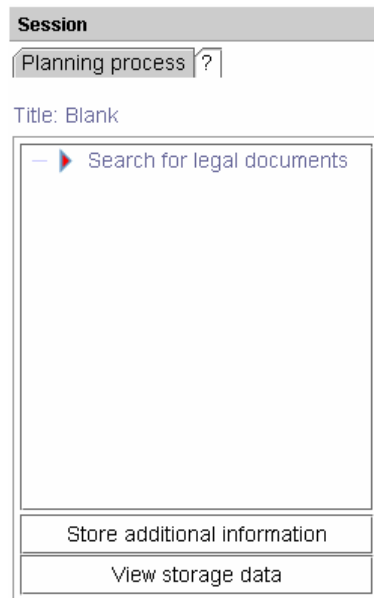


**Figure 16 – The user read „Search for documents“**



**Figure 17 – The user modifies the name of the node**

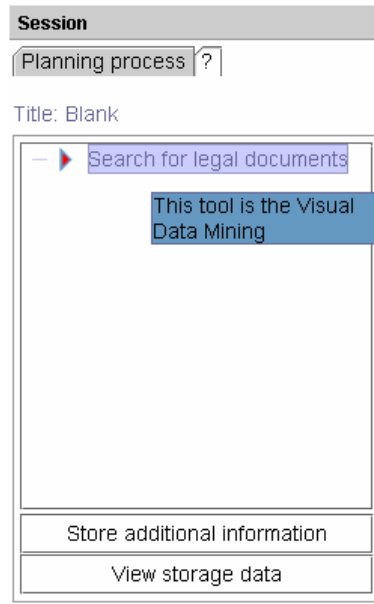
Figure 18 shows the node with the new name.



**Figure 18 – The user changes the name of the node**

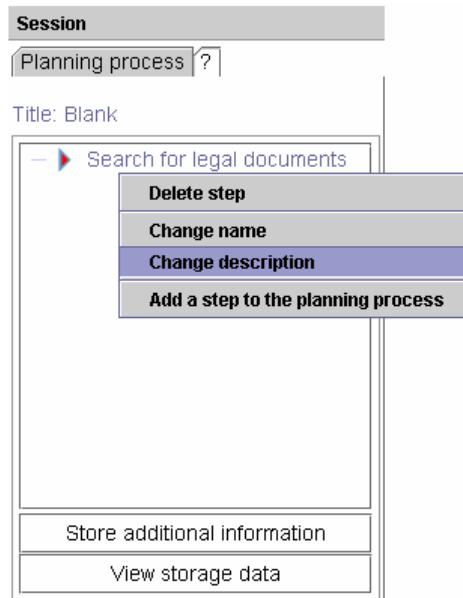
The description of the node is the text shown in the blue box when the user moves the mouse on a node. In the Figure 19 the description of the node “Search for legal documents” is “This tool is the Visual Data Mining”.



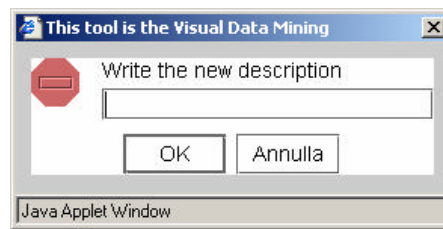


**Figure 19 – The description of a node**

If the user selects “Change description” from the popup menu (Figure 20), the Control Unit opens the windows depicted in Figure 21: here the user has to write the new description.

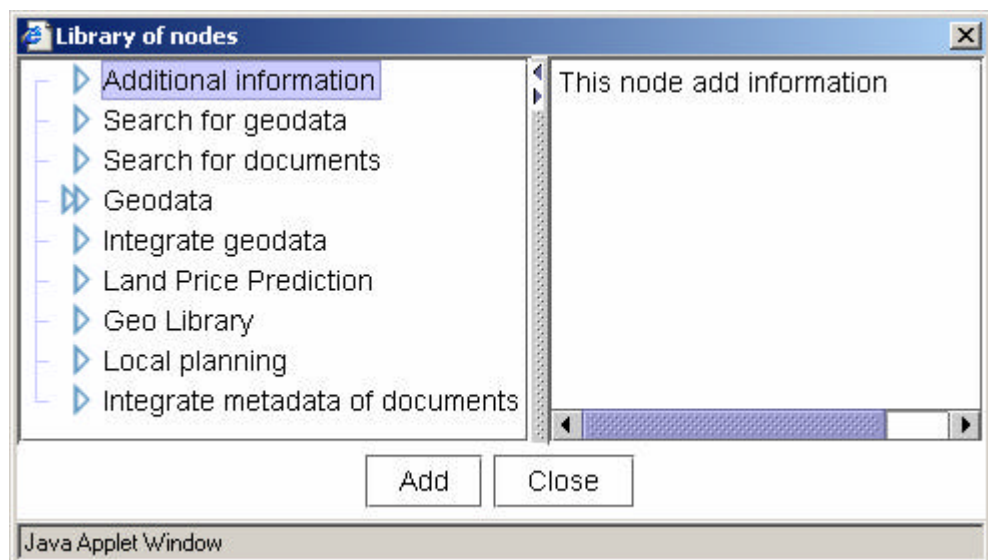


**Figure 20 – The user selects „Change description“**



**Figure 21 – Window for the insertion of description**

The last function in the popup menu is “add a step to the planning process” (Figure 14). Figure 22, shows the library that is a set of preconfigured tasks: on the left side we have the name of the tasks and on the right we have the description of the selected task.



**Figure 22 – The library**

The user does a simple selection in the library dialog and he adds new tasks to the Workflow displayed in the Control Unit.

### 3.2.3 Implementation technologies

The Control Unit is implemented as an applet that is a particular type of Java program embedded in a Web page. When user opens a Web page containing an applet, the applet runs locally (on the client machine that is running the Web browser), not remotely (on the system running the HTTP server). Consequently, security considerations are paramount, and applets

are restricted from performing various operations that are allowed in general Java applications. For instance, you might need to write a program that deletes files, but you certainly don't want to let applets that come in over the Web delete your files. An applet cannot read from the client disk and cannot write on the client. Moreover it cannot open network connections other than to the server from which the applet was loaded and it cannot discover private information about the user.

The applet of the Control Unit is in a Web page that contains JavaScript procedures, as all the pages of the INVISIP site.

JavaScript is a scripting language that is embedded in Web pages and interpreted as the page is loaded. JavaScript can discover a lot of information about HTML document it is in and can manipulate a variety of HTML elements. In the pages about site planning, JavaScript can be used as follows:

- to build HTML dynamically as the Web page is loaded;
- to monitor various user events and to take action when these event occur;
- to call Java methods and to control applets.

Using Java and JavaScript, the INVISIP client has these characteristics:

- JavaScript can access all the public methods of the Control Unit;
- Control Unit can call JavaScript procedures;
- Control Unit can interact with all the other components (that are applets) displayed on the client page.

Details about accessing Java from JavaScript and accessing JavaScript from Java are explained in 3.2.4 and in 3.2.5.

The Control Unit applet is implemented as front end to server-side programs: this means that the applet can open a direct connection to the server of INVISIP and they send information each other. This technique is

known as HTTP tunnelling since the server sends data to the applet (or it receives data from the applet) using a custom communication protocol embedded within the HTTP packets. All the data exchanged between client and server applications and between two client applications are XML documents that can be easily manipulated by a Java program.

XML stands for “Extensible Mark-up Language” and is a “meta” mark-up language used to describe the structure of data. This techniques has numerous advantages including being easy to read, easy to parse and extensible. In order to exchange data between the Control Unit and another application (both on server or client side) there are only two basic concepts that must be understand about XML documents. The first is that any XML document must be *well-formed* to be of any use and to be parsed correctly by all the applications involved in this process. The second basic concept is that XML documents must be *valid* that is they must be conformed to their schema. A schema is a XML Schema documents that defines the grammar and the tag set for a specific XML formatting: it establishes a set of constraints for XML document.

XML is currently a completed W3C Recommendation, meaning it is final and will not change until another version is released. For the complete XML 1.0 Specification, see <http://www.w3c.org/TR/REC-xml>. To learn more about XML Schema, visit <http://www.w3.org/TR/xml-schema-1> and <http://www.w3.org/TR/xmlschema-2>. A helpful primer on XML Schema is located at <http://www.w3.org/TR/xmlschema-0>.

### 3.2.4 Accessing Java from Javascript

JavaScript can access Java variables and methods simply by using the fully qualified name. For instance, using:

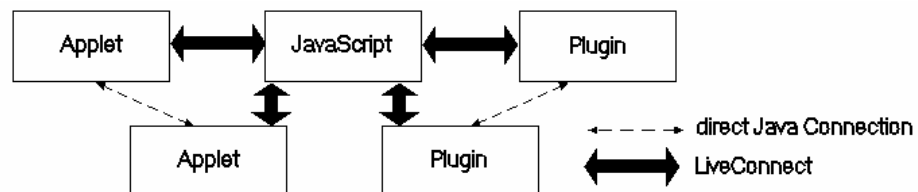
```
Java.lang.System.out.println(„Hello!“);
```

Will send the string „Hello“ to the Java console. Use of Java from JavaScript has two major limitations. First you cannot perform any operation that would not be permitted in an applet, so you cannot use Java to open local files, call local programs, discover the user's login name, or execute any other such restricted operator. Second, and most significantly, JavaScript provides no mechanism for writing Java methods or creating subclasses. As a result, you will want to create an applet for all but the simplest uses of `java.lang.System.out.println`.

JavaScript can access either through the `document.applets` methods array or, if the applet can is named, through `document.appletName`. Any public method of the applet can be called by JavaScript. For example, if an applet named „Sound“ contains the public method „Play“, we can call this method using `document.Sound.Play()`.

### 3.2.5 Accessing JavaScript from Java

Applets interact with JavaScript through the mechanism of LiveConnect that was introduced in Navigator 3.0. Now this technique is extended to the Internet Explorer browser due to the fact that everytime we download an applet, the browser opens a Java plug-in and „LiveConnect“ use this plug-in to interact with JavaScript. Connections among Java, JavaScript and the plugin are depicted in Figure 23.



**Figure 23 – Connections among JavaScript, Applet and Plugin**

To use LiveConnect facilities, you must have `netscape.javascript.JSObject` class that lets you use Java syntax to

access all JavaScript objects. In this way you can read and set all variable properties and call any legal method. Furthermore, you can use the `eval` method to invoke arbitrary JavaScript code when doing so is easier than using Java syntax.

The `JSObject` class is distributed in `jaws.jar` currently available in `j2sdk1.4`. If you want to implement code using this class you have to add `jaws.jar` to your CLASSPATH. The Java compiler knows how to look inside JAR file already.

Control Unit uses and contains `JSObject` so that when the user opens INVISIP site, the Control Unit works correctly in Netscape and in Internet Explorer.

If an applet wants to use `JSObject`, it has to import the `JSObject` calls. Then it has to obtain a JavaScript reference to the window containing the applet. At the end, it can read or set the JavaScript properties of interest and call JavaScript methods. Next paragraphs report sample codes about the `JSObject`.

### **3.2.6 Connecting Control Unit to a component displays in the Site\_Planning\_Frame**

All the components displayed in the `Site_Planning_Frame` can interact with the Control Unit following these three steps:

- They format a text that defines the name of the action and its parameters;
- They send the text to the Control Unit;
- If the action returns a result, they wait for a result from the Control Unit and then parse it.

The text that defines an action of the Control Unit uses the set of xml-tags defined in the `actionSCHEMA.xsd` (Figure 24).

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:complexType name="application">
    <xsd:sequence>
      <xsd:element name="action" type="actionType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="actionType">
    <xsd:sequence>
      <xsd:element name="action-name" type="xsd:string"/>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="param-list" type="param-listType" nullable="true"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="param-listType">
    <xsd:sequence>
      <xsd:element name="param-item" type="param-itemType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="param-itemType">
    <xsd:sequence>
      <xsd:element name="param-name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="param-value" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

**Figure 24 – actionSCHEMA.xsd**

A sample of text that a component can use is the following (Figure 25)

```

<action>
  <action-name>GoTo</action-name>
  <param-list>
    <param-item>
      <param-name>urlString</param-name>
      <param-value>http://www.invisip.de</param-value>
    </param-item>
  </param-list>
</action>

```

**Figure 25 – Sample of action**

The result of an action is formatted using the outputSCHEMA.xsd shown in Figure 26:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:complexType name="output">
    <xsd:sequence>
      <xsd:element name="action" type="actionType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="actionType">
    <xsd:sequence>
      <xsd:element name="action-name" type="xsd:string"/>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="param-list" type="param-listType" nullable="true"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="param-listType">
    <xsd:sequence>
      <xsd:element name="param-item" type="param-itemType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="param-itemType">
    <xsd:sequence>
      <xsd:element name="param-name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="param-value" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

**Figure 26 – outputSCHEMA.xsd**

### 3.2.6.1 HOW-TO Execute a Control Unit actions from JavaScript

To run an action of the Control Unit, the JavaScript code must be constructed as follows:

1. format the XML-code of the action setting the action-name and all the its parameters
2. call the Control Unit in the Control\_Frame and pass the XML-code to the function `EvaluateXMLAction()`.



For example, if a function in the Javascript code of the Site\_Planning\_Frame needs to call the GoTo action of the Figure 25, the JavaScript code must be:

```
var XMLString = "<action><action-name>GoTo</action-name><param-list><param-
item><param-name>urlString</param-name><param-value>http://www.invisip.de</param-
value></param-item></param-list></action>";
var result = top.Control_Frame.EvaluateXMLAction(XMLString);
```

**Figure 27 – Call an action from Site\_Planning\_Frame**

### 3.2.6.2 HOW-TO Access the Control Unit methods from applet

To execute an action of the Control Unit using java statements, the follow steps must be done:

- configure the xml string containing the details of the action and its parameters
- get a reference to the Control Unit applet
- evaluate the JavaScript statement

```
import netscape.javascript.JSObject;
. . .
//format the xml string
String XMLString = "<action><action-name>GoTo</action-name><param-list><param-
item><param-name>urlString</param-name><param-value>http://www.invisip.de</param-
value></param-item></param-list></action>";
//get a reference to the applet
JSObject window = JSObject.getWindow(this);
//evaluate JavaScript code
String result = window.eval("top.Control_Frame.EvaluateXMLAction(\"" + XMLString + "\")");
```

## 3.2.7 Control Unit Commands

### 3.2.7.1 SetVisible

The SetVisible command shows or hides the Control Unit.

**Input of the Control Unit:**

```
<action>
```

```

<action-name>setVisible</action-name>
<param-item>
  <param-name>hidden</param-name>
  <param-value>true or false</param-value>
</param-item>
</action>

```

**Details:** the parameter `hidden` must be `true` or `false`

### 3.2.7.2 GoTo

The `GoTo` action moves the `Site_Planning_Frame` to a new URL.

**Input of the Control Unit:**

```

<action>
  <action-name>GoTo</action-name>
  <param-list>
    <param-item>
      <param-name>urlString</param-name>
      <param-value>...</param-value>
    </param-item>
    <param-item>
      <param-name>frameName</param-name>
      <param-value>...</param-value>
    </param-item>
  </param-list>
</action>

```

**Details:**

- `urlString` is the new address
- `frameName`: it is optional. It can be `"Site_Planning_Frame"` or `"TitleBar"`. The default value is `"Site_Planning_Frame"`.

### 3.2.7.3 setNodeState

The `setNodeState` changes the state of a step.

**Input of the Control Unit:**

```

<action>
  <action-name>setNodeState</action-name>
  <param-list>

```

```

    <param-item>
      <param-name>State</param-name>
      <param-value>___</param-value>
    </param-item>
  </param-list>
</action>

```

**Details:**

- The Control Unit changes the state of the node of the tools shown in the Site\_Planning\_Frame
- State: is a number defined as follows:
- 0 when the action of the node is executed;
- 1 when the action of the node is not executed;
- 2 when the action of the node is in progress;
- 3 when there is an error

**3.2.7.4 setDataNode**

The command setDataNode saves data in a node of the tree representing the session of the user.

**Input of the Control Unit:**

```

<action>
  <action-name>setDataNode</action-name>
  <param-list>
    <param-item>
      <param-name>data_id</param-name>
      <param-value>value_1</param-value>
      <param-value>value_2</param-value>
    </param-item>
  </param-list>
</action>

```

**Details:**

- The Control Unit sets the data in the node of the tools shown in the Site\_Planning\_Frame

- `data_id` is the key associated to the (list of ) data we want to store in the node

**Result:** it returns true or false, meaning the action goes well or wrong.

```
<output>
  <action-name>setDataNode</action-name>
  <param-item>
    <param-name>Result</param-name>
    <param-value> true or false</param-value>
  </param-item>
</output>
```

### 3.2.7.5 getDataNode

The command `getDataNode` returns the data stored in a node of the session in the Control Unit.

#### Input of the Control Unit

```
<action>
  <action-name>getDataNode</action-name>
  <param-list>
    <param-item>
      <param-name> ParamName </param-name>
      <param-value> data_id </param-value>
    </param-item>
    <param-item>
      <param-name> ParamName </param-name>
      <param-value> data_id_2 </param-value>
    </param-item>
  </param-list>
</action>
```

#### Details:

- The Control Unit gets the data stored in the node of the tools shown in the `Site_Planning_Frame`
- `ParamName` is the key used to store data in the node. If this parameter is not in the xml string, it will be returned all data stored in the node.

**Result:** the result is formatted as follows:

```
<output>
```

```

<action-name>getDataNode</action-name>
<param-item>
  <param-name>...data_id ...</param-name>
  <param-value> </param-value>
</param-item>
</output>

```

### 3.2.7.6 resetDataNode

The command resetDataNode deletes all the data stored in the last node executed by the Control Unit.

#### Input of the Control Unit

```

<action>
  <action-name>resetDataNode</action-name>
  <param-list>
    <param-item>
      <param-name>resetChildren</param-name>
      <param-value> true or false</param-value>
    </param-item>
  </param-list>
</action>

```

#### Details:

- The Control Unit resets the data stored in the node of the tools shown in the Site\_Planning\_Frame
- resetChildren say if we want delete data in the node children. It can be true or false. Default value is false

**Result:** it returns true or false, meaning the action goes well or wrong.

```

<output>
  <action-name>resetDataNode</action-name>
  <param-item>
    <param-name>Result</param-name>
    <param-value> true or false</param-value>
  </param-item>
</output>

```

### 3.2.7.7 removeParamDataNode

The command removeParamDataNode deletes only the data stored in the last node of the session using a specific name.

#### Input of the Control Unit

```
<action>
  <action-name>removeParamDataNode</action-name>
  <param-list>
    <param-item>
      <param-name>ParamName</param-name>
      <param-value>... data_id ...</param-value>
    </param-item>
  </param-list>
</action>
```

#### Details:

- NodeName is the unique name of the node defined in the current workflow
- ParamName is the key used to stored data in the node
- 

**Result:** it returns true or false, meaning the action goes well or wrong.

```
<output>
  <action-name>removeParamDataNode</action-name>
  <param-item>
    <param-name>Result</param-name>
    <param-value> true or false</param-value>
  </param-item>
</output>
```

### 3.2.7.8 getLastExecutedNode

The command getLastExecutedNode returns the name of the last executed node.

#### Input of the Control Unit

```
<action>
  <action-name>getLastExecutedNode</action-name>
</action>
```

#### Details:

The Control Unit gives the unique identifier of the last node executed.

**Result:**

It returns the unique identifier of the last node executed.

```
<output>
  <action-name> getLastExecutedNode </action-name>
  <param-item>
    <param-name> NodeName </param-name>
    <param-value> unique_id </param-value>
  </param-item>
</output>
```

### 3.2.7.9 getCurrentNode

The command getCurrentNode returns the unique identifier of the node that the user is executing.

**Input of the Control Unit**

```
<action>
  <action-name>getCurrentNode</action-name>
</action>
```

**Details:**

- The Control Unit gives the unique identifier of the node that is executing and is associated to the tools displayed in the Site\_Planning\_Frame.

**Result:** the result of this command is formatted as follows:

```
<output>
  <action-name>getCurrentNode</action-name>
  <param-item>
    <param-name> NodeName </param-name>
    <param-value> unique_id </param-value>
  </param-item>
</output>
```

## 3.3 The server side

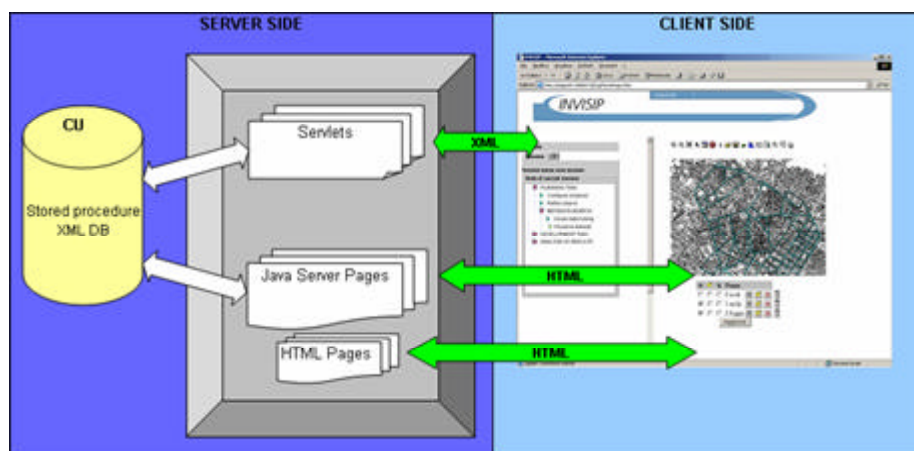
The client side of the Control Unit is always connected to the server. The client side is visible to the user and the second is running on the server and it is remote. The two parts are complementary: the client side stores locally

the results of the site planning and the server side store these results in a database.

The main goal of the server side of the Control Unit is to manage the data of the user and store the log of a session. It stores the selection he made during a site planning analysis and saves the session of the user on a database.

According with all the partners of INVISIP project, DAPP has developed the server side of the Control Unit using the following technologies:

- Apache Tomcat server: is the server that supports INVISIP site;
- HTML Pages: static contents of INVISIP site are in hypertext pages;
- Java Servlets : they are connected to the client side of the Control Unit and to the databases;
- Java Server Pages: they format data results that are sent to the user;
- Oracle platform and specific tools: the database is on Oracle platform. Data about sessions and workflows are stored using Oracle XML DB that provides a high-performance, native XML storage and retrieval technology. Oracle XML DB fully absorbs the W3C XML data model into the Oracle Database, and provides new standard access methods for navigating and querying XML.



**Figure 28 – The client and the server sides of the Control Unit**



Figure 28 depicts the client and the server sides of the Control Unit.

The dataflow between the servlets and the Control Unit are in XML:

- The data from the client to the server contains the login of the user and the unique identifier of the session;
- The data from the server to the Control Unit contains commands to be executed and data

The Java Server Pages and the HTML Pages send data that are displayed in the Site\_Planning\_Frame. Both the Java Server Pages and the servlets read data in the Oracle Database.

When the user looks for data in INVISIP website, the tools (for example the Metadata Browser of the Visual Data Mining) save the main data in the Control Unit. At the end, when he stores the session document, the Control Unit sends this data on the server in XML format.

## 4 Visual Data Mining

In this chapter a more detailed description of tools architecture and the functionalities is provided. All developed visualisations are presented and a simple application example is illustrated in order to make clear how user can take advantage of using the tools.

### 4.1 VDM-module

The VDM tool is characterised by visualisation techniques and interaction functionalities [Albertoni et al. 2003].

The *visualisation techniques* include two different types of visualisations: visualisations of one attribute (a pie chart and a histogram) and visualisations of multiple attributes (a table and a parallel diagram). Other visualisations can be included in the VDM tool in the future.

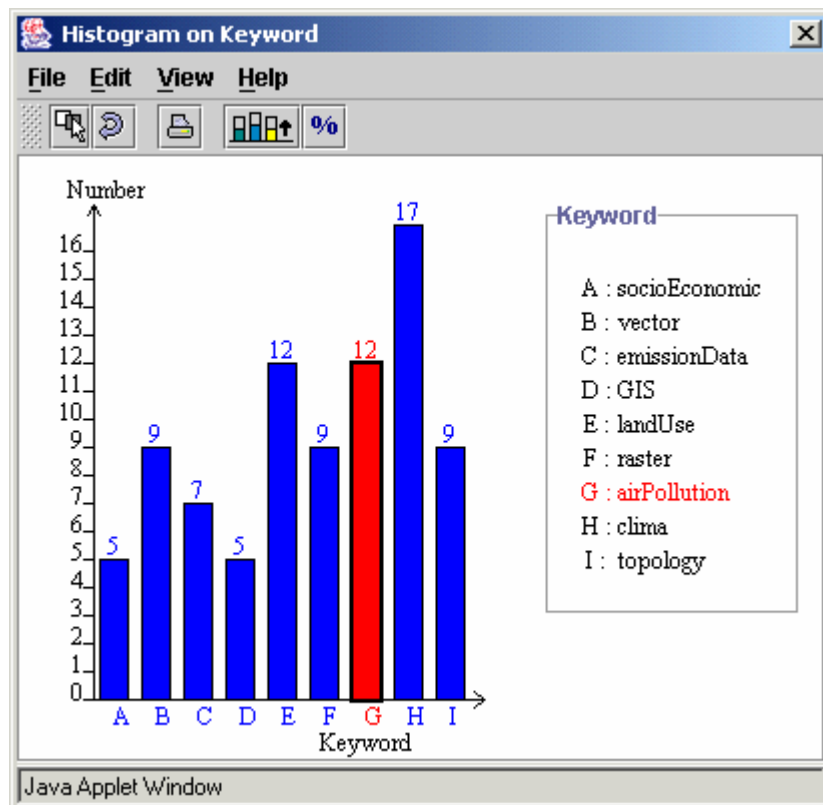
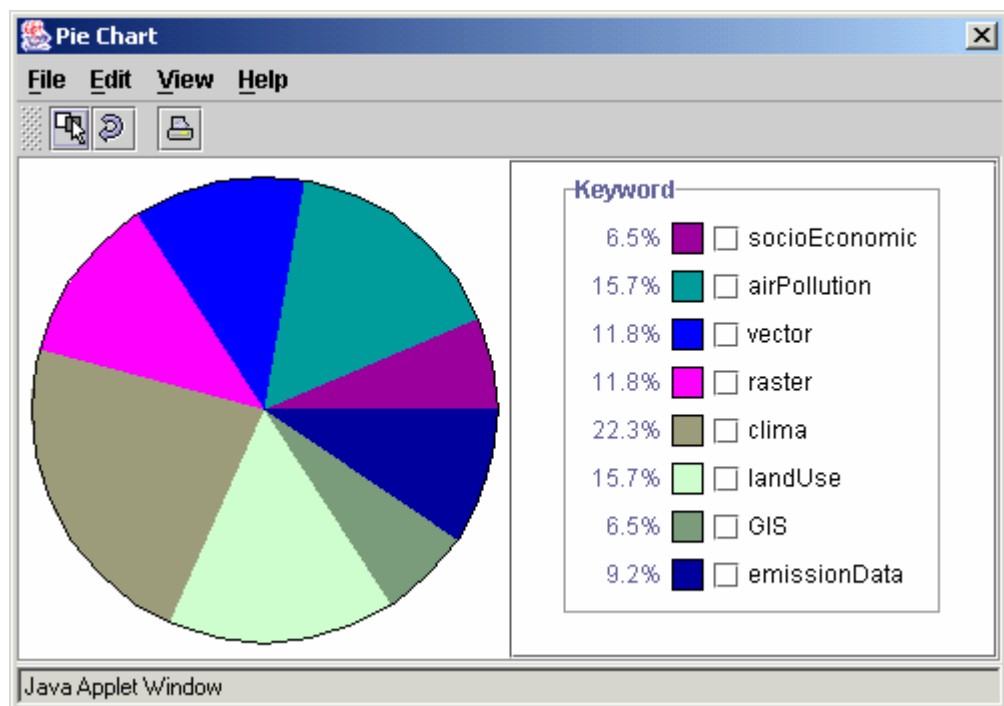


Figure 29 - The histogram visualisation

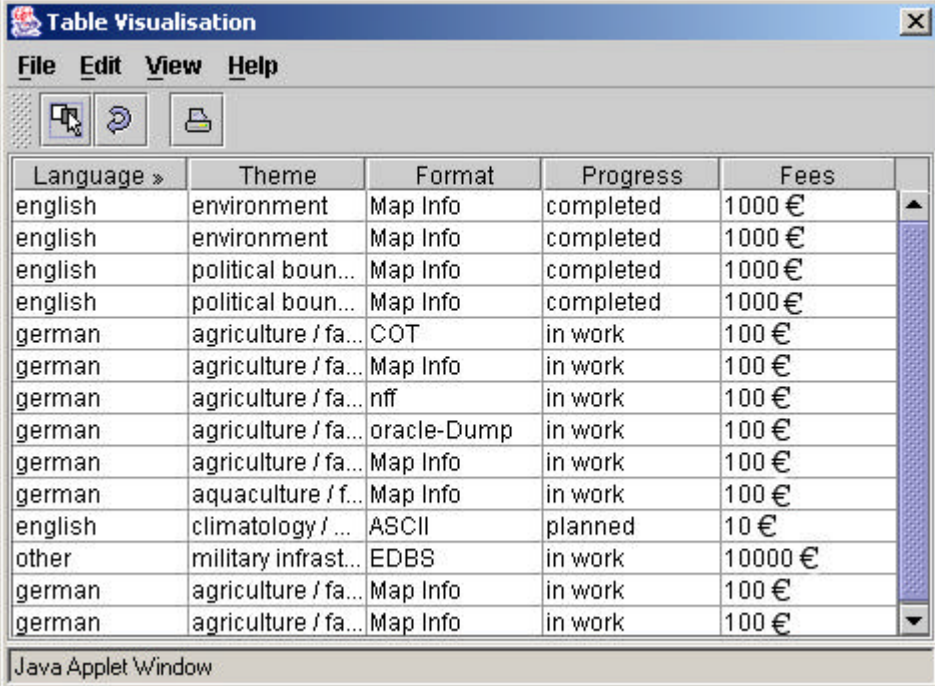
A **histrogram** (Figure 29) shows the number of objects for each value of the chosen attribute. It is useful to recognise the distribution of data objects and can help to identify potentially suspicious objects which can be removed from further analysis by appropriate selection.



**Figure 30 - A pie chart**

A **pie chart** (Figure 30) shows the proportional size of values of one chosen attribute. It is useful when the user wants to recognise a significant element within the attribute.

A **table visualisation** (Figure 31) allows the user to choose one or several attributes and visualise them in a table of values. The columns represent the metadata attributes and the rows the data objects. It is not a graphical visualisation but is nevertheless useful to display a large number of attributes when the data reduction has already been performed by previously using some other visualisation.



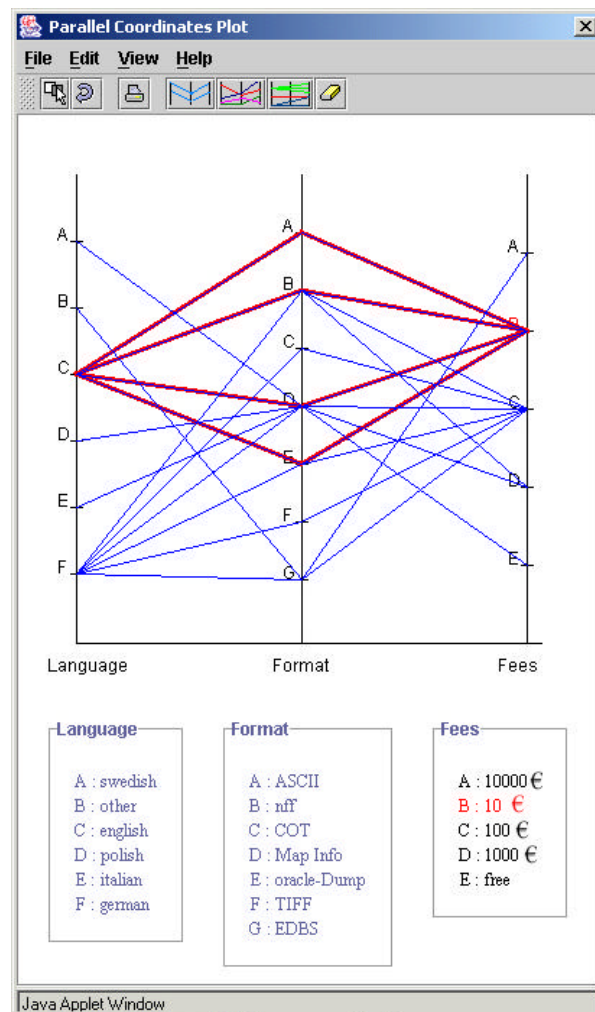
Language »	Theme	Format	Progress	Fees
english	environment	Map Info	completed	1000 €
english	environment	Map Info	completed	1000 €
english	political boun...	Map Info	completed	1000 €
english	political boun...	Map Info	completed	1000 €
german	agriculture / fa...	COT	in work	100 €
german	agriculture / fa...	Map Info	in work	100 €
german	agriculture / fa...	nff	in work	100 €
german	agriculture / fa...	oracle-Dump	in work	100 €
german	agriculture / fa...	Map Info	in work	100 €
german	aquaculture / f...	Map Info	in work	100 €
english	climatology / ...	ASCII	planned	10 €
other	military infrast...	EDBS	in work	10000 €
german	agriculture / fa...	Map Info	in work	100 €
german	agriculture / fa...	Map Info	in work	100 €

Java Applet Window

**Figure 31 - The Table visualisation**

A **parallel diagram** or a parallel coordinates plot (Figure 32) maps the attributes of the dataset onto vertical axes. Each data object in the dataset is represented as a continuous piecewise linear line connecting the axes. The line intersects the vertical axes at the points that correspond to its attribute values. Since the line representing an object connects different attributes, it is necessary to select at least two attributes for a non-trivial plot.

The VDM tool has two different kinds of interaction functionality: the interaction between a single visualisation and the user and the interaction among different visualisations.



**Figure 32 - A Parallel Coordinate Diagram**

The interaction between a visualisation and the user enables the user to explore the content of the visualisation and to graphically extract the selected subset of metadata. The possibility of a graphical selection of metadata exists in all the visualisation techniques, but varies according to the type of each technique. It links each graphic entity to the value of the attribute which it represents. Graphical entities in question can be angular segments of a pie chart, bars of a histogram, rows of a table or lines in a parallel diagram. The values of the attributes are shown in the legend next to the visualisation. The user can select a desired subset of objects by

clicking on the graphical entities that represent their attribute values or by choosing one or several values in the legend. A special type of graphical selection is implemented in the parallel diagram. Unlike in the other visualisations the polygonal lines represent the correlation among different attributes rather than one attribute value only. Figure 32 shows an example: the selected polygonal lines in red show the correlation between the specified cost of the datasets (10€) with their format and their language. The datasets with this cost can be obtained in four different data formats, but they are all in English.

The second kind of interaction helps to discover correlation among graphic entities represented in different visualisations. All the visualisations are interconnected according to the concept of *brushing and linking*. Brushing is an interactive selection process, while linking connects the selected data from the current visualisation to other open visualisations. If the user has several different visualisations open and decides to perform a selection of objects in one of them, the graphical entities that represent this selection and correspond to the same subset of selected data objects in each visualisation are highlighted, providing a better visual impression. When the selection is performed, all other graphical entities disappear from each open visualisation. Brushing and linking design is mainly based on the pattern “observer” and more technical details about how it has been implemented are provided in appendices.

#### 4.1.1 The Architecture of the VDM Tool

The VDM tool is designed as a Java applet in order to easily handle web-based explorations, however it is also possible to run it as a stand-alone application whenever user needs to analyse a local metadata repository. VDM tool consists of three main components: the control panel which integrates all components, the data manager connecting the VDM tool to different resources of metadata and the visualisation wrapper which provides a common template for the different visualisations.

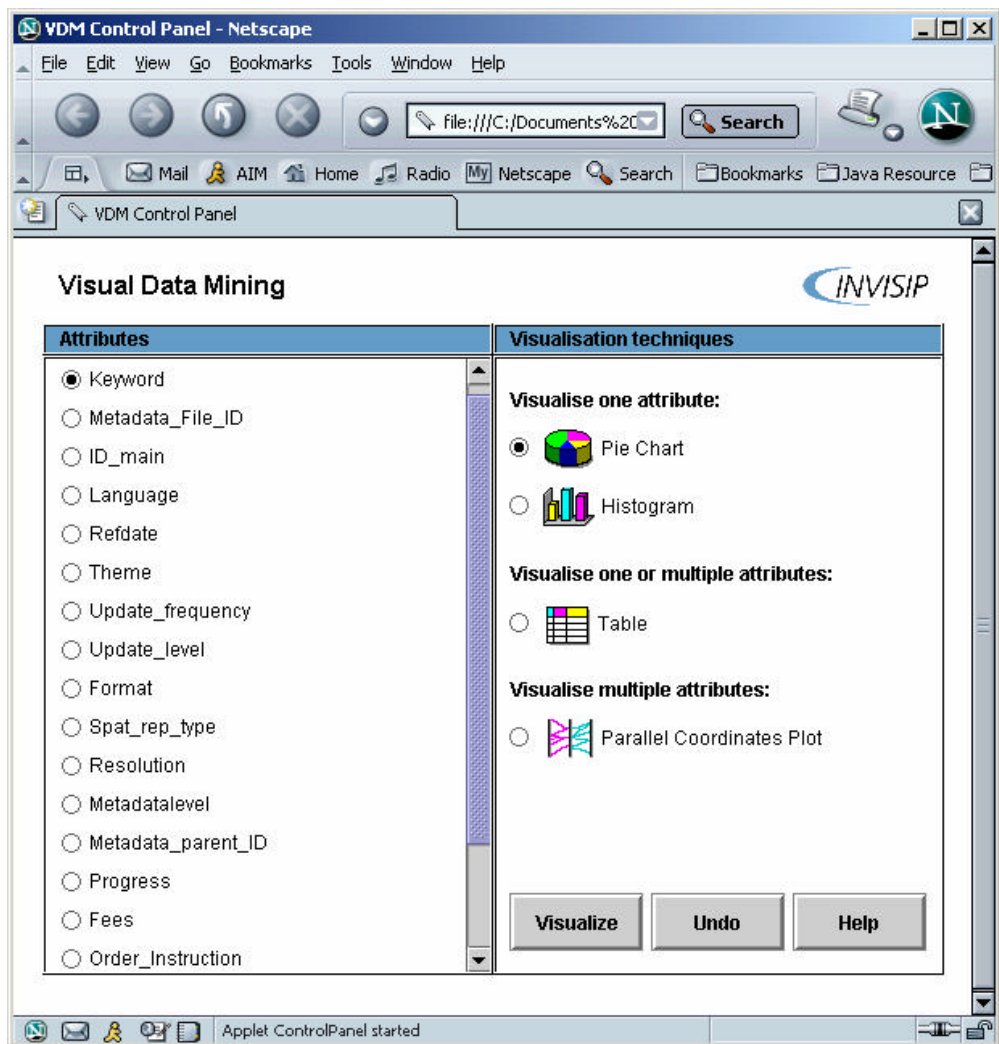


Figure 33 - The VDM control panel

The *control panel* is the main component of the VDM tool, providing a Graphical User Interface (GUI) as shown In Figure 33. The left side of the control panel shows the list of metadata attributes, while the right side shows available visualisations. The control panel activates the visualisations of selected attributes and manages the general layout of the different visualisations.

The *data manager* handles input and output of data. It is based on a table that contains all metadata that can be visualized. Data are dynamically

download each time that the control panel is activated. The download can be performed both by local data connection and remote data connection. The first one allows to connect to a local database whenever the VDM tool is running as a stand-alone application. Since it is based on ODBC, the VDM tool is open to integrate other database managers such as Oracle, SQL server and so on. The remote data connection has to be used whenever the VDM tool is running as applet managed by the Control Unit. The VDM tool can be applied both to Geographical and Document Metadata therefore the remote connection relies on a servlet that needs two parameters: the query and the database where the query has to be performed. If user has already reduced the initial set of available Metadata during a previous work session (by using the Metadata Browser or The 3D geoLibrary), only the subset of Metadata that has been chosen is downloaded.

All visualisation techniques are based on the *visualisation wrapper* Java class. It is an abstract class that all visualisations extend and provides the interface between the visualisations and the control panel, as well as functionalities to draw and to update the graphs contained in the wrapper. It is also responsible for the look & feel (colours, character fonts, etc.) of all visualisations and for the common characteristics such as toolbar and menu.

#### **4.1.2 An application example**

VDM tool can be applied both to analyze Geographical Metadata expressed on ISO 19115 standards and Document Metadata expressed on a format based on Dublin Core.

This section describes an application example of the VDM tool to analyze Geographical Metadata during the data acquisition phase, an example of use could be thought in the Document Metadata case as well.



Let us suppose that the goal of the user is to look for datasets that are cheap, complete and continually updated. Using the VDM tool, a similar scenario can be shown in Figure 34. Different visualisations are opened on several metadata variables: a parallel diagram on update frequency, the progress and the fees, and a histogram on reference date.

From these visualisations the user extracts useful information about the available geographical datasets. The parallel diagram suggests that all datasets are complete, but the datasets available for free are not updated frequently.

The histogram shows that some of the datasets are updated in February and others in August. Assuming that the user would like to purchase the most up-to-date datasets, he selects those produced in August. All open visualisations are updated according to this selection as shown in Figure 34. From the resulting parallel diagram Figure 34 it becomes obvious that the most current datasets are updated continually and cost 100€.

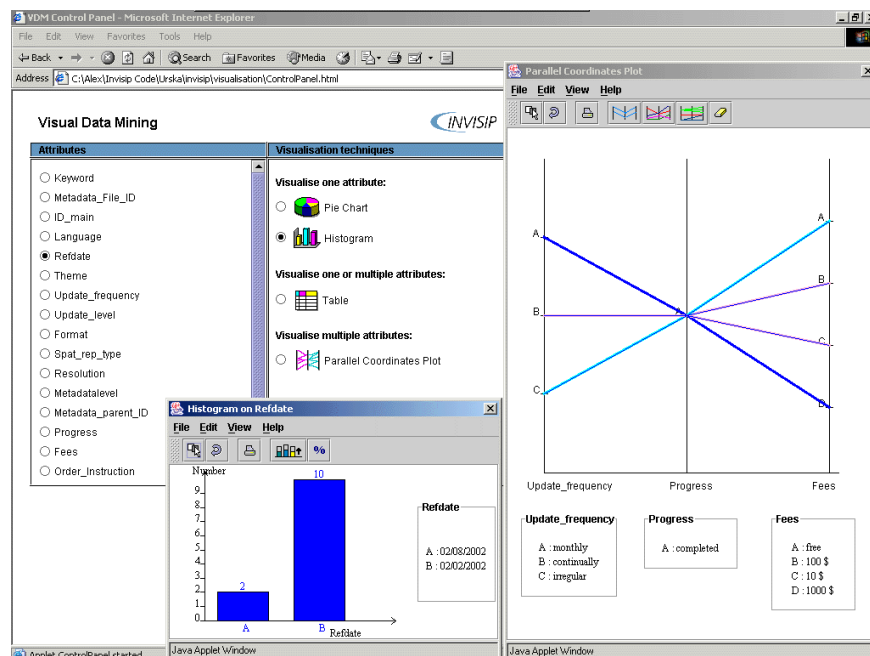
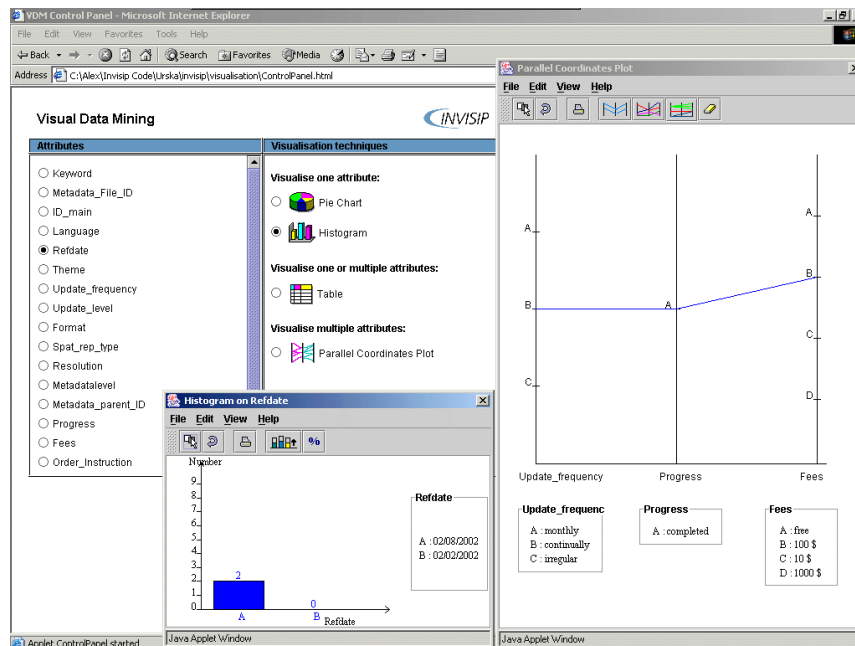


Figure 34 - A screen shot of what the tool looks like



**Figure 35 - The screen shot of the tool after a selection task**

Based on this result, the user can now decide if these datasets are suitable for him, e.g. if their cost is still low enough for him to afford. If the resulting datasets do not fulfill his criteria, he can start the analysis from the beginning with different initial conditions.

## 4.2 Land Price Prediction

The Land Price Prediction module is design to support a user with predicting prices of land plots (Figure 36). This task is achieved with the help of Artificial Intelligence (AI) methods. It is well known that processes, one of which might be the buying and selling of land plots, follow a number of patterns. For example, if a plot in question is near a city's center, its price rises, if according to the local plan it is a piece of land where flats might be built, the price rises too. There are several such patterns.

An expert might try to name a number of them, and then it would be a basis of a system which could tell how much a land piece might costs depending

on its characteristics, or the interested clients needs. The problem is that it very hard to name these patterns.

A different way would be to check a large number of selling transactions ant try to find these patterns. This might be done automatically with decision tree inference.

Therefore, the basic goals of including this tool in the Invisip project, was to show how Artificial Intelligence (AI) tools might be used in the standard site planning process.

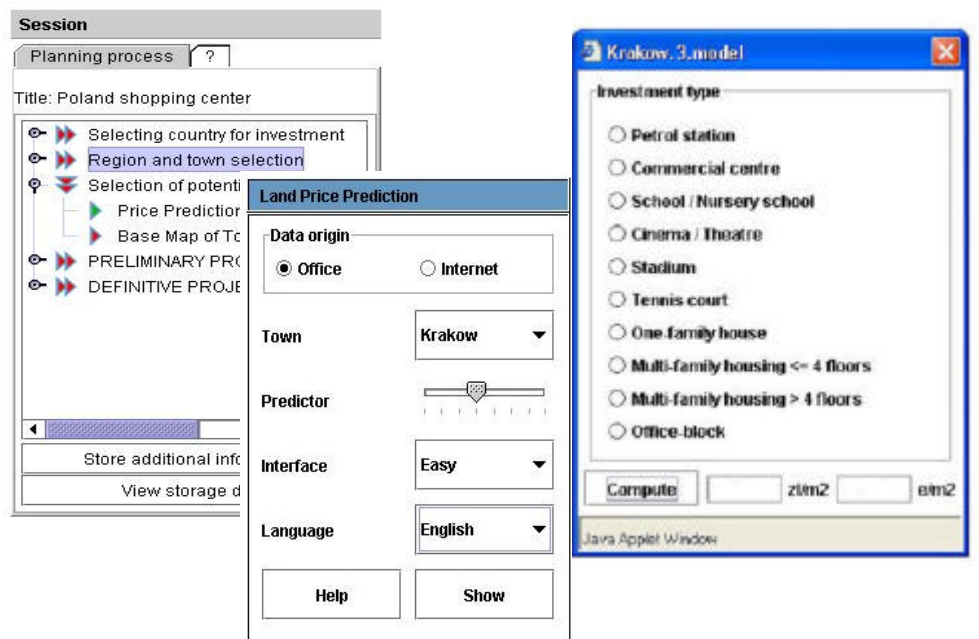


Figure 36 - The screen shot of Land Price Prediction Tool

#### 4.2.1 The decision tree methodology

This is a methodology based on the *divide-and-conquer* paradigm. Nodes in decision trees (Figure 37), built during learning from a set of independent

examples, involve testing particular attributes, usually with a constant. Depending on the outcome, other tests are selected, until the process reaches a leaf node. A leaf node gives a classification that applies to all instances that reach that leaf, or a set of classifications, or a probability distribution over all possible classifications.

To classify an unknown instance, it is routed down the tree to the values of the attributes tested in successive nodes, and when a leaf is reached, the instance is classified according to the class assigned to that leaf.

Attributes are basically of two main types: nominal, which can take on one of a number of set values, or numerical. If the attribute tested is nominal, the number of children of that node is frequently equal to the number of possible values, and one branch is then chosen. Numeric attributes are usually tested against a given constant, and a node with such a test might have two children: one for instances with attribute values below the constant, and one for these above. The actual number of children depends on the tree induction methodology and, naturally the data used during training.

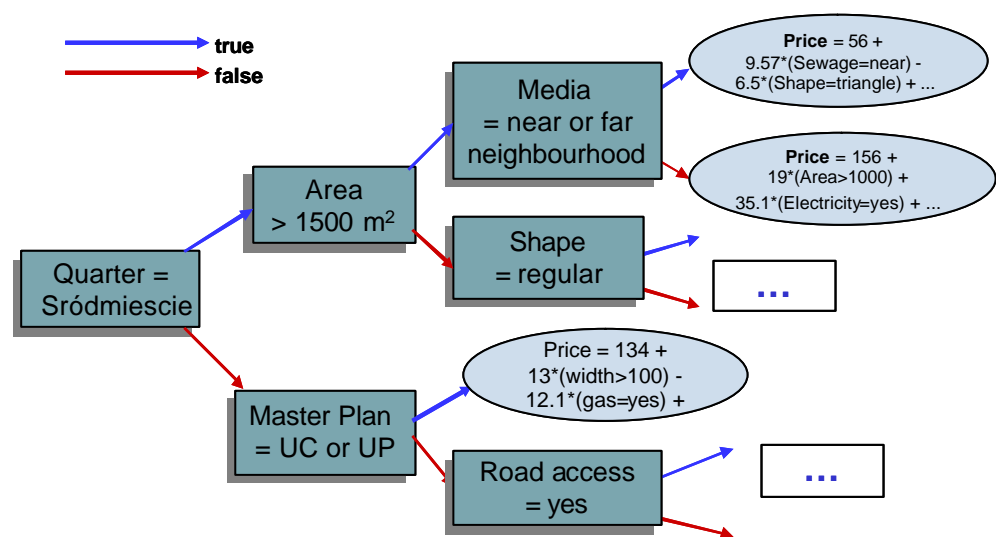


Figure 37 – Land Price Prediction Decision Tree

The most well known methods for tree induction is Quinlan's C4.5 [Quinlan].

Basically decision trees have constant values in their leaves, which are used for classification. In case of numerical prediction, and the land price prediction is such a case, two different methods might be used a number might be stored as classification, this approach does not give high accuracy instead of a single number, a formula that depends on all attributes present in the training set might be stored in the leaf; if the leaf is reached the formula is evaluated with the actual values in the instance; usually a linear regression is used

In our solution we have used the M5' [Witten] which build a regression tree from a set of examples. This methodology is a modification of the original decision trees which are suited only for classification in a set number of classes, for the case of numerical data prediction. In this approach a decision tree is built with tests on different attributes of the input data being performed at each step, and thus subdividing the data set. In the leaf nodes, unlike to the traditional approach, linear regression formulas are stored which are built using the data from the training set that reached that node. It is a quite successful methodology resulting in good generalization ratio, which is the accuracy of predictions for previously unseen data, i.e. data not included in the training data set.

The examples were taken from actual transactions in two different areas:

- the city of Krakow
- the gmina of Zabierzow, near Krakow

The following fields are present in the data:

- city quarter
- transaction date

- area, width, length, and the shape of the plot in question
- several local site plan codes
- information whether electricity, gas, water, sewage system, heat supply, and telephone are available, and if so, in what form
- what is the usage type of the plot, e.g. Habitable, industrial, commercial
- the transaction type, e.g. free or restricted market or tender
- whether there is a road access and the price of one square meter of the plot.

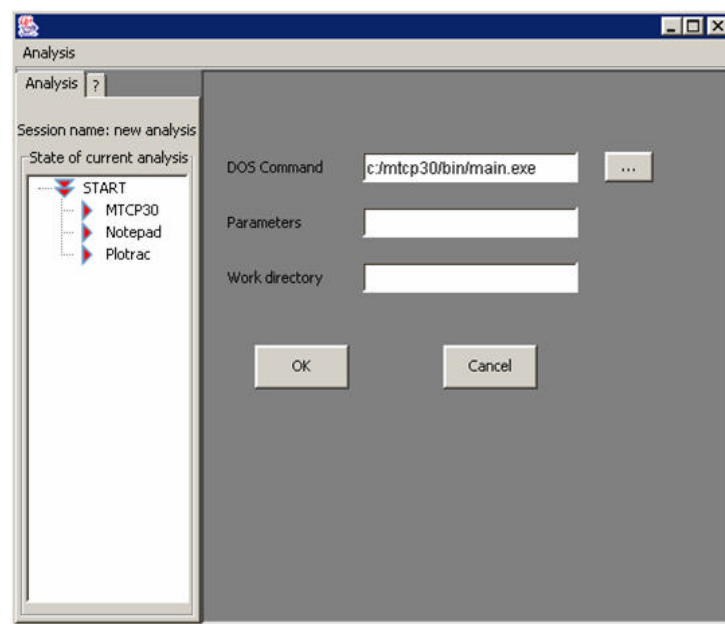
The data included around 2000 examples for both the Krakow and Zabierzow cases. The data were of good quality, with not too much missing values.

## 5 Configurator

### 5.1 The Configurator module

The Configurator is the component that follows the user during the site planning on his local machine. The graphic user interface of the Configurator is very similar to the Control Unit.

Technically the Configurator is a Java applet signed because it wants to access the local machine of the user.



**Figure 38 – The Configurator**

The left side of the Configurator shows the list of steps that the user has to do in order to complete the local analysis. For example, in the Figure 38, the local analysis consists of three steps:

- MTCP30 is the application involved into the traffic analysis;
- Notepad is the application dedicated to show the results;
- Plotrac is the application that shows the traffic flows.

On the right side of the window, the user has to select the application dedicated to the analysis and its input parameters and the working directory.

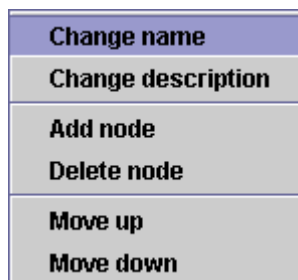
In the menu of the Configurator (Figure 39), the user can:

- create new analysis (selecting “**New**”);
- load an analysis reading a file on local machine (selecting “**Load**”);
- save the current analysis on local machine (selecting “**Save**” and “**Save as**”);
- load an analysis from INVISIP website;
- save the analysis on INVISIP website (selecting “**Save to URL**” or “**Save to URL as ...**”).



**Figure 39 – The menu of the Configurator**

When the user selects the right click of the mouse on a step of the analysis, the Configurator shows the popup menu (Figure 40).

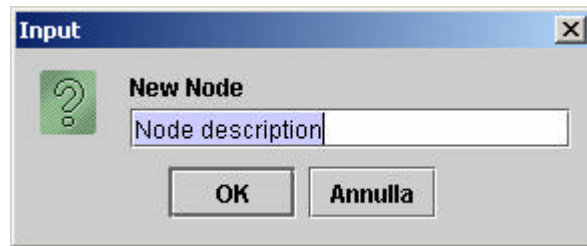


**Figure 40 – The popup menu**

Using the popup menu, the user can the interface of a step of the analysis. In fact when the user selects “**Change name**”, the title of the step of the



analysis is changed. When the user selects '**Change description**', the input dialog for the new description is shown (Figure 41).



**Figure 41 – The input dialog for the new description.**

The menu "Add **node**" or "**Delete node**" add or delete a step in the planning process.

The menu "**Move Up**" and "**Move down**" changes the order of the steps in the analysis.

## 6 Summary / Conclusion

The Demonstrator 2 of the Analyser is the implementation of the “Architectural Concept of the Analyser” described in Work Package 4.

The Demonstrator 2 gives more capabilities to the Analyser:

- the Control Unit is completely integrated with Visual Data Mining;
- the Visual Data Mining has new functionalities such as Highlighting on Brushing and Linking;
- all the tools of the Analyser are integrated.

The main result of the Analyser is the integration of different tools that show to the user different aspects of the same documents. Demonstrator 2 shows that it is possible to merge the results given by the Visual Data Mining and the other tools provided, using the INVISIP platform. The user has an effective step-by-step support in his daily site planning analyses.

## 7 Appendix 1

In this appendix we try to summarize the core concepts of Visual Data Mining module (VDM). It does not provide a complete description of technological design, but it aims to describe the concepts that should be known to integrate a visualisation, providing also an example (see Appendix 2).

### 7.1 The new VDM Module Design

New functionalities have been added to the new VDM design in order to support a complete implementation of Brushing and linking techniques (B&L). The most relevant one is the highlighting: whenever user selects a sub-set of available data by interacting on a visualisation, the corresponding subset of data is highlighted in all open visualisations. This allows at a glance to recognize characteristics of the subset providing useful insight in the user's exploration.

### 7.2 New VDM module classes

In the previous VDM version (without B&L), three different components were involved in the selection operations:

- A *Visualisation* abstract class that provides a template, each visualisation (Histogram, Parallel coordinate Diagram and so on) had to adhere to.
- *Control Panel* that manages the graphical user interface and initialises visualisation(s) and the data container (*CheckResultSet*).
- *CheckedResultSet* class that contains input and provides instruments to perform data reduction.

In order to support Brushing and linking with highlighting, some new classes have been developed:

- *CheckedResultSetBL* extends *CheckedResultSet* to manage the subset of highlighted data.
- *VisualisationBL* extends *Visualisation* abstract class and provides template for visualisation that support Brushing and Linking.
- *SelectionInformation* provides the information about who started the event and the kind of selection that has been performed, that is, selection or B&L.

Moreover some modifies have been needed, as follows:

- *ControlPanel* has to initialise a new data manager instance of *CheckedResultSetBL* instead of *CheckedResultSet*, it also opens visualisations that are instances of *VisualisationBL*.

To understand how the B&L works, let focus on a case study that shows how to add a new visualisation to the new design.

## 8 Appendix 2

In this appendix we describe how to integrate a visualisation that adheres to the design of VDM providing B&L with highlighting

### 8.1 Case study

The addition of a new visualisation follows the steps below:

1. Creation of a new visualisation.
2. Brushing and linking selection: this is performed each time a user interacts with a visualisation using a “feature”. Such a selection can be performed by selecting a graphical feature or an attribute values on the legend.
3. Selection: This is performed each time a user decides to reduce his result set according to the graphical selection he made. A button in the menu-bar of each visualisation lets the user perform it.
4. Undo the last Selection.
5. Selection after a change of (Visualisation) focus, whenever user starts to select i.e. features on a new visualisation.

Now we focus on each step, providing a brief description of how the system should behave:

1. A new visualisation is created and via its constructor `check_result_set` (instance of *CheckedResultSetBL*) it is read and visualized.
2. Brushing and linking selection
  - Each visualisation catches the event raised by mouse clicking and calls method `selectionBL()` that it has to implement. Later on, it communicates the selection to the attribute `checkedResultSetBL` via `selectHighlightedData(LinkedList[] pairs, VisualisationBL who)` method. The second parameter should be the pointer to the visualisation that calls the B&L selection. (JAVA “this” pointer)

- `SelectHighlightedData` performs the instructions that are needed to update the highlighted subset of data.
- `SelectHighlightedData` also raises the notification to the open visualisation by `NotifyObservers`. This method requires as parameter an object that also is an instance of `SelectionInformation`. Such an object specifies who started the event (parameter `who`) and if it is a B&L event (`highlightNotify` flag).
- Each open visualisation receives its notification and executes the update method. It is also possible to recognize which kind of event has been called and the `highlightNotify` flag provides this information: the flag is false whenever the updating has been caused by selection/undo event, whereas it is true when brushing and linking event occurs.
- B&L Selection and B&L unselecting of data are similarly treated because an unselecting has been processed, as if it was a new selection.

### 3. Selection

- Once the visualisation method `select()` has been called, it checks which graphical features are currently selected (highlighted) and sends this information to Control Panel via `select(...)` methods.
- Then `VisualisationBL.checkedresultsetBL.select(LinkedList[] pairs, VisualisationBL who)` sets highlighted attributes to an empty result set, the selection of `checked_result_set` is performed, and a notification is sent to all open visualisation.
- The changes are notified to all open visualisations by `NotifyObservers`, specifying (as instance of `SelectionInformation`) when the highlighting flag is set to false.
- Once the notification is received, each open visualisation executes the update method.
- After a selection no graphical features is selected (highlighted).

### 4. Undo

The *undo()* method is called by a menubar or a Control Panel Button and it calls a method provided by `VisualisationBL.checkedresultsetBL.undo()`

- The *undo()* method sets highlighted attribute to empty and notifies that a change on data has been performed (the parameter *who* of *SelectionInformation* has null value, and the *highlightNotify* flag is false).

5. Nothing happens, all selections and highlighting are kept until user performs a BL selection on the new focus owner.

## 9 References

### 9.1 Literature

[Albertoni et al. 2003] R. Albertoni, A. Bertone, U. Demšar, M. De Martino, H. Hauska, "Knowledge Extraction By Visual Data Mining Of Metadata In Site Planning", *Proceedings of the 9<sup>th</sup> Scandinavian Research Conference On Geographical Information Sciences*, 4<sup>th</sup>-6<sup>th</sup> June 2003.

[Witten 1997] Y. Wang and I. Witten, "Inducing model trees for continuous classes.", *Proc of Poster Papers, 9<sup>th</sup> European Conference on Machine Learning*, Prague, April

[Weiss 2003] Dawid Weiss, Jerzy Stefanowski, "Web search results clustering in Polish: experimental evaluation of Carrot ". *Advances in Soft Computing, Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM'03 Conference, Zakopane, Poland, vol. 579 (XIV), 2003, pp. 209-220*

[Quinlan, 1988] J. R. Quinlan, "Simplifying Decision Trees", *Knowledge Acquisition for Knowledge-Based Systems*, Academic Press, London, B. Gaines and J. Boose, pp 239--252, 1988