User manual for the MAISA metric tool

Version β

Juha Gustafsson and Lilli Nenonen

Helsinki October 13, 2000

UNIVERSITY OF HELSINKI
Department of Computer Science
MAISA project

Contact information

Jukka Paakki:	email Jukka.Paakki@cs.helsinki.fi
Inkeri Verkamo:	email Inkeri.Verkamo@cs.helsinki.fi
Juha Gustafsson:	email Juha.Gustafsson@cs.helsinki.fi
Lilli Nenonen:	email Lilli.Nenonen@cs.helsinki.fi

TEKES
Erkki Hietanen: email Erkki.Hietanen@tekes.fi

Kone
Markku Kakko: email markku.kakko@kone.com

Nokia Mobile Phones
Markku Ilvesmäki: email Markku.Ilvesmaki@nmp.nokia.com

Nokia Research Center
Anssi Karhinen: email Anssi.Karhinen@nokia.com

Space Systems Finland
Niklas Holsti: email Niklas.Holsti@ssf.fi

Version history

Version	**Date**	**Changes / Comments**
0.0	10.10.2000	The first version.

Contents

1 Introduction

The MAISA metric tool is designed to measure object-oriented software architecture descriptions that are given in the UML language. The measurement can thus be done in early phases of the software development process and the software life cycle. The tool combines ordinary measures with design pattern and anti-pattern recognition and makes use of known design patterns while measuring the UML description.

In Chapter 2 we present general ideas behing MAISA. In Chapter 3 we explain how to installate the MAISA tool in your personal computer.In Chapters 4 - 6 we tell how to use the windows of the MAISA tool. In Chapter 8 we tell how to use MAISA without the graphical user interface.

The MAISA metric tool is implemented as a result of the MAISA research project in the University of Helsinki. The project is sponsored by TEKES, Kone oyj, Nokia Mobile Phones, Nokia Research Center and Space Systems Finland.

This is the User manual for MAISA version β.

2 What MAISA is

The MAISA tool is intended to be used by software designers, test designers and quality managers. The MAISA user is expected to have basic knowledge on object-orientation, UML, design patterns, measurement and software quality, which helps him/her to understand basic concepts of the tool. However, he/she needs not to be an expert on these.

The MAISA tool is intended for measuring UML diagrams, which have been previously constructed by another tool. It is not a CASE tool or a diagram drawing program. It reads UML descriptions and prints out the measurement results but it does not modify the diagrams.

In addition to measuring single UML diagrams, the user can compare two existing diagrams. The metric results can be saved separately from the diagrams and later compared to each other.

The MAISA tool recognizes design patterns and anti-patterns that are used in the given software architecture. The tool has a pattern library, which contains information of both the structure of patterns and their influence on the properties of the architecture. The user can browse the pattern library. A professional user may also need to edit and update the library.

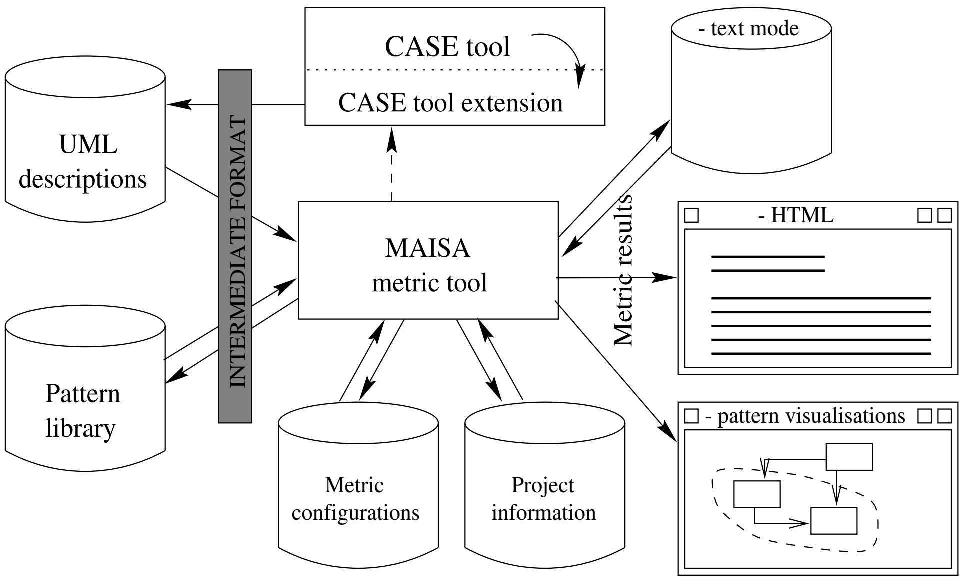

Figure 1: Overall architecture of MAISA.

Every measure consists of metrics. The user can browse the metrics and see what the measurement is based on. Because some of the metric results are estimates rather than accurate values, browsing the metrics is important especially for the professional user, who also may need to parametrize the metrics to tailor the tool for his/her needs. It must be noted that the MAISA tool calculates *accurate values of some metrics* and – in addition to that – *predicts the values of the other metrics*. In other words, the purpose of the MAISA tool is both calculating metric values of the architecture and foretelling some properties of the to-be software product.

Figure 1 describes the architecture of the tool in a general level. The solid arrows present data flow and the dashed one presents control flow. Because UML is a visual language, the diagrams must be saved in an intermediate file format which is analyzable and measurable. The UML diagrams are drawn with a separate CASE tool which generates the diagrams in the chosen intermediate format. In fact, an extension to the CASE tool produces the format, because at present no commercial CASE tool supports the intermediate file format used by the MAISA tool. The MAISA tool reads this representation and gives the results in textual form. The results can also be presented in HTML. The pattern library is separate from the UML descriptions. In addition to the metric results, the MAISA tool prints out a couple of other files used by the tool itself, not directly by the user (project and metric configuration files, discussed later). The control flow presents the possibility to run the CASE tool via the MAISA tool. Figure 1 lacks information of the files used only by the tool itself, never by the user.

The MAISA tool is a single user program and there are no usernames or passwords. The pattern library is implemented as a group of text files, not a database. Also the measurement results are saved as simple text files. The tool is implemented in Java^{TM}. It is designed to be run in Windows NT and Windows 95 environment, but the implementation language does not prevent running the MAISA tool on other operating systems.

3 Installation

The MAISA system can be downloaded in the web site of the MAISA project, which is at: `http://www.cs.helsinki.fi/group/maisa/`. The β version is delivered in two zip packages. The first package contains the actual program files and the other package contains the pattern library and on-line help files.

Windows:

1. Install the Java^{TM} 2 Runtime Environment if you don't have it already. It can be downloaded at `http://java.sun.com/products/jdk/1.2/jre/`.
2. Copy the zip packages into your computer and unzip them into `C:\maisa\`. This is called your maisa directory.
3. Add the above-mentioned directory in your classpath (an environment variable used by the Java^{tm} system). In Windows NT, Start Control Panel, select System and click the Environment tab. Set the CLASSPATH variable in the User Variables section. In Windows 95 or 98, add the classpath to the autoexec.bat file.

Linux:

1. Install the Java^{TM} 2 Runtime Environment if you don't have it already. It can be downloaded at `http://java.sun.com/products/jdk/1.2/jre/`.
2. Copy the zip packages into your home directory as `~/maisa/`. This is called your maisa directory.
3. Add the above-mentioned directory in your classpath (an environment variable used by the Java^{TM} system) with command `setenv CLASSPATH $CLASSPATH:$HOME/maisa/`.

Your maisa directory should now contain the Java^{tm} class files, the default metric configuration file and the pattern library and help directories.

Finally, the tool is started with command `java MAISA.MAISA_tool`.

If you do not need the graphical user interface (see Chapter 8), you do not need the class files of the directory `maisa/MAISA`, but you still need the directories `maisa/MAISA/`, `maisa/MAISA/datastruct` and the class files in the latter directory.

4 Main window

The MAISA tool is started with command `java MAISA.MAISA_tool` When the MAISA tool is started, the main window is opened. In this chapter we tell how to manage diagrams, projects, metric results and comparison results in the main window. Initially the main window is empty. Also other directories can be used, but their contents are not loaded automatically. The tool contains three basic windows: the main window, the metric window and the pattern (library) window.

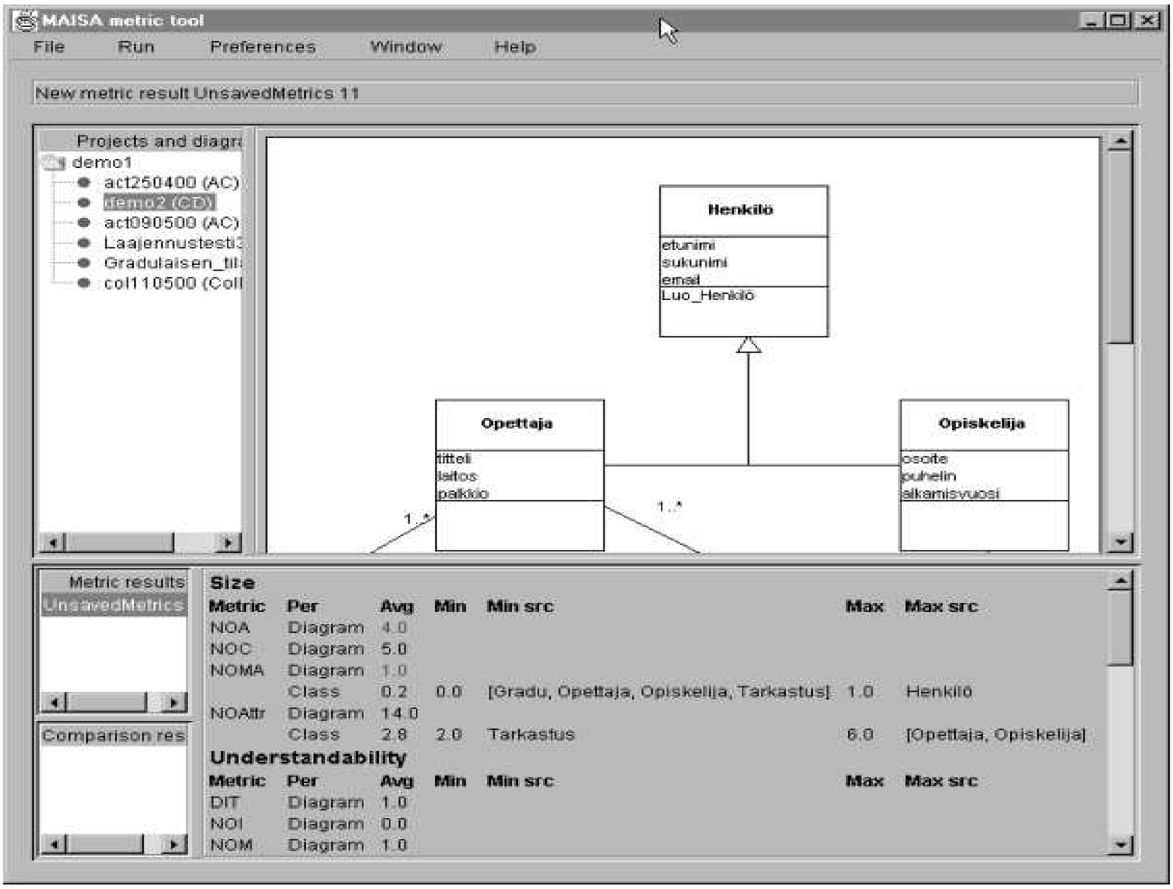

Figure 2: The main window.

Figure 2 presents the main window, with some projects, diagrams and metric results loaded in it. In the upper left part of the window there is a list of projects. A project contains one or more diagrams – the structure is shown in the tree-shaped list. Single diagrams (the diagrams that do not belong to any project) that are loaded in the window are shown under the projects in the list. We tell more about the projects in Chapter 4.1. In the lower left part there is two lists: one of the metric results, the other for the comparison results. We tell more about them in Chapters 4.2 and 4.3. The diagram that is selected in the project list is shown in the upper right part of the window. The metric result or the comparison result that is selected in the other lists is shown in the lower right part of the window. Only a metric result *or* a comparison result can be selected at a the same time. The borders between the lists ans panels can be dragged according to your needs.

Under the main menu there is status bar. The messages and error messages given by MAISA are written in the status bar. The main menu contains five menus: File, Run, Preferences, Window and Help. The File menu contains common file operations. The selectable operations that cannot be performed in the moment are disabled. They menus are classified by which files the file operations handle (projects or metric results):

Open: Project / Diagram Opens a project or a diagram in the project list. Both directories and single files can be opened: if you open a directory, it is concidered as a project; if you open a prolog file that ends with `.pro`, it is concidered as a diagram. If a project is selected when opening a diagram, the diagram becomes a property of the selected project during the session. If a diagram or nothing is selected when opening another diagram, the diagram is opened as a single diagram, not belonging to any project.

Open: Metric / Comparison result Opens a metric result in the metric result list or a comparison result in the comparison list. Only single files can be opened. If the file ends `.res`, it is considered as a metric result; if the file ends `.cmp`, it is considered as a comparison result. A project or a diagram must be selected in the project list before opening a result. The opened result becomes a property of the selected project or diagram during the session.

Close: Project / Diagram Removes the selected project or diagram (with their results) from the project list.

Close: Metric / Comparison result If a metric result is selected, removes the selected item from the metric result list or if a comparison result is selected, removes the selected item from the comparison list.

Save: Metric / Comparison result Makes a copy of the selected metric result of comparison result. Asks the user the file name where to copy.

Exit Exits the program.

The Run menu contains the measurement commands:

Measure Measures the selected project or diagram using the current metric configuration. A project ot a diagram must be selected before performing the operation. The calculated metric results appear in the metric list.

Compare Compares the two selected projects, diagrams, metric results or the selected combination of them. The two items must be selected before performing the operation. The calculated comparison results appear in the comparison result list. If two archotectures are selected, they are compared. Otherwise, if two metric results are selected, they are compared. Otherwise, the selected architecture and metric result are compared.

The Preferences menu contains the user settings of the tool. All the settings can be edited in the preference window. The boolean valued preferences can be edited also directly by selecting them in the menu. Note, that updating the preferences in the preference window saves the preferences in the disk, but setting the boolean valued preferences in the preference menu does not save them in the disk before you click the save item in the menu.

Edit... Opens the preference window where you can edit all the preferences.

Check unsaved files before closing A boolean preference. If it is selected, the program asks you a confirmation if you really are closing a result file that is not saved in the disk or closing an architecture containing result files that are not saved in the disk.

Check unsaved files before exiting A boolean preference. If it is selected, the program asks you a confirmation if you really are exiting the program but there is open files that are not saved in the disk.

Confirm pattern / pattern group deletion A boolean preference. If it is selected, the program asks you a confirmation if you really are deleting a pattern or a pattern group in the pattern library window.

Save Saves the preferences in the disk.

The Window menu contains commands to open the two other basic windows of the tool:

Measures and metrics Opens the metric window.

Pattern library Opens the pattern library window.

The Help menu gives help to the user:

Help... Opens the help window.

About Gives information of the MAISA project.

4.1 Architectures: projects and diagrams

A project is a set of diagrams. Grouping diagrams into projects is a way to measure more than one diagrams at the same time. A project can include different kinds of diagrams, one or more diagrams of a kind. The supported UML diagram types are *class diagram, activity diagram, state transition diagram, sequence diagram* and *collaboration diagram*. MAISA supports also *extended activity diagrams* that contain information of performance requirements [Gus2000]. A project is also a directory, containing all the diagrams belonging to the project. The name of the directory is the name of the project. **A diagram** is prectically a prolog file, presenting the information of a UML diagram. There can also be a figure file showing the visual diagram. Both projects and diagrams are called **architectures**. The project directory can contain following files read by MAISA, all the other files in the directory are ignored:

Diagram files in prolog format, named `xxx.pro` where xxx is the diagram name. Generated by an external CASE tool. These are the only files in the project directory that are essential for MAISA and metric calculation (actually, you don't need any other files to calculate metrics).

Diagram figure files in gif format, named `xxx.gif` where xxx is the diagram name. Generated by an external drawing tool. These are for the human user and for the user interface.

Metric result files the name of which ending `.res`. Produced by MAISA.

Comparison result files the name of which ending `.cmp`. Produced by MAISA.

You can open both projects and single diagrams in MAISA. The Open operation is located in the File menu. If you open a directory, MAISA considers it as a project – if you open a single file, it is considered as a diagram. You can also add new diagrams to a project or remove diagrams from a project. If a project is selected when opening a diagram, the opened diagram is added to the project; if a diagram is selected when opening another diagram, the diagram is opened as a single diagram, not belonging to any project. Architectures can also be removed from the project list by selecting the architecture and clicking the Close item in the File menu. Note, that these modifications are only temporary and they exist only during the session: using open and close operations do not modify the directory structure.

4.2 Measuring

To measure a diagram or a project, select the architecture and then click the Measure item in the Run menu. The metrics that are calculated are set in the metric window. The colelction of metrics that are calculated is called **metric configuration**.

Projects are the way of measuring more than one diagrams at the same time. A project can include different kinds of diagrams, one or more diagrams of a kind. However, certain metrics cannot be measured from every kind of diagrams. E.g. if number of classes (NOC) is calculated from an activity diagram, the result is zero. In Chapter 5 we list the different metrics and the diagram recommendations for them.

Diagrams are measured simply by calculating the metrics from the diagram (prolog) file. When a project is measured, the metrics are calculated from all its diagrams at the same time. The diagrams are unified and the metrics are calculated from these "unified diagrams", whenever possible. Class diagrams containing common classes can be unified using the common classes. Thus, the common classes are taken into account in metrics calculation only once. For example, a class diagram can present a class library, referenced from other class diagrams. The activity diagrams are unified in the same way. An activity diagram can present a procedure called from other activity diagrams. Note that unification is not yet in use in version β.

The metrics are given in certain relations, i.e. per diagram, per project and/or per a diagram element: per class, per object and so on. A calculated metric result (per an architectural element) is actually a set of results: minimum, maximum and average values and the element(s) that has been the source of those values. For example, number of associations (NOA) can be given per project, per diagram and per class. Per class the minimum value is e.g. 1, the maximum is 9 and the average 4.8 and the minimun source is e.g. class A and D (classes A and D both have only 1 association) and the maximum source class z (class Z have 9 associations). In the Figure 2 you can see how the metric values are listed. One row contains the name of the metric (header Metric), the relation of which it is given (Per), average value (Avg), minumum value (Min), the source of the minimum value (Min src), the maximum value (Max) and the source of the maximum value (Max src). The recognized patterns are listed below the other metric results. A pattern instance is a set of bindings between the pattern roles and architectural elements (i.e. classes, associations and so on). The recognized instances are numbered and the number (Header Instance x, where x is the number) and the bindings of them are listed in the window under the name of the pattern.

The measured values that are not within the user given thresholds (if the thresholds are given for the metric) they are written in red colour in the result panel – in other case they are written with black colour. The thresholds and other metric settings are given in the metric window.

Let's study a brief example. Figure 3 presents a class diagram of a company data system. The company is divided into departments and the employees and the managers who are persons, belong to a department. The departments use subcontractors, which are presented in Figure 4. The subcontractors produce products, which are composed of parts. The class Subcontractor in the latter class diagram is referenced from the former class diagram, and because is common to both class diagrams, they can be unified. The unified class diagram is shown in Figure 5.

As an example, the Number Of Classes (NOC) per the example project is 8, average NOC per diagram is 4.5.

MAISA supports a set of software metrics, traditionally calculated from source code. In MAISA they are applied to the UML diagrams. The metrics privided by MAISA are explained in Chapter 5

4.3 Comparing

To measure a diagram or a project, select two architectures in the project list, two metric results in the metric result list or an architecture and a metric result (comparison results cannot be compared). Then click the Compare item in the Run menu.

Comparison is performed by first measuring the possible selected architectures with the current metric configuration, and then comparing the results. Values of only those metrics who are common to both metric results are compared. If values are the same they are written with blue colour in the result panel – in the other case they are written with black colour. Note, that comparing concerns only the ordinary metrics, not the recognized design pattern instances and their role bindings. All the other metric values can be compared.

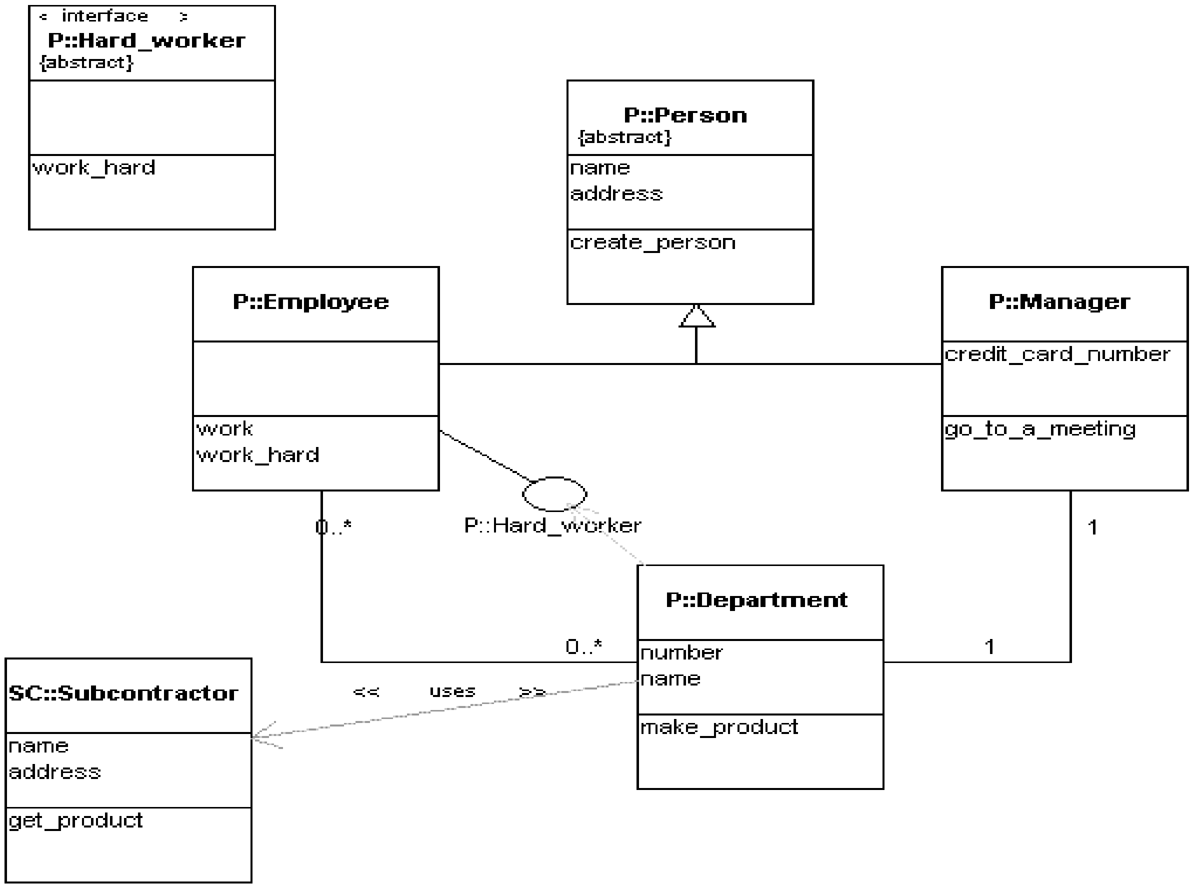

Figure 3: The company class diagram.

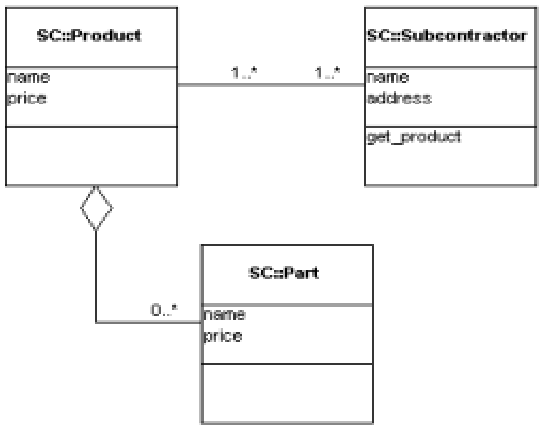

Figure 4: The subcontractor class diagram.

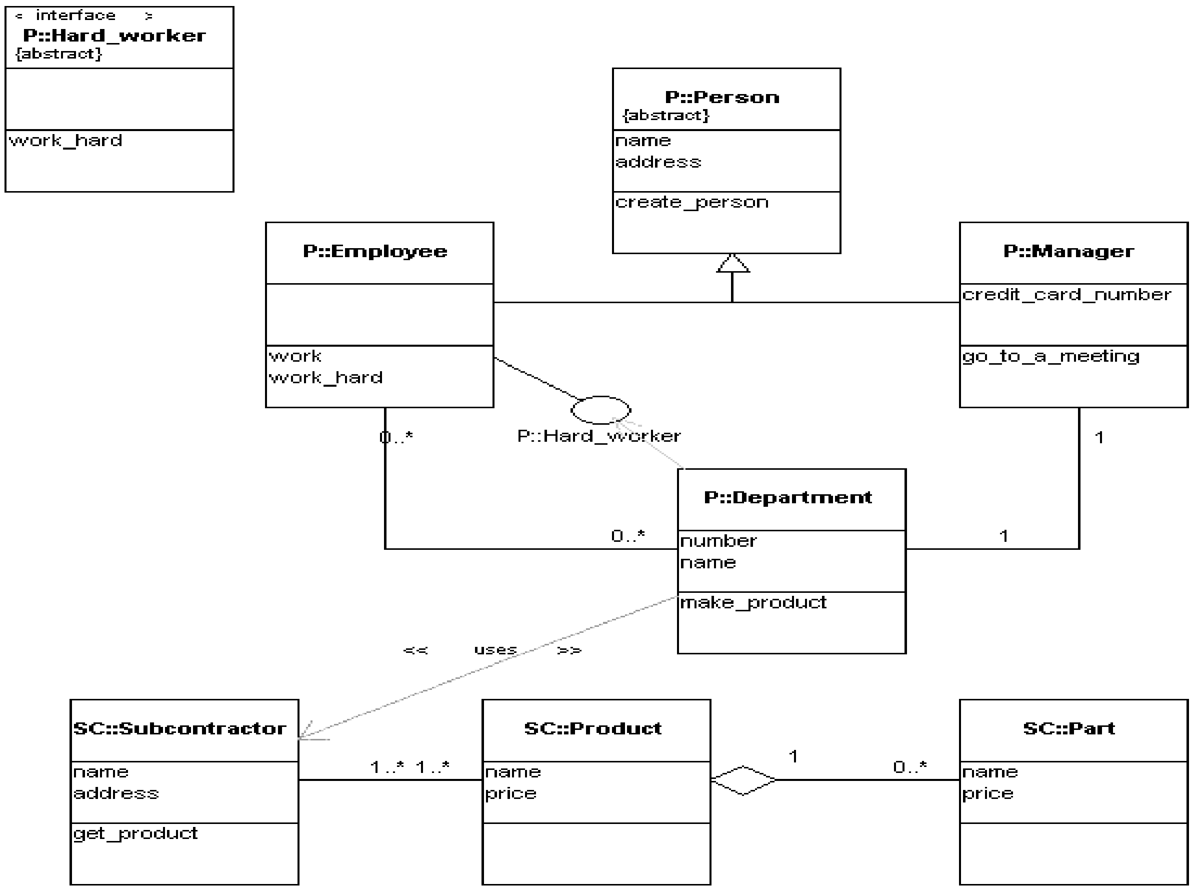

Figure 5: The unified class diagram.

4.4 Preferences

The user settings provided by MAISA are the following:

- **Confirm unsaved when closing** Boolean. If true, the program asks you a confirmation if you really are closing a file that is not saved in the disk.
- **Confirm unsaved on exit** Boolean. If true, the program asks you a confirmation if you really are exiting the program but there is open files that are not saved in the disk.
- **Confirm pattern / pattern group deletion** Boolean. If it is selected, the program asks you a confirmation if you really are deleting a pattern or a pattern group in the pattern library window.
- **Pattern directory** String. The path of the pattern library directory.
- **Default metric configuration** String. The path of the default metric configuration file.

You can edit the preferences with the preference window which is opened by clicking the Edit item in the Preferences menu. The boolean valued preferences can be set directly in the Preference menu.

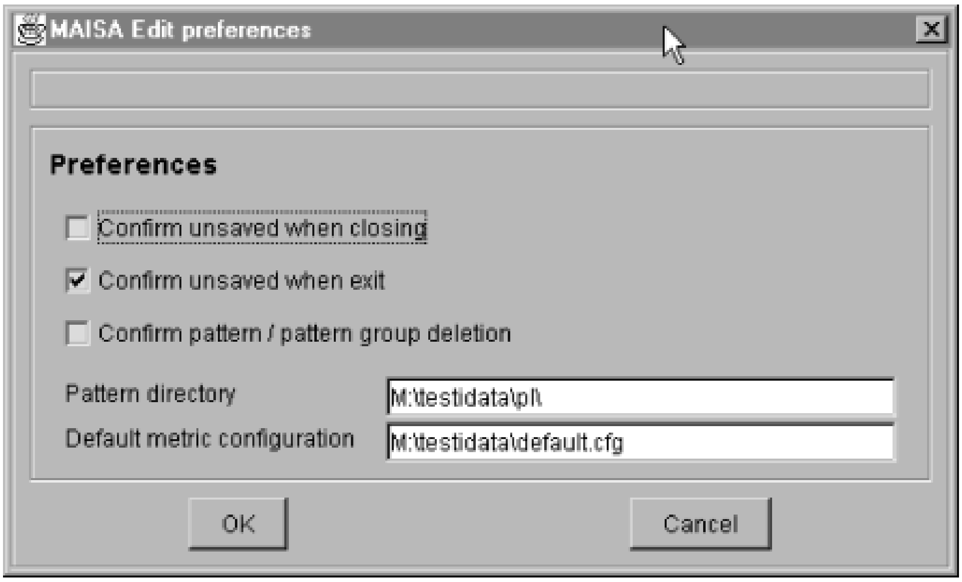

Figure 6: The preference dialog.

Figure 6 shows the preference window. The boolean valued preferences are on the top of the window and the string valued ones are below them. On the bottom on the window there are two buttons: Cancel and Ok. The preference values are saved in the disk with pressing Ok and the setting modifications are cancelled with pressing Cancel. If a text field is left empty, no value is given to the preference and a default value is used. The file names (pattern library directory, default metric configuration) are taken into use the next time you start the tool. The boolean values are in use immediately.

5 Metric window

In the metric window you can update the current metric configuration. The default metric configuration file is saved in your maisa direcdtory as `default.cfg`. Also other configurations can be used and edited in the metric window. The metric configuration contains up to three **measures**: Size, Performance and Understandability. Each measure consists of one or more **metrics**. Pattern Recognition is a special measure: on the other hand, some metrics belonging to other measures are based on pattern recognition, but on the other hand, there are some metrics connected to the pattern recognition. Thus, pattern recognition is a measure but at the same time it is an asset to another metrics, because the recognized patterns can be used as the basis of some metrics belonging to other measures.. In addition to the before-mentioned measures, the configuration contains the recognized patterns and some metrics concerning them.

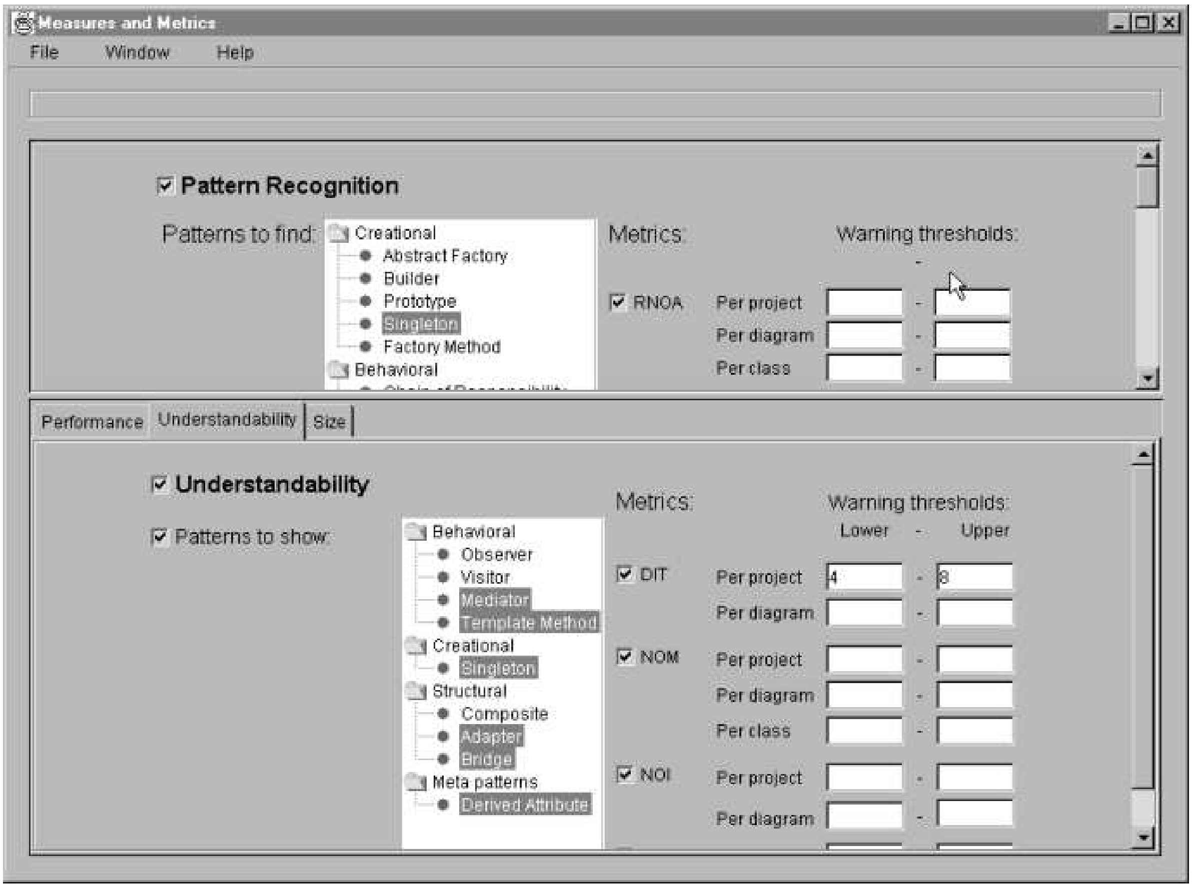

Figure 7: The metric window.

Figure 7 presents the metric window with an example metric configuration. Each measure is presented as a tab sheet. A tab sheet contains a checkbox for each metric belonging to the measure and text fields for upper and lower warning threshold for each metric. Warning thresholds – minimun or maximum or both – can be given for each metric in a certain relation. The user is informed if a calculated metric result is not between the thresholds. The metrics belonging to a measure cannot be selected before the measure is selected. The metrics are given in certain relations, ie per diagram, per project and/or per a diagram element: per class, per object and so on. Also the relations are checkboxes. The metrics are selected to be calculated by selecting the corresponding checkboxes, e.g. number of associations (NOA) per diagram in the metric (tab sheet) “Size”. Any number of checkboxes can be selected at the time.

As a special measure, pattern recognition is located above the tab sheets, and there is not a pattern recognition checkbox – it can be considered always selected. Some metrics shown in the tab sheets are based on pattern recognition. The recognized patterns are selected in the pattern recognition panel, above the tab sheets. The patterns that are used by the pattern based metrics are selected in the tab sheets. Note, that the pattern lists in the tab sheets contain only the patterns selected in the list in the pattern recognition panel above the tab sheets. The border between the pattern recognition panel and the tab sheets can be dragged according to your taste.

Under the main menu there is a status bar. The messages and error messages given by MAISA are written in the status bar. The main menu contains three menus: File, Window and Help. The file menu contains the file operations concerning the metric configuration files:

Open loads a new metric configuration in the window.

Save saves the current metric configuration in the disk with the current file name.

Save as... saves the current metric configuration in the disk with a name given by the user in the file chooser dialog is opened.

Set as default saves the current metric configuration as the default metric configuration.

Reset to default loads the default metric configuration in the window.

Close window closes the window.

The Window menu contains commands to open the two other basic windows of the tool:

Main opens the main window.

Pattern library opens the pattern library window.

The Help menu contains the help command that opens the help window, specific to the metric window.

Note, that certain metrics cannot be calculated from every kind of diagrams. e.g. if number of classes (NOC) is calculated from an activity diagram, the result is zero. In the following chapters we list the supported metrics and their diagram recommendations. The metric algorithms are discussed in detail in our report [NeG2000]. Sequence diagrams are not yet supported in version β.

5.1 Size metrics

Most of the metrics belonging to this category measure numbers of certain architectural elements, such as classes, associations and methods. The values of these metrics can be given in accurate numbers. Only one of the size metrics, lines of code (LOC) is an estimation. It is not yet in use in the version β.

- **LOC** Lines Of Code. An estimated value. Given per project, diagram or class. *Not yet in use.*
- **Pattern based LOC** As LOC, but the calculation based on the recognized patterns. *Not yet in use.*
- **NOA** Number Of Associations (cf. NOAttr). Calculated from class diagrams. Given per diagram and class.
- **NOC** Number Of Classes. Calculated from class diagrams. Given per project and diagram.
- **NOMA** Number Of All Methods (cf. NOM, an understandability metric). Calculated from class diagrams. Given per project, diagram and class.
- **NOMsg** Number Of Messages. Calculated from sequence and collaboration diagrams. Given per diagram and object.
- **NOOb** Number Of Objects. Calculated from sequence and collaboration diagrams. Given per diagram.
- **NOOp** Number Of Operations. Calculated from state diagrams. Given per diagram.
- **NOS** Number Of States. Calculated from state diagrams. Given per diagram.
- **NOT** Number Of Transitions. Calculated from state diagrams. Given per diagram and state.
- **NOAttr** Number Of Attributes (cf. NOA). Calculated from class diagrams. Given per project, diagram and class.

5.2 Performance metrics

The unit of these metrics is the one in the diagrams from which the metrics are calculated. The pattern base estimation is not yet in use in version β.

- **Pattern based estimation** performance estimation, based on patterns. *Not yet in use.*
- **Time requirement** Calculated from extended activity diagrams. Given as best, worst or typical case.

- **Memory requirement** Calculated from extended activity diagrams. Given as best, worst or typical case. *Not yet in use.*

The extended activity diagrams contain extra information used in performance estimation by MAISA. Note, that the typical case is not (necessary) the average case. The activity diagram extension is discussed in our report [NeG2000].

5.3 Understandability metrics

Understandability is not a clear concept to define. The following metrics are widely accepted as predictors of understandability of the source code and the architecture.

- **Design patterns** The recognized pattern that are printed to the user. Every pattern instance contain information on which architectural elements (classes, objects) have roles in the pattern. These bindings are given as role-element pairs.
- **DIT** Depth of Inheritance Tree. Calculated from class diagrams. Given per project and diagram.
- **NOM** Number Of public Methods (cf. NOA in the measure Size). Calculated from class diagrams. Given per project, diagram and class.
- **NOI** Number Of Interfaces. Calculated from class diagrams. Given per project and diagram.
- **McCabe** The McCabe cyclomatic complexity measure. Calculated from state diagrams and activity diagrams. Given per project and diagram.

5.4 Pattern recognition

The metrics belonging to this category indicate pattern coverage, i.e. they measure the propotion of architectural elements that are binded to the (recognized) design patterns. A corresponding size metric can be found to these relative metrics.

- **RNOA** Relative Number Of Associations. Calculated from class diagrams. Given per project, diagram and class.
- **RNOC** Relative Number Of Classes. Calculated from class diagrams. Given per project and diagram.
- **RNOOb** Relative Number Of Object. Calculated from sequence and collaboration diagrams. Given per project and diagram.
- **RNOMA** Relative Number of All Methods. Calculated from class diagrams. Given per project, diagram and class.

- **RNOMsg** Relative Number Of Messages. Calculated from sequence and collaboration diagrams. Given per diagram and object.
- **RNOAttr** Relative Number Of Attributes. Calculated from class diagrams. Given per project, diagram and class.

6 Pattern window

The pattern window provides the user interface to the pattern library, partly outside the metric system. The patterns library contain textual descriptions and user instructions for each pattern, which can be browsed in the pattern window and which are intended to be used by only human users of the library, not the metric algorithms. In addition to that, the pattern library contains formal descriptions of the patterns, written in Prolog language and an index file common to the whole library. These files are used by the pattern recognition algorithm and if you do not add your own patterns in the library, you do not need to handle these formal descriptions at all. Those The patterns are grouped into pattern groups by their nature. The patterns can be updated, removed and added to the library and the groups can be added and removed. Initially the pattern library contains the design patterns presented in the Design patterns book and so called (Pree's) meta patterns. See appendix A to see the full list of these patterns with a brief description of them.

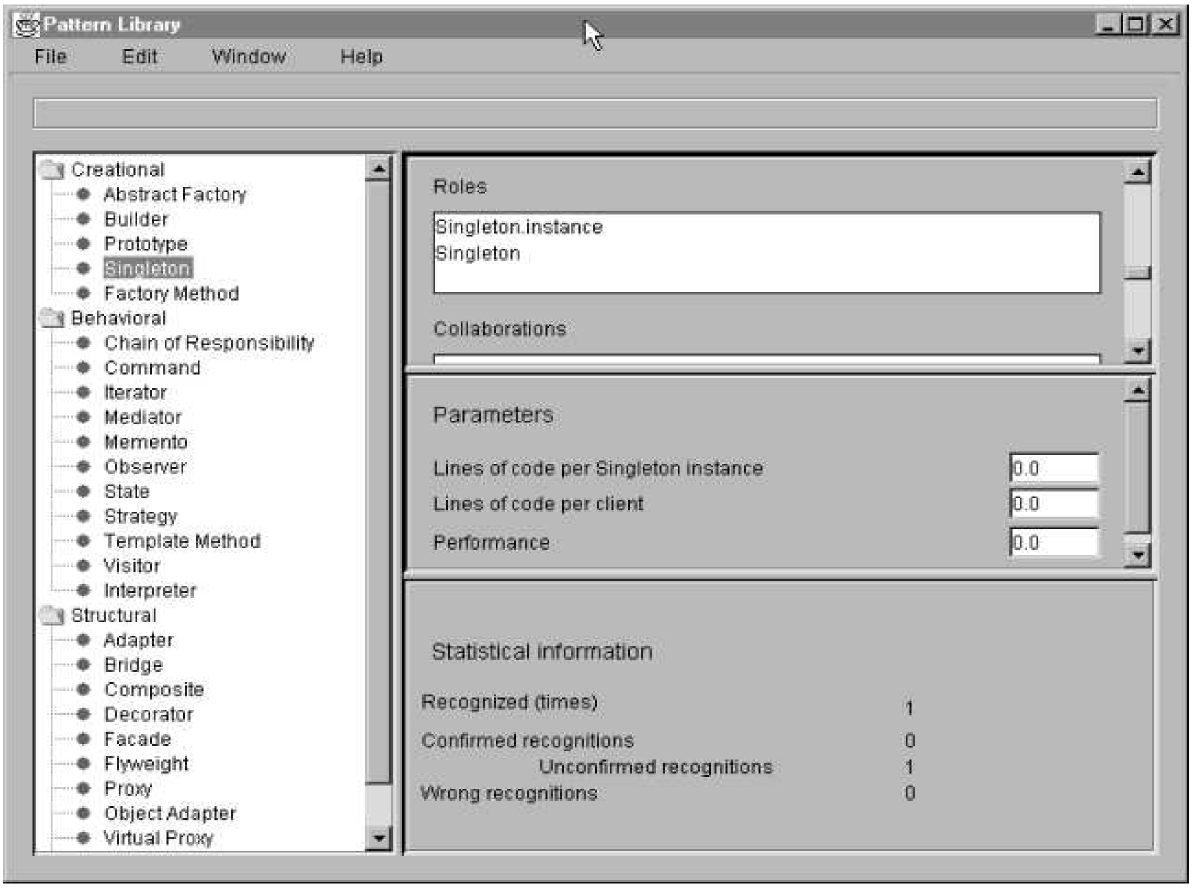

Figure 8: The pattern library window.

Figure 8 presents the pattern library window that works like a HTML frame. On the left there is a tree-shaped list of patterns and pattern groups. The data of the selected pattern is shown on the right side of the window, which is divided into three parts. The borders between the the parts and the pattern list can be dragged. In the upper part there is the textual pattern description, in the middle part there are the metric parameter of the pattern (they are not in use in the version β) and in the lower part there are the statistical values of the pattern: how many times it has been recognized and how many times the recognition has been confirmed or wrong. The pattern description part follows the documentation model used in the Design Patterns book [Gam1995]. The design pattern documentation is discussed in detail in Chapter 6.1.

Most of the operations in the main menu are directed to administrative use of the pattern library. If you just use the existing patterns in the directory, without updating the library, you have little use to the main menu. The file menu contains the operations that do not modify the structure of the pattern library:

Copy copies the selected pattern of pattern group to another directory selected by the user. Note, that only directories can be selected in the file chooser dialog.

Close window closes the pattern library window.

The edit menu contains the operations that modifies the pattern library in the disk:

Edit mode Sets the edit mode on / off.

Save Saves the current pattern in the disk.

Import: Pattern Copies a new pattern to the selected pattern group.

Import: Pattern group Copies a new pattern group to the library.

New pattern Creates a new pattern to the selected pattern group with a name asked from the user.

New pattern group Creates a new pattern group to the library with a name asked from the user.

Delete Deletes the selected pattern group of pattern from the library.

Setting the preference "Confirm pattern / pattern group deletion" true in the main window prevents you from deleting a pattern group or a pattern by accident. To avoid updating pattern information by accident, the window must set into edit mode in order to update the pattern description text. Figure 9 presents the pattern library window in the edit mode. In the edit mode the text areas become editable, the figure file can be set to the pattern and the prolog file of the pattern, the index file of the pattern library appear in the window and you can reset the statistic information by pressing the Reset Statistics button that appear on the right side of the statistic values.

If you administrate the pattern library, you should be familiar with the structure of the physical pattern library. The pattern library in the disk is a directory (it can be set as a preference,

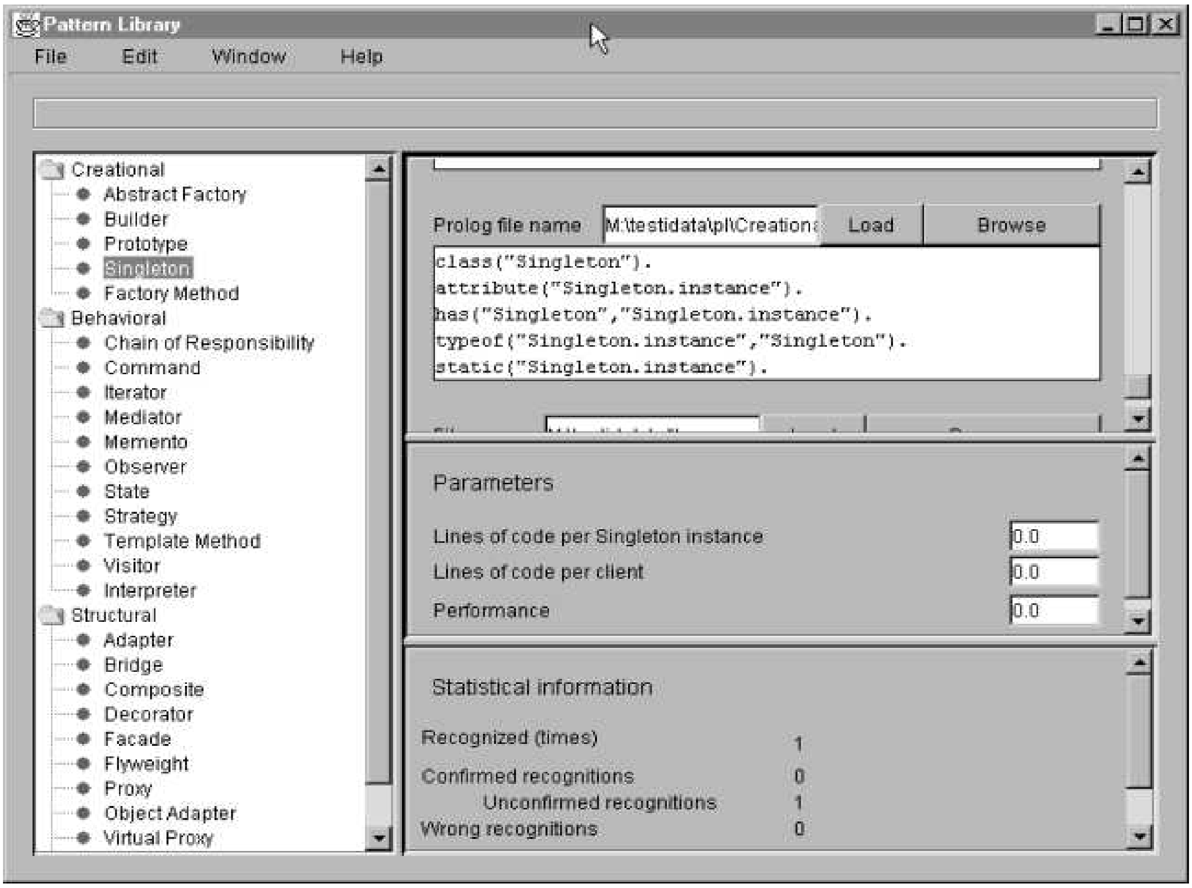

Figure 9: The pattern window in the edit mode.

otherwise a default value for the directory path is used). The pattern library directory consists of pattern group directories and the pattern group directories consist of pattern directories. A pattern directory contains all the necessary files describing the pattern. When importing a new pattern (group) to the library you select the corresponding pattern (group) directory in the file chooser dialog that is opened. The tool copies all the necessary files in the directory for you. The name of a pattern (group) directory is the name of the pattern (group), with the space charecters replaced with underline characters. That name modification is done because all operating systems do not support space characters in the file names. You do not have to care about it or create any directories yourself.

The Window menu contains commands to open the two other basic windows of the tool:

Main opens the main window.

Measures and metrics opens the metric window.

The Help menu contains the help command that opens the help window.

6.1 Pattern description

The textual description of the patterns are divided into sections, following the documentation model presented in the Design Patterns book [Gam1995]. The text in the chapters is given in certain formula. The following list tells what is the content of the sections and in what formula the text is written in the text areas of the pattern window. Every section is optional and can be left empty.

- **INTENT** The meaning of the pattern – a brief description of what it does. Any text except empty rows.
- **ALSO KNOWN AS** A comma-separated list of other names of the pattern, if there is any. For example:

  ```
  Name, Another name, Yet another name, The last name
  ```

- **MOTIVATION** A description of what design problem the pattern solves. Perhaps an example of the problem. Any text except empty rows.
- **APPLICABILITY** A list of points where the pattern can be used. The first row of each item begins with a hyphen. For example:

  ```
  - You can use the pattern if...
  Especially if ...
  In a situation...
  - An another place to use the pattern is...
  - The pattern is extremely effiecient when...
  Btw, there is...
  ```

- **PARTICIPANTS** A list of the roles of the pattern. The role names are written one name per a row. A role name row can be followed with a list of the tasks belonging to the role, each task beginning with a hyphen. For example:

```
Role A
- Task A1
- Task A2
Role B
Role C
- Task C1
```

- **COLLABORATIONS** A list of points where the elements of the pattern co-operate. The first row of each item begins with a hyphen.
- **CONSEQUENCES** A list of consequences of using the pattern. The first row of each item begins with a hyphen.
- **IMPLEMENTATION** Implementation specific tips on using the pattern. Any text except empty rows.
- **SAMPLE CODE** Sample code fragments, with possibly some ordinary text. Any text except empty rows.
- **KNOWN USES** Examples of the pattern found in real systems. Any text except empty rows.
- **RELATED PATTERNS** Pattern's relations to other patterns. Which other patterns should be used with this one. Any text except empty rows.

6.2 Edit mode

When the pattern window is set to the edit mode, the text areas and text fields and the pattern name header become editable. When you update the pattern description text, remember to save the pattern. The modifications are not written to the disk automatically and they are cancelled if you select another pattern without saving the modifications on the former one.

The textual descriptions of the pattern are written in the text areas as described in the previous chapter. In addition to them, you can give a figure file in gif format, describing the structure of the pattern and you must give the prolog file, describing the architectural constraints that a pattern instance must fulfil. The pattern library also has an index file that is common to all the patterns and describes the order in which the patterns are recognized. You can set the figure files and set and edit the two textual files in the edit mode. There is two buttons and a text field for each of the three files. The directory path of the file is written in the text field. You can write a directory path yourself or browse the directory structure by pressing the browse button. When the directory is choosed, you can load the file in the text area (or load the figure in the window) by pressing the load button. The textual files can be edited in the text areas. If you load and save an external file as e.g. the prolog file of the pattern, the external file is copied in the pattern directory. Note, that the text that is in the text areas are written to the prolog and index files. So you must load the text in the window if you want to e.g. load the prolog code from an external file - it is not enough to just write the directory path in the text area.

6.2.1 Prolog file

The heart of the pattern library is a collection of Prolog files (one for each pattern). These files are named **constraints.pro** and they are located in the directories of their respective patterns. Each file contains the facts that define the particular pattern. They can be edited while in the Edit mode of the Pattern library window or with any text editor.

The facts used in the Prolog-files are as follows[1]:

Fact	**Description**
class(X)	X is a class
method(X)	X is a method
attribute(X)	X is an attribute
public(X)	X is public
protected(X)	X is protected
private(X)	X is private
abstract(X)	X is abstract
final(X)	X cannot be redefined
static(X)	X is a class member
has(C,X)	class C has a member X
extends(X,Y)	Class X is a subclass of Y
typeof(A,C)	Attribute A is of type (class) C
contains(A,C)	Attribute A contains reference(s) to class C
implements(X,Y)	Method X implements the (virtual) method Y
returns(M,C)	Method M returns (a reference to the) class C
argument(M,C)	Method M gets (a reference to the) class C as a parameter
reads(M,A)	Method M reads the attribute A
calls(X,Y)	Method X calls method Y
creates(M,C)	Method M creates an instance of class C
same(X,Y)	Components X and Y are the same
binded(X)	Role is binded to a particular component X

Negation is represented with the !-symbol. It can be used with facts: !public("X"). or components: !"X".

Example: The Prolog description of the *Singleton* pattern is

```
class("Singleton").
```

[1] Note that the role names must be in quotes.

```
attribute("Singleton.instance").
has("Singleton","Singleton.instance").
typeof("Singleton.instance","Singleton").
static("Singleton.instance").
```

6.2.2 Index file

The index file, which is located at the root directory of the pattern library, is named **common.txt**. It is used to locate the Prolog-files of the patterns and to guide the search process itself.

The index file contains an entry for each pattern in the library. Each entry has four main parts:

- Pattern name: The line must begin with the character #, followed by the name of the pattern.
- Constraint file: This line must begin with *file:*, followed by the path name for the constraint file (relative to the root directory of the library).
- Pattern requirements: The names of those patterns that are prerequisites to this one. This line begins with a *requires:* keyword.
- Generated facts: The descriptions of those facts that are produced as a byproduct while searching for this pattern. These will be used when more patterns are searched. The keyword here is *generates:*, followed by the description of the generated facts (one per line). The names of the roles as well as that of the fact must be enclosed in quotes. For example, the line: "myfact"("Role")., would generate the fact: myfact(<binding>)., where <binding> is the binding *Role* had when this directive was triggered.

Example: The *Indirect Write* entry in the index file

```
#Indirect Write
file: Meta_patterns/Indirect_Write/constraints.pro
requires: Derived Attribute
generates:
        "writes"("First","Attribute").
        "indirectwrite"("First","Attribute").
```

7 Help window

The help window shows this user manual you are reading in the moment and it can be opened from the three basic windows of the tool (main, metric, pattern). It works like a HTML frame. On the left side of the window there is a list of the main chapters of the manual. The subchapters are not shown in the list but they are found in the text of the main chapters. The selected chapter is shown on the left, with the text on the top and the figures on the bottom. The text and the figures are shown in separate, scrollable panels. You can drag the borders between the chapter list and between the text and figure panels according to your taste. Note, that the figure numbering is not global but specific to the chapters.

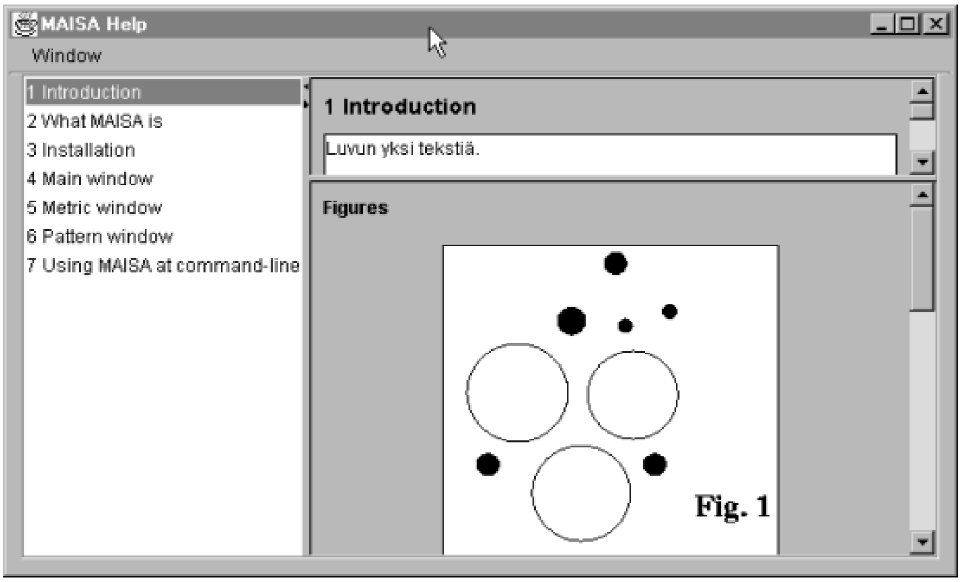

Figure 10: The help window.

The help window is shown in Figure 10. There is only one menu in the menubar: Window. The window menu contains items to open the basic windows:

Measures and metrics opens the metric window

Pattern library opens the pattern window

Main window opens the main window

Close closes the help window

8 Using MAISA in command-line

At its present stage, the command-line interface of the MAISA tool is rather limited as it does not include any patterns or selection of measures. On the other hand, only the MAISA.datastruct package is needed to use it, so the program is much smaller than the full version.

Command-line options and the possibility to use pattern recognition will be added in the future. However, any graphical features, such as the visualization of patterns, cannot be used. This feature serves best as a quick testing tool or to provide an overview of the architecture.

The program is started with the command[2]:

```
java MAISA.datastruct.MaisaStructure
```

The program asks the user for the project that is to be measured. For now, all simple (i.e. not pattern based) metrics are calculated for each diagram in the selected project. The result is written (into the project directory in question) to the file *result.txt*. See [Gus2000] for an example of the result file.

Next the program asks if the user wishes to continue. By answering *Yes*, the user can proceed to a new measurement, with a *No*, the user will exit the program.

8.1 Pattern library directory

The pattern library can be updated in the pattern window of the MAISA tool or with the common file handling tools of the operating system (the former way is more user friendly to novice users, while the latter is quicker and more efficient). Patterns or pattern groups can be added, edited and removed from the library. These operations are intended for administrative use. A pattern is identified with pattern name and group name. There can be patterns with the same name belonging to different pattern groups, for example versions of the same pattern.

The physical pattern library is a directory, located in `maisa/pl`. The root directory contains one subdirectory for each pattern group. The group name is the name of the subdirectory, with underline characters replaced with spaces. The group directories contain one subdirectory for each pattern belonging to the group. The name of the pattern directory is the name of the pattern, with spaces replaced with underline characters. The group names and the pattern names do not contain any underline charecters and the directory names do not contain any spaces, in order that the directory structure works in the environments where file names with spaces are not supported. The root directory also contains one prolog file which is common to all the patterns and which is called **common.txt**. It defines the order in which the patterns are recognized, as explained in the previous chapter. The user can edit the file with the MAISA tool but it is not generated by the tool. Each he pattern directory contains five files used by the MAISA tool: prolog, description, statistics, metadata and figure file (naturally it can also

[2] Naturally, the classpath needs to be correctly set.

contain other files connected to the pattern, for example sample source code and additional figures and diagrams).

Figure 11 shows the structure of the pattern library directory. Note that there is one group directory also for the metapatterns [Pre1995], because an instance of a metapattern can be construed as more than one design patterns presented by Gamma et. al. [Gam1995], depending on the semantics. Metapatterns themselves can also be recognized, without mapping them into design patterns. The following list explains how the pattern files are used.

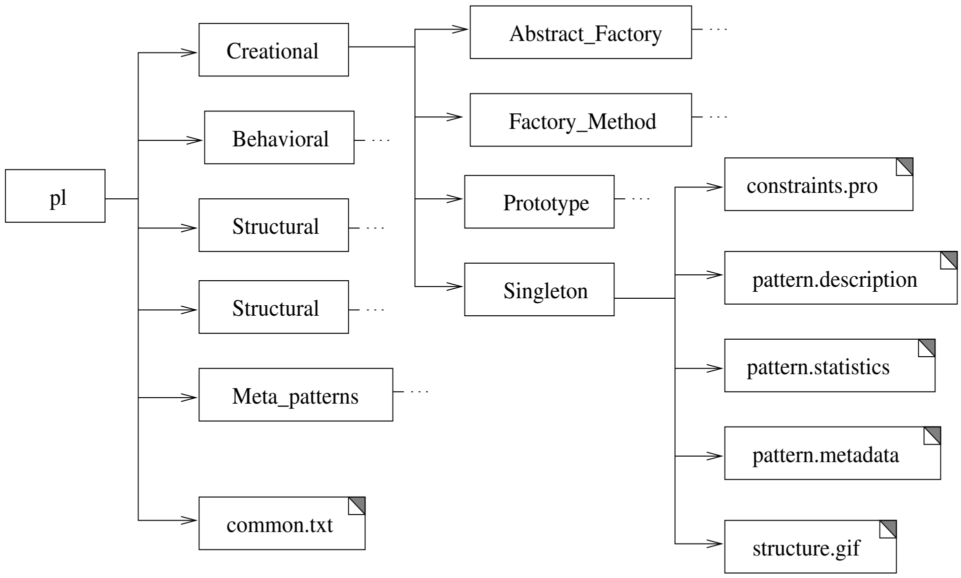

Figure 11: Structure of the pattern library directory.

constraints.pro The Prolog file that contains the structural requirements (or definition) of the pattern. The pattern recognition is based on these files. The file is not generated by the MAISA tool but the user can edit or set it with the tool.

pattern.description Text file that contains the same information of the pattern as the Design Patterns book [Gam1995] (see pages 29-31). The information of the file is for the human user, not the algorithms. The file is not used in the pattern recognition but only by the user interface of the MAISA tool. The file is generated by the MAISA tool on the basis of the data fed by the user.

pattern.statistics Text file that contains statistical information of the pattern: how many times it has been recognized, how many of these have been confirmed and how many of these have been false recognitions. The statistics file is automatically generated and updated by the MAISA tool and the user can reset the statistical information with the tool.

pattern.metadata Text file that contains metadata of the pattern that is used by pattern based estimation algorithms: LOC per pattern instance, LOC per client of the pattern and the performance factor of the pattern. The metadata file is generated by the MAISA tool on the basis of the data fed by the user.

structure.gif Figure file that describes the structure of the pattern. Optional. The file can not be edited or generated with the MAISA tool, but it can be set with the tool.

The formats of the text files are simple. The statistics and metadata files contain name-value pairs, one pair in a row. The name and the value are separated by a colon. The values are typically numeric, while the names are special keywords. The keyword sets can be easily extended in the future if needed. The keywords of the statistics file are:

- **WRONG** false recognitions
- **CONFIRMED** confirmed recognitions
- **TOTAL** all recognitions

A sample statistics file:

```
WRONG: 5
CONFIRMED: 12
TOTAL: 50
```

Metadata files are not yet in use, because the information contained in it will be used by the pattern based estimation algorithms, which are not yet in use in the MAISA tool. The exact types of the values are not yet fixed. The keywords of the metadata files are (the set can be extended in the future):

- **PERFORMANCE** performance factor
- **LOCPI** LOC per a pattern instance
- **LOCPC** LOC per a pattern client

The description file is divided into paragraphs, according to the documentation technique presented in the Design Patterns book [Gam1995]. The paragraphs are separated by an empty row and they can be in any order. The first line of the paragraph contains the mere paragraph name. The paragraphs are identified by their names, which are keywords. The structure of the paragraph depends on its name, as described in the following:

- **INTENT** What the pattern does. Any text except empty rows (the paragraph separator).
- **ALSO KNOWN AS** A comma-separated list of other names of the pattern.

- **MOTIVATION** What design problem the pattern solves. Any text except empty rows (the paragraph separator).
- **APPLICABILITY** A list of points where the pattern can be used. The first row of each list item begins with an asterisk.
- **PARTICIPANTS** A recursive list of the roles of the pattern. Each list item presents a role and contains a list of the tasks belonging to the role. The first row of a role item begins with an asterisk that is followed by the role name. The following rows of the role item begin with ">" that is followed by a task description.
- **COLLABORATIONS** A list of points where the elements of the pattern co-operate. The first row of each list item begins with an asterisk.
- **CONSEQUENCES** A list of consequences of using the pattern. The first row of each list item begins with an asterisk.
- **IMPLEMENTATION** Implementation specific tips on using the pattern. Any text except empty rows (the paragraph separator).
- **SAMPLE CODE** Sample code fragments, with possibly some ordinary text. Any text except empty rows (the paragraph separator).
- **KNOWN USES** Examples of the pattern found in real systems. Any text except empty rows (the paragraph separator).
- **RELATED PATTERNS** Pattern's relations to other patterns. Which other patterns should be used with this one. Any text except empty rows (the paragraph separator).

A sample description file:

```
INTENT
The pattern is used to...

ALSO KNOWN AS
Name 1, Name 2, Name 3

MOTIVATION
The pattern is useful, because...

APPLICABILITY
* Point 1
* Point 2
* Point 3

PARTICIPANTS
* Role A
> task 1
> task 2
* Role B
> task 3
> task 4
> task 5

COLLABORATIONS
* Point A
* Point B

CONSEQUENCES
* Point X
* Point Y
* Point Z

IMPLEMENTATION
The pattern can be implemented in this way...

SAMPLE CODE
Class MyClass{
...
}

KNOWN USES
The pattern has been used in...

RELATED PATTERNS
The pattern can be used with pattern X and...
```

8.2 Help directory

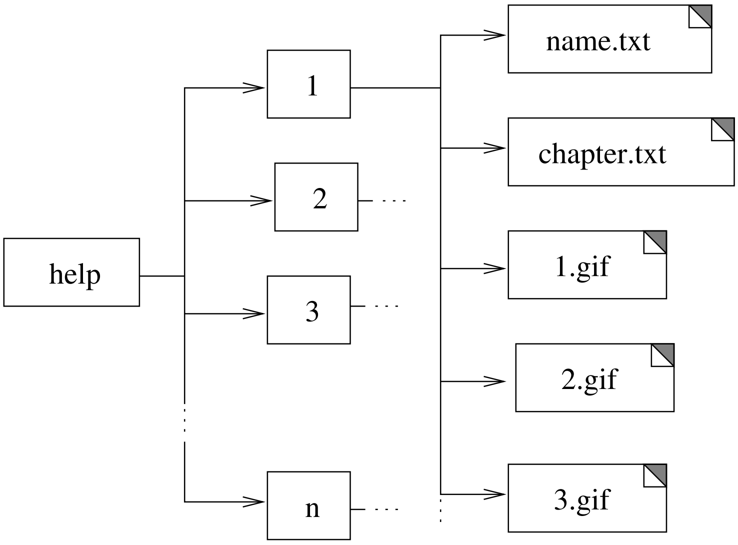

Figure 12: Structure of the pattern library directory.

As the pattern library, also the online manual is a directory. It is located in `maisa/help` and it consists of directories corresponding the chapters of the manual. The name of a chapter directory is the number of the chapter. Figure 12 shows the structure of the help directory. As shown in the figure, a chapter directoty contains the following files:

name.txt Contains the name of the chapter – only one row.

chapter.txt The chapter itself, or the text of the chapter – plain text.

x.gif The figure files, where x is an integer, presenting the figure number.

As in the graphical user interface, the numbering of the figures is specific to the chapter, not to the whole manual. The figure numbering begins from 1. Also the names of the chapter directories are integers. There is no subdirectories for the subchapters.

A Current patterns in the library

The search for patterns in MAISA is hierarchical. Each pattern may have one more prerequisite patterns and for each pattern to be searched, these prerequisite patterns are searched first. As the metapatterns given below are very general (and common), they take more time to find than the more specific patterns. Because the more specific patterns are defined using these metapatterns, searching for a small number of specific patterns can still lead to an extensive search. More detailed description of the following patterns can be found in [Gam1995, Pre1995].

Auxiliary structures:

These pattern-like elements are used to indicate some key structures found in object-oriented programs. Among such structures are inherited attributes and class references (i.e. associations). While they are not design patterns, they are treated in a similar fashion during the search. As the actual patterns are based on these structures, they are calculated in most cases even if they are not explicitly selected.

Name	**Description**	**Requirements**
Derived Attribute	Inherited attributes.	-
Indirect Create	Indirect creation of instances	-
Indirect Read	Indirect read-access of an attribute.	Derived Attribute
Indirect Write	Indirect write-access to an attribute.	Derived Attribute
Single Reference	A class has a reference to (an instance of) another class.	Derived Attribute
Container	A class that contains a collection of objects.	Iterator
Extended Container	The subclasses of a container are also containers.	Container
Container Attribute	Attributes whose type is a container.	Extended Container
Multiple Reference	A class has a reference to (multiple instances of) another class.	Container Attribute

Metapatterns:

Several of the Gang-of-Four's pattern are equivalent to these patterns. The *Template Method*, for example, matches *Unification*. Note that the *1:1 Connection* pattern consists of three separate cases.

Pattern	**Description**	**Requirements**
1:1 Connection Reference	An *1:1 Connection*, where the association exists through a reference attribute.	Indirect Read, Single Reference
1:1 Connection Argument	An *1:1 Connection*, where the association exists through a method parameter.	1:N Connection
1:1 Connection AskRef	An *1:1 Connection*, where the association exists through the return value of a method.	1:N Connection
1:N Connection	As 1:1 Connection Reference, but there can be references to several instances.	1:1 Connection Reference, Multiple Reference, Indirect Read
Unification	A method calls a virtual method (of same class). The virtual method gets defined in subclasses.	-
1:1 Recursive Unification	A request is passed along a chain of instances (of same class) until the request is captured.	Indirect Read, Single Reference
1:N Recursive Unification	As above, but the request can be passed on to several objects	Indirect Read, 1:1 Recursive Unification, Multiple Reference
1:1 Recursive Connection	An object has a reference to its own superclass.	Indirect Read
1:N Recursive Connection	As above, but there can be several references.	1:1 Recursive Connection, Container Attribute

Gang-of-Four's patterns:

An asterisk (*) after the name of the pattern indicates that the pattern is currently not supported by the MAISA tool. There are two reasons for this. The pattern may be only a variation of one of the metapatterns. On the other hand, the pattern may not be easily expressed using purely syntactic rules.

Creational patterns:

Pattern	**Description**	**Requirements**
Abstract Factory	An interface for creating related objects without specifying the concrete class.	Factory Method
Builder	Separates the construction process of an object from the representation.	Indirect Create, Indirect Read, Indirect Write
Factory Method	An interface for creating an object without knowing its concrete class.	Indirect Create
Prototype	New instances are created by copying a prototype.	Indirect Create
Singleton	Ensures that only a single instance of a class exists at a time.	-

Structural patterns:

Pattern	**Description**	**Requirements**
Class Adapter	The interface of a class is converted into another via multiple inheritance.	-
Object Adapter	The interface of a class is converted into another via object composition.	Indirect Read, Single Reference
Bridge*	Decouples an abstraction from its implementation.	
Composite*	Hides part-whole hierarchies into composite objects. Individual objects and compositions can then be treated uniformly.	
Decorator*	Extends the functionality of an object dynamically.	
Facade*	A simple interface for a complex (sub)system.	
Flyweight	Supports a number of fine-grained objects efficiently by sharing (and thereby reducing the number of objects actually needed.	Indirect Create, Indirect Read, Indirect Write
Proxy	A proxy acts as a surrogate for another object.	Indirect Read, Single Reference, Virtual Proxy
Virtual Proxy	The creation of expensive objects is deferred until needed (a special case of *Proxy*).	Indirect Create, Indirect Read, Indirect Write, Single Reference Factory Method

Behavioral patterns:

Pattern	**Description**	**Requirements**
Chain of Responsibility*	Requests are passes along a chain of objects until an object handles it.	
Command*	Requests are encapsulated as objects.	
Interpreter*	The grammar of a programming language is represented as a class structure. This representation can then be used to interpret the language	
Iterator	Access to an aggregate object is separated from the underlying representation.	Indirect Create, Indirect Read, Indirect Write, Single Reference
Mediator	Loose coupling is promoted by centralizing requests to go via a mediator object.	Single Reference, Multiple Reference
Memento	Captures the internal state of an object without violating encapsulation.	Indirect Create Indirect Read, Indirect Write
Observer	Any changes in the state of an object are automatically relayed to its dependents.	1:1 Connection Reference, 1:N Connection, Indirect Create, Indirect Read, Indirect Write
State*	The behavior of an object changes according to changes in its internal state.	
Strategy*	A family of algorithms is defined as a set of interchangeable classes.	
Template Method*	The constant part of an operation is separated from the variable parts. The variable parts can then be implemented as needed by subclassing.	
Visitor	New operations can be defined without the need to change the classes those operations affect.	-

References

[Gam1995] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley 1995.

[Gus2000] Juha Gustafsson: *Metrics calculation in MAISA*, University of Helsinki, Department of Computer Science, August 2000.

[Mis1997] Pauli Misikangas: *Suunnittelumallien automaattinen tunnistaminen olio-ohjelmista*, Master's Thesis, University of Helsinki, Department of Computer Science, C-1998-1, 1997.

[NeG2000] Lilli Nenonen, Juha Gustafsson: *Pattern recognition in the MAISA tool*, University of Helsinki, Department of Computer Science, August 2000.

[Pre1995] Wolfgang Pree: *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.