MCDA Toolkit

Developer Guide

Last modified: 10. October 2013

Version: 0.1

Outline

MCDA (Multi Criteria Decision Analysis) is a methodology to support decision making with the benefit of, simply put, being capable to compare apples and oranges. Starting from a goal with several alternatives users can interactively weight the importance of influencing criteria. This process is often used when decision makers try to find a stable agreement that can be accepted by all members of the group.

This guide covers the internal structure of the MCDA Toolkit originally developed by KIT [KIT].

Table of Contents

1. General structure

The MCDA Toolkit is separated in two parts: the computing engine (the CORE) and the user interface (the GUI). The intention behind this is to provide an engine that can be operated in batch mode and to provide some degree of freedom to custom frontends.

The CORE contains all necessary data structures, computation methods, and IO functionality to process MCDA problems. The CORE is presented in Chapter 2 “The CORE of the MCDA Toolkit“.

The GUI provides means to interactively change values and weights while at the same time providing means to visualize and analyse the results. The GUI is presented in Chapter 3 “The GUI of the MCDA Toolkit“.

1.1 Requirements

The MCDA tool is completely written in Java. To contribute to the development you need a Java Development Kit (JDK). You may use any computer and operating system that support Java 7 or up. The application itself neither requires high computation power nor a large amount of memory during runtime.

Several compatible implementations of the JDK are available on the internet. The most common JDK is provided by Oracle and can be downloaded from the according website [JDK]. The installation of the JDK is straightforward. Simply follow the instructions.

Both CORE and GUI are developed using Netbeans IDE (Integrated Development Environment) [Netbeans]. Several other IDEs for Java are available, however they are not compatible in general. Therefore the usage of Netbeans is mandatory.

The project is under version control using Mercurial as backend [Mercurial]. Several frontend applications for Mercurial are available. We recommend TortoiseHg [TortoiseHg] however you may use a different one especially if you are used to it already. On a site note Netbeans already fully supports Mercurial by default.

1.2 Sources

The sources of the CORE and GUI are organised in two different repositories. At this time each can be accessed through the internet. Retrieving the sources requires you to copy them to your local environment. The process of retrieving for the first time is called cloning. During cloning the complete history and all prior versions of the software are copied to the local environment. Following the instruction in Appendix C: “Managing Sources“ you end with two new directories named `MCDA-Core` and `MCDA-GUI`. We recommend to group them into single super directory with name eg. `MCDA`.

The CORE contains the following sub directories:

Directory	Description
`lib`	libraries required to compile and run the application
`nbproject`	Netbeans project information
`src`	the actual Java sources of the application

`.hg`	Mercurial information on the repository

The GUI contains the following sub directories:

Directory	Description
`config`	configuration files of the application
`customization`	customised look and feel like color definitions, icons, etc.
`docs`	inline help and documenting text files
`examples`	predefined MCDA problems to provide an easy start
`lib`	libraries required to compile and run the application
`nbproject`	Netbeans project information
`src`	the actual Java sources of the application
`.hg`	Mercurial information on the repository

After cloning the two repositories the projects can be opened with Netbeans. As the GUI is depending on the CORE some paths have to be adapted in the project configuration. However Netbeans will prompt you for the necessary data.

The sources are updated from time to time. To receive modifications you need to request them from the server. The process of requesting modifications from the server is called pulling. During pulling conflicts may occur which then have to be resolved. Resolving conflicts is beyond the scope of this document and should be handled by experts. It is desired that conflicts do not happen in the first place. Once pulling is complete you most likely would want to upgrade to the most recent version as the upgrade is not triggered by default. The process of upgrading or rather moving between versions in the repository is called updating.

Once you changed the sources and have tested your modification you should fix it by creating a new version. The process of fixing a versions is called committing. It causes the local repository to create a new version. At some time you may want to distribute the accumulated changes back to the original repository. The process of distributing back is called pushing. While cloning and pulling are basically allowed without restrictions, pushing requires a user authentication. The

2. The CORE of the MCDA Toolkit

According to the structure of MCDA and the best practise of Java developing the sources are distinguished in groups (the packages) and files (the classes). The topics of the CORE packages are basics, normalisation, aggregation, constants, and AHP [AHP]. This reflects in accordingly named packages.

The CORE on one hand provides the data structures to contain MCDA problems and on the other hand process them and provide the solution as a result set. In a simplified approach MCDA problems consist of meta data most likely in the form of text and actual values, i.e. a problem coordinator, the alternatives, and the criteria forming a matrix that has to be processed.

2.1 General Architecture of the CORE

The MCDA problem is encapsulated in a goal, which contains meta data, a list of alternatives and a list of criteria. For every alternative each criteria provides some data which leads to a matrix like structure. This matrix is implemented using hash maps based on the unique identifiers of criteria and alternatives. For the time being the data is limited to floating point values. Moreover the goal contains a tree like hierarchy to organise criteria into groups. Figure 1 gives a general overview on the architecture. Figure 2 presents a simplified UML diagram of the main components.

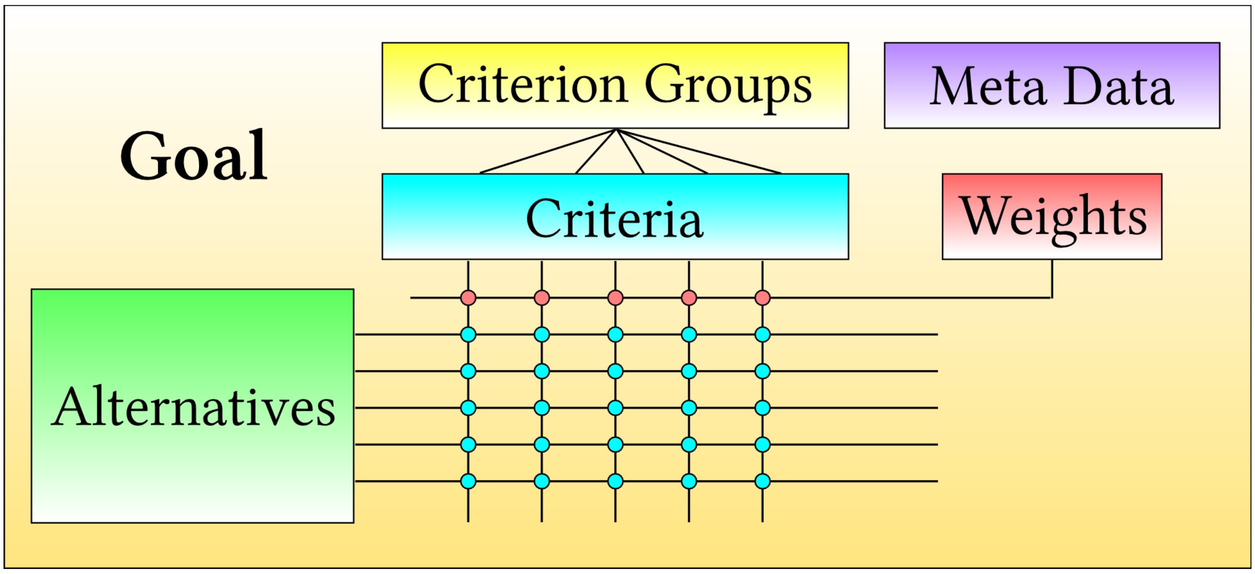


Figure 1: Overview on the general structure of the CORE.

The goal also provides means to resolve the MCDA problem and to retrieve the solution. Accessing the results triggers the computation process on demand: first the criteria are normalised according to their normalisation function. Then for each alternative the corresponding criteria values are aggregated according to an aggregation function taking into account the importance respectively weight of each criterion. Several normalisation and aggregation functions are available. The weights can be either set manually or will be generated using the AHP.

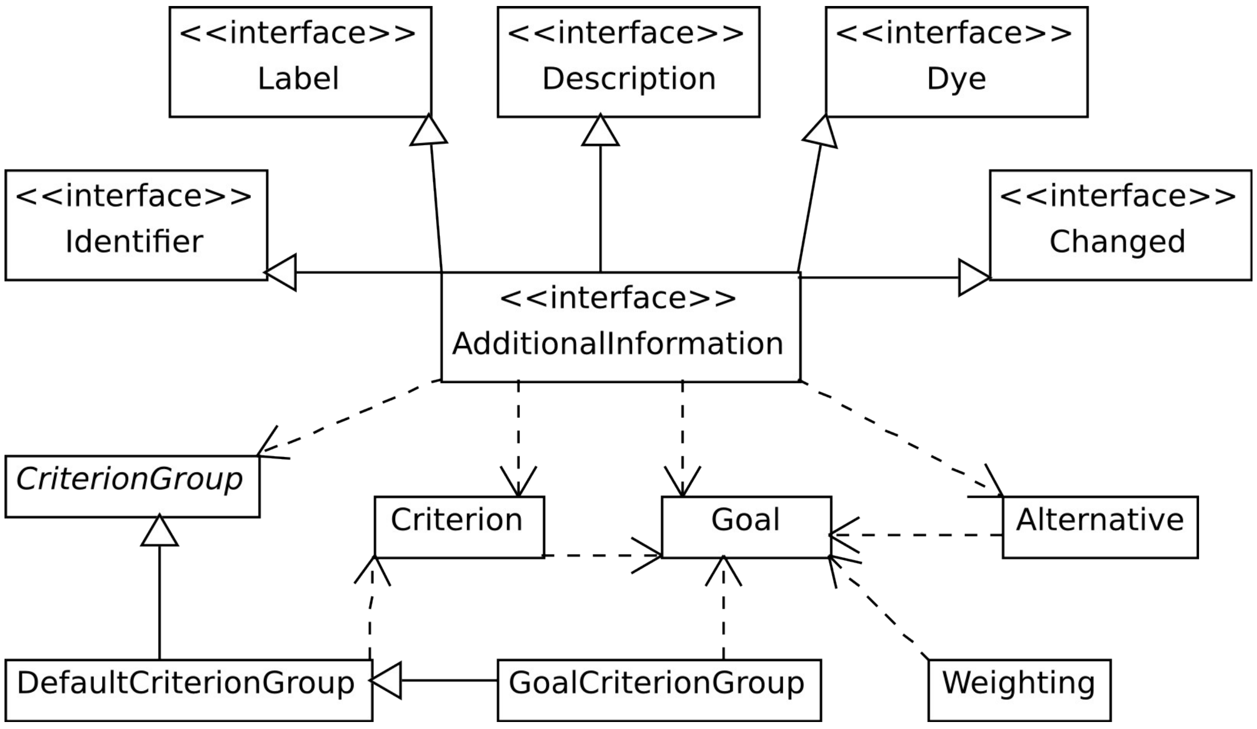


Figure 2: Simplified UML diagram of the main CORE components.

Reoccurring data structures and resulting interfaces

Some identical data structures are used in several classes and their instances. To simplify the handling corresponding interfaces have been created for each of this data structures.

Interface	Description
`Identifier`	Manage an unique identifier for hash maps and correlations in XML documents
`Label`	Manage multi-lingual labels of an instance not necessarily unique
`Description`	Manage multi-lingual descriptions of an instance
`Dye`	Manage the colour of an instance
`Changed`	Managing a flag that indicates changes in the data values of the instance
`AdditionalInformation`	Combining the interfaces `Identifier`, `Label`, `Description`, `Dye`, and `Changed`

Table 1: Overview on meta data interfaces and their purpose.

The classes `Goal`, `Alternative`, `Criterion`, and `CriterionGroup` implement these interfaces through the aggregating interface `AdditionalInformation`.

Managing Criteria

Criteria are stored as linked list and hash map in the class `Goal`. The hash map is based on the unique identifiers of the criteria. Each criterion contains two maps linking alternatives via the alternative unique identifiers to absolute values and normalised values. At the time of writing the values are plain floating point values. However the vision is to replace these simple values with multi-dimensional functions by the nature of fuzzy, probabilistic, look-up table, etc. whereas dimensions can be eg. time, enumeration, etc. The general structure of the CORE is already prepared to deal with such data structures. Nevertheless problems like normalising are not addressed yet.

A criterion also contains a link to the group it belongs to and furthermore a normalisation entity. Normalisation is explained in section 2.2 Criteria Normalisation.

Managing Alternatives

In the current implementation the data structures for alternatives simply cover meta data, which is implementation of the `AdditionalInformation` interface. The actual mapping of values of criteria to alternatives is handled by the criteria implementation.

2.2 Criteria Normalisation

Criteria reflect different measurements for alternatives, very often not quantitative but of qualitative nature. For comparison respectively aggregation the criteria need to be normalised. Many different methods are implemented their structure based on the common interface `AbstractNormalization`. Figure 3 provides an overview on the implemented normalisation methods and their relation.

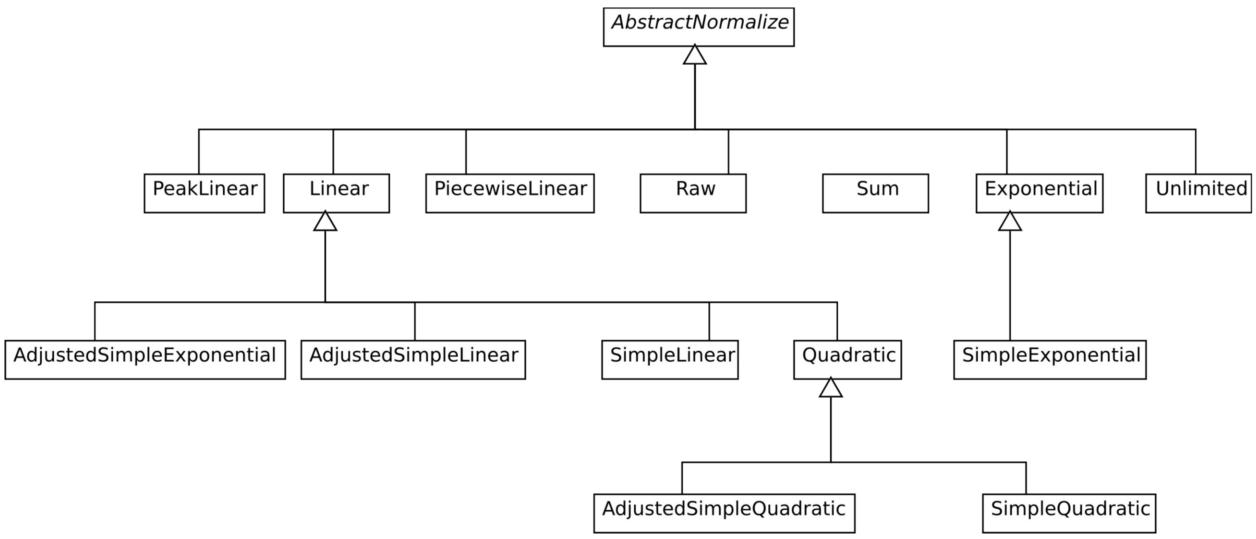


Figure 3: Class hierarchy of the normalisation methods.

Through the common interface all normalisation methods provide means to either normalise all values of a criterion or to normalise an external value in respect to the values of a criterion. Each criterion contains its own normaliser. Normalisation is triggered automatically during the resolving process if changes have been registered.

Normalisation parameters

Normalisation methods can have several parameters, which frequently are shared. Therefore parameters are defined as interface definitions which the normalisation methods implement according to their needs. Table gives an overview on the defined parameters.

Interface	Description
`DomainBounds`	Lower and upper bounds of the normalisation domain range
`Invertible`	
`Gain`	
`Speed`	
`Turn`	
`Curvature`	

2.3 Aggregation (Pending)

2.4 Weighting (Pending)

Manual Weighting (Pending)

AHP Weighting (Pending)

2.5 Criteria Groups (Pending)

2.6 Import and Export (Pending)

3. The GUI of the MCDA Toolkit

Wrapper Frame,Applet, Wizard, InternalFrames (update), General Appearance Buttons Functionality, Load/Save, Recent, Localization (updateLanguage), Customization,

3.1 Adding Frames

Frames are used to present MCDA data in a specific way eg. to graphically display and change the weights of criteria, edit the criteria values, etc. All frames are derived from the super class Template. By default frames provide a button bar that contains basic functions like open help, close frame, etc. The following sections introduce a step by step instructions on how to implement a new frame.

Initialisation

As an example we create a frame that displays the names of all alternatives. The class of the frame will be named `ShowAlternatives`. In Appendix D: Code of example frame ShowAlternatives the important parts of the code are presented.

Create the file `ShowAlternatives.java` and extend it from the class `Template`.

Adding Discard and Accept (Pending)

Adding Options (Pending)

Adding Help

All help texts are expected to be HTML encoded and to be located under the directory `MCDA-GUI/docs/help`. Files directly located in this directory are used as backup if language specific files are not available. Language specific texts are located in subdirectories with name of the two character ISO code according to the language, eg. help in German language is located under `MCDA-GUI/docs/help/de`. The files of a specific topic for different languages all have the same file name but are located in different directories.

By default the help button triggers the display of a localised help text. If not customised, the suffix `.html` is appended to the class name of the frame to search for the according file. To provide a help text for the upper example create the HTML file `MCDA-GUI/docs/help/ShowAlternatives.html` and fill it with appropriate content. To provide a German translation create the HTML file `MCDA-GUI/docs/help/de/ShowAlternatives.html` and fill it with German content, respectively. In general the files for backup and English language are the same.

If for any reason you want a different file name instead of the class based name override the method `getHelpText()` from the `Template` class to provide the desired name.

Logging and Debugging

The Template class provides a logging entity named `log` to simplify logging for debugging purposes. To output debugging information use `log.debug("your text")`. To address different tiers of severity use `log.debug`, `log.info`, `log.warn`, `log.error`, and `log.fatal`. Appendix A: How-To, Logging and Debugging explains how the logging process is controlled in general.

Appendix A: How-To

Logging and Debugging

For logging purposes the Apache log4j libraries are used [LOG4J]. The logging process is configured through the file `MCDA-GUI/config/log4j.xml`. Several output destinations are predefined and can be activated. Also the different log levels can be set here.

Appendix B: File Format

MCDA project data

MCDA result

MCDA special import and export

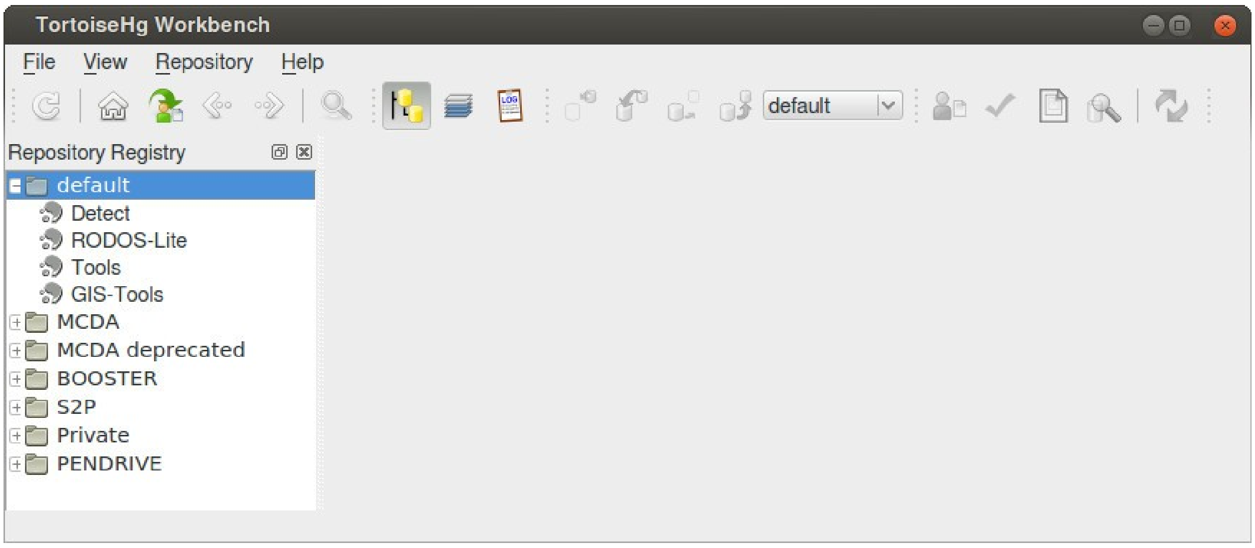
Appendix C: Managing Sources

The sources are available as Mercurial repository through the internet. In the following it is shown how to manage the sources using TortoiseHg for Linux.

Cloning

1. Create a location to clone the sources into. We recommend the path `projects/MCDA`.

2. Start TortoiseHg. The image below shows an already non-empty repository registry.

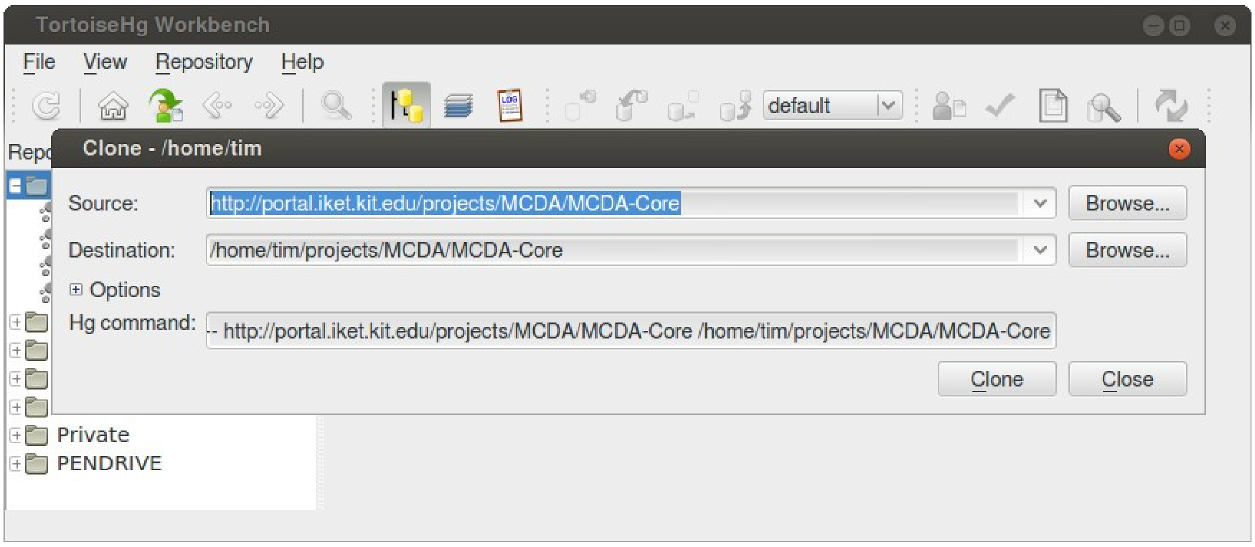


3. Clone the sources of the CORE. In the menu *File* select the item *Clone Repository*.

4. As source enter the following URL:
`http://portal.iket.kit.edu/projects/MCDA/MCDA-Core`

5. As destination enter your local location similar to:
`/home/tim/projects/MCDA/MCDA-Core`

6. The image below shows the screen after entering the values. Click *Clone*.



Source: http://portal.iket.kit.edu/projects/MCDA/MCDA-Core

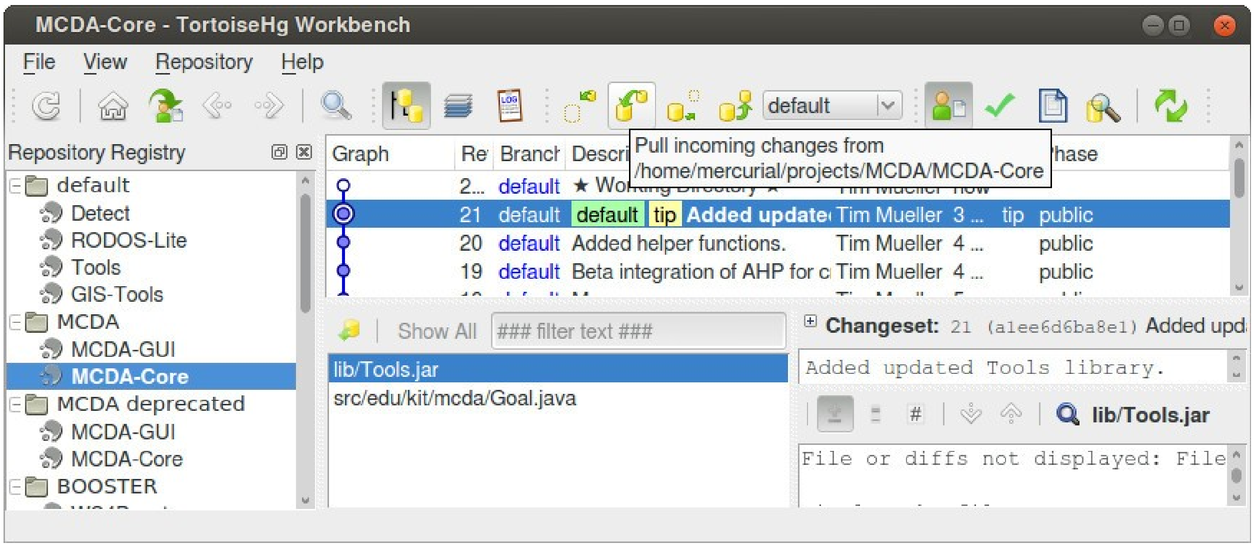
Destination: /home/tim/projects/MCDA/MCDA-Core

Hg command: -- http://portal.iket.kit.edu/projects/MCDA/MCDA-Core /home/tim/projects/MCDA/MCDA-Core

7. Clone the sources of the GUI the same way. As source enter the following URL:
`http://portal.iket.kit.edu/projects/MCDA/MCDA-GUI`

Pulling

1. The cloned repository MCDA-Core is displayed amongst the known repositories on the left side. Double click on the entry to open the repository. The image below shows the state after opening an repository.



Pull incoming changes from
/home/mercurial/projects/MCDA/MCDA-Core

Graph	Re	Branch	Descri		hase
	2...	default	★ Wo		
	21	default	default tip Added update	Tim Mueller 3 ... tip	public
	20	default	Added helper functions.	Tim Mueller 4 ...	public
	19	default	Beta integration of AHP for c	Tim Mueller 4 ...	public

lib/Tools.jar
src/edu/kit/mcda/Goal.java

Changeset: 21 (a1ee6d6ba8e1) Added upd

```
Added updated Tools library.
```

```
File or diffs not displayed: File
```

2. Either use the menu entry *Repository* → *Synchronize* → *Pull* or the highlighted button in the upper image to trigger the pull.

Updating

1. After pulling select the version in the tree view you want to move to (most likely the tip of the tree). Right click on it to open the popup menu, then click on *Update* to trigger the upgrade.

Commiting

1. Click on the top item of the tree view named working directory.

2. In the file view select all files you that want to commit (most likely all).

3. In the comment box add a comment describing the modification.

4. Click on *Commit*.

Pushing

1. Similar to the pulling process open the repository you want to distribute back to the original repository.

2. Either use the menu entry *Repository* → *Synchronize* → *Push* or the push button to trigger the distribution.

3. Pushing to the web server takes a lot of time. You will be prompted for a user name (`mcda`) and password (`mcda`).

Appendix D: Code of example frame ShowAlternatives

```java
package edu.kit.mcda.frames;

import edu.kit.mcda.Alternative;
import edu.kit.mcda.Goal;
import edu.kit.mcda.Main;
import edu.kit.mcda.constants.gui.Customization;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JCheckBox;
import javax.swing.JPanel;
import javax.swing.JTextPane;
import org.fzk.swing.CursorController;

public class ShowAlternatives
  extends Template {

////////////////////////////////////////////////////////////////////////////////
// constants definitions
////////////////////////////////////////////////////////////////////////////////
  /** Key for the display title option in the preferences. */
  private transient static final String PREFERENCES_KEY_DISPLAY_TITLE = "displayTitle";
  /** Default value for the display title option in the preferences. */
  private transient static final boolean PREFERENCES_VALUE_DISPLAY_TITLE = true;

////////////////////////////////////////////////////////////////////////////////
// field definitions
////////////////////////////////////////////////////////////////////////////////
  /** The text panel to display the report in. */
  private JTextPane tpText;
  /** Check box for generating title. */
  private JCheckBox cbTitle;
  /** Buffer to undo changes of {@link #cbTitle}. */
  private boolean bufferTitle;

////////////////////////////////////////////////////////////////////////////////
// constructor, main and initialization
////////////////////////////////////////////////////////////////////////////////
  public ShowAlternatives(Goal _goal, Main _main) {
    super(_goal, _main);
    preInit();
    init();
    postInit();
  }

  /**
   * Initializations before main initialization (of components).
   */
  private void preInit() {
  }

  /**
   * Main initializations (of components).
   */
  private void init() {
    initDefaults();
    initComponent();
  }

  /**
   * Initializations after main initialization (of components).
   */
  private void postInit() {
  }

  /**
   * Initializations of the default values
   */
  private void initDefaults() {
  }

  /**
   * Initializations of the components
   */
  private void initComponent() {
    setFrameIcon(Customization.REPORT_X16);
    setFrameTitle(getTranslator().getTranslation("general_analysis"));
    //
    initOptions();
    initReport();
  }

  private void initOptions() {
    final JPanel optionsContainer;
    //
    optionsContainer = new JPanel();
    optionsContainer.setBorder(BorderFactory.createEmptyBorder(10, 10, 5, 10));
    optionsContainer.setLayout(new BoxLayout(optionsContainer, BoxLayout.Y_AXIS));
    optionsContainer.setOpaque(false);
    //
    cbTitle = new JCheckBox(getTranslator().getTranslation("option_report_title"));
    cbTitle.setSelected(loadPreference(PREFERENCES_KEY_DISPLAY_TITLE, PREFERENCES_VALUE_DISPLAY_TITLE));
    cbTitle.addActionListener(new ActionListener() {
      @Override public void actionPerformed(ActionEvent e) {
        Runnable runnable = new Runnable() {
          public void run() {
            updateContent();
          }
        };
EventQueue.invokeLater(CursorController.wrapRunnable(getInstance().getRootPane(), runnable));
      }
    });
    optionsContainer.add(cbTitle);
    //
    setOptions(optionsContainer);
  }

  /**
   * Initialization of the report
   */
  private void initReport() {
    tpText = new JTextPane();
    tpText.setContentType("text/html");
    updateContent();
    //
    setContent(tpText);
  }

  /**
   * Initialization of the report text
   */
  private void updateContent() {
    StringBuilder builder;
    //
    builder = new StringBuilder();
    if (cbTitle.isSelected()) {
      builder.append("<h1>Alternatives</h1>");
    }
    for (Alternative tmpAlternative : getAlternatives()) {
      builder.append(tmpAlternative.getLabel(getTranslator().getLanguage()));
      builder.append("<br>");
    }
    tpText.setText(builder.toString());
  }

//////////////////////////////////////////////////////////////////////////////
// update methods
//////////////////////////////////////////////////////////////////////////////
  /**
   * Updates the language
   */
  @Override public void updateLanguage() {
    setFrameTitle(getTranslator().getTranslation("general_analysis"));
    updateContent();
  }

  @Override public void updateFrame(ArrayList<Changed> _flags) {
    if (_flags.contains(Changed.STRUCTURE)) {
      updateContent();
    }
  }

//////////////////////////////////////////////////////////////////////////////
// miscelaneous methods
//////////////////////////////////////////////////////////////////////////////
  @Override public void discardOptionsChanges() {
    cbTitle.setSelected(bufferTitle);
    updateContent();
  }
  @Override public void commitOptionsChanges() {
    bufferTitle = cbTitle.isSelected();
    storePreference(PREFERENCES_KEY_DISPLAY_TITLE, cbTitle.isSelected());
  }

}
```

Appendix E: Third-party Libraries and Licenses

- Oracle Java JDK:
  Oracle Binary Code License
- Alternative OpenJDK
  GNU GPL 2
- Java Help
  GNU GPL 2
- Apache Http Components
  Apache License
- JAXEN XML parsing
  Apache style License
- JDOM XML document model
  Apache style License
- JFreeChart charting
  GNU LGPL
- Log4j logging
  Apache License
- JScience
  free, copyright usage
- JCalendar date chooser
  GNU LGPL
- Netbeans Outline TreeTable
  GNU GPL 2 oder CDDL

Bibliography

[KIT] KIT, Karlsruhe Institute of Technology, http://www.kit.edu/english/

[JDK] Oracle, Java Development Kit (JDK), http://www.oracle.com/technetwork/java/javase/downloads/index.html

[Netbeans] Netbeans, Netbeans Integrated Development Environment, https://netbeans.org/

[Mercurial] Mercurial Community, Mercurial: Distributed Version Control System, http://mercurial.selenic.com/

[TortoiseHg] Steve Borho and Yuki Kodama, TortoiseHg: a frontend for Mercurial, http://tortoisehg.bitbucket.org/

[AHP] Wikipedia, Analytic hierarchy process, http://en.wikipedia.org/wiki/Analytic_hierarchy_process

[LOG4J] Apache Foundation, Log4j, http://logging.apache.org/log4j/1.2/