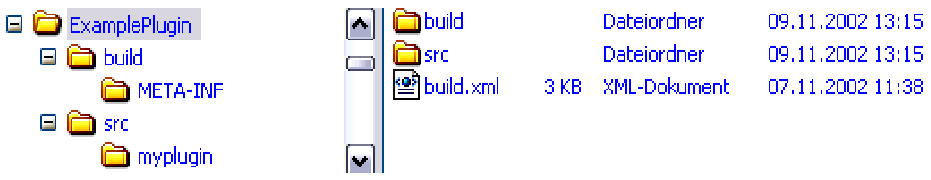
1. Writing a Plugin for fuzzyIDE:



```
⊟ ExamplePlugin
    ⊟ build
        META-INF
    ⊟ src
        myplugin
```

```
build                 Dateiordner     09.11.2002 13:15
src                   Dateiordner     09.11.2002 13:15
build.xml      3 KB   XML-Dokument    07.11.2002 11:38
```

- the easiest way to start is to use the structure given to you in the archive <exampleplugin.zip>
- the buildfile <build.xml> is for use with the AntFarm
- the <build/META-INF/manifest.mf> is important for the final plugin-jarfile

1.1 manifest.mf:

```
Manifest-Version: 1.2
Created-By: 1.3.0 (Sun Microsystems Inc.)
PluginMain-Class: myplugin.MyPlugin
```

- change the PluginMain-Class Value to the one you need (Mainclass of your plugin)
- in case your plugin needs any external libs use the “classpath” tag in this file.

1.2 build.xml:

- adjust this file for your purposes
- while building the final pluginjarfile there will be automatically put the manifest.mf into the jar-file.

1.3 getting started:

- always use the fuzzyIDE – API to get detailed information
- the <fuzzyide.jar> gives you all the classes you need
- use the <myplugin.java> file to have a framework to start with

- you have to implement several Interfaces to register your program as a plugin for the fuzzyIDE

1.3.1 The Interfaces to be implemented as needed:

1.3.1.1 Plugable:

- the MainInterface to register your program as a plugin

```
/**called from the PluginManager of the fuzzyIDE after construction of the object
 *@param  pluginID   every plugin gets an id for identification and authentification
 *@param  location   path to the plugin (must'nt be used)
 *@param  locale     represents a specific geographical, political, or cultural region
 *@param  applicationInterface  The ApplicationInterface*/
public void init(ApplicationInterface applicationInterface,
                 int pluginID,
                 File location,
                 Locale locale);

/** is called before the shutdown of the plugin (f.e. at the application-shutdown)*/
public void exit();
```

```
/**Gets the name of the Plugin */
public String getName();

/**@return    the pluginID which is given by the init-method*/
public int getID();

/** Gets the jMenuItem attribute of the Plugin, eventListener must be added by the
 * plugin
 *@return    The jMenuItem or null if this Plugin has no Menuentry*/
public JMenuItem getJMenuItem();

/**     a panel for the plugin, where the user can configurate the plugin
 *      events must be catched from eventListeners, added to the panel
 *@return    JPanel the configPanel or null if this Plugin has no Configpanel
*/
public JPanel getConfigPanel();
```

1.3.1.2 DataModelAccessable

- used to get access to the DataModel (XML-Model)

```
/*called when a datamodel is created gives the plugin the reference to the datamodel
 *inside this method the plugin should register his observation*/
 public void dataModelCreated(FuzzyModel fm);

 /* called when a datamodel is closed
  * after this method there is no dataModel available */
 public void dataModelClosed();
```

1.3.1.3 ModelObserver:

- used to get informed about changes on the DataModel
- extends java.util.Observer

```
 /** automatically called, when the model has changed*/
public void update(Observable obs, Object obj);
```

1.3.1.4 Viewable:

- used to get access to the desktop (using, changing Internalframes)

```
/*called when a the view is created gives the plugin the reference to the view
 *inside this method the plugin should register his observation */
 public void viewCreated(ViewInterface v);
```

1.3.1.5 ActionListener:

- handle all the ActionEvents in your plugin (JMenu …)

1.3.2 The AbstractPlugin – Class:

- already implements part of Plugable and Viewable
- you have just to implement the following Methods **of Plugable**:
  - void exit()
  - String getName()
  - JMenuItem getJMenuItem()
  - JPanel getConfigPanel()
- if you need to, you can also override methods of AbstractPlugin

```
public abstract class AbstractPlugin implements Plugable, Viewable
{
        protected int pluginID;
        protected ApplicationInterface application;
        protected ViewInterface view;
        protected File file;
        protected Locale locale;

        //inherited from Plugable
        public void init(ApplicationInterface applicationInterface, int pluginID, File location, Locale locale)
        {
                this.application = applicationInterface;
                this.pluginID = pluginID;
                this.file = file;
                this.locale = locale;
                this.initComponents();
        }

        //inherited from Plugable
        public int getID()
        {
                return this.pluginID;
        }

        //inherited from Viewable
        /** called when a the view is created gives the plugin the reference to the view
        *  inside this method the plugin should register his observation
        */
        public void viewCreated(ViewInterface v)
        {
                this.view = v;
        }

        /* here you can initialise all your GUI or other Components of the Plugin */
        public abstract void initComponents();
}
```