Christian-Albrechts-Universität zu Kiel

Bachelor Project

Configurations and Automated Execution in the KIELER Execution Manager

cand. inform. Sören Hansen

February 9, 2010

Department of Computer Science
Real-Time and Embedded Systems Group

Prof. Dr. Reinhard von Hanxleden

Advised by:
Christian Motika

Contents

List of Figures

Abbreviations

API Application Programming Interface

GUI Graphical User Interface

IDE Integrated Development Environment

KIEL Kiel Integrated Environment for Layout

KIELER Kiel Integrated Environment for Layout Eclipse Rich Client

KIEM KIELER Execution Manager

KIEMAuto Automated Executions for the KIEM

KIEMConfig Configurations for the KIEM

UI User Interface

1 Introduction

1.1 Description of KIELER

- layouter, structure based editing
- synchcharts
- simulater
- execution manager

1.2 Description of the KIEM

Execution Manager is used in KIELER as a framework to plug-in DataComponents for various tasks. Examples are:

- Simulation Engines
- Model Visualizers
- Environment Visualizers
- Validators
- User Input Facilities
- Trace Recording Facilities

These DataComponents can be executed using a graphical user interface (GUI). The scheduling order can also be defined by this GUI as well as other settings like a step/tick duration and properties of DataComponents. For information about KIEM see - `http://rtsys.informatik.uni-kiel.de/trac/kieler/wiki/Projects/KIEM`

1.3 Outline of this Document

The first part of this document is about the implemention of the Configuration plugin for the KIEM. The second part is about the implementation of an Automated Execution plugin for the KIEM. Both parts will have the same structure described below. Each part starts with a detailed outline of the problems that are to be solved by this thesis. The next section will be about the technologies used to solve the

problem. After that there will be a section about the concepts of how to solve the problem and some design decisions that were made at a very early stage in the development. Sections 5 and 6 are about the actual implementation with section 5 outlining the changes that were made to the KIEM plug-in itself and section 6 describing the newly created Configurations for the KIEM (KIEMConfig) plug-in. The final section of each part will summarize the results of the thesis and outline a few projects that could be based on it.

Part I

Configurations in the KIEM

2 Problem Statement

The objective of this project is to improve the configurability of the KIEM as outlined in the Diploma-thesis by Christian Motika [1]:

> KIEM currently does not have a preference page to save additional settings like DataComponent timeouts. Also execution schedulings might be similar for a common diagram type.
>
> It may improve the usability further to allow the user to customize execution schedulings for specific diagram types. An interface for these kind of settings could be realized as an Eclipse preference page.

This will be explained in more detail in the following sections.

2.1 Configurations

Currently every property in the KIEM has a hard coded default value. There is a text box for setting the aimed step duration for the currently loaded execution file but that value is lost once a new execution is loaded. To solve this problem an extension to the KIEM should attempt to provide the following:

1. Find a way that execution files can store values like the aimed step duration and the timeout. This mechanism should be implemented in a way that ensures that old files can be upgraded and new files are still valid in instances of the KIEM that don't use the configuration plug-in.

2. Find a way to load the configurations into the different parts of the KIEM as soon as an execution file is loaded from the file system.

3. Ensure that the user can edit all properties and maybe even create his own custom properties. This should be implemented in a way that doesn't clutter up the current user interface too much.

2.1.1 Default Configuration

The different properties stored in each execution file might sometimes not suit the users current needs and he might want to use a default value for some properties without having to manually set them in each new configuration. The solution could be implemented in a preference page that contains the following:

1. a way to set the default properties for all KIEM properties and possibly for user defined properties as well.

2. a way to set which of these properties should override the value stored in the execution file and which should only be used if the execution file doesn't contain one.

2.2 Easier configuration loading

The last objective of this thesis is to make it easier to load execution files. Currently all execution files are stored in the workspace at a place of the users choice. In a very large workspace it can be very hard to find the execution file that you need for your current simulation. The list of recently used documents that Eclipse provides is of little use since all opened documents are placed there not just execution files. This problem leads to the in the following tasks:

1. Finding a way to track recently used execution files and make it easier for the user to load them without having to locate them inside his workspace.

2. In addition to tracking recently used execution files the user might want to have a way to have a list of execution files that work for the currently active editor. This list should be sorted with the most likely candidates at the top to allow less experienced users to select an execution file that will most likely work.

3 Used technologies

- before explaining solutions
  - short explanation about the technology in question
  - small sketch to get an idea of the context
  - full explanation goes beyond the scope of this work

3.1 Eclipse

- KIELER part of Eclipse framework
  - Jave IDE but also lots of other languages (C++, Latex, ...)
  - can build other IDEs with it
  - citation: IDE an for anything, and nothing in particular [2]

3.1.1 Plug-ins

- KIEM itself is a plugin, this work plugin dependant on KIEM
  - different components of an Integrated Development Environment (IDE)
  - can operate by themselves or depending on other plugins
  - in addition to Application Programming Interface (API) of each plugin: extension point mechanism
  - extension points to interact with other plugins
  - extension point mechanism used by runtime environment to check which plugins to load
  - plugins can be used to plug into the standart Eclipse feature

Extension point mechanism

3.1.2 Preference Pages

- One of the plugins used in this work in that fashion is the org.eclipse.ui.preferencePage plugin. It is used to create new preference pages at a specific location inside the normal tree of preference pages accessable through Window>Preferences. The programmer only has to take care of the contents of the actual page and not worry about addional buttons or integrating it into the PreferenceDialog.

Preference Store

- Closely coupled with the preference pages is the eclipse preference store. It is basically a text file for each plug-in where the plug-in can deposit simple Strings

under a given key to ensure that information is kept between execution runs.

For addional information about eclipse see www.eclipse.org or literature [3]

4 Concepts

The solution to the problems outlined in chapter 2 can be achieved with the help of the Eclipse plug-in technology described in chapter 3.

4.1 Configurations

The first approach to saving additional configuration information in the execution files would be to actually modify the format of those files and write the information into them. This would most likely be the easiest approach but would destroy backward compatibility of those files since the basic KIEM would not know how to deal with the modified files. The approach taken in this thesis is based on the list of DataComponents.

Each execution file carries a list of its own DataComponents and their properties to ensure that the component are properly initialized the next time the execution is loaded. That list can be loaded even if there are DataComponents contained in it that are not in the current runtime configuration. The KIEM will just show a warning but load the rest of the execution anyway.

These DataComponents basically just carry a list of KiemProperties which are just (key, value) pairs. This makes them ideal for storing simple information like the value of a text field.

To solve our problem we will just construct a new type of DataComponent and register it through the extension point in the KIEM. This ensures that the component can be added to any execution file. The KIEM ensures that all properties contained in our component will be saved with the execution file and loaded the next time the file is opened. All we have to do is find our component inside the DataComponent list, get the properties saved inside it and set them inside the KIEM.

4.1.1 Default Configuration

In order to provide a place to manage the default configurations we will be using the Eclipse preference pages.

The root page for the KIEM should contain the basic KIEM properties like the aimed step duration, timeout and the view elements of the Configuration plug-in.

Then we will construct another page for managing the different schedules and their editors. For that we will adapt the LayoutPreferencePage by Miro Spönemann as seen in Figure 4.1 where we will just replace the layouts with the list of registered schedules.

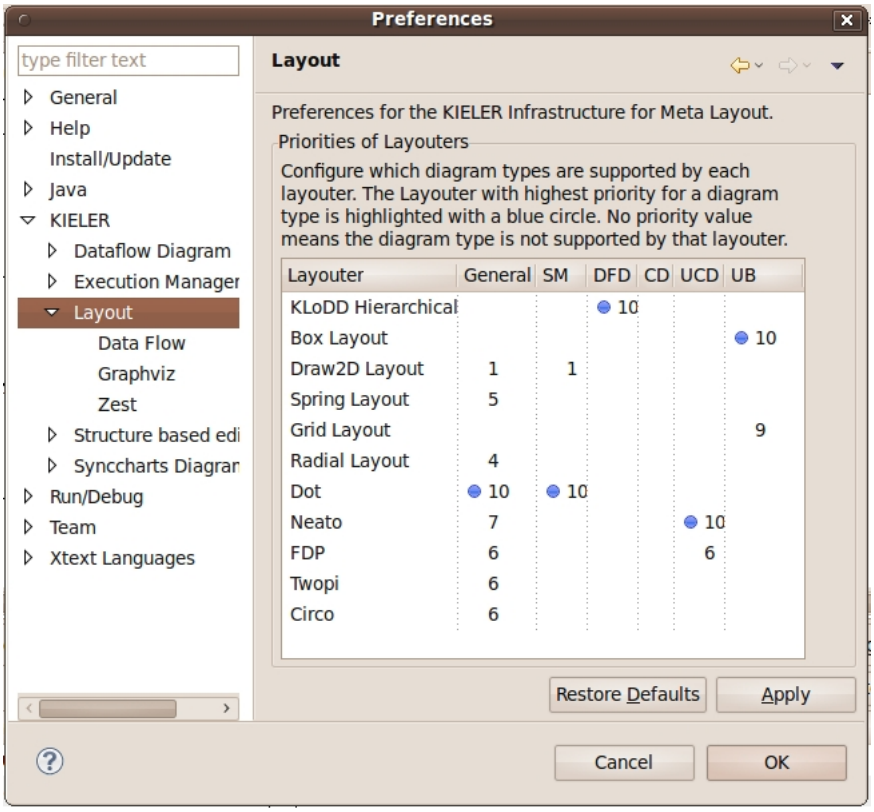

Figure 4.1: Layout Preference Page by Miro Spönemann[1]

In order to sort the schedules that match the currently opened editor they will need to have user defined priorities which can be easily set up with the table shown above.

The last page will be used to allow the user to set up his own properties and give them default values.

The values entered in those pages will be stored inside the Eclipse Preference Store.

4.2 Easier Configuration loading

For easier configuration loading we will add two ComboBoxes to the existing tool bar inside the KIEM view.

One of them will display the list of recently used schedules the other one the list of schedules that work for the currently active editor.

As soon as the user opens a new execution file through the normal workspace view we will be notified of that event by the KIEM. We will then create a new schedule and store the path to the execution file in it along with the editor that was opened at the time that the schedule was created. The new schedule will also be added to

the list of recently used schedules that we maintain through the use of the Eclipse preference store.

When the user selects one of the previously saved schedules in one of the ComboBoxes we will retrieve the saved path and offer it to the KiemPlugin to load it in the hopes that the execution file is still in the same location and wasn't deleted, renamed or moved by the user.

5 Code changes in the KIEM

Although the project attempts to realize most of the objectives without changing the KIEM itself some changes were necessary. This mostly involves adding new methods to the API in order to gain access to previously hidden properties.

Also some changes had to performed where properties were loaded from hard coded default values. These were changed so that the hard coded default is only used if the KIEMConfig plug-in is not registered to supply previously saved properties.

However all changes that were made to the KIEM plug-in were merely additions and shouldn't break any plug-ins relying on the old implementation.

				Test Component	
Model file	Iteration	Status	Finished on step	Iteration	Step finished
/test/default.kids	0	Done	7	0	7
	1	Done	7	1	7
	2	Failed	2	2	2
	3	Done	7	3	7
/test/test.strl	0	Done	7	0	7
	1	Done	7	1	7
	2	Failed	2	2	2
	3	Done	7	3	7

				Test Component	
Model file	Iteration	Status	Finished on step	Iteration	Step finished
/test/default.kids	0	Done	7	0	7
	1	Done	7	1	7
	2	Failed	2	2	2
	3	Done	7	3	7
/test/test.strl	0	Done	7	0	7
	1	Done	7	1	7
	2	Failed	2	2	2
	3	Done	7	3	7

Figure 5.1: The interface for Toolbar Contribution Providers[1]

5.1 Schema files and Interfaces

In order to provide additional functionality for other plug-ins we choose the extension point mechanism described in section 3.1.1. This is done is order to retain the old functionality of the plug-in while on the other hand giving options to ask extending plug-ins for their contributions.

The extension points described in greater detail in the next sections consist of a schema file for defining the extension point and an interface that contains the methods that extending components have to supply.

5.1.1 Toolbar Contribution Provider

The purpose of the toolbar contribution provider is to allow other plug-ins to put items onto the toolbar. There are two reasons for using the extension point mechanism rather than making the toolbar available and have other plug-ins put their contributions directly on it:

1. At the moment the toolbar and all contributions are created programatically. Switching the entire native toolbar of the Execution Manager to adding the actions to the toolbar through the org.eclipse.ui.toolbar extension point would require major code changes.

2. A programatic approach gives control over the contributions to the Execution Manager. This means that the order of the native Execution Manager buttons is always the same and in the same place. It also means that the Execution Manager can choose to ignore contributions if the toolbar gets too crowded.

The schema file for components that want to add contributions to the toolbar is quite simple. It only requires them to implement the interface shown in figure 5.1. The implementing class provides an array with all ControlContributions they want to add to the toolbar. A ControlContribution for a toolbar can be almost any swt.Widget like Labels, Buttons, Comboboxes, ...

When the Execution Manager starts to build the views toolbar it will start by asking the contributors for their contributions, add them to the toolbar and then adds its own elements. This causes the toolbar to have the native elements always in the same order. The contributed elements will be added from left to right in the order that they are in the array. However there is no guarantee on the order in

which the extending plug-ins are asked. Figure 5.2 shows the Execution Manager toolbar with the two combo boxes belonging to KIEMConfig contributed through the extension point. Figure 5.3 shows the same toolbar without the contributions.

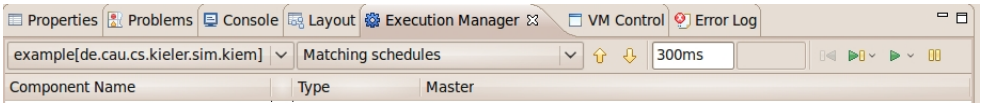

Figure 5.2: The Execution Managers Toolbar with two contributed Comboboxes[2]

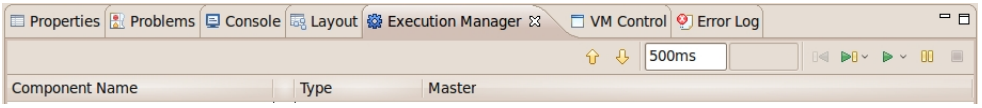

Figure 5.3: The Execution Managers Toolbar without contributions[3]

5.1.2 Configuration provider

The purpose of the configuration provider is to allow internal attributes of the Execution Manager to be stored in another plugin.

This is done through an extension point to allow any plug-in to listen to changes in the Execution Manager's attributes. It also means that there may be multiple plug-ins that provide values for properties and not all plug-ins may have the value for a property needed by the Execution Manager. Through the plug-in mechanism the KIEM can ask all providers for values and choose the one he would like to use.

The two methods from the interface shown in Figure 5.4 work in the following way:

5.1.3 Event Manager

The main purpose of the Event Manager is to inform DataComponents of events happening in the KIEM during execution. This behavior has been modified to include events that happen while the execution is not running. This modification lead to the creation of another extension point in order to allow other plug-ins to be notified on any of these events as well. The classes implementing the interface (see figure 5.5) required by this extension point will be notified on any event that happens inside the KIEM.

The semantics of the methods will be explained in detail in the following two subsections.

provideEventOfInterest()

This method is directly derived from the method with the same name in the AbstractDataComponent class of the KIEM plug-in. It is called by the EventManager

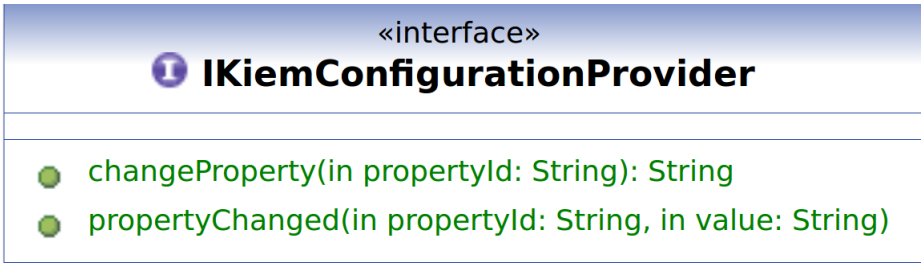

Figure 5.4: The Interface of the Configuration Provider[4]

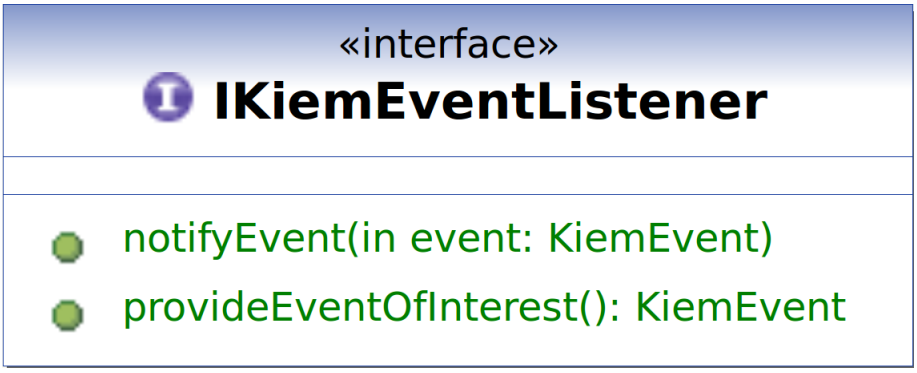

Figure 5.5: The Interface of the Event Listener[5]

to determine which events the implementing class is interested in. This is done to improve effiency and not flooding components with events they are not interested in.

Based on the response the EventManager puts the component into lists along with the DataComponentWrappers already inside.

notifyEvent(KiemEvent event)

This method is called by the EventManager when something happens inside the KIEM that the implementing classes might be interested in.

5.2 KIEMPlugin.java

The main Activator class contains almost the entire API of the KIEM. Therefore any additions to that has to performed in this class which means that most of the adjustments were made here.

1. A couple of methods were added to inform the listeners registered through the ConfigurationProvider extension point of changes the user made through the U (UI) of the KIEM.

2. The method that takes care of loading an execution file was split. This was done to allow other plug-ins to pass an IPath object directly to the method and perform a load of that file without having to go through the UI. This method was also shifted around a little in order to detect missing execution files before the load enters the UIThread. This was necessary to make is possible for the callers of the method to catch the resulting exception. Another modification was that the method has to be able to process files that are not in the currently running workspace but were added through an extension point.
3. getter for configuration elements
4. relay to KiemView to trigger updates

5.3 KIEMView

- changes to toolbar creation
- provide means to set view as dirty when changes happen
- refresh method to reload values, not needed before because no change except by user

6 KiemConfigurationPlugin

- own plugin for modularity, reuse of old code, no interference with other plugins

6.1 Implementation of extension point

6.1.1 The implementing classes

- toolbar provider: link to contribution manager, forward array of contributions
- configuration provider: link to configuration manager, forward requests
- event listener: handles events received by the configData component listen to load/save events, can be disabled when KIEMConf is about to trigger load/save

6.1.2 Manager classes

- bundled control behavior
- no cluttering up the main activator class
- singleton pattern

Abstract manager superclass

- provides methods that all managers need
- load/save through the plugins preference store
- add/remove/notify listeners

Schedule manager

- all schedule data
- track recently used schedule Ids
- ask KIEMPlugin to load a saved schedule, deal with error
- add/remove schedules, update schedule priorities
- handle user load/save events

Configuration manager

- manages current/default configuration, supply filtered lists for preference components
- find current configuration and wrapper in data component wrapper list
- get values for properties (where to look, ignored keys, save to current when found)
  - check if there is a current configuration, check if the key is not on the ignored key list, try to find in current configuration
  - on failure: try to find in default configuration, update current configuration, return found value
  - on failure: if default value supplied, update default configuration, update current configuration, return default value
  - on failure: throw exception
- add/remove properties, update values
  - try to update saved values
  - if it doesn't exist create it
- restore default values

Editor Manager

- all editor definitions
- add/remove/find editors
- default editors

other managers

- contribution manager: create combos, save visibility
- property usage manager: track all keys where default config should be used rather than current

6.1.3 Utility and Data classes

ConfigDataComponent

- extends AbstractDataComponent from KIEM, registers through extension point
- can be added and removed by the user to update old files or downgrade new ones
- contains an array of KIEMProperty (key, value pairs)

- contains methods to more easily modify the properties (update, remove, add, find)
- used as current/default configuration
- reference to wrapper file to get serialized properties and set for serialization
- delegate events received from the Kiem Event Manager

ScheduleData

- tracks one specific .execution file
- contains file location and list of priorities of supported editors
- getting/setting supported priority, adding/removing editors

other data classes

- EditorDefinition: holds editorname, editorId pair gotten from plugin definition
- KiemConfigEvent: for event dispatching
- Wrapper classes: type safety, prevent null values

Tools

- Tools class:
  - holds a lot of Strings used in multiple classes
  - methods for converting array to list and back
  - displaying error dialog
- MostRecentlyUsedList: list for tracking recent uses, adds new elements at index 0, pushes down rest, pulls element already in list to top

MostRecentlyUsedList

The MostRecentlyUsedList 6.1 is a new list type that is can be used for simulating the behavior found in 'Open recent' menu item of almost any text editing application. To avoid the list growing too long it can be given a maximum capacity. After that capacity is reached the oldest entry will be deleted when a new one enters the list. The data of the list is stored in an ArrayList with an initial capacity of the maximum capacity of the list. Most operations are directly delegating to the operations of the underlying ArrayList. The only exception is the add(item : T) method that works in a different way:

1. check if the item is already in the list and remove it

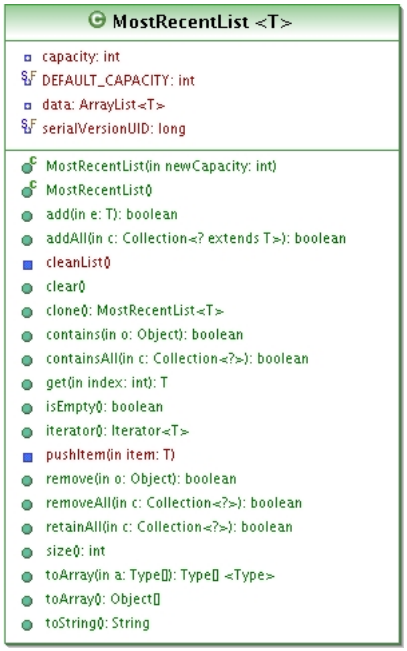

Figure 6.1: MostRecentlyUsedList[1]

2. add the last item at the end of the list and increment the index of all other items

3. set the first index to the newly added item

4. remove the last item if the list has grown beyond the capacity

This list is used to track the most recently used schedules and display them in the corresponding combo box.

6.2 Preference pages

- place to easily configure settings, all KIELER preferences in one place
- integrated into eclipse look and feel

6.2.1 Configuration page

- changing default configuration for internal properties
- check boxes for changing visibility of the combos

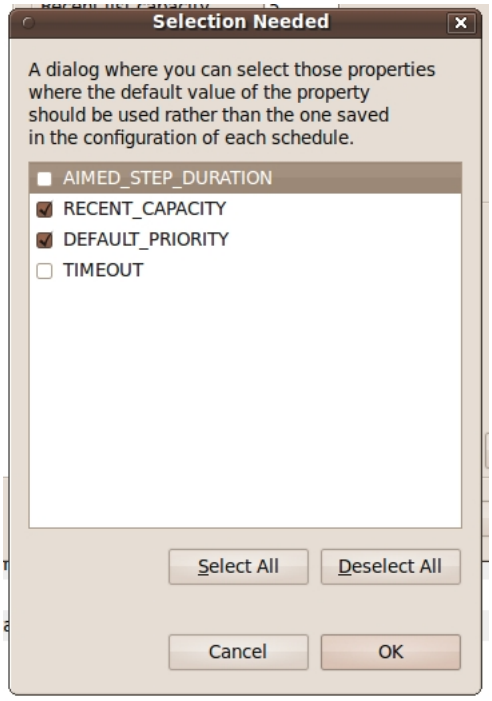

Figure 6.2: The Property Usage Dialog[2]

User defined properties page

- adding/removing properties
- modified from msp, table view with providers

Property usage dialog

This dialog shown in figure 6.2 is used for selecting which properties should always be taken from the default configuration rather than the configuration component contained in every .execution file. The dialog used for this is a ListSelectionDialog which just receives the list of all keys as input and the list of PropertyKeys from the PropertyUsageManager as default selection. After the user is finished with selecting attributes and hit the 'Ok' Button the dialog passes the new list of selected items back to the PropertyUsageManager.

6.2.2 Scheduling page

This preference page is used to manage the schedules and the editors that they belong to. This page is basicly a modified version of the LayoutPrioritiesPage by msp.

- table with schedules / editors and their priorities

- modified from msp , editors = diagram types, schedules = layouters
- modify priorities
- add/remove editors, remove schedules, selecting default editor

Adding and removing editors, Selecting a default editor

On the scheduling preference page there are routines for adding and removing editors as well as selecting a default editor. All of these actions use the same basic method for displaying an ElementListSelectionDialog 6.3 that takes a list of editor ids and returns the one selected by the user.

- The editor adding dialog gets a list of all editors currently registered on the active workbench. The user can select a single editor which is then added to the table.
- The editor removal dialog gets a list of all editors currently available for assignment of support properties. The editor selected by the user is removed from the table. It is also removed from all schedules. This is done to prevent the schedule objects from growing to monstrous size over time when editors are getting added and removed.
- The default editor selection dialog gets the same list as the removal dialog. The selected editor is then set as default editor. The default editor is used when there is no currently active editor on the workbench. It is used:
  1. to determine which editors to show in the Matching combo box.
  2. when a new schedule is created as an editor id.

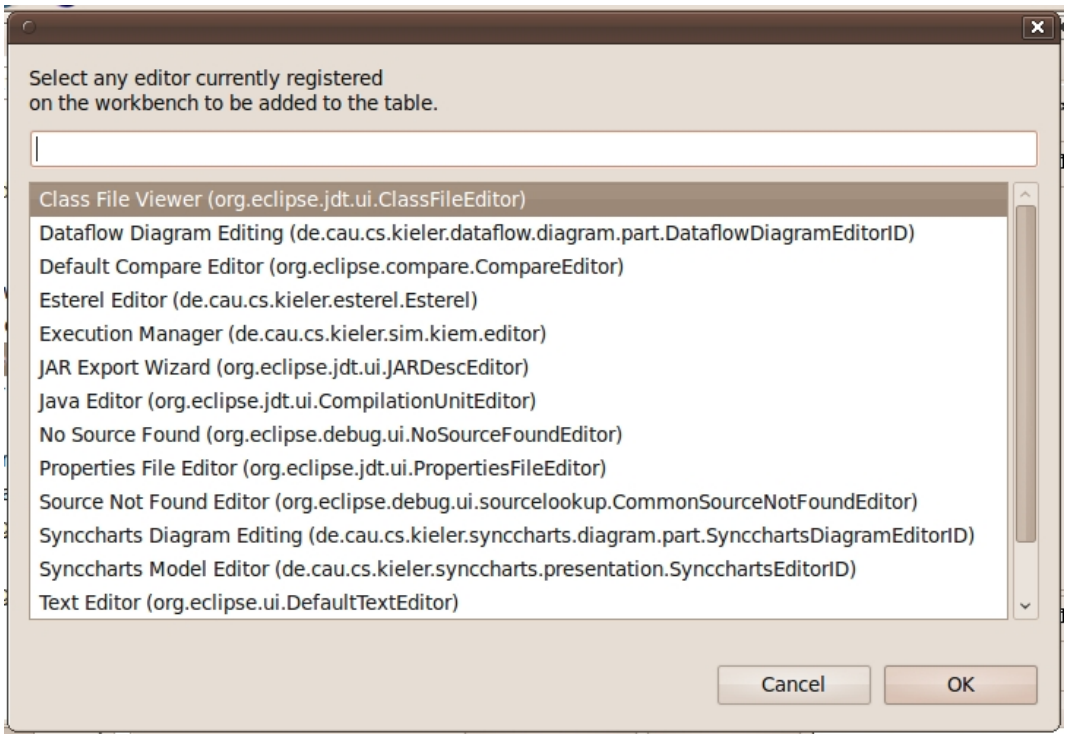

Figure 6.3: The Editor Selection Dialog[3]

7 Conclusion

7.1 Results

- problems solved

7.2 Further improvements

- allow Objects (Serializable) to be saved
- improve dynamic layout
- refactore entire execution manager to use the runtime mechanism

Part II

Automated Execution in the KIEM

8 Problem Statement

In order to do validations or recording automatically in a batch like mode, the Execution Manager needs to be extended by an Eclipse Plug-in that some kind of remote controls it (API) and has/supports for example:

- An own eclipse view (GUI)
- Loading and saving batch files
- Some way to specify batch jobs (e.g., a special notation language)
- Possibly a way to do all this from the command line

Currently the KIEM works in a way that you select an execution file and then press play. The components then have to gather all information they need themselves like model files, trace files and so on. The execution runs until the user or a component stops it. The user then has to manually set up another execution run, possibly even rewriting his components if the model files and trace files are hard coded. This is very unsatisfactory if you have a large number of model files that should be tested with a one or more execution files and several trace files.

The problem can be broken down in 4 parts which are explained in detail in the following sections.

8.1 Setting up an Automated Run

The first objective is to find an easy way for the user to efficiently set up an automated run. This involves selecting the model files and execution files needed for the automation as well as entering initial properties.

8.2 Input for the Automation

The second objective is to enable the components to receive inputs. Each component should receive all information it needs prior to each execution run in order to move away from having to hard code the locations of model files and trace files into the component. We will have to define an API for this information passing process as well as an API to trigger an automated execution from other plug-ins.

8.3 Automate the execution

The third objective is to automate the process of execution itself. This would involve the following:

1. loading an execution file
2. stepping the execution to the required step
3. gather the information produced by the components
4. properly shut down the execution so that a new one can be started

8.4 Output of the results of the execution

The last objective is to display the information in a meaningful way. This should involve at least two methods of output:

1. A formatted string possibly in an XML fashion that can be parsed and used by other plug-ins for automatic analysis.
2. Some graphic component that will display the information in way that is easy to read for most users.

9 Used technologies

The technologies used in this thesis are basically the same as in the first part (see Chapter 3).

9.1 Eclipse WorkbenchJob API

Since we don't want the UI to block while we are doing our automated execution run we will use the Eclipse Job API to run our automated execution thread. The Job API is specifically designed to accommodate very long running tasks which makes it perfect for our purposes since an execution run with several model files and hundreds of trace files might a whole night. An example for the use of jobs in the normal Eclipse architecture is the SVN commit operation seen in Figure 9.1.

9.2 Eclipse Wizards

To allow the user to set up the execution run we will be using the Eclipse Wizard API. Wizards are used to guide the user through the process of creating complex items by taking the information in a structured way and then generating the item from it. One example inside the Eclipse Architecture is the Java Class Creation Wizard (see Figure 9.2) In theory it is possible to just open a text file and enter all the information manually. However if the wizard is used the user only has to select the class he wants to extend and the interfaces he wants to implement, activate one check box and then the wizard will create the class body, all required methods and comments for each element (see Listing 9.1) This makes it very easy for even inexperienced users to create new classes without knowing the exact syntax.

Auflistung 9.1: Code generated by the wizard

```
package test;

import org.eclipse.jface.action.ControlContribution;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;

/**
 * @author soh
 */
public class MyClass extends ControlContribution implements Runnable {
```

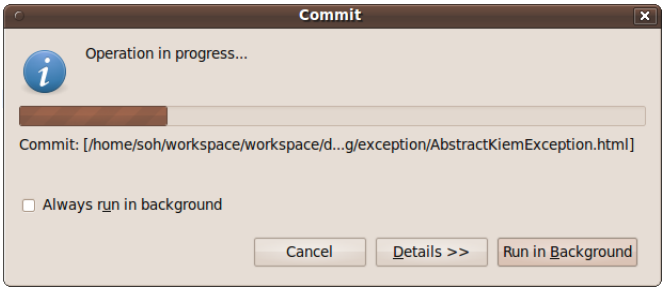

Figure 9.1: The SVN commit job[1]

```
    /**
     * Creates a new MyClass.java.
     *
     * @param id
     */
    public MyClass(String id) {
        super(id);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    protected Control createControl(Composite parent) {
        return null;
    }

    /**
     * {@inheritDoc}
     */
    public void run() {
    }

    /**
     * @param args
     */
    public static void main(String[] args) {

    }
}
```

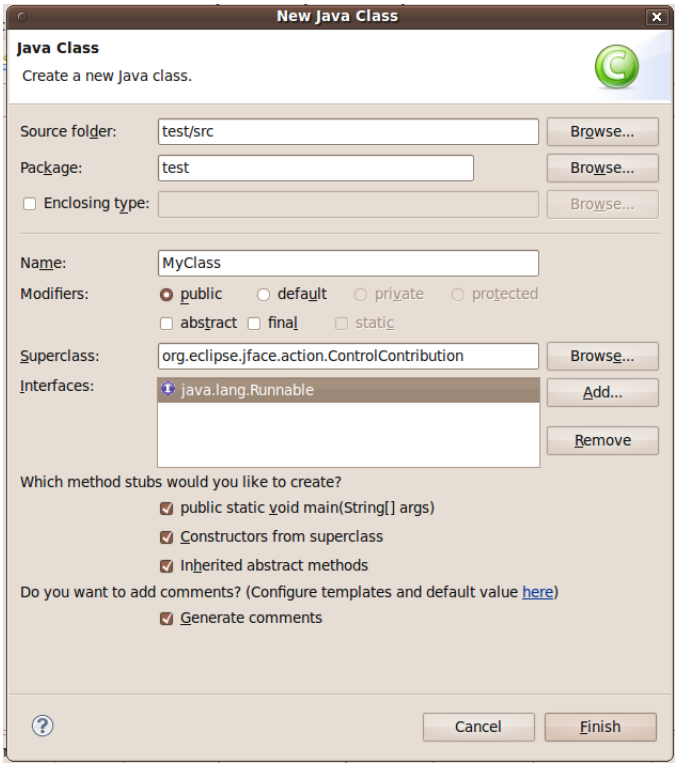

Figure 9.2: The Class Creation Wizard[2]

10 Concepts

10.1 Setting up an Automated Run

There are several possibilities of how to solve the problem of accumulating large amounts of information prior to a long running action. The first possibility would be to have the user enter the paths to the necessary files in text files, parse those files and start a run with the parsed information. While this is a good method for performing static runs from a console environment it has several disadvantages inside the GUI of an Eclipse RCA:

- Manually entered file names in a text file are prone to have erroneous information. It is very hard to manually enter the correct file name of any file and the entered location only works on one OS. Aside from that it takes a long time to manually enter the possible wast amount of files used.
- There is no way to quickly adjust the file if other models or execution files should be used.
- It also means more files cluttering up the workspace.
- It is not very intuitive and the user has to know the exact syntax that the execution needs.

A far easier approach is the selection of the files through the use of a dialog. Here the first option is to write a new Dialog from scratch. While this option ensures flexibility since only the elements that are really needed are on it in exactly the way they are needed it has also a few downsides:

- It involves a lot of work since every widget has to be manually placed on the dialog.
- It involves even more work to get the layout of the dialog just right.

The easiest way it to use one of the dialogs provided by Eclipse specifically the wizard type dialog. Eclipse itself uses a host of wizards as explained in [ref here]. The wizard has several advantages over the other methods explained here:

- Even inexperienced users can be guided to set up a valid execution run.
- The entered information is most likely valid since the wizard only displays valid files.
- It is quicker to program and easier to adjust than any of the other methods.

10.2 Input for the Automation

In order to input information into the DataComponents the first decision has to be in what form the information will be supplied. The chosen form is that of a list of key, value pairs. It allows for the most flexiblility while still being very generic and simple to read and write on. This list of properties will at least include the path to the model file in order for components to be executed with several different model files without having to alter the code between runs.

The next question is how the components will get the information.

The first possibility would be to have the component ask the plug-in for the information in question. The upside of this would be that components are sure to get all the information they need before the execution can start since they can keep asking for it. However this would likely mean that the component would have to ask multiple times as they cannot know when the required information will be available which constitutes additional workload. Furthermore this situation would likely mean that multiple components might request information at the same time. This means that there would be the need for substantial synchronization mechanisms to ensure consistency of data.

Therefore the way chosen in this thesis is that the Execution Manager will inform interested components about all properties that were accumulated and then starts the execution run. This ensures that a run is started in any event and keeps communication between the components and the manager simple.

10.3 Automate the execution

Automating the execution itself requires the plug-in to interact with the KIEM. There is already an API defined for loading an execution file by supplying a path so that is what will be used in this project. Then it is neccasary to initialize the execution and step through it using the API methods provided in the Execution. For this the EventListener extension point of the KIEM can also be used in order to determine when a step has finished executing and a new one can be dispatched. After the execution is finished all components should be called again to be given a chance to provide some information for the display in the next step. This information will be gathered in the same form and way as described in [ref to pref section].

10.4 Output of the results of the execution

On the subject of displaying the information again several options are available.

The last objective is to display the information in a meaningful way. This should involve at least two methods of output:

1. A formatted string possibly in an XML fashion that can be parsed and used by other plug-ins for automatic analysis.

2. Some graphic component that will display the information in way that is easy to read for most users.

11 Code changes in KIEMPlugin

- interfaces and API changes to allow access to execution
- changes to load values rather than hard coded defaults

11.1 Schema files and Interfaces

- event listener mentioned in part I, for listening to KIEM execution events
- interfaces added to KIEM itself to avoid components breaking when automated plug-in is not loaded

11.2 Automated Component

An automated component is any DataComponent that wants to interact with the automated execution plug-in. As seen in the diagram automated components have to implement three methods:

11.2.1 provideProperties()

This method enables components to receive information prior to each execution run. The list is implemented as an array of key, value pairs stored inside KiemProperty objects. At the every least the list contains the location of the model file and the index of the currently running iteration (that is how many time the current model has already been executed with the current execution file). This allows components to load additional files that are always in the same path as the execution file and determine which of those to load based on the iteration index.

11.2.2 wantAnotherStep()

This method is called after every step of the execution manager. All components are asked if they want to perform another step and if one components answers with TRUE the execution manager will perform another step. This makes it unnecessary to know the exact number of steps needed for the execution before the execution starts.

11.2.3 wantsAnotherRun()

This method is the equivalent for the wantAnotherStep() method in the context of entire execution runs. After no component wants to perform another step all components are asked if they want to perform another run. If one component answers with TRUE the entire simulation is stopped and then started again with the same model file and execution file but an incremented iteration index. This can for example be used to execute the same model with a different set of trace files by just naming the trace files the same way as the model files with a number at the end.

11.3 Automated Producer

This interface extends the AutomatedComponent interface. In addition to the inherited methods it provided one additional method. This method is called after an iteration has finished and asks the components if they want to publish any information about the results of their execution. This information is gathered by the plug-in and the accumulated results are either passed to the calling plug-in or displayed in the specially designed view (see chapter about the view).

12 The Automated Executions Plug-in

- new plugin - handles the setup, control flow and display of automated execution - consists of 3 parts explained in detail below - wizard, manager, view

12.1 The wizard

12.1.1 File selection page

- wizard is used to set up the execution run - extends ResourceImportWizard for displaying a folder/file structure for selecting files from - easily usuable, select whole folders, filter file types - can be given an initial selection, on close will save the selection, store it in preference store and restore it on load - additional dialog for selecting execution files that are not in the workspace but imported - for simplicity assume that files ending .execution are execution files and all other selected ones are model files, wizard can not check if valid since formats are not known - only allow user to proceed if at least one execution and one model file is selected

12.1.2 Property Setting page

- set up the additional arguments passed to the execution - simple adding and removing of key, value pairs - same as file page, on close results are saved to preference store and restored for initial selection on next open

12.1.3 Processing the information

- gather execution files and model files from file selection page - gather properties from property page - store information for next open - invoke the execution manager

12.2 The Execution manager and job

- handles control flow during the automation - takes information from either call through the API or wizard

12.2.1 Execution Manager

- handles the overall control flow - takes the execution files, model files and properties as argument - if progress monitor is registered it is informed about the progress of the evaluation - The control flow:

- iterate over all execution files
- open execution file
- tell view to set up display for the first execution file
- iterate over all model files
- get first model file from list
- ask components how many more runs they need, take maximum and perform runs before asking again
- pass model file, execution file and index of iteration
- initialize the execution
- pass properties to components, at least receive model file and iteration
- start worker thread that listens for monitor canceling, step timing out
- ask components how many more steps they need, take maximum and perform steps before asking again
- perform one step, lock self inside semaphore, stay locked until either worker thread or event listener notifies (step done)
- when no component wants more steps pause
- gather information from all IAutomatedProducers
- tell view to show information for this iteration
- stop execution inside the KIEM and perform cleanup
- proceed to next iteration
- inform monitor of progress
- proceed to next model
- proceed to next execution
- when done inform monitor of done and terminate the job

12.2.2 Execution Job

- workbenchjob with progressmonitor - used to display the progress in the progress view and a dialog with progress bar - long running task, doesn't want to block the rest of the workbench - triggers execution in the manager

12.3 The View

- displays the information in a structured way - start a new table for each execution file - one row for each iteration with each model file - prerequisite needed here: always the same outputs throughout the entire execution file - first columns display model file name, iteration index and current status

12.3.1 Toolbar

- button to start the wizard - button for clearing the view - text field showing the step that was just processed

13 Conclusion

13.1 Results

13.2 Further improvements

- implement macro step support as soon as KIEM does

Index

Bibliography

[1] Christian Motika, Semantics and Execution of Domain Specific Models, 2009.

[2] Object Technology International, Inc. Eclipse Platform Technical Overview, 2003.

[3] Eric Clayberg and Dan Rubel. Eclipse Plug-ins. Addison Wesley, 2009.