Structure Based Editing (KSBasE)

Project Overview

Related Theses:

- Michael Matzen, *A Generic Framework for Structure-Based Editing of Graphical Models in Eclipse*, March 2010 (pdf)
- Hauke Fuhrmann, *On the Pragmatics of Graphical Modeling*, Disputation: 2011-05-05 (pdf)

The KSBasE ***[Kay-space]*** project enables developers to add structure based features to an EMF-based editor. The term *structure based* refers to the fact that defined features are based on the editors EMF meta-model.

Java Projects

The following Java projects belong to this project:

- de.cau.cs.kieler.ksbase - Base classes of KSBasE feature, including extension points
- de.cau.cs.kieler.ksbase.ui - UI contributions, e.g. a preference page (found under Preferences->KIELER->KSBasE)
- de.cau.cs.kieler.ksbase.feature - Feature file with the latest KSBasE-Editor

Example Projects

- de.cau.cs.kieler.synccharts.ksbase - Synccharts example project
- de.cau.cs.kieler.keg.ksbase - Graph editor example project
- de.cau.cs.kieler.kaom.ksbase.ptolemy - Dataflow example project

Requirements

- Eclipse Indigo
- Eclipse Modeling Tools (EMF, GMF, XTend)

Install

- The KSBasE project is now available in the KIELER RCP (http://rtsys.informatik.uni-kiel.de/~kieler/files/nightly/) and the KIELER update site (http://rtsys.informatik.uni-kiel.de/~kieler/updatesite/nightly/).
- Or simply check out the KSBasE projects and add it to your workspace.
- If you want to use the KSBasE UI / Viewmanagement project, you will also need the KIML and the Viewmanagement projects. Be sure to check out all sub projects too.

KSBasE Extension Libraries

- We provide a set of libraries to support the creation of transformations:
- Projects/KSBasE/UserDialogs

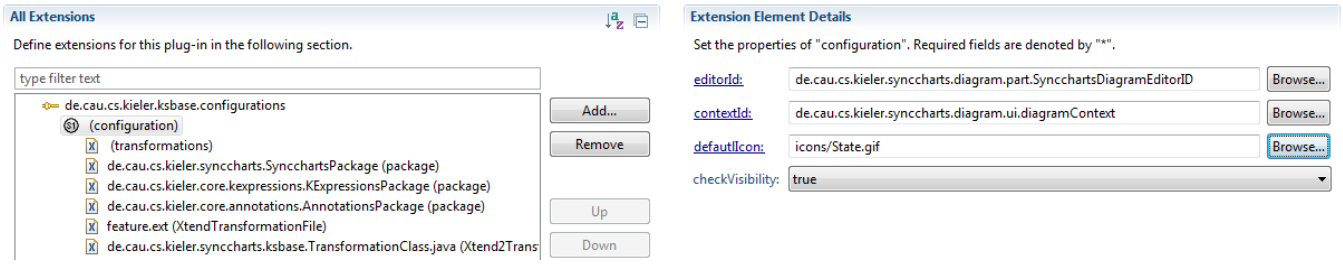
How it works

Creating structure based features using extensions

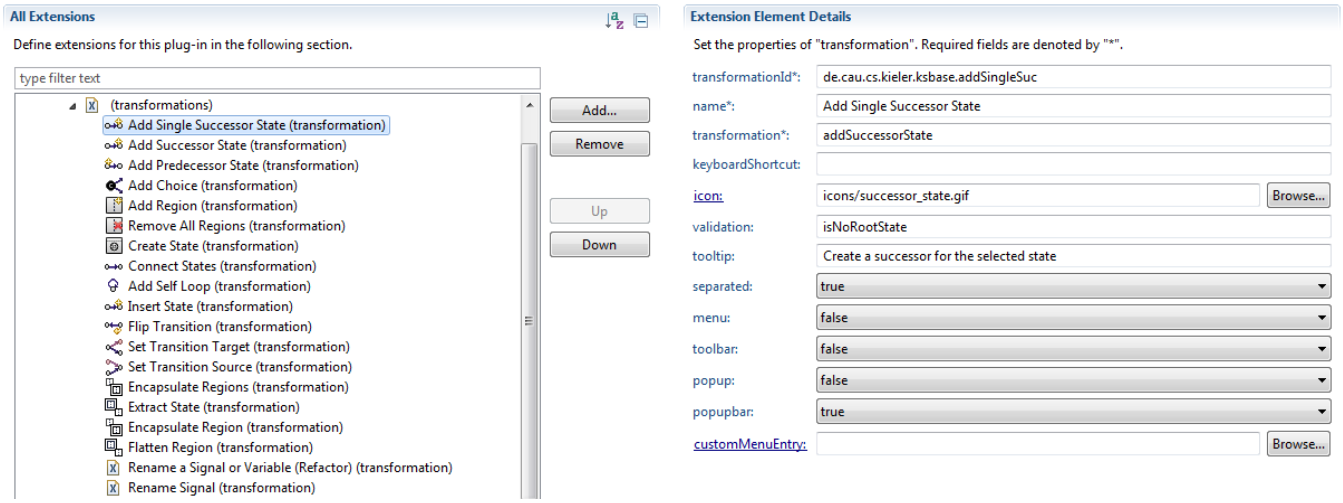
Currently KSBasE supports Xtend1 transformations as well as Java transformations (i.e. Xtend2).

It is recommend to create a new plug-in project for adding structure based editing features to your editor!
To add fancy new features to a diagram editor you can use the eclipse extension point mechanism:

- First thing you might want to do is: Write the transformation files (see the examples below).
- Now to be able to use the extensions you need to add the **de.cau.cs.kieler.ksbase** project to your project dependencies (in the plugin.xml)
- If you are using a model file which is located in another project, you have to add this project to the dependencies, too.
- Last but not least: to select a diagram editor, you have to add the diagram project of the target editor to the dependencies (look for a project called *youreditor.diagram*)
- You can now create extensions by opening the tab 'Extensions' and click the 'Add' button.
- Select **de.cau.cs.kieler.ksbase.configuration** from the extension points list.
- Now add a configuration by right clicking the new entry and selecting 'New' -> 'configuration':



- When you open the new entry, you will see the details page. Please set all properties and note the tool tip informations.
- You also have to add some more elements to the configuration. To do this click on 'configuration' and select 'New' ->
  - 'package' this is the ecore package used by the diagram you want to do transformations on. There may be more than one of this.
  - 'XTendTransformationFile' The Xtend1 transformation file holding your transformation methods.
  - 'Xtend2TransformationClass' This may be any java class holding your transformation methods. This includes classes generated by Xtend2. There should be either an XTendTransformationFile or at least one Xtend2TransformationClass present in your extension point.
- To specify the transformations you have to use the 'transformation' sub entry which should already be present. If that is not the case add it by using the new dialog.
- Again, you will have to set all the properties in the detail page. Please note that the property 'transformation' needs to be the **exact name** of the method in the transformation file you selected in the configuration element.
- If you want to add icons to your commands, be sure to copy them to the current project.
- If the transformation is only valid for a subset of the parameters, e.g. a root element may not have a successor, you can insert additional validations in the *validation* attribute, you can enter multiple methods by separating them with commas. Note that those transformations **need to return a boolean value** that is used by the KSBasE framework to disable or hide the corresponding UI contributions.
- Last but not least, you have to configure where the transformations should appear in your editor. You do this by setting the respective boolean field in the transformation subentry. I.e. if you set 'popup' to true the transformation button will be visible in the context menu.
- If the 'separator' property to true the will be a separator added before the menu entries of this transformation.
- It is also possible to manually add menu entries using the eclipse extension point mechnism. If you do this add the respective command id to the 'customMenuEntry' property.



Creating transformations

We are now creating some nice and simple features for the Thin Kieler SyncCharts Editor:

The first step for extending an editor is to create the model2model transformations. For the KSBasE features, those transformations are defined using Xtend.
To create the transformations, we are using a new package called *de.cau.cs.kieler.synccharts.transformations* and create a file called *feature.ext*.
Attention: You can use any package name, but the package **must be included in the build path** or else the Xtend code completion will not work.
Now we can start creating the transformation file. For now we will only create 2 transformations:

- Add a new state to an existing

- Flip the source and target of a transition

(If you'd like to see all transformations currently defined for the Syncharts Editor, you can have a look at the repository file)

The implementation of these transformations is easy:

```
import synccharts; //First import the synccharts metamodel

//Connects two states
Void connectStates(State source, State target):
let transition = new Transition:                //Create new transition
transition.setSourceState(source) ->            //Set source
transition.setTargetState(target) ->            //and target state
setSelection(transition)                        //Select the transition.
;

//Adds a successor to the given state
Void addSuccessorState(State source):
 let target = new State:                              //Create a new target state
 connectStates(source, target) ->                     //Call the connectStates extension
 source.parentRegion.innerStates.add(target) ->       //Add the new state
 setSelection(target)                                 //Select the new state
;

//Creates a default SyncChart
Void createDefault(Region rootRegion):
let state = new State:                           //Create a new root state
let innerState = new State:                      //Create a new inner state
let region = new Region:                         //Create a new region for the root state
state.setLabel("SyncChart") ->                   //Set name of the state
state.regions.add(region) ->                     //Add region to the state
innerState.setLabel("Initial") ->                //Set label of the inner state
innerState.setIsInitial(true) ->                 //The state type to initial
region.innerStates.add(innerState) ->            //Add inner state
rootRegion.innerStates.add(state) ->             //Add root state
setSelection(innerState)                         //Select inner state
;
```

A few hints on creating Xtend in-place model-to-model transformations:

- You always have to use *Void* as return value, since everything else will define non in-place transformations
- You can define local variables using the *let* keyword (no other type assignments are allowed!)
- Because Xtend is a declarative language, consecutive commands are separated using the -> operator

FAQ

- **The syntax highlighting and code completion does not work when editing the .ext file!**
  - Did you forgot to add the 'Xtend nature' to your project? Right click on the project and select 'Configure -> Add Xpand/XTend Nature'
- **I've added the Xtend nature, but the code completion is not working!**
  - Did you put the Xtend file in a non-source folder? Either right click the folder that contains the Xtend file and select 'Build path -> Use as source folder, or copy the file to a folder which already is a source folder.
- **Ok, now the code completion works, but Xtend does not recognize my meta model**
  - You have to add the project that contains the meta model ecore file to your project dependencies and you need to import the meta model by using the import statement with the exact name of the root element of your model (e.g. *import synccharts;* with synccharts. ecore)
  - Please remember that it's not possible to import meta models by their namespace URI !