SCCharts Development

This is a light-weight tutorial for developing transformations/additions for SCCharts in KIELER. It will use Eclipse, EMF, and Xtend and therefore, finishing the corresponding tutorials could prove useful. However, they are not a strict requirement for this tutorial.

Outdated!

The KiCo part "Model-to-Model Transformations with KiCo" is outdated. We will update this tutorials in the near future.

Preliminaries

There's a few things to do before we dive into the tutorial itself. For example, to do Eclipse programming, you will have to get your hands on an Eclipse installation first. Read through the following sections to get ready for the tutorial tasks.

Required Software

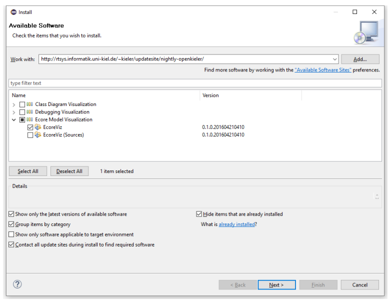
As you're going to develop for KIELER SCCharts, we recommend to use the Oomph setup as described in Getting Eclipse (Oomph Setup). However, you could also install all componentes by yourself. Please consult the other tutorials if you want to do that. You would need to install the Modeling Tools and the Xtext SDK.

Additionally, install the **EcoreViz** from the **Ecore Model Visualization** category from the **OpenKieler** update site: http://rtsys.informatik.uni-kiel.de/~kieler/updatesite/nightly-openkieler/. For this, choose *Install New Software...* in the *Help* tab.

Due to the ongoing migration you have to install a workaround for EcoreViz to function. You have to install the KLighD diagram view directly from http://rtsys.informatik.uni-kiel.de/~kieler/updatesite/release_pragmatics_2016-02/. Select the features

- KIELER Lightweight Diagrams - Developer Resources and
- KIELER Lightweight Diagrams Generic Diagram View.

(This step should be obsolete in the near future.)



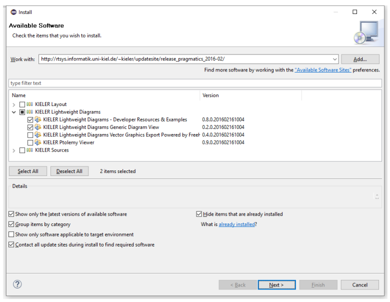
Work with: http://rtsys.informatik.uni-kiel.de/~kieler/updatesite/nightly-openkieler/

Find more software by working with the "Available Software Sites" preferences.

Name	Version
☐ Class Diagram Visualization	
☐ Debugging Visualization	
■ Ecore Model Visualization	
☑ EcoreViz	0.1.0.201604210410
☐ EcoreViz (Sources)	0.1.0.201604210410

1 item selected

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?



Work with: http://rtsys.informatik.uni-kiel.de/~kieler/updatesite/release_pragmatics_2016-02/

Find more software by working with the "Available Software Sites" preferences.

Name	Version
☐ KIELER Layout	
■ KIELER Lightweight Diagrams	
☑ KIELER Lightweight Diagrams - Developer Resources & Examples	0.8.0.201602161004
☑ KIELER Lightweight Diagrams Generic Diagram View	0.2.0.201602161004
☐ KIELER Lightweight Diagrams Vector Graphics Export Powered by FreeH	0.4.0.201602161004
☐ KIELER Ptolemy Viewer	0.9.0.201602161004
☐ KIELER Sources	

2 items selected

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

Recommended Tutorials

We recommend that you have completed the following tutorials before diving into this one (or at least sweep over them). However, this is not a strict requirement.

1. Eclipse Plug-ins and Extension Points
2. Eclipse Modeling Framework (EMF)

a. This tutorial needs the turingmachine.ecore and the controller you've implemented in the EMF tutorial. If you did not complete the EMF tutorial, you may download a working turing machine here... (in the future).

3. Xtend 2 - Model Transformations

Helpful Tutorials

When developing within the KIELER semantics team, you will most likely be confronted with Xtext and Lightweight Diagrams (KLighD). The following tutorials may be helpful but not required for this tutorial.

1. Xtext 2 - Creating a Grammar from Scratch
2. Lightweight Diagrams (KLighD)

Finding Documentation

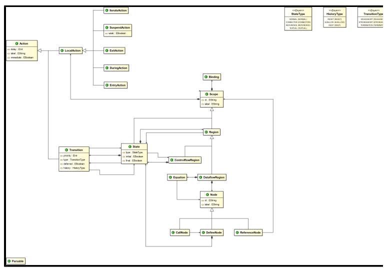
You can find additional documentation to the aforementioned topics in the corresponding tutorials. If you get stuck with a particular topic, please consult that tutorial. For SCCharts, you should read the SCCharts confluence page in our wiki: SCCharts (pre 1.0)

As usual, documentation often gets obsolete or wrong if not maintained regularly, so please, if you find missing, misleading, or outdated information, please let us know.

Additionally, the following list will give a short overview over the most important publications:

- **Main paper:**
  Reinhard von Hanxleden and Björn Duderstadt and Christian Motika and Steven Smyth and Michael Mendler and Joaquín Aguado and Stephen Mercer and Owen O'Brien. SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications. In Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14), Edinburgh, UK, June 2014. ACM. pdf, talk, bib
- **SLIC Compilation:**
  Christian Motika and Steven Smyth and Reinhard von Hanxleden. Compiling SCCharts—A Case-Study on Interactive Model-Based Compilation. In Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2014), volume 8802 of LNCS, page 443–462, Corfu, Greece, October 2014. The original publication is available at http://link.springer.com. pdf, bib
- **Theoretical Foundations:**
  Reinhard von Hanxleden and Michael Mendler and Joaquín Aguado and Björn Duderstadt and Insa Fuhrmann and Christian Motika and Stephen Mercer and Owen O'Brien and Partha Roop. Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation. ACM Transactions on Embedded Computing Systems, Special Issue on Applications of Concurrency to System Design, 13(4s):144:1–144:26, July 2014. pdf, bib
- **Overview and High-Level Transformations in Detail:**
  Reinhard von Hanxleden and Björn Duderstadt and Christian Motika and Steven Smyth and Michael Mendler and Joaquín Aguado and Stephen Mercer and Owen O'Brien. SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications. Technical Report 1311, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, December 2013. ISSN 2192-6247. pdf, bib

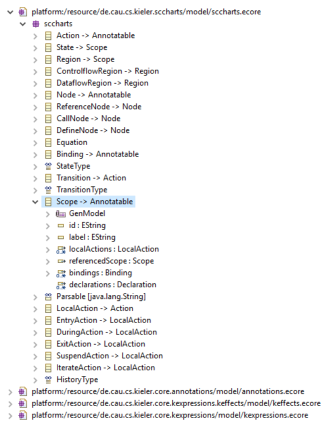
The SCCharts Metamodel



Navigate to the `models` folder of the plugin `de.cau.cs.kieler.sccharts`. Here, open the `sccharts.ecore` and right-click on the `sccharts.ecore` file and select *Visualize Ecore Model*. Since you also installed **EcoreViz** from the OpenKieler Suite, you should now see a graphical representation of the SCCharts metamodel. Every SCChart will be a model of this metamodel.

To see all class information check *Attributes/Literals* in the Diagram Options on the right.

EcoreViz gives you an overview over the selected Ecore diagram. However, Ecore model definitions may depend on other Ecore definitions that are not displayed in the diagram. You can open new diagrams for these Ecore files as described before or use the Ecore tree editor to inspect all classes.



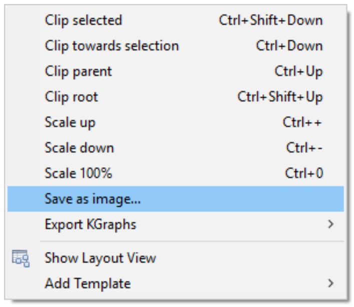
```
platform:/resource/de.cau.cs.kieler.sccharts/model/sccharts.ecore
  sccharts
    Action -> Annotatable
    State -> Scope
    Region -> Scope
    ControlflowRegion -> Region
    DataflowRegion -> Region
    Node -> Annotatable
    ReferenceNode -> Node
    CallNode -> Node
    DefineNode -> Node
    Equation
    Binding -> Annotatable
    StateType
    Transition -> Action
    TransitionType
    Scope -> Annotatable
      GenModel
      id : EString
      label : EString
      localActions : LocalAction
      referencedScope : Scope
      bindings : Binding
      declarations : Declaration
    Parsable [java.lang.String]
    LocalAction -> Action
    EntryAction -> LocalAction
    DuringAction -> LocalAction
    ExitAction -> LocalAction
    SuspendAction -> LocalAction
    IterateAction -> LocalAction
    HistoryType
platform:/resource/de.cau.cs.kieler.core.annotations/model/annotations.ecore
platform:/resource/de.cau.cs.kieler.core.kexpressions.keffects/model/keffects.ecore
platform:/resource/de.cau.cs.kieler.core.kexpressions/model/kexpressions.ecore
```

Try to understand most parts of the metamodel. You don't have to understand every detail but you should get the idea.

Model Task

- Answer the following questions
  1. How do you describe a superstate in the model?
  2. Outline the relationship between states, regions, transitions, and valued objects.
  3. Name the class of the root element of an SCChart.
  4. What is a valued object?
  5. How do you get the type of an interface variable?
  6. What other metamodels are needed for the SCCharts metamodel and write down which one is needed for what?
- Write down (on paper, text editor, etc) how the following SCChart models look like
  1. Open the wiki page that explains the Textual SCCharts Language SCT.
  2. Search the *SCChart, Initial State, State, Transition and Immediate Transition* example and ...
     a. write down (on paper or text editor, etc) how the model of that SCChart looks like.

   b. The user now marks C as final. What has to be changed in the model? What semantic problem do you see?
3. Now, navigate to the *Super State: Strong Abort Transition* example. Write down (on paper) how the model of that SCCharts looks like.
4. And finally a more sophisticated model: Write down the model of ABO (from Examples).



KLighD Screenshots

By the way: You can *right-click* on the Diagram View surface and select *Save as image...* to create a screenshot!

Creating SCCharts Models Programmatically

Creating a Test Project

We need a project for testing. Do the following:

- If you used the standard KIELER Oomph installation setup, create a new Working Set named Tutorial in the Package Explorer. Then...
- Create a new empty *Plug-In Project*.
- Add the project that contains the sccharts metamodel as a dependency of your new project through the *Plugin Manifest Editor*.
- Create a simple Java class that implements a main method. Hint: In a new Java class, simply type main and hit Ctrl+Space. Eclipse content assist will create the method for you.

Creating a Model

To create a model programmatically you cannot directly use the Java classes generated for the model. Instead, the main package contains interfaces for all of your model object classes. The `impl` package contains the actual implementation and the `util` package contains some helper classes. Do not instantiate objects directly by manually calling `new`. EMF generates a Factory to create new objects. The factory itself uses the singleton pattern to get access to it:

```
SCChartsFactory sccFactory = SCChartsFactory.eINSTANCE;
State state = sccFactory.createState();
Transition transition = sccFactory.createTransition();
```

Important: The SCCharts grammar is build on top of several other grammars. Therefore, not all language objects can be found in the SCCharts factory. For example, all expression elements are part of the KExpressions grammar and hence, have their own factory. If you need other factories, don't forget to add the corresponding plugin to your plugin dependency list.

```
KExpressionsFactory kFactory = KExpressionsFactory.eINSTANCE;
BoolValue boolValue = kFactory.createBoolValue();
```

For all simple attributes, there are getter and setter methods:

```
state.setId("Init");
boolValue.setValue(true);
```

Simple references (multiplicity of 1) also have getters and setters:

```
transition.setTrigger(boolValue);
```

List references (multiplicity of > 1) have only a list getter, which is used to manipulate the list:

```
state.getOutgoingTransitions().add(transition);
```

Plugin Dependencies

You may have noticed that is was not necessary to add a dependency for the kexpressions classes. The SCCharts plugin reexports the dependencies of the other EMF metamodels. Look at the plugin.xml in the SCCharts plugin in the dependency tab for more information.

Saving a Model

EMF uses the Eclipse Resource concept to save models to files and load models from files. It can use different *Resource Factories* that determine how exactly models are serialized. We will use the XMIResourceFactoryImpl to save our models to XML files:

- Add a dependency to the `com.google.inject, org.eclipse.core.resources, and de.cau.cs.kieler.sccharts.text` plug-ins.

  Additional Dependencies

  Don't worry. You will be experienced enough to add mandatory dependencies quickly in the future. However, for now just add the dependencies to proceed with the tutorial.

- Use something like the following code to save the model from above:

```
// Create a resource set.
ResourceSet resourceSet = new ResourceSetImpl();

// Register the resource factory -- only needed for stand-alone!
SctStandaloneSetup.doSetup();


// Get the URI of the model file.
URI fileURI = URI.createFileURI(new File("myABO.sct").getAbsolutePath());

// Create a resource for this file.
Resource resource = resourceSet.createResource(fileURI);

// Add the model objects to the contents. Usually, this is the root node of the model.
resource.getContents().add(myModel);

// Save the contents of the resource to the file system.
try
{
    resource.save(Collections.EMPTY_MAP); // the map can pass special saving options to the operation
} catch (IOException e) {
    /* error handling */
}
```

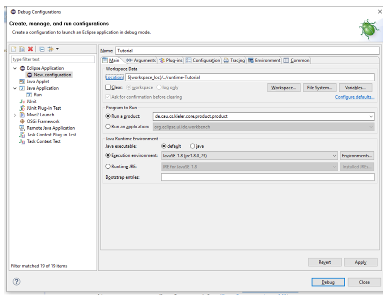
  File Extensions

  File extensions are important! They define the parser/serializer that EMF uses. Always use the file extension that is defined for a particular model.

Model Creation Task

Root.sct

```
scchart Root {
  initial state Init
  --> Init with true;
}
```



Create, manage, and run configurations
Create a configuration to launch an Eclipse application in debug mode.
Name: Tutorial
Workspace Data
Location: ${workspace_loc}/../runtime-Tutorial
Clear: workspace log only
Ask for confirmation before clearing
Program to Run
[✓ Run a product:] de.cau.cs.kieler.core.product.product
Run an application: org.eclipse.ui.ide.workbench
Java Runtime Environment
Java executable: [✓ default] java
[✓ Execution environment:] JavaSE-1.8 (jre1.8.0_73)
Runtime JRE: JRE for JavaSE-1.8
Bootstrap entries:
Filter matched 19 of 19 items

You are now equipped with the fundamentals you need to create models programmatically. Let's try it:

- The code fragments listed above do not suffice to create a grammatically correct model. Try to generate a model that corresponds with the serialized model listed on the right side.
  1. Run the `main()` method by right-clicking its class and selecting *Run as -> Java Application*. Note that this runs your `main()` method as a simple Java program, not a complete Eclipse application. EMF models can be used in any simple Java context, not just in Eclipse applications.
  2. Execute the main method.
  3. Inspect your SCT file. (Press F5 to refresh your file view.)

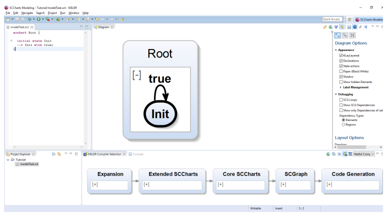
  Kext Warning

  It is possible that kext generates a Null Pointer Exception when you save your model this way. This is a known issue. We're working on it. Just ignore it for now.

- Now, create a new Java class and proceed as before to generate a model of ABO in the `main()` method.

- Start your SCChart Editor Eclipse instance and load your SCT file. KLighD should now be able to visualize your ABO correctly.
    1. For this, open tab *Run* and select *Run Configurations...*
    2. Create a new *Eclipse Application* and name it appropriately.
    3. As product select `de.cau.cs.kieler.core.product.product`.
    4. Click *Run* (or *Debug* if you opened *Debug Configuration...*)
    5. Create a new project and add you generated model.

Model-to-Model Transformations with KiCo



You can use the Kieler Compiler (KiCo) to handle all the model input/output tasks and concentrate on the actual transformation. If you executed the **Model Creation Task correctly**, you should now have a complete running SCT Editor instance that looks like the one on the right. You should see the *KIELER Compiler Selection* n the lower right part of the working space. Here you can select specific transformations that will be applied to the actual model. Simply select a transformation to test it.

Creating a new Transformation

Now, you're going to write your own transformation with **Xtend** *drumroll*, a programming language that looks very similar to Java, but which adds some very convenient features. Xtend code compiles to Java and and was developed using Xtext. In fact, once you gain experience working with Xtend you will probably appreciate the power of Xtext even more. Xtend is particularly useful to browse & modify EMF models. You get the point... we like it. :)

Transformation Creation Task

- Create your first transformation...
    1. Create a new project within your tutorial working set as before.
    2. Add plugin dependencies to `de.cau.cs.kieler.kico` and `de.cau.cs.kieler.sccharts`.
    3. Create a new *Xtend class* in you package and name it "DoubleStates" and use `AbstractProductionTransformation` as superclass.
    4. Here, you can automatically add the missing Xtend files by using the quickfix *Add Xtend lib to class path.* Alternatively you can simply add `com.google.guava`, `org.eclipse.xtext.xbase.lib`, `org.eclipse.xtend.lib`, and `org.eclipse.xtend.lib.macro` to you plugin dependencies.
    5. Click on *Add unimplemented methods*.
    6. Now, you should have a class similar to the following one.

```
package tutorial.transformation

import de.cau.cs.kieler.kico.transformation.
AbstractProductionTransformation

class DoubleStates extends AbstractProductionTransformation {

    override getProducedFeatureId() {
        throw new UnsupportedOperationException("TODO: auto-
generated method stub")
    }

    override getId() {
        throw new UnsupportedOperationException("TODO: auto-
generated method stub")
    }

}
```

Xtend Infos

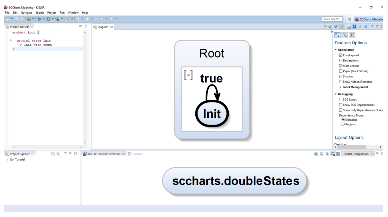
- Lines in Xtend code don't have to and with a semicolon.
- We have been explicit about the method's return type, but we could have easily omitted it, letting Xtend infer the return type.
- The keyword `val` declares a constant, while `var` declares a variable. Try to make do with constants where possible.
- The methods you call should be declared as `def private` since they are implementation details and shouldn't be called by other classes.
- You may be tempted to add a few global variables that hold things like a global input variable or a pointer to the current state. While

you could to that, `def create` methods might offer a better alternative...

- As you can see, it is mandatory to add an id for the transformation and another id of the feature that this transformation produces. Name your transformation **tutorial.doubleStates** and the id of feature you want to produce is **sccharts.doubleStates**.

Programming Guidelines

You should really think about some constants here. You can also look at the sccharts transformation and features constants in the sccharts plugin.

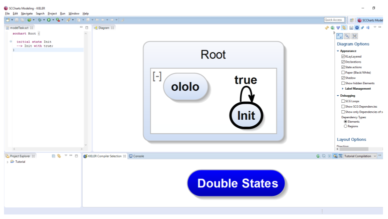


- KiCo must know about the new feature and also about your new transformation.
  1. Add a new Xtend class with Feature as superclass. Add all unimplemented methods. Also set **sccharts.doubleStates** as Id.
  2. Go to the Extension tab inside your plugin configuration.
     a. Add a new Extension Point de.cau.cs.kieler.kico.feature. Create a new featureClass and point it to your new feature class.
     b. Also add a new Extension Point de.cau.cs.kieler.kico.transformation. Create a new productionTransformationClass and point it to your transformation class.
     c. Finally, you have to link your transformation to the SCT Editor. Add the Extension point de.cau.cs.kieler.kico.ui.transformation. Create an editor link and fill in the following values:
        i. editor: `de.cau.cs.kieler.sccharts.text.sct.Sct`
        ii. features: sccharts.doubleStates
        iii. label: Tutorial Compilation
        iv. priority: 101
        v. preferred: (leave it blank)

     Plugin Tasks

     In general it is bad to mix non-ui plugins/tasks with ui plugin/tasks because (in the context of KiCo) even if you're not working with an active UI your transformations should work (e.g. a command line compiler). To keep this tutorial simple, you can add this dependency to your plugin nevertheless. However, you shouldn't do this in real products. Always keep the UI separated.

  3. If you start your KIELER instance now, you should get a new compilation chain which has only one transformation: yours, which doesn't do anything.
- If you want to rename your feature in the Compiler Selection (without changing its Id), override the `getName` method and return a new name. Rename your feature appropriately.



- Now, fill your transformation with life:
  1. Inside your transformation class, add a new method with the following signature: `def State transform(State rootState, KielerCompilerContext context)`. This transformation will be executed if the feature is selected in the Compiler Selection.
  2. Add thew following body to the function and try to understand the Xtend code. Import unknown class via code assist.

transform

```
    def State transform(State rootState,
KielerCompilerContext context) {
        val newState = SCChartsFactory.eINSTANCE.createState
=> [
            id = "ololo"
            label = "ololo"
        ]

        rootState.regions.filter(ControlflowRegion).head.
states += newState

        rootState
    }
```

  3. When selecting your transformation, the SCChart gets transformed and looks like the version on the right.

- Extend the transformation so that the transition is split up in two and connected via a transient state meaning that the original transformation should point to the new state and a new immediate transformation then points to the original target state. Try it out.
- Xtend supports extensions that can be used to extend the function set of you classes (i.e. models). Add `com.google.inject` to the dependencies of your plugin. Now, add the following code fragment to the beginning of your class.

Code injection

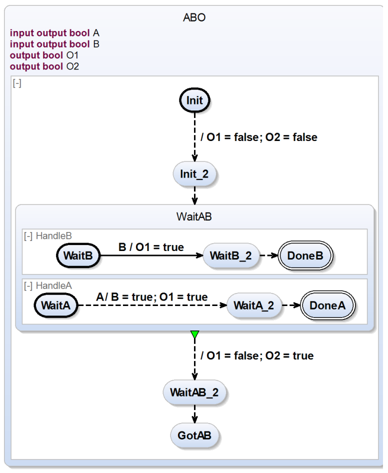
```
    @Inject
    extension SCChartsExtension
```

There are several Extensions classes within the KIELER project that extend the functionality of various classes. Basically, there are one or more for each metamodel (e.g. SCCharts, SCG, KExpressions, etc). You don't want to invent the wheel again. Use these methods. For example: there is a method that gives you all contained states of a state in a list: `getAllContainedStatesList`. You can use it on your `rootState`: `rootState.allContainedStatesList`. There are also several convenient methods for creating model elements so that you don't have to use the factories directly.

Extensions Naming Scheme

Extensions are also just classes. You can add your own to improve the structure of your own projects. In KIELER all extensions end with "Extensions"; except SCChartsExtension for legacy reasons. This will be renamed in after the next snapshot to SCChartsExtensions. So, if you're going to add new extensions to the project, please name them accordingly.

- Extend your transformation so that it is applied on all states (except the root state). Try your new transformation with ABO. The result should look like the example on the right.



The existing Compilation Chain

Congratulations. You added and executed your own KiCo transformation. Nevertheless, often you want to extend the existing compilation chain. To do this, you proceed as before but instead of creating your own compilation chain, you must modify the existing chains (e.g. the netlist compilation in de.cau.cs.kieler.sccharts.ui). To add a specific transformation at a specific point in the chain, you must tell KiCo what features are required for the transformation. For that you must override the method getRequiredFeatureIds and return a set with all required features.

Also, if you're developing for the master chain, you should obey the package structure. Look at the sccharts plugins. All features, transformation, extensions, the metamodel, ui elements, etc are separated from each other. You should always do the same!

We will add more content to this subsection in the future...

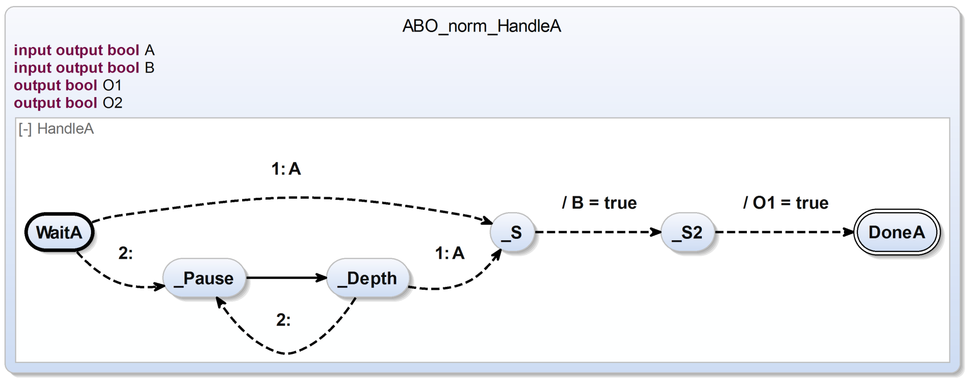
Model-to-Model Transformations between Metamodels

Transformations from one model to another may be performed within the same metamodel or from metamodel to a different metamodel. Both methods are used in KIELER and in principle they do not really differ in implementation. Nevertheless, if working within the same metamodel you should keep in mind that you're potentially changing the actual model instead of changing another instance (after copying). When transforming to another metamodel, you're always generating a new model. So there is no in-place transformation. Both is possible. Just make sure that you know what you're doing.

Now, you're going to transform the normalized form of HandleA from ABO to an SCG. The Sequentially Constructive Graph is a control-flow graph which can be seen as another representation of the same program. The SCG of the normalized version of ABO's HandleA is depicted on the right.

```scchart ABO_norm_HandleA```	

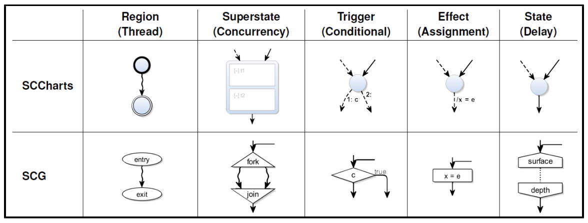
```
{

inpu
t
outp
ut
bool
A;

inpu
t
outp
ut
bool
B;

outp
ut
bool
O1;

outp
ut
bool
O2;

regi
on
Hand
leA:

init
ial
stat
e
Wait
A

-->
_S
imme
diat
e
with
A

-->
_Pau
se
imme
diat
e;

fina
l
stat
e
Done
A;

stat
e _S

-->
_S2
imme
diat
e
with
/ B
=
```


ABO_norm_HandleA

```
input output bool A
input output bool B
output bool O1
output bool O2
```

```
true;

state _S2

--> DoneA immediate with / O1 = true;

state _Pause

--> _Depth;

state _Depth

--> _S immediate with A

--> _Pause immediate;
}
```

The next figure depicts the direct mapping from normalized SCCharts to their corresponding SCG.



Inspect the metamodel of the SCGs in plugin de.cau.cs.kieler.scg. SCGs are used for analyses and optimization and include a lot of additional elements. However, for this tutorial it should be sufficient to look at the SCGraph class, its nodes attribute, the important node classes and the controlflow class. Important nodes for this SCG are entry, exit, assignment, conditional,

Transformation Creation Task 2

Write a transformation that transforms your normalized version of ABO's HandleA into its corresponding SCG.

- Proceed as before. Create a new plugin (or copy your last one) Make sure, you also add de.cau.cs.kieler.scg to your dependencies.
- Write a transformation that is able to transform `ABO_norm_HandleA` into its corresponding SCG.

- **Verify your generated SCG**. If you added your transformation correctly, the SCG should be displayed automatically as soon as selected. If your SCG looks like the SCG depicted earlier, then everything is fine.
- Check your SCG semantically. Is there anything you could improve/optimize?
    1. Write a second transformation (just as before) and add it to the transformation chain right after the transformation you already added.
    2. Optimize the given SCG and compare the result with the previous one.
    3. Make sure that the two SCGs are still semantically identical.

Congratulations! You finished the SCCharts Development Tutorial. Ask your supervisor for further instructions!