Textual SCCharts Language SCT

Online Compiler | Command Line Compiler

Quick Start Guide

An Xtext -based Project.

Project Overview

Responsible:

- Christian Motika, Steven Smyth

Formerly Responsible / Previous Projects:

- Christian Schneider (Textual SyncCharts (KITS))

Related Theses:

- Mirko Wischer, *Textuelle Darstellung und strukturbasiertes Editieren von Statecharts*, February 2006 (pdf)
- Özgün Bayramoglu, *KIELER Infrastructure for Textual Modeling*, December 2009 (pdf)
- Christian Schneider, *Integrating Graphical and Textual Modeling*, February 2011 (pdf)

SCCharts are typically modeled using the textual SCT language defined as an Xtext grammar in KIELER. Generally, if you do not know which elements can be placed at a certain cursor position you are assisted by a **content-assist that can be called pressing <Ctrl>+<Space>**. It will display all possible valid elements also considering scoping of variables.

In the following we will demonstrate SCT using the famous ABRO example (the hello world of synchronous programming/modelling). We will then give details for modelling other SCCharts language constructs with SCT. For details on their semantics please be referred to our PLDI paper [1].

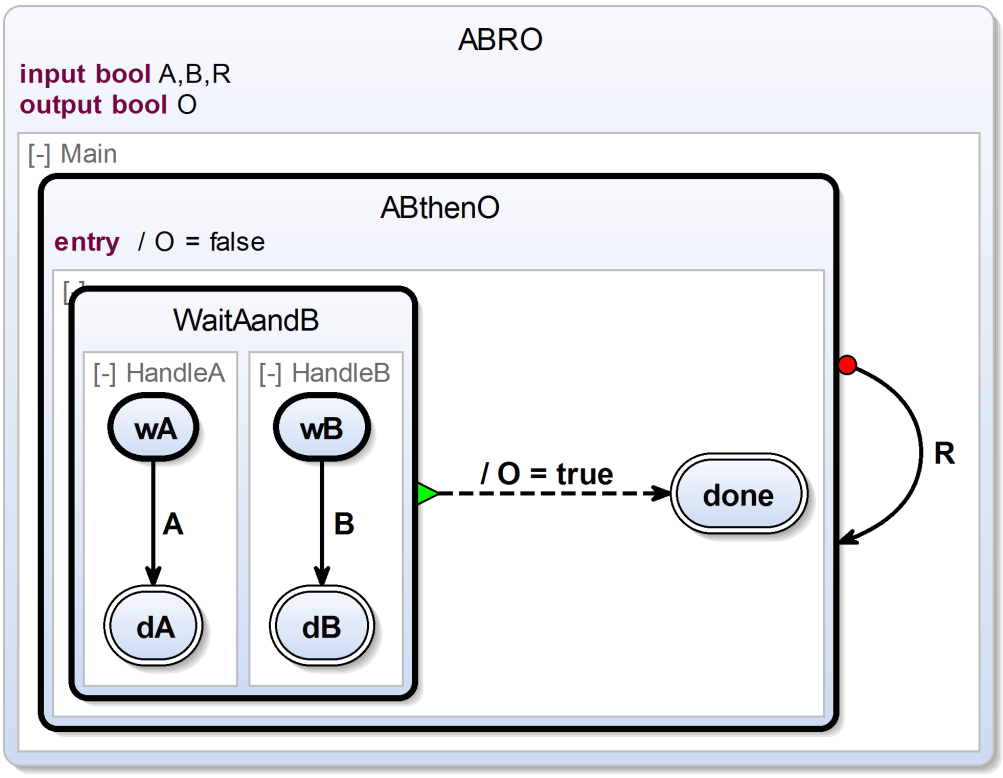
- Overview of Operators

[1] R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. 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. (pdf)

ABRO Example

In the following we will describe some basic elements using the famous ABRO example:

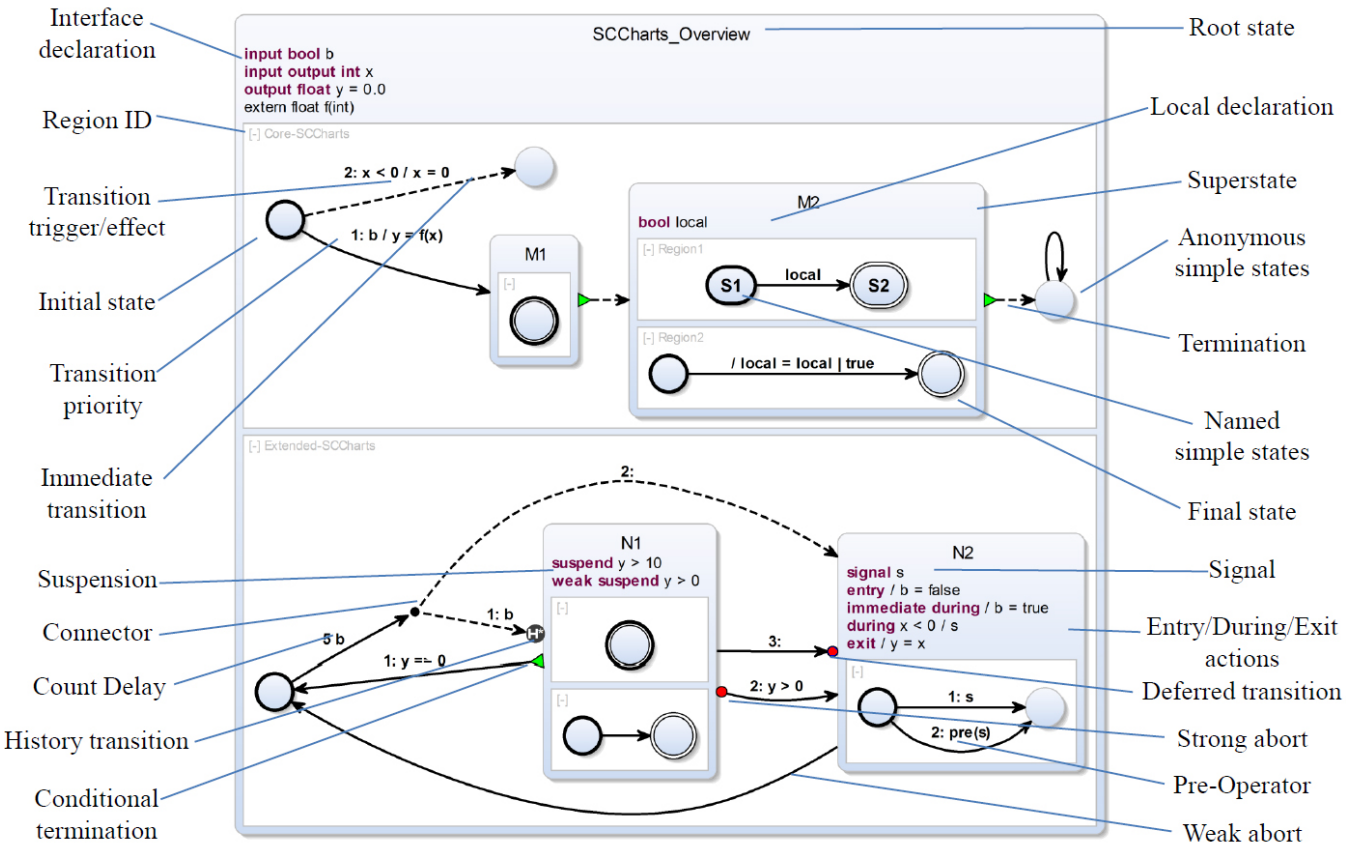
```
@HVLayout
scchart ABRO {
  input bool A, B, R;
  output bool O = false;
  region Main:
  initial state ABO "ABthenO" {
    entry / O = false;
    initial state WaitAandB {
      region HandleA:
      initial state wA
      --> dA with A;
      final state dA;
      region HandleB:
      initial state wB
      --> dB with B;
      final state dB;
    }
    >-> done with / O = true;
    final state done;
  }
  o-> ABO with R;
}
```



1. In the first line you see how an SCChart is defined using the *scchart* keyword where the ID of the SCChart will be *ABRO*. An optional label can be inserted after *ABRO* using "<LABEL>".

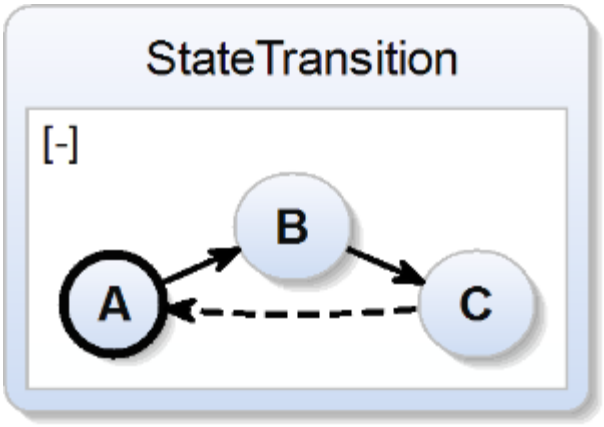
2. In the next three lines variables are declared, namely, *A*, *B*, *R* and *O*, where *O* is initialized with the value false. *A*, *B*, and *R* are inputs which must not be initialized and get there valued from the environment.
3. An SCChart typically contains concurrent regions which are introduced with the keyword *region* as shown in Line 9.
4. Every region must at least have one state, and every region must exactly have one initial state. An initial state *ABO* is defined for region *Main* in Line 6.
5. Every state is terminated by a *;* as shown in line 11 for state *HandleA*.
6. If you like to specify internal behavior of a state, you can add concurrent regions to a state in *{* <regions> *}* as done for state *ABO* or state *WaitAB*.
7. Transitions outgoing from a state must be declared right before a state is terminated with *;*. For example a transition from state *wA* to state *dA* is declared in Line 11.
8. Transitions can have triggers and effects which are separated by a dash: <*trigger*>/<*effects*>. Multiple sequential effects are separated by a *;*. The transition in Line 11 declares just a trigger *A* (a dash is not necessary in this case), while the transition from line 18 declares only an effect *O* = *true* (here the dash is mandatory).
9. There are three types of transitions: 1. normal/weak abort transitions -->, 2. strong abort transitions *o*-> and 3. termination/join transitions >->.

Detailed SCT Syntax of SCCharts Elements



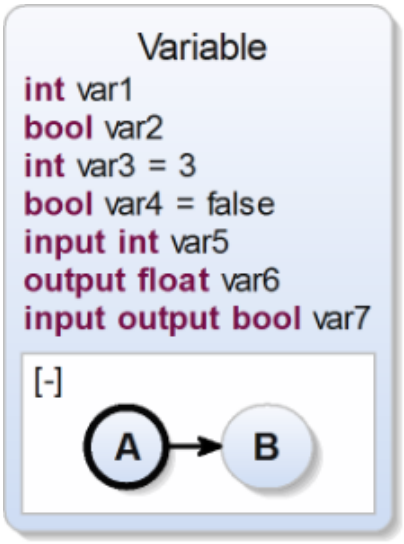
SCChart, Initial State, State, Transition and Immediate Transition

```
 scchart StateTransition {
  initial state A
  --> B;
  state B
  --> C;
  state C
  --> A immediate;
}
```



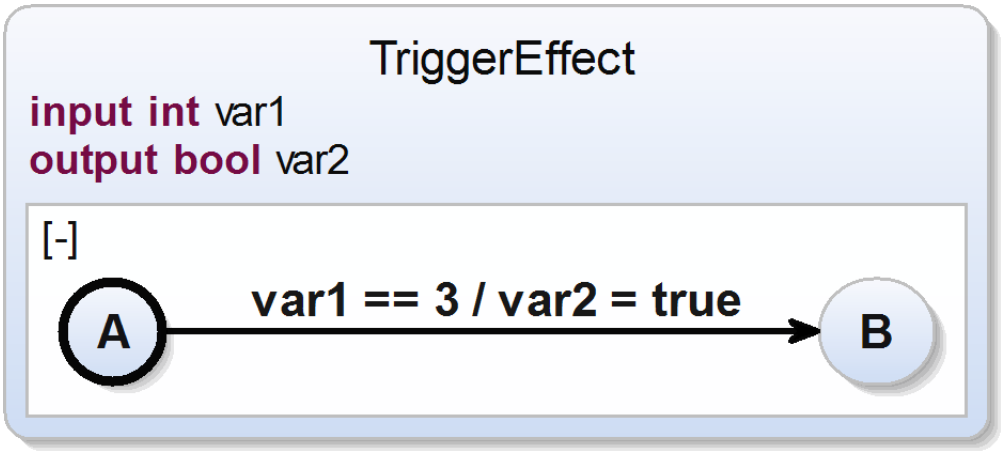
Variable

```
scchart Variable {
  int var1;
  bool var2;
  int var3 = 3;
  bool var4 = false;
  input int var5;
  output float var6;
  input output bool var7;
  initial state A
  --> B;
  state B;
}
```



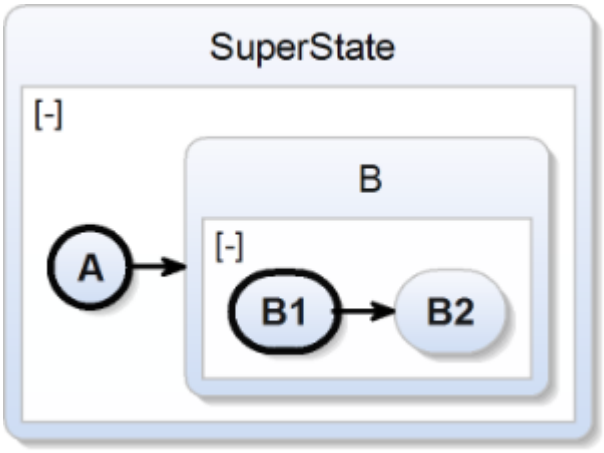
Transition: Trigger & Effect

```
scchart TriggerEffect {
  input int var1;
  output bool var2;
  initial state A
  --> B with var1 == 3 / var2 = true;
  state B;
}
```



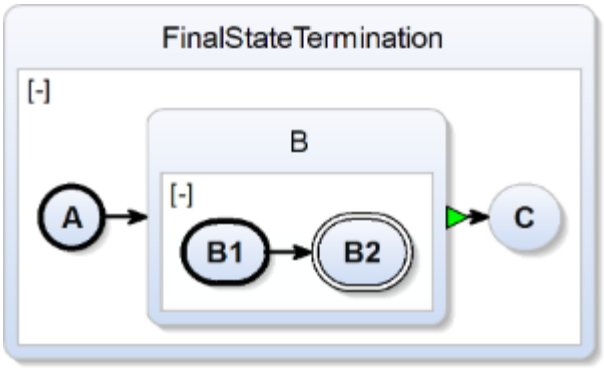
Super State

```
scchart SuperState {
  initial state A
  --> B;
  state B {
    initial state B1
    --> B2;
    state B2;
  };
}
```



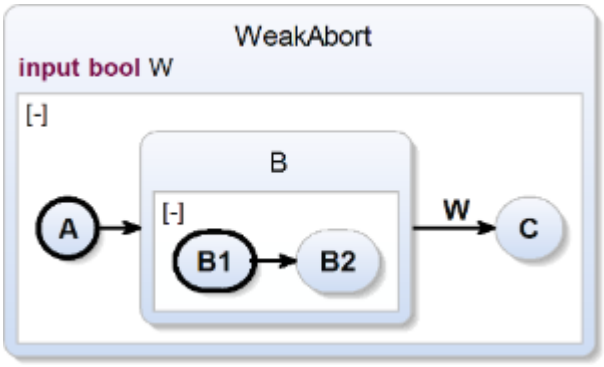
Super State: Final States & Termination Transition

```
scchart FinalStateTermination {
  initial state A
  --> B;
  state B {
    initial state B1
    --> B2;
    final state B2;
  }
  >-> C;
  state C;
}
```



Super State: Weak Abort Transition

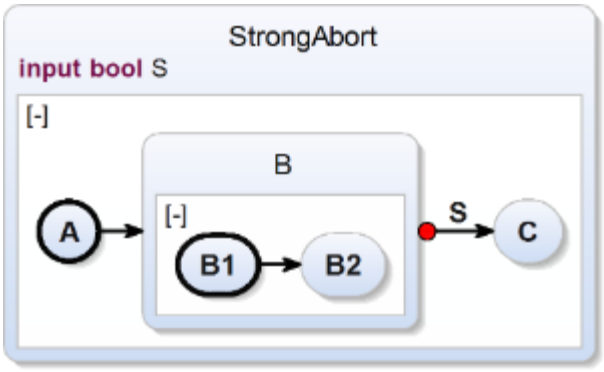
```
scchart WeakAbort {
  input bool W;
  initial state A
  --> B;
  state B {
    initial state B1
    --> B2;
    state B2;
  }
  --> C with W;
  state C;
}
```



Super State: Strong Abort Transition

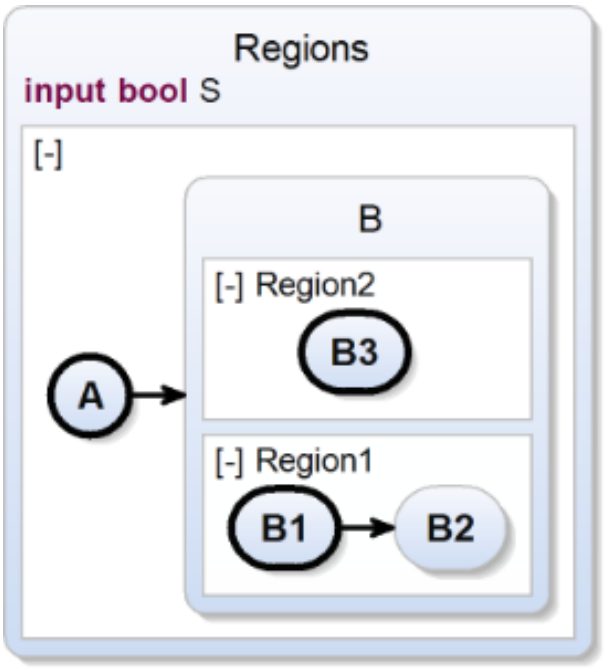
```
scchart StrongAbort {
  input bool S;
  initial state A
  --> B;
  state B {
    initial state B1
    --> B2;
    state B2;
  }
  o-> C with S;

  state C;
}
```



Concurrent Regions (inside a Super State)

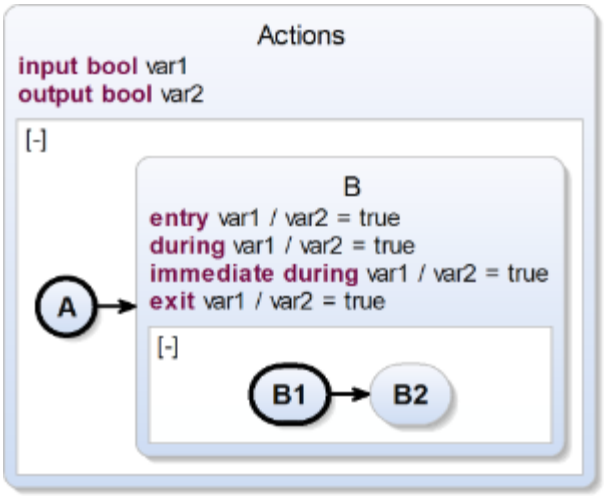
```
scchart Regions {
  input bool S;
  initial state A
  --> B;
  state B {
    region Region1 :
    initial state B1
    --> B2;
    state B2; region Region2 :
    initial state B3;
  };
}
```



Entry Action, During Action, Exit Action

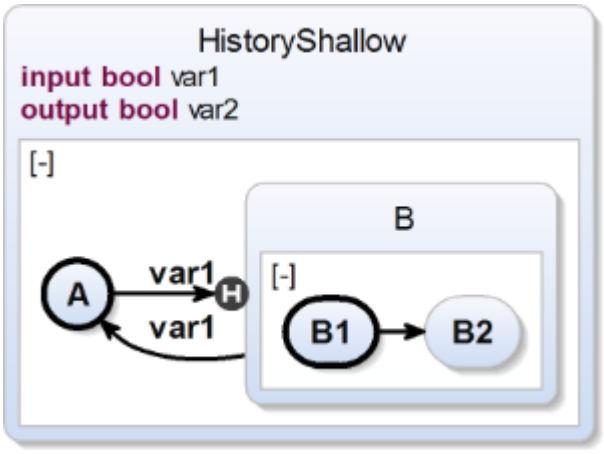
```
scchart Actions {
  input bool var1;
  output bool var2;
  initial state A
  --> B;
  state B {
    entry var1 / var2 = true;
    during var1 / var2 = true;
    immediate during var1 / var2 = true;
    exit var1 / var2 = true;
```

```
    initial state B1
    --> B2;
    state B2;
  };
}
```



Shallow History Transition

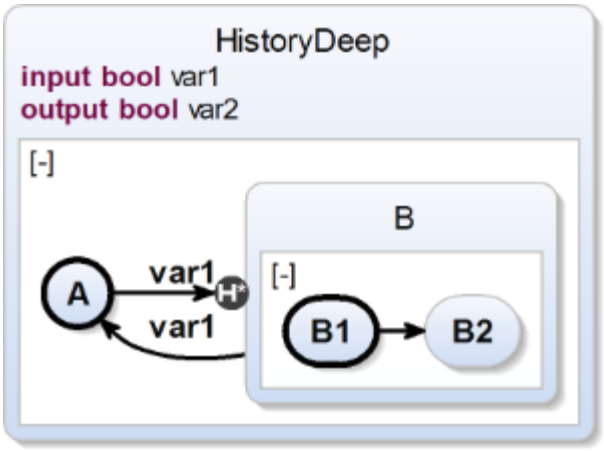
```
scchart HistoryShallow {
  input bool var1;
  output bool var2;
  initial state A
  --> B shallow history with var1;
  state B {
    initial state B1
    --> B2;
    state B2;
  }
  --> A with var1;
}
```



Deep History Transition

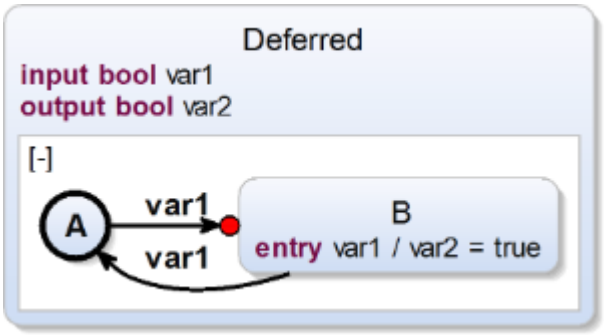
```
scchart HistoryDeep {
  input bool var1;
  output bool var2;
  initial state A
  --> B history with var1;
  state B {
```

```
    initial state B1
    --> B2;
    state B2;
  }
  --> A with var1;
}
```



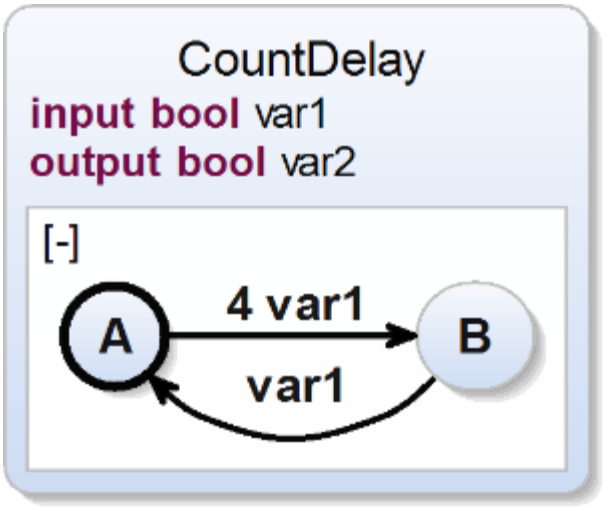
Deferred Transition

```
scchart Deferred {
  input bool var1;
  output bool var2;
  initial state A
  --> B deferred with var1;
  state B {
    entry var1 / var2 = true;
  }
  --> A with var1;
}
```



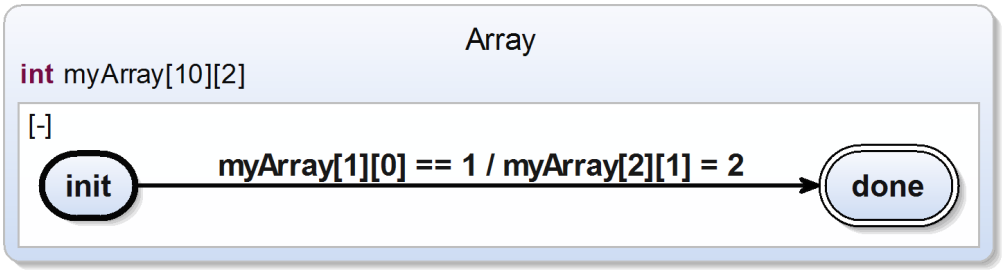
Transition with Count Delay

```
scchart CountDelay {
  input bool var1;
  output bool var2;
  initial state A
  --> B with 4 var1;
  state B
  --> A with var1;
}
```



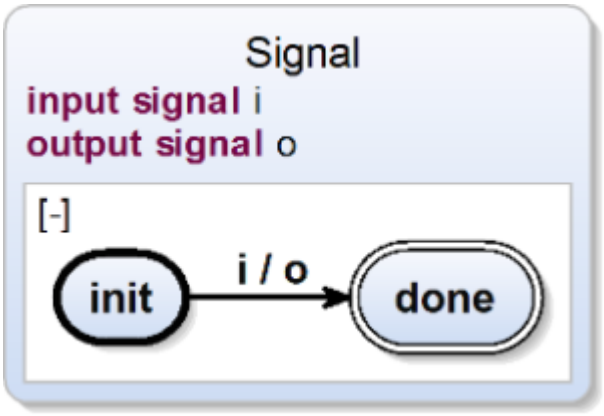
Array

```
scchart Array {
  int myArray[10][2];
    initial state init
  --> done with myArray[1][0] == 1 / myArray[2][1] = 2;
  final state done;
}
```



Signal

```
scchart Signal {
  input signal i;
  output signal o
    initial state init
  --> done with i / o;
  final state done;
}
```



Reference States

Important: To use the referenced SCCharts feature, activate the Xtext nature for your project!

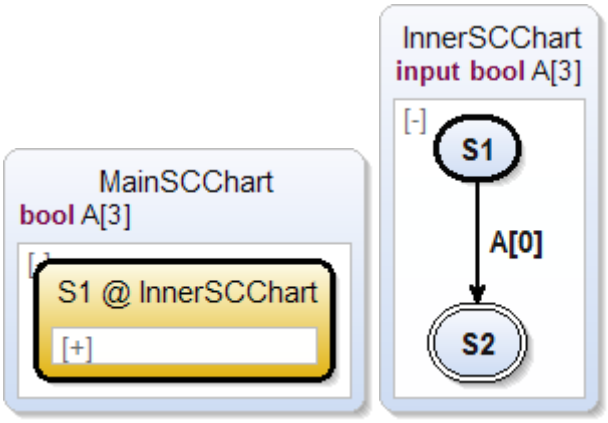
```
scchart MainSCChart {
  bool A[3];

  initial state S1 references InnerSCChart;
}

scchart InnerSCChart  {
  input bool A[3];

  initial state S1
  --> S2 with A[0];

  final state S2;
}
```



Hostcode

You can also use you host language directly.

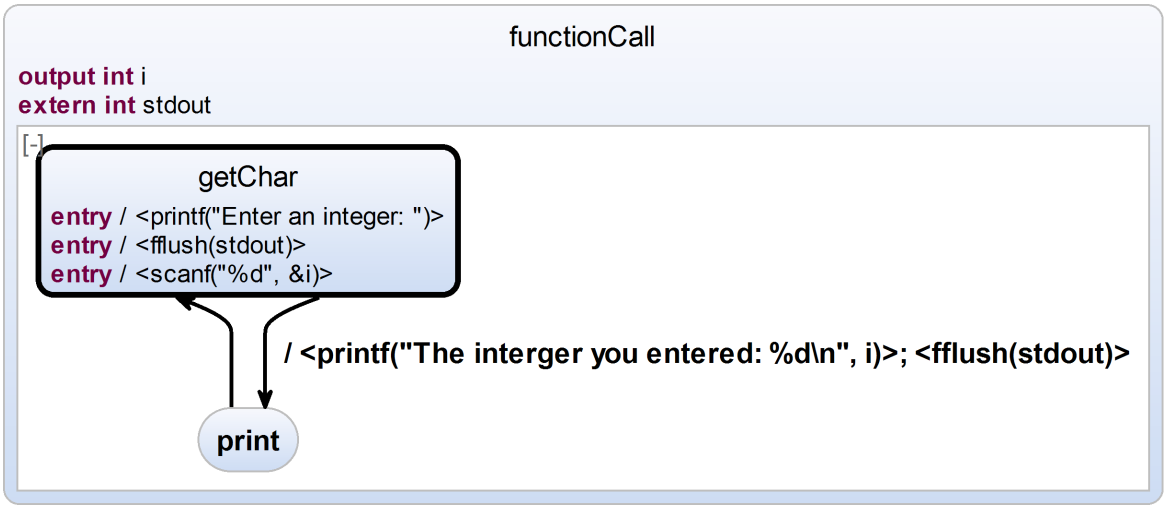
Function Calls

Call host code functions with dependency analysis. Use angle brackets to point to an extern function. You can also declare extern variables.

```
@hostcode "#include <stdio.h>"
@hostcode "#include <stdlib.h>"
scchart functionCall {
  output int i;
  extern int stdout;

  initial state getChar {
    entry / <printf("Enter an integer: ")>;
    entry / <fflush( stdout )>;
    entry / <scanf("%d", &i)>;
  }
  --> print with / <printf("The interger you entered: %
d\\n", i)>;
                                      <fflush( stdout )
>;

  state print
  --> getChar;
}
```



Hostcode (inline)

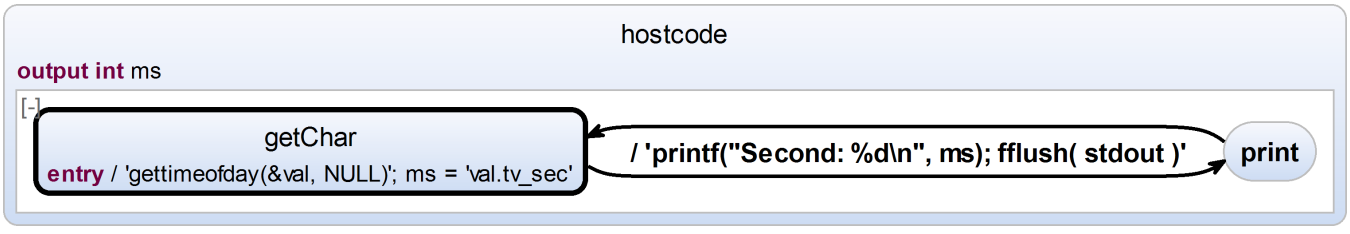
Inline hostcode just as it is. Use the @hostcode annotation to add hostcode line before the tick function. Also use hostcode in single quotes as hostcode effects.

Important: Inline hostcode is not type-safe. Be careful.

```
@hostcode "#include <stdio.h>"
@hostcode "#include <stdlib.h>"
@hostcode "#include <unistd.h>"
@hostcode "#include <sys/time.h>"
@hostcode "struct timeval val;"
scchart hostcode {
  output int ms;

  initial state getChar {
    entry / 'gettimeofday(&val, NULL)';
                    ms = 'val.tv_sec'
  }
  --> print with / 'printf("Second: %d\\n", ms); fflush
( stdout )';

  state print
  --> getChar;
}
```



Annotations

The textual SCCharts language supports several annotations to influence the visual representation of the model.

Annotation are processed in sequential order.

Pattern	Usage	Description	Example

<table>
<tr>
<td><code>@diagram
[<key>]
<value></code></td>
<td>
<table>
<tr><td>Location:</td><td>scchart</td></tr>
<tr><td><key></td><td>The name of the synthesis option. The given name is evaluated case-insensitive and whitespace-ignoring. The options are searched for the first matching prefix.</td></tr>
<tr><td><value></td><td>The value type depends on the option type:

CheckBox: true or false

Choice: Name of choice item

Slider: Float value</td></tr>
</table>
</td>
<td>Sets the synthesis option identified by <key> to the given value.

The available synthesis options for a diagram are displayed in the sidebar of the diagram view.

The values from the sidebar will be ignored if a corresponding annotation is present.</td>
<td><pre>@diagram[paper]
true
scchart Testing {
  initial state A
  --> B;
  final state B;
}</pre></td>
</tr>
<tr>
<td><code>@layout
[<key>]
<value></code></td>
<td>
<table>
<tr><td>Location:</td><td>scchart, state, region, transition</td></tr>
<tr><td><key></td><td>The ID of the layout option. The options are searched for the first matching postfix.</td></tr>
<tr><td><value></td><td>The value type depends on the option type. The value is parsed case-sensitive.</td></tr>
</table>
</td>
<td>Sets the layout property identified by <key> to the given value on the annotated element.

The available layout options are documented here.

Layout options will only affect the annotated element and no underlying hierarchy levels.

If a layout direction is specified with this annotation it overrides the layout direction set by HV-/VH-Layout in any parent element for this element.

Special case: If the direction is set on the scchart element (top level) it overrides the default alternating layout.
<table>
<tr><td><code>direction</code></td><td>Layout direction</td></tr>
<tr><td>priority</td><td>Can influence the order of regions</td></tr>
</table>
</td>
<td><pre>scchart Testing {
    @layout
[algorithm] de.
cau.cs.kieler.
graphviz.circo
    region:
    initial final
state A
    --> B;
    state B
    --> C;
    state C
    --> A;
}</pre>
<pre>scchart Testing {
    @layout
[direction] UP
    region "up":
    initial state
A
    --> B;
    final state B;
    @layout
[direction] LEFT
    region "left":
    initial state
A
    --> B;
    final state B;
}</pre></td>
</tr>
<tr>
<td><code>@HVLayout
@VHLayout</code></td>
<td>
<table>
<tr><td>Location:</td><td>scchart, state, region</td></tr>
</table>
</td>
<td>Defines the order of the alternating layout directions.

The annotation can be mixed and nested in the SCChart and will only affect succeeding hierarchy levels.

The default is an implicit HVLayout starting at the top level state.</td>
<td><pre>@VHLayout
scchart Testing {
  initial state A
  --> B;
  final state B;
}</pre></td>
</tr>
<tr>
<td><code>@collapse
@expand</code></td>
<td>
<table>
<tr><td>Location:</td><td>region</td></tr>
</table>
</td>
<td>The annotated region will be initially collapse or expanded.</td>
<td><pre>scchart Testing {
  @collapse
  region:
  initial state A</pre></td>
</tr>
</table>

			`--> B;`
			`final state B;`
			`}`

`@hide`	**Location:** scchart, state, region, transition	The annotated element will be excluded from the diagram.
		
		Transitions with a hidden source or target state will be hidden as well.
			`scchart Testing {`
			`initial state A`
			`--> B;`
			`@hide`
			`final state B;`
			`}`

Overview of Operators

The following briefly describes the operators that can be used in expressions.

Assignment Operator	Description	Example
=	Assignment operator	x = 42

Arithmetic Operators	Description	Example
+	Addition	2 + 1
-	Subtraction	2 - 1
*	Multiplication	3 * 2
:	Division	6 : 2

Unary Operator	Description	Example
+	Indicate a positive number	+2
-	Negate a number	-2
++	`Increment`	x++
--	Decrement	x--

Boolean Operator	Description	Example
==	Equal to	x == 2
!	Negate a boolean value	! (x == 2)
!=	Not equal to	x != 2
>	Greater than	x > 2
>=	Greather than or equal to	x >= 2
<	Less than	x < 2
<=	Less than or equal to	x <= 2
&&	Conditional-AND	x > 0 && x < 9
||	Conditional-OR	x < 0 || x > 9