X. Extension of Tiger to support hierarchically structured languages

Lately Tiger was extended to support hierarchically structured visual languages by allowing to visually embed symbols inside other symbols that function as containers. This chapter describes the extensions made to Tiger an how to use them. For simple graph-like languages this extension can be simply ignored, since it doesn't change the process of designing such languages.

x.1 Containment

It is now possible to declare certain symbol types a container symbol types. Symbols of these types may then contain other symbols (called content symbols) so that the figure of a content symbol appears inside the figure of the container symbol. Furthermore you can restrict which content types are allowed within which container types by specifying containment relation pairs. The drawing canvases (the white background) of a generated editor and the startgraph editor are considered as the „root container“ which is the root of the containment hierarchy tree.

To achieve that, the *SymbolType* object of the abstract syntax package was extended by two attributes „*containerNode*“ and „*onRootContainer*“ and a reference between *SymbolTypes* establishing a possible containment relation as shown in Fig. 1.

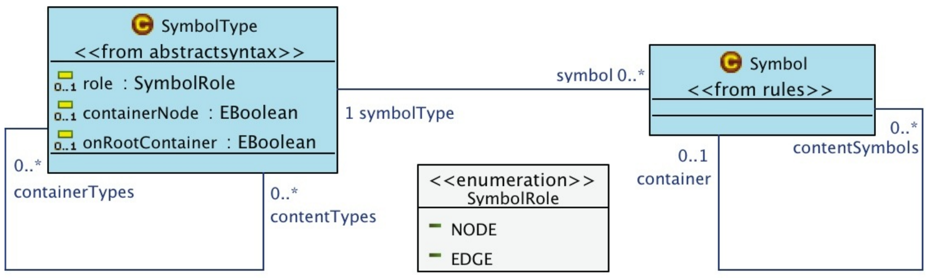


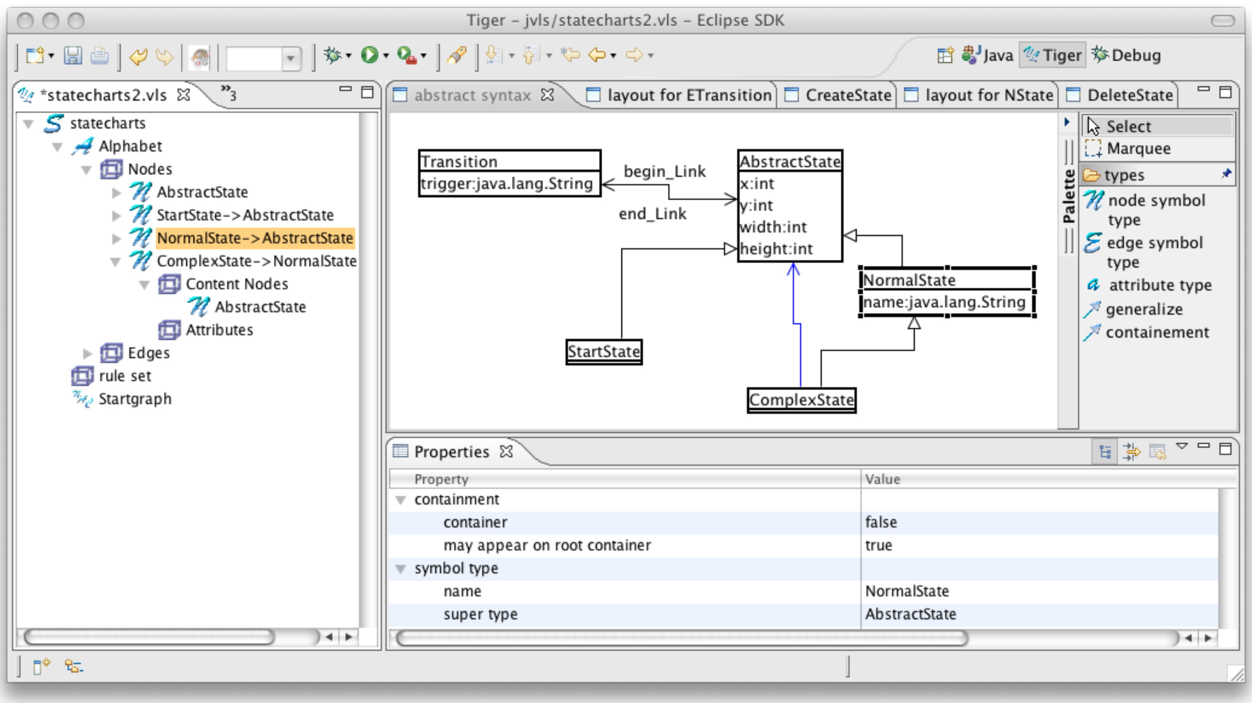
Figure 1: New properties of SymbolTypes an Symbols

The attribute "*containerNode*" states whether nodes of this type are containers and the attribute "*onRootContainer*" declares whether these nodes may be placed directly on the root container (as top level objects). Newly created *SymbolTypes* are not containers but may be placed on the root container by default.

Both these attributes can be set in the properties view of a symbol type selected in the VLSpec tree view or the abstract syntax editor. To declare a containment relation use the "*containment" tool* from abstract syntax editor's palette to draw a blue line between a container symbol type an another symbol type. The possible content types of a container type are also shown in the VLSpec tree view.

The property of being a container type is inherited. Likewise the containment relations are inherited, so that all subtypes of a container type can contain the same content types (a well as their subtypes).

After declaring container types and containment relations you can simply create symbols inside containers by selecting a tool (in the rule and startgraph editors) or a create-rule (in a generated editor) an clicking on a suitable existing container.



Property	Value
containment	
container	false
may appear on root container	true
symbol type	
name	NormalState
super type	AbstractState

Figure 2: new designer elements for node containment

x.2 A containers content pane

Since figures of content symbols appear inside of their container's figure there needs to be a place where the content figures can reside. This place is called the content pane. It is one of the shapes that make up the container's figure. Using the properties view for a selected shape you can declare that shape to be the content pane. This is particularly useful if you want certain parts of a containers figure to not be overlapped by content figures. If no shape is explicitly set to be the content pane , the primary (outermost, first drawn) shape is used for that purpose.

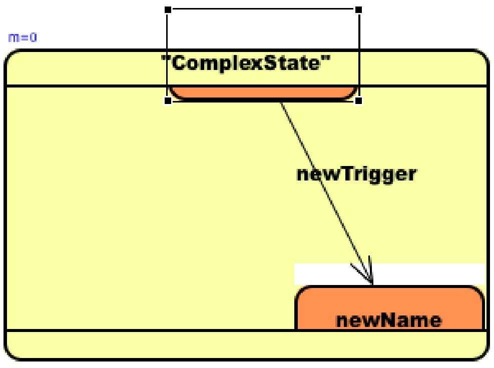


Figure 3: Container with 2 content nodes in the rule editor

Fig. 3 depicts a container symbol with two content symbols in the rule editor. The container's figure consist of a rounded rectangle shape with a border layout. In it there are two thin bars at the top (used as a title bar) and the bottom position . In the center there is a rectangle shape used as the content pane. As you can see the figures of the content symbols are constrained by the bounds of the container's content pane.

Being a normal shape, the content pane has one layout manager arranging it's interior. This holds true for figures of other symbols placed inside the content pane. Using the XYLayout content symbols can be freely positioned and moved around inside their container (provided a move rule exists for the content types). Other layout managers arrange content nodes automatically. The border layout is not supported for content pane shapes, since there is no simple way of specifying one of the five positions it offers and it would be "full" after adding five symbols to the container.

There is a new layout manager, the *ToolbarLayout*, that arranges content nodes in a single row or column. It is described in the next section.

A special ability of the content pane is to be scrollable. If the nodes placed in a container extend the space of the content pane, scrollbars appear and let you choose the visible portion of it's interior. In the properties view of the concrete syntax designer you can choose whether a content pane should be scrollable or not.
Fig. 4 depicts some containers using different layout managers and scrollbars.

When designing container figures be aware of some facts

- in draw2d every shape uses a rectangular area even if just a portion of it is visible. So if you use an ellipse shape as a content pane, the embedded figures will not be visually constrained by the ellipse but by a rectangle of the ellipse's width and height
- in draw2d there are only rectangular borders. If you specify a border for a non-rectangular shape in Tiger an outline will be drawn around the shape. Content figures will paint over this outline (in contrast to a real border on a rectangular shape)

x.3 ToolbarLayout

Tiger supports a new kind of layout manager: the *ToolbarLayout.* It arranges embedded figures either vertically or horizontally (like on an icon toolbar in many desktop applications). As for other layout managers you can specify the spacing between figures using Tiger's layout dialog. You can also specify whether the figures are to be stretched in the major direction (the selected direction: vertically or horizontally) and the minor (the other) direction. Stretching always respects the minimum and maximum size set for a shape.

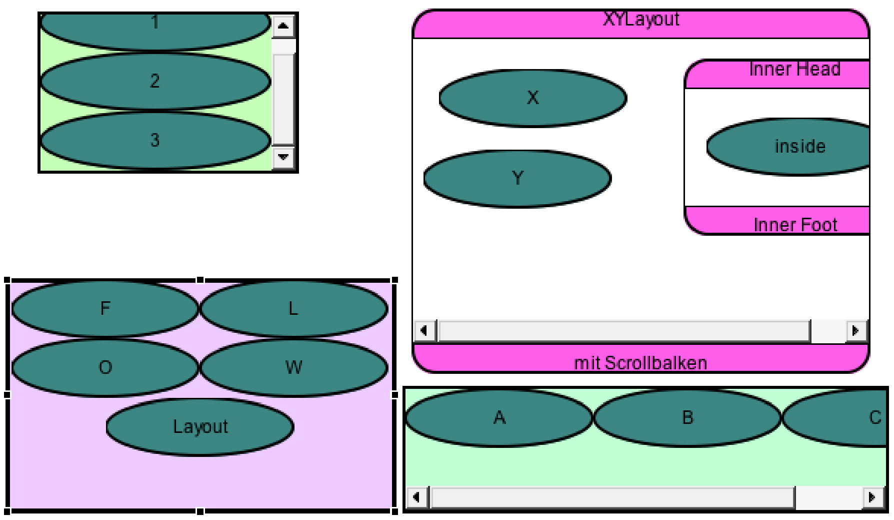


Figure 4: Containers using different layout managers

Fig. 4 shows four container symbols with content symbols arranged using different layout managers: vertical *ToolbarLayout*, *XYLayout, FlowLayout* and horizontal *ToolbarLayout.* You can also see the scrollbars that appear if a container's content becomes too big.

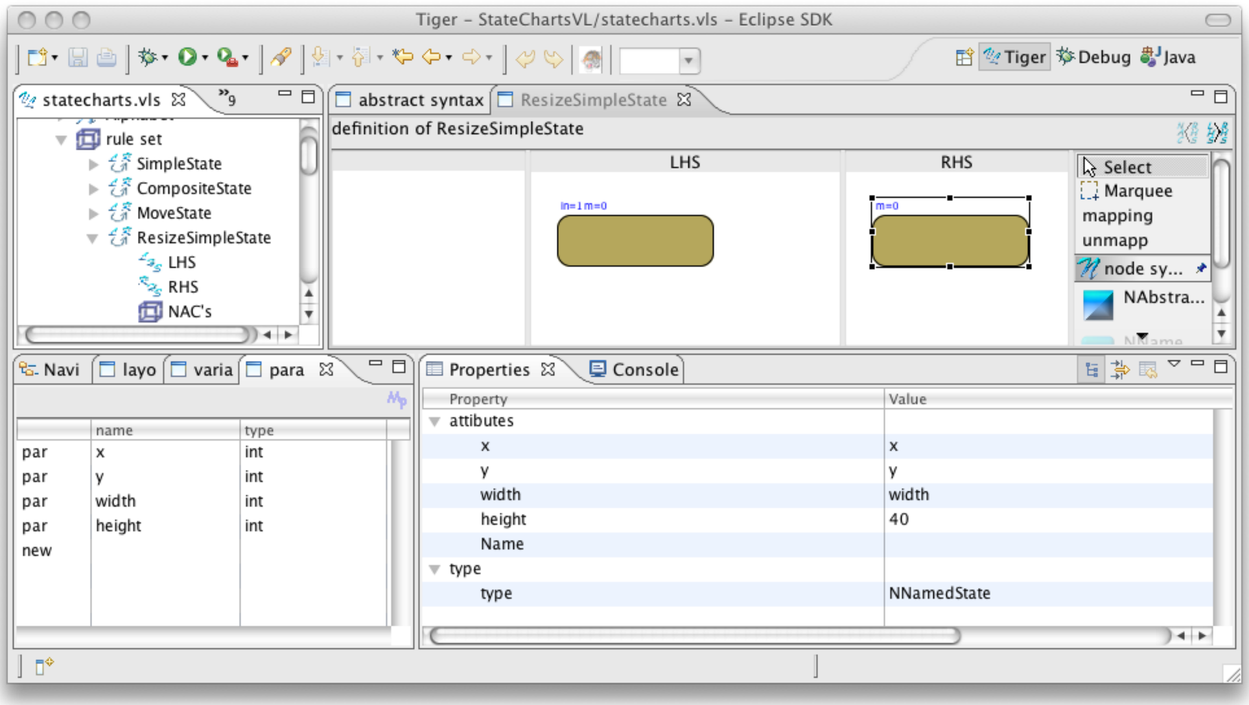
x.4 Resizablility

Tiger was extended to allow altering the size of symbol figures which is especially useful for container symbols because their required size may not be known at design-time. To allow resizing of a figure, new attribute for shapes are introduced that can be set in the properties view when a shape is selected in the concrete syntax editor:

- *resizable* this attribute specifies whether a shape is resizable in principle
- *minimum size* specifies the minimum size of that shape

A symbols figure is resizable if the new attribute *resizable* is *true* on the primary (outermost, first drawn) shape of the figure.

As shown in Fig. 2 (at the symbol type called "AbstractState" in the abstract syntax view) every node symbol now has *width* and *height* attributes that store the size of a symbol's figure in the generated editor. These attribute can be set in the startgraph editor and in rules, especially move rules, just like the attributes *x* and *y.*
In a generated editor resizable symbols have extended resize handles when selected. Dragging one of the points of these handles determines the new size. Note that resizing (as well as moving) only works for top level symbols and such within a container using *XYLayout* for it's content pane.



- rule set
  - SimpleState
  - CompositeState
  - MoveState
  - ResizeSimpleState
    - LHS
    - RHS
    - NAC's

	name	type
par	x	int
par	y	int
par	width	int
par	height	int
new		

Property	Value
attibutes	
x	x
y	y
width	width
height	40
Name	
type	
type	NNamedState

Figure 5: move/resize rule

Fig. 5 shows a typical move/resize rule. There are the 4 required parameters: *x*, *y*, *width* and *height* which are set by the generated editor when moving/resizing a symbol figure. In this example the the height parameter is not applied to the symbol's height attribute so this rule only alters the width of a selected figure.

x.5 Create rules

Create rules work as they did before introducing containers. The only thing new is that you can now create complex hierarchical structures of symbols.

But behind the scenes some things have changed though. Since all symbols in a model are now organized in a containment tree, every symbol except the root container is nested in a container symbol. So when creating a new symbol using a create-rule there needs to be a container to put the new symbol into.

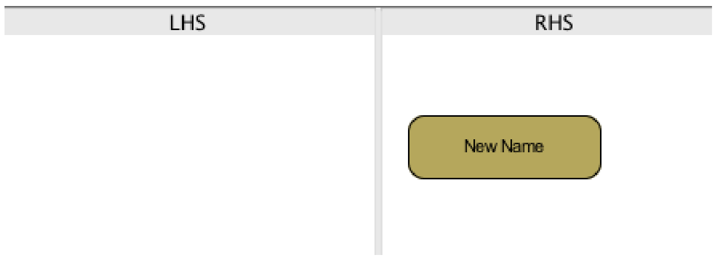


Figure 6: creation rule example 1

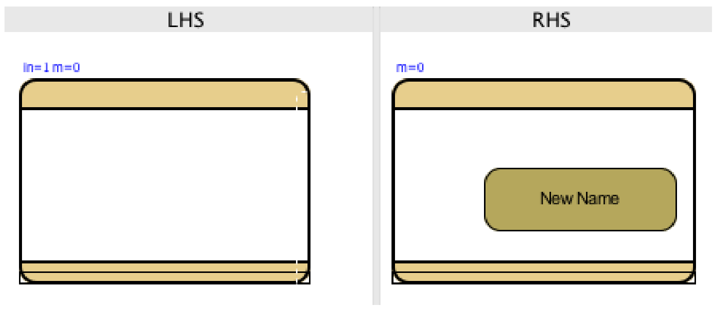


Figure 7: creation rule example 2

There are two ways to achieve that:

1. simply create symbols in the RHS of your rule without a container. (Fig. 6)
2. explicitly include a container symbol in the LHS and RHS of the rule and add the symbol(s) to be created to it in the RHS. Declare the container as an input parameter symbol. (Fig. 7)

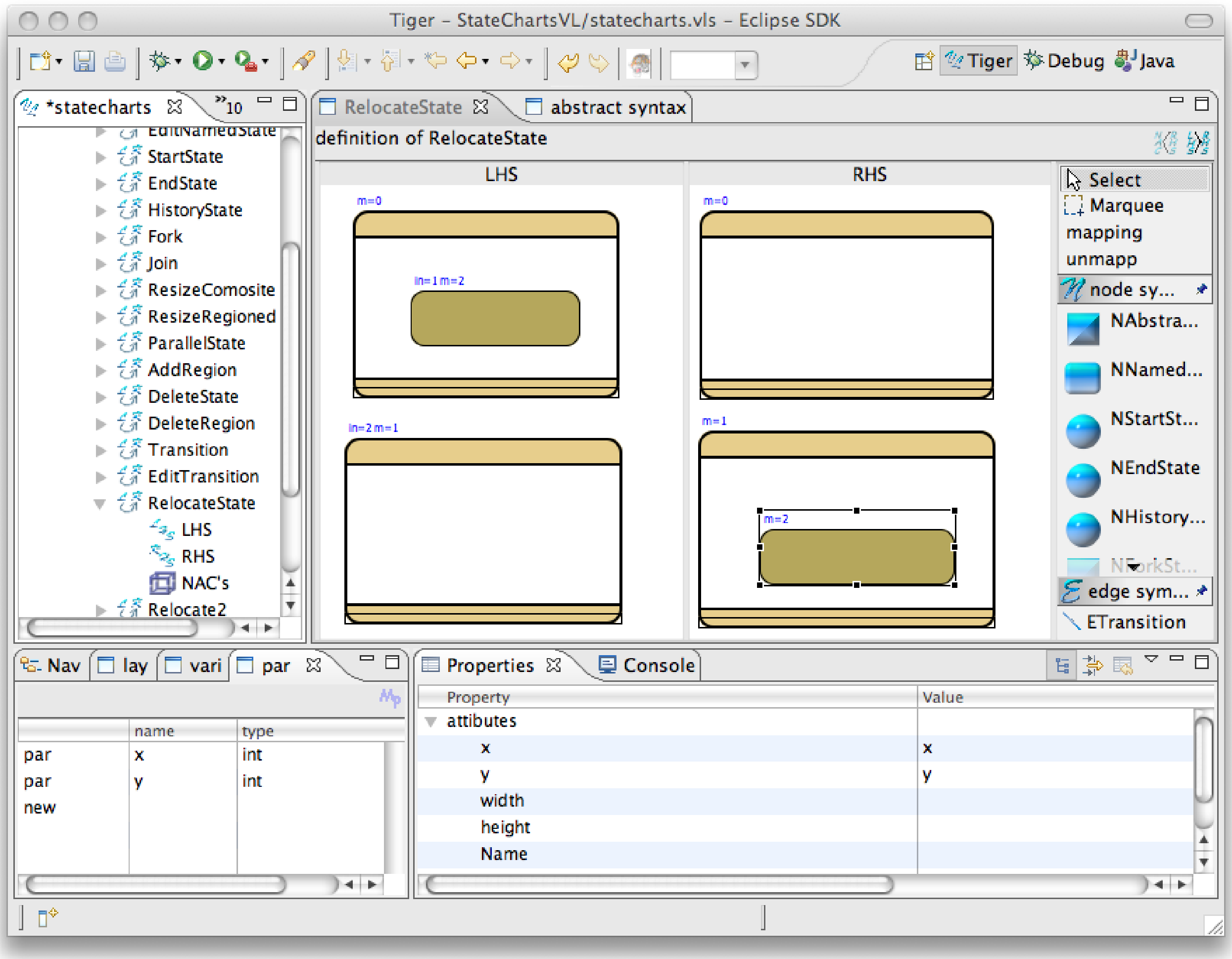
The second way is quite simple. In the generated editor you apply the rule by clicking on a container symbol of the type given in the rule's LHS. A disadvantage besides more work designing the rule is that you need to create such rules for every container type you want to create a certain symbol in (though a rule will work for subtypes of the container type used).

In the first case the rule is extended by the Tiger Generator. It adds an internal abstract container symbol that can match to any suitable existing container when applying the rule. If there are no input parameter symbols in the LHS this abstract container is matched to the symbol you click on when applying the rule. That may also be the root container if allowed. If there are input parameter symbols in the LHS one of all suitable containers is (pseudo-indeterministically) chosen to house the new symbols. This way allows you to not think about containers at all, if you don't need them e.g. for typical graph-based languages like Petri Nets.

x.6 Relocate Rules

A Relocate Rule allow the user of a generated editor to drag a symbol from one container to another. This cannot be done with move-rules. A typical relocate-rule is shown in Fig. 8. It has three symbols, two containers and one content symbol, on the LHS and the same symbols on the RHS. The only difference is that the content symbol is located in the first (source) container in the LHS but in the second (target) container in the RHS. The rule uses two input parameter symbols:

1. the content symbol to be relocated. It is matched to the symbol being dragged when applying the rule
2. the target container: it matches to the symbol that the content symbol is dropped onto



	name	type
par	x	int
par	y	int
new		

Property	Value
attibutes	
x	x
y	y
width	
height	
Name	

Figure 8: relocate rule

If the rule includes the two input parameters named *x* and *y* the generated editor will provide the drop-position relative to the target container. These parameters should be applied to the symbol's attributes *x* and *y*.

You can even omit the source container in relocate rules. This way symbols can be dragged from any container (e.g. the root container) to the target container.

[Note: there is no way to relocate a symbol from a container to the root container, yet]

x.7 Iconification

For container symbol types an alternative figure can be designed to serve as an icon. To create an icon figure the new attribute *iconifiable* must be set to *true* in the properties-view of a node symbol type. Then the layout editor for the icon figure can be opened by choosing the command '*show icon layout*' in the context menu for the symbol type in the *VLSpec* tree view. The icon layout will also be shown in the layout container view.

In a generated editor the context menu of an iconifiable container shows the entry '*(de)iconify'* that switches between the normal and the icon figure of the selected container. When iconified the content symbols of a container wont be visible.

x.8 Content pages

The contents of a container symbol can be displayed in an additional page in a generated editor.

That is useful if the substructure of a container is too complex to be displayed in the main diagram, especially in conjunction with iconifiable containers. To enable additional pages for a certain container type set the property '*has own page*' to true. In a generated editor the context menu of the container will the hold the menu actions '*Show content page'* respectively '*Close content page*'. At the bottom of a generated editor you will see tabs for the different pages currently open.

x.9 Simple rule creation

When designing a visual language using Tiger a lot of time is spend in creating rules for creating, editing and moving/resizing symbols. To ease that process Tiger can generate simple, working rules that can be used as-is or as a starting point for more complex rules. To generate such rules use one of the following actions from the context menu of a node or edge symbol type in the VLSpec tree view:

- '*create simple create rule*' generates a create rule for node symbol types.
- '*create simple connection rule*' generates a create rule for edge symbol types. The generated rule will have two node symbols in the LHS of the types specified for the links of the connection. Those node symbols are mapped to the RHS and there connected by the edge symbol to be created.
- '*create simple edit rule*' generates a rule for editing a node or connection symbol.
- '*create simple delete rule*' generates a rule for deleting a node or connection symbol.
- '*create simple move/resize rule*' generates a rule for moving and possibly resizing a node symbol, depending on whether the primary shape is declared resizable.