

# Language Specification (Appendix A)

Deliverable 2.2

Owners	Alessandra Cavarra and Jim Davies
Approvers	Alan Hartman, Ian Craggs, Laurent Mounier, Klaudia Dussa-Zieger
Status	public
Date	08/08/2001

AGEDIS

---

# Contents

<b>A State Diagrams: abstract syntax</b>	<b>3</b>
A.1 States and transitions . . . . .	3
A.1.1 Start states . . . . .	4
A.1.2 Finish states . . . . .	4
A.1.3 Simple states . . . . .	4
A.1.4 Sequential composite states . . . . .	5
A.1.5 Concurrent composite states . . . . .	5
A.1.6 Transitions . . . . .	5
A.2 Diagrams . . . . .	6
A.2.1 Closed under reference . . . . .	6
A.2.2 Start states, top states, and connectivity . . . . .	7
A.2.3 Composite States . . . . .	8
A.2.4 Well-formed diagrams . . . . .	9

# A State Diagrams: abstract syntax

The informal explanation of the graphical syntax should be adequate for users of the modelling language, but a more formal explanation is required within the AGEDIS project itself: we must be able to determine whether or not any particular combination of boxes, lines, and text comprises a valid diagram—we must be certain of the grammar of our language. We will use the Z notation to produce a mathematical representation of state diagrams.

We regard events, actions, and guards as elements of given sets:

$$[Event, Action, Guard]$$

We recognise that there is a special set of events, corresponding to the stereotype  $\langle\langle create \rangle\rangle$ .

$$| \quad createEvents : \mathbb{P} Event$$

Each of these sets contains a special, null element.

Each state diagram will comprise a collection of states, and a collection of linking transitions. We will represent each of these collections as a function, from identifiers (or references) to state or transition objects. No object can be both a state and a transition, so it makes sense to introduce two sets of identifiers:

$$[Transition^{\dagger}, State^{\dagger}]$$

To distinguish between the different types of state, we introduce an enumerated type:

$$StateType ::= start | finish | simple | sequential | concurrent$$

## A.1 States and transitions

A state has several components:

$\begin{array}{l} \textit{State} \\ \textit{type} : StateType \\ \textit{contents} : \mathbb{P} State^{\dagger} \\ \textit{incoming}, \textit{outgoing} : \mathbb{P} Transition^{\dagger} \\ \textit{deferred} : \mathbb{P} Event \\ \textit{internal} : Event \leftrightarrow Action \\ \textit{entry}, \textit{exit} : Action \end{array}$
--

It may contain other states. We make *contents* a set of references, and not a set of state values, because composite states do not guarantee encapsulation. Transitions may refer to, or depend upon, states within states: the composite state mechanism in StateCharts is not compositional.

It may be associated with any number of incoming and outgoing transitions; as these are shared with other state objects, they are represented by sets of references. It may also be associated with a set of deferred events, a collection of internal transitions, an entry action, and an exit action.

In a well-formed diagram, every state must be one of the following: a start state, a finish state, a simple state, a sequential composite state, or a concurrent composite state.

### A.1.1 Start states

A start state can have no actions, internal transitions, or incoming transitions. It must have exactly one outgoing transition:

<i>StartState</i>
<i>State</i>
<i>type = start</i> <i>contents = ∅</i> <i>incoming = ∅</i> <i>#outgoing = 1</i> <i>deferred = ∅</i> <i>internal = ∅</i> <i>entry = null</i> <i>exit = null</i>

### A.1.2 Finish states

A finish state can have no actions, internal transitions, or outgoing transitions. It can have any number of incoming transitions:

<i>FinishState</i>
<i>State</i>
<i>type = finish</i> <i>contents = ∅</i> <i>outgoing = ∅</i> <i>deferred = ∅</i> <i>internal = ∅</i> <i>entry = null</i> <i>exit = null</i>

### A.1.3 Simple states

A simple state has no contents, but there are no constraints upon the other attributes:

<i>SimpleState</i>
<i>State</i>
<i>type = simple</i>
<i>contents = ∅</i>

### A.1.4 Sequential composite states

A sequential composite state has at least one state inside it. It can also have its own actions and internal transitions:

<i>SequentialState</i>
<i>State</i>
<i>type = sequential</i>
<i>#contents ≥ 1</i>

### A.1.5 Concurrent composite states

The same is true of a concurrent composite state, although here there must be at least two states inside:

<i>ConcurrentState</i>
<i>State</i>
<i>type = concurrent</i>
<i>#contents ≥ 2</i>

The set of all valid states is then given by

$$\text{ValidState} \hat{=} \text{StartState} \vee \text{FinishState} \vee \text{SimpleState} \vee \text{SequentialState} \vee \text{ConcurrentState}$$

### A.1.6 Transitions

A transition object has references to two states—its source and its target—and three value components:

<i>Transition</i>
<i>source, target : State</i> <sup>1</sup>
<i>event : Event</i>
<i>guard : Guard</i>
<i>action : Action</i>

There are no additional constraints on transitions: they can be annotated with any combination of guard, action, and event.

## A.2 Diagrams

A diagram comprises a collection of indexed, valid states, a collection of indexed transitions, and a single index *top*, used to identify the top-level state:

<i>Diagram</i> $state : State^1 \rightarrow State$ $transition : Transition^1 \rightarrow Transition$ $top : State^1$
$top \in \text{dom } state$ $\text{ran } state \subseteq ValidState$

### A.2.1 Closed under reference

Our first constraints are applicable only to the textual syntax, and concern the use of references to express the relationships between state objects and transition objects.

Every reference must be to a state or transition object in the current diagram:

<i>ClosedUnderReference</i> <i>Diagram</i>
$\forall Transition \mid \theta Transition \in \text{ran } transition \bullet$ $\{source, target\} \subseteq \text{dom } state$
$\forall State \mid \theta State \in \text{ran } state \bullet$ $incoming \cup outgoing \subseteq \text{dom } transition \wedge$ $contents \subseteq \text{dom } state$

For any ( $\forall$ ) collection of variables corresponding to a transition object (*Transition*) such that this combination of variables and values—this object—( $\theta Transition$ ) is one of the objects in the diagram ( $\in \text{dom } state$ ), the set of references  $\{source, target\}$  is a subset of the set of references associated with states in the current diagram.

For any state object in the diagram ( $\forall State \mid \theta State \in \text{ran } state$ ), every reference in the set union  $incoming \cup outgoing$  must be a reference to a transition in the current diagram ( $\subseteq \text{dom } transition$ ), and every reference in *contents* must be a reference to a state in the current diagram ( $\subseteq \text{dom } state$ ).

Furthermore, these references must be used consistently:

<i>ReferencesConsistent</i>
<i>Diagram</i>
$\forall s : State^1; t : Transition^1 \bullet$ $(\exists State; Transition \mid \theta State = state\ s \wedge \theta Transition = transition\ t \bullet$ $source = s \Leftrightarrow t \in outgoing \wedge$ $target = s \Leftrightarrow t \in incoming)$

For every pair of references  $s$  and  $t$ , if we let *State* declare the variables of the state object referenced by  $s$  and *Transition* declare the variables of the transition object referenced by  $t$ , then  $s$  is the *source* if and only if  $t$  is one of the outgoing transitions, and  $s$  is the target if and only if  $t$  is one of the incoming transitions.

Notice how we use the schemas to create a local scope in which the variable names are meaningful. An alternative form of the constraint would be

$$\forall s : State^1; t : Transition^1 \bullet$$

$$(transition\ t).source = s \Leftrightarrow t \in (state\ s).outgoing \wedge$$

$$(transition\ t).target = s \Leftrightarrow t \in (state\ s).incoming$$

## A.2.2 Start states, top states, and connectivity

To express the remaining constraints upon diagrams, we need to make some pieces of information more explicit.

<i>DiagramStartStates</i>
<i>Diagram</i>
$topState : State$
$topContents, startStates : \mathbb{P}\ State^1$
$topState = state\ top$
$topContents = topState.contents$
$startStates = \text{dom}(state \triangleright StartState)$

The top state object can be obtained by applying the dereference (or indexing) function *state* to the reference *top*. The set *startstates* denotes the set of all references to start states.

The top state is a sequential composite state with exactly one start state.

<i>TopStateSequential</i>
<i>DiagramStartStates</i>
$topState \in SequentialState$
$\#(startStates \cap topContents) = 1$

Every sequential composite state has at most one start state:

$\begin{array}{l} \textit{SingleStartStates} \\ \textit{DiagramStartStates} \\ \forall \textit{SequentialState} \mid \theta \textit{State} \in \textit{ran state} \bullet \\ \#(\textit{contents} \cap \textit{startStates}) \leq 1 \end{array}$
--

and no transition leading from a start state has an event, except for the one in the top state, which may be triggered by a create event:

$\begin{array}{l} \textit{StartEventsNull} \\ \textit{DiagramStartStates} \\ \forall \textit{Transition} \mid \theta \textit{Transition} \in \textit{ran transition} \bullet \\ \textit{source} \in \textit{startStates} \setminus \textit{topContents} \Rightarrow \textit{event} = \textit{null} \wedge \\ \textit{source} \in \textit{startStates} \cap \textit{topContents} \Rightarrow \textit{event} \in \textit{createEvents} \end{array}$
---

### A.2.3 Composite States

The remaining syntactic constraints concern the hierarchy of composite states. We may define two infix relations—*parentOf* and its inverse, *childOf*—from the *contents* attributes:

$\begin{array}{l} \textit{DiagramCompositeStates} \\ \textit{Diagram} \\ \_ \textit{parentOf} \_ : \textit{State}^! \leftrightarrow \textit{State}^! \\ \_ \textit{childOf} \_ : \textit{State}^! \leftrightarrow \textit{State}^! \\ \textit{descendants} : \textit{State}^! \leftrightarrow \mathbb{P} \textit{State}^! \\ \forall s1, s2 : \textit{dom state} \bullet \\ s1 \textit{parentOf} s2 \Leftrightarrow s2 \in (\textit{state} s1).\textit{contents} \\ (\_ \textit{childOf} \_) = (\_ \textit{parentOf} \_)^{\sim} \\ \forall s1 : \textit{dom state} \bullet \\ \textit{descendants} s1 = (\_ \textit{parentOf} \_)^* (\{s1\}) \end{array}$
---

The reflexive transitive closure  $(\_ \textit{parentOf} \_)^*$  associates each state reference with (itself and) each of its descendants; to obtain the set of all descendants, we take the relational image of a reference under this closure.

Everything referenced has a unique ancestor, except for *top*:

$\begin{array}{l} \textit{UniqueAncestor} \\ \textit{DiagramCompositeStates} \\ (\_ \textit{childOf} \_) \in (\textit{dom state} \setminus \{\textit{top}\}) \rightarrow \textit{dom state} \end{array}$
--

The assertion that *childOf* is not merely a total function  $\rightarrow$ , but is also surjective  $\rightarrow$  repeats our assertion that every element of *contents* must be a reference to a state in the current diagram.

Everything immediately inside a concurrent composite state is a sequential composite state:

$$\begin{array}{l}
 \textit{SequentialInsideConcurrent} \\
 \textit{DiagramCompositeStates} \\
 \hline
 \forall \textit{ConcurrentState} \mid \theta \textit{State} \in \textit{ran state} \bullet \\
 \textit{state}(\textit{contents}) \subseteq \textit{SequentialState}
 \end{array}$$

Our final, most complex constraint concerns transitions that cross boundaries into states within (sequential composite states within) concurrent composite states. Whenever this happens, every other sequential composite state within the enclosing concurrent state must have a start state.

$$\begin{array}{l}
 \textit{StartOtherRegions} \\
 \textit{DiagramStartStates} \\
 \textit{DiagramCompositeStates} \\
 \hline
 \forall \textit{ConcurrentState} \mid \theta \textit{State} \in \textit{ran state} \bullet \\
 \forall \textit{Transition} \mid \theta \textit{Transition} \in \textit{ran transition} \bullet \\
 \forall s1, s2 : \textit{contents} \bullet \\
 \textit{target} \in \textit{descendants } s1 \wedge \textit{source} \notin \textit{descendants } s1 \Rightarrow \\
 (\textit{state } s2).\textit{contents} \cap \textit{startStates} \neq \emptyset
 \end{array}$$

#### A.2.4 Well-formed diagrams

The set of all valid state diagrams is then given by the following schema:

$$\begin{array}{l}
 \textit{ValidStateDiagram} \hat{=} \\
 \textit{ClosedUnderReference} \wedge \textit{ReferencesConsistent} \wedge \\
 \textit{TopStateSequential} \wedge \textit{SingleStartStates} \wedge \\
 \textit{StartEventsNull} \wedge \textit{UniqueAncestor} \wedge \\
 \textit{SequentialInsideConcurrent} \wedge \textit{StartOtherRegions}
 \end{array}$$