



Event Parameter Constraints

Version 1.0

August 8, 2003

PathMATE Technical Notes

Pathfinder Solutions LLC
33 Commercial Drive, Suite 2
Foxboro, MA 02035 USA
www.PathfinderMDA.com
888-662-7284

Table Of Contents

- 1. Introduction..... 1
- 2. Different Events to Same State 1
- 3. Composite States 1
- 4. Composite States with History 1
- 5. Constraint Checking 2

1. Introduction

This Technical Note is an addendum to the Statechart Execution Semantics Technical Note. It describes the event parameter constraints on events into composite states and composite states with history. These constraints are intended to be as open and flexible as possible, while still insuring correct and consistent models.

This document supercedes the constraints described in the Statechart Execution Semantics Technical Note in the areas described.

2. Different Events to Same State

Multiple events can cause a statechart to enter a new state. The event parameters are used as inputs to the state's entry action. If transitions into a state have different events, then the parameters of the events' signatures must match in name, type, and order.

This is an existing constraint.

3. Composite States

Entry actions of composite states may access event parameters. If an event goes to the composite state, the entry action of that state may access the events parameters. As a side effect, these parameters must be available every time the entry action is executed. Therefore, *every event to a nested state of the composite state must also carry parameters from the event into the composite state that match in name and type (but not order)*. These parameters will be passed to the composite state's entry action.

It is the transition logic method that pulls the necessary parameters from the incoming event and passes them as necessary to the composite state entry actions.

If a composite state has no events to it, then its entry action cannot access parameters.

4. Composite States with History

When the flow of control returns to a state through a history state, the state's entry action (and all of its parent composite states, if deep history is used) is executed. If the entry actions of the nested states require event parameters, as shown by events on explicit transitions to the state, then the history state must provide those parameters.

Since an event into a history state can result in an implicit transition to any nested state, the history state must be able to provide parameters to all nested states. Therefore, *events to the history state must carry the union of all event parameters into the composite state containing the history state*. The generated entry action for the history state will then distribute the parameters, based on name, to the appropriate entry actions.

For example, if a composite state S with history, has events A and B to nested states within S, and A has parameter Integer P1 and B has parameter Real P2, then, event H to the history state must carry both parameters Integer P1 and Real P2.

This constraint opens up state modeling to be as open and flexible as possible, while constraining only the events to the history pseudo-state.

5. Constraint Checking

As a side effect of sections 3 and 4, an additional constraint on the naming of event parameters is introduced. *Event parameters of a class (or it's supertypes) cannot have the same name but different types, even if they are on different events.*

All these constraints are checked at translation time by the templates. If a constraint fails, an error message will be injected into the generated code that will cause the module to fail to compile. Once templates have the ability to produce Springboard error messages, the constraint checking will use this method of reporting.