Scope of elements

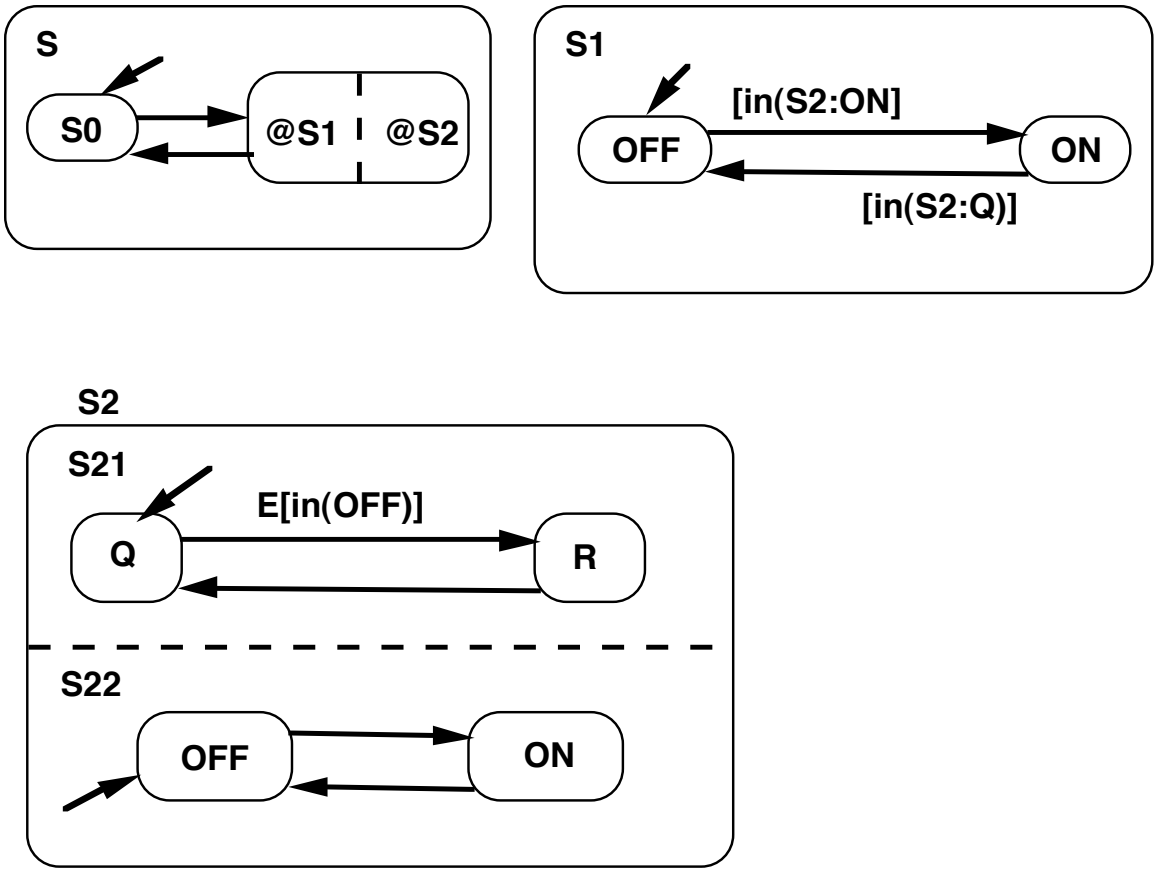
In large projects, there are many components. Most of the information required in a component is usually local to that component and not relevant elsewhere. On the other hand, some information clearly must be shared on a wider basis if different components are to be coordinated. To the extent that it is feasible, it is useful to avoid having irrelevant details in one component interfere with those in another. Hence, issues of scope are important.

The **scope** of an element is a set of charts in which the element is known and can be used. Scoping rules relate the place an element is defined and where it is visible. The specific rules for visibility and the resolution of reference elements differ for different types of elements.

Charts themselves are given names, and these names must be unique and are global. Graphical elements — boxes, arrows, and connectors — are defined in the chart in which they are drawn. Arrows have no names and cannot be referred to in other charts. The names of boxes must be unique among its siblings boxes.

States may be referred to (e.g., events and conditions) in any state that belongs to the same *logical* statechart. States in other pages are preceded by the appropriate chart name (e.g., chart-name:state-name).

Example



In statechart S2, the state OFF referred to in the label E[in(OFF)]is understood to be the instance in state S22 since it appears in the same statechart, even though there is another state of the same name. However, in the statechart S!, the condition in(S2:ON) refers not to the state of that name in S1, but to the state in chart S2.

Scopes for textual elements — events, variables, and actions — are associated with a defining chart with a separate notation (kept in the Data Dictionary by Rhapsody). Visibility extends throughout the defining chart and all its descendents.

The top level

Since the overall organization is hierarchical, there will always be a “root” chart. This provides a natural location for the incorporation of global facilities.

Of course, any practical system will include built-in facilitates and libraries of supporting definitions. Also, the ability to animate and test these reactive models is critical, and statecharts are amenable not only to prototyping, but to automated code generation from the specification. Two highly regarded systems or prototyping statecharts are Rhapsody by I-Logix and Rational’s UML system, Rose.