State Design Pattern [Gamma et al]

This pattern deals with the runtime modification of behavior or state of object. The behavior is basically a state class in the pattern hierarchy. It advocates creating object-oriented state machines, where the functionality of an object changes fundamentally according to its state. Usually the conditions that trigger alteration are business rules that drive the functionality and subject the object to change its state. Below are the variants of State Pattern:

State Machine Pattern [67]

State Machine pattern improves State and inherits its main idea to encapsulate the state-dependent behavior in a separate class. While using State Machine it is possible to design state classes independently. Thus the same state class could be used in several automata. This eliminates the major disadvantage of State reuse issues. In State transition logic is distributed throughout state classes which introduces coupling between them. State Machine addresses this issue. It separates transition logic and the behavior in a particular state. Following is the UML Diagram:

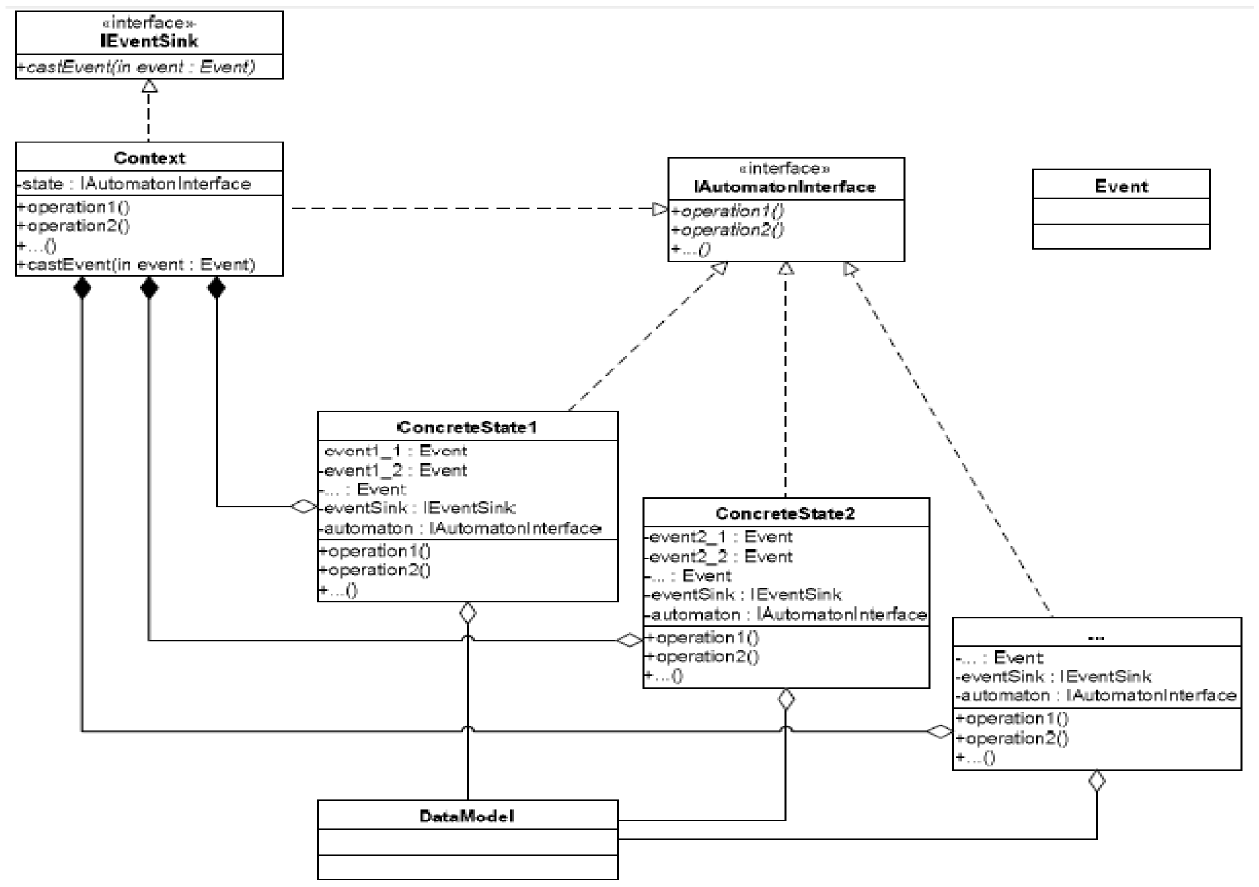

Figure 1.23 State Machine Pattern

StateMaps [68]

StateMaps is used to determine the actual next state, so that a derived state class may “replace” it with the next state appropriate for the derived machine. State machine designers can easily construct complex state machines using object-oriented techniques, while sharing code for actions. This implementation technique permits the use of complex hierarchies in practice. Following is the UML Diagram:

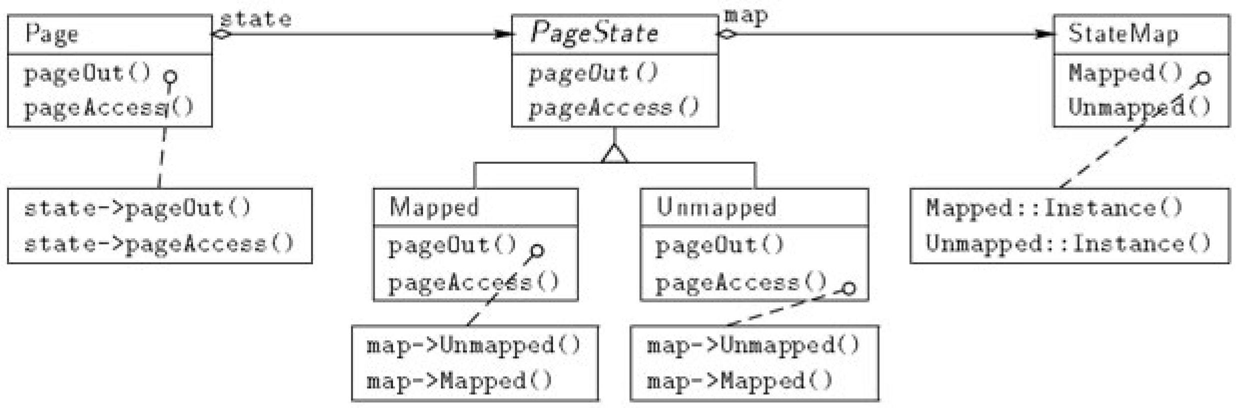

Figure 1.24 UML Diagram of StateMaps

Three-Level Finite State Machine [69]

This variant is applied in any context where behavior may be controlled by more than one finite state machine. Following is the demonstration of three level FSM:

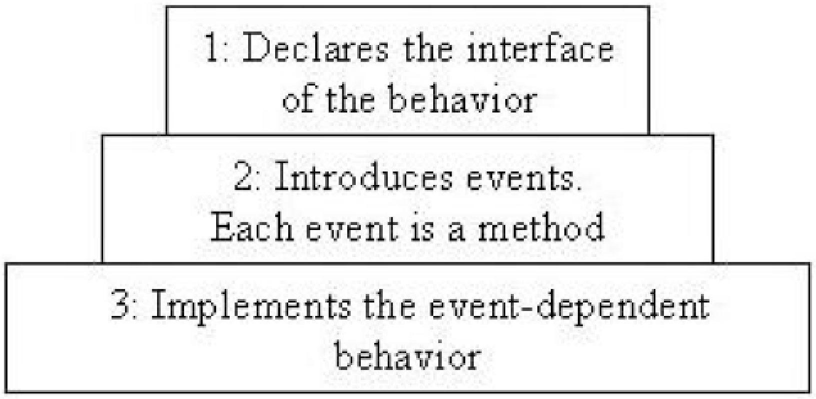
1: Declares the interface of the behavior
2: Introduces events. Each event is a method
3: Implements the event-dependent behavior

Figure 1.25 Three level abstraction of FSM pattern

Reflective State [69] [70]

This variant is applicable when the number of states are relatively large in number. This variant suggests to separate the application into two levels: the finite state machine (FSM) level and the application level. The FSM level corresponds to the meta-level of the Reflective architecture, while the application level corresponds to the base level of the architecture.

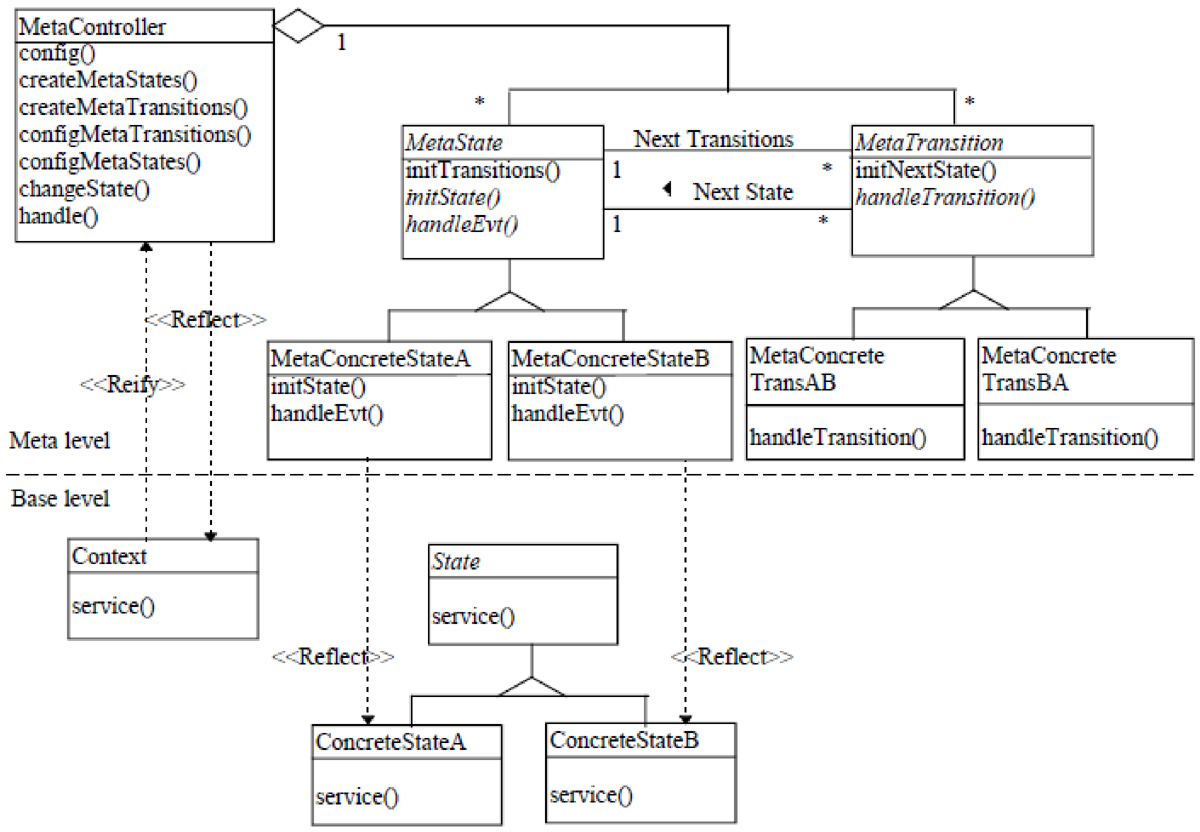

Figure 1.25 UML diagram of reflective state pattern

Persistent State Machine [71]

A persistent object is an OO representation of a database entity. An entity may be related to a process or a work. The Persistent State Pattern has been presented in two forms. In the simple form, where the pattern is sufficient for managing the state transition logic of a persistent object in a managed transaction framework. The simple form is sufficient for the most part of the information systems, where states are related to business processes or workflows. The state of an entity that is part of a process or workflows usually reflects simply the state of the process or workflow themselves. If the persistent object needs to behave differently for each different state, the behavioral form of the Persistent State Pattern must be used. In this case, the basic of the State pattern is added to the simple Persistent State Pattern, and the result is a pattern that is able to manage different behaviors for persistent objects in an environment with managed database transactions. Below is the UML structure:

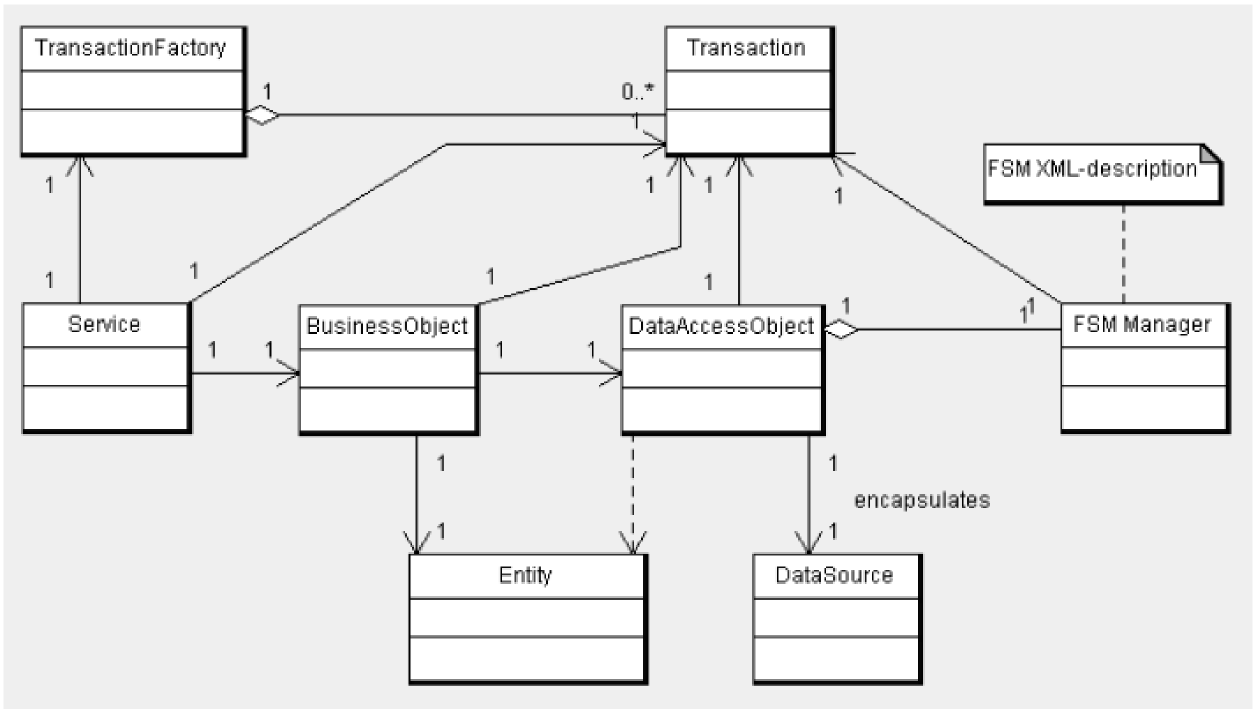

Figure 1.26 Persistent State machine