**1.1 State Compartment**

The state pattern is used to manage the state of objects. The tasks include are saving the current state and calculating the state transitions. The behavior of the objects depends on their own state.

**Documentation**

The classes are GenericComponent and StatePattern class. The GenericComponent class

represents components to which states should be assigned. This can play all the roles. This is possible by playing the state roles over a role-based solution to get a state. So there is no aggregation between the Role Component and AbstractState to save his state. The role Component represents an element, which has a state and its behavior depends on this. It has methods for configuration by the initial assignment of a state.

In these algorithms, state transitions can be implemented. The role Component has an interface called Operation which represents the methods of algorithm. These are from client side. The changeState functions are for managing the state. The client is represented by role Client. It can configurate through her methods. The state roles can be divided into concrete roles and abstract state. The AbstractState has a function changeState that changes the state of the Component. It owns the methods of the algorithm part. The role Group ConcreteStates have these methods. In abstracts state has the methods

In the role of the abstract

State, these are specified as an interface with a standard implementation. In the

concrete states, the methods of the algorithm portion are refined, in which the state spec

? behave is implemented.

In a meaningful instantiation of the design pattern, there is at least one component

which exactly has an abstract state. That is why the role of abstract states

played as often as the role Component. At the same time, it does not always have to be a client

give. Even the roles of concrete states do not have to be at any time by a class

to be played. From this information, the occurrence constraints can be derived.

**Figure 1.1 State Pattern Class Model**

**Evaluation**

The following characteristics are documented by the Gang of Four in the components applicability.

1. The Compartment uses classes to represent elements of state spaces. The behavior of an object depends on its condition.

2. The design pattern separates the state-specific from the general behavior of a class.

3. The participants are context, state and the concrete states.

4. The call of a state-specific method on the component is delegated to its state. The characteristic is also documented in the components Collaborations and Code.

5. The class State is abstract.

6. The number of concrete states in the design pattern is not limited.

7. The Context class decrypts the interface for methods and the client accesses these methods.

.

8. The Context class has an abstract state as an attribute, which is given by a concrete state Object.

9. The State class interface implements for the methods of concrete states.

10. The classes of concrete states implement the state-specific behavior.

11. The Context class can pass itself as parameter to methods of the states,

so that it can be applied to the context.

12. The clients do not call methods directly on the states, but only on

the class context. This provides an interface for this and delegates the calls to the

Conditions.

13. An object of the Context class must first be configured, in which you have a concrete one

State is assigned.

a) This can be done by the client.

b) This can be done by the Context class in its constructor.

14. State transitions are the responsibility of the

a) context class.

b) concrete state classes.

15. New states and state transitions have the encapsulation properties.

16. This compartment has to deal with many distributed classes.

17. State transitions are accomplished by a type change of state.

18. A context does not have to take care of the consistency of its condition,

because state transitions can be controlled by the states.

19. State objects can be shared without instance variables. Then have 2 context objects.

One is the same state-specific behavior and the other one is state can be different

Classes which are public.

//////////

20. The state transfer depends on the stability of state model.

a) The management of state transitions through the context is unexible and should only

be used in an xed state model.???????????? Need to check

b) The management of state transitions through the concrete states is exible. therefore

The state model can be changed at any time without much effort. These

Variant is documented in the component example code.

21. The state transitions can be controlled by the concrete states, the context needs to provide a method for this..

22. The following can happen with the state objects during the state change:

a) The old state is destroyed while the new state is generated. This saves a context of the currently used states.

b) The old state is not destroyed but only marked as inactive. The new condition which referenced in the context is set as an active state. So a context must always store all states that it once possesses.

23. In delegation-based programming languages, dynamic inheritance can be used to state.

24. The elements of the state space do not have to be disjoint.

25. The patterns participate in the compartment are Component and State.

26. There is an aggregation relation between the two roles of the compartments.

27. There is a roll prohibition between the two roles of the design pattern.

**Figure 1.3 State Pattern features in 3 representations**

**Figure 1.3 State features in 3 representations**

The context and the associated class in the model is called as component. The abstractness of a Class must be attributed. There should have an abstract character which can not be played without its counterpart with the concrete character. One component

playing the class always automatically plays the role of the abstract state.It can be also that an uncontrolled component has not yet assigned a concrete state. In this case, it turns out that the State role has no abstract character and thus will violate the feature 5.

Feature 9. is represented by a role-based solution. A component playing the component Class also takes on the state roles that encapsulate the desired state-specific behavior. Thus, there is no aggregation relation as presented in the Gang of Four and Dirk Riehle. The features 10th, 11th, 14th and 15th are explained by the explicit statements of the methods represented in the corresponding roles. Characteristic 12 is implicitly given by the role model, because a component also performs its functions by playing the state roles.

Therefore, they can run on the same class, so no data is passed to other classes. Input parameters must be made accessible by methods.

Features 17 to 20 and 22 can be presented by the CROM which consider the role-based presentation. So in this model, the code is not distributed among many classes, but in many roles. It is not the type of state attribute of a component that is changed, but the state role which is playing as class.

The component in CROM representation assumes the corresponding state roles. This is how the state-specific behavior of the component is realized. This solution contradicts

the two features specified by Riehle. The solution presented here uses the role-based

But programming is stronger than Riehle's approach.

?????????????? Last parahgraph. Which is represenyted in CROM??

