2.1 Bridge Pattern

The bridge pattern allows separation of abstraction of a class from its implementation. Therefore, the implementation can be changed at runtime. The design pattern also allows that the abstraction and implementation can vary and evolve independently.

Figure 2.1 Bridge Pattern Class Model

Documentation

Need to add later !!!!!!!!!!

Evaluation

The following features are shown in the diagram by the **Gang of Four**. Unless otherwise specified, they are described in textual terms in the applicability component.

1. The Bridge pattern separates the abstraction of a class from its implementation. The implementation of a class can be changed at runtime. This feature is additionally documented textually in the component intention and consequences.

2. The abstraction, as well as the implementation can be refined by an inheritance structure.

3. Changes in the implementation have no influence on the code of the clients.

4. The pattern allows the classification of a class into two aspects, which is realized by the two inheritance structures.

5. A particular implementation can be shared by the design pattern of multiple objects. This feature is also described textually and with example code in the Implementation component.

The following features are documented **graphically and in textual** participants.

6. The participatants in the design pattern: Abstraction, Implementor, the concrete implementation classes, and the refined abstraction classes.

7. There are 2 inheritance structures. One refines the class Abstraction, while the other implements the specialist. These can be extended independently of each other. This feature is also described textually.

8. The class Abstraction is abstract.

9. The class Abstraction provides the interface for the implementation classes.

10. The class Abstraction has an object of the implementation classes.

11. The class Implementor is abstract.

12. The class Implementor provides the interface for the concrete implementation classes.

13. The interface of the abstraction classes can be more complex than that of the implementation classes. This feature is only described textually in the Participants component.

14. The concrete implementation classes implement the methods of the class Implementor.

The following features are graphically displayed in the Structure part.

15. A client acts with the class Abstraction.

16. For a method call in the abstraction class, it delegates the call to the implementation. This feature is also mentioned textually in the Collaborations component.

The following features are described textually.

17. Separating the abstraction from the implementation saves compile time because the implementation and abstraction classes are compiled independently.

18. The separation of the abstraction from the implementation can be used for a better structuring of the system. At the highest architectural level, only the abstraction and the abstract implementation class need to be known.

19. The separation of the abstraction from the implementation can be used to hide implementation details such as reference counts.

The following features are described textually in the Implementation component.

20. Even with only one implementation, the design pattern can still be used to separate the abstraction from its implementation.

21. The decision as to which concrete implementation class is instantiated can be realized by various means.
a) An abstraction class that knows all implementation classes chooses the right one. The decision may depend on the given parameters of its constructor.
b) By default, the abstraction class instantiates an implementation class. This is then changed at runtime at certain events.

c) The decision is delegated to another object that knows the implementation classes. This variant is shown in the sample code section.

The features 7,9,10 and 16 are also explained in example code.

The features 1, 9, 13 and 21 are additionally described textually by Dirk Riehle.

The following features are to be found in the representation of **Dirk Riehle** design patterns . Unless otherwise indicated, these are only presented graphically.

22. The design pattern includes the following roles: client, abstraction, and implementer.

23. The client does not act with the implementation roles, but with the role Abstraction. This feature is also described textually.

24. The role Abstraction has an aggregation to the role Implementer.

25. The composition constraints from the Role Relationship Matrix [Rie09, p. 34] apply.

The Feature Table is Shown as follows.

	Features	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21a	21b	21c	22	23	24	25
GoF	Texual	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X							
	Code					X																						
	Graphically						X	X	X	X	X	X	X		X	X	X											
Riehle	Texual	X								X				X								X	X	X	X	X	X	X
	Graphically																											
CROM	Graphically																											