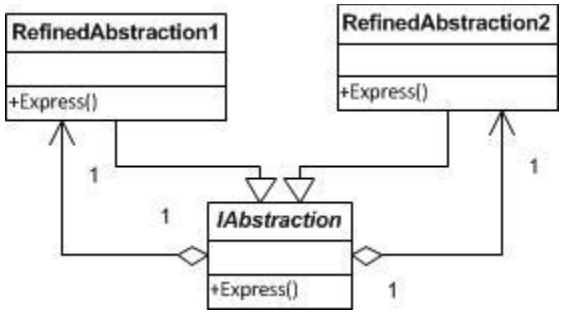
7. Bridge Pattern (Handle / Body idiom)

Software applications are developed to support business requirement and needs. As these requirements changed or evolve with time the need to change in existing structure may be possible by incorporating new functionalities. To compensate such changes, application classes are not purely developed to adjust that change during the maintenance period.

Abstraction can be implemented in a number of ways to support the multiple forms of an object, which brings inheritance on board to achieve that particular form / state of the object. Though it solves the problem but at the same time abstraction is completely coupled with all implementers and here comes in the big problem when we need to reuse, modify or extend the already implemented functionality. Bridge pattern was discovered to tackle such problem that resides in the structure of the application. [1] describes how to decouple the abstraction from the underlying implementation so that both can have a little or no impact of change when modified reused or extended. Following are the variants

7.1 Cheshire Cat Idiom / Degenerate Bridge [1]

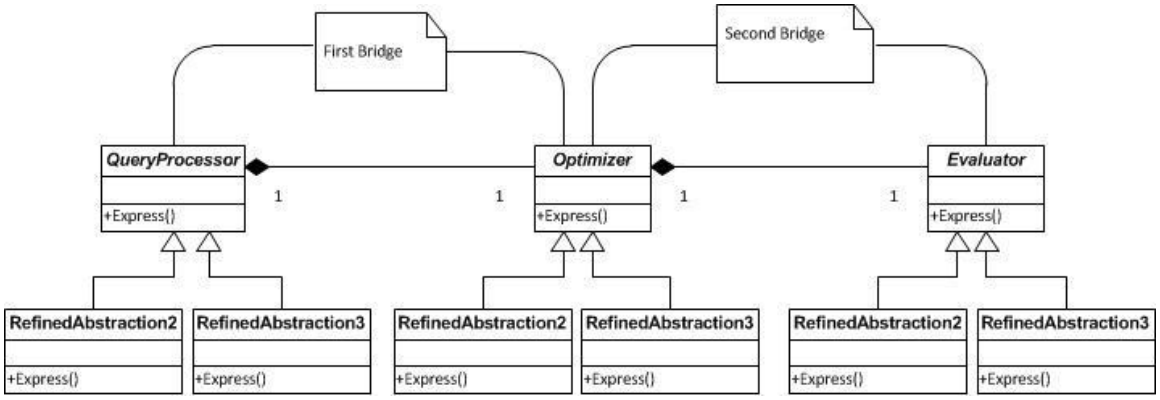
This variant was first discussed by [25] and then later on [1] named it as an implementation of bridge pattern. This variant reported to use only one abstraction implementation eliminating the abstract implementation class. It specifies one to one relationship between the abstraction and all its implementer classes.



7.1 UML Diagram of Degenerate Bridge pattern [1]

7.2 Cascading Bridge [25]

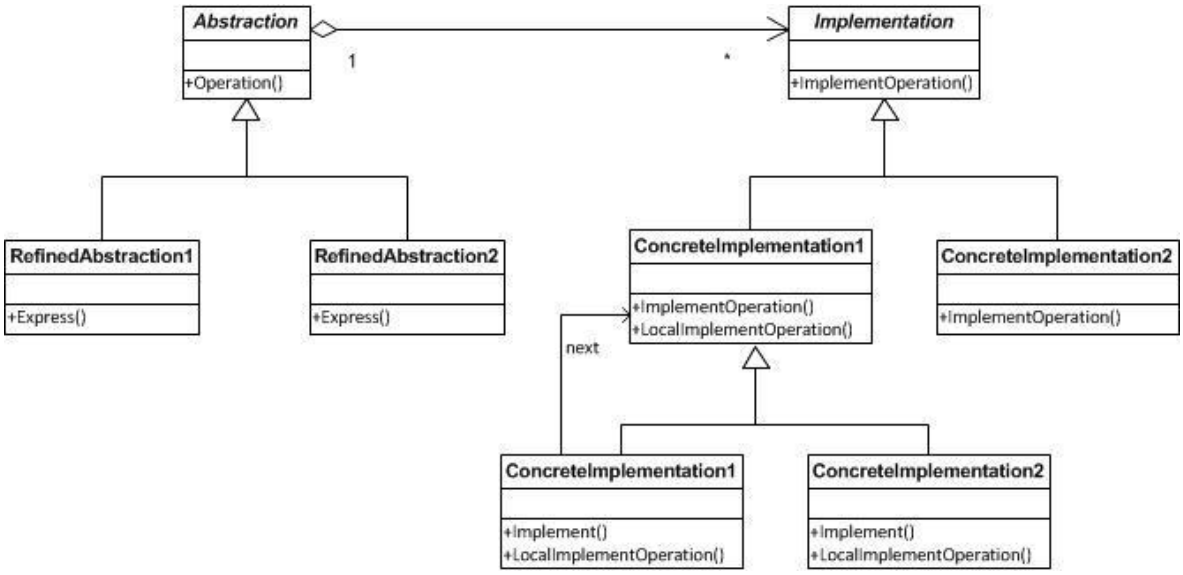
This variant defines a new concept of linking multiple bridges in which the middle component plays a two way role i.e. implementation role for the first bridge and the abstraction role for the second bridge. Such linking of bridges in a chain holds the opportunity that specific set of operations are performed by a component and transfer the remaining set of operations to its immediate sibling in the chain and so on.



7.2 UML Diagram of Multiple bridges [25]

7.3 Folded cascading bridge [25]

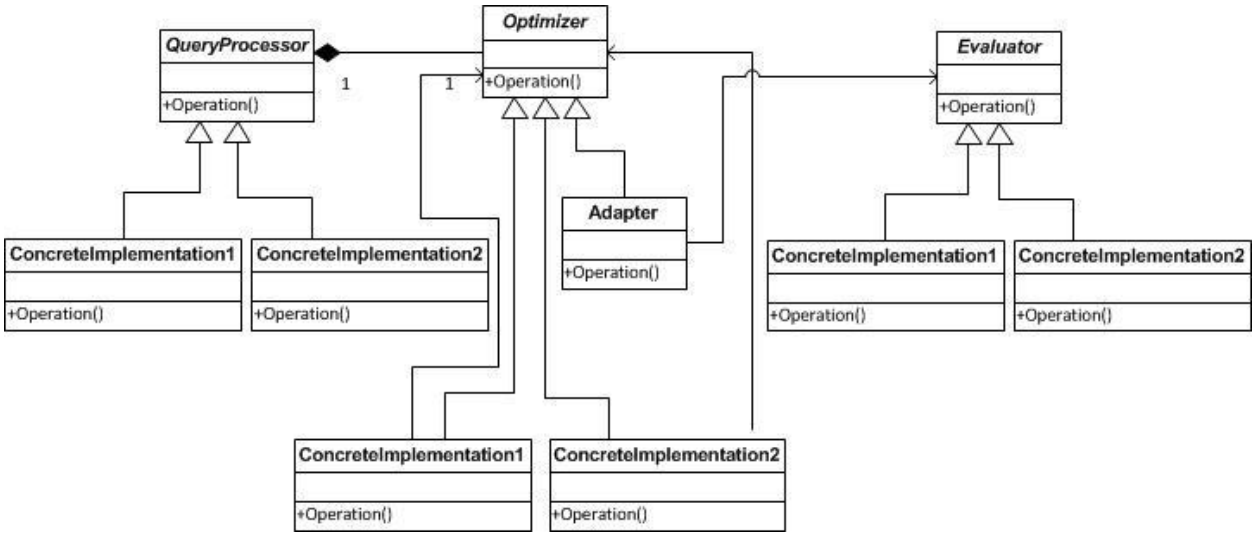
New component introductions into the bridge chain usually require the length of the chain vary anytime. Such a scenario can be best symbolized by multi-processor or distributive environments where a certain task in an execution space is driven by remote machine or processor. This variant involves a recursive link in likely to be varied components in terms of order or sequence and a common interface is shared among the underlying implementations which maintains the reference to the next link in the chain.



7.3 UML Diagram of folding cascading bridge pattern [25]

7.4 Partially-Folded Cascading Bridge [25]

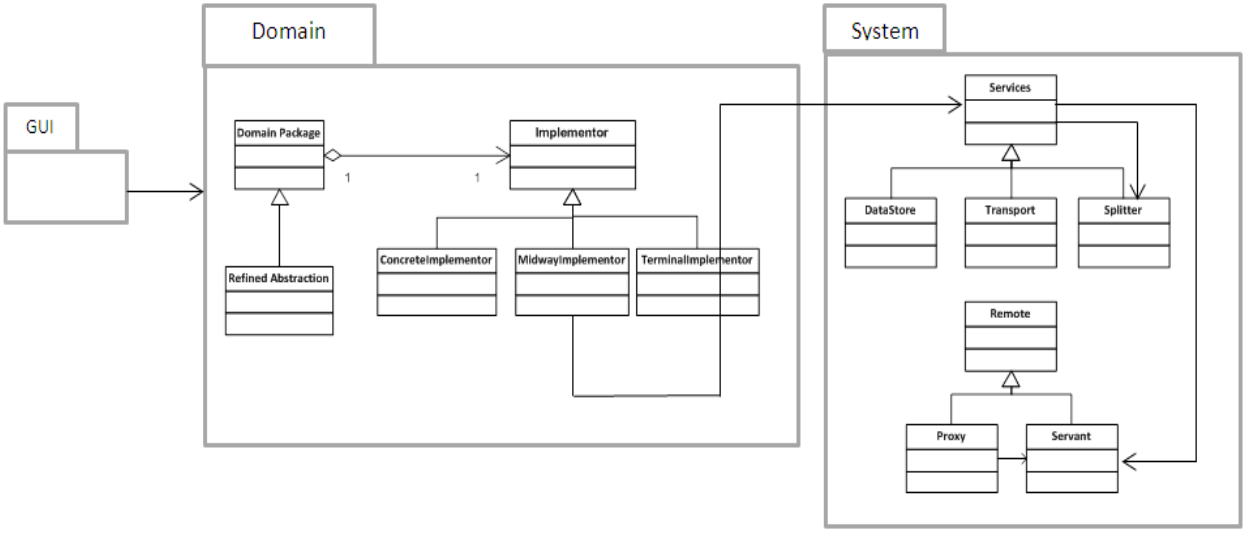
This variant is a compound implementation folded and unfolded cascading bridge pattern [25] which describes a concrete implementation class acting as an adapter for another component. The other implementation classes maintain a reference to its shared abstraction.



7.4 UML Diagram of partially folded cascading bridge [25]

7.5 Architectural Cascading Bridge [25]

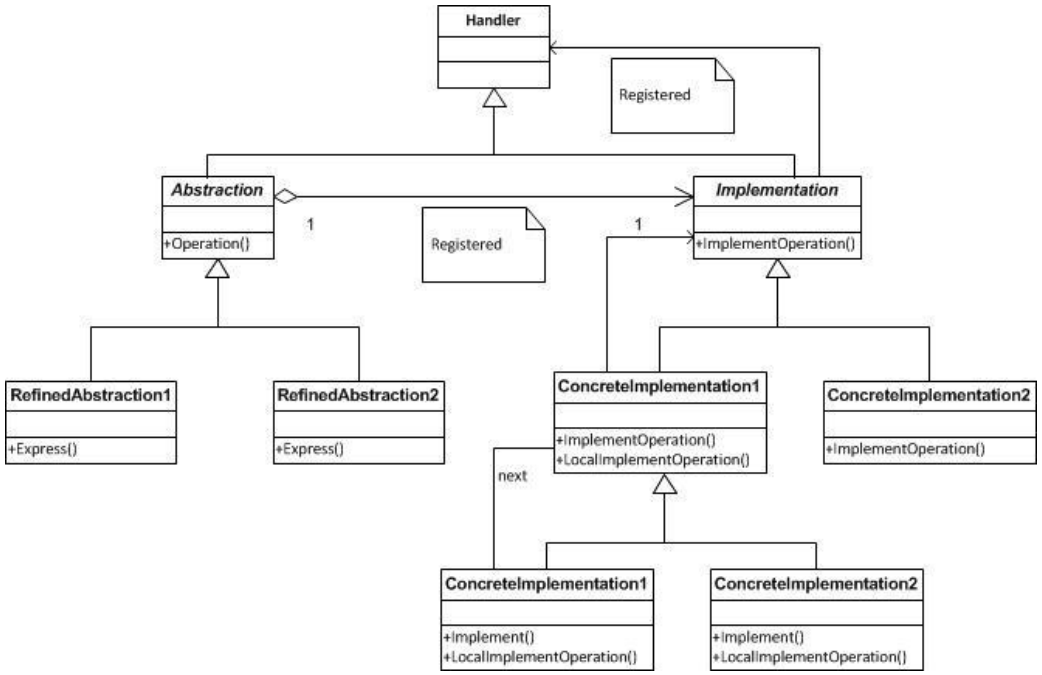
This variant of bridge pattern involves groups of classes working together belonging different packages modeled to represent subsystems. Clients (GUI subsystems) interacts with the service layer indirectly through key methods of the Domain Package which further interact with back end System Package where the concrete implementations of classes are defined.



` 7.5 UML Diagram of Architectural cascading bridge [25]

7.6 Bi-Directional Cascading Bridge [25]

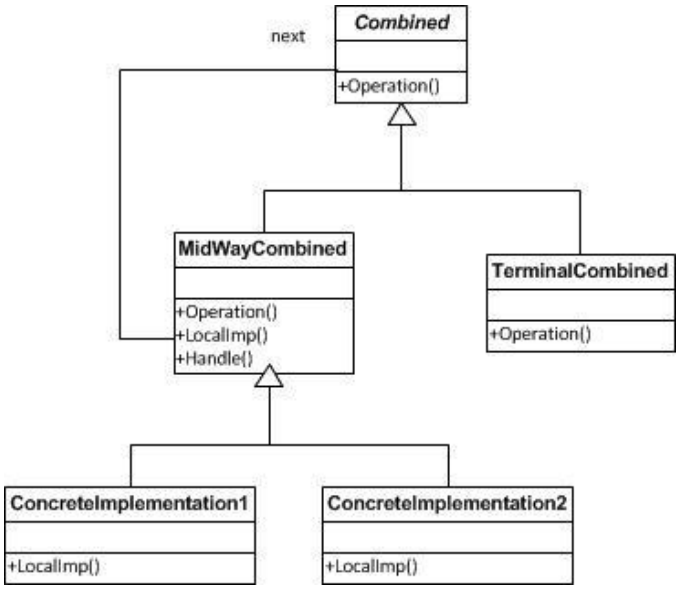
In this variant the interactions between the abstraction and the implementation versus implementation and implementation takes place. Each implementor notifies its registered sibling with an acknowledgement in response that a certain event has been occurred. The author [25] quoted change notification as an example.



7.6 UML and class diagram of bi directional cascading bridge.

7.7 Single Protocol Cascading Bridge [25]

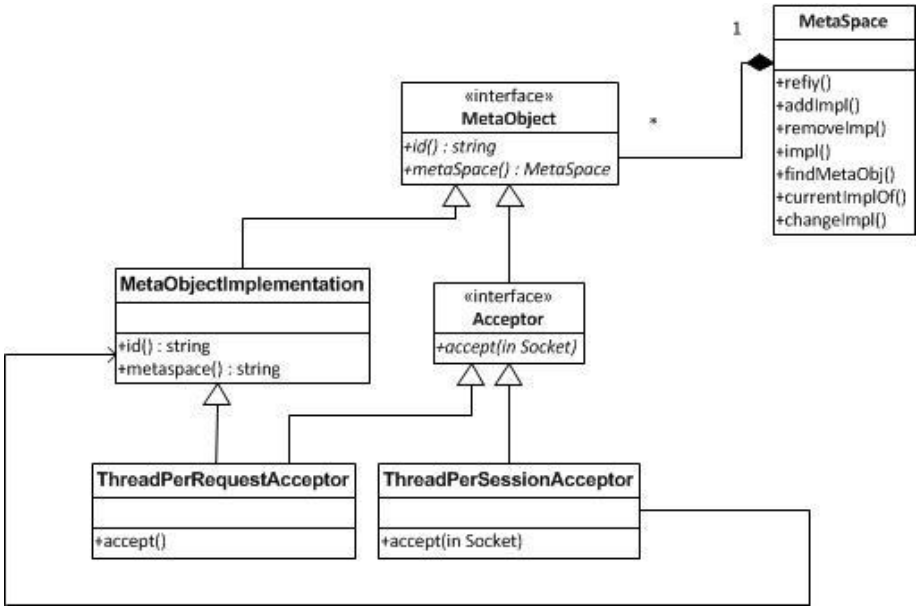
This variant describes the implementation where the implementors having same underlying abstraction protocol.



7.7 UML Diagram of Single protocol cascading diagram

7.8 Dynamic Switcher Bridge [26]

The variant quoted by [26] was used in the implementation of Webserver. The intent of its use is to dynamically switch the implementors of an abstraction (Meta object) at runtime. The benefit is to provide alternative implementations to the Meta objects despite any failure in accessing the system probably shutdown. This variant represents a central point interface which is a derived from abstraction interface. All the implementors will override the accept method differently.



7.8 UML Diagram of dynamic switch Implementor Bridge