Architectural pattern composition catalogue

Technical Report

IRISA/ArchWare-2013-TR-03

Minh Tu Ton That[1] and Salah Sadou[1]

[1] IRISA, University of South Brittany, France
{minh-tu.ton-that, salah.sadou)@irisa.fr

Abstract. This report documents possible cases of architectural pattern composition. Unit patterns as well as combined patterns are gathered from the literature. A pattern can have various variants. Each variant can be the application of different structures within the pattern or the combination of the other variants.

1 Pattern composition catalogue

For each pattern we show the source from which we study the pattern as well as a brief description about the pattern. For the sake of simplicity, we do not document the context, the problem, the solution, the implementation, known uses and consequences of pattern. Instead, we concentrate on the structure of the pattern which plays an important role in our study about pattern composition. Especially, for each combined pattern, we show the constituent patterns from which the pattern is composed. For the pattern variant that is formed from other pattern variants, we only show the source variants, the variant's example itself can be deduced from the examples of source variants.

1.1. **Enabled Cycle Component [1][2][3]**

The pattern states that not non-adjacent components can share their data through their connectors.

Figure 1 shows an example of the Enabled Cycle Component pattern. Component 1 can be connected to Component 3 even though they are not adjacent (Component 2 in the middle)

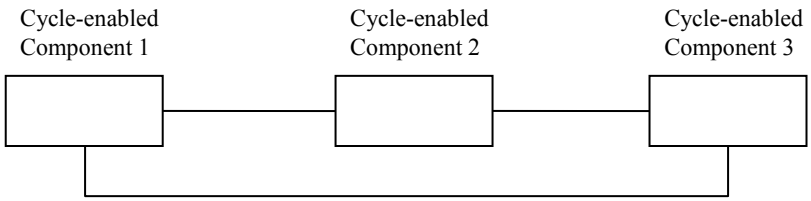

Figure 1. Cycle enabled component

1.2. **Forbidden Cycle Component [2]**

The pattern states that only two adjacent components can share data through their connector, but not non-adjacent components.

Figure 2 shows an example of the Forbidden Cycle Component pattern. Component 1 cannot be connected to Component 3 because they are not adjacent (Component 2 in the middle)

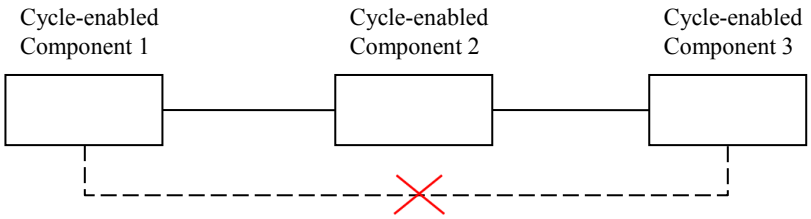

Figure 2. Forbidden Cycle Component pattern

1.3. **Shield [4]**

In this pattern, one or more components act as 'shields' for a set of components that form a subsystem. No client should be allowed to access members of the subsystem directly, but access should happen only through these 'shields'.

Figure 3 shows an example of the Shield pattern. Clients can only access to subsystems through shields. Direct access from a client to a subsystem (for instance, from Client 3 to Subsystem 3) is forbidden.

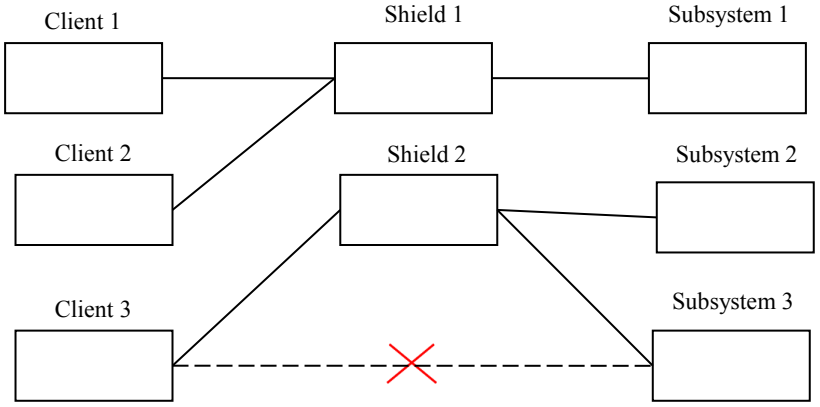

Figure 3. Shield pattern

1.4. **Layers [2][3]**

We consider the most described Layers pattern in the literature as the Basic Layer variant to distinguish from the other variants of Layers pattern.

1.4.1. **Basic Layer [2][3]**

In the Basic Layers pattern, the system is structured into Layers so that each layer provides a set of services to the layer above and uses the services of the layer below.

Figure 4 shows an example of the Layers pattern with three layers: Layer 1, Layer 2 and Layer 3

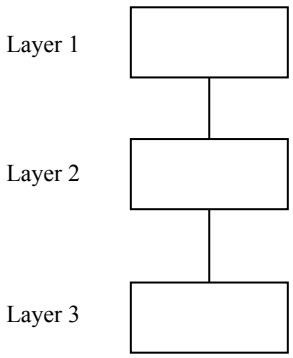

Figure 4. Basic Layers pattern

1.4.2. **By-passed Layers [2]**

In this variant of Layers pattern, layers can be by-passed: higher-level layers access lower-level layers without passing through the layer beneath.

This variant is the composition of two constituent patterns: the first pattern is Enabled Cycle Component (section 1.1) and the second pattern is Basic Layer (section 1.4.1). The composition is formed by creating the new element *Cycle-enabled Layer* through the overlapping of the *Cycle-enabled Component* in the first pattern and the *Layer* in the second pattern.

Figure 5 shows an example of the By-passed Layers pattern with three layers: Cycle enabled Layer 1, Cycle enabled Layer 2 and Cycle enabled Layer 3. As we can observe, the Cycle enabled Layer 1 can access the Cycle enabled Layer 3 by bypassing the Cycle enabled Layer 2.

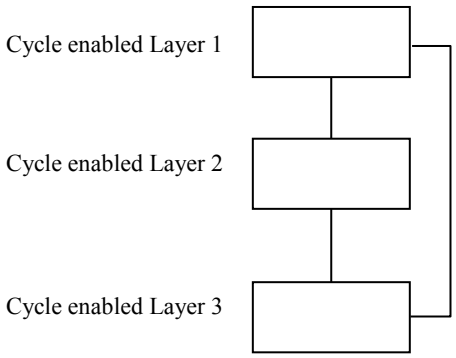

Figure 5. By-passed Layers

1.4.3. Not by-passed Layers [2][3]

In the pure form of the pattern, layers should not be by-passed: higher-level layers access lower-level layers only through the layer beneath.

This variant is the composition of two constituent patterns: the first pattern is Forbidden Cycle Component (section 1.2) and the second pattern is Basic Layer (section 1.4.1). The composition is formed by creating the new element *Cycle-forbidden Layer* through the overlapping of the *Cycle-forbidden Component* in the first pattern and the *Layer* in the second pattern.

Figure 6 shows an example of the Not by-passed Layers pattern with three layers: Cycle forbidden Layer 1, Cycle forbidden Layer 2 and Cycle forbidden Layer 3. As we can observe, the Cycle forbidden Layer 1 cannot directly access the Cycle forbidden Layer 3 because it cannot bypass the Cycle forbidden Layer 2.

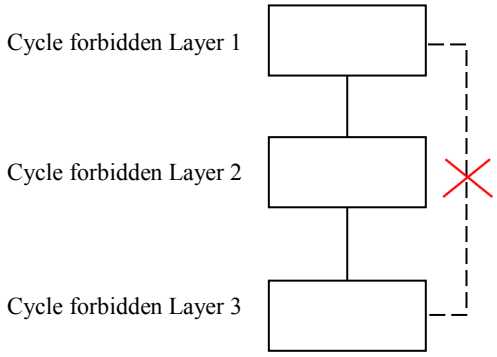

Figure 6. Not by-passed Layers

1.4.4. Client-Server Layers [2]

In this variant of Layers, two adjacent layers can be considered as a Client-Server pair, the higher layer being the client and the lower layer being the server.

This variant is the composition of two constituent patterns: the first pattern is Client-Server (section 1.10) and the second pattern is Basic Layer (section 1.4.1). The composition is formed by creating two new elements: The first new element *Client Layer* is formed through the overlapping of the *Client* in the first pattern and the *Layer* in the second pattern. The second new element *Server Layer* is formed through the overlapping of the *Server* in the first pattern and the *Layer* in the second pattern.

Figure 7 shows an example of the Client-Server Layers pattern with two layers: Client Layer and Server Layer. The Client Layer requests information or services from Server Layer.

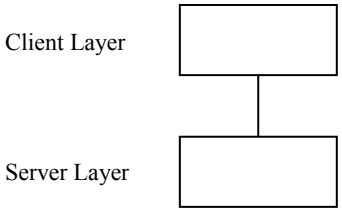

Figure 7. Client-Server Layers

1.4.5. **Filtered Layers [2]**

In this variant of Layers, Pipes and Filters can be used for communication between layers, if data flows through layers are needed.

This variant is the composition of two constituent patterns: the first pattern is Pipes and Filters (section 1.5) and the second pattern is Basic Layer (section 1.4.1). The composition is formed by creating one new element *Filtered Layer* which is formed through the overlapping of the *Filter* in the first pattern and the *Layer* in the second pattern.

Figure 8 shows an example of the Filtered Layers pattern with two layers: Filtered Layer 1 and Filtered Layer 2. These two layers are connected using a Pipes and Filters pattern. Except for Filtered Layer 1 and Filtered Layer 2, the three other filters are: Filter 1, Filter 2, Filter 3.

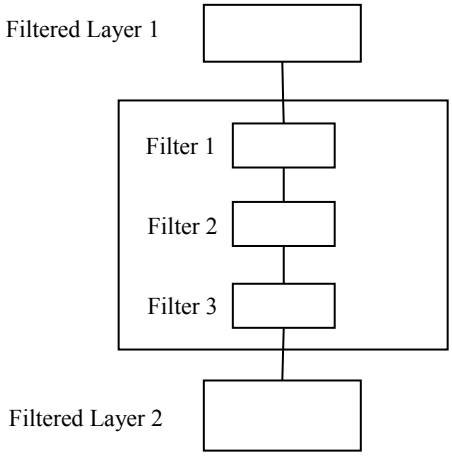

Figure 8. Filtered Layers pattern

1.5. **Pipes and Filters [2][3]**

We consider the most described Pipes and Filters pattern in the literature as the Basic Pipes and Filters variant to distinguish from the other variants of Pipes and Filters pattern.

1.5.1. **Basic Pipes and Filters [2][3]**

In a Basic Pipes and Filters pattern a complex task is divided into several sequential sub-tasks. Each of these sub-tasks is implemented by a separate, independent component, a Filter, which handles only this task.

Figure 9 shows an example of the Pipes and Filters pattern with three filters: Filter 1, Filter 2 and Filter 3

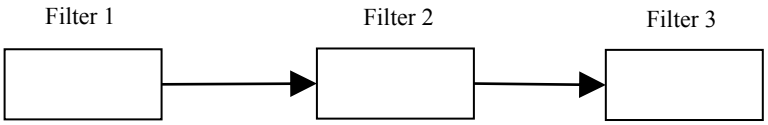

Figure 9. Basic Pipes and Filters pattern

1.5.2. **By-passed Pipes and Filters [2]**

In this variant of Pipes and Filters pattern, filters can be by-passed: filters can access non-adjacent filters.

This variant is the composition of two constituent patterns: the first pattern is Enabled Cycle Component (section 1.1) and the second pattern is Basic Pipes and

Filters (section 1.5.1). The composition is formed by creating the new element *Cycle-enabled Filter* through the overlapping of the *Cycle-enabled Component* in the first pattern and the *Filter* in the second pattern.

Figure 10 shows an example of the Pipes and Filters pattern with three filters: Cycle enabled Filter1, Cycle enabled Filter2 and Cycle enabled Filter 3. As we can observe, the Cycle enabled Filter 1 can access the Cycle enabled Filter 3 by bypassing the Cycle enabled Filter 2.

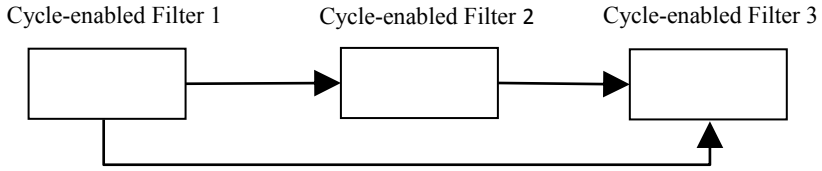

Figure 10. By-passed Pipes and Filters

1.5.3. Not by-passed Pipes and Filters (or Pipeline) [2][3]

In this variant of Pipes and Filters pattern, only two adjacent filters can share data through their pipe, but not non-adjacent filters.

This variant is the composition of two constituent patterns: the first pattern is Forbidden Cycle Component (section 1.2) and the second pattern is Basic Pipes and Filters (section 1.5.1). The composition is formed by creating the new element *Cycle-forbidden Filter* through the overlapping of the *Cycle-forbidden Component* in the first pattern and the *Filter* in the second pattern.

Figure 11 shows an example of the Not by-passed Pipes and Filters pattern with three filters: Cycle forbidden Filter 1, Cycle forbidden Filter 2 and Cycle forbidden Filter 3. As we can observe, the Cycle forbidden Filter 1 cannot directly access the Cycle forbidden Filter 3 because it cannot bypass the Cycle forbidden Filter 2.

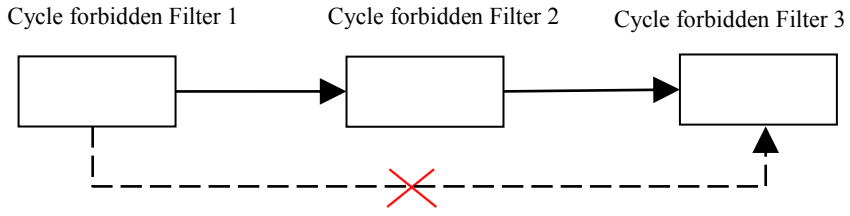

Figure 11. Not by-passed Pipes and Filters

1.5.4. Internally layer-structured Pipes and Filters [2]

In this variant of Pipes and Filters pattern, the Layers pattern can be used for structuring the internal architecture of Filters.

This variant is the composition of two constituent patterns: the first pattern is Layers (section 1.4) and the second pattern is Basic Pipes and Filters (section 1.5.1). The composition is formed by creating the new element *Internally layer-structured*

Filter through the overlapping of the entire *first pattern* the *Filter* in the second pattern.

Figure 12 shows an example of the Internally layer-structured Pipes and Filters pattern with three filters: Internally layer-structured Filter 1, Internally layer-structured Filter 2 and Internally layer-structured Filter 3. All these three filters are internally structured by Layer architecture.

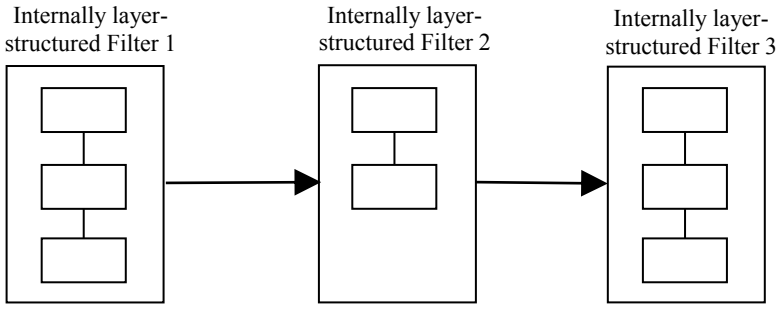

Figure 12. Internally layer-structured Pipes and Filters pattern

1.5.5. Data sharing Pipes and Filters [2]

In this variant of Pipes and Filters pattern, it can be combined with data-centered architectures like Shared Repository to allow for data-sharing between Filters

This variant is the composition of two constituent patterns: the first pattern is Shared Repository (section 1.6.1) and the second pattern is Basic Pipes and Filters (section 1.5.1). The composition is formed by creating the new element *Internally layer-structured Filter* through the overlapping of the entire *first pattern* the *Filter* in the second pattern.

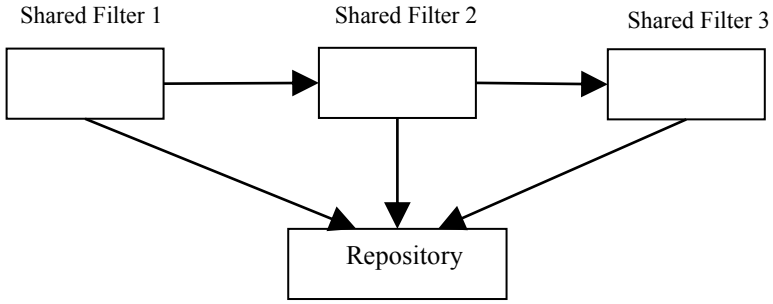

Figure 13. Data sharing Pipes and Filters pattern

1.6. **Shared repository [2][3][5]**

1.6.1. **Basic Shared repository [2][3][5]**

In the Shared Repository pattern one component of the system is used as a central data store, accessed by all other independent components.

Figure 14 shows an example of the basic Repository pattern with three Data Accessors: Data Accessor 1, Data Accessor 2 and Data Accessor 3.

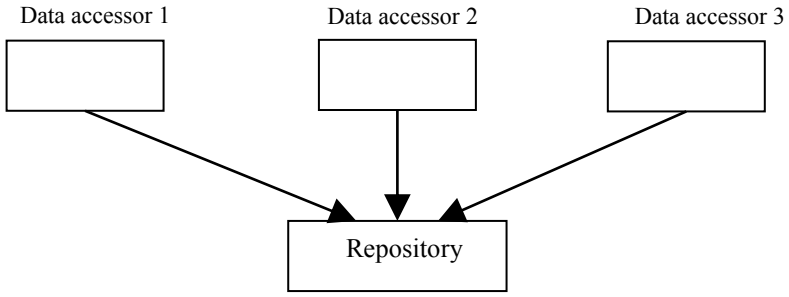

Figure 14. Basic Repository pattern

1.6.2. **Internally Layer structured Shared repository [2]**

In this variant of Shared repository pattern, the Layers pattern can be used for structuring the internal architecture of Repository.

This variant is the composition of two constituent patterns: the first pattern is Layers (section 1.4) and the second pattern is Basic Repository (section 1.6.1). The composition is formed by creating a new element named *Internally layer-structured Repository* which is formed through the overlapping of the *entire Layers* pattern and the *Repository* in the Repository pattern.

Figure 15 shows an example of the Internally Layer structured Shared repository pattern with 3 Data accessors: Data accessor 1, Data accessor 2 and Data accessor 3. The Repository is internally structure by a Layers pattern with 3 layers.

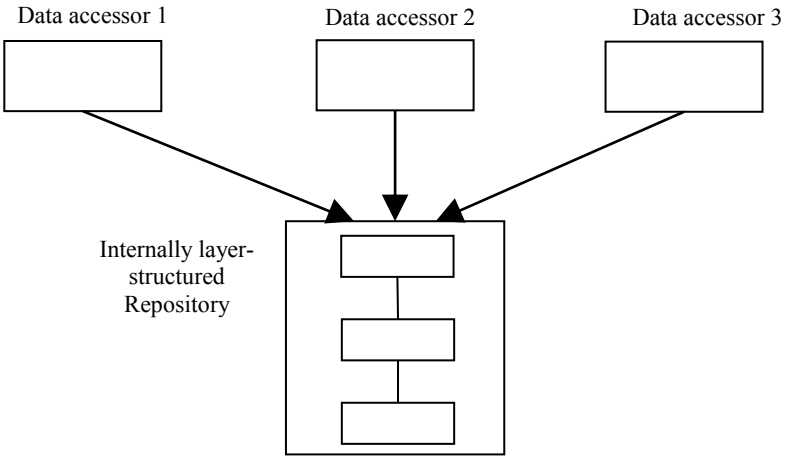

Figure 15. Internally Layer structured Shared repository pattern

1.7. **Microkernel [2]**

1.7.1. **Basic Microkernel [2]**

A MICROKERNEL realizes services that all systems need and a plug-and-play infrastructure for the system-specific services. Internal servers are used to realize version-specific services and they are only accessed through the Microkernel. On the other hand, external servers offer APIs and user interfaces to clients by using the Microkernel. External servers are the only way for clients to access the Microkernel architecture.

Figure 16 shows an example of the basic Microkernel pattern with one Client, three External Servers, one Microkernel and three Internal Servers.

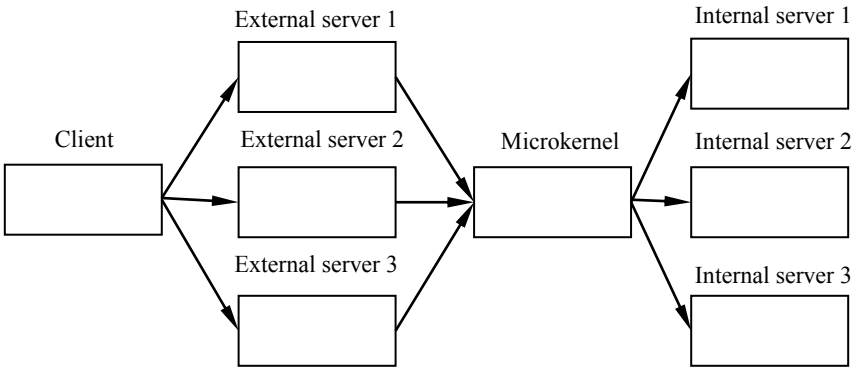

Figure 16. Basic Microkernel pattern

1.7.2. **Broker between Client and External Server [2]**

In this variant, Microkernel pattern can be combined with the Broker pattern to hide the communication details between Clients that request services and Servers that implement them.

This variant is the composition of two constituent patterns: the first pattern is Broker (section 1.13) and the second pattern is Basic Microkernel (section 1.7.1). The composition is formed by creating two new elements: *Client* which is formed through the overlapping of the *Client* in the Broker pattern and the *Client* in the Microkernel pattern, External Server which is formed through the overlapping of the *Server* in the Broker pattern and the *External Server* in the Microkernel pattern.

Figure 17 shows an example of the Broker between Client and External Server pattern.

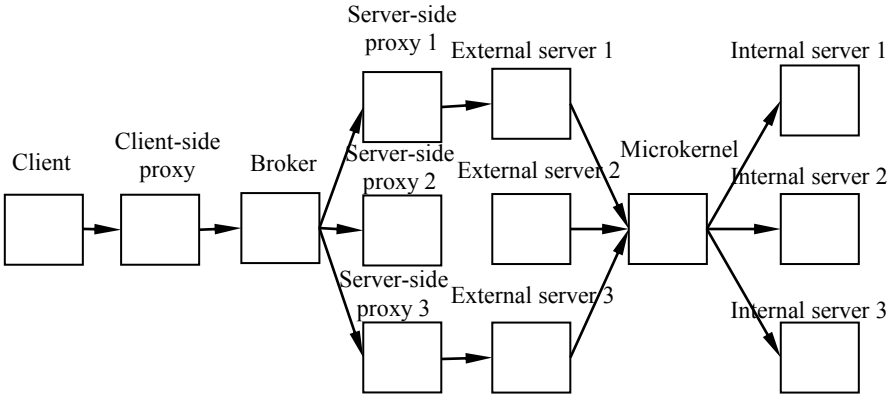

Figure 17. Broker between Client and External Server pattern

1.8. **PAC [2]**

The system is decomposed into a tree-like hierarchy of agents. Every agent is designed according to MVC.

Figure 18 shows an example of PAC pattern.

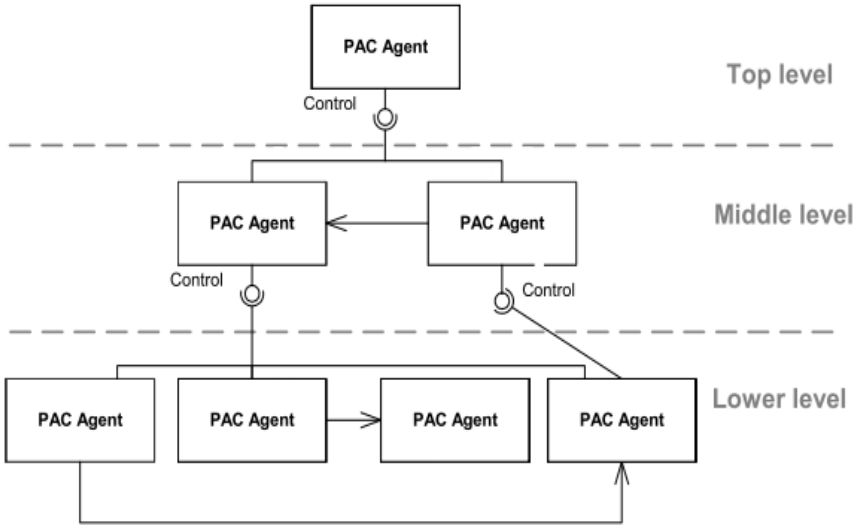

Figure 18. PAC pattern

1.9. **Indirection Layer [2]**

An Indirection Layer is a Layer between the accessing client and the the sub-system that needs to be accessed.

This variant is the composition of two constituent patterns: the first pattern is Shield (section 1.3) and the second pattern is Basic Indirection Layer. The composition is formed by combining i) The Client from the Shield pattern with the Client from the Indirection Layer pattern, ii) The Shield from the Shield pattern with the Indirection Layer from the Indirection Layer pattern, iii) The Sub-system from the Shield pattern with the Sub-system from the Indirection Layer pattern

Figure 19 shows an example of the Indirection Layer pattern.

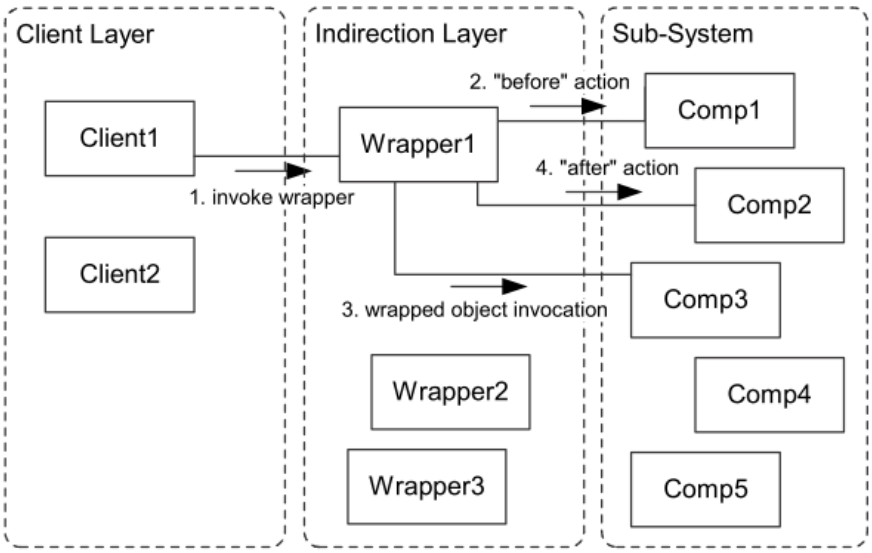

Figure 19. Indirection layer pattern

1.10. Client-Server [2][3][5]

1.10.1. Basic Client-Server [2][3][5]

The Client-Server pattern distinguishes two kinds of components: Clients and Servers. The Client requests information or services from a Server.

Figure 19 shows an example of the Client-Server pattern with three clients: Client 1, Client 2, Client 3.

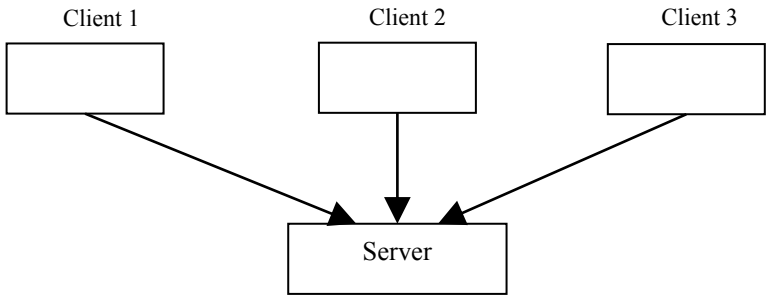

Figure 20. Client-Server pattern

1.10.2. Client-Server with Broker [2]

Sophisticated, distributed Client-Server architectures usually rely on the Broker pattern to make the complexity of the distributed communication manageable.

This variant is the composition of two constituent patterns: the first pattern is Broker (section 1.13) and the second pattern is Basic Client Server (Section 1.10.1). The composition is formed by combining i) The Client from the Broker pattern with the Client from the Client-Server pattern, ii) The Server from the Broker pattern with the Server from the Client-Server pattern.

Figure 21 shows an example of the Client-Server with Broker pattern.

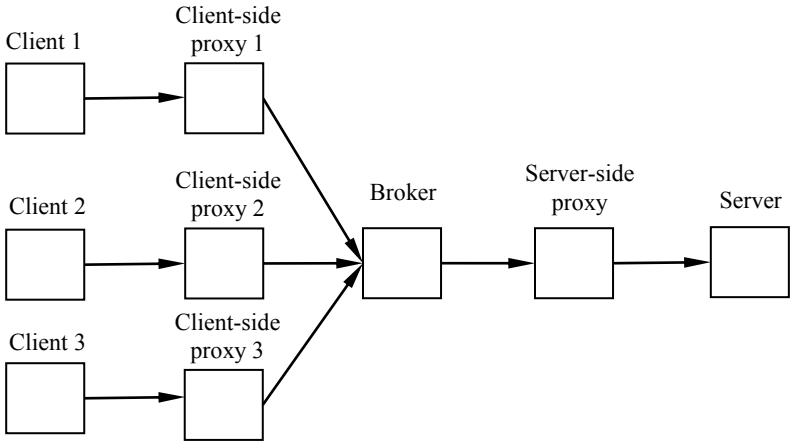

Figure 21. Client-Server with Broker pattern

1.10.3. **Client-Server through Microkernel [2]**

In this variant, a Microkernel introduces an indirection that can be useful in certain Client-Server configurations: a client that needs a specific service can request it indirectly through the Microkernel, which establishes the communication to the server that offers this service. In this sense all communication between clients and servers is mediated through the Microkernel, for reasons of e.g. security or modifiability.

This variant is the composition of two constituent patterns: the first pattern is Microkernel (section 1.7) and the second pattern is Basic Client Server (Section 1.10.1). The composition is formed by combining i) The Client from the Microkernel pattern with the Client from the Client-Server pattern..

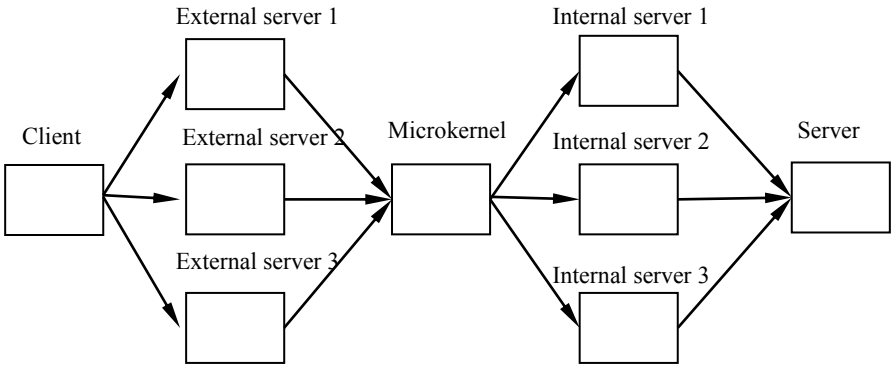

Figure 22. Client-Server through Microkernel pattern

1.11. MVC [2]

The system is divided into three different parts: a Model that encapsulates some application data and the logic that manipulates that data, independently of the user interfaces; one or multiple Views that display a specific portion of the data to the user; a Controller associated with each View that receives user input and translates it into a request to the Model.

Figure 23 is an example of MVC pattern

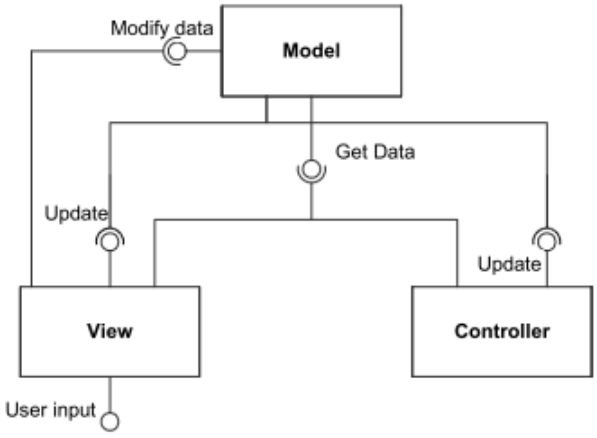

Figure 23. MVC pattern

1.12. **Proxy [3]**

This pattern let the client communicate with a representative rather than the component itself. This representative-called a proxy-offers the interface of the component but performs additional pre- and post- processing

This variant is the composition of two constituent patterns: the first pattern is Shield (section 1.3) and the second pattern is Basic Proxy. The composition is formed by combining i) The Client from the Shield pattern with the Client from the Basic Proxy pattern, ii) The Shield from the Shield pattern with the Proxy from the Proxy pattern and iii) The Sub-system from the Shield pattern with the Original from the Proxy pattern.

Figure 24 shows an example of the Proxy pattern.

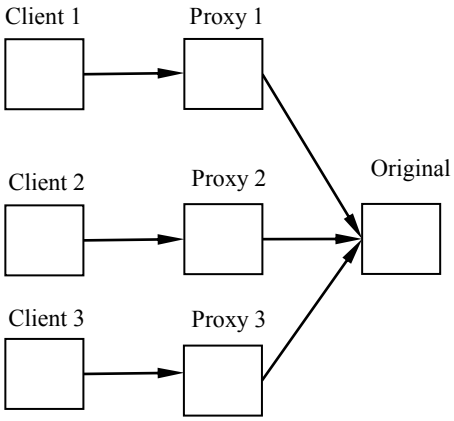

Figure 24. Proxy pattern

1.13. **Broker [2][3]**

In the Broker pattern, a Broker decouples Clients from Servers. Servers register themselves with the Broker, and make their services available to Clients through method interfaces. Clients access the functionality of servers by sending requests via the Broker.

The Broker is the composition of 2 Proxy pattern, one from Client to Broker and the other from Broker to Server

Figure 25 shows an example of Broker pattern. The proxies are applied on the client side and the server side.

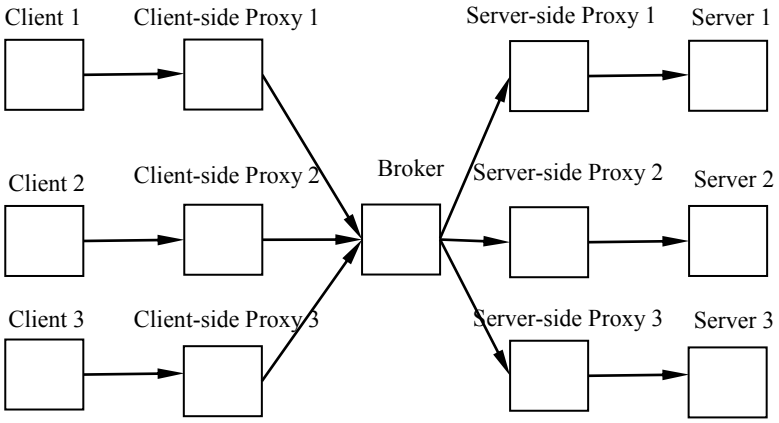

Figure 25. Broker pattern

1.14. Façade [4][5]

A façade component is used to abstract a part of the component architecture with negative coupling potential to legacy components. All legacy components must go through the Façade to communicate with the component

This variant is the composition of two constituent patterns: the first pattern is Shield (section 1.3) and the second pattern is Basic Façade. The composition is formed by combining i) The Client from the Shield pattern with the Legacy Component from the Basic Façade pattern, ii) The Shield from the Shield pattern with the Façade from the Façade pattern.

Figure 26 shows an example of Façade pattern.

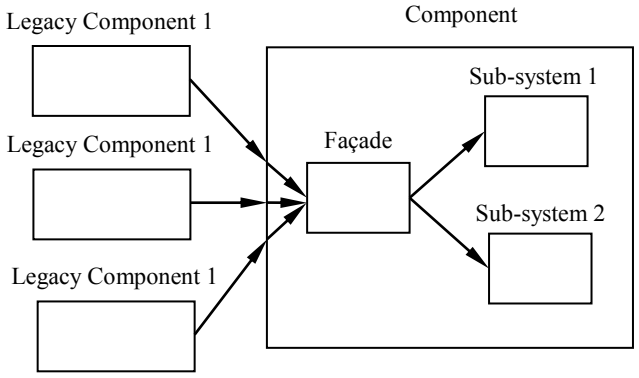

Figure 26. Façade pattern

1.15. **Legacy Wrapper [5]**

In this pattern, legacy components are encapsulated with wrappers to insure a seamless communication.

This variant is the composition of two constituent patterns: the first pattern is Shield (section 1.3) and the second pattern is Basic Legacy Wrapper. The composition is formed by combining i) The Client from the Shield pattern with the Consumer from the Basic Legacy Wrapper pattern, ii) The Shield from the Shield pattern with the Legacy Wrapper from the Basic Legacy Wrapper pattern, iii) The Sub-system from the Shield pattern with the Legacy Component from the Basic Legacy Wrapper pattern.

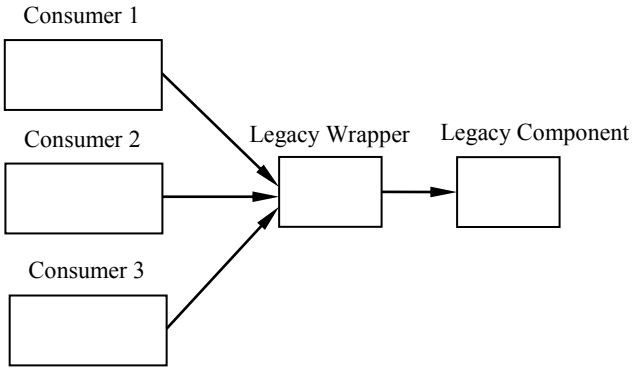

Figure 27. Legacy Wrapper pattern

1.16. **Pipes and Filters + Repository [5]**

In this pattern, the sink filter in the Pipes and Filters pattern is connected to the Repository in the Repository pattern to perform data reading/writing operations.

This variant is the composition of two constituent patterns: the first pattern is Pipes and Filters (Section 1.5) and the second pattern is Repository (Section 1.6). The composition is formed by a stringing composition: a new connector (playing the role of both the Pipe in the Pipes and Filters pattern and the Data reading/writing connector in the Repository pattern) is added to connect the first and the second pattern.

Figure 28 shows an example of the Pipes and Filters + Repository pattern.

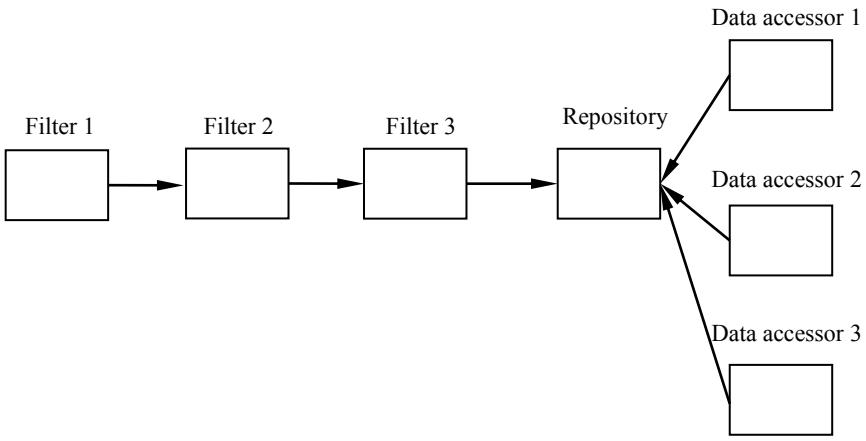

Figure 28. Pipes and Filters + Repository pattern

References

[1] Krakowiak, Sacha. "Middleware Architecture with Patterns and Frameworks." (2007).

[2] Paris Avgeriou and Uwe Zdun. Architectural patterns revisited a pattern language. In 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, pages 1–39, 2005.

[3] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture: a system of patterns. John Wiley & Sons, Inc., 1996.

[4] Uwe Zdun and Paris Avgeriou. A catalog of architectural primitives for modeling architectural patterns. Inf. Softw. Technol., pages 1003–1034, 2008.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. Documenting Software Architectures, Views and Beyond. Addison Wesley, 2010.

[6] Erl, T. SOA Design Patterns. Prentice Hall, December 2008.