5. Activity 3. Skeleton Architecture Design

Objectives of the Chapter

- ❖ To understand what Skeleton Architecture is
- ❖ To learn how to choose candidate architecture styles for a target system
- ❖ To learn how to systematically evaluate the applicability of candidate architecture styles
- ❖ To learn how to integrate selected architecture styles into a skeleton architecture

5.1. Overview of the Activity

This activity is to choose most appropriate architecture styles for a target system and integrate the styles into a stable and core structure of the target system, called a skeleton architecture. It is considered as the most essential activity in designing the whole system architecture because subsequent architecture design activities are performed based on the skeleton architecture.

A skeleton architecture is defined as a stable and core structure of a target system, and it specifies all the essential structural elements and their relationships of the system. The skeleton architecture should be well-defined to reflect the characteristics of the system and to accommodate view-specific design elements. Furthermore, it should be defined to support the architectural design decisions that realize the non-functional requirements of the system.

A skeleton architecture is characterized by the following features.

✦ **Specifying Structural Elements of the Architecture**

A software architecture consists of coarse-grained structural elements such as tiers, layers, partitions, and external services. The skeleton architecture defines all the essential structural elements and their connectivity.

✦ **Stability of the Structure and Connectivity**

A skeleton architecture becomes the fundamental structure of a target system like cornerstones of a building structure. Hence, a skeleton architecture should provide a high stability over the system lifetime. That is, it is hardly ever modified during the whole architecture design and its system implementation. Furthermore, it should be designed to be stable with a series of system maintenance.

✦ **Manifesting the Characteristics of Target System**

A skeleton architecture should be devised by reflecting and manifesting the characteristics of the target system. Each system reveals the system-specific characteristics in terms of the structure property, interaction, computational behavior, quality requirements, and constraints. A skeleton architecture should be designed to align with the system-specific characteristics.

✦ **Specifying Placeholders for further Architectural Decision Elements**

The structural layout of a skeleton architecture specifies various placeholders that can accommodate further architectural design decisions such as functional components, data components, workflow-type controllers, plugin components, and external services. The skeleton architecture should also be designed to accommodate architectural design decisions for non-functional requirements as well as functional requirements.

Architecture Styles for Deriving Skeleton Architecture

Skeleton Architecture can effectively be designed by utilizing architecture styles. An architectural style is a general and reusable solution to a family of commonly occurring structures and their properties. Each style specifies its components and connectors which form the fundamental structure of the system, as shown in the following examples.

- **Client-Server architecture style → Client tier, Server tier, and Invocation by Client**
- **Pipe-and-Filter architecture style → Filter, Piple, and Data Streaming**

The benefit of utilizing architecture styles is the high quality of the resulting skeleton architecture and the increased design productivity.

The skeleton architecture can systematically be designed with the steps shown in Figure 5-1.

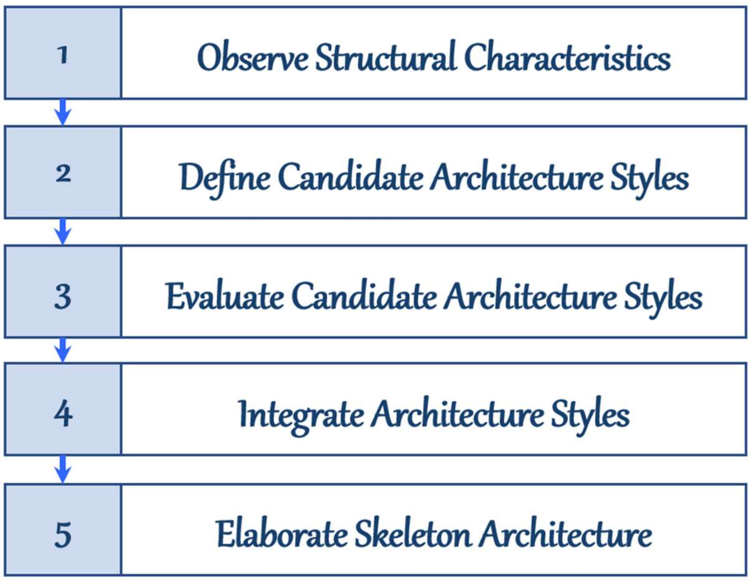
1
Observe Structural Characteristics
2
Define Candidate Architecture Styles
3
Evaluate Candidate Architecture Styles
4
Integrate Architecture Styles
5
Elaborate Skeleton Architecture

Figure 5-1. Steps in Activity of Skeleton Architecture Design

Step 1 is to make observations on the structural characteristics of a target system. The structural observations become the basis for performing subsequent steps of the activity.

Step 2 is to identify appropriate architecture styles for the target system by referring to the observations acquired in step 1.

Step 3 is to evaluate the applicability of each candidate architecture style by considering the applicable situation, pros, and cons of the architecture style.

Step 4 is to integrate the set of selected architecture styles into a skeleton architecture.

Step 5 is to elaborate the resulting skeleton architecture with the description on the structural property, strengths, and limitations of the architecture.

The input artifacts to designing the skeleton architecture are listed in the following.

✦ **Software Requirement Specification (SRS)**

The SRS of a target system specifies the functional and non-functional requirements of the system and hence the SRS should become the basis for comprehending the system characteristics and deriving the skeleton architecture.

✦ **System Context Model**

The context model of a target system provides the boundary context, functional context, information context, and behavior context of the system. Hence, it should be utilized as the basis for deriving the applicable architecture styles.

The output artifact of this activity is the following.

✦ **Design of Skeleton Architecture**

The result of applying this activity is a well-defined skeleton architecture that consists of structural elements and connectors.

5.2. Step 1. Define Candidate Architectural Styles

This step is to define a set of candidate architecture styles of a target system. We perform the following three tasks to identify the right candidate architecture styles.

This step is to observe the architectural characteristics of the target system by performing the following two tasks. The architectural characteristics become the basis for identifying appropriate architecture styles, from which its skeleton architecture is constructed.

5.2.1. TASK 1. OBSERVE ARCHITECTURAL CHARACTERISTICS

This task is to observe architecture-relevant elements of the system. The observations become the basis for choosing architecture styles and eventually the skeleton architecture.

The observation can be made by considering the following aspects. Describe the acquired observations in textual form and include any necessary figures in the description.

✦ **Structural Elements implied in the SRS**

Observe structural elements implied in the SRS such as tiers, sub-systems, layers, partitions, and components. The structural elements often suggest the applicable architecture styles.

✦ **External Computing Entities interacting with the System**

Observe external computing entities implied in the SRS such as external systems, cloud services, or micro services implied in the SRS. The external computing entities often suggest the applicable architecture styles or structural components of some architecture style.

✦ **Hardware Devices of the System**

Observe hardware devices connected to the system such as sensors, actuators, cameras, and IoT devices. The connected devices often suggest structural components of some architecture style such as *Hardware Abstraction Layer* of a layered architecture style.

✦ **Types of the System Functionality**

Observe the types of system functionality described in the SRS such as intensive data manipulation, computation with high complexity, intensive user-interaction, intensive collaboration among users or components. The functionality types often suggest applicable architecture styles or structural components of some architecture style.

✦ **Types of the Computation Behavior**

Observe the types of computation behavior implied in the SRS such as series of data transformation, streaming-type of data transfers, event-based invocation, central orchestration of various activities, peer-to-peer collaboration, and distributed processing. The behavior types often suggest applicable architecture styles or structural components of some architecture style.

✦ **Observe any other characteristics of the system**

Observe any other characteristics of the system beyond the aspects discussed above.

✦ **Observe NFRs affecting the Architecture**

This task is to observe NFRs specified in the SRS and identify aspects of the NFRs that should be reflected in the design of the skeleton architecture. In general, NFRs do not directly or strongly suggest specific architecture styles and its resulting skeleton architecture. However, there can be a rare case that a NFR can well be realized by applying a specific architecture style.

For example, a high level of availability for a cloud service system would suggest a an architecture style for load-balancing among replicated servers. The dispatcher architecture style or broker architecture style could be suggested to provide the required availability.

5.2.2. Task 2. Identify Types of the System

This task is to identify the type(s) of a target system. Note that a system may belong to more than one system type when the system is characterized by the features of multiple system types.

This task is motivated by the fact that an effective way to derive the system characteristics is to understand the types of the system. There does not exist a standard taxonomy of system types, but we can identify a set of system types that are often stated in literatures and referred in industry projects. We summarize some of the commonly system types.

✦ **Tiered System**

This type of system is characterized by the structural layout that employs a number of tiers where a tier is a physical computing device. A tier of a system performs a specific high-level role of the system and interacts with other tiers of the system. The structure of a system with multiple tiers is shown in Figure 5-2

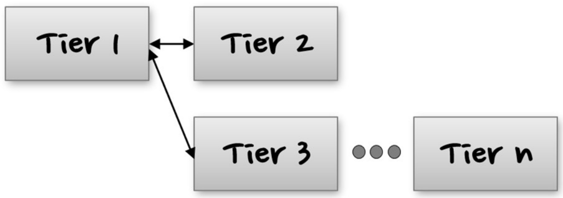

Figure 5-2. System with Multiple Tiers

For example, an eCommerce system may consist of a mobile client tier for end-users and a server tier that is managed by system administrators. The role of the client tier is distinct from the role of the server tier, but they interact each other in terms of information exchanges and service invocations.

✦ **Layered System**

This type of system is characterized by the structural layout that employs a number of layers in a tier and order the layers vertically. Each layer provides a virtual machine, i.e., abstraction, to its immediate upper layer. A system with multiple layers that are organized in a vertical hierarchy is shown in Figure 5-3.

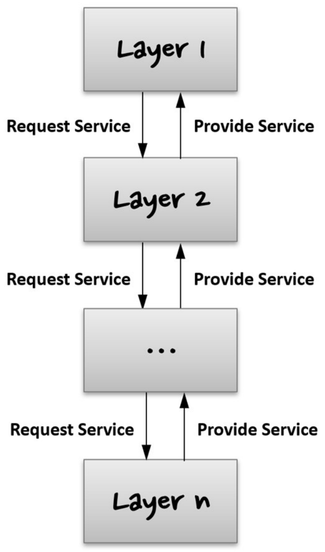

Figure 5-3. System with Multiple Layers

For example, an embedded system typically consists of a presentation layer, logic layer, data access layer, physical database, hardware abstraction layer, and hardware layer.

✦ **Load-Balancing System**

This type of system is characterized by the structural layout that includes multiple instances of replicated server and a load balancer that balances the computation load among the replicated servers. The structure of a system that balances the load among replicates is shown in Figure 5-4

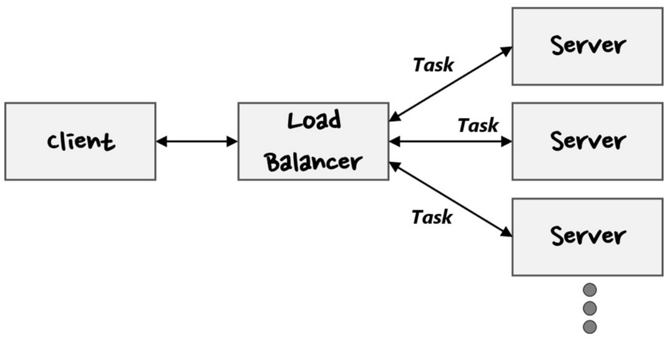

Figure 5-4. System with Replicated Servers

The benefits of load-balancing systems include the high level of performance, reliability, and availability. For example, a cloud service system can be configured with a number of replicated cloud servers that are geographically distributed to provide a high availability and reliability.

✦ Service-based System

This type of system is characterized by the structural layout that includes external services and a client application invoking the services over network. The structure of a service-based system is shown in Figure 5-5.

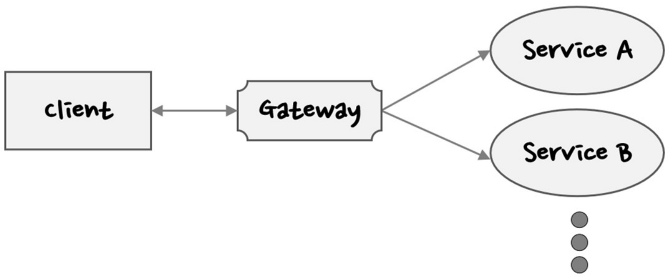

Figure 5-5. System invoking External Services

A key benefit of utilizing external services is the reduced effort of development and maintenance. However, the service-based systems have drawbacks of the propagation of service faults and the network latency in invoking the services.

For example, a personal information management (PIM) system invokes external services for retrieving traffic congestion information, personal schedules, and contact information.

✦ Data Flow System

This type of system is characterized by the structural layout that includes multiple data manipulation components and supports intensive data flows among the components. The structure of a system with data manipulation components and data flows is shown Figure 5-6.

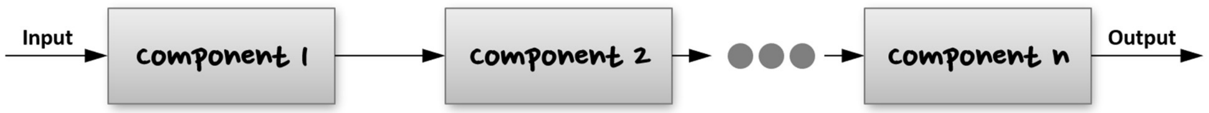

Figure 5-6. System with Data Manipulation Components

For example, a conventional compiler for a programming language consists of several data manipulation components. A tokenizer component reads a source program such as C program, divides the input into individual tokens and return them. Its next component, parser, reads the streams of tokens, convert them into an internal structure such as abstract syntax tree. The next component, linker, reads the parsed object files, combine them with libraries, and generates an executable file.

✦ Data Centered System

This type of system is characterized by the structural layout that includes a large volume of data repository and let multiple data accessors, i.e., applications, sub-systems or components, share the data repository efficiently. The structure of a data-centered system is shown in Figure 5-7.

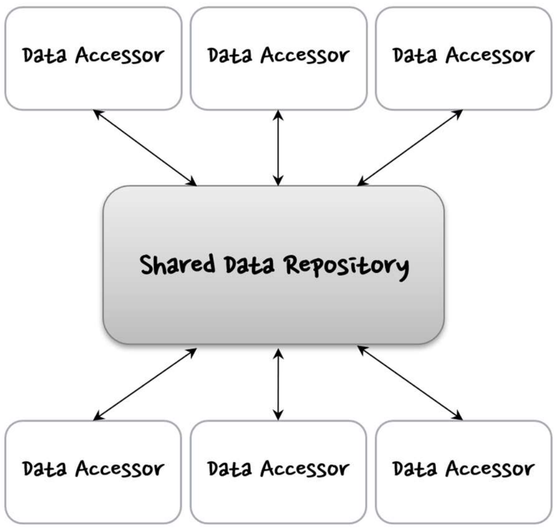

Figure 5-7. Multiple Data Accessers sharing the Data Repository

For example, a banking system consists of multiple applications including Deposit Management application, Loan Processing application, Foreign Exchange application, and Credit Card processing application. These applications share a moderate amount of account and transaction data.

✦ Event-based System

This type of system is characterized by the structural layout that consists of event sources and event sinks and it is also characterized by the behavior of event occurrences and invocation of the event-specific handler. The communication between event emitter and event sinks is asynchronous, providing benefits of parallelism, modularity, and extendibility. The structure of a event-based system is shown in Figure 5-8.

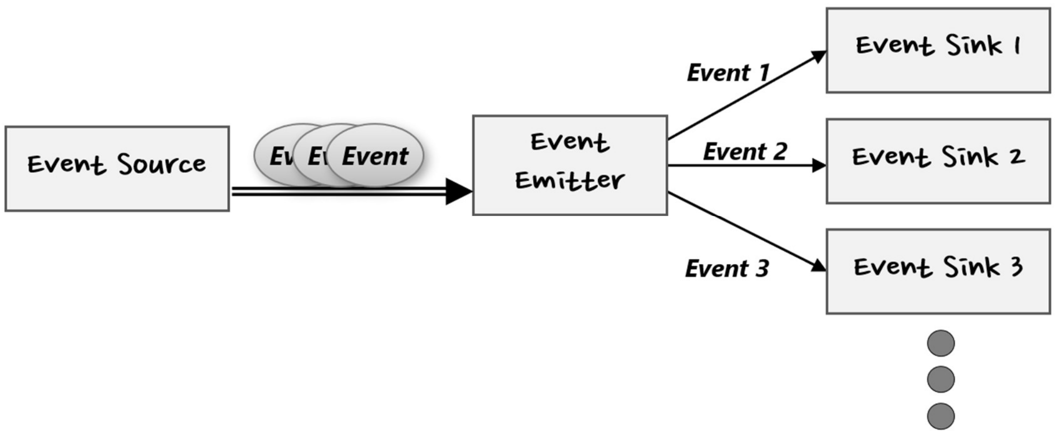

Figure 5-8. System with Event Sources and Event Sinks

For example, a smart home platform acquires a video stream from an attached surveillance camera, recognize a visitor by utilizing a people recognition machine learning model, generates an event if the visitor is not registered in the system, and transmits the event. Then, an event sink, i.e., an event handler, can handle the event accordingly.

✦ **Adaptive System**

This type of system is characterized by the structural layout that handles the variability on the functionality, datasets, and behavior of the system. This type of system is designed with design methods and architectural schemes realizing the open-closed principle. The structure of an adaptive system is shown in Figure 5-9.

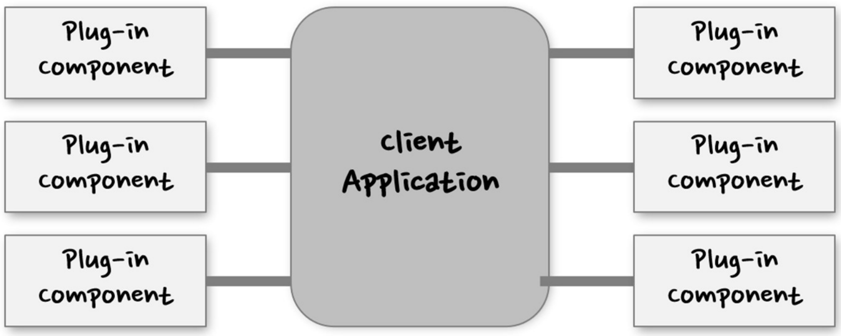

Figure 5-9. System providing Adaptibility with Plug-in Components

For example, a software development platform such as Eclipse can be designed with plug-in components to support the dynamic adaptability of the functionality and behaior.

✦ **Other Types of System**

There can be other types of system beyond the system types discussed above.

By referring the description of system types, we now identify the applicable types of a target system. The target system may reveal the characteristics of multiple system types. Consider the following examples of identifying the system types.

✦ **Example) System Types for Car Rental Management System**

The requirement specification for developing Car Rental Management System (CRMS) is given in Appendix **Error! Reference source not found.**. Based on the given requirements and the domain knowledge, we can infer the characteristics of the system types.

- **Multiple Tiers specified in the SRS → Tiered System**

  The SRS of the system specifies a deployment with a mobile app for customers, a system for Rental Center staffs, and a system for headquarter staffs.

- **Characteristics of MIS-type Systems → Layered System**

  Each tier in the system reveals the characteristics of conventional MIS-type systems that consist of layers for user interface, business processes, and persistent datasets.

- **Reliability and Availability of Systems required → Load-balancing System**

  Both the Rental Center system and Headquarter system should highly be reliable and available for customers and staffs. This type of requirements is well be achieved in load-balancing systems.

✦ **Example) System Types for Digital Mirror System**

The requirement specification for developing Digital Mirror System (DMS) is given in Appendix 14. Based on the given requirements and the domain knowledge, we can infer the characteristics of the system types.

- **Digital Mirror Client and Server → Tiered System**

  The SRS of the system specifies a deployment with two tiers; Mirror Tier and Machine Learning Tier.

- **Characteristics of MIS-type Server → Layered System**

  Each tier in the system reveals the characteristics of conventional MIS-type systems that consist of layers for user interface, business processes, datasets, and hardware abstraction layer.

- **Events generated from Touch Screen, Microphone, and Camera → Event-based System**

  The system acquires inputs from the touch screen, microphone, and camera, and converts them into events. Then, it invokes event-specific functionality.

- **Microservices for Weather, Traffic, and Calendar → Service-based System**

  The system utilizes microservices to retrieve weather information, traffic condition, and personal calendar.

5.2.3. TASK 3. DERIVE CANDIDATE ARCHITECTURE STYLES

This task is to list the set of architecture styles that are applicable to each system type and choose the most applicable architecture style from the applicable styles. The selected architecture style becomes a candidate architecture style for the target system.

✦ Listing Applicable Architecture Styles

Table 5-1 shows applicable architecture styles for each type of system. Note that the list of application architecture styles is not meant to be complete; there can be other applicable architecture styles.

Table 5-1. Architecture Styles for System Types

System Type	Architecture Styles
Data Flow System	Batch Sequential Architecture Style
	Pipe and Filter Architecture Style
Data Centered System	Shared Repository Architecture Style
	Active Repository Architecture Style
	Blackboard Architecture Style
Tiered System	Client Server Architecture Style
	N-Tier Architecture Style
	Peer to Peer Architecture Style
Layered System	Layered Architecture Style
	Model View Controller (MVC) Architecture Style
	Variation of MVC Style
Load-Balancing System	Broker Architecture Style
	Dispatcher Architecture Style
	Mas. ter Slaver Architecture Style
	Edge-based Architecture Style
Service-based System	Microservice Architecture Style
	Service-Oriented Architecture (SOA) Style

Event-based System	Event-driven Architecture Style
	Publish and Subscribe Architecture Style
	Sensor Controller Actuator Architecture Style
Adaptive System	Microkernel Architecture Style
	Blackboard Architecture Style

✦ **Choose Candidate Architecture Styles**

Using the table of applicable architectural styles, we choose the most appropriate architecture style for each determined system type as shown in Figure 5-10.

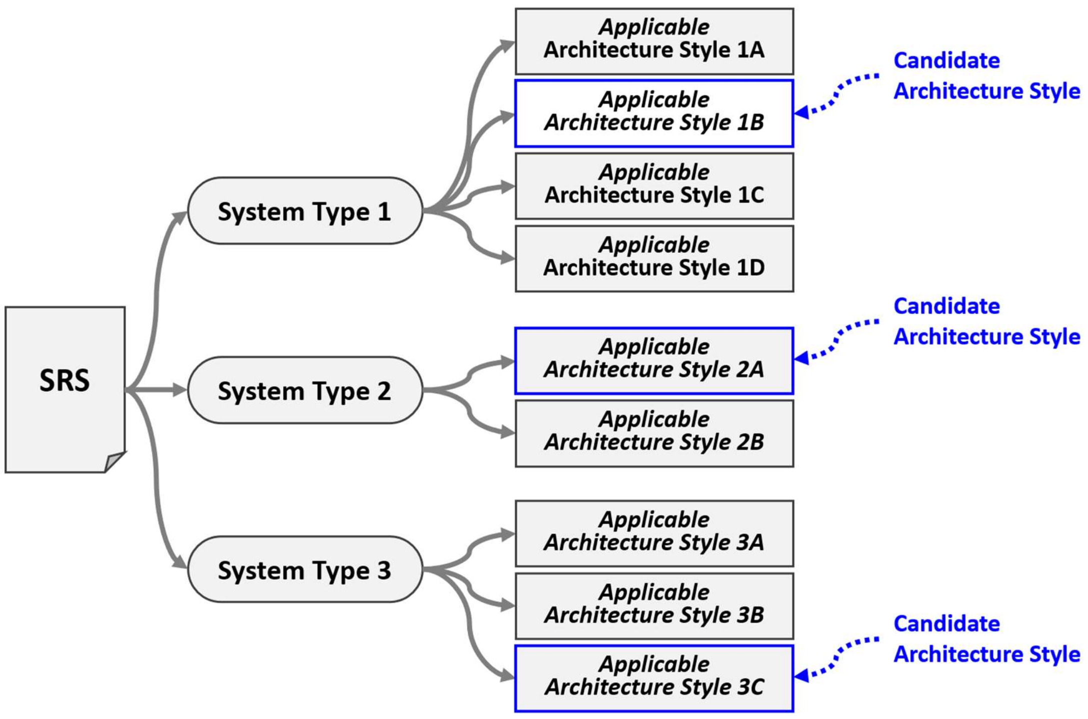
SRS
System Type 1
System Type 2
System Type 3
Applicable Architecture Style 1A
Applicable Architecture Style 1B
Applicable Architecture Style 1C
Applicable Architecture Style 1D
Applicable Architecture Style 2A
Applicable Architecture Style 2B
Applicable Architecture Style 3A
Applicable Architecture Style 3B
Applicable Architecture Style 3C
Candidate Architecture Style
Candidate Architecture Style
Candidate Architecture Style

Figure 5-10. Defining Candidate Architecture Styles

In the figure, three system types are identified by referring to the SRS of a target system. Then, we choose the most suitable architecture style for each system type. In the case of *System Type 1*, four architecture styles are applicable and the architecture style *1B* is selected as the most suitable architecture style. That is, the architecture style *1B* becomes a candidate style. Similarly, architecture styles *2A* and *3C* are also selected as candidate. As the result, the architecture styles *1B, 2A, and 3C* become the candidate architecture styles of the target system.

For example, Digital Mirror System is characterized by the four system types, and we choose a candidate architecture style for each system type as shown below.

- **Tiered System → Client-Server Architecture Style**
- **Layered System → Layered Architecture Style**
- **Event-based System → Event-Driven Architecture Style**
- **Service-based System → Microservice Architecture Style**

5.3. STEP 2. EVALUATE CANDIDATE ARCHITECTURAL STYLES

This step is to evaluate the applicability of candidate architecture styles to ensure that each candidate architecture style is well suitable. We apply the following criteria for evaluating each candidate architecture style.

✦ **Applicable Situations of Style**

This criterion is to check if the target system reveals the applicable situations of a candidate architecture style.

✦ **Benefits of Style**

This criterion is to check if the target system would be advantaged by the benefits of the style.

✦ **Drawbacks of Style**

This criterion is to check if the target system would experience any risks due to the drawbacks of the style

By evaluating candidate architecture styles using these criteria, we can confidently finalize the selection of architecture styles to apply.

5.3.1. TASK 1. EVALUATE APPLICABLE SITUATIONS OF CANDIDATE ARCHITECTURE STYLE

This task is to evaluate if the applicable situations of the style match to the situations of the target system. An architecture style is only applicable when its situations match well to the system situations.

An architecture style is defined with a set of situations where the style can be effectively applied. If a target system reveals all or most of the applicable situations, the style can be applied.

✦ **Example of Pipe-and-Filter architecture style**

Pipe-and-Filter architecture style has the following situations.

- **The system has a functionality of data manipulation. That is, the system consists of functional components that mainly manipulate datasets.**
- **There is a need for transferring data between components. That is, one component generates its output which is consumed by another component.**
- **The data between components is transferred in a streaming mode, rather than the whole batch.**
- **Others**

If a target system reveals all or most of the applicable situations, then the style can be applied in defining the skeleton architecture. If a target system revealing a feature that contradicts the situations of a style, the style cannot be applied. For example. data manipulating components of a system need to transfer the datasets in the whole batch, this style cannot be utilized. This is because the data streaming paradigm of pipe-and-filter architecture style forbids the data transfer in batch mode.

✦ Table for Evaluating the Situations

We use the following table for evaluating the applicable situations.

Table 5-2. Table for Evaluating Applicable Situation

Applicable Situations of Style	Match	Situations of Target System

- **Applicable Situations**

  List the applicable situations of the style.
- **Situations of Target System**

  Describe the situations of the target system from the viewpoint of structural properties and requirements.
- **Match**

  Enter a degree of matching the situations of the style and the target system by entering one of three symbols.
  - O for High Matching
  - Δ for Partial Matching
  - X for No or Low Matching

5.3.2. TASK 2. EVALUATE BENEFITS OF CANDIDATE ARCHITECTURE STYLE

This task is to evaluate if the target system would be advantaged by the benefits provided by the style. An architecture style should be applied only when its benefits contribute to the target system significantly.

✦ **Example of Pipe-and-Filter Architecture Style**

Pipe-and-Filter architecture style provides the following benefits;

- **Filter element is used to effectively model data components with input and output.**
- **Pipe element is used to effectively model the data streaming among data components.**
- **The parallel processing can be modeled with multiple filters running in parallel.**

  A filter element can have output pipes that are consumed by multiple filter elements, and a filter element can have multiple input pipes.
- **Others**

If a target system requires the data components and data streaming, the system can be designed effectively. If the system needs to run components in parallel, the system can take the advantage.

If the system should be configured flexibly for different consumer bases or servicing regions, the system can take the advantage.

✦ **Table for Evaluating Benefits of Style**

We use the following table to check if the benefits of the style can be applied to the target system.

Table 5-3. Table for Evaluating Benefits

Benefits provided by Style	Match	Advantages to Target System

- **Benefits provided by Style**

  List the potential benefits provided by the style.
- **Advantages to Target System**

  Describe how the target system can be advantaged by the benefits of the style.
- **Match**

  Enter a degree of taking the advantages on the target system from the benefits provided by the stye.
  - O for High Advantage

- Δ for Partial Advantage
- X for No or Low Advantage

5.3.3. Task 3. Evaluate Drawbacks for Candidate Architecture Style

This task is to evaluate if the target system would experience any risks or negative effects due to the drawbacks of the style. An architecture style should be applied only when its drawbacks do not cause significant risks or negative effects to the target system.

✦ **Example of Pipe-and-Filter Architecture Style**

Pipe-and-Filter architecture style does not have significant risks; rather it has some limitations on its applicability.

- **There must be an interface compatibility between every pair of filter and pipe.**
- **This style can only be applied to systems with data transfers in streaming mode. Hence, systems with data transfers in batch cannot be designed with this style.**

✦ **Table for Evaluating Drawbacks of Style**

We use the following table to check if the drawbacks can be mitigated in some way for the target system.

Table 5-4. Table for Evaluating Drawbacks

Drawbacks of Style	Match	Mitigating Drawbacks

- **Drawbacks of Style**
  List the limitations and disadvantages of the style.
- **Mitigating Drawbacks**
  Describe how the drawbacks can effectively be mitigated on the target system.
- **Match**
  Enter a degree of effectively mitigating and so resolving the drawbacks on the target system.
  - O for Effective Mitigation
  - Δ for Some Mitigation
  - X for No or Low Mitigation

5.3.4. Task 4. Decide Applicability

This task is to decide the applicability of each style on the target system. The decision is based on the degree of matches in the three evaluation tables. The decision of accepting each architecture style can be done with logical reasoning or weight system.

✦ **Decision based on Logical Reasoning**

Consider the entries in the evaluation tables. Each table would have a small number of entries.

- **If most of the entries in the evaluating tables have 'O', then accept the style.**
- **If most of the entries with 'O' or 'Δ', accept the style.**
- **If number of entries with 'X' is considerable high, do not accept it.**
- **If the distribution of entries with 'O', 'Δ', or 'X' is in the intermediate level between the acceptance and rejection criteria, make a careful decision by reviewing the evaluation results in depth.**

✦ **Decision based on Weight System**

Let count(O) be the total number of matches with 'O' in the tables. And let count(Δ) and count(X) be defined accordingly. Then, assign a weight for each count().

- **1 for count(O)**
- **0 for count(Δ)**
- **-1 for count(X)**

Let *numEntries* be the total number of entries in all three evaluation tables. Compute the evaluation score, *score*, as the following.

- **score ← [(weight * count(O)) + (weight * count(Δ)) + (weight * count(X)))] / numEntries**

Then, the range of *score* will be between -1 and 1 inclusively.

- **If the *score* is 1, choose the architecture style.**
- **If the *score* is near 1, choose the architecture style.**
- **If the *score* is between 0 and 1, choose the architecture style with careful justification.**
- **If the *score* is 0 or near 0, do not choose the architecture style with careful justification.**
- **If the *score* is lower than 0, do not choose the architecture style.**

✦ **Considering on count('X') on Drawback**

In addition to the decision criteria, the value of count('X) should carefully be examined. The negative impact of a drawback ranges from being minor to being significant.

- **Minor Drawback**

  A drawback may be a minor limitation that the application of the architecture style would not create any significant risks. We can largely ignore the minor drawbacks even if they have 'X'.

- **Major Drawback**

  A drawback may be a major limitation that the application of the architecture style would create significant risks to the system operations. Then, we should not apply the architecture style regardless of the evaluation results on the applicable situation and benefits.

5.4. Step 3. Integrate Architecture Styles

This step is to integrate all the selected architecture styles. The integration will result in a definition of the skeleton architecture for the target system as in Figure 5-11.

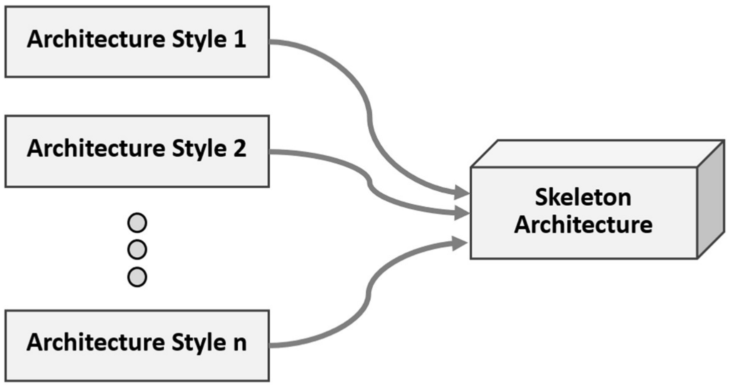

Figure 5-11. Architecture Styles integrated into Skeleton Architecture

✦ **Principles of Integrating Architecture Styles**

- **Integration using Common Components**

  An architecture style is defined with components and connectors. Identify components with a same role between two architecture styles to integrate. A component in one architecture style could have a same role of some component in the other architecture style, although the names of two common components could not be the same. In some cases, one component in an architecture style may map to multiple components in the other architecture style.

- **Example of Integrating of Shared Repository and MVC architecture style**

  The two styles are integrated around two common components as shown in Figure 5-12.

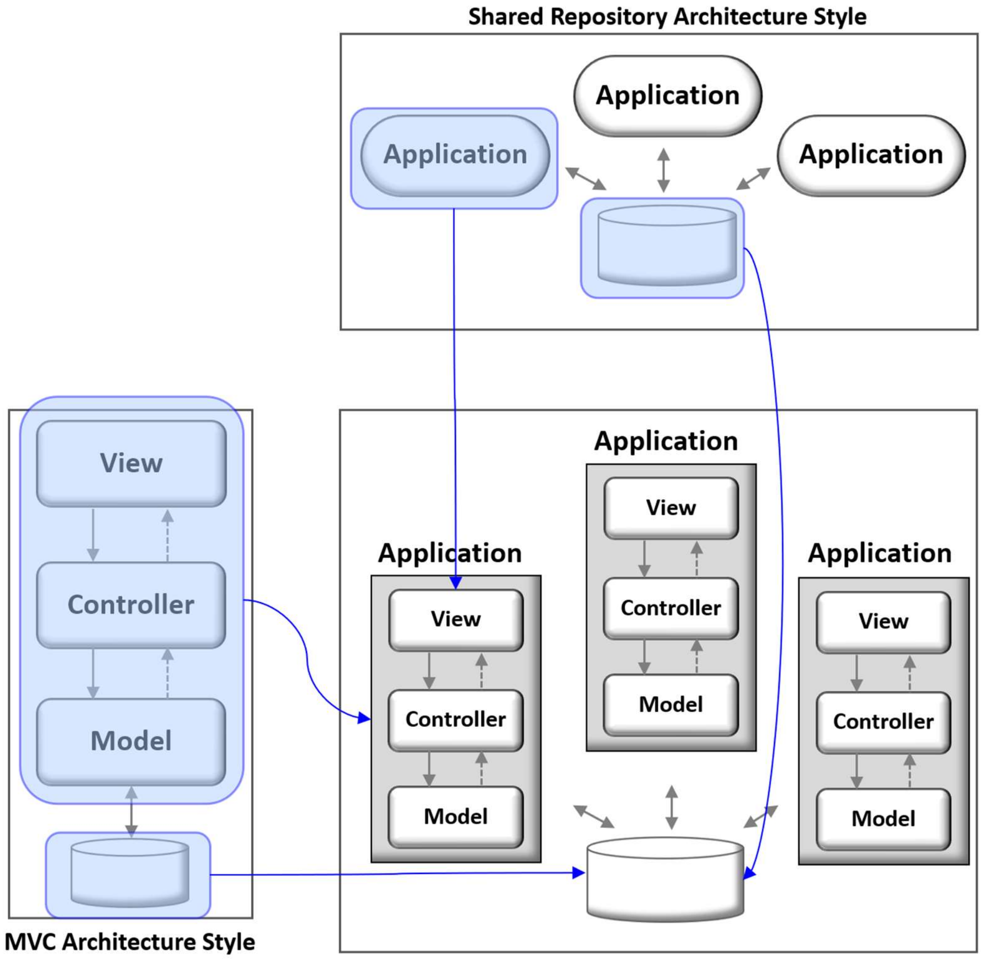

Figure 5-12. Integrating Shared Repository and MVC Architecture Styles

- One to Many mapping

  The *Application* component of Shared Repository architecture style maps to the set of three layers in MVC architecture style, i.e. View, Control, and Model.

- One to One mapping

  The *database* components of the two styles have one-to-one mapping.

- **Integration using Connectors**

  There can be a case where the common components between two architecture styles do not exist. In this case, define connectors to enable the interactions between elements in the two styles.

- **Handling Conflicts among Architecture Styles**

  There can be a case where the two architecture styles to integrate are not compatible or conflict each other. If the conflict cannot be resolved property, the integration cannot proceed and the list of selected architecture styles should be reviewed and refined.

5.4.1. Task 1. Apply Architecture Styles with Tiers

This task is to instantiate tiers needed by the target system. A tier in software architecture is the largest unit of structural element, and we apply the tier-based architecture first in defining the initial version of skeleton architecture.

The architecture styles in this category are listed in the following.

- **Client Server architecture style**
- **N-tier architecture style**
- **Peer-to-Peer architecture style**

Load-balancing architecture styles also require one or more tiers with specific roles, and they are listed in the following.

- **Broker architecture style**
- **Dispatcher architecture style**
- **Mas. ter Slave architecture style**
- **Edge-based architecture style**

For the target architecture, we first determine the number of tiers required in the system by by considering the requirement specification. Then, we assign an application-intrinsic and descriptive name for each tier. For example, Client Server architecture style was selected for Smart Mirror System, and the result of applying the style is shown in Figure 5-13.

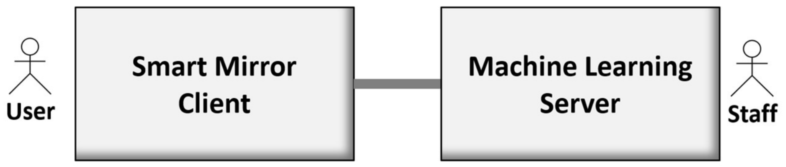

Figure 5-13. Client and Server Architecture Style applied to Smart Mirror System

As shown in the figure, the client tier of the style is instantiated as Smart Mirror Client which provides the system functionality to users. The server tier of the style is instantiated as Machine Learning Server which maintains the database for training purpose and generates machine learning models for the analytics of appearance age and face emotion.

As another example, consider a typical banking system which is used by bank customers, cashiers, managers. The system is typically designed with N-tier architecture style for high reliability, performance, and data security. The result of applying the style to a typical banking system is shown in Figure 5-14.

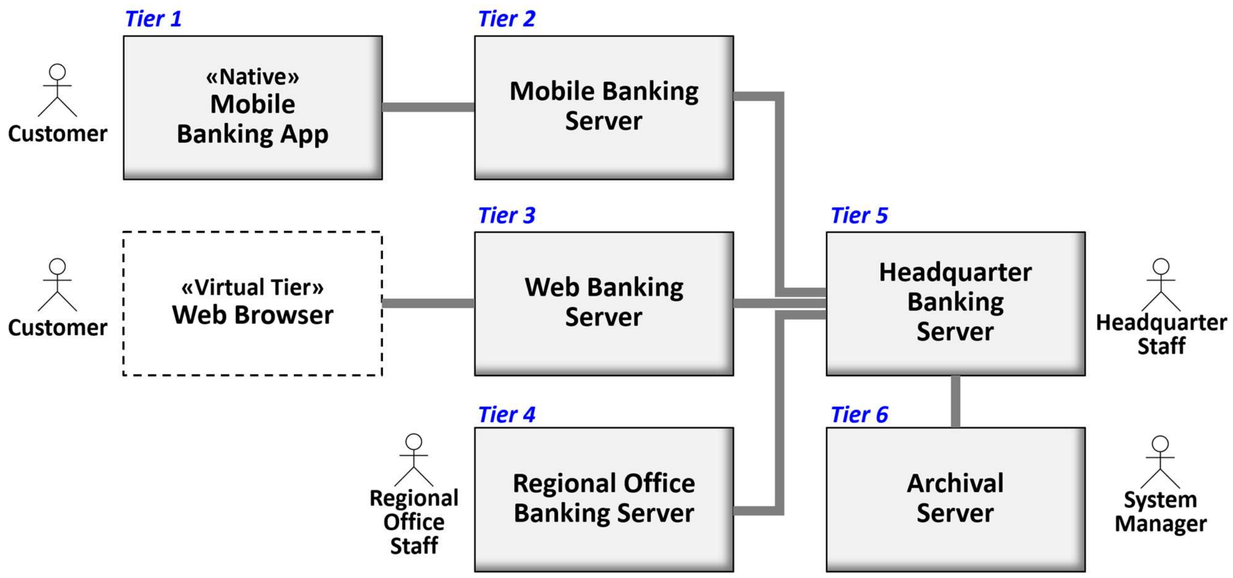

Figure 5-14. N-tier Architecture Style applied to Banking System

The banking system provides an internet banking service to customers with Mobile Banking App and web service with web browser. As shown in the figure, three tiers are to provide the internet banking.

- **Tier 1 as Mobile Banking App such as Android app or iOS app**
- **Tier 2 as Mobile Banking Server servicing the customers of mobile banking app**
- **Tier 3 as Web Banking Server servicing the customers using web browser for banking**

Note that Web Browser itself is not considered as an application tier. It is shown in the figure merely to depict that a web browser is used to access the web banking service.

There are two types of tiers that are used by banking personnel.

- **Tier 4 a Regional Office Banking Server providing the office-level banking functionality to Regional Office Staffs such as cashier.**
- **Tier 5 as Headquarter Banking Server providing the headquarter-level banking functionality to Headquarter Staffs such as Bank Manager and Director.**

The security of datasets for banking system is highly important, and hence banking systems typically maintain a separate archival server as shown in the figure.

- **Tier 6 as Archival Server maintaining the banking datasets in secure way.**

As shown in the examples, the number of tiers to instantiate and the roles of the instantiated tiers depends on the target system characteristics.

5.4.2. Task 2. Apply Architecture Styles with Services

This task is to apply architecture styles with network-based services. A network-based service like microservice is not treated as a tier of the target system. The service itself is not physically a part of the system even if it provides its specific functionality to the target system.

The architecture styles in this category are listed in the following.

- **Microservice architecture style**
- **Service-Oriented Architecture (SOA) style**

For each architecture style in this category, define the instances of services and their relevant elements. Then, define descriptive names for the services.

For example, Smart Mirror Client tier utilizes Microservice architecture style to retrieve weather forecast, traffic information, and personal calendar. Hence, we can define three services as shown in Figure 5-15.

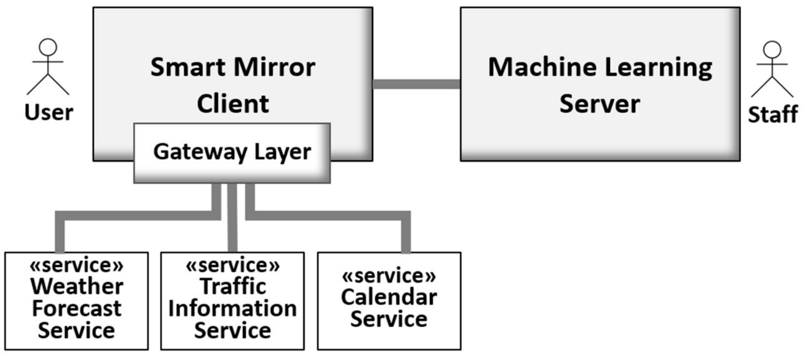
User
Smart Mirror Client
Gateway Layer
Machine Learning Server
Staff
«service» Weather Forecast Service
«service» Traffic Information Service
«service» Calendar Service

Figure 5-15. Microservice Architecture Style applied to Smart Mirror System

In the figure, Smart Mirror Client tier is shown to interact with three microservices. The Gateway Layer element in Smart Mirror Client provides the network-based connectivity for microservices and manages the faults of the services with configurations with multiple service providers for each type of service.

5.4.3. Task 3. Apply Architecture Styles with Layers

This task is to apply architecture styles with layers. A tier in a system typically consists of multiple layers where each layer plays a specific role in the system.

The architecture styles in this category are listed in the following.

- **Layered architecture Style**
- **MVC architecture style**
- **Variations of MVC architecture style such as;**
  - **H**ierarchical Model View Controller (HMVC)
  - Model View Adapter (MVA)
  - Model View Presenter (MVP)
  - Model View View Model (MVVM)

For example, Smart Mirror Client tier utilizes Layered architecture style and the result of applying the style is shown in Figure 5-16.

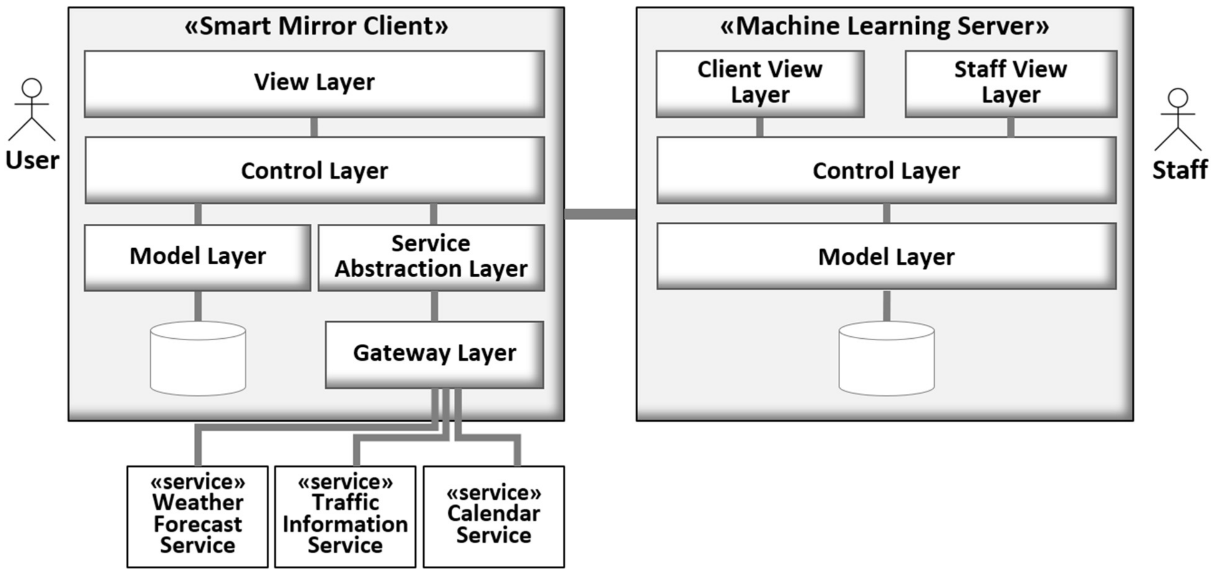
«Smart Mirror Client»
View Layer
User
Control Layer
Model Layer
Service Abstraction Layer
Gateway Layer
«Machine Learning Server»
Client View Layer
Staff View Layer
Staff
Control Layer
Model Layer
«service» Weather Forecast Service
«service» Traffic Information Service
«service» Calendar Service

Figure 5-16. Layered Architecture Style applied to Smart Mirror System

Smart Mirror Client is defined with the following layers and each layer provides its specific functionality.

- **View Layer**

  This layer provides a user interface for the users of the system.

- **Control Layer**

  This layer provides the core functionality of the client system including digital mirroring, public information retrieval, and software analytics.

- **Model Layer**

  This layer provides the functionality of accessing persistent objects managed by the client system.

- **Service Abstraction Layer**

  This layer is not an implementation of functionality, but a stable and standard interface for accessing external services such as weather forecast service. The components in Control Layer can access the external services using this interface.

- **Gateway Layer**

  This layer provides components that implement the interface defined in Service Abstraction Layer. Each component in this layer is an implementation of a specific external service.

- **Database**

  The Database element below Model Layer is not a software, but represents the physical database that is managed by the client system.

- **External Services**

  Each external service below Gateway Layer is an external system providing its specific service over network. For example, Calendar Service is a web service managing the calendars of users such as Google Calendar.

Smart Mirror Server is defined with the following layers and each layer provides its specific functionality.

- **Client View Layer**

  This layer provides a user interface for Smart Mirror Client system and the interaction between Control View Layer and the Client System is made over network.

- **Staff View Layer**

  This layer provides a user interface for staffs who manage the machine learning components and generated models.

- **Control Layer**

This layer provides the core functionality of the server system including the management of training sets machine learning models.

- **Model Layer**

  This layer provides the functionality of accessing persistent objects managed by the server.

- **Database**

  The Database element below Model Layer is not a software, but represents the physical database that is managed by the server system.

In the figure, each layer treats its immediate lower layer as a virtual machine and hence a layer accesses the functionality of the immediate lower layer through a pre-specified interface. For example, a component in Control Layer can access the database, i.e. instances of persistent classes, via the components defined in Model Layer

5.4.4. TASK 4. APPLY ARCHITECTURE STYLES WITH BEHAVIOR PATTERNS

This task is to apply architecture styles that define a specific runtime behavior of the system, i.e. the control flow of the system. The control flow is defined by specific types of components and connectors, which should be realized in the skeleton architecture.

The architecture styles in this category are listed in the following.

- **Batch Sequential architecture style**
- **Pipe-n-Filter architecture style**
- **Event-Driven architecture style**
- **Publish-and-Subscriber architecture style**

As an example, Event-Driven architecture style specifies three types of components and a control flow among them, as shown in Figure 5-17.

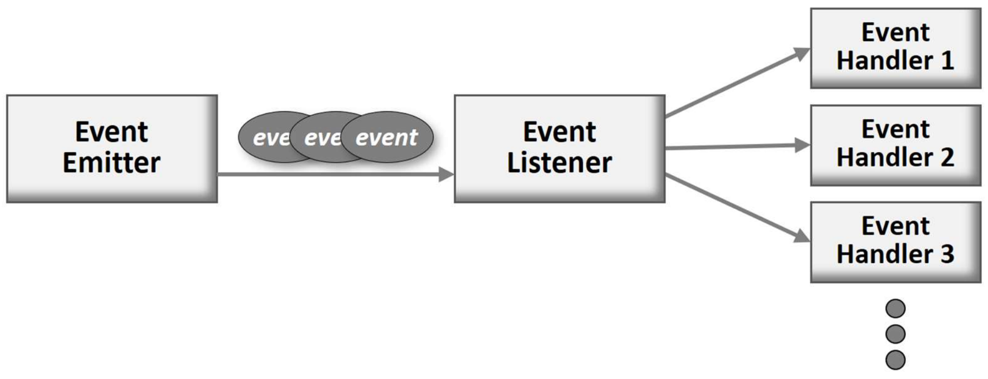

Figure 5-17. Runtime Behavior specified by Event-Driven Architecture Style

In the figure, three types of components are specified; Event Emitter, Event Listener, and Event Hander. The architecture style specifies the asynchronous interaction between Event Emitter and Event Listener. An event listener keeps waiting for an arrival of an event, which can be transmitted by event emitters. Upon arrival, the event listener locates an appropriate event hander for the received event, and the event listener component processes the event in its way.

As illustrated in this example, Event-Driven architecture style specifies three types of components inside Control Layer, that should be reflected in the skeleton architecture. Figure 5-18 shows the skeleton architecture with Event-Driven architecture style.

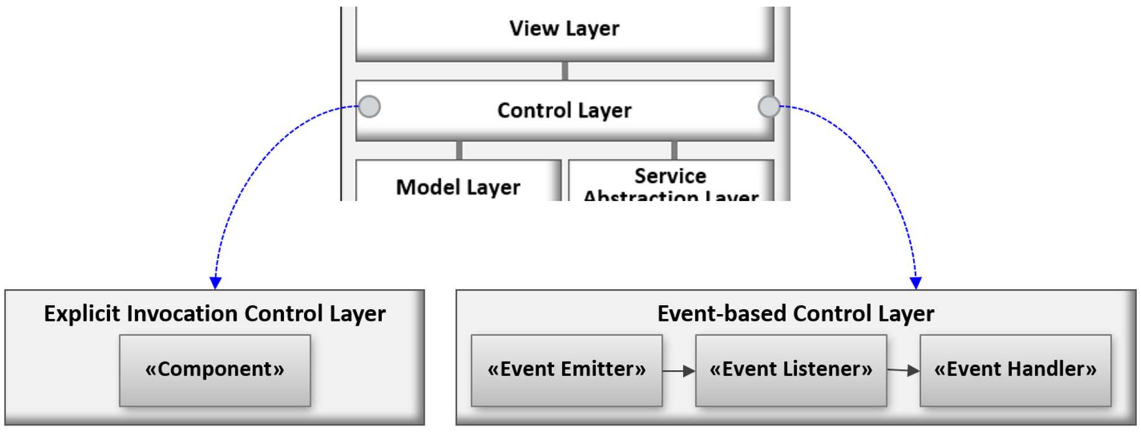

Figure 5-18. Skeleton Architecture reflecting Event-Driven Architecture Style

Control Layer of the initial skeleton architecture is refined into two types of control layer;

- **Explicit Invocation Control Layer**

  This layer contains the functional components that should run in explicit invocation manner.

- **Event-based Control Layer**

  This layer specifies the three types of components needed to run event-driven control flow, as indicated with stereotypes; «Event Emitter», «Event Listener», and «Event Handler». There can be multiple components implementing each stereotype.

5.4.5. TASK 5. APPLY ARCHITECTURE STYLES WITH VARIABILITY

This task is to apply architecture styles supporting architectural variability. Considering variability in system design becomes an essential strategy to increase the applicability and modifiability. The treatment of variability becomes more important for system-level software systems such as operating system, middleware software, platform software, and cloud service systems.

The architecture styles in this category are listed in the following.

- **Micro Kernel architecture style**
- **Microservice architecture style**
- **Blackboard architecture style**

Each style in this category defines a structural layout that specifies place holders for hosting the commonality and the variability of the system. And it also specifies a software scheme for binding variants for a variation point. The components and connectors used for specifying the structural layout and binding scheme should be realized in the skeleton architecture.

As an example, Micro-Kernel Driven architecture style specifies a variability-relevant structural schema as shown in Figure 5-19.

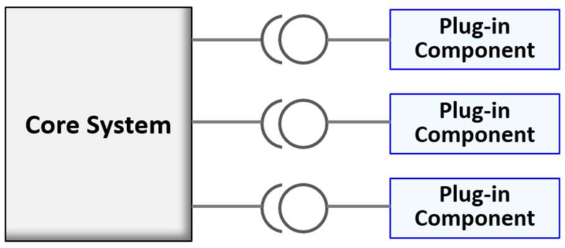

Figure 5-19. Components of Micro Kernel Architecture Style

The system with this architecture style consists of the following elements;

- **Core System**

  This is the essential functionality of the system that interacts with plug-in components. The functionality of the core system is fixed and hence is designed to be closed. The core system itself is incomplete; it becomes complete and functional only when all the required plug-in components are bound to the core part of the system.

- **Required Interface**

  A system with this style defines one or more required interfaces, that should be realized in implementing plug-in components. Each required interface defines a set of method signatures, that eventually specify some functionality required by the system. That is, a required interface specifies a functionality that can be implemented in various ways.

- **Plug-in Component**

  Each plug-in component is an implementation of a required interface in its specific way. Therefore, each plug-in component becomes a valid variant that can be bound to the required interface.

As illustrated in this example, architecture styles with variability is defined with common and variable parts of the system, and the open-closed principle should be applied to designing systems with these styles.

5.4.6. Task 6. Apply Other Architecture Styles

This task is to apply architecture styles which are not integrated yet. Since there is no complete taxonomy of architecture styles, there can be a style that does not belong to the categories listed above. Apply the components and connectors of such architecture style onto the skeleton architecture.

5.4.7. Task 7. Define Interaction Paths

An interaction path defines a specific communication means for components to interoperable. There are two types of interaction paths in skeleton architecture.

✦ **Style-defined Interaction Path**

Each architecture style defines a specific structural schema that specifies the style-specific interaction paths, i.e., Style-defined Interaction Path.

A skeleton architecture defined with architecture styles will inherit all the style-defined interaction paths as default. Examples of style-defined interaction paths are shown in the following.

- **Layered Architecture Style → Invoke-n-Reply path between 2 Adjacent Layers**
- **Pipe and Filter Architecture Style → Data Streaming path from Source Filter to Target Filter**
- **Event-driven Architecture Style → Event Emission Path from Event Source to Event Hander**

✦ **Disabling Style-defined Interaction Path**

A style-defined interaction path of a skeleton architecture should be not disabled. If so, the selected architecture style may not provide its intended structural and behavioral properties.

There may be a rare case that an architect decides to disable a style-defined interaction with a careful justification. If disabled, the resulting skeleton architecture should carefully be re-examined for validity.

✦ **Custom Interaction Path**

An architect may decide to define additional interaction paths that are not derived from the architecture styles. There can be a case where the set of style-defined interaction paths for a skeleton architecture would not be sufficient. The additional interaction paths provide an increased level of collaboration among the architectural components.

5.5. Step 4. Elaborate Skeleton Architecture

This step is to elaborate the resulting skeleton architecture in greater details using mostly textual description.

5.5.1. Task 1. Describe Architectural Strength

This task is to describe the architectural strength provided by the skeleton architecture. Describe first how the skeleton architecture successfully fulfills the structural aspects and requirements of the target system. Then, elaborate any additional architectural strength.

5.5.2. Task 2. Describe Architectural Limitation

This task is to describe the architectural limitation imposed by the skeleton architecture. Describe the structural aspects and requirements of the target system that are not fulfilled by the skeleton architecture with justification. Then, elaborate any additional architectural limitation.