.

.

#guideline# Layering

Summary

This guideline provides recommendations for partitioning the system into layers.

Main Description

Layering is one of the most commonly used approaches for structuring and decomposing systems. Layering logically partitions the system into sets of components with certain rules regarding how relationships can be formed between them. Layering provides a way to restrict **inter-subsystem dependencies**, with the result that the system is more loosely coupled and more easily maintained.

Consider the number and purpose of the layers carefully. Do not over-complicate the solution by defining more layers than are needed to meet the needs of the solution. More layers can always be added in the future to meet new requirements. Removing layers is not always as easy and may introduce risks into the project.

The criteria for grouping elements into layers follows a few patterns:

- **Visibility**: Elements may **depend only on** components in the same layer and the next-lower layer.
- **Volatility**:
  - In the highest layers, put elements that vary when user requirements change.
  - In the lowest layers, put elements that vary when the implementation platform changes (hardware, language,

operating system, database, and so forth).
    - Sandwiched in the middle, put elements that are generally applicable across wide ranges of systems and implementation environments.
    - Add layers when additional partitions within these broad categories help to organize the model.
- **Generality**: Abstract elements tend to be placed in the lower layers. If not implementation-specific, they tend to gravitate toward the middle layers.
- **Number of layers**. For a small system, three layers are typically sufficient. A three-layer architecture of **Presentation**, **Business**, and **Data layers** is very common in information systems. For a complex system, five to seven layers could be appropriate. For any degree of complexity, more than 10 layers should be viewed with suspicion that increases with the number of layers.

Failure to restrict dependencies according to the visibility criteria mentioned above can cause architectural degradation and make the system difficult to extend and maintain.

Allowable exceptions to the visibility rule include cases where components need direct access to lower-layer services beyond the next-lower layer. Make a decision about how to handle primitive services(@@**Architecture Mechanisms**) that are needed throughout the system, such as printing, sending messages, and so forth. There is little value in restricting messages to lower layers if the solution is to effectively implement call pass-throughs in the intermediate layers. This usage of a less strict rule on dependencies down through the layers is sometimes called a **Relaxed Layered Architecture** ([BUS96]).

- **Partitioning patterns**

  Within the top layers of the system, additional partitioning may help organize the model. The following guidelines for partitioning present different issues to consider:

    - **User organization:** Elements may be organized along lines that mirror the organization of functionality in the business organization (partitioning occurs along departmental or user role lines). This partitioning often occurs early in the design due to an existing enterprise model that is strongly partitioned according to the structure of the organization. This pattern usually affects only the top few layers of application-specific services and can often disappear as the design evolves.

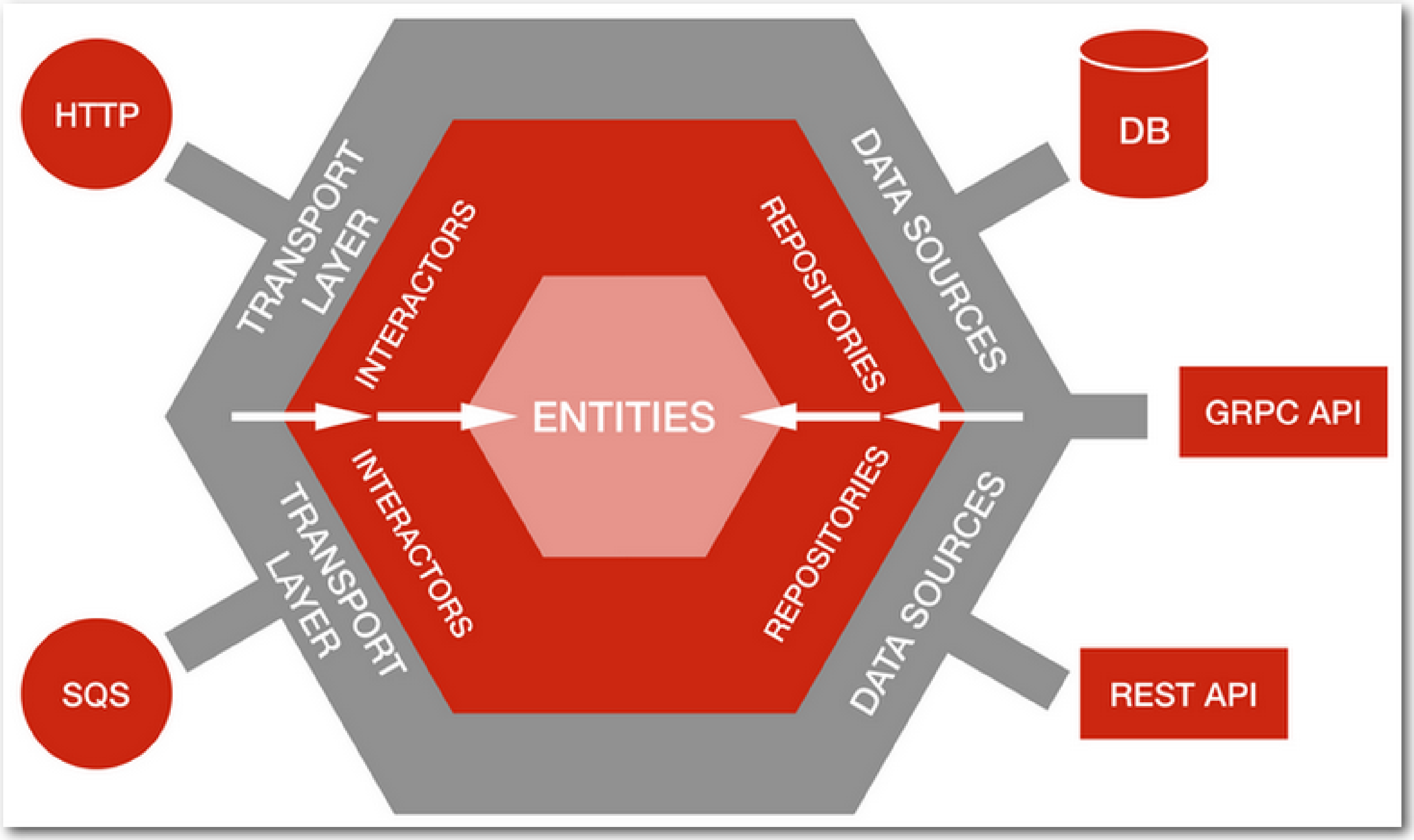
    - Partitioning along user-organization lines can be a good starting point for the model.
    - The structure of the user organization is not stable over a long period of time because business reorganizations occur; therefore, it is not a good long-term basis for system partitioning. The internal organization of the system should enable the system to evolve and be maintained independently of the organization of the business that it supports.

- **Areas of competence and skills:** Elements may be organized to partition responsibilities among different groups within the development organization. Typically, this occurs in the middle and lower layers of the system, and reflects the need for specialization in skills during the development and support of an infrastructure based on complex technology. Examples of such technologies include network and distribution management, database management, communication management, and process control, among others. Partitioning along competence lines may also occur in upper layers, where special competency in the problem domain is required to understand and support key business functionality. Examples include telecommunication call management, securities trading, insurance claims processing, and air traffic control, to name a few.
- **System distribution:** Within any of the layers of the system, the layers may be further partitioned horizontally, so to speak, to reflect the distribution of functionality.
    - Partitioning to reflect distribution of functionality can help you visualize the network communication that will occur as the system runs.
    - Partitioning to reflect distribution can also, however, make the system more difficult to change if the deployment model changes significantly.

- **Secrecy areas:** Some applications, especially those requiring special security clearance to develop or support, require additional partitioning according to security access privileges. Software that controls access to secrecy areas must be developed and maintained by personnel with appropriate clearance. If the number of people with this background on the project is limited, the functionality requiring special clearance must be partitioned into subsystems that will be developed independently from other subsystems, with the interfaces to the secrecy areas the only visible aspect of these subsystems.
- **Variability areas:** Functionality that is likely to be optional, and therefore delivered only in some variants of the system, should be organized into independent components that are developed and delivered independently from the mandatory functionality of the system.

@@

20200311 Ready for changes with Hexagonal Architecture - Netflix TechBlog, EN



.

.

#guideline# Modeling the Architecture

Summary

This guidelines provides some recommendations on the types of visual architecture models you may want to develop.

Main Description

When architecting a system, developing visual models of the architecture can be quite helpful as such models provide an excellent high-level picture of the system, its Components and their relationships. Specifically, you may find it useful to develop these three Unified Modeling Language (UML) diagrams:

- **Layer map** (represented as a **class diagram** using packages) that describes the upper-level layers of the architecture

- **Deployment diagram** (represented by a **deployment diagram**) that outlines the expected network topology
  - **Implementation and Deployment Viewpoint:**

    The implementation and deployment viewpoint shows how one or more applications are realized on the infrastructure. This comprises the mapping of applications and components onto artifacts, and the mapping of the information used by these applications and components onto the underlying storage infrastructure

  - **Technology Usage Viewpoint:**

    The technology usage viewpoint shows how applications are supported by the software and hardware technology: the technology services are delivered by the devices; system software and networks are provided to the applications. This viewpoint plays an important role in the analysis of performance and scalability, since it relates the physical infrastructure to the logical world of applications. It is very useful in determining the performance and quality requirements on the infrastructure based on the demands of the various applications that use it.

- **Key Abstractions diagram** (represented by a **class diagram**) that shows the key abstractions/components and important relationships among them.
  - **Application Cooperation Viewpoint:**

    The application cooperation viewpoint describes the relationships between applications components in terms of the information flows between them, or in terms of the services they offer and use. This viewpoint is typically used to create an overview of the application landscape of an organization. This viewpoint is also used to express the (internal) cooperation or orchestration of services that together support the execution of a business process.

  - **Application Usage Viewpoint:**

    The application usage viewpoint describes how applications are used to support one or more business processes, and how they are used by other applications. It can be used in designing an application by identifying the services needed by business processes and other applications, or in designing business processes by describing the services that are available. Furthermore, since it identifies the dependencies of business processes upon applications, it may be useful to operational managers responsible for these processes.

For more information on visual modeling, see **#guideline# Using Visual Modeling**.

More Information

- **#guideline# Using Visual Modeling**

.

-
-

-