Architecture

Alex Sverdlov
alex@theparticle.com

1 Introduction

Architecture is the high-level design of the system. It helps to visualize the important concepts or patterns of the system.

There are different kinds of architectures:

- Data Centric Architecture: the primary thing is the data, with all-sorts of components accessing the data store.
- Data Flow Architecture: the primary thing is how data flows from one module to the next—what transformations happen where and when.
- Call-and-Return Architecture: a hierarchy of components, the main program controls a few sub-components which in turn may control other components.
- Object Oriented Architecture: components encapsulate data, communicate via messages (method calls).
- Layered Architecture: project/system is partitioned into layers, often with lower levels handling more details than higher levels.
- Model-View-Controller: system is segregated into three primary modules/components: the Model, handling business logic and storage. The View: handling UI interactions and various ways of displaying model objects. The Controller: manager module that connects users to the model.

There are also high level questions that must be addressed, such as where and how are things controlled (does control pass from component to component, or do components operate asynchronously).

Also important consideration during architecture design is: how is data passed from component to component.

2 Designing Components

A component is a module with code. Often encapsulates business logic and hides data and implmenetation details. Often deployable as a unit, and represents a replaceable part of the system (can be replaced with another component with different implementation). Often accessed via interfaces.

In object oriented view, a component is a set of classes that collaborate in some way to solve a business problem.

Components should be open to extension, but close to modification. Easy to add functionality, but not change how it behaves.

Design components/classes to be deep. The API should hide complexity—if you notice that API is getting wide—a lot of API hiding relatively shallow implementation, rethink the architecture.