Software Architecture

PROF. DR. VALTER VIEIRA DE CAMARGO
ADVANSE/DC/UFSCAR (2018)

Definições

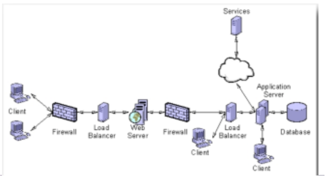
- **Software architecture** refers to the **high level** structures of a **software** system and the discipline of creating such structures and systems. Each **structure** comprises **software** elements, relations among them, and properties of both elements and relations
- Architecture is the **fundamental organization** of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.» [ANSI/IEEE Std 1471-2000]
- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [SEI]

Some "words" of Architecture

1. Quality Attributes ...
2. Difference between logical and physical architecture
3. Abstractions
   - Examples ?
4. Layers
5. MVC (Model-View-Controller)
6. Architectural Constraints
7. The concept of Architectural Deviation/Violation/Drifts
8. How to represent graphically ?
9. What about the mapping to source code ?

Why Software Architecture is importante ?

- Evoluções que vem ocorrendo na Engenharia de Software tem aumentado a importância de se pensar na Arquitetura do Software:
  - Escalabilidade
  - Distribuição
  - Segurança
  - Manutenibilidade
  - Disponibilidade

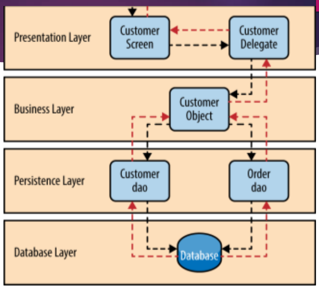


Architecture defines Structure

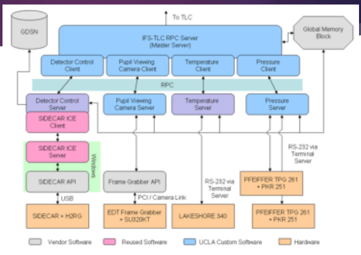
- Decomposition of system into :

1. Software Elements
   - Elements are captured as abstractions
   - Correspond to high level system modules or components
2. Component interfaces
   - External visible properties of elements
   - Describe element features exposed to others
   - Typically represent services provided to other elements
3. Component responsibilities
   - What does a component precisely do?
4. Relationships of elements
   - How do elements interact with others?

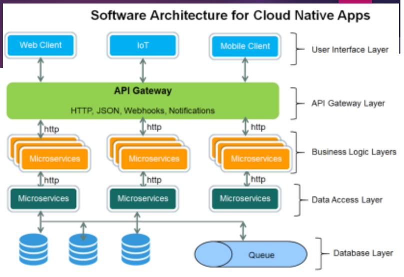
Examples



Examples



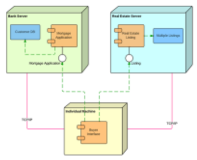
Examples



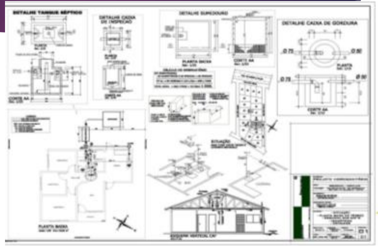
What is an "Explicit Architecture" ?

- An Explicit Architecture is having the System Architecture documented ! It is not only in the head of Software Engineers....
- Examples:

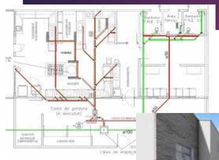
  Diagrams and Documents

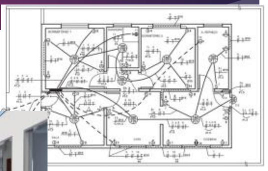


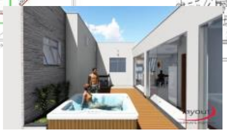
The concept of "Architecture Views"



The concept of "Architecture Views"

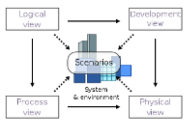






The concept of "Architecture Views"

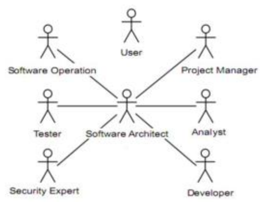
- To understand/build the architecture of a system is not enough having only one **view**
- That´s is the same with UML...... There are many diagrams that focus on different aspects of the software
- Philippe Kruchten have proposed the **"4 + 1" views:**
  - ***Development view***: components....
  - ***Logical view***: functionality.... State diagrams...
  - ***Physical view***: deployment.....
  - ***Process view***: runtime aspects....
  - ***Scenarios*** (5th):



The concept of “Architecture Views”

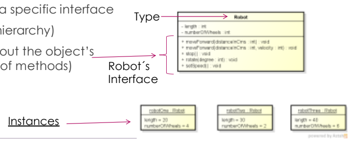
- ***Development view***: Illustrates a system from a **programmer's perspective** and is concerned with software management. This view is also known as the **implementation** view. This can be documented with UML Component Diagram.
- ***Logical view***: The logical view is concerned with the functionality that the system provides to **end-users**. This can be documented with UML State Diagrams.
- ***Physical view***: The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the **physical layer as well as the physical connections between these components**. This view is also known as the deployment view. This can be documented with UML Deployment Diagrams
- ***Process view***: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on **the runtime behavior of the system**. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML diagrams to represent process view include the activity diagram.[2]
- ***Scenarios***: The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view. The scenarios describe sequences of interactions between **objects and between processes.** They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the **use case view**.

The Software Architect



The Important Concept of “Interface”

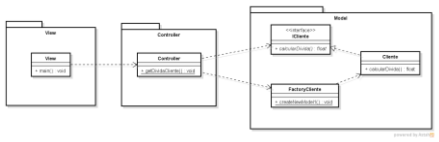
- Interfaces
  - Interfaces are the set of signatures an object exposes
  - The interface of an object states all the requisitions can be sent to it
  - A Type is a name used to denote a specific interface
  - Objects can have several types (hierarchy)
  - Interfaces do not say anything about the object's implementation (i.e., the internals of methods)



16

The Important Concept of “Interface”
Visibility of Package

- Interfaces + visibility limited to the package is perfect for encapsulate things...



Architectural Patterns/Styles

- Architectural Patterns is a **structural organization** schema for software systems; It is a way for structuring the system !
- An Architectural style determines the **vocabulary** of components and connectors that can be used in instances of that style, together with a set of **constraints** on how they can be combined. These can include **topological constraints** on architectural descriptions (e.g., no cycles).

Architectural Patterns/Styles

- What is the difference between Design Patterns and Architectural Patterns ?
  - Design Patterns offer a common solution for a common problem in the form of **classes** working together
    - Smaller in scale than architectural patterns, where the **components** are subsystems rather than **classes**
  - Design Patterns do not influence the fundamental structure of a system
    - Only affect a single subsystem
    - They may help implementing architectural patterns

Categories of Architectural Patterns

- Data Flow Architectures
  - DFD (Data Flow Diagrams) **(this is not an Architectural Pattern)**
  - Pipes and Filters
  - Batch Sequencial
- Independent Components
  - Client-Server
  - Paralel Processes
  - Arquitetura Baseada em Eventos
- Repository Architectures
- Layered Architectures

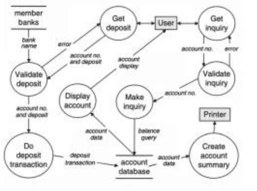
Categories of Architectural Patterns
Data Flow Architectures: DFD

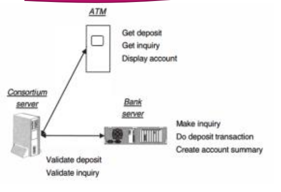
- DFD is a graphical technique for modeling a system focusing on the data flowing among **processing units**
- It must be used when the **data** flow among processes and the **processing units** are important to be seen/analyzed !
- The idea is that each processing units is designed independently of the others

Categories of Architectural Patterns
Data Flow Architectures: DFD

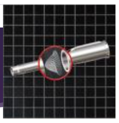
DFD is just a modeling technique... Like a UML diagram....

- **Banking ATM Application**





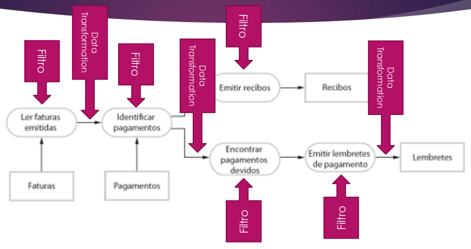
Categories of Architectural Patterns
Data Flow Architectures: Pipes and Filters



- In a pipe and filter style each component has a set of inputs and a set of outputs
  - The focus is on the transformations applied over data. Data are transformed as long as they pass through the pipes....
- The **connectors** of this style serve as conduits for the streams, transmitting outputs of one filter to inputs of another. Hence the connectors are termed "pipes"
- Invariants:
  - Filters must be independent entities: in particular, they should not **share state** with other filters
  - Filters do not know the identity of their upstream and downstream filters
  - Their specifications might restrict what appears on the input pipes or make guarantees about what appears on the output pipes

What does it mean ?

Categories of Architectural Patterns
Data Flow Architectures: Pipes and Filters

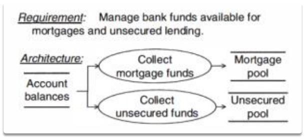


Categories of Architectural Patterns
Data Flow Architectures: Pipes and Filters

- Common specializations of this style include
  - **Pipelines**, which restrict the topologies to linear sequences of filters;
  - **Bounded pipes**, which restrict the amount of data that can reside on a pipe and
  - **Typed pipes**, which require that the data passed between two filters have a well-defined type
- Pipe and filter systems have a number of nice properties:
  - First, they allow the designer to understand the overall input/output behavior of a system as a simple composition of the behaviors of the individual filters.
  - Second, they support reuse: any two filters can be hooked together, since they agree on the data that is being transmitted between them.
  - Third, systems can be easily maintained and enhanced: new filters can be added to existing systems and old filters can be replaced by improved ones.
  - Finally, they naturally support concurrent execution. Each filter can be implemented as a separate task and potentially executed in parallel with other filters

Categories of Architectural Patterns
Data Flow Architectures: Batch Sequencial

- Occurs when batches of data are given for the processing elemens.
  - Batches of data are an enormous volume of data...
- This is characterized when the functions (processing units) are executed using ALL THE INPUT DATA for a given "run"

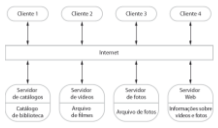


Categories of Architectural Patterns
Independent Components

- Consists of components working in paralel (in principle) and communicating with each other from time to time
- The term "Component" usually is used for characterize as:
  - a portion of software which is independent;
  - does not require information from the software that use it
  - they are self-contained, i.e., they have a wel defined functionality
  - They use other componentes by agregation
- Eclipse is a very good example because of its plug-ins

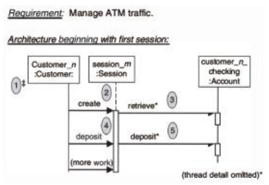
Categories of Architectural Patterns
Independent Components: Tiered and Client-Server

- In a client-server architecture, the server component serves the needs of the client upon request.
- Specifications:
  - A set of autonomous servers that provide services, such as Printing, Data Management, etc
  - A set of clients that request services
  - A network that allows clients to access servers
  - Remote method calls: CORBA, Web Services….

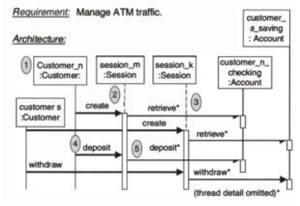


Categories of Architectural Patterns
Independent Components: Paralel Communicating Processes

- It is characterized by several processes (or threads) executing at the same time;



Categories of Architectural Patterns
Independent Components: Paralel Communicating Processes



Categories of Architectural Patterns
Independent Components: Event-Based

30

- ...instead of invoking a method directly, a component can announce one or more **events**.
- Then …. other components in the system **can register an interest in an event by associating a procedure with the event.**
- When the event is announced, the system itself invokes all of the procedures that have been registered for the event.
- Thus an event announcement ``implicitly" causes the invocation of procedures in other modules
- The most conventional use is in graphical interfaces, robotic applications and real time

Chapter 6 Architectural design

Categories of Architectural Patterns
Independent Components: Event-Based

- Architecturally speaking:
  - the components are modules whose interfaces provide both a collection of **procedures** and a set of **events**.
  - Procedures may be called in the usual way. But in addition, a component can register some of its procedures with events of the system. This will cause these procedures to be invoked when those **events are announced** at run time.
  - Thus the connectors include traditional procedure call as well as **bindings** between **event announcements** and procedure calls
  - The main **invariant** that announcers of events do not know which components will be affected by those events. Thus components cannot make assumptions about order of processing, or even about what processing, will occur as a result of their events

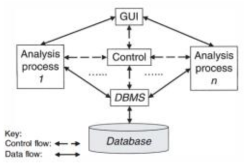
Categories of Architectural Patterns
Independent Components: Event-

```
public class TestProgramm {

    public static void main(String[] args){
      TestProgramm a = new TestProgramm ();
      a.go();
    }

    public void go (){
      MyLightListener lightListener = new MyLightListener ();
      MyTouchListener touchListener = new MyTouchListener ();

      LightSensor light = new LightSensor (SensorPort.S1);
      TouchSensor touch = new TouchSensor (SensorPort.S2);

      SensorPort.S1.addSensorPortListener(lightListener);
      SensorPort.S2.addSensorPortListener(touchListener);

      LCD.drawString("LightListener:", 0, 1);
      LCD.drawString("TouchListener:", 0, 4);

      Button.waitForPress();
   }
```

```
class MyLightListener implements SensorPortListener {


    @overrides

     public void stateChanged(SensorPort source, int oldValue,
    int newValue) {
          LCD.drawString("alt: "+ oldValue + "   neu: " + newValue,
    0, 2);
      }
     }


 class MyTouchListener implements SensorPortListener {

         @overrides
       public void stateChanged (SensorPort source, int
    oldValue, int newValue){
           LCD.drawString("alt: " + oldValue + "   neu: " +
    newValue, 0, 5);
       }
     }
     }
```

Categories of Architectural Patterns
Repository Architecture

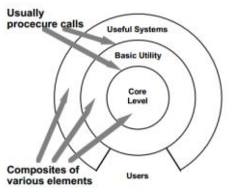
- It is an architecture having a data repository as a central unit
- The most common type is systems that use databases;
- Notice that, storing data is not always using a database...
  - IDEs – repositor of source code

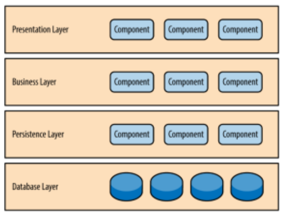


Architectural Style: Layers

- A layered system is organized hierarchically, each layer **providing service** to the layer above it and serving as a client to the layer below
- Topological constraints include **limiting** interactions to adjacent layers
- Support the incremental development of a system. When the **interface** of a layer change, only the adjacente layer is affected.

Architectural Style: Layers


Usually procecure calls
Useful Systems
Basic Utility
Core Level
Composites of various elements
Users


Presentation Layer
Component
Component
Component
Business Layer
Component
Component
Component
Persistence Layer
Component
Component
Component
Database Layer