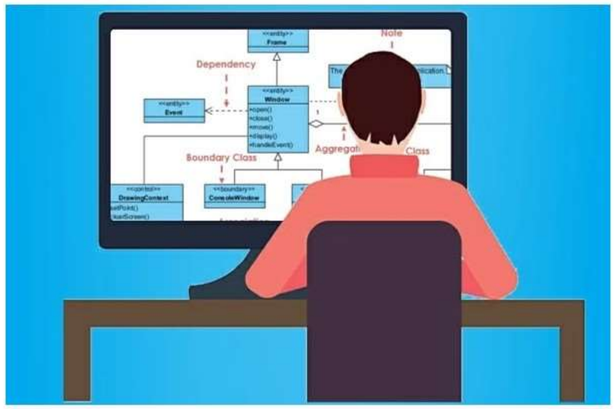
# Software Design & Architecture Spring 2022 - Week-10




مدرس: مہندس ماجد کلیم

جامعہ بحریہ، واقعگاہ کراچی

Engr. Majid Kaleem

## WEEKLY AGENDA

| TENTATIVE WEEKLY DATES | | TENTATIVE TOPICS |
|---|---|---|
| 1 | Mar 7$^{th}$ – Mar 11$^{th}$ | INTRODUCTION TO THE COURSE; DEFINING SOFTWARE ARCHITECTURE & DESIGN CONCEPTS |
| 2 | Mar 14$^{th}$ – Mar 18$^{th}$ | DESIGN PRINCIPLES; OBJECT-ORIENTED DESIGN WITH UML |
| 3 | Mar 21$^{st}$ – Mar 25$^{th}$ | SYSTEM DESIGN & SOFTWARE ARCHITECTURE; OBJECT DESIGN, MAPPING DESIGN TO CODE |
| 4 | Mar 28$^{th}$ – Apr 1$^{st}$ | FUNCTIONAL DESIGN; UI DESIGN; WEB APPLICATIONS DESIGN **ASSIGNMENT & QUIZ #1** |
| 5 | Apr 4$^{th}$ – Apr 8$^{th}$ | MOBILE APPLICATION DESIGN; PERSISTENCE LAYER DESIGN |
| 6 | Apr 11$^{th}$ – Apr 15$^{th}$ | CREATIONAL DESIGN PATTERNS |
| 7 | Apr 18$^{th}$ – Apr 22$^{nd}$ | STRUCTURAL DESIGN PATTERNS **ASSIGNMENT & QUIZ #2** |
| 8 | Apr 25$^{th}$ – Apr 29$^{th}$ | BEHAVIORAL DESIGN PATTERNS |
| | | ← MID TERM EXAMINATIONS → |
| 9 | May 9$^{th}$ – May 13$^{th}$ | INTERACTIVE SYSTEMS WITH MVC ARCHITECTURE; SOFTWARE REUSE |
| 10 | May 16$^{th}$ – May 20$^{th}$ | ARCHITECTURAL DESIGN ISSUES; ARCHITECTURE DESCRIPTION LANGUAGES (ADLS) |
| 11 | May 23$^{rd}$ – May 27$^{th}$ | ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES |
| 12 | May 30$^{th}$ – Jun 3$^{rd}$ | ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES **ASSIGNMENT & QUIZ #3** |
| 13 | Jun 6$^{th}$ – Jun 10$^{th}$ | QUALITY TACTICS; ARCHITECTURE DOCUMENTATION |
| 14 | Jun 13$^{th}$ – Jun 17$^{th}$ | ARCHITECTURAL EVALUATION TECHNIQUES |
| 15 | Jun 20$^{th}$ – Jun 24$^{th}$ | MODEL DRIVEN DEVELOPMENT **ASSIGNMENT (PRESENTATIONS) & QUIZ #4** |
| 16 | Jun 27$^{th}$ – Jul 1$^{st}$ | REVISION WEEK |
| | | ← FINAL TERM EXAMINATIONS → |

## Software Architecture Design Issues

- There could a number software architecture design issues, some important issues and challenges are highlighted below:
  - Requirements volatility
  - Inconsistent development processes
  - Fast, and ever-changing technology
  - Ethical and professional practices
  - Managing design influences

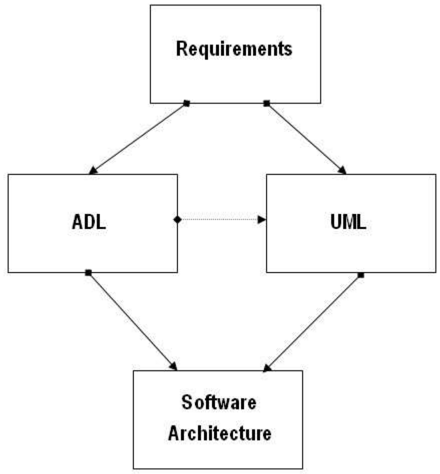
3

## Architectural Description Languages (ADL)

- Architecture Description Language (ADL) is defined as "*a language (graphical, textual, or both) for describing a software system in terms of its architectural elements and the relationship among them*".
- In other words, ADL is a language enabling formalization, description, specification, modeling and reasoning on software architectures.
- Each of these features should be fulfilled by a language that is proclaimed to be ADL.
- A good ADL must provide abstractions that are adequate for modeling a large system.
- Each ADL embodies a particular approach to the specification and evolution of architecture.

4

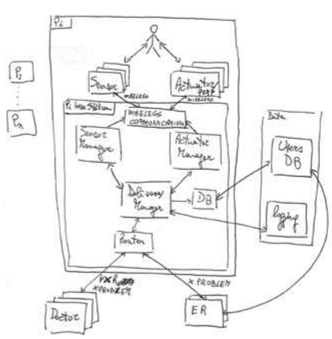
## ARCHITECTURAL DESCRIPTION LANGUAGE



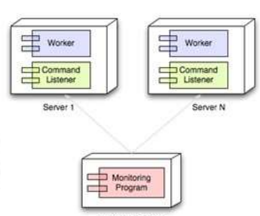



## ADL (ACCORDING TO ISO/IEC/IEEE 42010)

- ADL = Architecture Description Language = any mode of expression used in an architecture description
- It could be informal, UML-based, and formal as given below:



Informal




UML-based

```
process control_processing
features
input: in data port;
output: out data port;
end control_processing;

process implementation control_processing.speed_control
subcomponents
control_input: thread control_in.input_processing_01;
control_output: thread control_out.output_processing_01;
control thread group: thread group
control threads.control thread set 01;
set point data: data set point data type;
end control_processing.speed_control;

thread control_in
end control_in;

thread implementation control_in.input_processing_01
end control_in.input_processing_01;

thread control out
end control_out;

thread implementation control out.output_processing_01
end control_out.output_processing_01;

thread group control_threads
end control_threads;

thread group implementation control_threads.control_thread_set_01
end control_threads.control_thread_set_01;

data set point data type
end setpoint_data_type;
```

Formal

## ARCHITECTURAL DESCRIPTION LANGUAGES (ADL)

- It may be a formal language (like Acme, Darwin, AADL), a UML-based notation, as well as any other means you may have used to describe a software architecture.
- An ADL is tailored to specify SA concepts (components, connectors, interfaces, …) through different viewpoints.

A model that describes the structure of a software system in terms of computational *components*, the *relationships* among components, and the *constraints* for assembling the components.

That is, a software architecture can be defined in terms of the following elements:

*Software Architecture = {components, relationships, constraints}*

7

## ARCHITECTURAL DESCRIPTION LANGUAGES (ADL)

1. **Components**.

- Components are the computational elements which collectively constitute an architecture.
- A software architecture is typically decomposed into ***subsystems***, which in turn may be decomposed into ***modules***.
- Further decomposition is also possible. (For example in an ***object-oriented design***, ***modules*** may be decomposed into ***classes***.)
- Examples of components include clients, services, and persistent stores.

8

## ARCHITECTURAL DESCRIPTION LANGUAGES (ADL)

2. **Relationships**.
- Relationships are the logical connections between architectural components.
- Examples of abstract component relationships include ***dependency***, ***aggregation***, and ***composition***.
- Examples of concrete component relationships include client-server protocols and database protocols.

9

## ARCHITECTURAL DESCRIPTION LANGUAGES (ADL)

3. **Constraints**.
- Constraints provide ***conditions*** and ***restrictions*** for ***component relationships***.
- They connect the architecture to system requirements.
- Examples of constraints include restrictions on parameters types for communication protocols and high availability requirements for fault tolerance.

10

## Architectural Description Languages (ADL)

- An ADL is a language that provides features for modelling a software system's conceptual architecture.
- Architecture description languages (ADLs) are formal languages that can be used to represent the architecture of a software-intensive system.
- By architecture, we mean the ***components*** that comprise a system, the behavioral specifications for those components, and the patterns and mechanisms for interactions among them.
- Note that a single system is usually composed of more than one type of component: modules, tasks, functions, etc.
- An architecture can choose the type of component most appropriate or informative to show, or it can include multiple views of the same system, each illustrating different components.

11

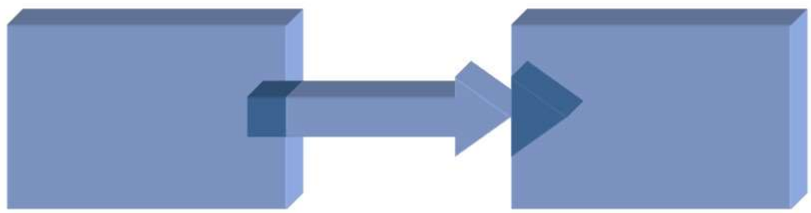
## Architectural Description Languages (ADL)

- The positives
  - ADLs represent a formal way of representing architecture
  - ADLs are intended to be both human and machine readable
  - ADLs support describing a system at a higher level than previously possible
  - ADLs permit analysis of architectures – completeness, consistency, ambiguity, and performance
  - ADLs can support automatic generation of software systems
- The negatives
  - There is no universal agreement on what ADLs should represent, particularly as regards the behavior of the architecture
  - Representations currently in use are relatively difficult to parse and are not supported by commercial tools
  - Most ADL work today has been undertaken with academic rather than commercial goals in mind
  - Most ADLs tend to be very vertically optimized toward a particular kind of analysis

12

## Software Architecture – ADL Perspective

- The ADL community generally agrees that Software Architecture is a set of components and the connections among them.
  - components
  - connectors
  - configurations
  - constraints



13

## Architectural Description Languages

- Some ADLs are listed below:
  - ACME (CMU/USC)
  - Rapide (Stanford)
  - Wright (CMU)
  - Unicon (CMU)
  - Aesop (CMU)
  - MetaH (Honeywell)
  - C2 SADL (UCI)
  - SADL (SRI)
  - Lileanna
  - UML
  - Modechart

14

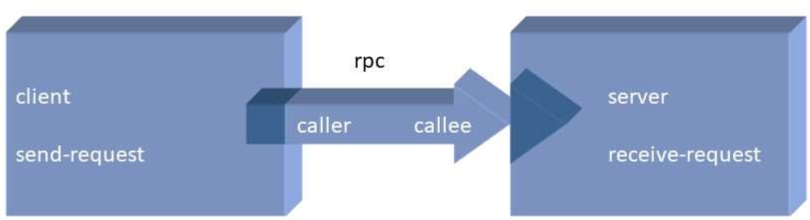
## ACME

- ACME was developed jointly by Monroe, Garlan (CMU) and Wile (USC)
- ACME is a general purpose ADL
- ACME as a language is extremely simple (befitting its origin as an interchange language)
- ACME has no native behavioral specification facility so only syntactic linguistic analysis is possible

15

## ACME - EXAMPLE

```
System simple_cs = {
        Component client = {Port send-request}
        Component server = {Port receive-request}
        Connector rpc = {Roles {caller, callee}}
        Attachments : {client.send-request to rpc.caller;
            server.receive-request to rpc.callee}
}
```




16

## RAPIDE

- Rapide was developed by Dr. David Luckham at Stanford
- Rapide is a general purpose ADL designed with an emphasis on simulation yielding partially ordered sets of events (posets)
- Rapide as a language is fairly sophisticated, including data types and operations
- Rapide has a fairly extensive toolset
- Rapide is a concurrent, object-oriented , event-based simulation language
- Defines and simulates behavior of distributed object system architectures
- System requirements are expressed as constraints on time and concurrent patterns of events

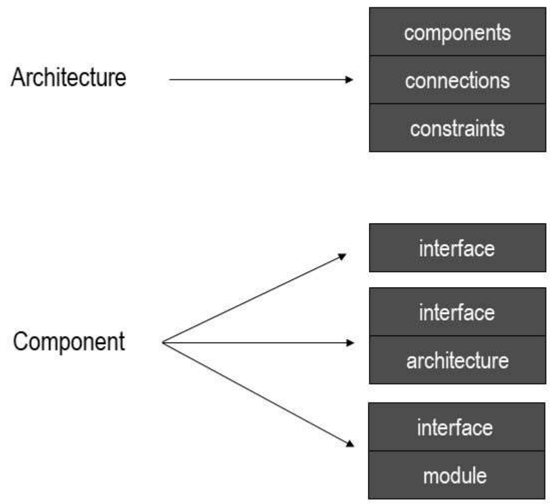
17

## RAPIDE MODEL

- Components execute independently
- Components both observe and generate events
  - Each event represents the occurrence of an activity
- Generates dependent events
  - Reactive rules in interface behaviors (i.e. transition rules)
  - Reactive processes in modules (i.e. when statements)
  - Events generated by sequential execution
  - Shared objects via references
- Generates timed events
  - Interface behavior or module can be timed
  - Events receive start and finish times within scope of its clock
  - Events can be synchronized to a clock

18

## RAPIDE ARCHITECTURAL ELEMENTS




19

## RAPIDE ARCHITECTURAL ELEMENTS

```
type Producer (Max : Positive) is interface
    action out Send (N: Integer);
    action in Reply(N : Integer);
behavior
    Start => send(0);
    (?X in Integer) Reply(?X) where ?X<Max => Send(?X+1);
end Producer;

type Consumer is interface
    action in Receive(N: Integer);
    action out Ack(N : Integer);
behavior
   (?X in Integer) Receive(?X) => Ack(?X);
end Consumer

architecture ProdCon() return SomeType is
    Prod : Producer(100); Cons : Consumer;
connect
    (?n in Integer) Prod.Send(?n) => Cons.Receive(?n);
    Cons.Ack(?n) => Prod.Reply(?n);
end architecture ProdCon;
```

20

## WRIGHT

- Wright was developed by Dr. David Garlan at CMU
- Wright is a general purpose ADL designed with an emphasis on analysis of communication protocols
  - Wright uses a variation of CSP to specify the behaviors of components, connectors, and systems
    - CSP - Communicating Sequential Processes - process algebra developed by C. A. R. Hoare
- Wright as a language focuses primarily on the basic component/connector/system paradigm
  - Wright is very similar syntactically to ACME and Aesop
- Wright analysis focuses on analyzing the CSP behavior specifications.
  - Any CSP analysis tool or technique could be used to analyze the behavior of a Wright specification
- Wright has minimal native tool support (but CSP tools could be used)

21

## A SIMPLE SPECIFICATION IN WRIGHT

```
System simple_cs
    Component client =
        port send-request = [behavioral spec]
        spec = [behavioral spec]
    Component server =
        port receive-request= [behavioral spec]
        spec = [behavioral spec]
    Connector rpc =
        role caller = (request!x -> result?x ->caller) ^ STOP
        role callee = (invoke?x -> return!x -> callee) [] STOP
        glue = (caller.request?x -> callee.invoke!x
              -> callee.return?x -> callee.result!x
                  -> glue) [] STOP
    Instances
        s : server
        c : client
        r : rpc
    Attachments :
        client.send-request as rpc.caller
        server.receive-request as rpc.callee
    end simple_cs.
```

22

## AESOP

- Aesop was developed by Dr. David Garlan at CMU
- Aesop is a general purpose ADL emphasizing architectural styles
  - Aesop is also a toolset and a framework
- Aesop the ADL is very similar to ACME/Wright
  - Emphasis on styles reflected in more sophisticated hierarchical facilities centered around subtyping and inheritance
- Wright analysis focuses on analyzing the CSP behavior specifications.
  - Any CSP analysis tool or technique could be used to analyze the behavior of a Wright specification

23

## UNICON

- Unicon was developed by Dr. Mary Shaw at CMU
- Unicon is a general purpose ADL designed with an emphasis on generation of connectors
  - Unicon developed to support treatment of connectors as first class objects by providing for the generation of systems with explicit connectors
- Unicon as a language focuses primarily on the basic component/connector/system paradigm but with an emphasis on architectural styles
  - Emphasis on styles simplifies generation efforts
- Unicon has a generation capability

24

## UML and ADL

- Unified Modeling Language (UML) is a formal graphical language considered a *de facto* industrial standard.
- Although the language has been initially created as a graphical language that supports object oriented software analysis and design, the language has been revised a couple of times and today, it is a general formal language capable of describing a software system.
- The UML has a well-defined formal syntax and semantics, and can be machine checked and processed.
- UML includes a set of graphical notation techniques to create abstract models of specific systems.

25

## Others

- MetaH
  - Developed by Honeywell, a domain specific ADL aimed at guidance, navigation, and control applications with ControlH
  - Sophisticated tool support available
- C2 SADL
  - Developed by Taylor/Medvidovic (UCI), style specific ADL, emphasis on dynamism
  - Still in prototype stage
- SADL
  - Developed by Moriconi and Riemenschneider (SRI), emphasis on refinement mappings

26

```
If(anyQuestions)
{
 askNow();
}
else
{
 thankYou();
 submitAttendance();
 endClass();
}
```



## REFERENCES


1. ***Software Architecture***, *Perspectives on an Emerging Discipline By Mary Shaw & David Garlan*
2. ***The Art of Software Architecture***, *Design Methods & Techniques By Stephen T. Albin*
3. ***Essential Software Architecture***, *By Ian Gorton*
4. ***Microsoft Application Architecture Guide***, *By Microsoft*
5. ***Design Patterns***, *Elements of Reusable Object-Oriented Software By by Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides*
6. ***Refactoring, Improving the Design of Existing Code***, *By Martin Fowler & Kent Beck*