SENG 401
Software Architecture

Lecture 1

Jan 12, 2021

Gias Uddin
Assistant Professor, University of Calgary

http://giasuddin.ca/

Agenda

- Overview
  - Expected outcomes
  - How the course is structured
  - Grading Scheme
- Reading Topic 1. Introduction to Software Architecture
- Case Study 1. Architecture of scalable web architecture and distributed systems

Overview

Learning Outcomes

At the end of the course, students should be able to do the following:
• Understand the concepts, terminology, and notation of software architecture
• Describe and explain the major challenges with developing and maintaining large software systems
• Model and document software architecture using a variety of modelling notations
• Gain exposure into the architecture of existing large-scale software systems
• Design functional and non-functional properties of a software using principles of software architecture
• Discuss the merits and problems of different architectural decisions
• Describe and explain common architectural principles, patterns, and connectors, etc.

Overview

Lecture Organization

- Every lecture has two components:
  1. Reading Topic
  2. Case Study
- Reading Topic
  - Highlights from book chapters
  - Highlights from research papers
- Case Study
  - An overview of the architecture of a real-world software system
  - The subsequent lab will be based on the case study

Overview

Grading Scheme

Item	Number	Grade
Assignment (Take Home) – Individual	2	20%
Quiz (Take Home) – Individual	1	10%
Midterm - Individual	1	20%
Project Presentations – Group of 6 Students	3	15%
Project Final Report Writeup – Group of 6 Students	1	20%
Project Code + Model + Data – Group of 6 Students	1	15%

Books Used/Cited in this Course

ID	Title
B1	Software Architecture: Foundations, Theory, and Practice. Taylor, Medvidovic, Dashofy. Wiley, 2009. *Available at UofC Library*
B2	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume I. http://www.aosabook.org/en/index.html
B3	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume II. http://www.aosabook.org/en/index.html
B4	Documenting Software Architecture: Views and Beyond, 2nd Edition. Addison-Wesley, 2010. Clements et al. *Available at UofC Library*
B5	Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Nick Rozanski. Addison-Wesley Professions, 2011. *Available at UofC Library*
B6	Software Architecture in Practice. Bass, Clements, Kazman. Addison-Wesley, 2012 *Available at UofC Library*

SCHULICH
School of Engineering

Reading Topic 1
Introduction to Software Architecture

References: B1 - C1, C2

Overview

What is Software Architecture? Why is it important?

- A software architecture is the set of principal decisions made during the design, development, and evolution of a system
- Software architecture helps to reason about the complex organization of components (e.g., software modules), the interactions and connections among the components in large-scale software systems.
- It is generally possible to write toy program, small software with couple of dozens of classes, or scripts without thinking much about the overall system design.
- It is next to impossible to design any software larger beyond that with a proper knowledge of software architecture principles.
- For example, the open-source Eclipse IDE ecosystem has more than 240 million lines of code from 1,631 committers as of June 2020 (source: Wikipedia https://en.wikipedia.org/wiki/Eclipse_Foundation)

Fundamentals
The origins

- Software Engineers have always employed software architectures – very often without realizing it.
- Large scale Software Engineering (SE) is complicated and difficult
  - SE is a young field with rapid expansion and tremendous expectations
  - A software application is a product by itself
  - But SE is not useful by itself unlike a car which by itself is useful, but a software application in a smart car is useful.
- Any large-scale software application has to handle two types of difficulties during its inception, development and maintenance

Fundamentals

SE Difficulties in Large-scale Software

- Any large-scale software application must handle two types of difficulties during its inception, development and maintenance
  - Accidental – can be eliminated with careful practice
    - Solutions exist but they are possibly waiting to be discovered
    - Example – inadequate programming construct and abstractions
    - Example – Difficulty of using heterogeneous systems
  - Essential – can be lessened but not eliminated
    - Only partial solutions exist, and they can be case-specific
    - Cannot be abstracted away:
      - Complexity
      - Conformity
      - Changeability
      - Intangibility

Essential Difficulties

Complexity

- Any large-scale software application must handle two types of difficulties during its inception, development and maintenance
  - Accidental – can be eliminated with careful practice
    - Solutions exist but they are possibly waiting to be discovered
    - Example – inadequate programming construct and abstractions
    - Example – Difficulty of using heterogeneous systems
  - Essential – can be lessened but not eliminated
    - Only partial solutions exist, and they can be case-specific
    - Cannot be abstracted away:
      - Complexity
      - Conformity
      - Changeability
      - Intangibility

Analogy of SA with Other Domains

The Architecture of Buildings

- Like a software system, a building has an architecture, which is a concept separate form, but linked to the physical structure itself.
- Properties (e.g., safety measures) of a building are induced by the design of its architecture. For software, functional and non-functional properties (e.g., security) are determined by the design of the architecture.
- Architects of buildings require a broad training. Similarly, simple skill in programming is not sufficient to create complex large software systems.
- Process is not as important as the architecture
- Building architecture has matured over the years into a discipline.
- An architectural style puts a set of constraints

Analogy of SA with Other Domains
The Architecture of Buildings

- It is more about the architect than the process itself. An architect is
  - A distinctive role and character in a project
  - Has very broad training
  - Amasses and leverages extensive experience
  - Has a keen sense of aesthetics
  - Has Deep understanding of the domain
    - Properties of structures, materials, and environments
    - Needs of customers

- Exercise: What are the limitations of this analogy?

Analogy of SA with Other Domains

The Architecture of Buildings: Limitations of the Analogy

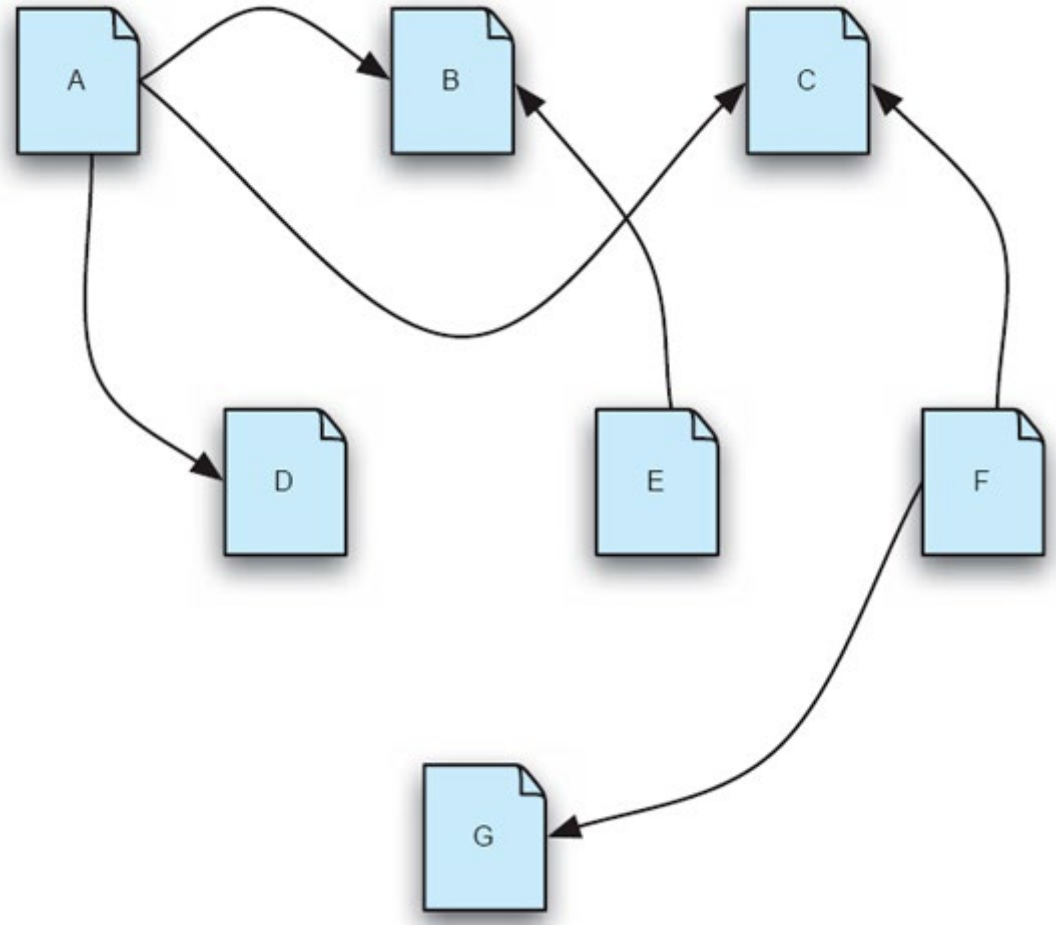
- We know a lot about buildings, much less about software
- The nature of software is different from that of building architecture
- Software is much more malleable than physical materials
- The two “construction industries” are very different
- Software deployment has no counterpart in building architecture
- Software is a machine; a building is not

The Needs for an Efficient Software Architect

Giving more power to software architect means

- Giving preeminence to architecture offers the potential for
  - Intellectual control
  - Conceptual integrity
  - Effective basis for knowledge reuse
  - Realizing experience, designs, and code
  - Effective project communication
  - Management of a set of variant systems
- Limited-term focus on architecture will not yield significant benefits!

The Power and Necessity of Big Ideas for SA
The Architecture of the Web (WWW)

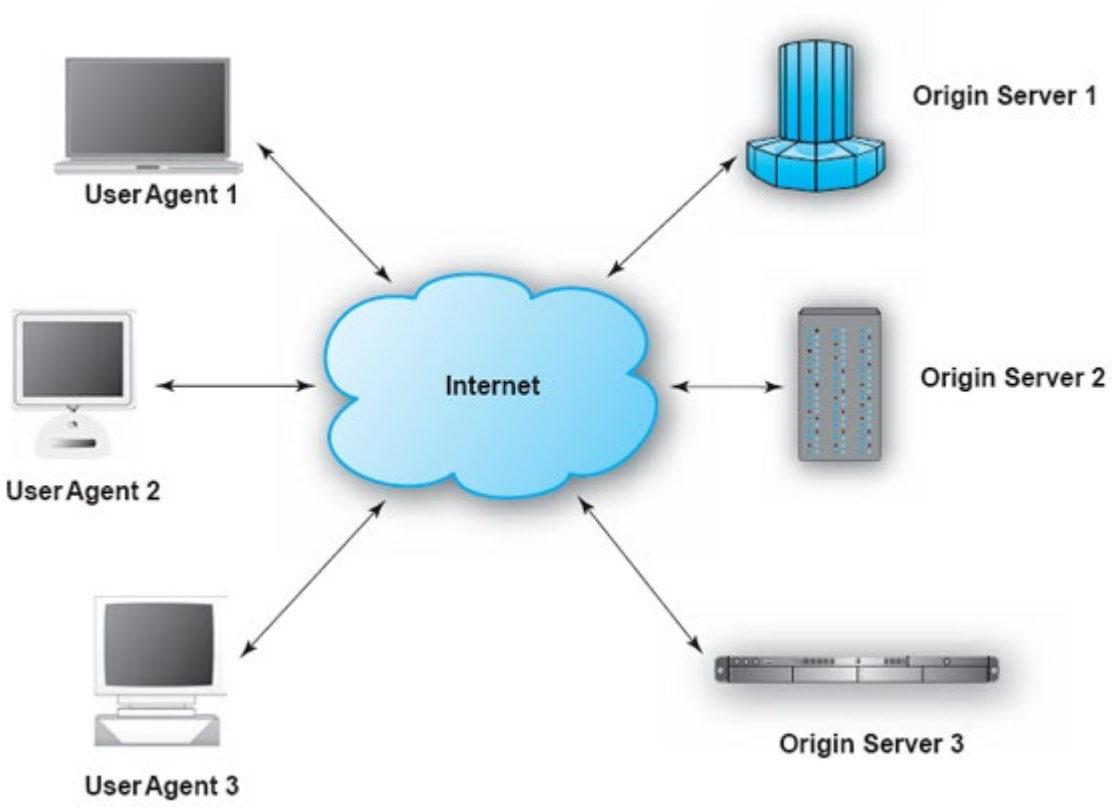


What is the Web?

In one view, focusing on the user's perception, the Web is a dynamic set of relationships among collections of information. In another view, focusing on coarse-grained aspects of the Web's structure, the Web is a dynamic collection of independently owned and operated machines, located around the world, which interact across computer networks. In another view, taking the perspective of an application developer, it is a collection of independently written programs that interact with each other according to rules specified in the HTTP, URI, MIME, and HTML standards.

Considering these perspectives in turn, the view of the Web as a collection of interrelated pieces of information as illustrated right.

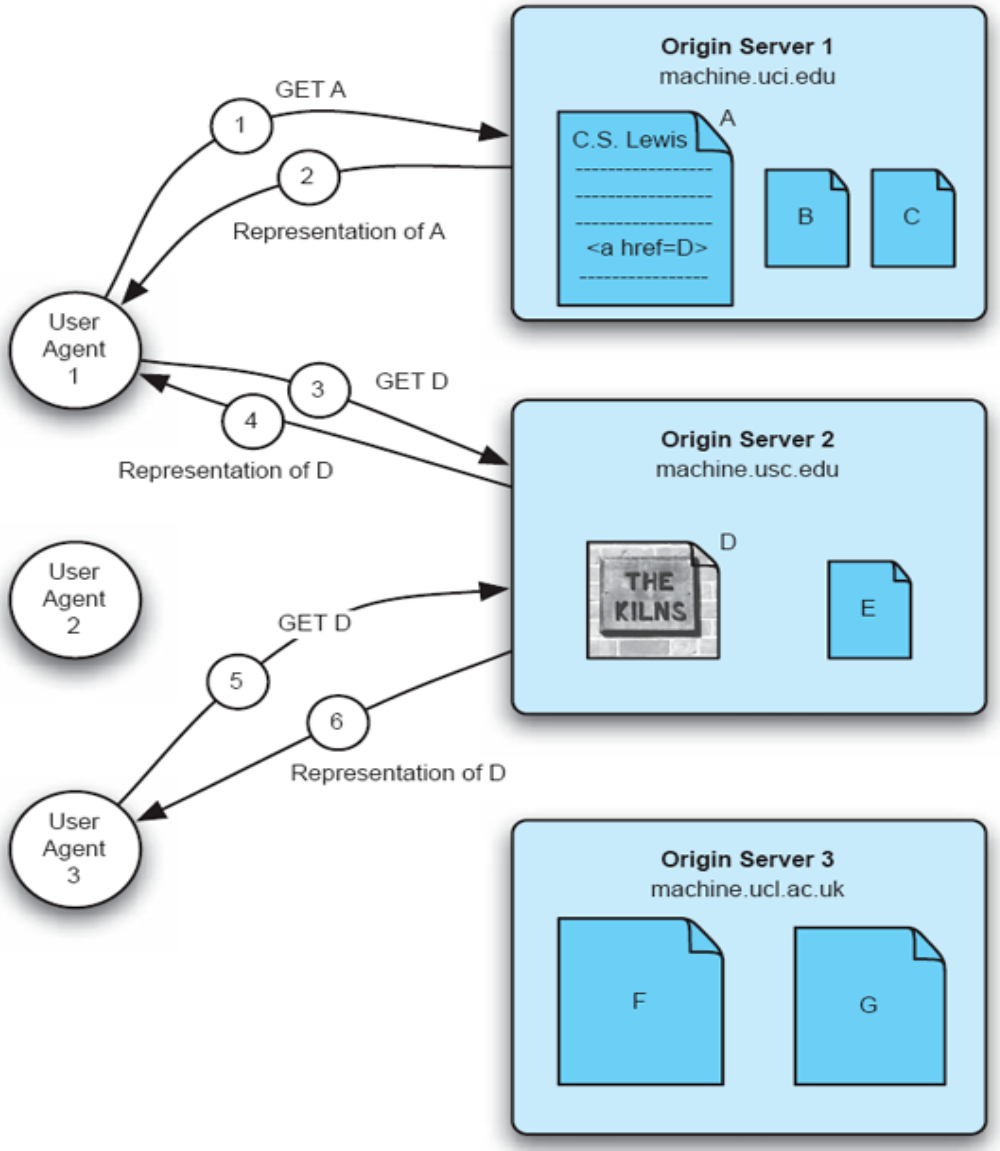
The Power and Necessity of Big Ideas for SA
The Architecture of the Web (WWW)



What is the Web?

This figure shows the Web as a collection of computers interconnected through the Internet. The machines are shown here as being of two kinds: *user agents* and *origin servers*. User agents are, for instance, desktop and laptop machines on which a Web browser is running. Origin servers are machines that serve as the permanent repositories of information, such as the aforementioned documents pertaining to C. S. Lewis. In this view, the Web is understood as a physical set of machines whereby a user at a user agent computer can access information from the origin servers. The view is quite sketchy, however, and does not present any real insight into how the information is obtained by the user agents or how the information in one document is related to information in another. Nonetheless, the abstraction it presents is accurate insofar as it goes, and can be useful in explaining some Web concepts.

The Power and Necessity of Big Ideas for SA
The Architecture of the Web (WWW)



What is the Web?

In this view, the user agents and origin servers of previous figure are once again shown, but now the location of documents A to G are shown. The figure also shows a set of specific interactions among two of the user agents and two of the origin servers, corresponding to a particular pattern of accessing the C. S. Lewis hypertext. In this example, User Agent 1 requests, by means of the HTTP method GET, a copy of document A, a biography of Lewis. This interaction is shown in the diagram by the arrow marked 1. A representation, or "copy," of the biography is returned, shown as the arrow marked 2. Since the biography has within it a hypertext reference (href) to document D, a JPEG of The Kilns, User Agent 1 issues another GET, this time to machine.usc.edu, to obtain the picture. This request is marked 3 in the diagram. A copy of the image is returned, as shown by the arrow marked 4. Further interactions between user agents and servers are shown in the diagram as interactions 5 and 6.

The Power and Necessity of Big Ideas for SA

The Architecture of the Web (WWW)

SCHULICH School of Engineering

- The Web is a collection of resources, each of which has a unique name known as a uniform resource locator, or “URL”.
- Each resource denotes, informally, some information.
- URI’s can be used to determine the identity of a machine on the Internet, known as an origin server, where the value of the resource may be ascertained.
- Communication is initiated by clients, known as user agents, who make requests of servers.
  - Web browsers are common instances of user agents.
- Resources can be manipulated through their representations.
  - HTML is a very common representation language used on the Web.
- All communication between user agents and origin servers must be performed by a simple, generic protocol (HTTP), which offers the command methods GET, POST, etc.
- All communication between user agents and origin servers must be fully self-contained. (So-called “stateless interactions”)

The Power and Necessity of Big Ideas for SA

WWW vs Source Code

- Architecture of the Web is wholly separate from the code
- There is no single piece of code that implements the architecture.
- There are multiple pieces of code that implement the various components of the architecture.
  - E.g., different Web browsers
- Stylistic constraints of the Web's architectural style are not apparent in the code
  - The effects of the constraints are evident in the Web
- One of the world's most successful applications is only understood adequately from an architectural vantage point.

Architecture in Action
Desktop

- pipes and filters in Unix?
    - ls invoices | grep –e august | sort
- a *filter* is a program that takes a stream of text characters as its input and produces a stream of characters as its output. A filter's processing may be controlled by a set of parameters. A *pipe* is a way of connecting two filters, in which the character stream output from the first filter is routed to the character stream input of the second filter.
- The critical observation here is that this application's structure can be understood on the basis of a very few rules. Given understanding of that structure and knowledge of the functioning of the individual filters, the function of the complete application can be understood. Knowledge of those same rules and of the functioning of a few dozen, preexisting, basic filter programs allows one to understand hundreds of other useful applications. Similarly, a developer may readily create new applications based upon those same rules and knowledge of the filters.

Architecture in Action

Product Line

- Motivating example
  - A consumer is interested in a 35-inch HDTV with a built-in DVD player for the North American market.

    Such a device might contain upwards of a million lines of embedded software.

    This particular television/DVD player will be very similar to a 35-inch HDTV without the DVD player, and also to a 35-inch HDTV with a built-in DVD player for the European market, where the TV must be able to handle PAL or SECAM encoded broadcasts, rather than North America's NTSC format.

    These closely related televisions will similarly each have a million or more lines of code embedded within them.
- Building each of these TVs from scratch would likely put Philips out of business
- Reusing structure, behaviors, and component implementations is increasingly important to successful business practice
  - It simplifies the software development task
  - It reduces the development time and cost
  - it improves the overall system reliability
- Recognizing and exploiting commonality and variability across products

Architecture in Action

Promoting Reuse as the Big Win

- Architecture: reuse of
  - Ideas
  - Knowledge
  - Patterns
  - engineering guidance
  - Well-worn experience
- Product families: reuse of
  - Structure
  - Behaviors
  - Implementations
  - Test suites…

SCHULICH
School of Engineering

Case Study 1
Scalable Web Architecture and Distributed Systems

References: B4 – C2

http://www.aosabook.org/en/distsys.html

Acknowledgment

Part of the lecture contents are taken from the following lecture slide contents with explicit permission from author Medvidovic:

Software Architecture: Foundations, Theory, and Practice. Taylor, Medvidovic, Dashofy. Wiley, 2009.