Visualizing Software Architecture
Perla Velasco-Elizondo

November 2021
What is software architecture?
Software Architecture Defined

“The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.”

Structures …

Let's revise this analogy …

Human body comprised of multiple real structures

A static view of one human structure

A dynamic view of the same structure
It’s usually difficult to show all structures, from all perspectives, on a single representation.
Architectural View

- A view is a **diagram**.
- A view is a **abstract** representation of one or more structures of an architecture that illustrates how the architecture addresses one or more architectural requirements.
Some well-known views are:

1. Static
2. Dynamic
3. Physical
Static View

- Shows structures with elements and relationships that exist in development and correspond to implementation units.
- Shows the system partitioning and assignment of responsibilities.
Dynamic View

- A.k.a. Components & Connectors View
- Shows structures with elements and relationships that exist in execution time.
sd ActivateAll

:Security Guard
activate() → soundActivatedAlarm()

par
monitor() →
loop 1, * [fire] fire = isTriggered()

critical
fire() →
soundFireAlarm()

monitor() →
loop 1, * ([intruder] & [fire]) intruder = isTriggered()

opt [fire]
intruder() →
soundIntruderAlarm()
Physical View

- A.k.a Allocation View
- Shows structures with elements and relationships that exist in deployment time.
Deployment diagram for Internet Banking System - Live
An example live deployment scenario for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Abstractions

- Abstraction is about reducing detail; important characteristics are made more visible by leaving details aside.
- Abstractions help us reason about a big and/or complex software system
A common set of abstractions/views is more important than a common notation

Simon Brown
Notations

- There are different notations for documenting architectural views
- Each notation has its benefits and shortcomings
- **However**, when using a notation for documenting a view, **you must remember** some important aspects ...
What do you think about this view?

What kind of view is it?

Is it useful?

What is missing?
What do you think about this view? What is missing? Is it useful? What kind of view is it? What do you think about this view?
Software Architecture Defined

“The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.”

Aspects to remember

General

- The view has a title
- It is clear the type of the view
- It is clear the scope of the view
- The view has a key/legend
Aspects to remember

Elements
- Every element has a name
- It is clear the type of every element? (e.g. software system, component, etc)
- It is clear what the element does
- Where applicable, it is clear the technology choices associated with every element
- It is clear the meaning of all acronyms and abbreviations
- It is clear the meaning of all shapes/icons
- It is clear the meaning of all border styles (e.g. solid, dashed, etc)
- It is clear the meaning of all element sizes (e.g. small vs large boxes)
Aspects to remember

Relationships

- Every line has a label describing the intent of that relationship
- Where applicable, it is clear the technology choices associated with every relationship? (e.g. protocols for communication)
- It is clear the meaning of all acronyms and abbreviations used
- It is clear the meaning of all colours used
- It is clear the meaning of all arrowheads used
- It is clear the meaning of all line styles (e.g. solid, dashed, etc)
Aspects to remember

Other

- Each view fits on one page
- Do not mix runtime and static elements in the same diagram
- There is more than one view to describe the whole system
- Keep structural and semantic consistency across diagrams
Notations

There are three main notations:

1. Informal
2. Semi-formal
3. Formal
The C4 model for visualising software architecture

Context, Containers, Components, and Code

In a hurry? Read the Wikipedia page and the 5 minute introduction to the C4 model at InfoQ

Or watch "Visualising software architecture with the C4 model"

Uses and benefits

The C4 model is an easy to learn, developer friendly approach to software architecture diagramming. Good software architecture diagrams assist with communication inside/outside of software development/product teams, efficient onboarding of new staff, architecture reviews/evaluations, risk identification (e.g. risk-storming), threat modelling (e.g. STRIDE/LINDDUN), etc.
C4 Model

- **Context**: A high-level diagram that sets the scene; including key system dependencies and actors.
- **Container**: A container diagram shows the high-level technology choices, how responsibilities are distributed across them and how the containers communicate.
- **Component**: For each container, a component diagram lets you see the key logical components and their relationships.
- **Classes**: This is an optional level of detail and I will draw a small number of high-level UML class diagrams if I want to explain how a particular pattern or component will be (or has been) implemented.
Containers

Software Architecture for Developers - Volume 2. Leanpub.
Components

Tooling

For design sessions, you might find a whiteboard or flip chart paper better for collaboration, and iterating quickly. For long-lived documentation, the following modelling and diagramming tools can help create software architecture diagrams based upon the C4 model.

Modelling tools (recommended):

- **Structurizr**
  - Structurizr is a collection of tools to create software architecture diagrams and documentation based upon the C4 model.
  - Structurizr was started in 2014 by Simon Brown (creator of the C4 model), and has grown into a community of tooling, much of which is open source.
  - Structurizr is unique in that it supports diagrams in code (Java, C#, Python, JML, TypeOracles, PHP, Python, Go, or text (DSL or YAML)) via a number of different authoring methods, with it being possible to render diagrams using a number of different tools (Structurizr cloud services, premade installation, PlantUML, Mermaid, Webbased/NextDiagram, etc.).

- **Archi**
  - Archi provides a way for you to create C4 model diagrams with Archimate. See C4 Model, Architecture Viewpoints and Archi 4.7 for more details.

- **Sparx Enterprise Architect**
  - Lieberknecht Software has built an extension for the C4 model, based upon the MDG Technology built into Sparx Enterprise Architect.

- **MooD**
  - MooD has support for the C4 model via a set of blueprints.

- **Astah**
  - Astah has support for the C4 model via a C4 model plugin.

Diagramming tools:

- **PlantUML**
  - There are a number of extensions for PlantUML to assist in the creation of C4 model diagrams:
    - C4-PlantUML by Ricardo Nocita
    - C4-PlantUML by Savvas Kliarnthous
    - codeilluminator by Visar Lupu
    - plantumlR by Thibault Morin
  - You can also create C4-PlantUML diagrams using C4 code via the C#Sharp library.

- **diagrams.net**
  - diagrams.net includes support for the C4 model, and there are also a number of plugins that allow you to create diagrams using pre-built shapes:
    - C4-Draw.io by Chris Kaminski
    - C4-Draw.io by Tobias Hochgertl
    - C4-Draw by Mark Schein

- **OmniGraffle**
  - Dennis Laumen has created a C4 model stencil for OmniGraffle, that allows you to create diagrams using pre-built shapes.

- **Microsoft Visio**
  - "yShape" has created a C4 model template for Microsoft Visio, that allows you to create diagrams using pre-built shapes.

- **Visual Paradigm**
  - Visual Paradigm supports the C4 model via some pre-built shapes.

- **yEd**
  - Ferhat Kalkon has created some C4 model shapes for yEd.
"Communication works for those who work at it"
Visualización de la arquitectura del software

Perla Velasco Elizondo
Senior Product Manager @ Kueski
Profesora @ Universidad Autónoma de Zacatecas