Τεχνολογία Λογισμικού

Β.Βεσκούκης

Software architectural styles/patterns Distributed systems

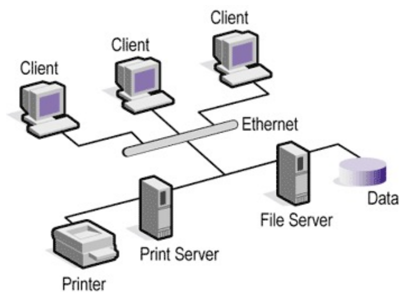
Reusable solutions in common problems
on software architecture

Software architecture

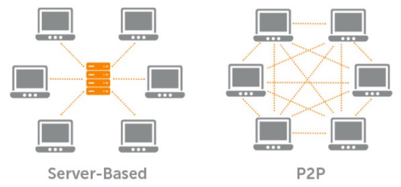
Fundamental structural and design decisions about software, which, once taken, is expensive: requires prohibitedly much effort, resources, etc. to change.

- What are the elements of a software system (structure)
- How are they connected to each other (interfaces)
- How are functional and non-functional requirements assigned to elements and interfaces (responsibilities)

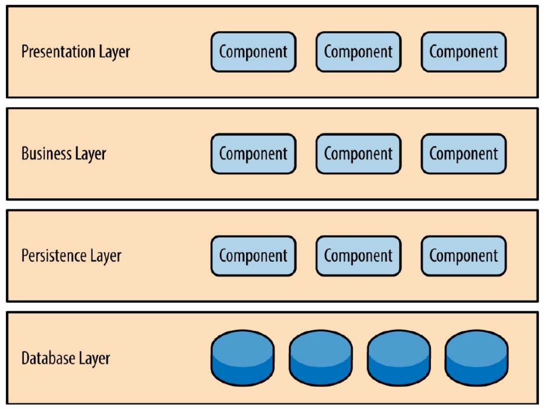
Client-Server



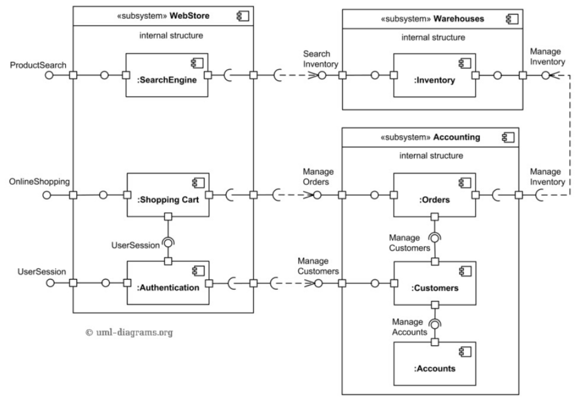
Peer-to-peer (P2P)



Layered/N-tier



Component-based

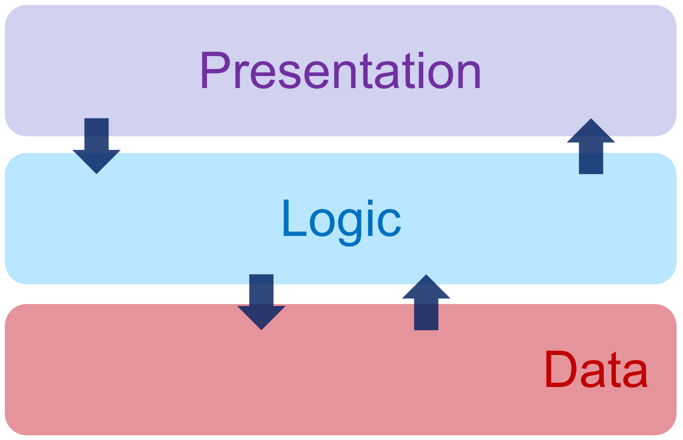


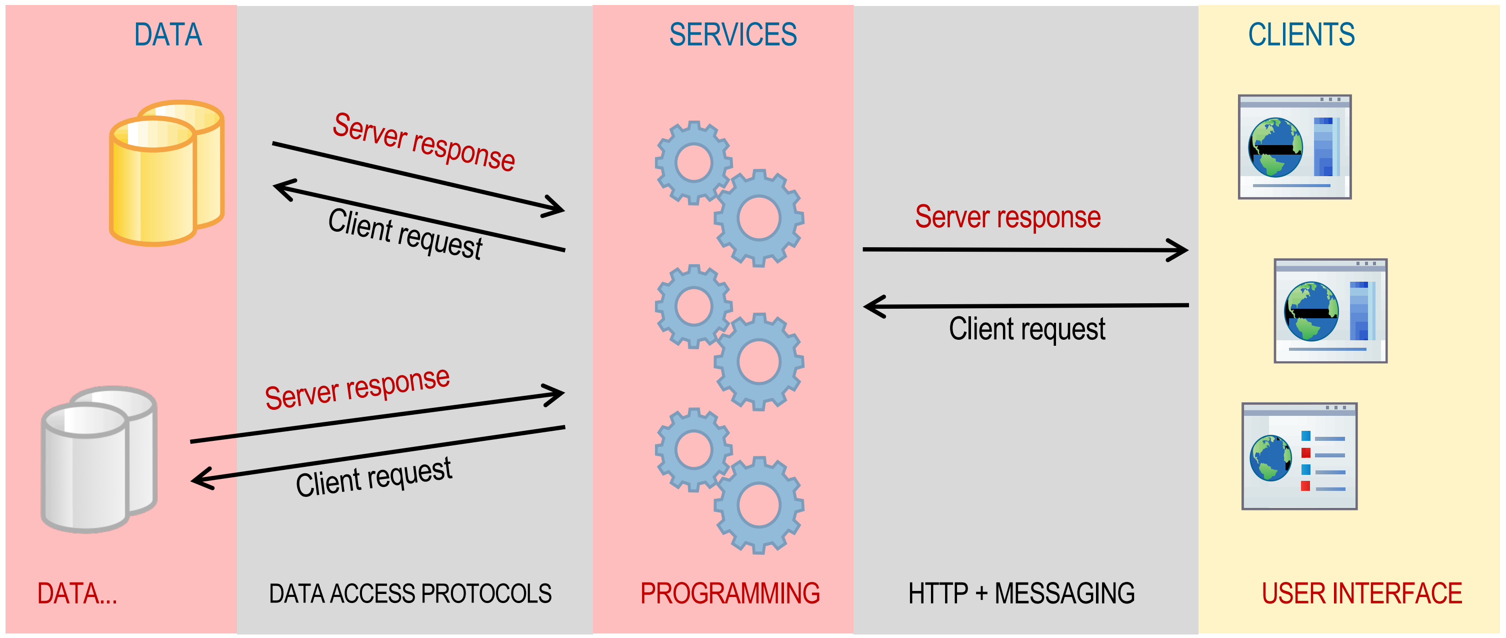
3-tier: data, processing, presentation

Data: What data is needed

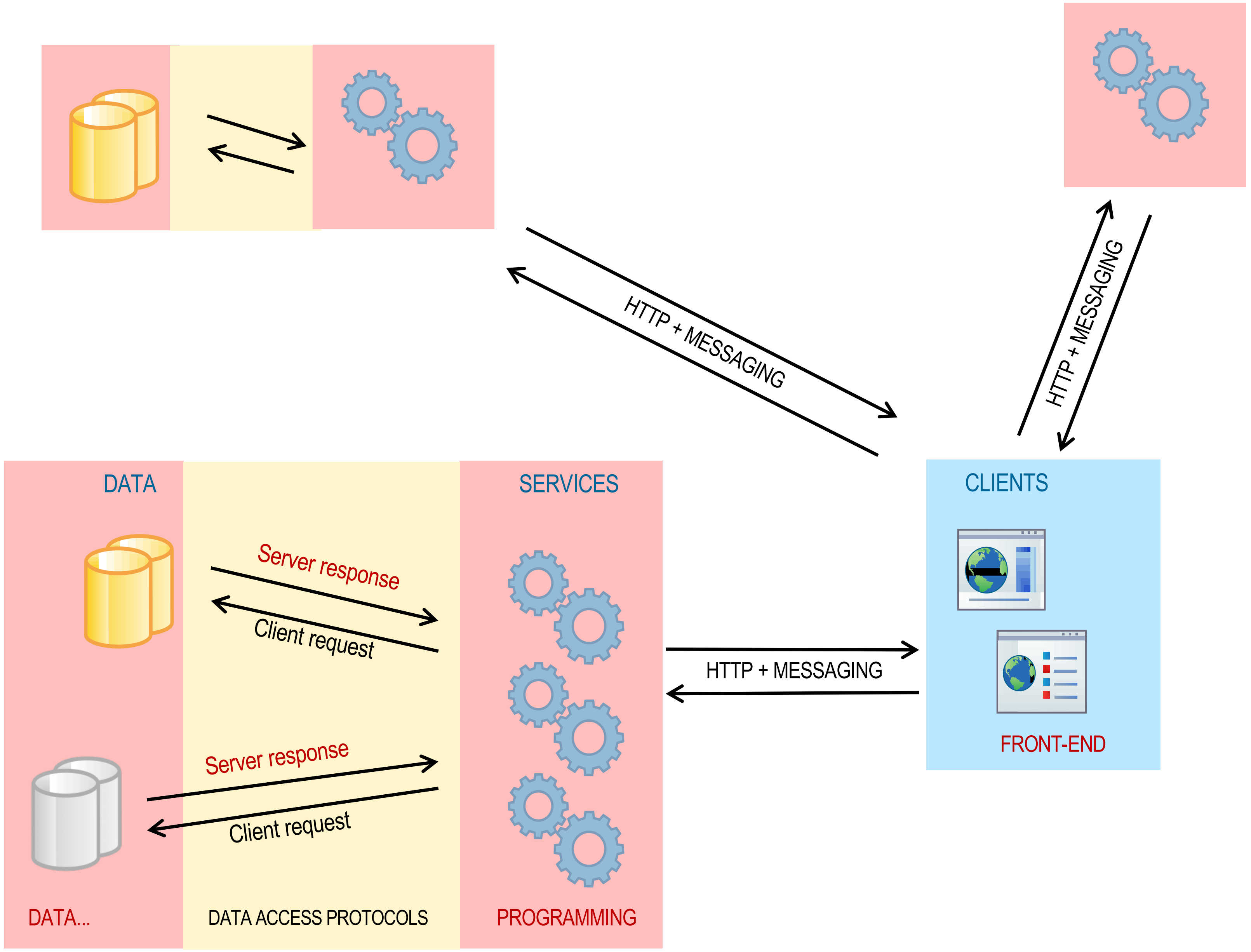
Logic: What to do with data

Presentation: Web, desktop, mobile



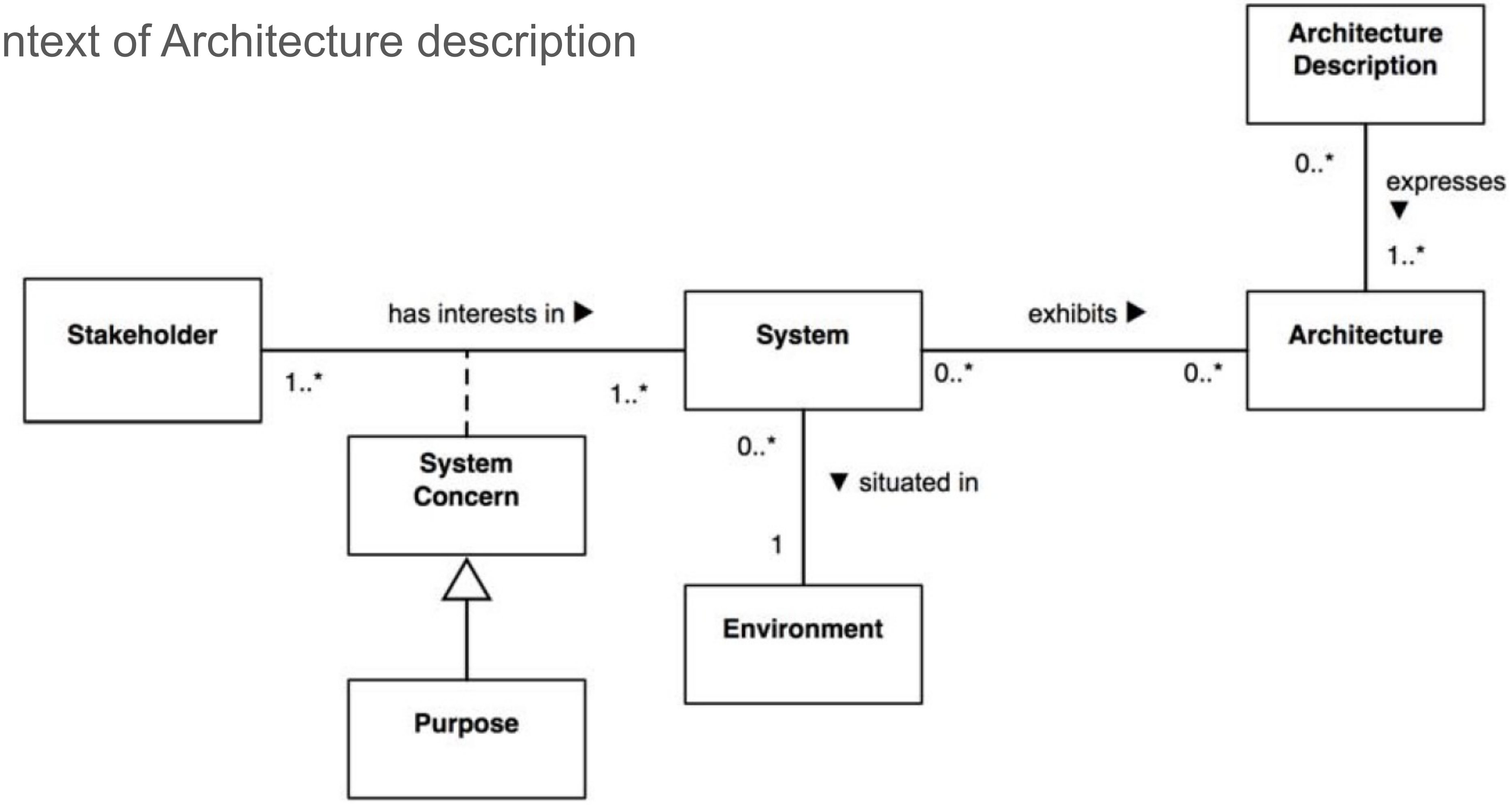


3-tier: data, processing, presentation



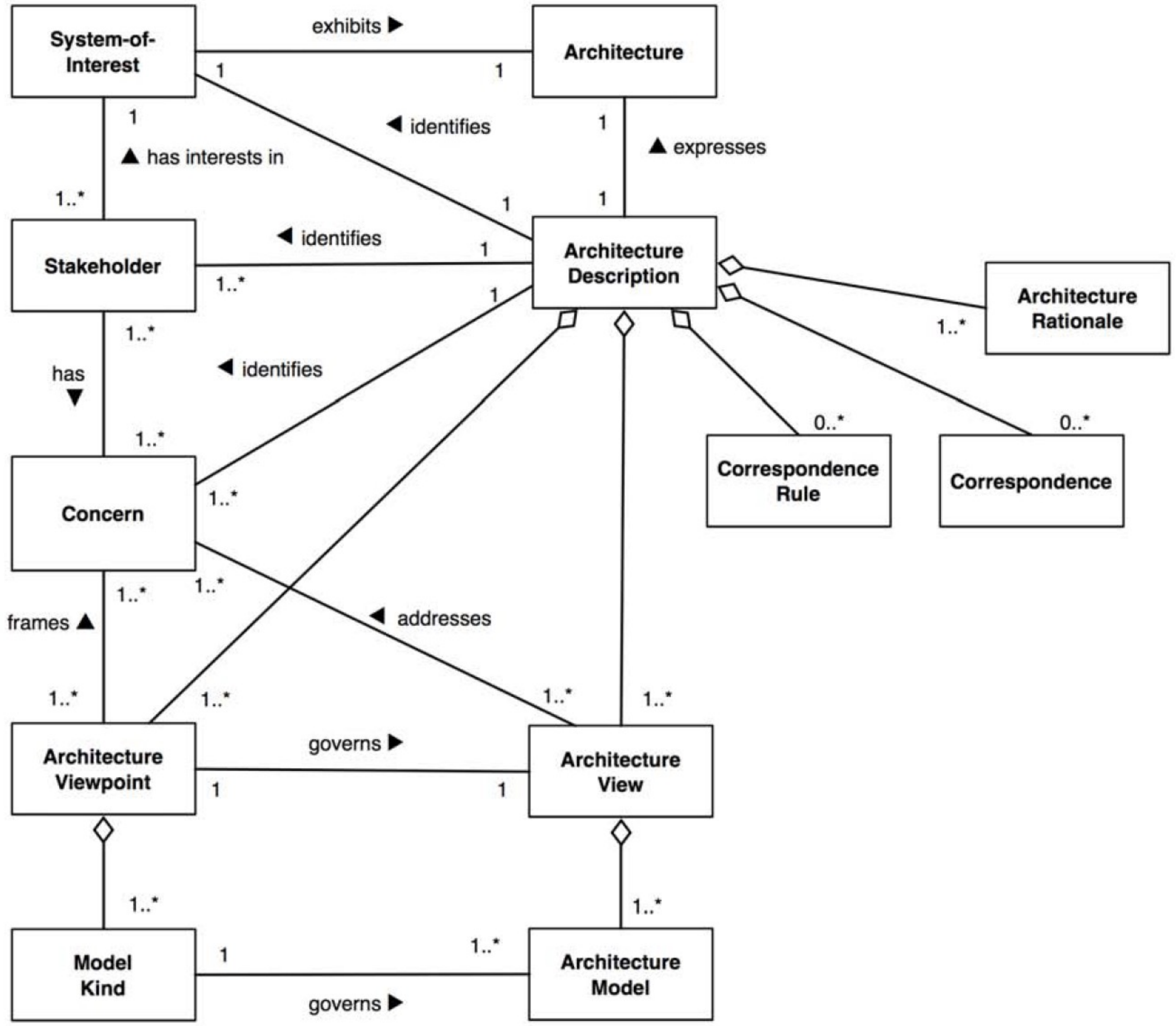
ISO/IEC/IEEE 42010:2011

Context of Architecture description



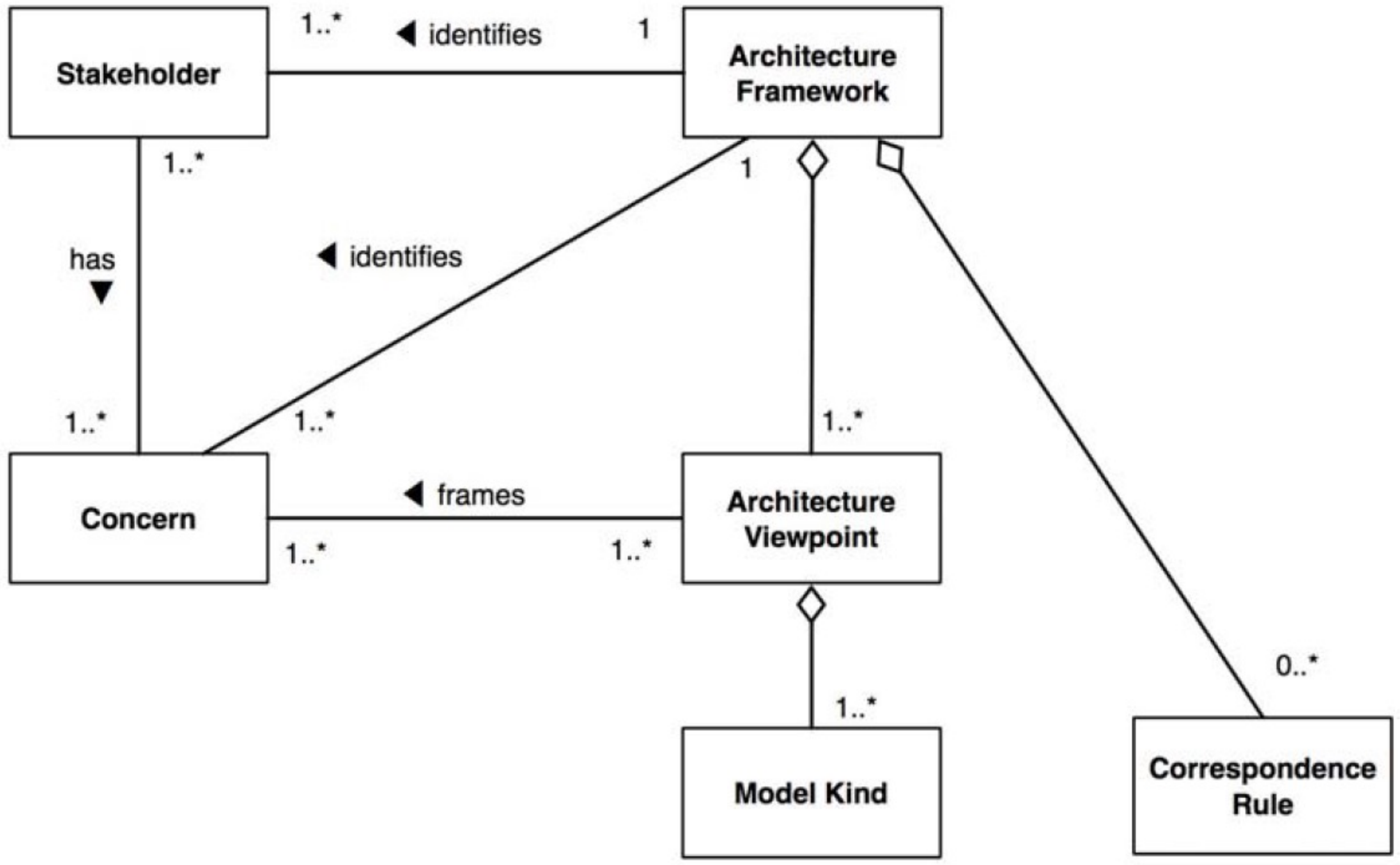
ISO/IEC/IEEE 42010:2011

Conceptual model
of Architecture description

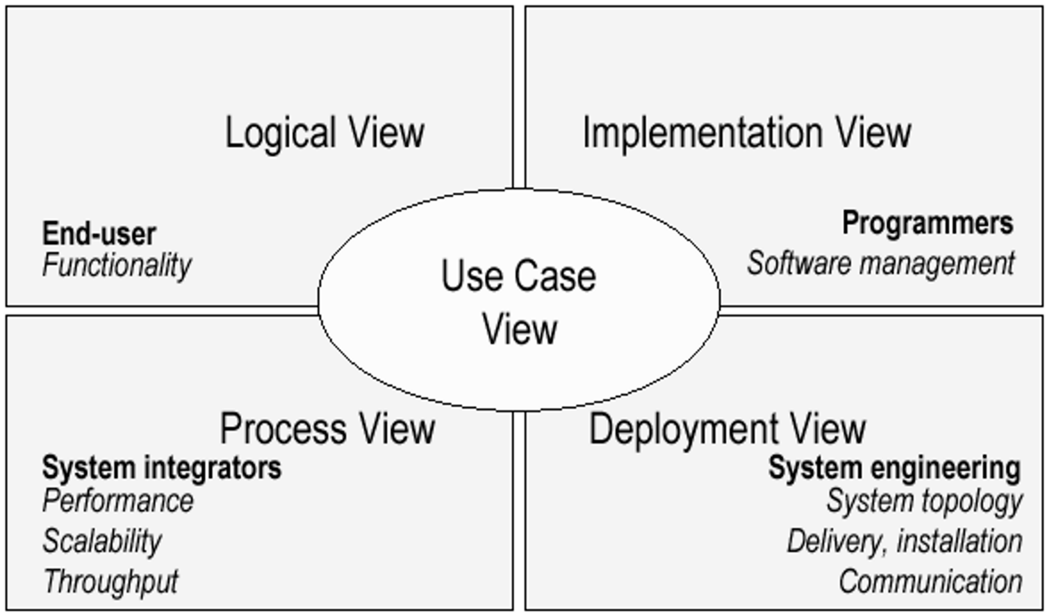


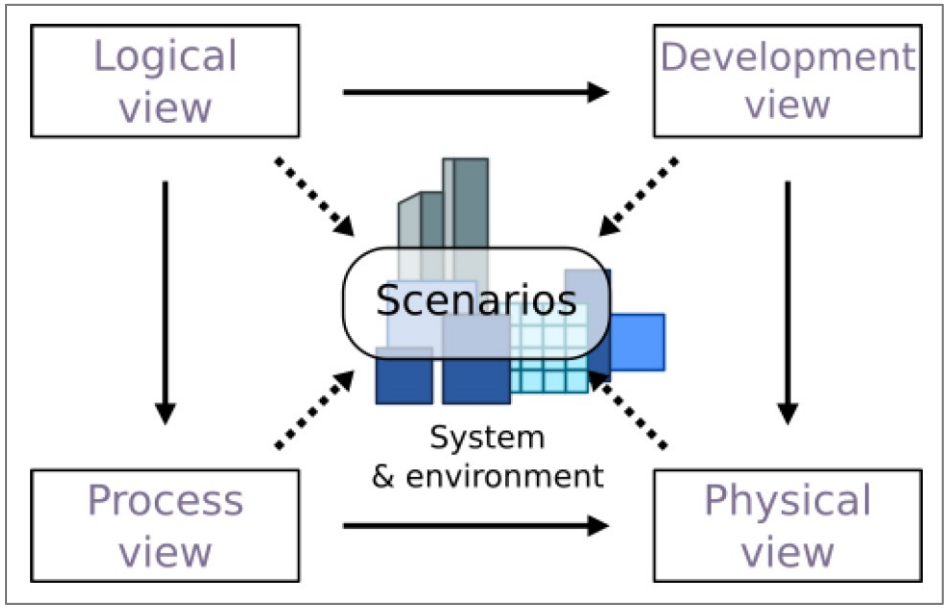
ISO/IEC/IEEE 42010:2011

Conceptual model
of Architecture framework



Architectural views





Architectural patterns & styles

Architectural style: a set of attributes that define some "identity"

Architectural pattern: a set of methods to implement a specific style

Not always easy to differentiate styles & patterns in software

- Specific development / operation environments
- Definition of possible subsystems
- Specific roles / responsibilities / attributes to each subsystem

Architecture & Design principles

SOLID principles

- **Single responsibility**: one software entity should be responsible for a single part of a program's functionality.
- **Open-closed**: software entities should be open for extension but closed for modification.
- **Liskov substitution**: if S is a subtype of T, then objects of type T may be replaced with objects of type S without altering any of the desirable properties of the program.
- **Interface isolation**: more specific interfaces are better that one "general".
- **Dependency inversion**: both high-level and low-level objects must depend on the same abstraction. Abstractions should not depend on details. Details should depend on abstractions.

Architecture & Design principles (repeated!)

Keep It Simple, Stupid Principle - KISS

You Ain’t Gonna Need It Principle - YAGNI

Don't Repeat Yourself Principle - DRY

Single Responsibility Principle - SRP

Open/Closed Principle - OCP

Liskov Substitution Principle - LSP

Interface Segregation Principle - ISP

Architecture & Design principles

You ain't gonna need it - YAGNI

- ..if it won’t come handy later, don’t do it now. Developers usually implement many things that they really don’t need. Implementing something which is not at all required now, will always consume cost, time, and effort...

Don't Repeat Yourself - DRY

- …programmers usually write enormous duplicate codes accidentally or un-accidentally. This principle forces us to avoid this habit. It means, every piece of knowledge in a system should have exactly one and unambiguous representation.

Architecture & Design principles

Single Responsibility Principle - SRP

- Code should be focused, narrow, and should do one thing and only one thing well. In context of object-oriented design, a class should have only and only one responsibility so that it has to change less frequently.
- Code should be cohesive: it should have one, and only one, reason to change.
- …if there is a piece of code that does several things, then that module has to change constantly which is more expensive because when a programmer comes to change it, he has to make sure that that changes do not break other things and really being affected and it becomes very expensive.
- …when a piece of code is broken into several pieces where each piece does one and only one thing well, then cost of changing that piece is much easier.

Architecture & Design principles

Open/closed

- A piece of code (class, module, or a component) should be open for extension and closed for modification.
- Classes, methods, or functions should be written in a way that it is ready for adopting/adding new features but not interested in any modification.
- In the context of object-oriented design, this could be achieved with the concept of Abstraction and Polymorphism.

Liskov substitution

- A function that uses references of parent classes must be able to use object of child classes as well without knowing it. Methods that can use superclass type must be able to work with object of derived class without any issue.
- The user of a base class should be able to use an instance of a derived class without knowing difference.

Architecture & Design principles

Interface Segregation Principle (ISP)

- A client should not be forced to implement an interface if they don’t need it. This happens when an interface contains more than one function and a client needs only one of them, but not all. The client should not need to implement all the functionality of that interface.
- One must always keep interface cohesive and narrow and focused on one and only one thing.

Items to notice

Not all architectures are possible in all IT ecosystems

Some architectural decisions are determined only by the runtime environment, which may be a non-technical decision itself

Quantification of quality attributes of software architectures is difficult!

Adoption of some architectural style does not exclude all the others: mixed architectures are possible and this is not always a bad idea

Distributed systems concepts

Fundamental attributes

- Consistency (**C**): The latest update of data should be available everywhere
  - Should other options exist?
- Availability (**A**): All requests should be answered
  - What happens if some request cannot be answered?
- Network partition (**P**): Networks may fail, systems may be divided
  - Should the system be operational if this occurs?

Distributed systems concepts

- Latency (**L**): Delays in desponse matter, should be ninimized
  - Can this be more measurable?
- Throughput (**T**): Requests served per time unit
  - The higher the better, right?
- Scalability (**S**): Possibility to scale up to handle increased requests
  - Buy more servers or what?

Distributed systems concepts

Relational DBMS "ACID" attributes

- **A**tomicity (transactions-centered)
- **C**onsistency (by construction)
- **I**solation (transation execution)
- **D**urability (persistence)

Non-relational databases "BASE" attributes

- **Ba**sically available (replication across nodes)
- **S**oft-state (consistency is implemented by developers – not the DBMS)
- **E**ventually consistent (finally!)

Common misconceptions in distributed systems

The network is reliable:
all networks requests are served correctly

Latency is zero:
the other end's response is immediate

Infinite bandwidth:
I can experience all the network speed I need and/or that has been promised to me

The network is secure:
No sniffing, identities are confirmed, no intruders

Common misconceptions in distributed systems

Topology doesn't change:
what is once there, will always be there

There is one administrator:
no comment needed

Transport cost is zero:
no fees for network traffic

The network is homogeneous:
services are available everywhere and behave in the same manner

Critical architectural and design decisions

What to consider

- Cost
- Time
- Quality
- Performance
- Availability
- Future-proofness

- Impacts on real-life workflows and business processes

Distributed systems theorems

CAP

It is impossible for a distributed system to provide more than two of:

- Consistency
- Availability
- Partition tolerance

CAP application example: metro ticketing system

CAP
It is impossible for a distributed system to provide more than two of: Consistency, Availability, Partition tolerance

Metro ticketing system (notes):

Consistency (**C**): The latest update of data should be available everywhere (Should other options exist?)
Availability (**A**): All requests should be answered (What happens if some request cannot be answered?)
Network partition (**P**): Networks may fail, systems may be divided (Should the system be operational if this occurs?)
Latency (**L**): Delays in desponse matter, should be ninimized (Can this be more measurable?)
Throughput (**T**): Requests served per time unit (The higher the better, right?)
Scalability (**S**): Possibility to scale up to handle increased requests (Buy more servers or what?)

The network is reliable: all networks requests are served correctly
Latency is zero: the other end's response is immediate
Infinite bandwidth: I can experience all the network speed I need and/or that has been promised to me
The network is secure: No sniffing, identities are confirmed, no intruders
Topology doesn't change: what is once there, will always be there
There is one administrator: no comment needed
Transport cost is zero: no fees for network traffic
The network is homogeneous: services are available everywhere and behave in the same manner

Distributed systems theorems

PACECL

If network partitioning (P) occurs then

- choose between Availability and Consistency

else

- choose between Latency and Consistency

PACECL application example: metro ticketing system

PACECL
If network partitioning (P) occurs then (choose between Availability and Consistency) else (choose between Latency and Consistency)

Metro ticketing system (notes):

Consistency (**C**): The latest update of data should be available everywhere (Should other options exist?)
Availability (**A**): All requests should be answered (What happens if some request cannot be answered?)
Network partition (**P**): Networks may fail, systems may be divided (Should the system be operational if this occurs?)
Latency (**L**): Delays in desponse matter, should be ninimized (Can this be more measurable?)
Throughput (**T**): Requests served per time unit (The higher the better, right?)
Scalability (**S**): Possibility to scale up to handle increased requests (Buy more servers or what?)

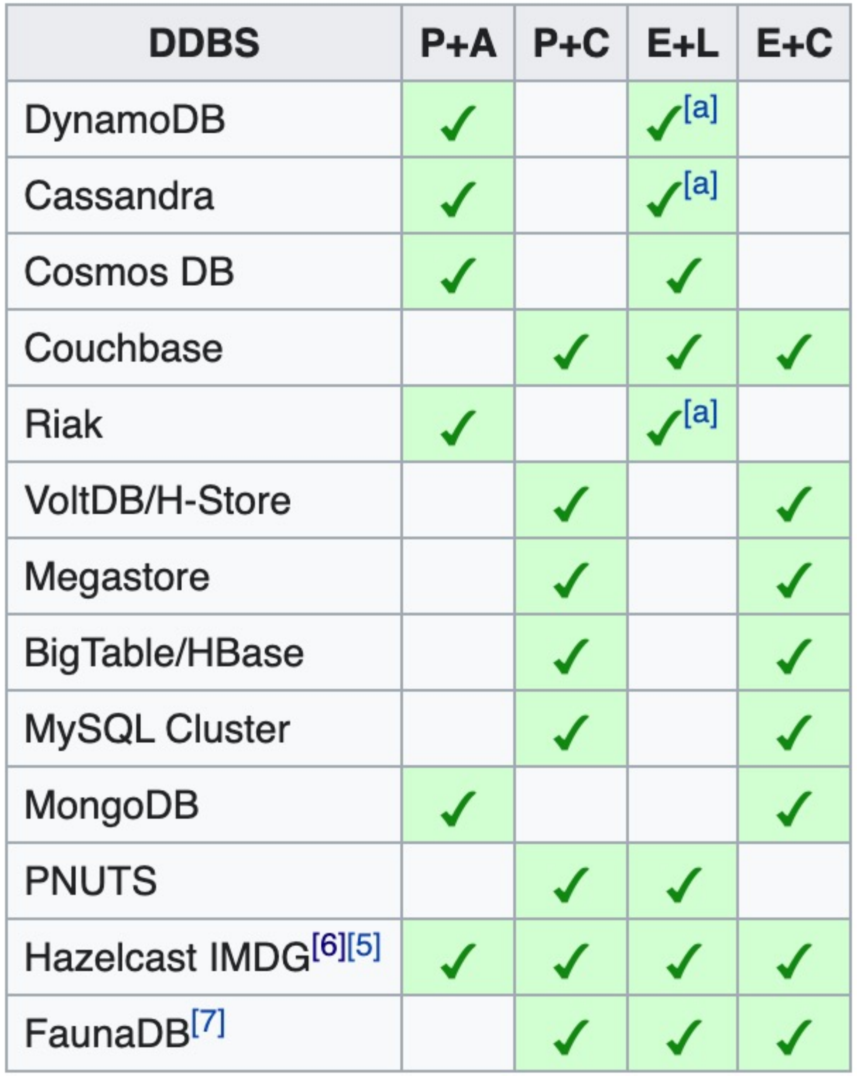
The network is reliable: all networks requests are served correctly
Latency is zero: the other end's response is immediate
Infinite bandwidth: I can experience all the network speed I need and/or that has been promised to me
The network is secure: No sniffing, identities are confirmed, no intruders
Topology doesn't change: what is once there, will always be there
There is one administrator: no comment needed
Transport cost is zero: no fees for network traffic
The network is homogeneous: services are available everywhere and behave in the same manner

A classification of distributed systems

PA/EL:
if (network partitioning) choose between Availability and Latency

PC/EC:
if (network partitioning) choose between Consistency and Consistency

PC/EL:
if (network partitioning) choose between Consistency and Latency



DDBS	P+A	P+C	E+L	E+C
DynamoDB	✓		✓[a]	
Cassandra	✓		✓[a]	
Cosmos DB	✓		✓	
Couchbase		✓	✓	✓
Riak	✓		✓[a]	
VoltDB/H-Store		✓		✓
Megastore		✓		✓
BigTable/HBase		✓		✓
MySQL Cluster		✓		✓
MongoDB	✓			✓
PNUTS		✓	✓	
Hazelcast IMDG[6][5]	✓	✓	✓	✓
FaunaDB[7]		✓	✓	✓

Source: wikipedia

Application example: metro ticketing system type?

PACECL
If network partitioning (P) occurs then (choose between Availability and Consistency) else (choose between Latency and Consistency)

Metro ticketing system (notes):

Consistency (**C**): The latest update of data should be available everywhere (Should other options exist?)
Availability (**A**): All requests should be answered (What happens if some request cannot be answered?)
Network partition (**P**): Networks may fail, systems may be divided (Should the system be operational if this occurs?)
Latency (**L**): Delays in desponse matter, should be ninimized (Can this be more measurable?)
Throughput (**T**): Requests served per time unit (The higher the better, right?)
Scalability (**S**): Possibility to scale up to handle increased requests (Buy more servers or what?)

The network is reliable: all networks requests are served correctly
Latency is zero: the other end's response is immediate
Infinite bandwidth: I can experience all the network speed I need and/or that has been promised to me
The network is secure: No sniffing, identities are confirmed, no intruders
Topology doesn't change: what is once there, will always be there
There is one administrator: no comment needed
Transport cost is zero: no fees for network traffic
The network is homogeneous: services are available everywhere and behave in the same manner

Typical styles

Client-Server

Component-based

Event-Driven

Layered / N-tier

Master-slave/Master-replica

Message-driven/Publish-subscribe

Microservices

Model-View-Controller (MVC)

Model-View-View-Model (MVVM)

Peer-to-peer (P2P)

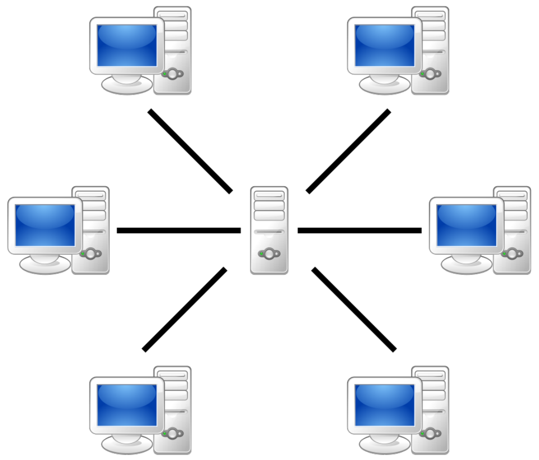
Pipeline / Pipe-filter

Representation State Transfer (REST)

Service-oriented

Share-nothing

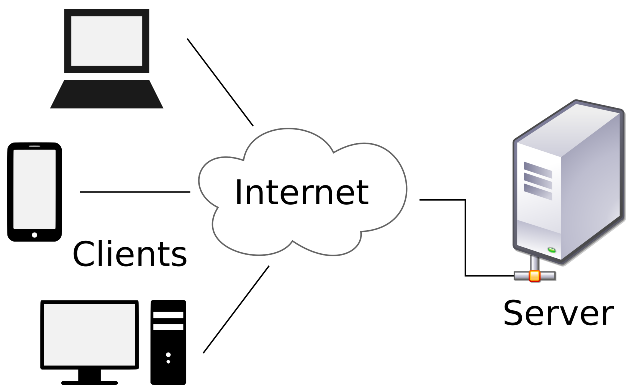
Client-server

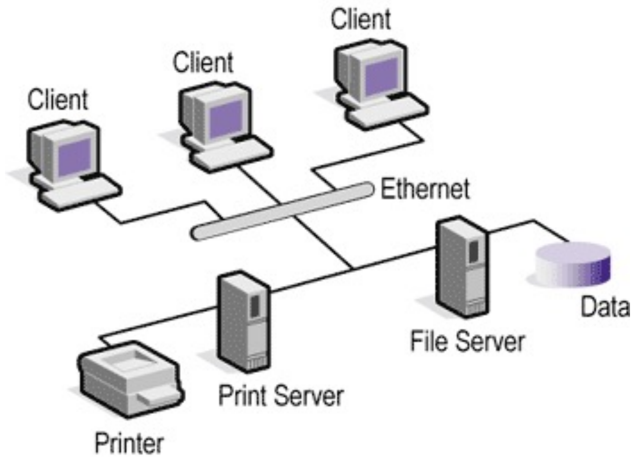


N clients, 1 server

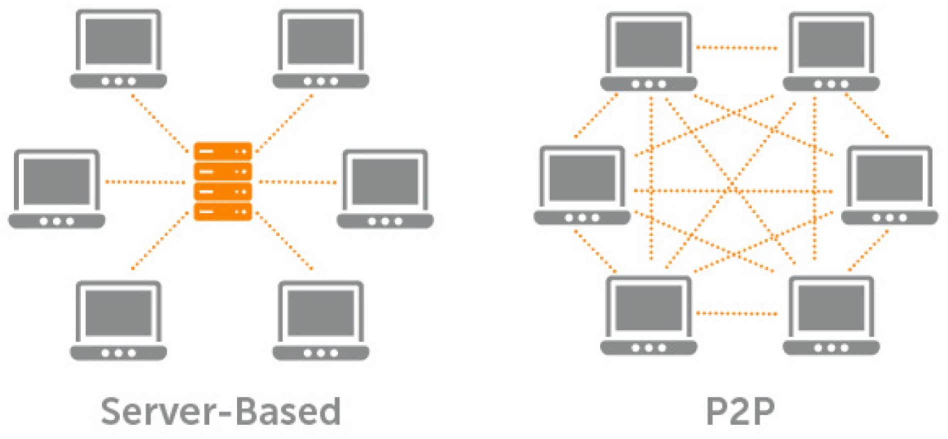
Communication based on some protocol

- File sharing, print sharing (early protocols)
- Access to DBMS
- HTTP, IMAP, POP3, FTP, SSH





Peer-to-peer (P2P)



Network of "peers"

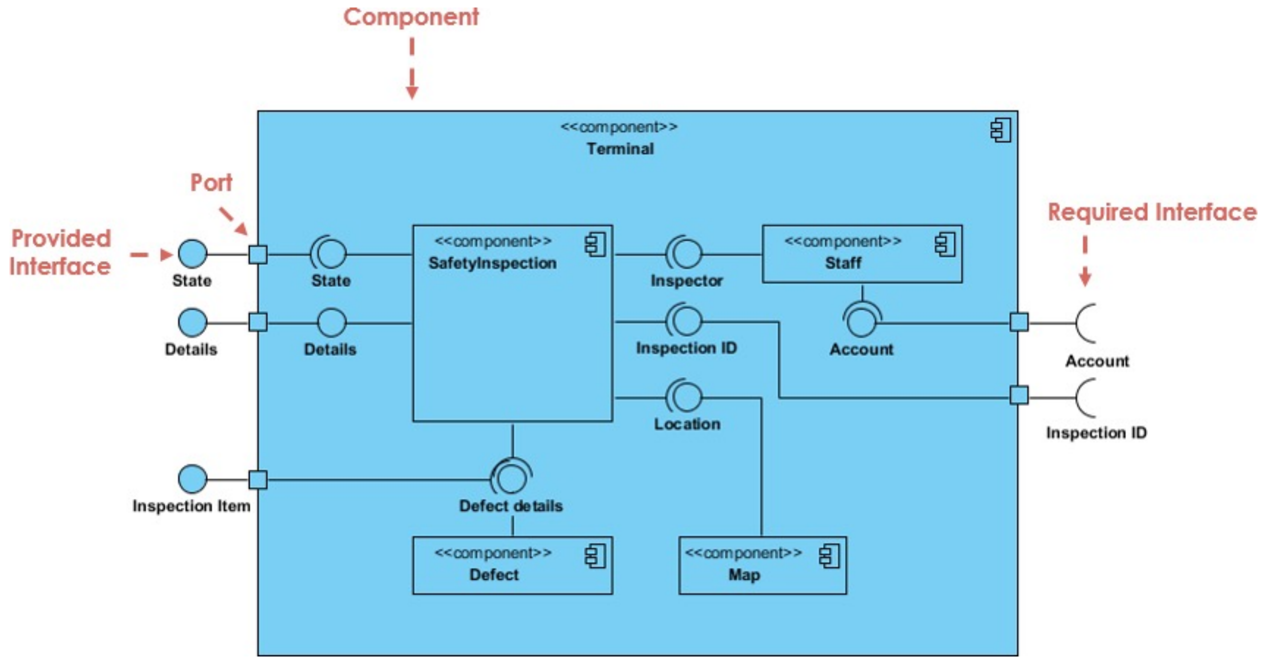
Every peer can be client and server

Communication based on some protocol

Old-time peers: samba

Modern peers: blockchain

Component-based



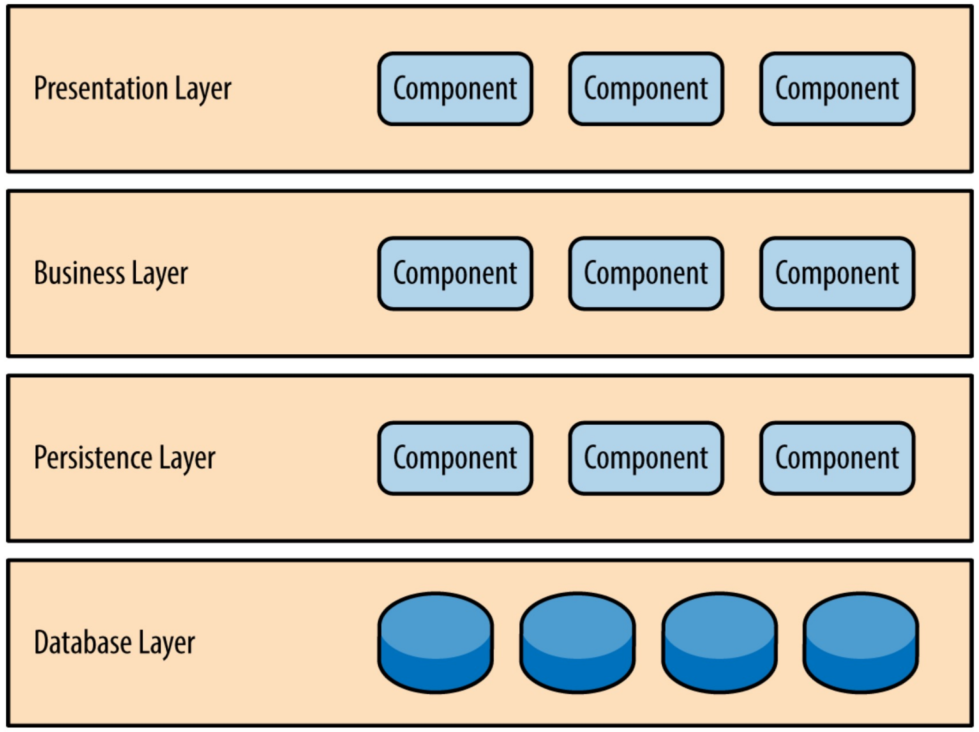
Interactions between component are possible over interfaces, which is the central design element

Components may both offer and consume interfaces

Loose coupling, separation of concerns

Components live in nodes, either small or big, e.g. application servers

Layered (n-tier)



Responsibilities/roles are assigned to servers (server-based)

Logical (e.g. virtual machines, docker, etc) or phisical servers

Internet application hosts

Frameworks for different ecosystems provide ready-to-use interfaces, components and layers

Component diagrams / Package diagrams

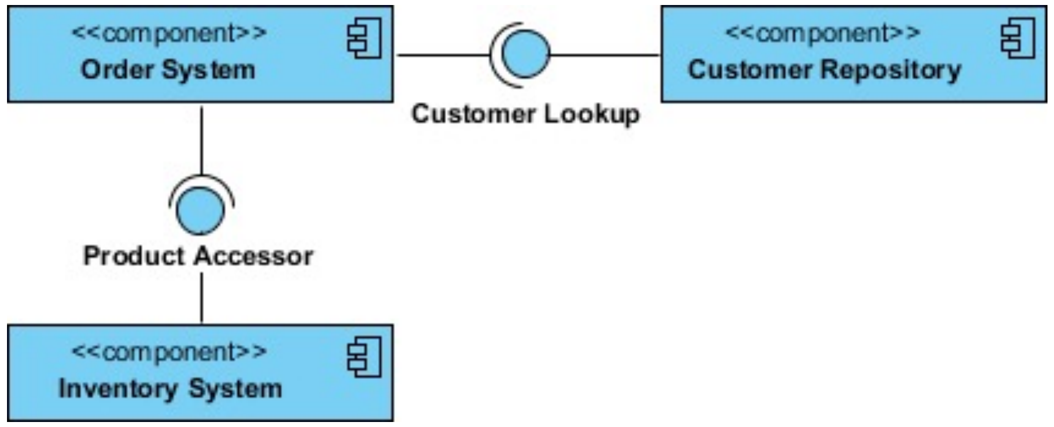
Display components in a system and their dependencies + interfaces

- Explain the structure of a system
- Usually a physical collection of classes

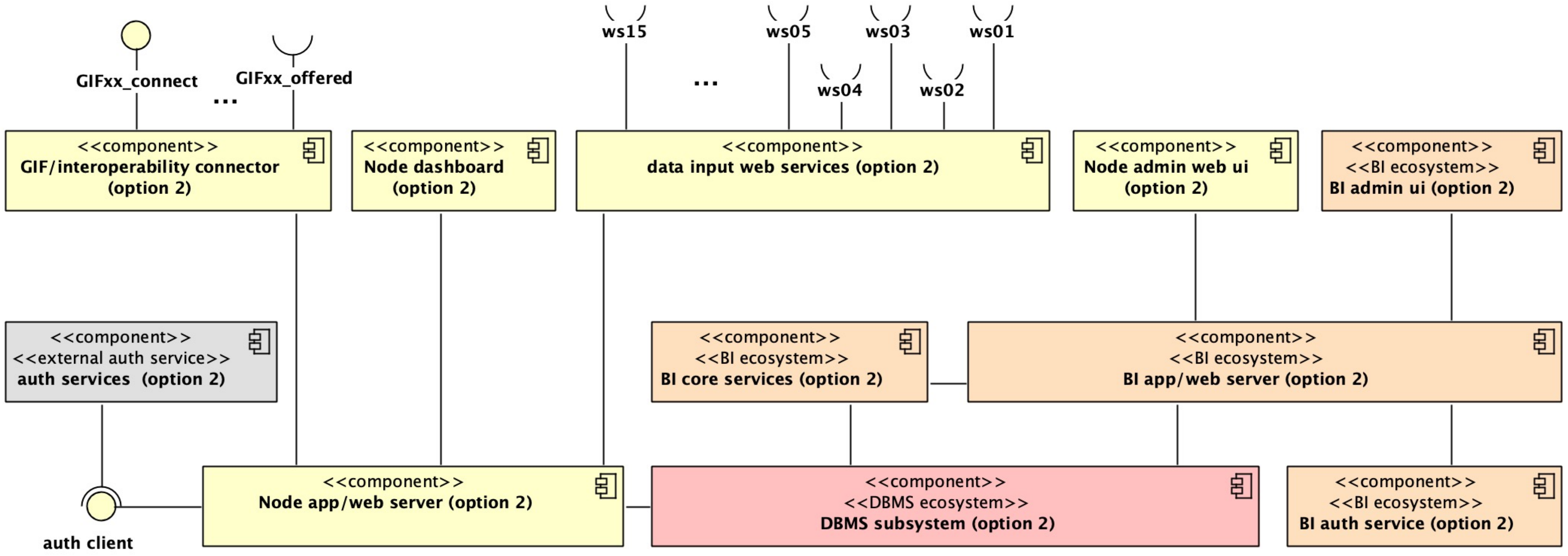
Component vs package Diagrams:

- Component: all of the model elements are private with a public interface
- Package: only display public items

Both are used to group elements into logical structures

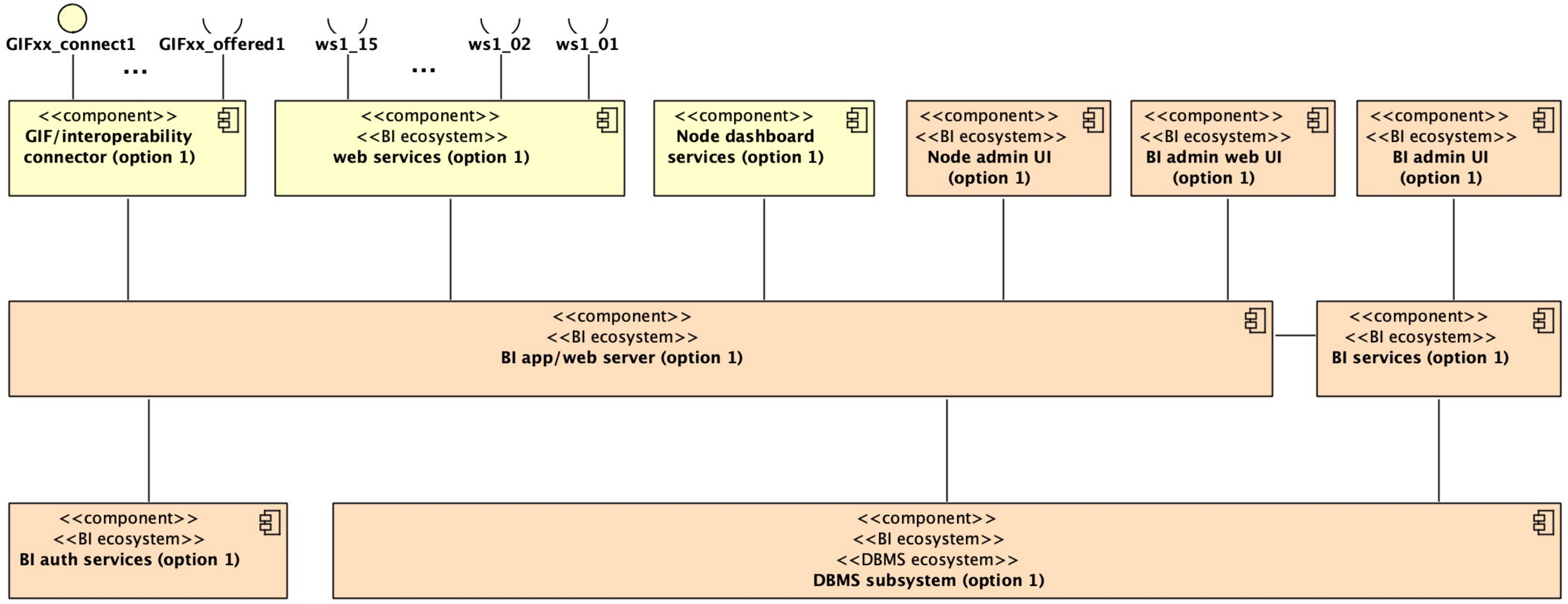


Component diagram examples



GIFxx_connect
...
GIFxx_offered
ws15
...
ws05
ws04
ws03
ws02
ws01
<<component>> GIF/interoperability connector (option 2)
<<component>> Node dashboard (option 2)
<<component>> data input web services (option 2)
<<component>> Node admin web ui (option 2)
<<component>> <<BI ecosystem>> BI admin ui (option 2)
<<component>> <<external auth service>> auth services (option 2)
<<component>> <<BI ecosystem>> BI core services (option 2)
<<component>> <<BI ecosystem>> BI app/web server (option 2)
auth client
<<component>> Node app/web server (option 2)
<<component>> <<DBMS ecosystem>> DBMS subsystem (option 2)
<<component>> <<BI ecosystem>> BI auth service (option 2)

Component diagram examples


GIFxx_connect1
GIFxx_offered1
ws1_15
...
ws1_02
ws1_01
...
<<component>> GIF/interoperability connector (option 1)
<<component>> <<BI ecosystem>> web services (option 1)
<<component>> Node dashboard services (option 1)
<<component>> <<BI ecosystem>> Node admin UI (option 1)
<<component>> <<BI ecosystem>> BI admin web UI (option 1)
<<component>> <<BI ecosystem>> BI admin UI (option 1)
<<component>> <<BI ecosystem>> BI app/web server (option 1)
<<component>> <<BI ecosystem>> BI services (option 1)
<<component>> <<BI ecosystem>> BI auth services (option 1)
<<component>> <<BI ecosystem>> <<DBMS ecosystem>> DBMS subsystem (option 1)

Deployment diagrams

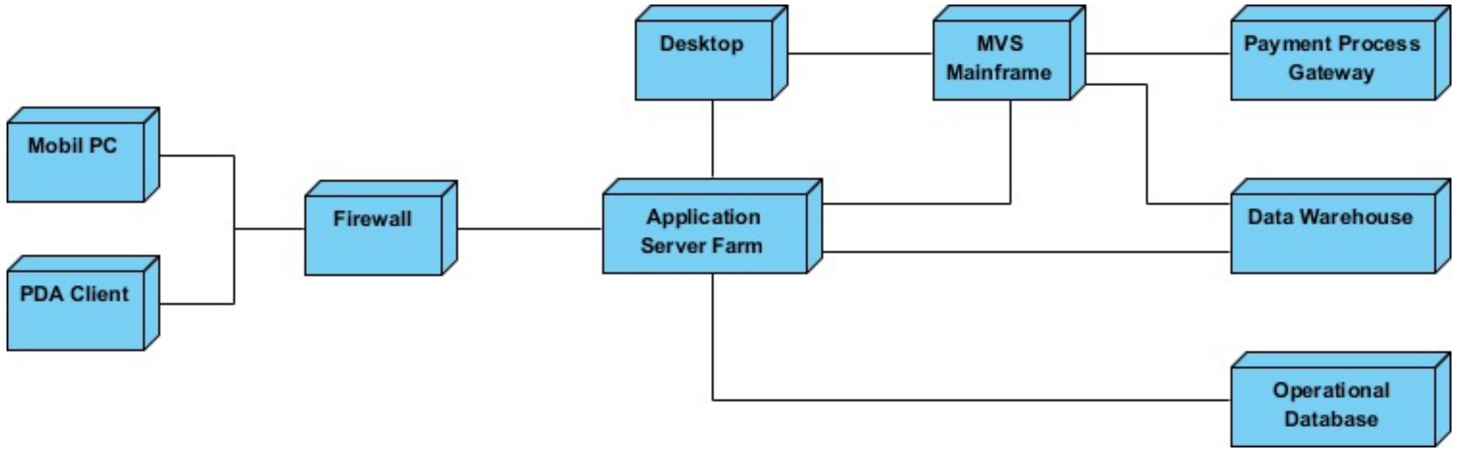
Show the physical architecture of the hardware and software of the deployed system

Nodes

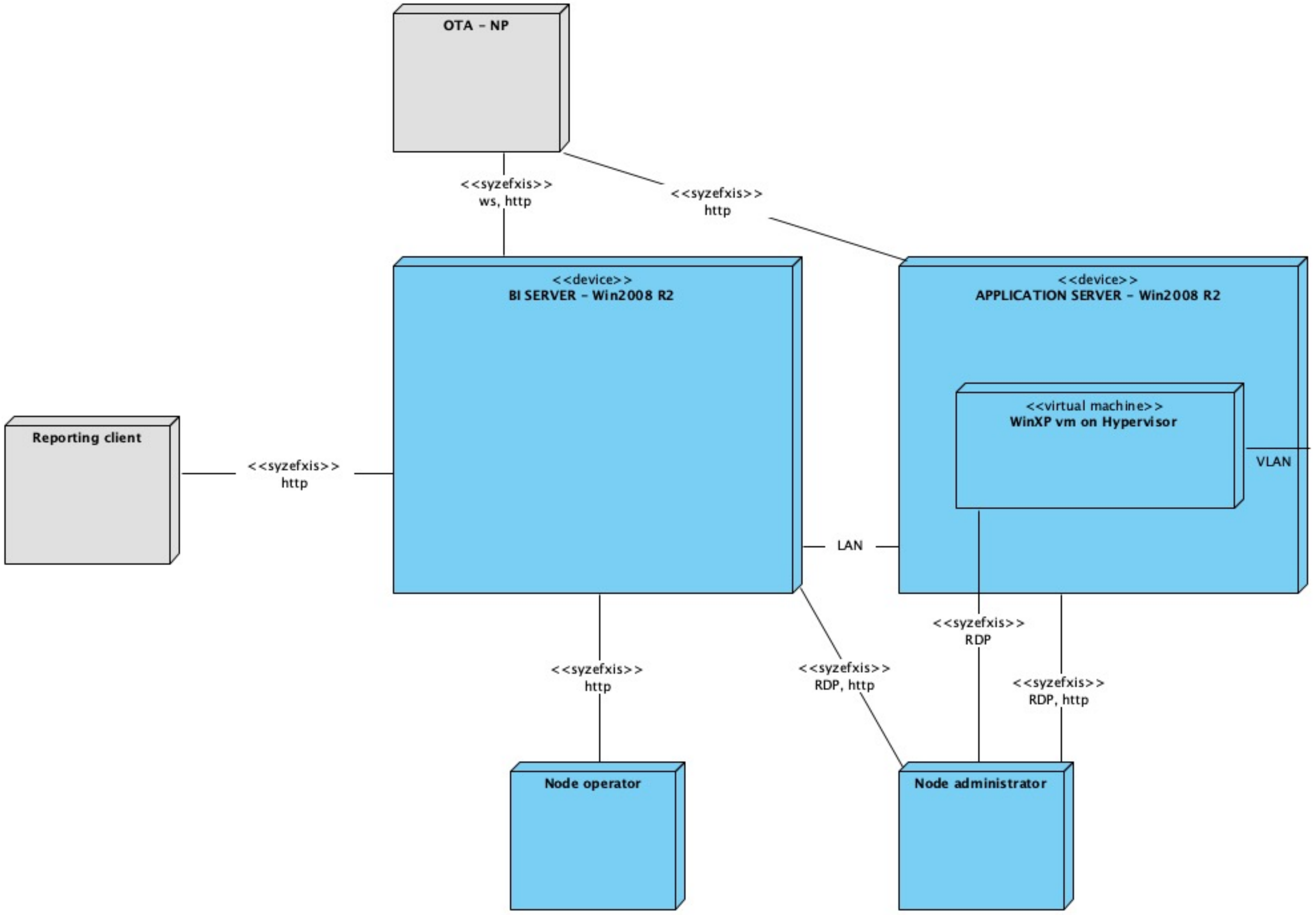
- Typically contain components or packages
- Usually some kind of computational unit; e.g. machine or device (physical or logical)

Physical relationships among software and hardware

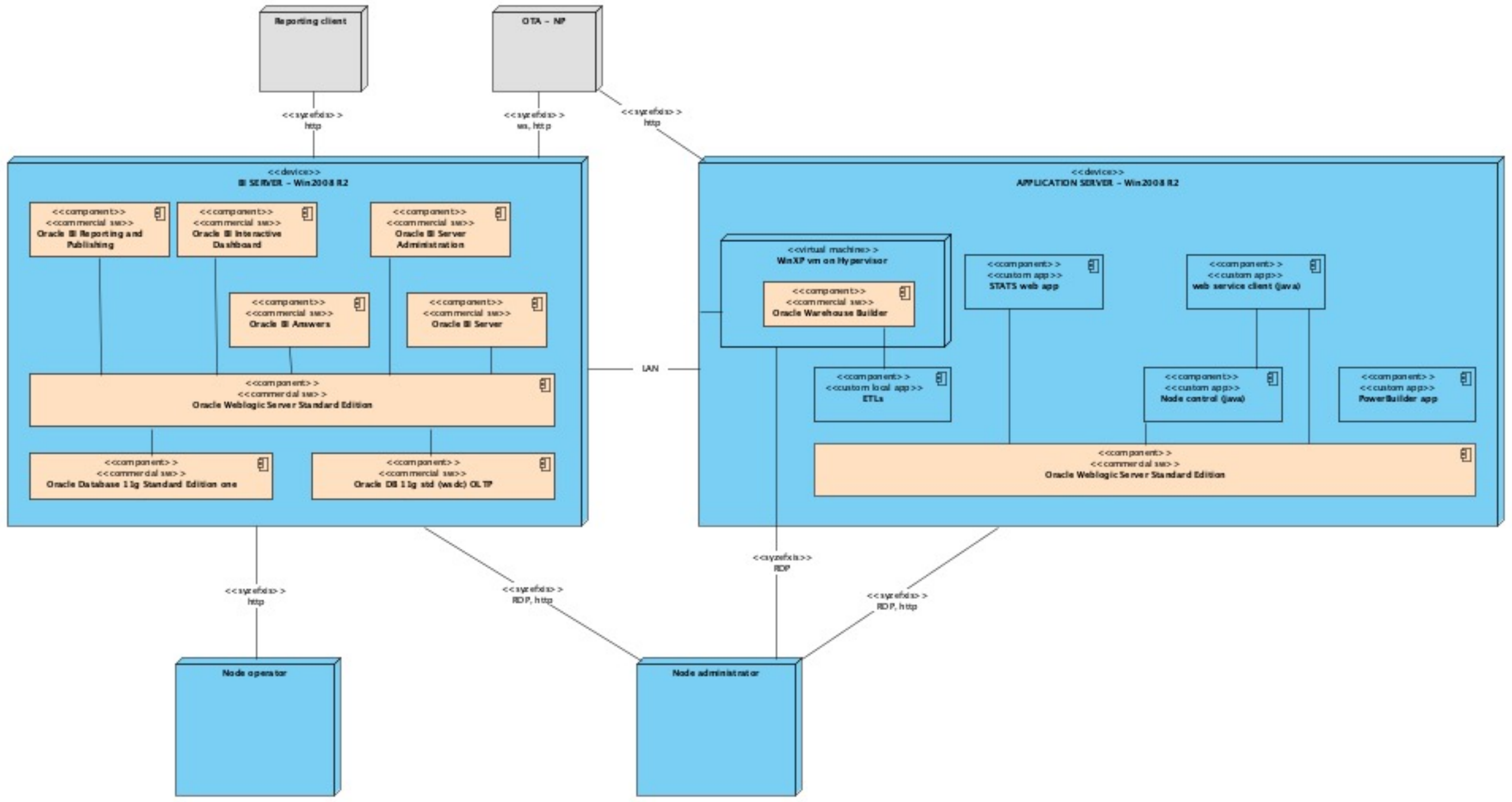
- Explain how a system interacts with the external environment



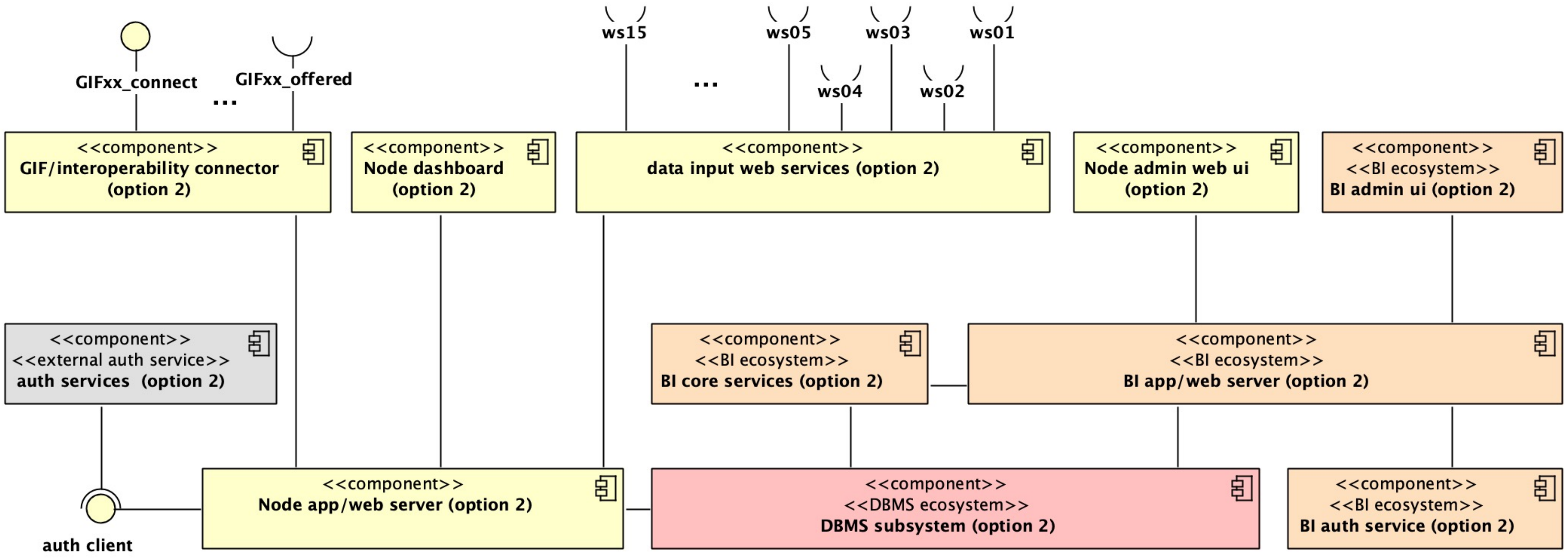
Deployment diagram example 1a



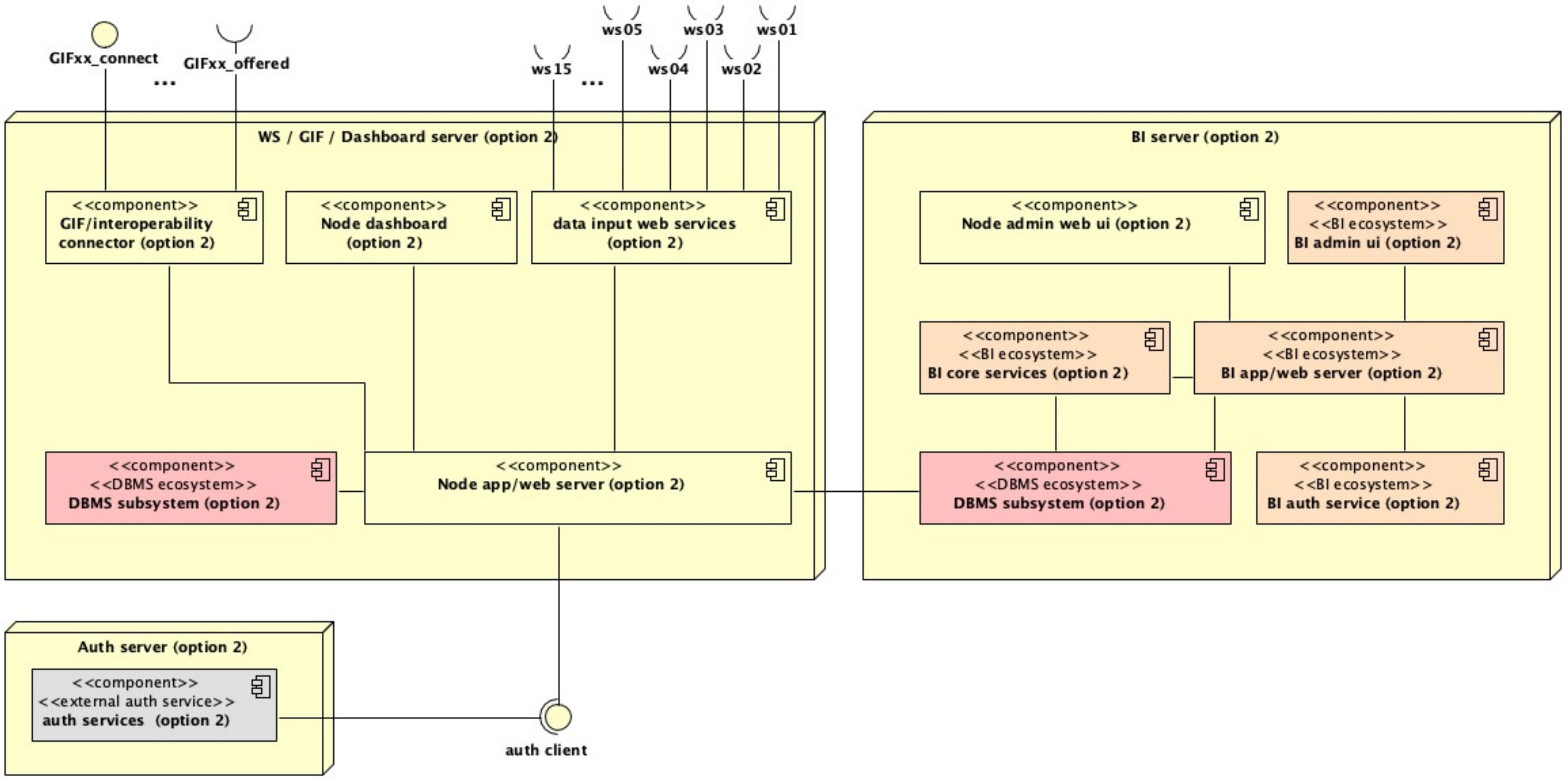
Deployment diagram example 1b


Reporting client
OTA – NP
<<syzefxis>> http
<<syzefxis>> ws, http
<<syzefxis>> http
<<device>> BI SERVER – Win2008 R2
<<component>> <<commercial sw>> Oracle BI Reporting and Publishing
<<component>> <<commercial sw>> Oracle BI Interactive Dashboard
<<component>> <<commercial sw>> Oracle BI Server Administration
<<component>> <<commercial sw>> Oracle BI Answers
<<component>> <<commercial sw>> Oracle BI Server
<<component>> <<commercial sw>> Oracle Weblogic Server Standard Edition
<<component>> <<commercial sw>> Oracle Database 11g Standard Edition one
<<component>> <<commercial sw>> Oracle DB 11g std (wsdc) OLTP
LAN
<<device>> APPLICATION SERVER – Win2008 R2
<<virtual machine>> WinXP vm on Hypervisor
<<component>> <<commercial sw>> Oracle Warehouse Builder
<<component>> <<custom app>> STATS web app
<<component>> <<custom app>> web service client (java)
<<component>> <<custom local app>> ETLs
<<component>> <<custom app>> Node control (java)
<<component>> <<custom app>> PowerBuilder app
<<component>> <<commercial sw>> Oracle Weblogic Server Standard Edition
<<syzefxis>> http
<<syzefxis>> RDP, http
<<syzefxis>> RDP
<<syzefxis>> RDP, http
Node operator
Node administrator

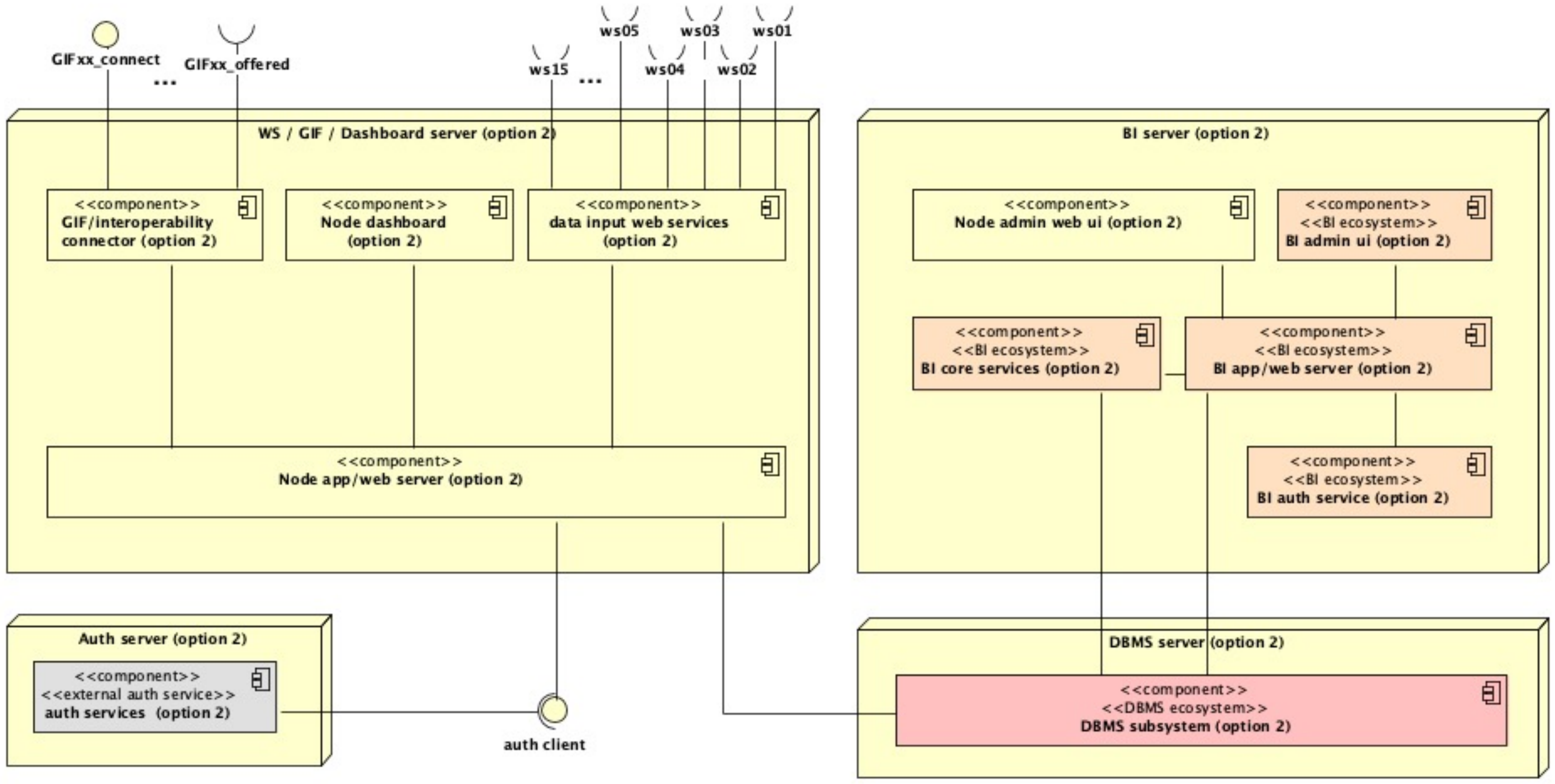
Deployment diagram example 2a


GIFxx_connect
...
GIFxx_offered
ws15
...
ws05
ws04
ws03
ws02
ws01
<<component>> GIF/interoperability connector (option 2)
<<component>> Node dashboard (option 2)
<<component>> data input web services (option 2)
<<component>> Node admin web ui (option 2)
<<component>> <<BI ecosystem>> BI admin ui (option 2)
<<component>> <<external auth service>> auth services (option 2)
<<component>> <<BI ecosystem>> BI core services (option 2)
<<component>> <<BI ecosystem>> BI app/web server (option 2)
auth client
<<component>> Node app/web server (option 2)
<<component>> <<DBMS ecosystem>> DBMS subsystem (option 2)
<<component>> <<BI ecosystem>> BI auth service (option 2)

Deployment diagram example 2b


GIFxx_connect
GIFxx_offered
...
ws15
...
ws05
ws04
ws03
ws02
ws01
WS / GIF / Dashboard server (option 2)
<<component>> GIF/interoperability connector (option 2)
<<component>> Node dashboard (option 2)
<<component>> data input web services (option 2)
<<component>> <<DBMS ecosystem>> DBMS subsystem (option 2)
<<component>> Node app/web server (option 2)
BI server (option 2)
<<component>> Node admin web ui (option 2)
<<component>> <<BI ecosystem>> BI admin ui (option 2)
<<component>> <<BI ecosystem>> BI core services (option 2)
<<component>> <<BI ecosystem>> BI app/web server (option 2)
<<component>> <<DBMS ecosystem>> DBMS subsystem (option 2)
<<component>> <<BI ecosystem>> BI auth service (option 2)
Auth server (option 2)
<<component>> <<external auth service>> auth services (option 2)
auth client

Deployment diagram example 2c


GIFxx_connect
GIFxx_offered
...
ws15
...
ws05
ws04
ws03
ws02
ws01
WS / GIF / Dashboard server (option 2)
<<component>> GIF/interoperability connector (option 2)
<<component>> Node dashboard (option 2)
<<component>> data input web services (option 2)
<<component>> Node app/web server (option 2)
BI server (option 2)
<<component>> Node admin web ui (option 2)
<<component>> <<BI ecosystem>> BI admin ui (option 2)
<<component>> <<BI ecosystem>> BI core services (option 2)
<<component>> <<BI ecosystem>> BI app/web server (option 2)
<<component>> <<BI ecosystem>> BI auth service (option 2)
Auth server (option 2)
<<component>> <<external auth service>> auth services (option 2)
auth client
DBMS server (option 2)
<<component>> <<DBMS ecosystem>> DBMS subsystem (option 2)