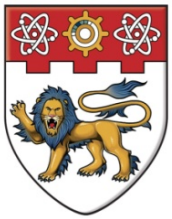


NANYANG TECHNOLOGICAL UNIVERSITY

Software Architecture Basic Designs & UML

presented by

Shar Lwin Khin
Research Scientist
SCSE
lkshar@ntu.edu.sg
N4-02c-76

Courtesy of Kevin Anthony Jones' slides

Foreword

- An foundational skill for junior architects is to recognize 'sw architecture' inferred in models showing only 'hw architecture'. Recall that (1) hw components contain sw components, and (2) applications can be run over networks with various arrangements of its constituent components . This lesson gives examples of basic architecture for common hw configurations including networks
- Three direct prerequisites to recognising sw architecture are sw components, UML, and sw reusables
- In today's lesson, we will walkthrough several common and hallmark examples of sw architecture, whose structure-only is modeled in UML
- Students may practice after the lesson; let's begin

As a student of SSAD course, by end of this session, you will be able to …

…model appropriately with UML, several basic designs of application architecture in standalone, networked, complex, and game engine configurations

Bloom's degree of learning

- Final suggested learning consideration: expected depth of learning for the topic. Eminent educationalist Benjamin Bloom identified six degrees of cognitive assimilation of new knowledge: first is the lowest, most superficial degree, and sixth is the highest, deepest, most pervasive
- The learning for this topic is third degree, application : student must correctly use facts, elements, and techniques of basic architecture designs to design and model different architectures in a consistent manner for different scenarios
- Assessing application: How to design a basic architecture for distributed software, ie., across network? How to correctly model my design using UML? How to model components being connected to but not being built? How to derive architecture design and UML model for sample descriptive diagrams? How would you model directories for reusable engines?
- Examinability: You are expected to design basic application architectures and model them with UML for junior-level case study/requirements

Prepare for incoming!

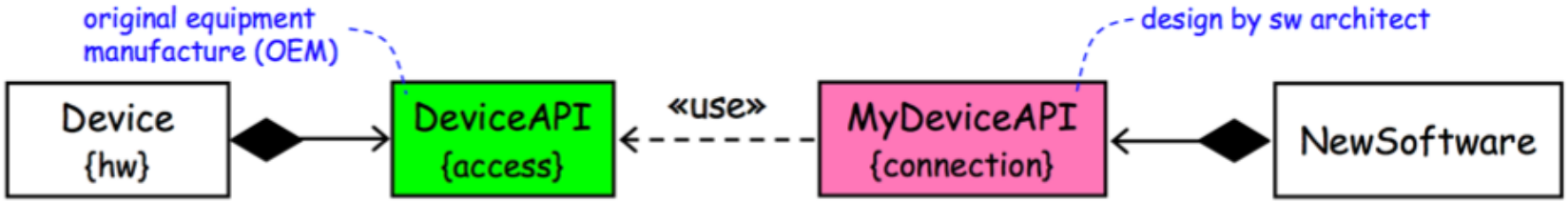
aka., activate your cognitive foundation for today's lesson

Remind me about API in architecture (sw components lesson)
Remind me, observer design pattern (course 2006)

Remind me, what is 'thin'/'fat' client? (architecture style lesson)
Remind me, offering functions for external use (course 2002)

Remind me about API in architecture

[recall Software Components] Per the system perspective aspect of software architectural thinking, hardware is separated from the software via the 'application programming interface' (API). The API resides on the hardware and contains a set of protocols and exposed operations for the hardware. The new software's architect builds a "connection" Component that invokes the operations required by the new software. By convention, this component is named "MyDeviceAPI" to plainly announce its intended service within the new software [see below]

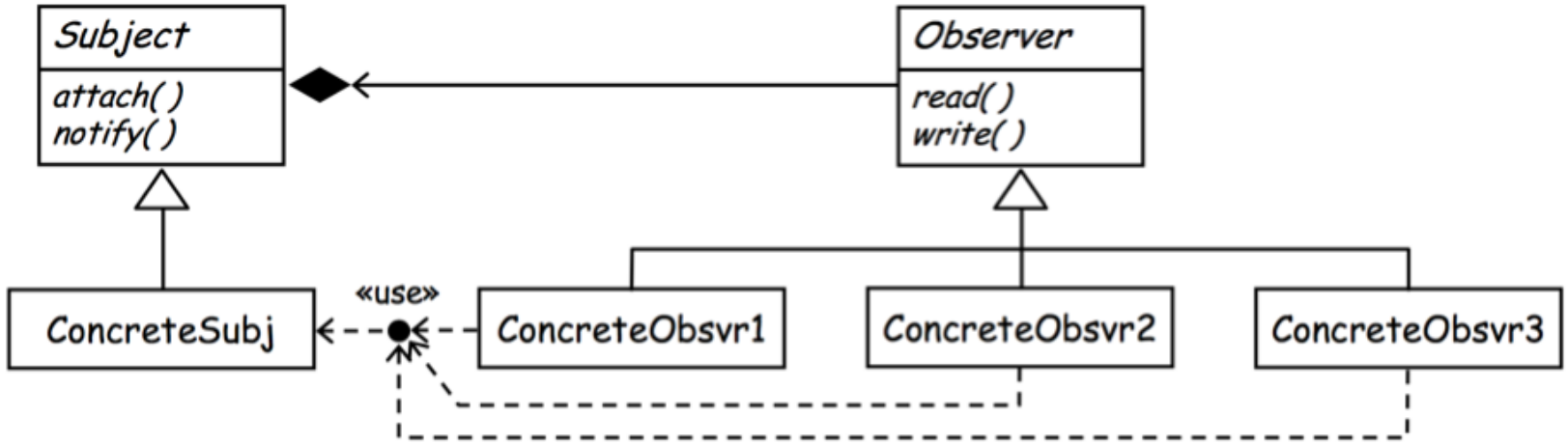


More than just for hardware, APIs are incorporated into the design of reusable software like frameworks and (game) engines so that their functionality can be extended into the new software. APIs are also attached to shared services so that the new software can attain a standard capability deemed necessary by the owning organization with negligible risk. Typically, these shared services are offered on a computer network

Remind me, observer design pattern

From course 2006 "Software Engineering", the observer pattern is the engine for the model-view-controller architecture style [see below]

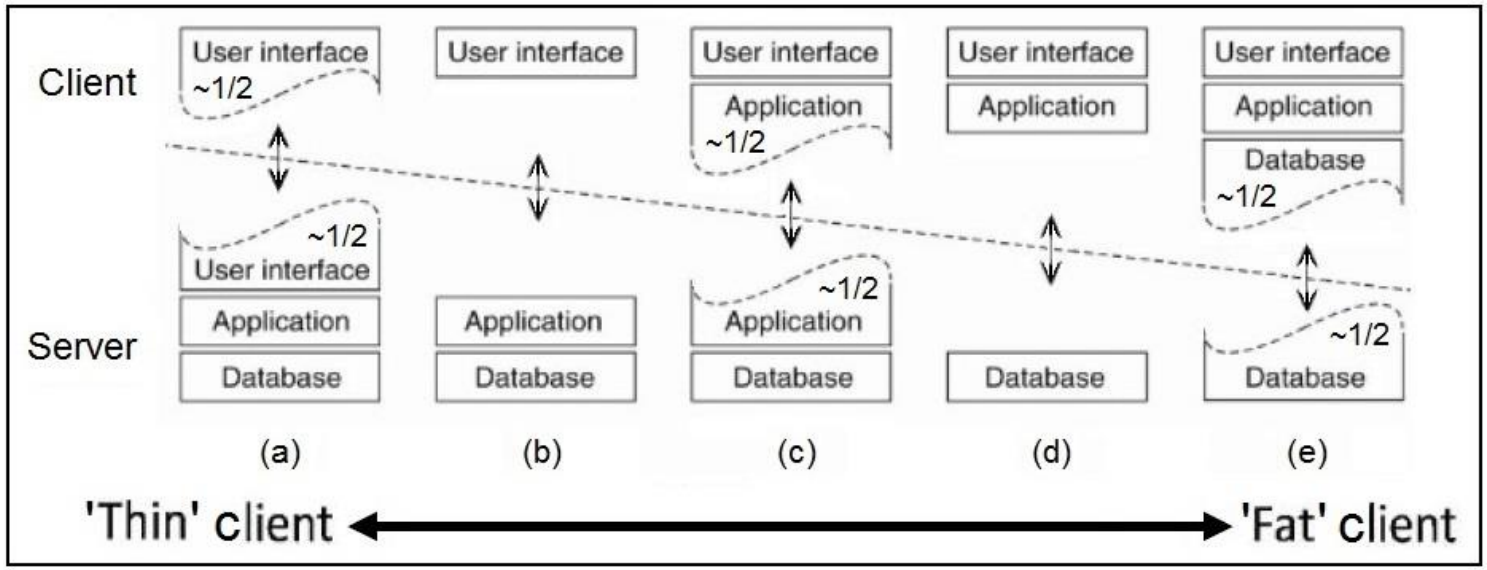
Observer is a behavioural design pattern that implements a distinct logic boundary between the user interface and business rules. It has one subject and many independent observers. The subject automatically notifies its observers of relevant state changes. Its weakness is memory leakage known as the lapsed listener problem: observers being kept alive by the subject, as it is strongly referencing them, and with an explicit deregistration not done. The observer pattern is implemented in numerous programming libraries, GUI toolkits, and frameworks



Remind me, what is 'thin'/'fat' client?

[recall architecture style] A key consideration in the client-server architecture style is how the application's split between client and server is proportioned. It need not be equal; various different proportions are feasible [see below], and depend on the rationale for incorporating that style, eg., if ...

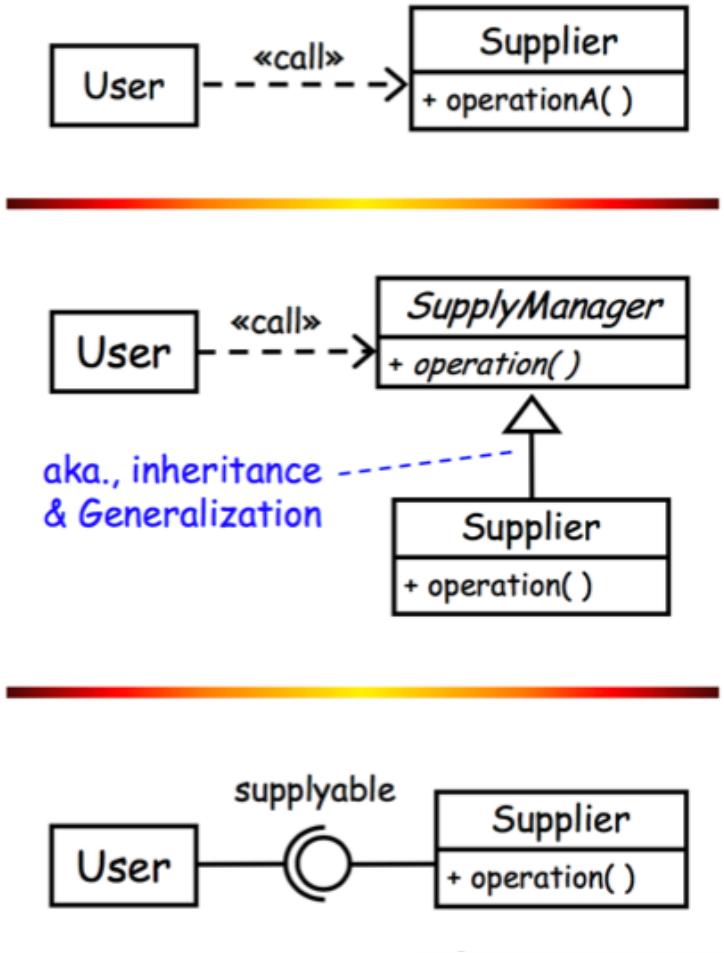
- one part of the application — usually the server — is to be offered as a shared service, the design of the service will decide the split proportions
- The application's client-server parts are installed into different machines, the hw's capacity, configuration, and processing power normal decide the split proportions



Remind me, offering functions for external use

From course 2002 "Object-oriented programming", there are several approaches that using objects follow to invoke behaviour offered by supplying objects. Three will be considered for this module: direct calling, extension, and interface

1. Direct calling [see upper right]–using object calls a concrete Operation directly from the supplying object
2. Extension [see middle right]–using object calls an abstract Operation from an abstract Class (with Properties) that triggers a process to identify the proxy supplier (concrete child to the abstract parent). Once identified, the behaviour is invoked under the control of the using object as if it had called it directly
3. Realization [see lower right]–same as extension except that the abstract Operation is contained in an Interface (with nil Properties)that the supplying object implements



Where does this fit?

aka., appreciating the big picture

Other UML keywords for Components

Software architecture shows connection to network

Full design of network entry-exit

Other UML keywords for Components

Recall the use of keyword in UML for the box notation is integral and mandatory to distinguish the particular structure concept being represented by that oft-repeated graphic shape. In the Component diagram, keyword="component" specifies that the box is representing a set of classes. That this semantic clearly emerges from the architecture diagram is greatly significant for all follow-on designers

The software architect is always questioning the informative-ness of the architecture model. But when it comes to replacing "component" with user-designated keyword to portray the custom nature of some component, using only official keywords from the UML specification [see next slide] is strongly encouraged. These are popular and meaningful official keywords for specialized semantics

- "subsystem" –many Classes that run as one coherent and executing program, ie., like a ".exe., ".com", or ".sys". Also, since any system may in turn be a subsystem for a larger system, the finality of the term "system" is actually unnecessary and not official
- "service" –on-demand particular and singular function that does not keep final result "entity" –definitive and persistent data for consistent running of program
- "process" –individual indivisible part of a chain of operations in which each part must succeed plus all parts must succeed for the transaction to be successful

... UML keywords for Components (cont)

Keyword	Meaning	Header in
buildcomponent	system level dev activity, eg. versioning	box
component	set of classes imbued w/structure+behaviour	box
entity	cmpnt defining/having biz concept data	box
framework	cmpnt being a commercial framework product	box
implement	cmpnt of program lang (physical) w/o design	box
process	cmpnt that is transaction-based	box
provided interfaces	list of <these>	compartmnt
realizations	list of realizing (physical impl) Classes	compartmnt
required interfaces	list of <these>	compartmnt
service	cmpnt of stateless specific fcn, eg., calculator	box
specification	cmpnt of design w/o physical essence	box
subsystem	cmpnt in hierarchy of larger system	box
utility	cmpnt of stateless common/reusable static fcns	box

Sw architecture shows connection to network

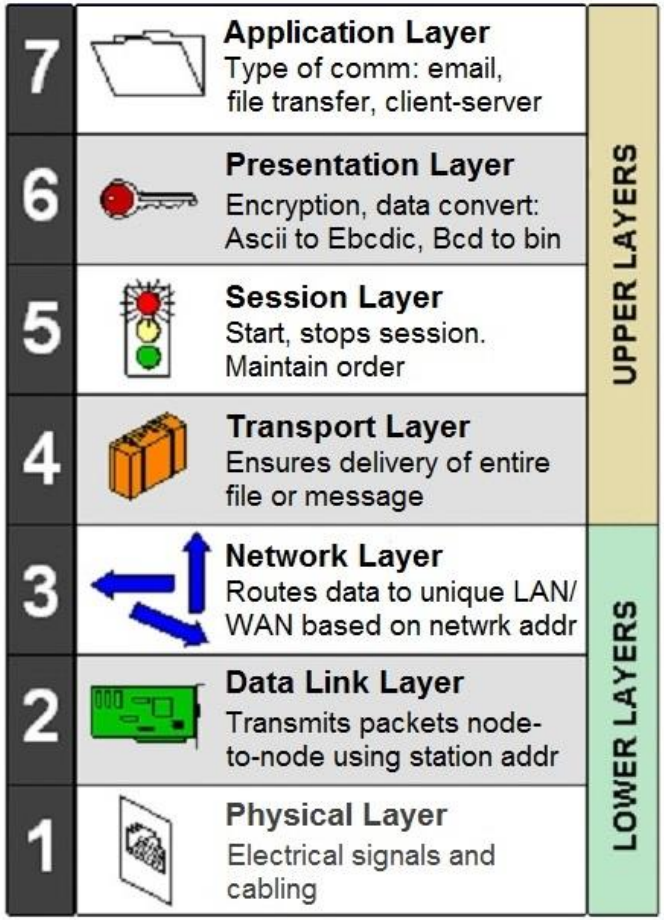
[see right] Software architecture of every computer network is designed according to Open Systems Interconnection (OSI) Reference Model, aka., OSI 7498 and X.200. It comprises:

- basic reference or seven-layer model
- set of specific protocols for each layer

Software in each layer implements its chunk of functionality, interacting directly and only with the layer immediately beneath, while providing facilities for use by the layer above. Protocols ensure the functions in the receiving computer's layer are undone inexact opposite order to the way they were done in the same layer of the sending computer, as if the sender fills up a stack, then the receiver empties it

OSI model does not specify the interfaces inside computers like network sockets; these are implementation dependent

This full form is commonly called "interconnection stack"

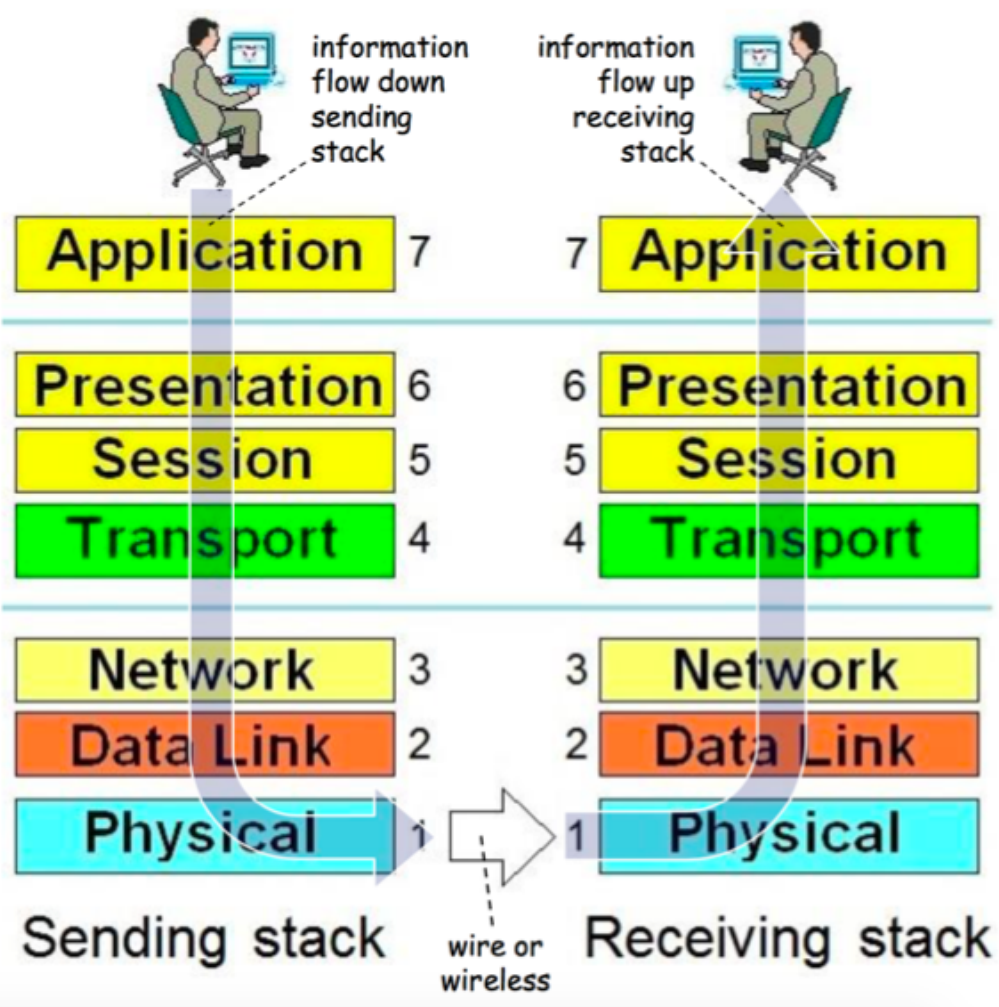


... arch shows connection to network (cont)

When applications communicate across a computer network, they do so as sending and receiving interconnection stacks [see right].

Information travels "down" through sending stack's layers, then out of that stack, across to the bottom of the receiving stack and "up" through its corresponding layers to its top. OSI protocols at each inter-layer boundary enact a virtual communication directly between corresponding layers.

Hence, the application's architecture has to address two connections, ie., into: 1) its stack, and 2) the receiving application



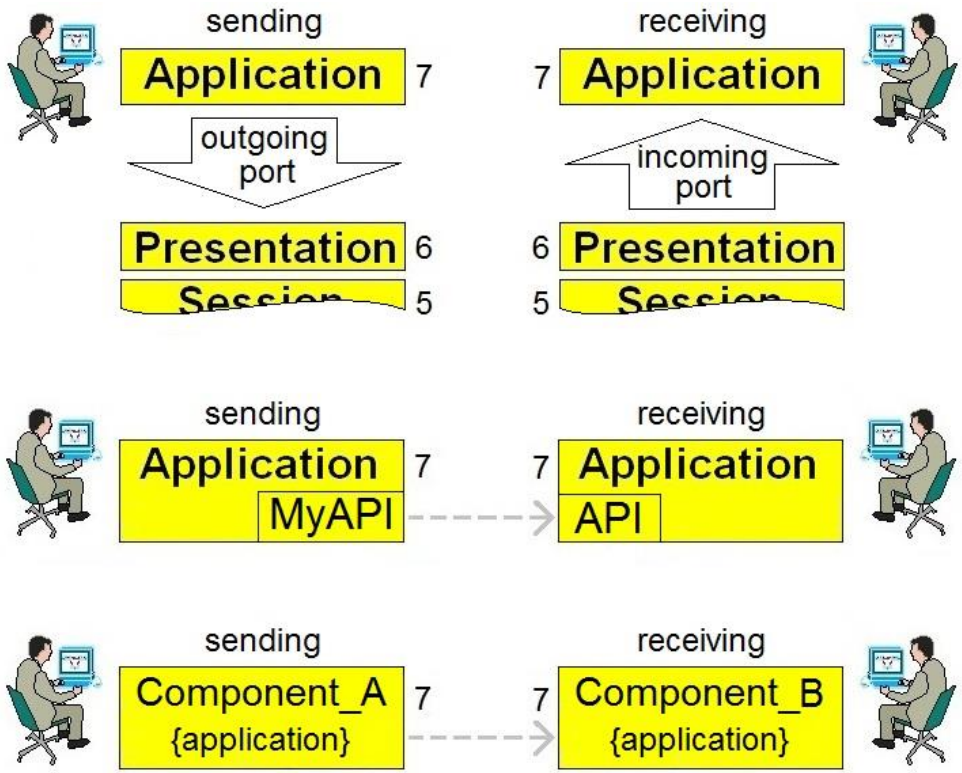
NANYANG TECHNOLOGICAL UNIVERSITY

... arch shows connection to network (cont)

The connection into its stack is effected by a software port [re-call UML Modeling Arch Pt1 les-son] at the application-presentation inter-layer boundary. There is one port for each stack in the sending-receiving pair

The connection into the receiving application is effected by an API[recall earlier section]

If the computer network is joining two software components belonging to the same application, then an API is incongruent

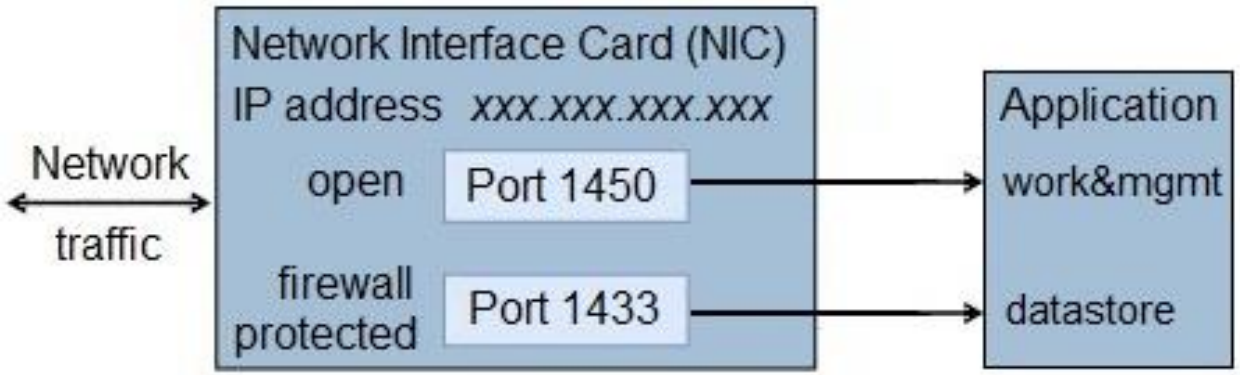


In all the above, it is crucial that the software architect not include the details of the computer network architecture into the design model for the new software. Instead, the architect need include in the new software architecture model only an annotation making reference to the network

Full design of network entry-exit

Stating the application-presentation inter-layer boundary interconnection is via a software port is a simplification. In actuality, the full mechanism is a network socket comprising its own API to the application (provided by the OS) and address on the network (port number and computer IP address). This is loaded into a network interface card (NIC) which effects the physical interface between the computer motherboard to the network cable, and holds the software for the whole network protocol stack, including the software port

[see below] This NIC shows two ports: 1450 is assigned to Tandem Distributed Workbench Facility, and 1433 (which is firewall protected) is assigned to MicrosoftSQL Server database management system



You expect me to know what?

aka., new knowledge in today's lesson

Software architecture in PC with OS: client-server

Standalone application's architecture: 3-tier, MVC, and 3-layer

Modeling networks in software architecture

Networked application architecture:
DB server, print server, and three terminals

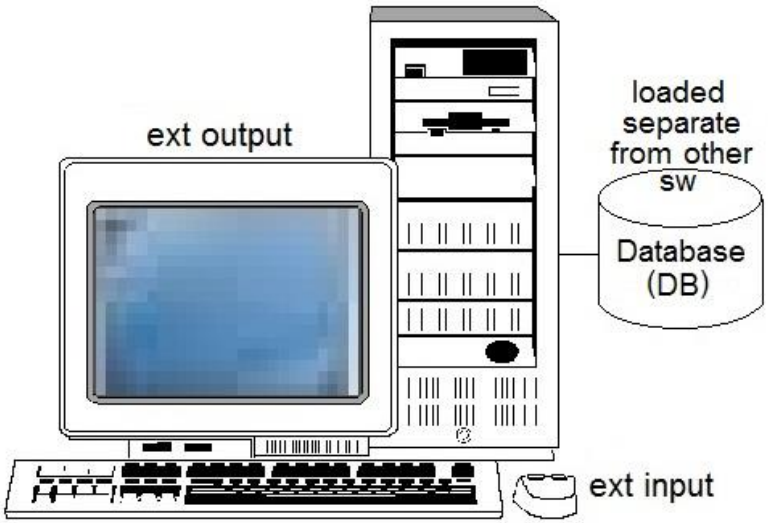
Complex networked architecture:
application hosting and application on file server

Architecture for application on world wide web

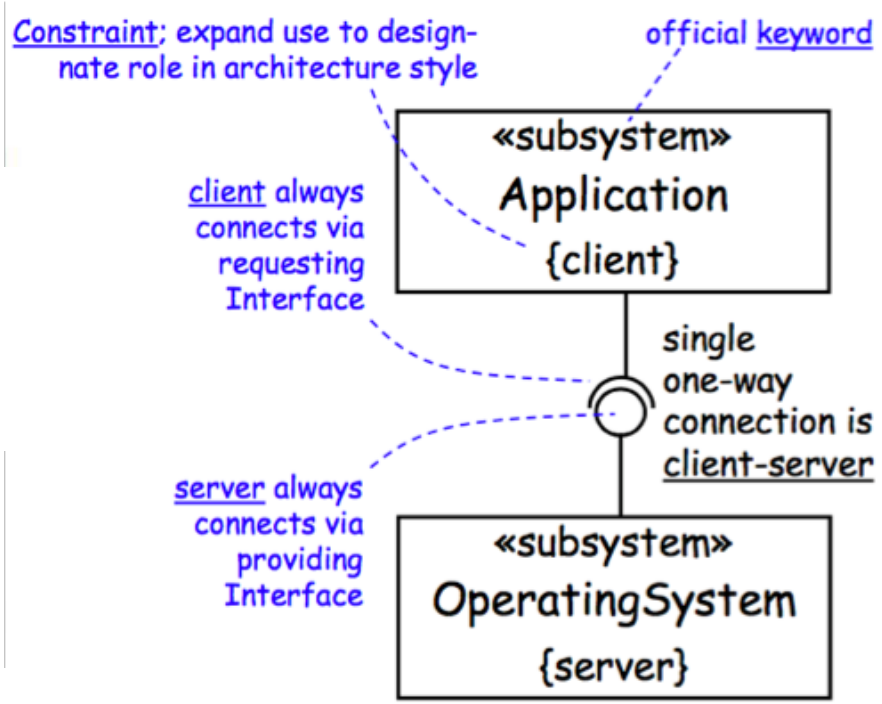
Architecture for reusable software: game application

Sw architecture in PC with OS: client-server

[see lower left] The Personal Computer(PC)-based workstation is the prevalent configuration of today's digital computing hardware. An application loaded there-on is auto-enrolled into a client-server architecture style: it being a client to the server OS that provides all possible required hardware and software services [see lower right]



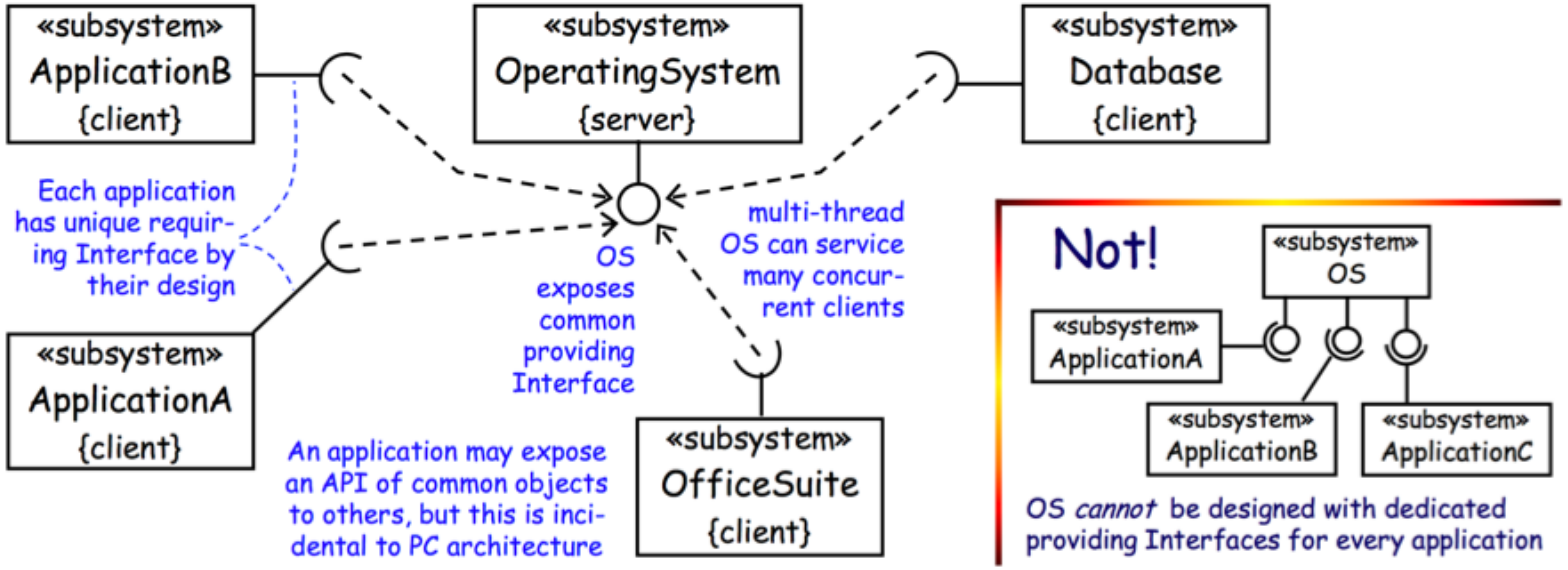
Personal Computer (PC)-based Workstation



Sw arch in PC with OS: client-server (cont)

It is not mandated that a server must service many clients. In fact, the majority of software servers have a dedicated client only, and even if there are several clients, each is serviced singly

However, modern OS are exceptionally powerful software with multi-thread programming able to support multiple client's multitasking. [see lower left] This is amore substantive model of the PC-based software client-server [not! is lower right]



Standalone application's architecture: 3-tier

An application's first-level decomposition typically exposes three components; corresponding architecture styles for this are 3-tier, MVC, and 3-layer

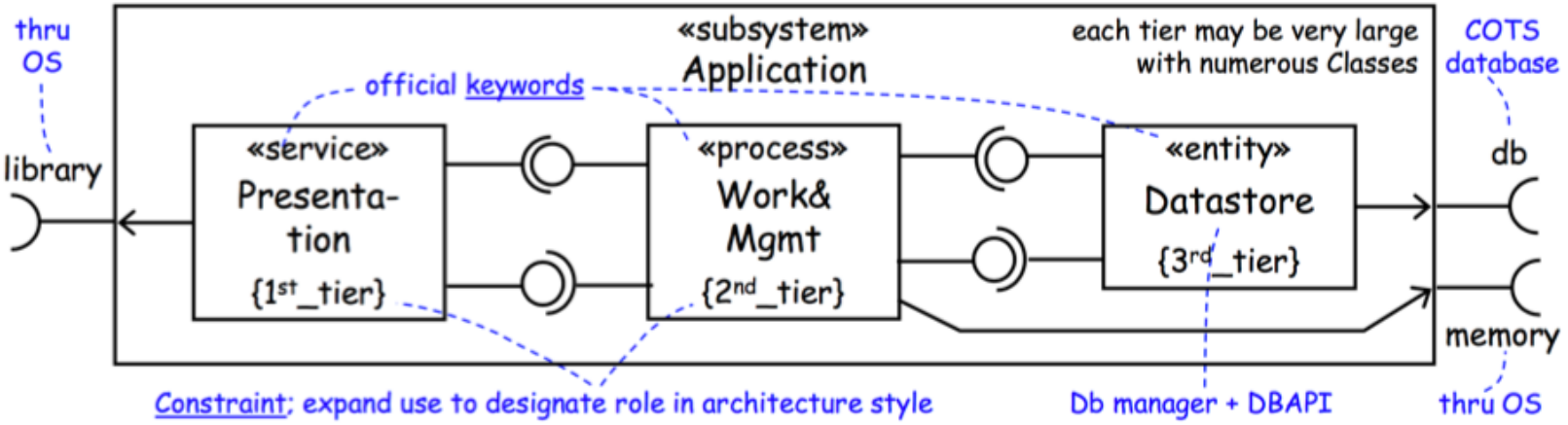
3-tier is the prevailing software architecture style in today's applications. It is distinguished only by organization of its functions; the number and arrangement of components and connections are not pertinent to its definition. The functions are presentation, datastore, and work&management (work&mgmt)

- Presentation – enables human users to command and control the application. It is norm-ally effected as a visualization (through a monitor), but may also be auditory (microphone and speakers) or kinaesthetic (joystick). It provides the human users with an immersive environment, and conducts exchange of the application's inputs and outputs
- Datastore – permanently secures current state and history info of the application. This ensures stable and coherent operation of the application over any length of time for any number of consequent jobs taken on by the user. The database (DB) need not be in the datastore; if the DB is external, datastore reduces to DB manager/DBAPI with entity definitions, data-handling protocols, and translation of application's language to SQL
- Work&mgmt. – executes autonomously or by human command all the functions of the application, excepting those belonging to fore-mentioned presentation and datastore, and including connections to APIs and function/process plugins

Standalone application's arch: 3-tier (cont)

Each subsystem requests distinct services from the OS, ie., visualization types to populate the screen for presentation tier, connectivity to external commercial-off-the-shelf (COTS) DB for datastore tier, and memory + utilities + devices for work&mgmt tier. Each subsystem passes control and exchanges data across Connectors operating concurrently in both directions [see below]

3-tier can be the new software's overall style if all functionality encased. All requiring Interfaces connect to the OS. The only design variance is the owner-ship of the DB, either by the OS (typical; direct resourcing) or datastore



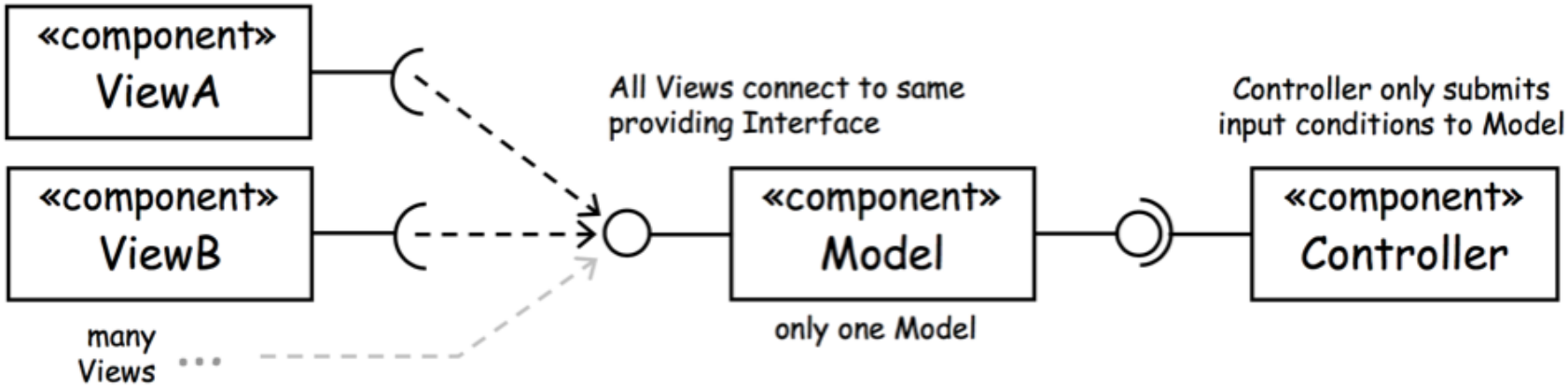
Standalone application's architecture: MVC

Another software architecture style based on three components and common in today's applications is model-view-controller (MVC) [see observer pattern]. It is distinguished by organization of functions and by the number and arrangement of its components and connections. The functions are model, view, and controller

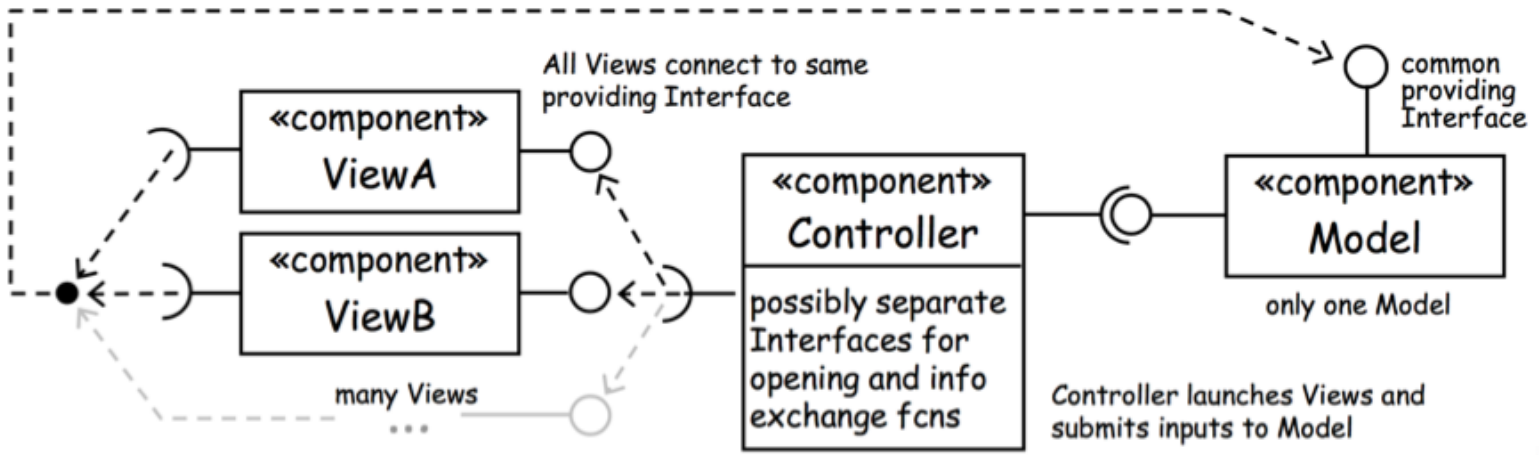
- Model – analyser-calculator. It takes inputs (current state of the application and environmental conditions), applies rules and formulations to analyse candidate responses, and selects/calculates the most authentic and appropriate. Model is not datastore, but it may derive its reference and precedence from information in a datastore. Each MVC instance may have one model only
- View – outputs information unique by its perspective, packaging, and function. Each MVC instance must have at least two views, and each must be absolutely consistent with all others by their its source and ontology. Every view should be independent from others
- Controller – controls operation of the MVC such as sending input parameters to model, launching views, and synchronizing views and model. The quantity of controllers is not definitive, nor are the connections; may be many sub-controllers, one for each view

MVC is not appropriate as the overall style for the whole of any new software that does more than just provide views. The entity DB is completely separate from the model, and the controller only has functionality for the model and views

Standalone application's arch: MVC (cont)

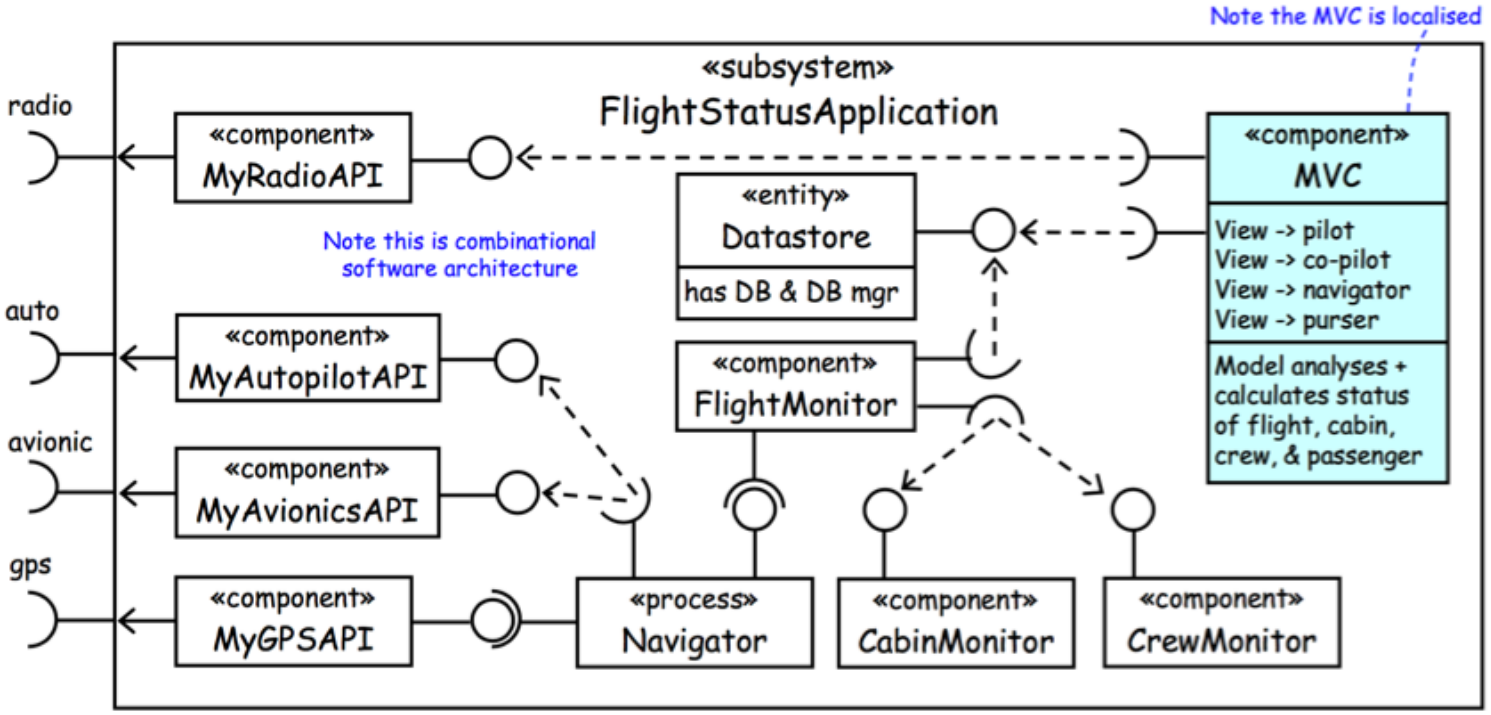


Shown are two common configurations of the controller's connections, either only to the model [see above], or to both the model and all views [see below]



Standalone application's arch: MVC (cont)

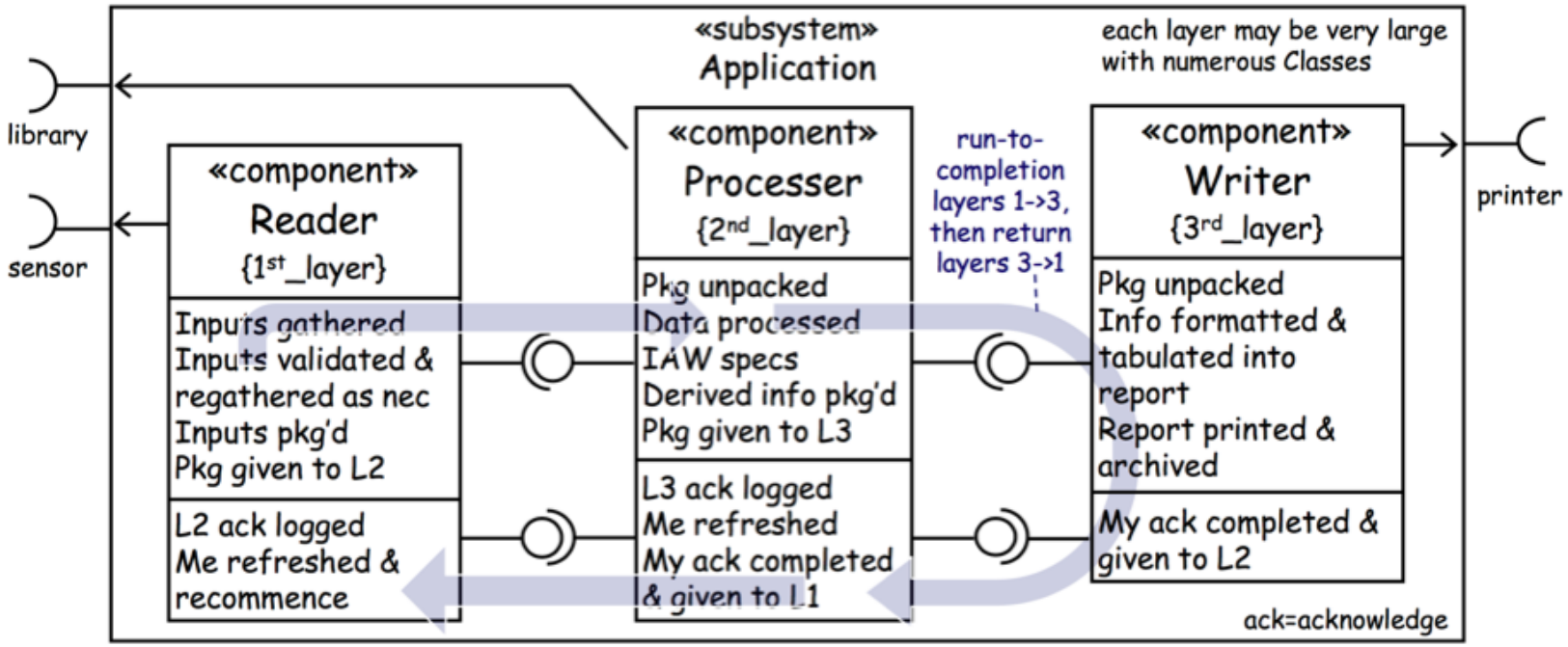
[see below] This is a typical scenario of the MVC not encasing all functionality, and therefore not being the overall style. The MVC is shown as a black box (its configuration is incidental to overall architecture) that controls all the graphic user interfaces (GUIs) for the human users. This is modeled from a fly-by-wire passenger jet with a four man aircrew + cabin crew all connected during flight



Standalone application's architecture: 3-layer

3-layer is uncommon architectural style for applications. It is defined by a strict regime of 'do-complete-pass2next'.

Unlike 3-tier and MVC, the function organization is not definitive; only the regime and its uni-directionality are definitive, eg.,[see below]. Layer can be the new software's overall style if all functionality is encased

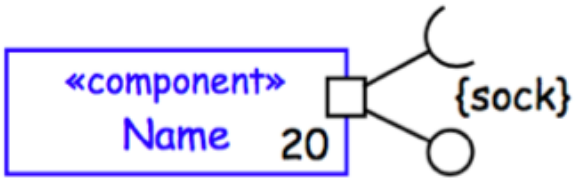


Modeling networks in software architecture

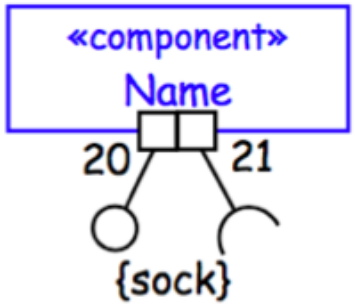
The only portion of the network that is modeled in the architecture diagram is the application's entry and exit points, ie., the Port and Interface [recall UML Modeling Pt1] into the socket API [recall earlier section]

Presume that all networks support two-way traffic, hence always design the Interface as requiring-providing pair. Use Constraint {sock} for network socket

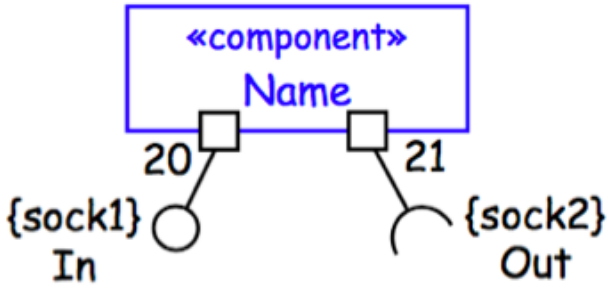
Three configurations for the Port and socket are: single Port-sock, pair Ports single-sock, and separate Ports-socks [see below]. Numbering ports is optional, however, it can be very useful to the Class designers and programmers



Config#1 – one Port on one sock handling both incoming and outgoing comms



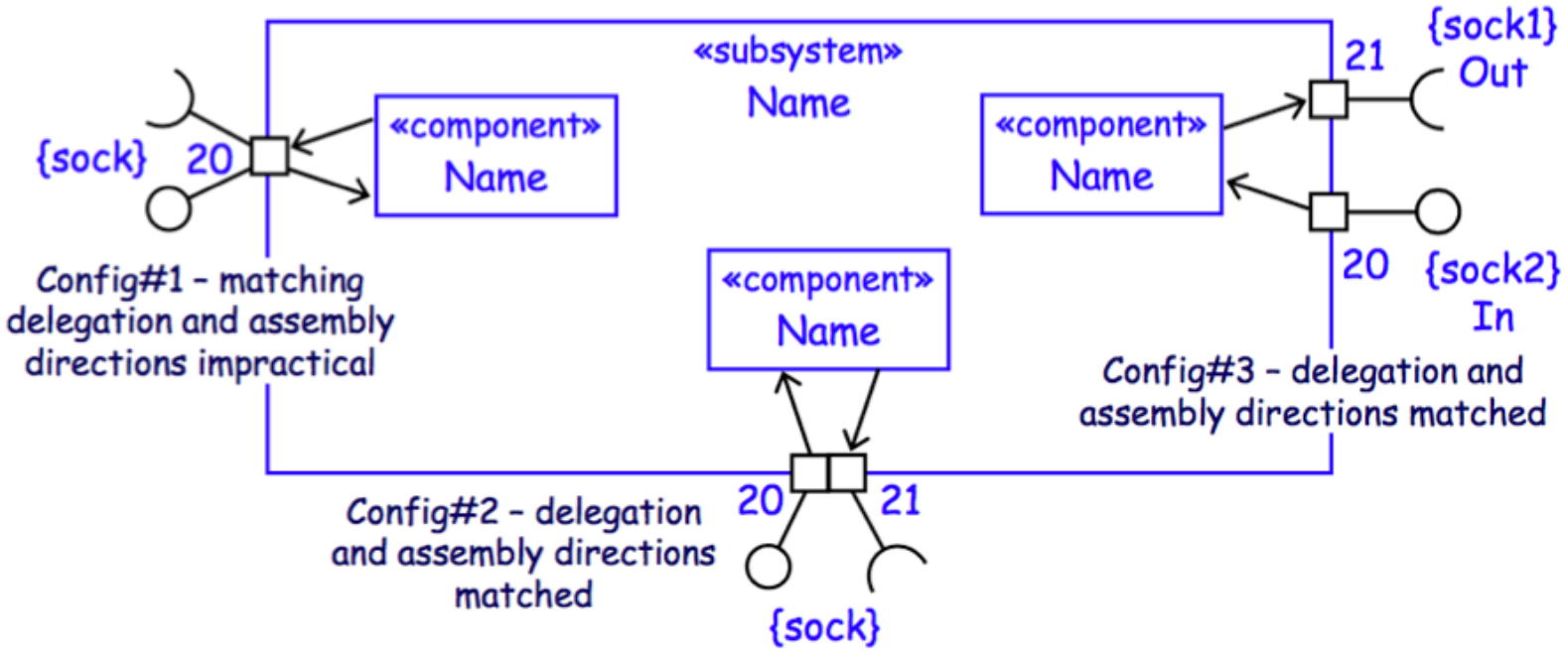
Config#2 – two Ports separating incoming and outgoing on one sock



Config#3 – one Port on one sock for incoming, and second Port on second sock for outgoing

Modeling networks in software arch (cont)

If the previous three Port-socket configurations are used in the decomposition form of the diagram, Delegation Connectors follow the established norm of their standard pairing with Interface/Assembly Connector [recall UML Modeling Pt1].[see below] Model of the three Port and socket configurations with Delegation connectors for decomposition diagram



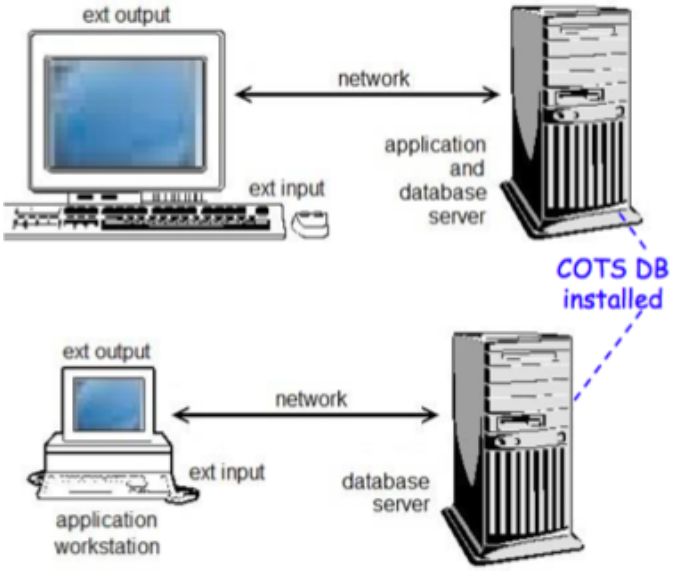
NANYANG TECHNOLOGICAL UNIVERSITY

Networked application architecture: DB server

Now let's explore the design considerations for applications intended to operate in various hardware devices connected to one another in a computer network. We will only consider 3-tier as the base style upon which to build our designs; there is far too much variation in the 3-layer to make it practical to study. Fin-ally, recall 'thin'/'fat' client concept in an earlier section

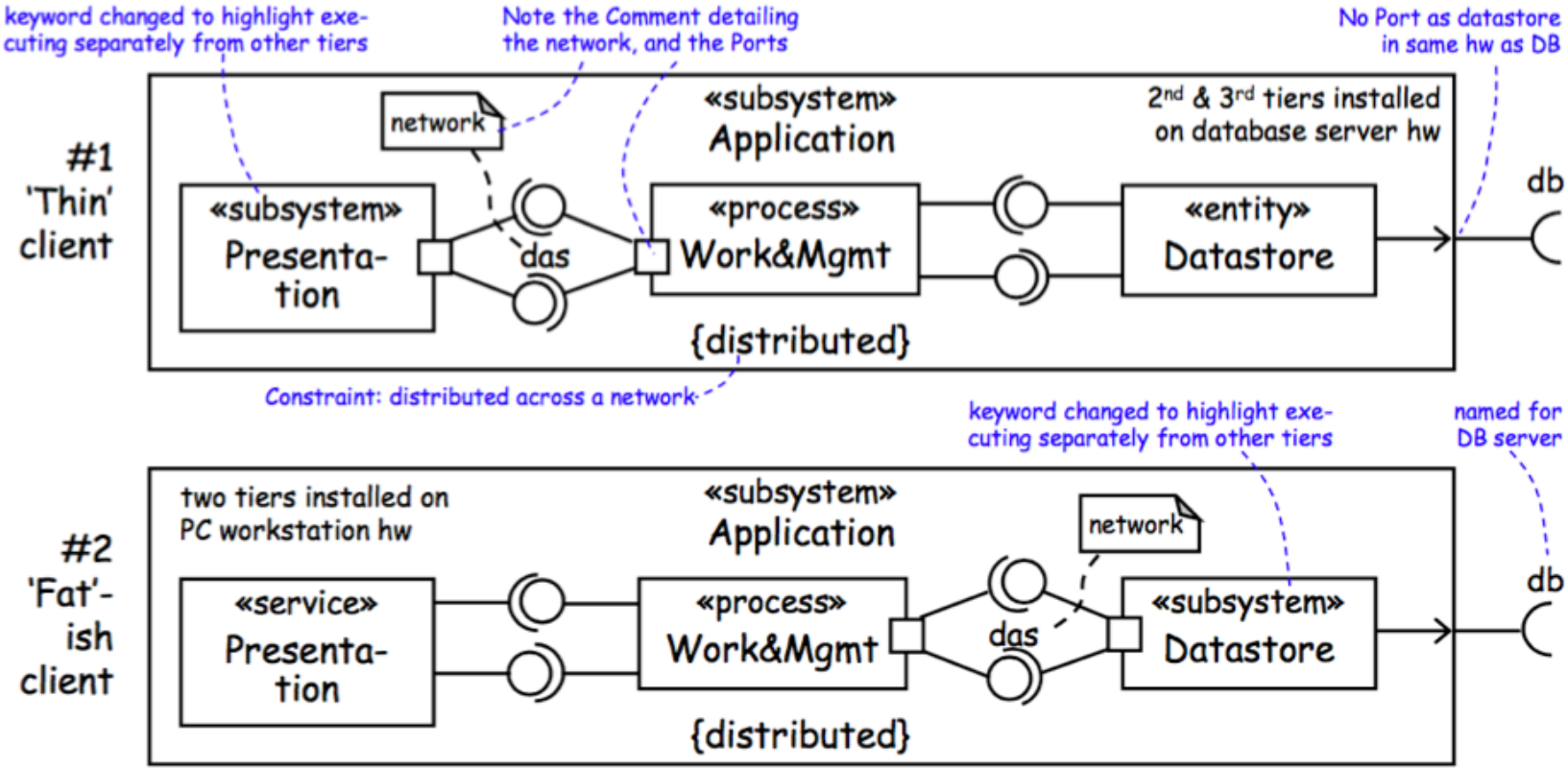
Our first case study is the networked DB server. Typically, for performance, maintenance, and expandability, a COTS DB may be deployed in its own hi-powered server, with the using application its client.
If the DB is solely for the application's use, ie., not shared, the datastore tier would be deployed with the DB. Then, for the remaining tiers there are two deployment variations

1. [see right upper] Work&mgmt is co-located with datastore in the DB server, while presentation is in a 'thin' client terminal
2. [see right lower] First two tiers are located in a 'fat'-ish client PC-based workstation



Networked application arch: DB server (cont)

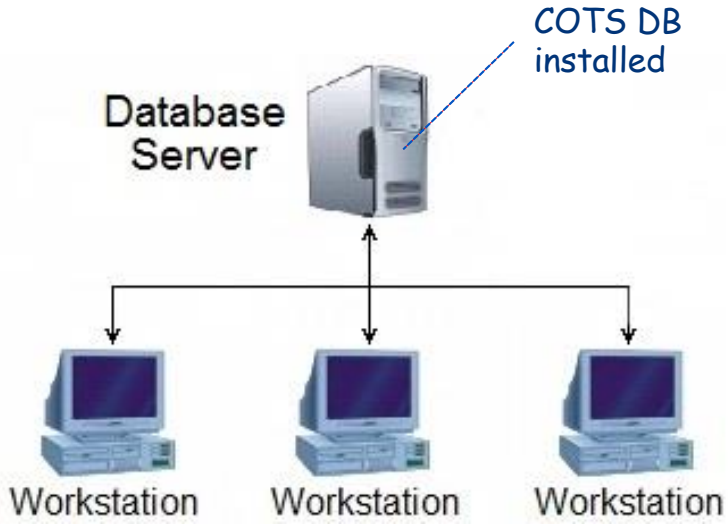
These models show the sw architecture for the 'thin' and 'fat'-ish client variations; the datastore is in the DB server regardless. Note two suppressions from previous 3-tier diagram: no connections to OS, and no Constraint identifying tier number

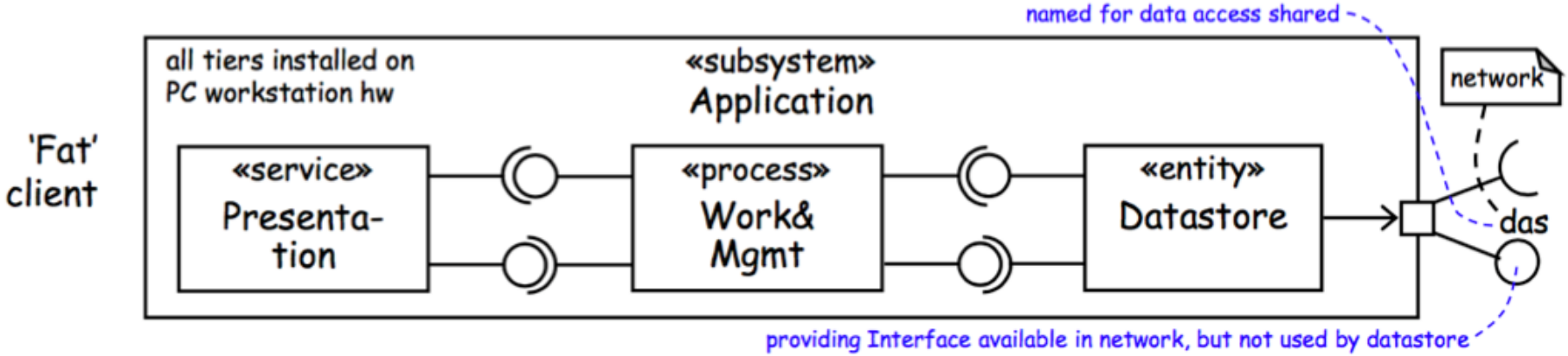


Networked application arch: DB server (cont)

[see right] DB server hardware may be net-worked to serve many applications as a shared resource. As with OS, multi-thread support depends on capacity of the COTS DB

Datastore tier is unique for every application, so in this case it is not co-located with the DB since it would encumber the performance of the DB server. Also, the 'thin' client con-figuration is incongruent; only the 'fat' client architecture [see below] is appropriate



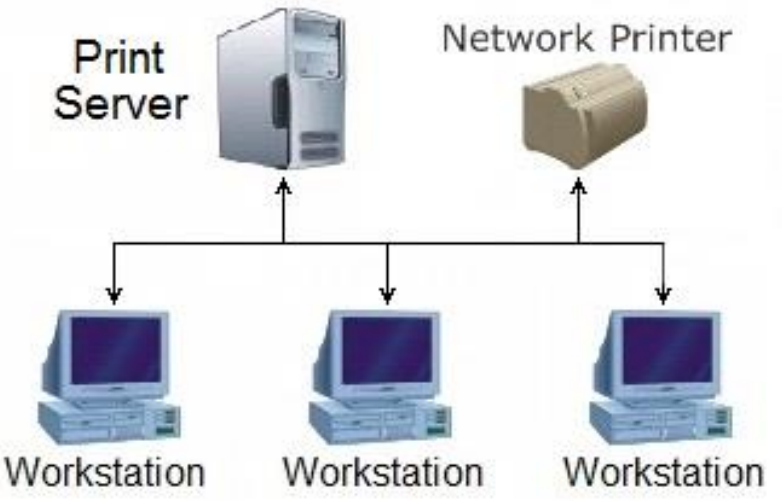


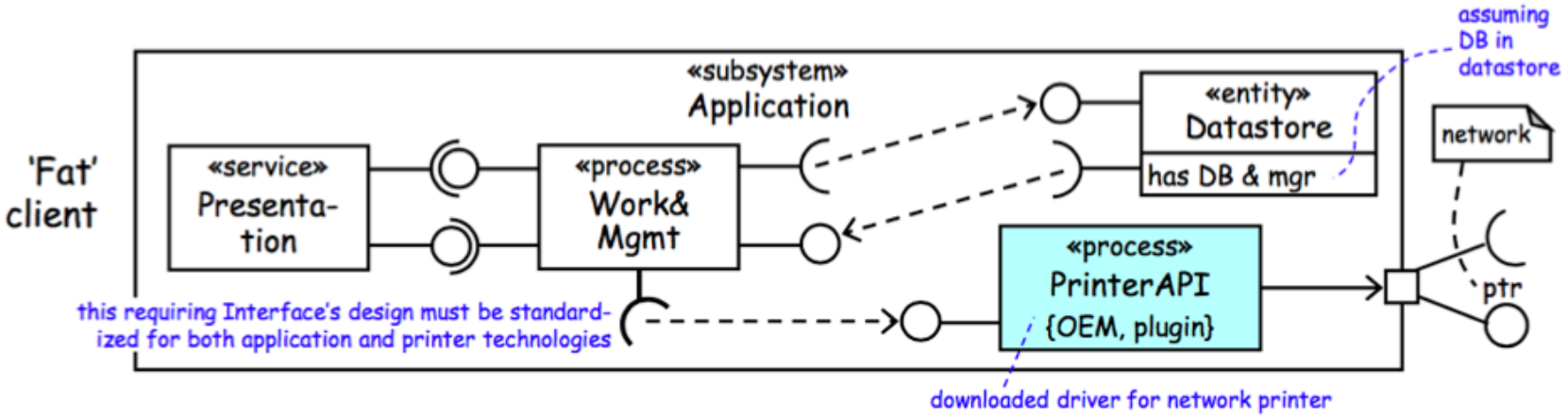
NANYANG TECHNOLOGICAL UNIVERSITY

Networked application arch: print server

[see right] A example of a commonly used network shared resource that requires an API to be plugged into the using application, is the print server. With printer and application technologies on different paths, and the printer being a service for the application, it is expected that new printers would supply an API. The interesting part is it plugging into the new sw

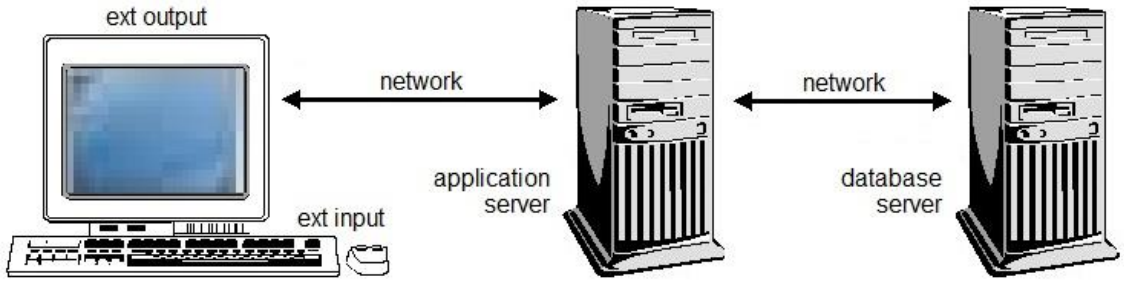
The sw architecture design shows the printer API plugin into the 'fat' client



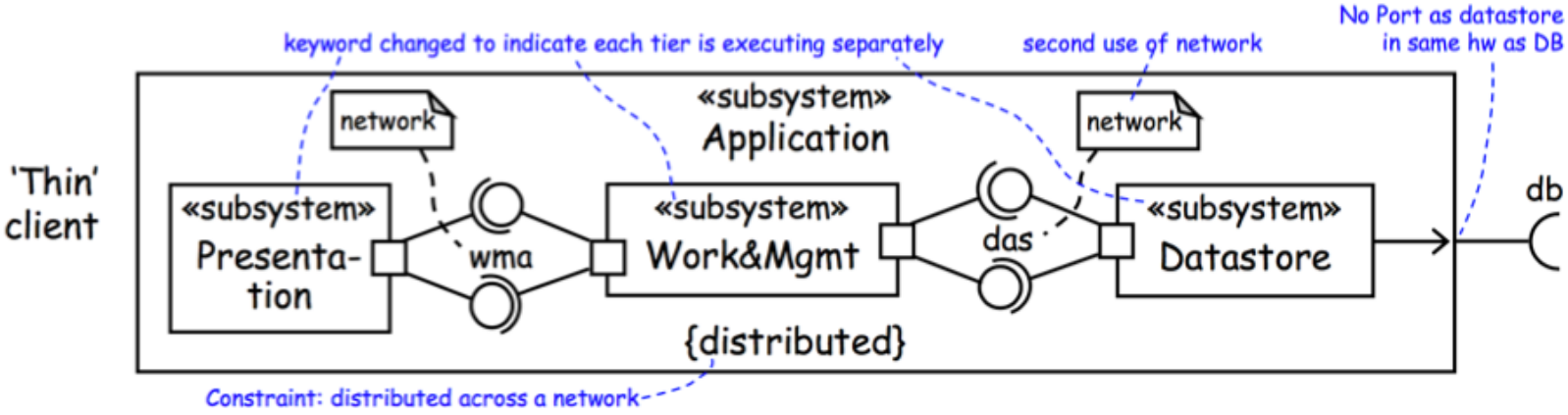


Networked application arch: three terminals

[see right] A variant of the earlier 'thin' client configuration — presentation in the display terminal, and work&mgmt + datastore co-located in a DB server dedicated to the single application — is to re-position work&mgmt out of the DB server and into its own, ie., application networked across three terminals

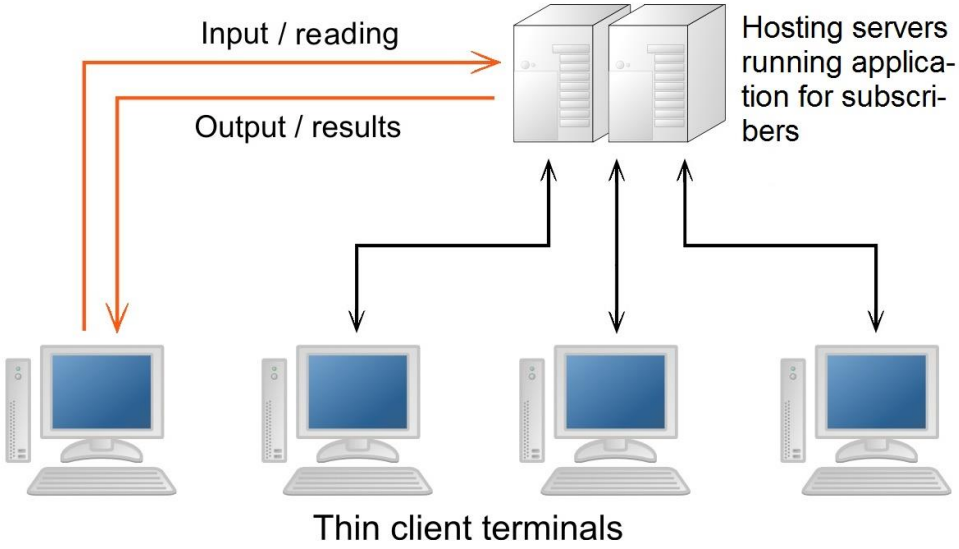


The consequent sw architecture design shows two changes: second connection into the network, and all tiers being executing programs (keyword="subsystem")



Complex networked arch: application hosting

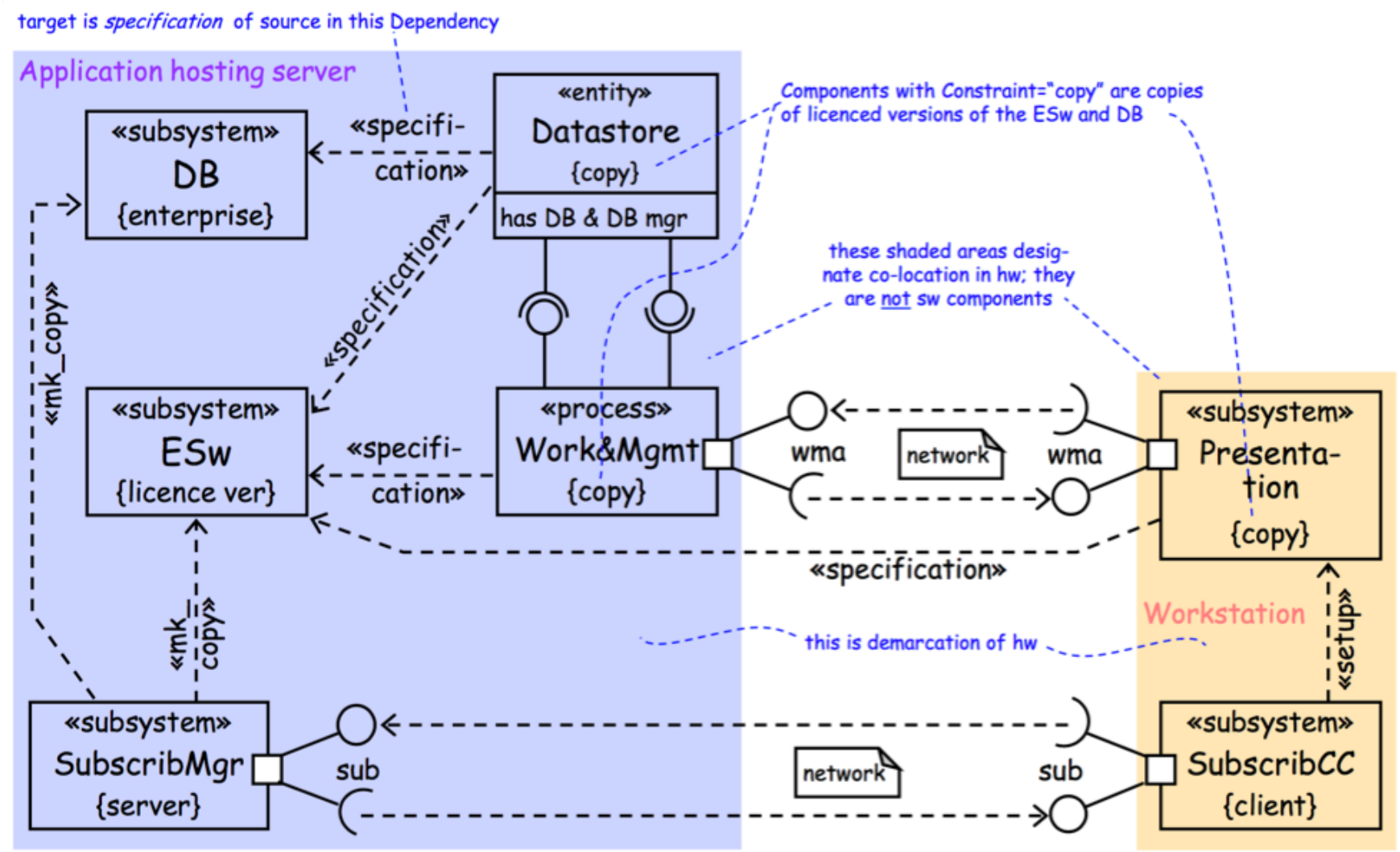
[see right] Increasingly popular hw configuration is application subscription hosting over a network. Application and COTS DB licensed copies are on the hosting server. Once a subscription is approved, the subscribe client is downloaded to the workstation. The client directs the subscribe manager to allocate an instance of the application and DB on the hosting server solely



for the subscriber, and sends the presentation subsystem 'thin' client to the work-station. Once the subscriber begins using the application and saving data, the hosting organization is responsible for all aspects of quality, eg., integrity, maintenance, and performance of the application and DB

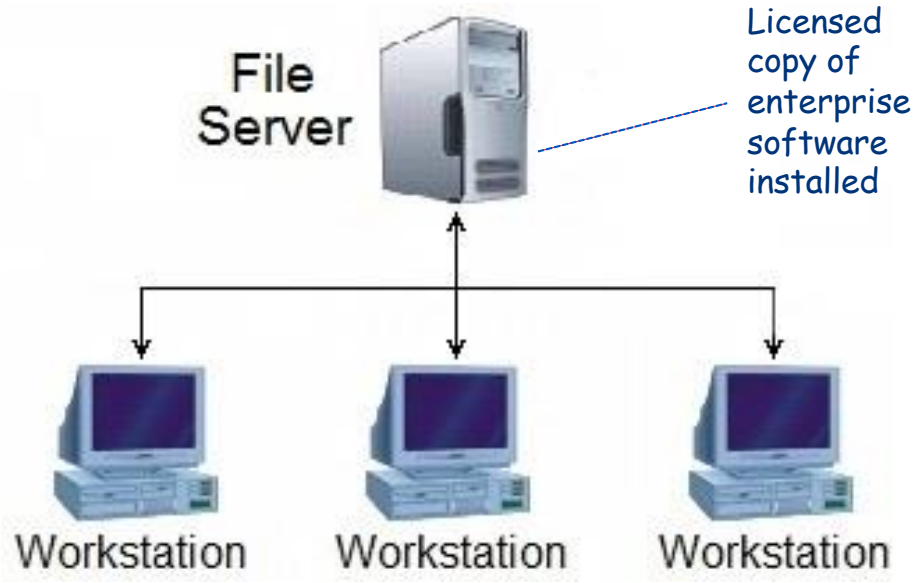
[see next page] Sw architecture shows the hosting server with the licensed vers and hosted copies of the application and DB, and the subscribe manager

Complex net arch: application hosting (cont)



Complex net arch: application on file server

[see right] In some organizations, the licensed copy of enterprise software (ESw) is held centrally on a networked file server. Users download an instance of ESw to their workstations where they do their work. Upon completion, the data is uploaded from each work-station and reconstituted in the DB server (also on the same network, but not shown here; see earlier diagram)



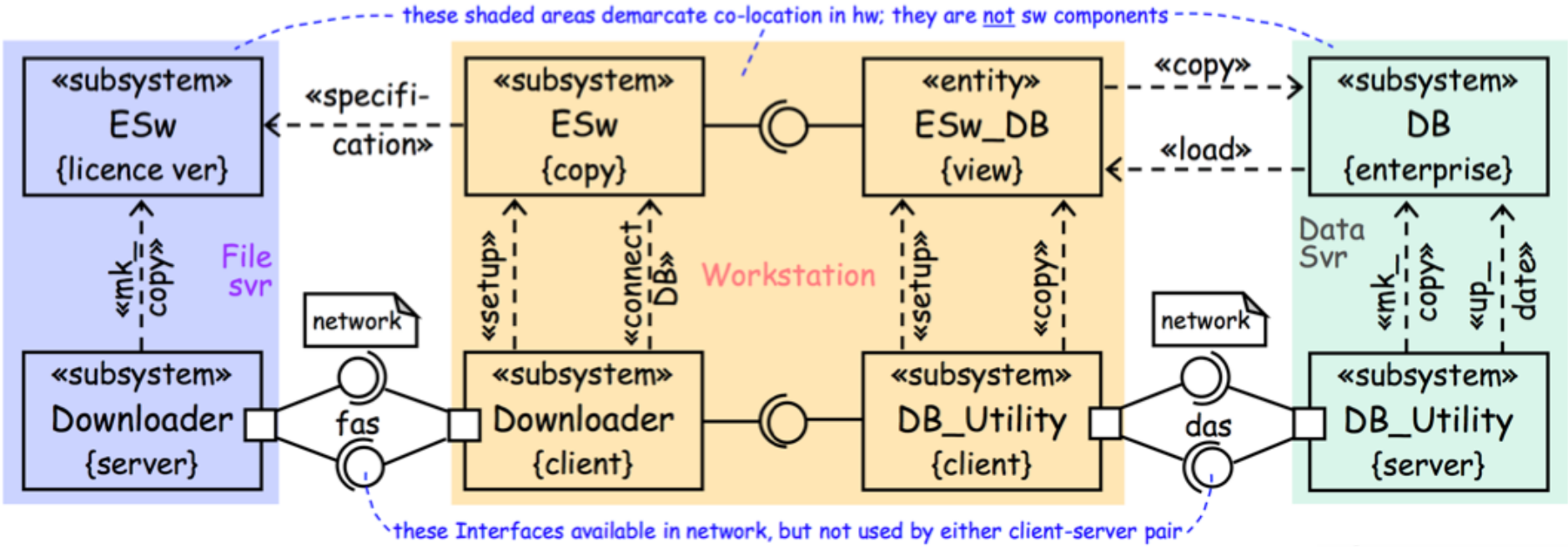
In additional to ESw (itself designed with the 'fat' client configuration), there are two other client-server utility applications (from the sw architect) both with their clients in the workstation

- Downloader utility – downloads and sets up the ESw in the workstation; its server is in the file server. The user, via the utility's client, requests service commencement. The utility server creates the ESw instance, then packages, and delivers it to the workstation. Upon receipt, the client unpacks it, and sets up the ESw

... net arch: application on file server (cont)

- DB utility – synchronizes the data; its server is in the DB server. Triggered by the Esw installation in the workstation, the utility client calls for a data download. The utility server creates a view of the latest data on the DB server, and sends it to the client which connects it to the ESw. After the session is closed, the utility client copies the ESw data and sends it to the client for upload as latest version

Sw architecture shows the workstation with the ESw copy, DB view, and the two utility's clients, plus the file and DB servers with the utility's servers [see below]



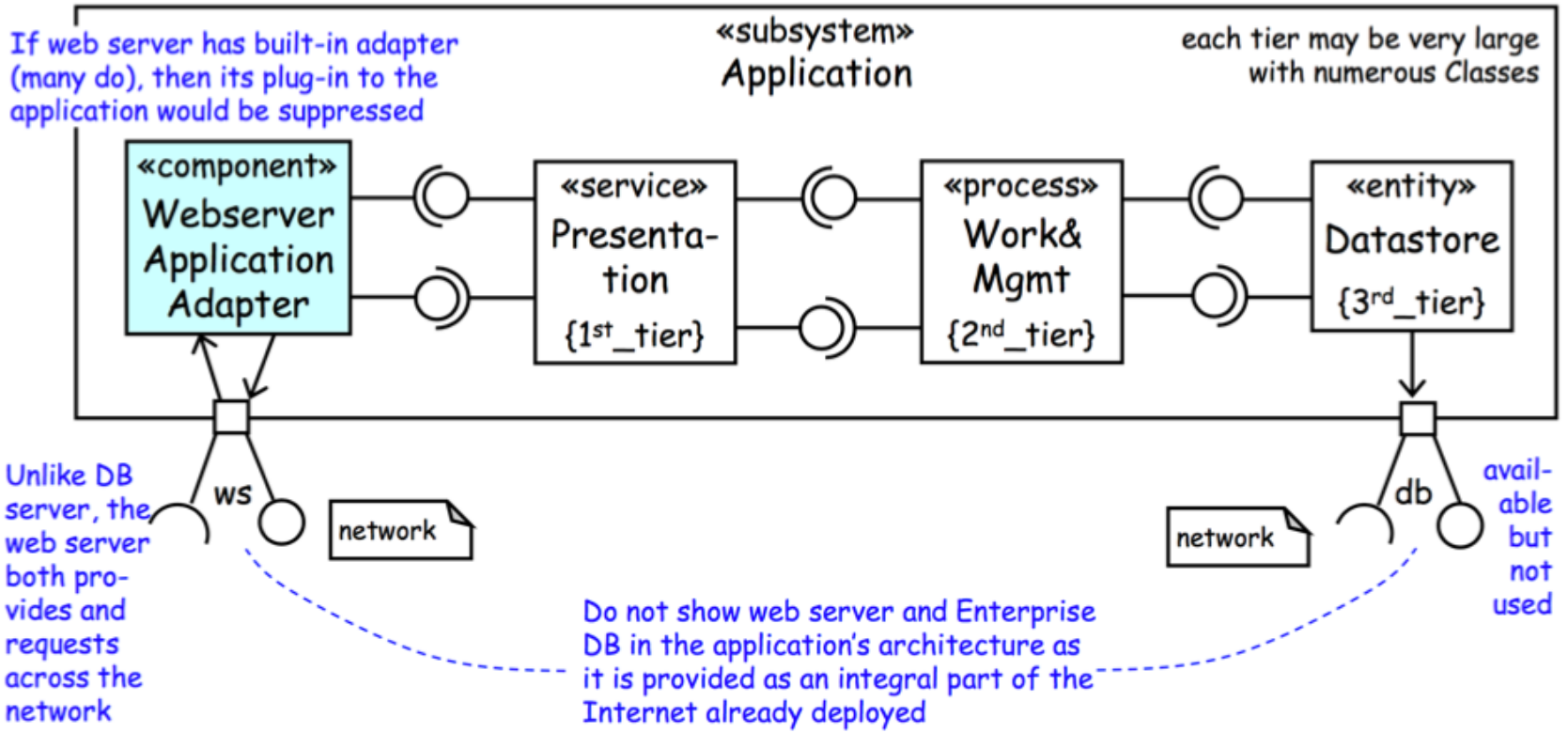
Arch for application on world wide web

[see below] Sw architecture for world wide web (WWW) info exchange via the Internet is 3-tier, with the web browser being the presentation tier, web server being the work&mgmt tier, and content servers being the datastore tier. The browser requests in a hypertext transfer protocol (HTTP) compatible manner, access to the content offered at a uniform resource locator (URL) address. The web server requests the applicable content from the content servers. After receiving it, the web server incorporates the content into web pages (in hypertext markup language(HTML) format), and then distributes the pages to the requesting browser

Web pages are a re-packaging of content and not the content or application itself. So, they being copied into the workstation is not a download of a 'thin' client

Arch for application on WWW (cont)

[see below] Sw architecture shows the content servers with the application and DB. Importantly, there is a new component to adapt the application's language (C++,Java, etc) to the language used in the web server (HTML)

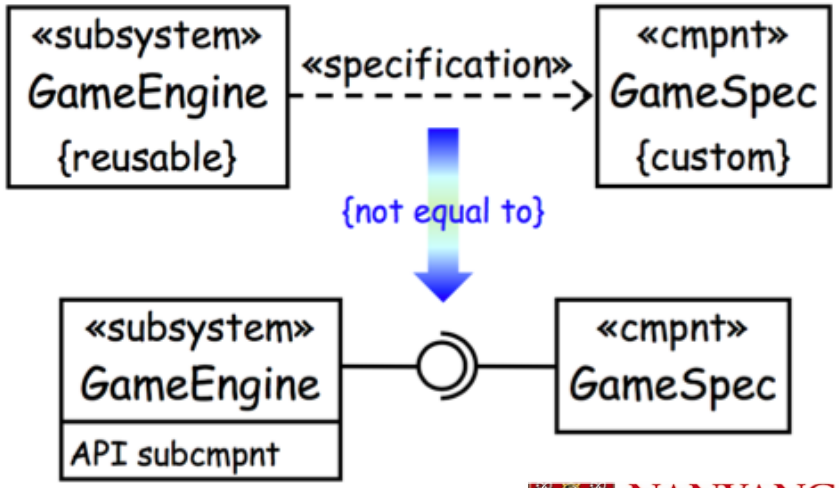


NANYANG TECHNOLOGICAL UNIVERSITY

Arch for reusable software: game application

Increasingly, developers are leveraging reusable software to increase production of applications while reducing risk. One such is the game engine, eg., Unity [see right], effecting data-driven development of games, ie., the engine renders (says "how") while the developer specifies (says "what, where, when, and why")

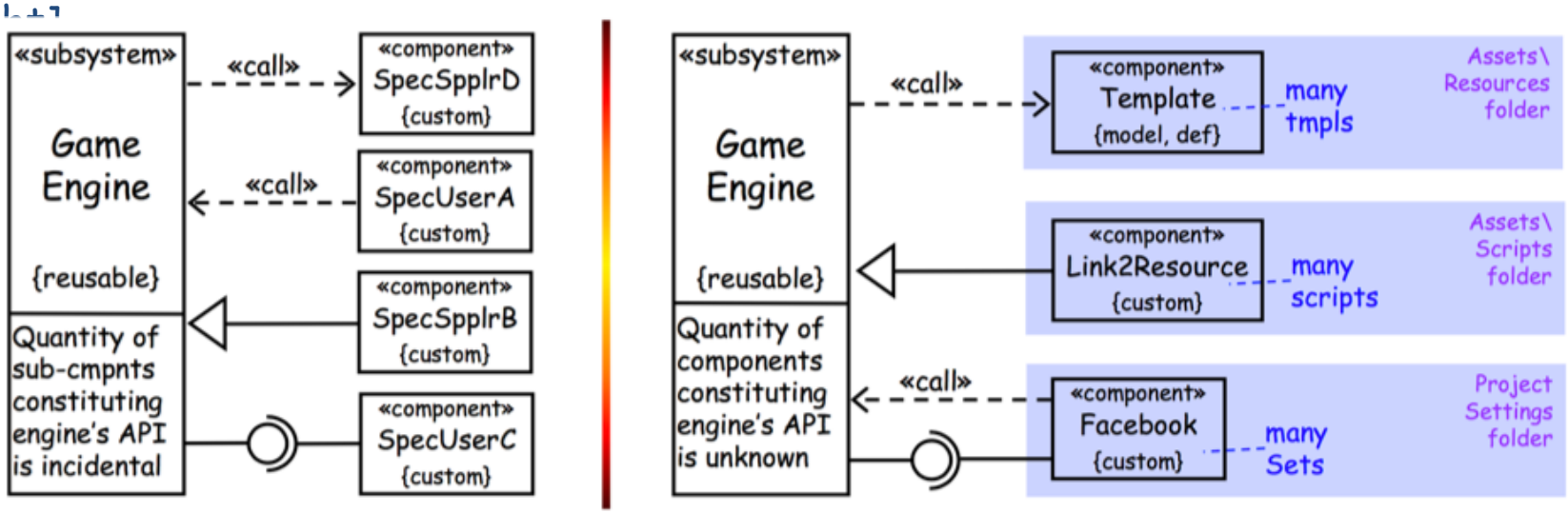
This specification data is much more complex in content and nature than data as typical parameters. Commensurately, the API through which the engine accepts the specification data is also more complicated than the single interface we have used in previous architectures [see right]



Arch for reusable sw: game application (cont)

The engine's API may use all the approaches to invoke external behaviour [recall an earlier section], ie., direct calling (game is supplying and using), extension (game is using), and interface (game is supplying). Clearly, it is not in a client-server relationship with the specifying components [see below left]

Instead, all custom components are placed in directories where the engine expects to find them, eg., Unity expects to find all game resources in "Assets", and all play-ability settings in "Project Settings". This is not an architecture design, pushing the UML Component diagram to its limit of modeling effectiveness [see below right]

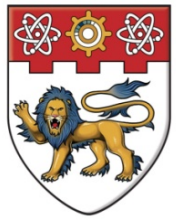


What did I just do? aka., learning wrap-up

- Software architecture in PC with OS is client-server
- Architecture for standalone application is 3-tier, MVC, and 3-layer
- Modeling networks in software architecture
- Architecture for networked application is ...

  DB server, print server, and three terminals
- Complex networked architecture for ...

  application hosting, and application on file server
- Architecture for application on world wide web
- Architecture for reusable software: game application



NANYANG TECHNOLOGICAL UNIVERSITY

Software Architecture Basic Designs & UML

presented by

Shar Lwin Khin
Research Scientist
SCSE
lkshar@ntu.edu.sg
N4-02c-76

Courtesy of Kevin Anthony Jones' slides