Architectural styles:

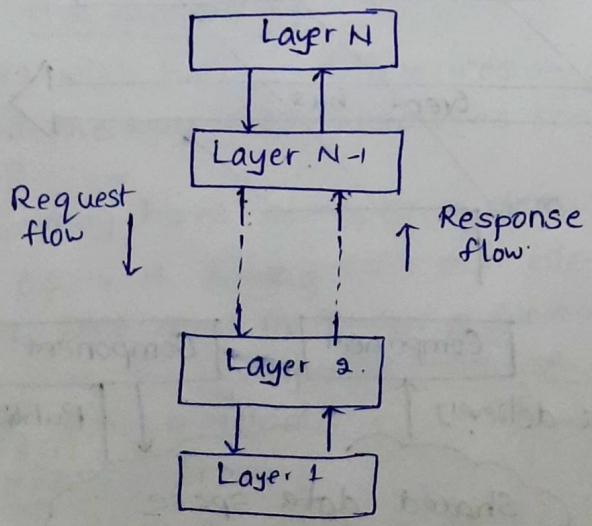
The notion of an architectural style is important. Such a style is formulated in terms of components, the way that components are connected to each other, the data exchanged between components and finally how these elements are jointly configured into a system.

Important architectural styles are:

1) Layered architectures
2) Object-based architectures
3) Data-centered architectures.
4) Event-based architectures.

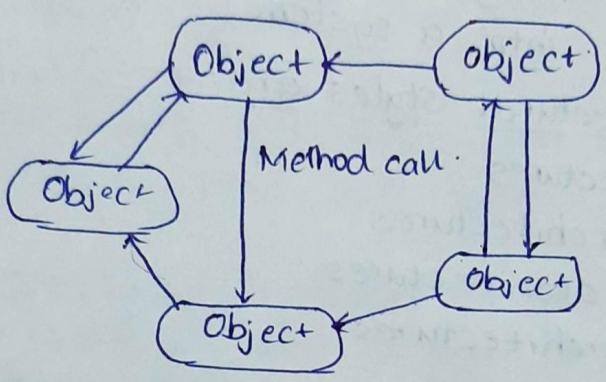
1) Layered architectures:

- In this components are organized in a layered fashion where a component at Layer L_i is allowed to call components at the underlying layer L_{i-1}, but not the other way around.
- Control generally flows from layer to layer.
  - requests go down the hierarchy.
  - Results flow upward.



Object-based architectures

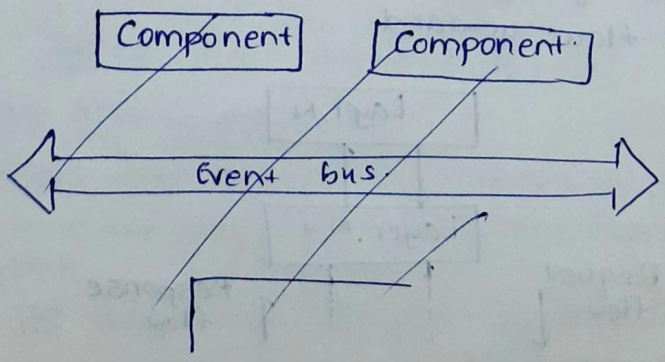
- Each object corresponds to as a component and these components are connected through a (remote) procedure call mechanism.
- This software architecture matches the client-server system architecture.

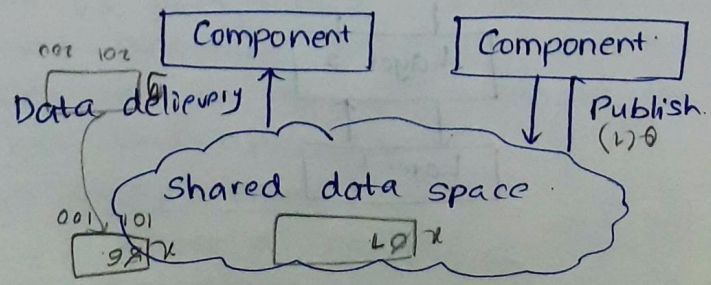


3) Data-centered architectures:

- These evolve around the idea that processes communicate through a common repository
- These are as important as above architectures.

Eg: A wealth of networed ~~architectures~~ applications developed that rely on shared distributed file system. communicated through files.

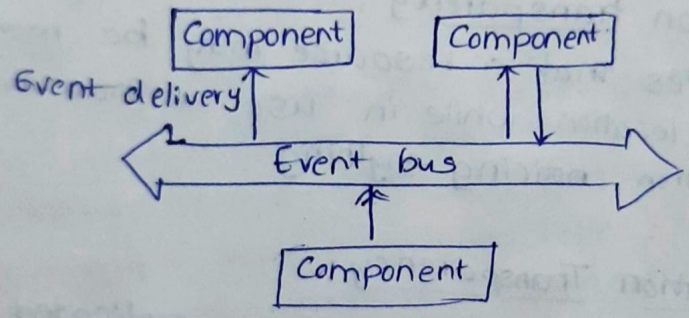




4) Event-based architectures :

- In this processes essentially communicate through the propogation of events, which optionally also carry data.
- For DS, event propogation is associated with publish/subscribe systems.
- Basic idea is that processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.

Advantage : Processes are loosely coupled i.e need not explicitly refer to each other. Also known as decoupled in space or referentially decoupled.



Different types of transparency :

1) Access transparency :
- It deals with hiding differences in data representation and the way that resources can be accessed by users.

Eg : A DS may have computer system that run different OS, each having their own file-naming conventions. Differences in naming conventions, as well as how files can be manipulated, should all be hidden from users and applications.

2) Location transparency :
- It hides where a resource is located.
- Location transparency can be achieved by

assigning only logical names to resources, that is, names in which the location of a resource is not secretly encoded.

Eg: URL http://www.prenhall.com/finder.html which gives no clue about the location of Prentic Hall's main web Server.

3) Migration transparency:

- It hides that a resource may move to another location without affecting how those resources can be accessed.

Eg: Mobile users can continue to use their wireless laptops while moving from place to place without ever being disconnected.

4) Relocation transparency:

- It hides that a resource may be moved to another location while in use without the user or application noticing anything.

5) Replication Transparency:

- It hide that a resource is replicated to increase availability or to improve performance.
- To hide replication from users, it is necessary that all replicas have the same name.
- System that supports replication transparency should also support location transparency to refer to replicas at different locations.

6) Concurrency transparency:

- It hide that a resource may be shared by several competitive users.

Eg: Two independent users may each have stored their files on the same file server or may be accessing the same ~~dat~~ tables in a shared database. In such case, it is important that each user.

high level data flow system that renders

and similar ... for Analytical skills

does not notice that the other is making use of the same resource.

- Issue is Consistency (locking mech, transactions)

6) Failure transparency:

- It hide the failure and recovery of a resource
- Masking failures is one of the hardest issues
- Difficulty in masking failures lies in the inability to distinguish b/w dead resource and a painfully slow resource

Eg: when contacting a busy dead ~~resource~~ ~~and~~ web server, a browser will eventually time out and report that the web page is ~~web~~ unavailable. ~~At~~ that point user cannot conclude that the server is really down.

Examples

1) Access: File System in NFS, SQL queries and Navigation of the web
2) Location: File system in NFS, pages of the web.
3) Concurrency: NFS, Automatic Teller machine network.
4) Replication: Distributed DBMS and Mirroring of web pages
5) Failure: Database Management Systems
6) Migration: NFS, web pages.

Goals of Distributed system:

Four goals that should be met to make building a DS worth the effort

1) should make resources easily accessible
2) should reasonably hide the fact that resources are distributed across a network
3) should be open.
4) should be scalable.

1) Making Resources Available Accessible:

- Main goal of a DS:
  - make it easy for the users to access remote resources
  - to share them in a controlled and efficient way
- Resources can be anything i.e printers, computers etc

Accessibility issues:

- security
- unwanted communication.

2) Distribution Transparency:

- Goal: To Hide the fact that its processes and resources are physically distributed across multiple computers.

Write different forms of transparency here. (table)

Degree of transparency: The transparency should be upto a limit. There are situations in which attempting to completely hide all distribution aspects from users is not a good idea.

Issues:

- Timing
- Synchronization
- Performance.
- Consistency
- Context Awareness.
- Limits of Possibility.

3) Openness : (Open DS)

Goal : To offer services according to standard rules that describe the syntax and semantics of those services.

Eg :

DS : services are specified through interfaces, which are often described in an Interface Definition Language (IDL)

- (Interface definitions allow an arbitrary process that needs a certain interface to talk to another ~~inter~~ process that provides that interface. It allows 2 independent parties to build completely diff. implementations of those interfaces, leading to 2 separate DS that operate exactly in the same way
- Proper specifications are complete and neutral.
- Completeness and neutrality are important for interoperability and portability) Your wish
- Another goal is that it should be easy to configure the system out of different components and should be easy to add new components or replace existing ones without affecting those components that stay in place.

4) Scalability :

Scalability of a system is measured with respect to

Size : can easily add more users and resources to the system.

Geographic extent : A geographically scable system is one in which the users and resources may lie far apart.

Administrative scalability: can be easy to manage even if it spans many independent administrative organizations.

Openess and scalability (Sir blog).

Pitfalls:

False assumptions that everyone makes when developing a ~~ds~~ distributed application for the first time

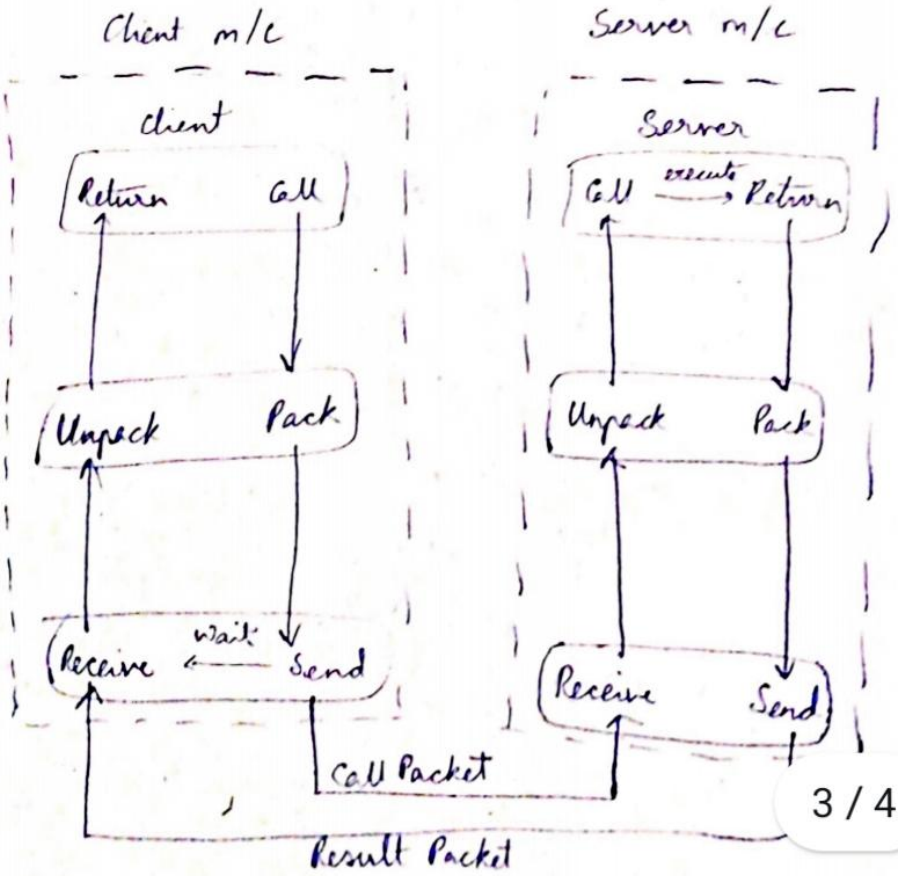
1) The network is reliable.
2) The network is secure.
3) The network is homogeneous.
4) The topology does not change.
5) Latency is zero.
6) Bandwidth is infinite.
7) Transport cost is zero.
8) There is one administrator.

RPC	LRPC
- Conventional RPC involves multiple threads that must send signals & switch context.	- In LRPC, the kernel changes the addr space for the caller's thread & it continues to run in server domain.
- RPC has general stubs, but many of its features are infrequently needed	- LRPC uses a stub generator that produces simple stubs in Modela 2+ assembly language.
- Comm. btw 2 or mor machines.	- Comm. within machine.
- High cost	- ~~low~~ Cost less then RPC
- High overhead	- Less overhead then RPC.

Remote Procedure Call (RPC)

RPC is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same addr space as the calling procedure. The 2 processes may be on same system (or) on diff systems with a n/w connecting them.

Working of RPC



Implementation of RPC.

3 / 4

Scanned by CamScanner

Steps during RPC

1.) A client invokes a client stub procedure, passing parameters in usual way. The client stub resides within client's own addr space

2) The client stub marshalls (pack) the parameters into a msg.

3.) The client stub passes the msg to transport layer, which sends it to remote server m/c

4.) On the server, the transport layer passes the msg to a server stub, which demarshalls (unpack) the parameters and calls the desired server routine using regular procedure call mechanism.

5.) When server procedure completes, it returns to the server stub, which marshalls the return values into a msg. The server stub then hands msg to transport layer.

6.) The transport layers sends result msg back to client transport layer, which sends back to client stub.

7.) The client stub demarshalls the return parameters & execution returns to caller.

Adv

1.) RPC support process oriented & thread oriented models.

2.) The internal msg ~~processing~~ passing mechanism is hidden from user.

3) The effort to re-write & re-develop the code is minimum

4.) Can be used in distributed & local environment

5.) Many protocol layers are omitted by RPC to improve performance.

Disadv

1.) It can be implemented in diff ways and not standard.

2.) There is no flexibility in RPC for H/W architecture & only interaction based

3.) Increase in costs.

Scanned by C

Light weight RPC (LRPC)

LRPC is a comm facility that is designed & optimized for communication btw protected domains in the same m/c.

- LRPC simplifies aspects of RPC as control transfer, data transfer, linkage and stubs.
- The LRPC is a facility designed & optimized for cross-domain (comm btw domains on same m/c) comm. where user level server processes have its own address space.
- LRPC is safe & transparent which represents a viable comm alternative for microkernel OS.
- It is used in small-kernel OS to avoid cost incurred by using RPC.

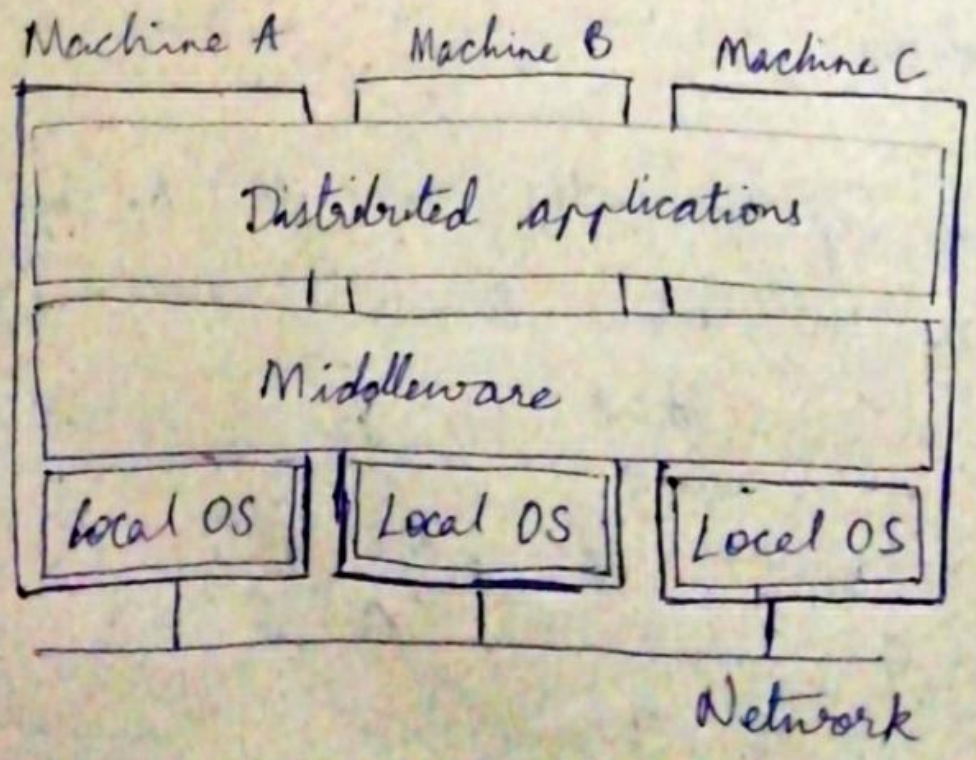
Features

has high overhead. A lighter weight solution is necessary.

- **Lightweight RPC features**:
  - **Simple control transfer:** Conventional RPC involves multiple threads that must send signals and switch context. In LRPC, the kernel changes the address space for the caller's thread and lets it continue to run in the server domain (at the same priority?).
  - **Simple data transfer:**Shared buffers are pre-allocated (when the caller imports the LRPC module) for the communication of arguments and results. The caller copies the data onto the A-stack, after which no other copies are required.
  - **Simple stubs:** RPC has general stubs, but many of its features are infrequently needed. LRPC uses a stub generator that produces simple stubs in Modula2+ assembly language.
  - **Design for concurrency:**Idle processors in multi-processor machines cache domain contexts to reduce context-switch overhead. Counters are used to keep the highest activity LRPC domains active on idle processors.
- **Performance**:
  - 3X speedup for single processor.
  - Additional speedups for multiprocessor machines.

Role of middleware in DS

Middleware is a software which is designed to act as communication bridge between the distributed applications and the platforms. As the diagram below has depicted middleware sits btw distributed applications and OS which is the platform of different users. The distributed application is the only component with which the end users have interaction.



> The role of middleware is to confine distributed complexion of a system from the users. In other words the middleware is able to keep the system active in multiple (distributed) locations but doesn't often reveal details that is part of a complex system. These details would be programming codes.

> The middleware has a role to handle heterogeneous environments of a DS. The term heterogeneity refers to how users can access and use distributed applications through a range of devices and networks. This heterogeneity applies to network, h/w, s/w or programming languages.

> Provide uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported and made to interoperate.

> Supply a set of common services to perform various general purpose functions to avoid duplicating efforts, and to facilitate collaboration btw applications.

Scanned by CamScanner