PROCESS MODEL

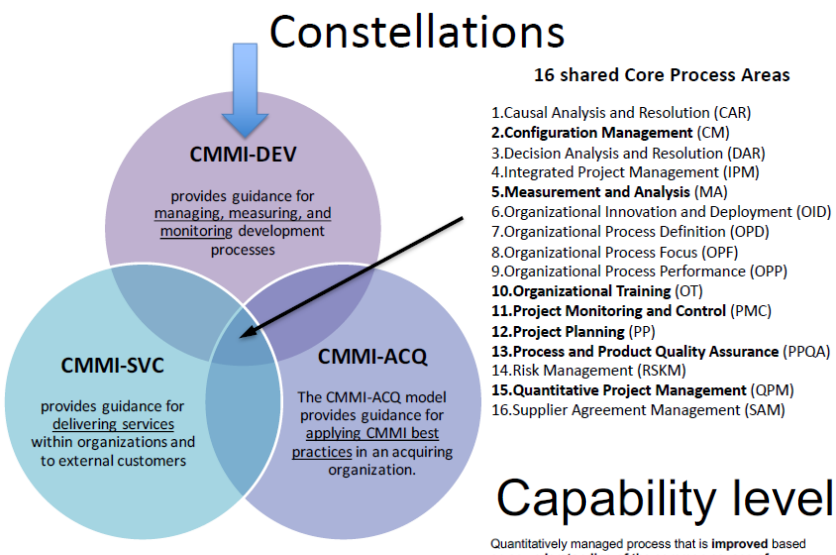
A process model is a structured collection of practices that describe the characteristics of effective processes. It includes practices proven by experience to be effective.

A process model is used:

1. To set process measurable objectives and priorities
2. To ensure stable, capable and mature processes
3. As a guide for improvement of projects and organizational processes
4. To diagnose/certify the state of an organization’s current practices

CMMI (Capability Maturity Model Integration)

The Software Engineering Institute (SEI) developed CMM during late 1980s, the customer was DoD (American Department of Defense). CMMI integrates some of CMMs.

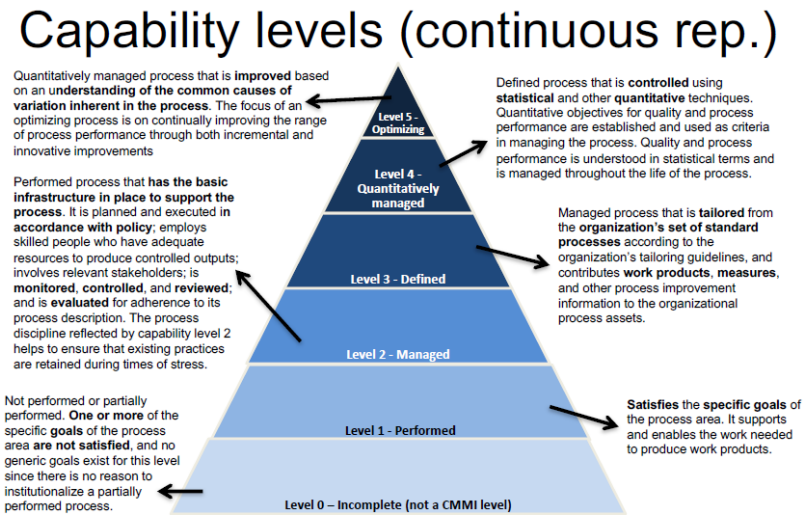


16 shared Core Process Areas

1.Causal Analysis and Resolution (CAR)
2.Configuration Management (CM)
3.Decision Analysis and Resolution (DAR)
4.Integrated Project Management (IPM)
5.Measurement and Analysis (MA)
6.Organizational Innovation and Deployment (OID)
7.Organizational Process Definition (OPD)
8.Organizational Process Focus (OPF)
9.Organizational Process Performance (OPP)
10.Organizational Training (OT)
11.Project Monitoring and Control (PMC)
12.Project Planning (PP)
13.Process and Product Quality Assurance (PPQA)
14.Risk Management (RSKM)
15.Quantitative Project Management (QPM)
16.Supplier Agreement Management (SAM)

CMMI is a **process improvement** approach that provides organizations with the essential elements of effective processes. CMMI can be used in process improvements as a collection of best practices or as a framework for organizing and prioritizing activities.

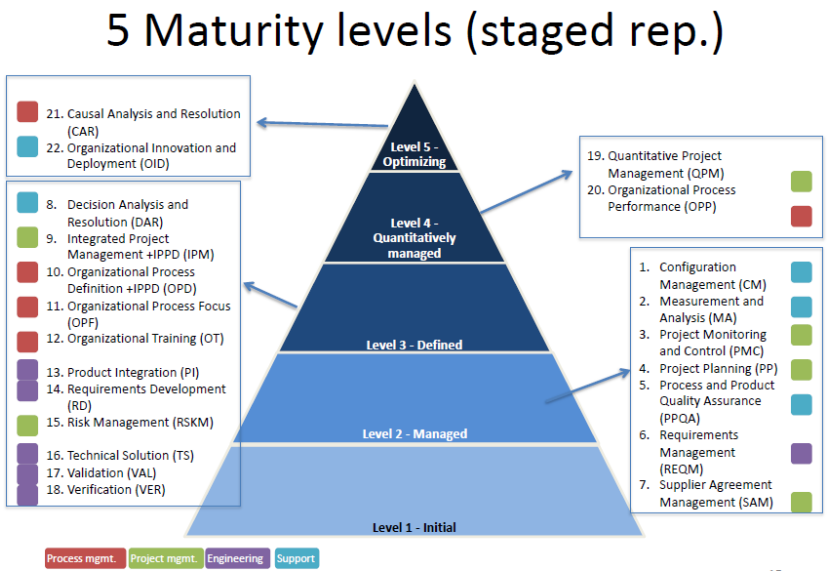
CMMI best practices are described in model, each addressing a different area of interest. CMMI framework is the structure that organizes the components used in generating models. Components in the CMMI framework are organized into constellations, which facilitate construction of approved models.



Process Areas:

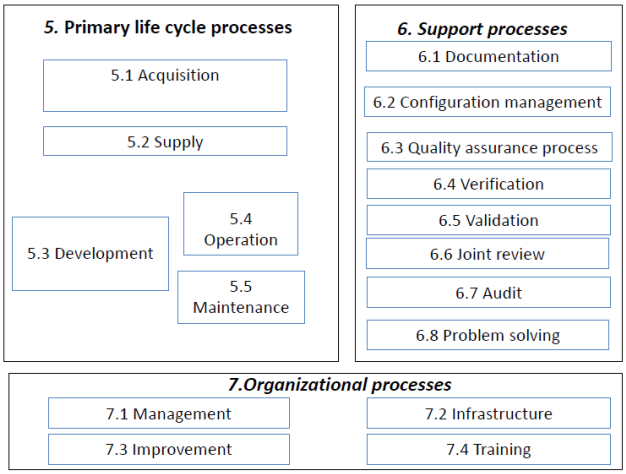
All CMMI models contain multiple Process Areas (Pas), generic goals and practices apply to all Pas (performed process, managed process, defined process). Each PA has 1 to 4 goals, and each goal is comprised of practices.

There are 22 process areas, divided into four category.



ISO 12207 standard – software lifecycle processes:

It defines and structures all activities involved in the Software Development Process. Its main goal is to provide a common language to involve stakeholders. It is based on a set of coordinated activities transforming an input in an output (functional approach). Five primary lifecycle processes related to primary involved agents, eight supporting life cycle processes and four organizational processes.



The standard is based on two basic principles, **modularity** and **responsibility**. Modularity means processes with minimum coupling and maximum cohesion. Responsibility means to establish a responsibility for each process, facilitating the application of the standard in projects where many people can be legally involved.

Each process has a set of outcomes associated with it and is detailed in terms of activities.

ISO 9000 family of standards – quality management systems:

It is intended for being used in any organization which designs, develops, manufactures, install and/or services any product or provides any for of service. It provides requirements which an organization needs to fulfill if it is to achieve customer satisfaction through consistent products and services which meet customer expectations. It is the target of the certification process.

Advantages: better efficiency, continual improvement, less waste, consistent control of key processes, possible reduction in insurance premiums, provision of a vehicle for training new employees, the effective management of risk, increasing the potential for world-wide recognition.

Disadvantages: Too abstract, costly to obtain and maintain, lengthy time-scale to obtain certification, time-consuming development, difficult to implement, organizational resistance to change, staff resistance to change, hard to maintain enthusiasm for the system, more documentation.

The Quality System building blocks:

- Quality management system
  It deals with general and documentation requirements that are the foundation of the management system. The general requirements are how the processes of the management system interact to each other, what resources do you need to run the processes and how you will measure and monito the processes.
- Management responsibility
  The must are: know customers' requirements at a strategic level, make a commitment to meet these requirements as well as statutory and regulatory requirements, set policies and objectives, plan how the objectives will be met, ensure that there are clear internal communications and that the management system is regularly reviewed.
- Resource management
  It deals with the people and physical resources needed to carry out the process.
- Product/service realization
  It deals with the processes necessary to produce the product or to provide the service (act of converting the input to the output), for a manufacturing organization, for a service organization, for a software organization.
- Measurement, analysis and improvement
  It deals with the measurements to enable the system to be monitored.

ISO Certification

ISO does not certify organizations, there are accreditation bodies (that have mutual agreements in order to have a worldwide acceptance) that authorize certification bodies. Both accreditation bodies and certification bodies charge fees for their services.

A set of process requirements and resources that constitute the Quality Manual (QM) of the organization. QM specifies the organization's quality policy regardless specific commitments and customers.

ISO 9001 certification requires that processes are described in specific documents: Quality Manual (QM) and Quality Policy (QP).

Software Development Process

In order to manage a software project the manager needs special methods, the monitoring is based on the explicit definition of ***activities*** *to be performed* and ***documents*** *to be produced*. Produced documents allow to monitor the evolution of the process and provide means to evaluate its quality.

- The waterfall model
  Separate and distinct phases of specification and development (each phases has to be complete before moving onto the next phase):
  1. Requirements analysis and definition
  2. System and software design
  3. Implementation and unit testing
  4. Integration and system testing
  5. Operation and maintenance

  Problems: there is the difficulty of accommodating changes after the process in underway, the initial uncertainty, customer has to wait till a working version is available.

- Evolutionary development
  Specification, development and validation are interleaved.
- Component-based software engineering
  The system is assembled from existing components.
- Formal models
  Requirements are expressed in a formal language. Logic or algebraic formalisms for requirement specification, development and test.

Process iteration:
System requirements always evolve in the course of a project so project iteration is always part of the process for large systems and can be applied to any of the generic process models. Two approaches are *incremental delivery* and *spiral development*.

Prototypal model: (customer interaction → prototype → test with the customer →to the top)
Working prototype implements some basic functions, the main objective is collecting and disambiguating requirements involving the customer.

Incremental model:
Similar to prototypal model but intermediate versions are full working, it allows for a more accurate design.

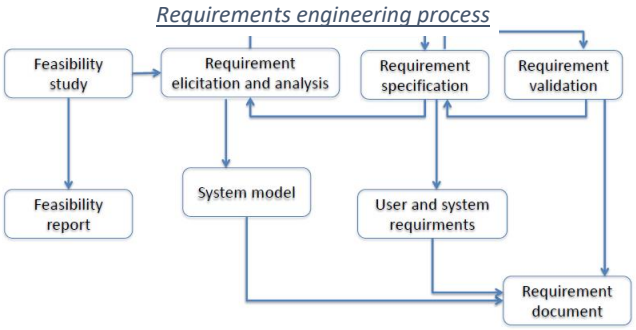
Incremental development:
Delivering part of the required functionality, user requirements are prioritized and the highest priority requirements are included in early increments. Customer value can be delivered with each increment so system functionalities are available earlier. Early increments act as a prototype to help elicit requirements for later increments. Lower risk of overall project failure.

Extreme programming:
Part of Agile model family, based on the development and delivery of very small increments of functionalities.

Spiral development:
Process is represented as a spiral of activities, each loop in the spiral represents a phase in the process. Risks are explicitly assessed and resolved throughout the process. Sectors: objective setting, risk assessment and reduction, development and validation, planning.

Core process activities:

- Software specification
- Software design and implementation
- Software validation
- Software evolution



The process of establishing what services are required and the constraints on the system operation and development:

- Functional requirements: what the software should do
- Non-functional requirements: how the software should do

Software design and implementation:

The process of converting the system specification into an executable system. Design a software structure that realizes the specification (software design), translate this structure into an executable program (implementation).

These two activities are closely related and may be inter-leaved.

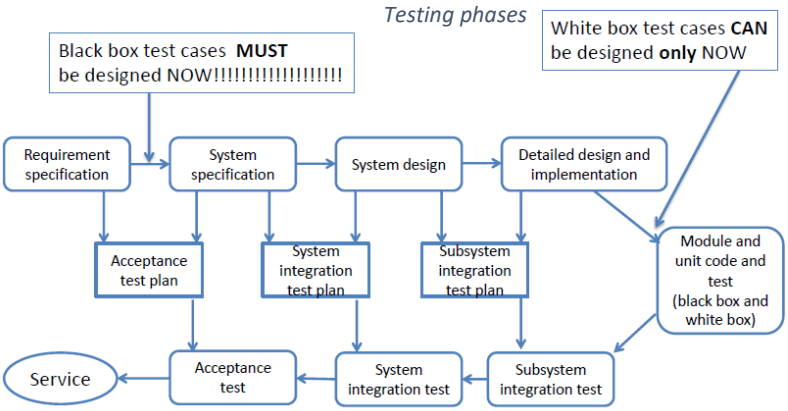
Design activities: architectural design, interface design, component design, data structure design and algorithm design.

Systematic approaches to software design, documented as a set of graphical models (object model, sequence model, state transition model, structural model, data-flow model).

Translating a design into a program and removing errors from that program. Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

The debugging process locate errors, design error repair, repair error and re-test the program.

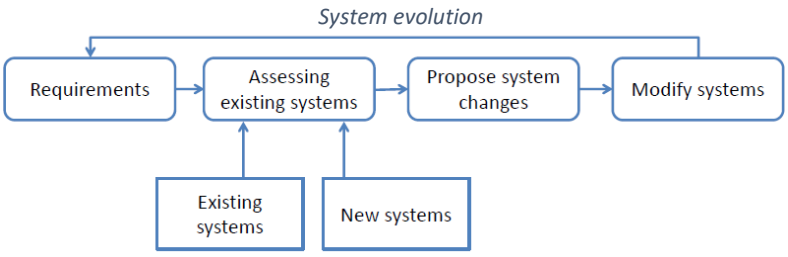
Software validation:



Verification (checks and reviews test with the specification) and Validation (test on the real data of the user) (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer. Involves checking and review processes and system testing.

The testing stage includes component or unit testing, system testing and acceptance testing.

Software evolution:

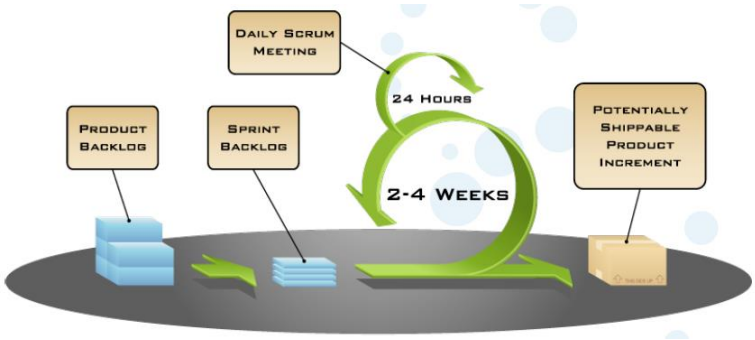


SCRUM

Scrum is an agile process that allows to focus on delivering the highest business value in the shortest time. It allows to rapidly and repeatedly inspect actual working software.

Characteristics:

- Self-organizing teams
- Product progresses in a series of 2-4 week “sprints”
- Requirements are captured as item in a list of “product backlog”
- No specific engineering practices prescribed
- Uses generative rules to create an agile environment for delivering projects
- One of the “agile processes”



Sprints:

Scrum projects make progress in a series of “sprint”, typically them duration is 2-4 weeks or a calendar month at most. Product is designed, coded and tested during the sprint. During the sprint there are no changes.

Roles

- Product owner: define the features of the product, decide on release date and content, adjust features and priority every iteration. Accept and reject work results.
- ScrumMaster: represents management to the project. Removes impediments, ensure that the team is fully functional and productive.
- Team: typically 5-9 people. Cross-functional, members should be full-time.

Ceremonies

- Sprint planning: team selects item from the product backlog they can commit to completing. Sprint backlog is created, tasks are identified and each is estimated. High-level design is considered. The daily scrum avoid other unnecessary meetings, it is not for problem solving.
- Sprint review: team presents what it accomplished during the sprint, typically takes the form of a demo of new features or underlying architecture.
- Sprint retrospective: periodically take a look at what is and is not working, done after every sprints.
- Daily scrum meeting

Artifacts

- Product backlog: the requirements, a list of all desired work on the project (user stories).
- Sprint backlog: sprint goal is a short statement of what the work will be focused on during the sprint. Individuals sign up for work of their own choosing and the work for the sprint emerges.
- Burndown charts: a display of what work has been completed and what is left to complete.

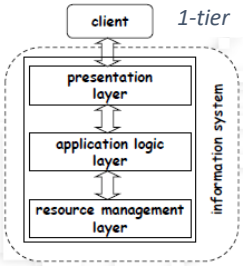
DISTRIBUTED PROGRAMMING – BASIC TECHNOLOGIES

Remote Procedure Call (RPC) & Message Oriented Middleware (MOM)

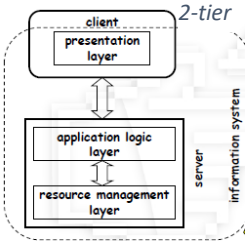
RPC & MOM in Java: Java RMI & JMS

Layers: (logical level) presentation, application logic, resource management. The client is not a layer, it is an entity that does not must be a human, it can be also a program. Presentation layer is the part of the system the client interacts with. Application layer decides what the system should do. The resource management layer is used by application layer, it contains data.

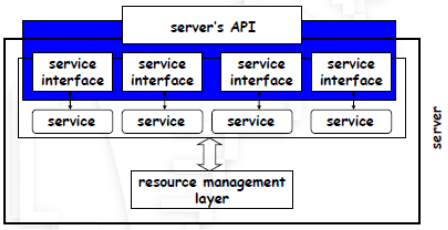
Thiers: (physical level)



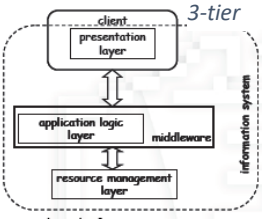
- 1-tier architecture: all is centralized, managing and controlling resources is easier. No forced context switches in the control flow, everything happens within the system.



- 2-tier architecture: client are independent of each other, one could have several presentation layers depending on hat each client wants to do.



It introduces the concept of API (Application Program Interface), an interface to invoke the system from the outside. The resource manager only sees one client, the application logic.



- 3-tier architecture: middleware is a level of indirection between clients and other layers of the system. It introduces an additional layer of business logic encompassing all underlying systems.

Middleware

Blocking interaction (synchronous)

Distributed applications use blocking calls, synchronous interaction requires both parties to be "on-line". Single thread application, the client block (the client sends a request to a service and waits for a response of the service to come back before continuing doing its work).

Disadvantages: connection overhead, high prob of failures, one-to-one system.

Sessions: If the server crashes, I want to be able to send a message to the server to recognize what the server was doing for the client (CONTEXT, e.g. cookies are object that the web server send to the client to remember the session).
Synchronous interaction requires a context for each call and a context management system for all incoming calls.
Solution to synchronous approach:

- *Enhanced support*: client/server systems and middleware platforms provide a number of mechanisms to deal with the problems created by synchronous interaction.
  Transactional interaction to enforce one execution semantics and enable more complex interactions with some execution guarantees.
  Service replication and load balancing to prevent the service from becoming unavailable when there is a failure.
- *Asynchronous interaction*: messages are stored somewhere until the receiver reads it and sends a response.
  Non-blocking invocation.
  Persistent queues.

Non-blocking interaction (asynchronous)

There is no synchronization between client and server, it can happen that the client sends a message before the server was born. Client sends a message on a queue and continues to go on. Server elaborates the message and responses to the client on another queue.

Transactional RPC, Object oriented RPC (RMI), Asynchronous RPC are three types of middleware.

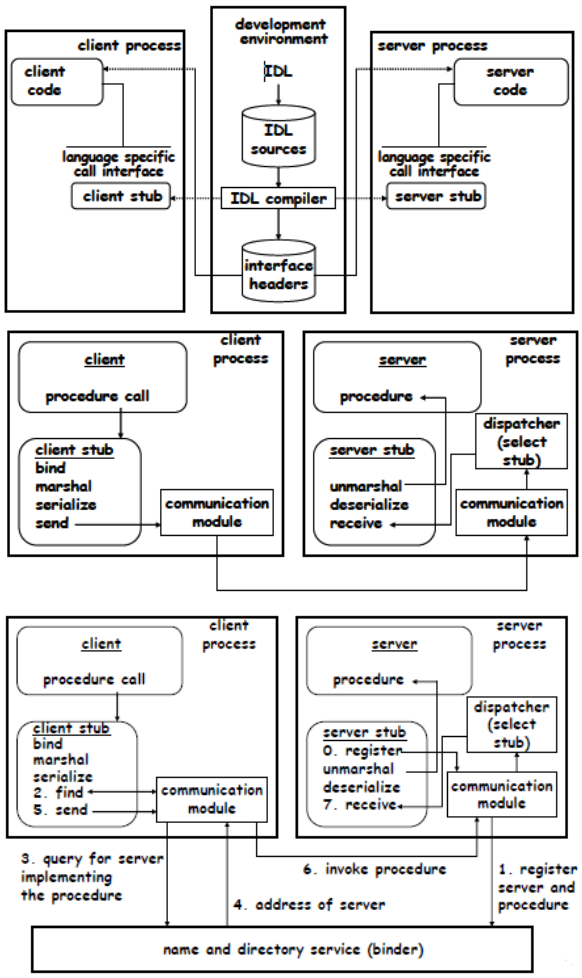
- Remote Procedure Call: hides communication details behind a procedure call and helps bridge heterogeneous platforms.
- Sockets: operating system level interface to underlying communication protocols.
- TCP, UDP: UDP transports data packets without protocols, TCP verifies correct delivery of data streams.
- Internet Protocol (IP): moves a packet of data from one node to another.

Middleware as a Programming Abstraction
Middleware is primarily a set of programming abstractions developed to facilitate the development of complex distributed systems. Programming languages evolve towards higher levels of abstraction, hiding hardware and platform details.

Middleware as Infrastructure
As the programming abstractions reach higher and higher levels, the underlying infrastructure implementing the abstractions must grow accordingly.
The infrastructure is intended to support additional functionality that makes development, maintenance and monitoring easier and less costly. (RPC – Transactional RPC)

RPC

RPC is a point to point protocol in the sense that it supports the interaction between two entities. RPC treats the calls as independent of each other (however they are not independent). Hides distribution behind procedure calls. It is not a good idea to use RPC after a partial system failures.



IDL (Interface Definition Language) compiler generates the stubs, the middleware is the IDL compiler. *IDL is useful to describe services.*

The client calls the server and the client stub mimic the server, the server stub mimic the client.

At run time: in server I have the dispatcher that selects the server. Marshal is that part that transform interfaces in string (transform invocation into a sequence of command on the network).

In client the serialization transforms a string into an array of char (in java this is done by the library).

If the interface is neutral and standard, server and client can have different languages, CORBA is the subject that make the communication possible.

What I need to ensure that the middleware runs is the name and the directory service. It is a module that convert logical name to IP address and port n, like DNS.

When I switch to the server, it does the register operation (in which it say that it is an object, IP, port n) and response to the object name. When a client wants to connect to the server, it finds the specific server object and then does all the procedures.

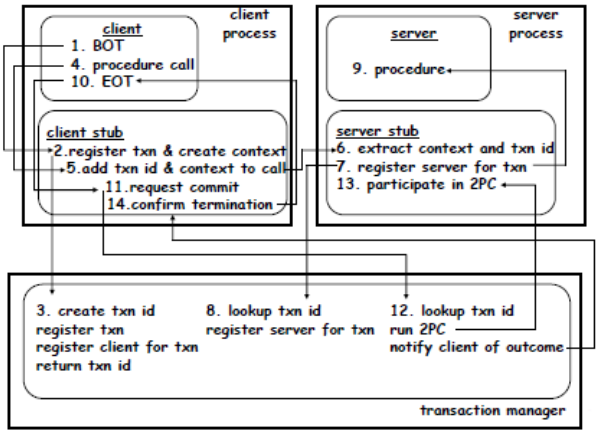
Transactional RPC (TRPC)

It is the first type of middleware.

Same concept as RPC plus additional language constructs and run time support to bundle several RPC calls into an atomic unit. It includes an interface to DBs for making end-to-end transactions using the XA standard.

BOT (begin of transaction), EOT (end of transaction).

JDBC is an example of this, it implements transactional RPC.

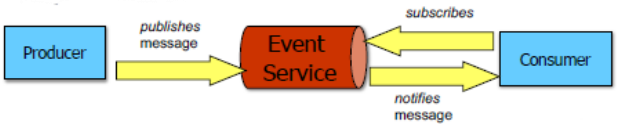


Message Oriented Middleware (Asynchronous/non-blocking interaction)

- Queuing model
- (Primitive) load balancing with shared queue

Publish/Subscribe

A many to many anonymous interaction, the participants are divided into two groups (publishers and subscribers).
Communication takes place through a central entity that receives messages from publishers and cross-check them with the interests of subscribers.



Two types of subscriptions:

- Topic-based: information is divided into topics, ideal logical channel that connects a publisher to all subscribers. With this type the client maybe has to filter for the message.
- Content-based: filters can be used for a more accurate selection of information to be received. With this type the client register with a condition (query) for the message at the level of the instance, over a stream of solution.

Middleware on the internet

All protocols runs on a specific port. On web only specific protocol on specific port are available, because of the security policy. Protocol http, port 80.

RPC became something that do not run over UDP/IP but on http that allows for the port 80 to pass the firewall.

WEB SERVICES

A Web Service is a programmatically available application logic exposed over the Internet. Any piece of code and any application component deployed on a system can be transformed into a network-available service. The functionality is exposed over the internet.
Services reflect a new "service-oriented" approach to programming, based on the idea of composing applications by invoking network-available services rather than building new applications to accomplish some task.

Application Service Providers
An ASP rents applications to subscribers, it hosts the entire application and the customer has little opportunity to customize it beyond the appearance of the user interface.

Software-as-a-Service
The concept here is that the customer pay for the service. Typically download on demand and the software runs on the client.

Web Service
Many different providers offer their backend functionalities. Someone repack the functionalities to offer the system to the client. The software runs partially on the web and partially on the client.
Informational services are services of relatively simple nature. They either provide access to content interacting with an end-user by means of simple request/response sequences, or expose back-end business applications to other applications. *Complex services* involve the assembly and invocation of many pre-existing services possibly found in diverse enterprises to complete a multi-step business interaction.
Properties:

- The *functional service* description details the operational characteristics that define the overall behavior of the service.
- The *non-functional* description targets service quality attributes

State:

- Services that can be invoked repeatedly without having to maintain context or state they are called *stateless*.
- Services that require their context to be preserved from one invocation to the next are called *stateful*.

Web services are *loosely coupled*, they connect and interact more freely across the Internet. They need not know how their partner applications behave or are implemented.
Simple services exhibit normally a request/reply mode of operation are of *fine granularity*.
Complex services involve interactions with other services and possibly end-users in a single or multiple sessions, they are *coarse-grained* (larger and richer data structures).
Synchronicity:

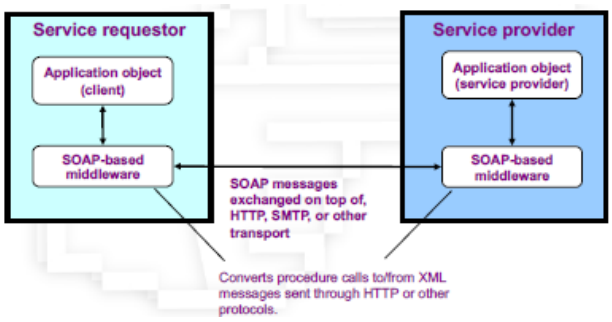
- Synchronous or remote procedure call (RPC)-style
- Asynchronous or message (document)-style

Service interaction must be well-defined. The *web services description language (WSDL)* allows applications to describe the rules for interfacing and interacting to other applications.
The *service interface* defines service functionality visible to the external world and provides the means to access this functionality. The *service implementation* realizes a specific service interface whose implementation details are hidden from its users.
Roles:

- Service providers, organizations that provide the service implementations.
- Service clients, end-user organizations that use some service.
- Service registry, searchable directory where service descriptions can be published and searched.

Service-Oriented Architecture (SOA) is a logical way of designing a software system to provide services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces.
Quality of Service (QoS) refers to the ability of a Web Service to respond to expected invocations and perform them at the level commensurate to the mutual expectations of both its provider and its customers.

Communication & SOAP



SOAP is the standard messaging protocol used by Web Services. SOAP's primary application is inter application communication. SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a means for transport.

SOAP supports two possible communication styles: RPC and document (or message).

RPC-style: appears as a remote object to a client application. The interaction centers around a service-specific interface. Clients express their request as a method call with a set of arguments, which returns a response containing a return value. An XML file containing information in the header and the request of the operation in the body. RPC over http.

Document (message)-style: the SOAP <body> contains an XML documents fragment. The SOAP run-time environment accepts the <body> element as it stands and hands it over to the application it is destined.

The typical binding for SOAP is http.

I have SOAP engine, SOAP infrastructure (CXF apache). XML schema are used to communicate what the server expect.

Service description & WSDL

A Web Service must be defined in a consistent manner to be discovered and used by other services and applications. Web Service consumers must determine the precise XML interface of a Web Service.

Service description is a machine understandable standard describing the operations of a Web Service. Service description and SOAP infrastructure isolates all technical details.

Web Services Description Language is the XML-based service representation language used to describe the details of the complete interfaces exposed by Web Services and thus is means to accessing a Web Service. It is an XML document that describes the mechanics of interacting with a particular Web Service, it represents a "contract" between the service requestor and the service provider.
Sections:

- The *service-interface definition* contains all the operations supported by the service, the operation parameters and abstract data types.
- The *service implementation part* binds the abstract interface to a concrete network address, to a specific protocol and to concrete data structures.

This enables each part to be defined separately and independently and reused by other parts.
WSDL describe 4 patterns and there are 2 basic communication, request/response or the message:

- One way messaging from the sender (client) to the receiver (server)
- Request/response messaging like RPC
- One way messaging from the receiver (server) to the sender (client)
- Solicit/response messaging

Registry and UDDI (Universal description, discovery and integration)

It is a registry standard for Web Service description and discovery together with a registry facility that supports the WS publishing and discovery processes. Enables a business to describe its services, discover other services and integrate with other services.

RESTful Services

REST refers to simple application interfaces transmitting data over HTTP without additional layers as SOAP. Web page meant to be consumed by program as opposed to a Web browser, this require an architectural style to make sense of them, because there's no smart human being on the client end to keep track.

URIs = nouns
Verbs = describe actions that are applicable to nouns (GET, POST, PUT)
State = means the application/session state, maintained as part of the content transferred (in XML) from client to server back to client

REST is a kind of RPC, except the methods have been defined in advance. Its principles are addressability (each resource an URL), uniform interface (http GET, PUT,…), stateless interaction (no session), self-describing messages (metadata) and hypermedia (hyper-links).
Rest is incompatible with "end-point" RPC, either you address data objects or you address "software components", REST does the former and end-point RPC does the latter.