# Software Architecture

Lecture 8
Service-Oriented Architectures

João Pedro Sousa
George Mason University

## previously

**data flow**
- batch sequential
- dataflow network (pipe & filter)
  - acyclic, fan-out, pipeline, Unix
- closed loop control

**call-return**
- main program/subroutines
- information hiding - objects
- stateless client-server
- SOA

**interacting processes**
- communicating peers
- event systems
  - implicit invocation
  - publish-subscribe

**data-oriented repository**
- transactional databases
  - stateful client-server
- blackboard
- modern compiler

**data-sharing**
- compound documents
- hypertext
- Fortran COMMON
- LW processes

**hierarchical**
- tiers
  - interpreter
  - N-tiered client-server

## previously
## call-return styles

- single process flavors
  - main-subroutine, layers, modules, objects
- distributed flavors
  - components, tiers
  - implementing distributed call-return: RPC, RMI
- large-scale, open-ended distributed flavors
  - SOA



## today
## large-scale, distributed call-return

- enabler: the Internet
- widely-distributed client-server
  - example: World-Wide Web
- the hinge of service-orientation:
  service discovery
- mainstream implementations of SOA
  - web services
  - composition
  - UDDI, SOAP

Acknowledgment
some of the material presented in this course is adapted from 17655, taught to the MSE at CMU by David Garlan and Tony Lattanze

## the Internet began as cold war project
within ARPA
later Defense Advanced Research Projects Agency

- communications system that would survive a nuclear exchange: ARPANET
  - no centralized control point
  - impervious to EMP (electromagnetic pulse)
- 1967 - initial plan for connecting 4 research sites
  - ultimately create a public utility to transmit computer data
- 1971 - 15 nodes on the ARPANET
  - UCLA, SRI, UCSB, U. Utah, BBN, MIT, RAND, SDC, Harvard, Lincoln Labs, Stanford, UIUC, CWRU, CMU, NASA/Ames
  - Ray Tomlinson, a scientist from Massachusetts, sends himself an email between two computers
  - the initial *killer app* became e-mail



## ARPA establishes TCP/IP in 1982
Transmission Control Protocol/Internet Protocol

- network software and hardware was non-standard and hand-crafted before TCP/IP
- TCP/IP
  - provides *layered abstraction* of network services
  - sets the stage:
    - Local Area Networks (LANs)
    - an internet as a set of LANs connected via IP
    - an *intranet* as a private/corporate internet
    - Internet as the global network

# ISO's OSI reference model in 1983

Open Systems Interconnection

## extends the ideas in TCP/IP

| | |
|---|---|
| 7 | application |
| 6 | presentation |
| 5 | session |
| 4 | transport |
| 3 | network |
| 2 | data link |
| 1 | physical |

goal: separation of concerns
enables good implementation
at each level

- each layer is independent
of the ones on top
- layer n depends on the spec of n-1,
but not on its implementation/manufacturer



# the OSI reference model

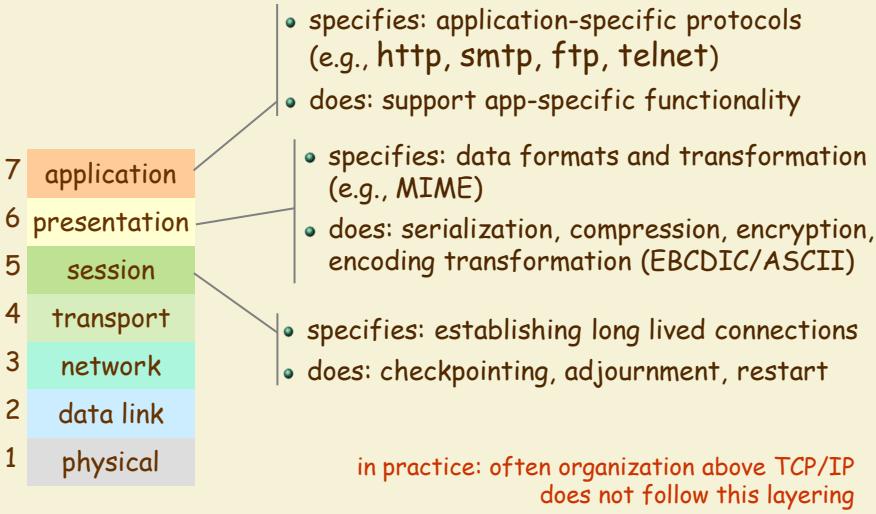
## is roughly adhered in practice

| | | |
|---|---|---|
| 7 | application | |
| 6 | presentation | |
| 5 | session | |
| 4 | transport | segments |
| 3 | network | packets |
| 2 | data link | frames |
| 1 | physical | bits |

realm of middleware
app-specific (SMTP, http…)
or independent (RMI, CORBA…)

TCP/IP protocol stack

# the upper layers of the OSI




in practice: often organization above TCP/IP does not follow this layering



# Internet expanded fast

- 1984 - hosts on the Internet tops 1,000
  - *DNS* introduced
    Domain Name System
    more in a bit
- 1987 - Number of hosts tops 10,000
- 1988 - worm burrows through the net affecting 6,000 of 60,000 hosts on the Internet
  - CERT formed by DARPA
    Computer Emergency Readiness Team
- 1989 - number of hosts tops 100,000

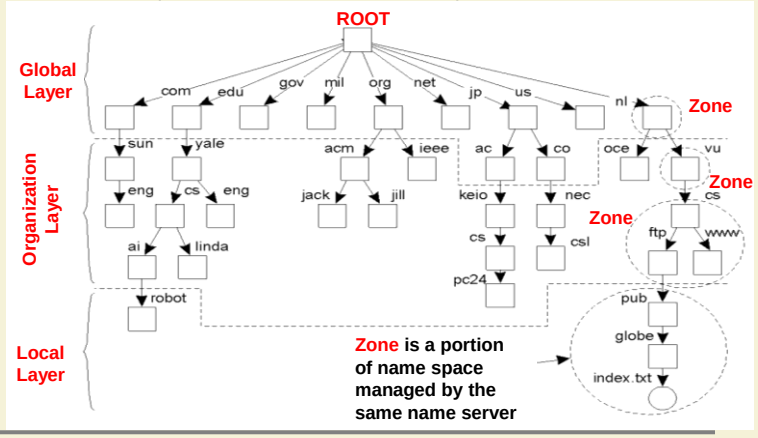
# to manage scale: human readable URLs

Universal Resource Locators

example ftp://ftp.cs.vu.nl/pub/globe/index.txt

access protocol | host name | local name

resources are servers and files



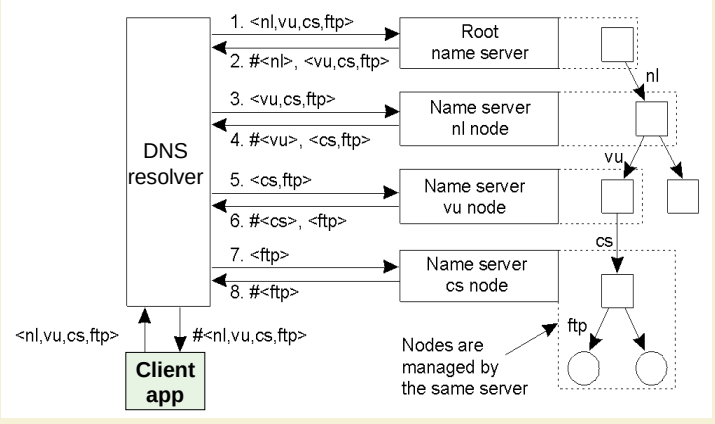



# DNS servers resolve URLs to IP addresses

example: resolve *ftp://ftp.cs.vu.nl/pub/globe/index.txt*

in practice: a resolver on your PC, another on your ISP…

each with caching




#<...> stands for 32-bit IP address of server <...>

## the development of the internet led to the client-server style

- replaces mainframe/dumb-terminal model

different viewpoints:

- physical (allocation viewtype)
  - *clients* are user computers (many)
  - *servers* are central computers (few)
  - the connectors are the physical telecom infrastructure (cables, routers, etc.)
- run-time (C&C viewtype)
  - *clients* are user processes
  - *server* are central resources/processes
  - the connectors are software working over TCP/IP

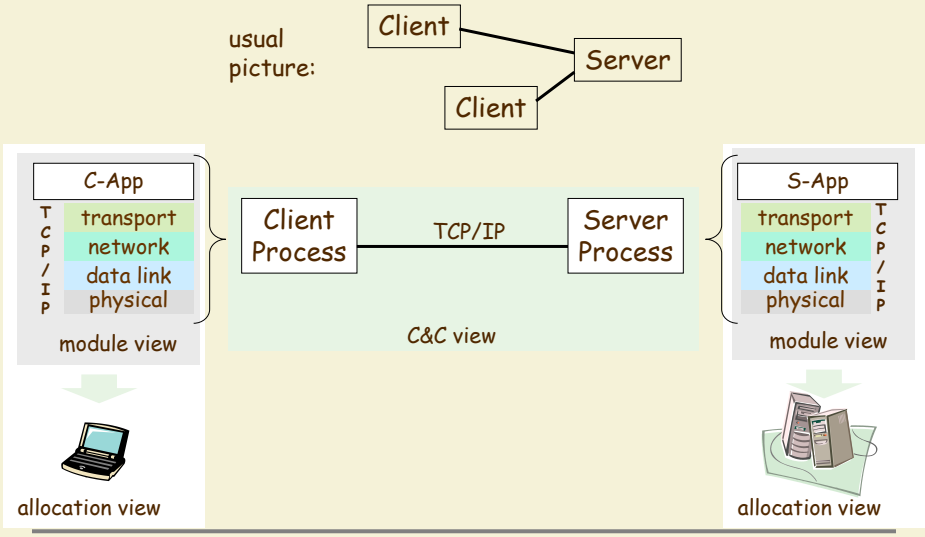


## architecturally, the client-server style:

- elements
  - clients, servers, call-return connectors
    - connector implementations: $RPC_{1976}$ , $RMI_{1995}$, $SOAP_{1998}$... over $TCP/IP_{1982}$ or $http_{90's}$
- topology
  - star, tiered (hierarchical star)
  - servers don't know the identities/number of clients that will request services
  - clients know the identity of a server$_{<1998}$ or can *discover* it$_{>1998}$

more in a bit

# relation among views
## client-server

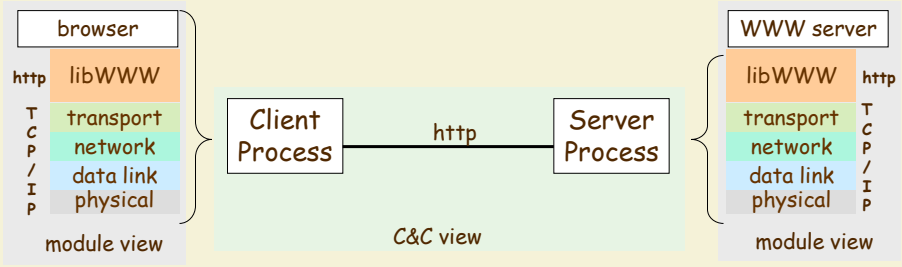





## client-server example
# World-Wide Web

- hypertext idea first published in 1965
  by Ted Nelson as part of the *Xanadu* project
- Tim Berners-Lee, a software engineer at CERN,
  saw the potential of this idea for the Internet in 1989
- initially rejected, Tim re-circulated his proposal
  and writes first WYSIWYG browser in 1990
  - libWWW supports new protocol http, on top of TCP/IP
    hypertext transfer protocol
  - new format for documents: HTML
    HyperText Markup Language
  - the next *killer app* became web browsing (aka surfing)

# client-server example
# World-Wide Web




- thin clients, called browsers
  - invoke DNS to resolve URLs
  - request documents to WWW servers over http
  - interpret HTML documents
- servers
  - host data in many formats (in addition to HTML)
  - serve http requests

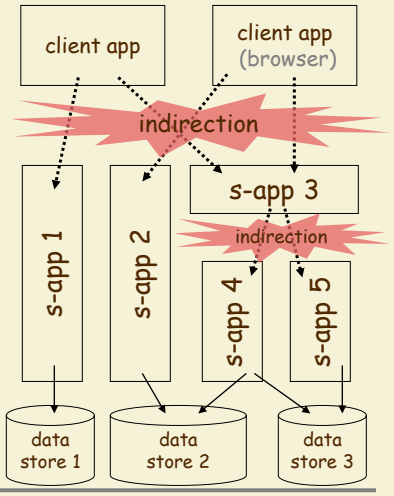


# role of Internet continued to expand to business applications

- client-server applications have been around since the early days of the internet
  - early 80's: proprietary app protocols over TCP/IP
  - late 80's: candidate standards emerge, e.g. CORBA, DCOM... with some success, sometimes in specialized domains
- in the late 90's push for business over the Internet
  - requirements considerably different from
    - original vision of ARPANET => led to TCP/IP
    - vision at CERN/Berners-Lee => led to http
- vision SOA: Internet-wide protocol for e-business
  - the next *killer app* is e-services
  - will web services fill the role of supporting technology?

# Service Oriented Architectures are evolution of tiered style

- complex apps already existed
  - normally all components hosted/maintained by the same organization
- SOA adds level of indirection
- service
  - is a unit of work
  - several candidate providers
    - maybe hosted by diff organizations
  - a provider may be *discovered*
    - before deployment, or
    - dynamically at run time






# take 5

# outline

- enabler: the Internet
- widely-distributed client-server
  - example: World-Wide Web
- the hinge of service-orientation:
  service discovery
- current implementation of SOA
  - web services
  - composition
  - UDDI, SOAP



# concept of *service discovery* develops in ubiquitous computing circa 1998

- an application may need to find
  some component with certain capabilities
- discovery is guided by the capabilities,
  not the identity of servers, aka service providers
  examples:
  - find a duplex printer < 100 ft away and with < 2 minute wait
  - find a weather forecast website with wind details for x ZIP code
  - find a speech recognizer with > 95% accuracy

## the *service* becomes more important than the *identity* of the server (URI/URL)

- business benefits from competition among providers that offer similar functions/services
- fundamental tenet is shared vocabulary to describe services
  some combination of:
  - name, aka service type
    e.g. printing, weather forecasting, ticket reservation…
  - semantic description, e.g. ontology
  - API signature specification: methods, parameters, results
    - WSDL, etc. with IDL ancestors all the way back to in the 80's
      Interface Description Languages



## discovery mechanisms use different patterns

- directed discovery
  - clients are configured with a list of address to go ask for services
- client-initiated broadcast (aka aggressive)
  - clients broadcast service requests on demand
- supplier-initiated broadcast (aka lazy)
  - suppliers broadcast their capabilities periodically
- directory-based discovery
  - suppliers post their capabilities on a directory
  - clients query the directory

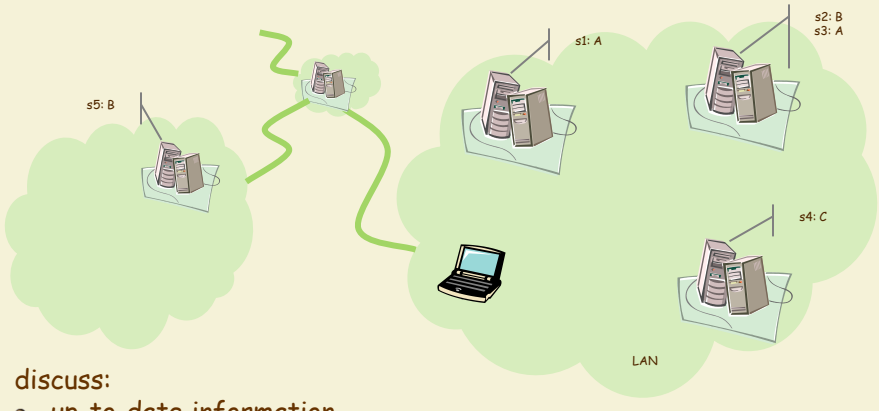
## example: client needs A at t1, and B at t2
# directed discovery


discuss:

- up-to-date information
- scalability/scope
- security/trust
- applicability/practicality



## example: client needs A at t1, and B at t2
# client-init broadcast




discuss:

- up-to-date information
- scalability/scope
- security/trust
- applicability/practicality

example: client needs A at t1, and B at t2

## server-init broadcast

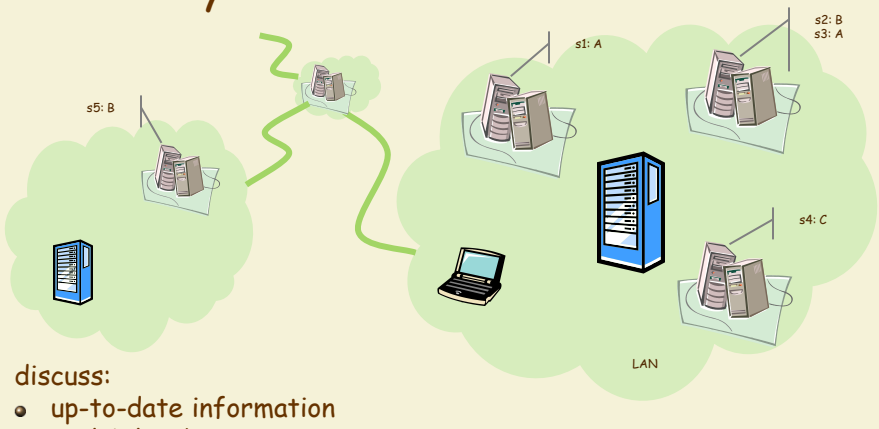

discuss:

- up-to-date information
- scalability/scope
- security/trust
- applicability/practicality



example: client needs A at t1, and B at t2

## directory-based




discuss:

- up-to-date information
- scalability/scope
- security/trust
- applicability/practicality
- discover directory?

## discovery example
## Jini(*)

- "discovery"
  - service consumers & providers broadcast their existence in the hope of finding a "lookup service" (directory)
- "join"
  - service provider registers with lookup service(s)
- "lookup"
  - service consumer queries lookup service for *service name*
  - service stub is shipped to consumer site handles remote communication with service via RMI
- robustness
  - service registration is "leased" (expires)

(*) initially by Sun in 1995, now at Apache: River project



## other discovery examples

- SLP: Service Location Protocol
  - language independent
  - open source
  - many commercial applications
- Salutation
  - open source
  - IBM leadership
- UPnP: Universal Plug and Play
  - Microsoft leadership
  - multicast announcement
- many research prototypes

## service discovery vs. event publish-subscribe

### discovery

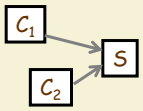
- broadcast or directory
- service providers
  - announce/register capabilities
- service consumers
  - lookup providers
- service requests
  - directed (call-return) from one consumer to one provider

### pub-sub

- broadcast or event bus
- event producers
  - determine types of events
- event consumers
  - register interest in events
- event producers
  - announce events
- events
  - delivered to all (maybe zero) registered consumers



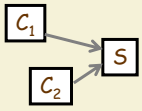
## service-orientation




- promote scalability
  - easy to add service consumers/clients
  - feasible to add replicas of a supplier/server (usual technique)
  - possible to add new service suppliers
- promote robustness
  - upon failure, consumer may find another service supplier
- promote maintainability
  - assemble new features from available services
  - deploy and announce enhanced services

easy with dynamic discovery

# service-orientation




- promote security
  (relative to event systems)
  - each server may set up encryption and access control to authorized clients
- conceptual integrity
  reliability
  & performance may be challenges
  - service consumers depend entirely on service providers



# outline

- enabler: the Internet
- widely-distributed client-server
  - example: World-Wide Web
- the hinge of service-orientation:
  service discovery
- current implementation of SOA
  - web services
  - composition
  - UDDI, SOAP

# web services
## come in as an integration technology

- focus on bridging existing technologies
  - it's about how to access a service
  - unlike previous middleware,
    it is not an implementation infrastructure
- raises level of abstraction
  - avoid proprietary APIs
  - SOAP originally Simple Object Access Protocol @ $MS_{1998}$
    - based on sending XML messages over http
    - no SOAP API or ORB
- wider industrial support than previous middleware
  - CORBA, Microsoft's .net, IBM's Webshpere, Sun's J2EE



# web services
## introduces a set of specifications

| | |
|---|---|
| directory | UDDI<br>universal description, discovery and integration |
| service description | WSDL<br>web services description language |
| messages | SOAP<br>simple object access protocol /<br>service-oriented architecture protocol |

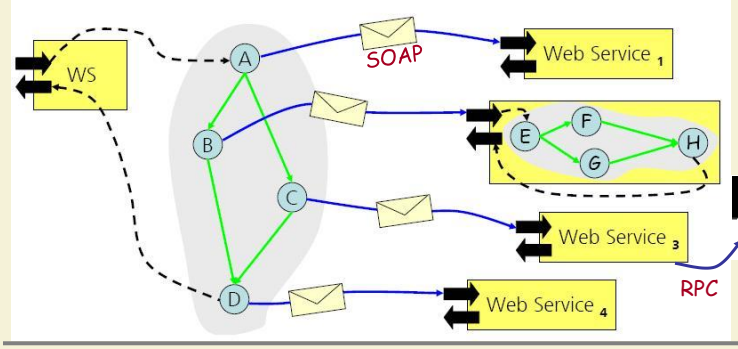
which are defined on top of:

| | |
|---|---|
| data types | XML Schema |
| data | XML<br>eXtensible markup language |

...and:

# multiple proposals for service composition and coordination

- Business Process Execution Language BPEL
- Web Services Conversation Language WSCL
- Web Services Coordination WS-C
- Web Services Transaction WS-Tx




credits: Frank Leymann, Kai Guentzel



# WS descriptions may be posted on UBRs

## UDDI Business Registry

1. SW companies, standards bodies, and programmers populate the registry with descriptions of different types of services

4. Marketplaces, search engines, and business apps query the registry to discover services at other companies

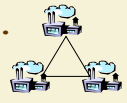
UDDI Business Registry

2. Businesses populate the registry with descriptions of the services they support

Business Registrations

Service Type Registrations

3. UBR assigns a programmatically unique identifier to each service provider



5. Business uses this data to facilitate easier integration with each other over the Web

credits: Tevfik Bultan

# foundation of SOAP

XML-RPC in Microsoft 1998 -> W3C 2003-7

- works on top of HTTP/HTTPS
  or SMTP (less popular)
- critics of this decision point out that HTTP
  was not designed for calling services back and forth
  - e.g. a SOAP operation implemented on top of HTTP *get*
    may not be idempotent as the semantics of *get* implies
- supporters point out that it's normal to tunnel
  protocols on top of each other
  and that it saves a lot of work
  (e.g. dealing with firewalls – a challenge for DCOM)



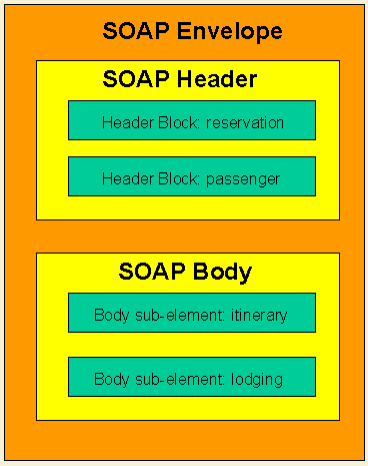
# SOAP is…

- stateless, one-way message exchange
  applications can create more complex interaction
  patterns (request/response, request/multiple responses, etc.)
  - combining one-way exchanges with
    features provided by the underlying protocol
  - application-specific logic
- silent on the semantics of any data it conveys

but

- describes the actions required of
  a SOAP node upon receiving a SOAP message

# structure of a SOAP message




example: travel reservation

- optional
  extension mechanism
  e.g., directives on how to process the message
- application payload



```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <t:transaction xmlns:t="http://thirdparty.example.org/transaction"
                   env:encodingStyle="http://example.com/encoding"
                   env:mustUnderstand="true" >5</t:transaction>
  </env:Header>
  <env:Body>
    <m:chargeReservation
          env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
          xmlns:m="http://travelcompany.example.org/">
      <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
        <m:code>FT35ZBQ</m:code>
      </m:reservation>
      <o:creditCard xmlns:o="http://mycompany.example.com/financial">
        <n:name xmlns:n="http://mycompany.example.com/employees">
          Åke Jógvan Øyvind </n:name>
        <o:number>123456789099999</o:number>
        <o:expiration>2005-02</o:expiration>
      </o:creditCard>
    </m:chargeReservation>
  </env:Body>
</env:Envelope>
```

# example
## call-return communication in SOAP

## many implementations of SOAP today

- Apache SOAP/Axis – Java/C++
- PocketSOAP – COM/C++
- SOAP::Lite - Perl
- PHP SOAP - PHP
- gSOAP – C++
- SOAP4R - Ruby
- Python web services project – Python
  - and these are only the open source ones…

rely on a common understanding of the structure and meaning of the exchanged messages



## in summary

- SOA combines
  - distributed call-return connectors
  - service discovery mechanisms
- web services propose a set of technologies/protocols to implement SOA
  - currently does not support dynamic discovery
- dynamic service discovery plays a key role in achieving QAs
  - scalability
  - robustness
  - maintainability

these are general considerations: remember that a real analysis requires QA scenarios