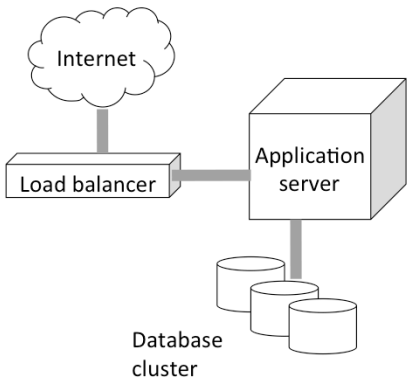
Cloud best practices

TIE-23600 Palvelupohjaiset järjestelmät

Cloud architectures: Transactional web application architecture

- Separation into
  - presentation,
  - business logic, and
  - data storage



[CloudArchitectures]

Siirtäminen pilveen voi tarkoittaa käytettävän ohjelmiston, tallennustilan tai infrastruktuurin siirtämistä verkkopohjaisiin palveluihin tai korvaamista niillä.

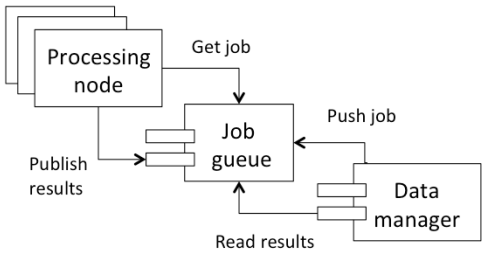
Arkkitehtuuri on olennainen asia, joka vaikuttaa siihen, miten sovelluksen siirtäminen pilveen onnistuu. Ensinnäkin sovelluksella tulee olla selkeä ja dokumentoitu arkkitehtuuri. Tällä ja seuraavalla Kalvolla on esitetty kaksi Web sovelluksille tyypillistä arkkitehtuurityyliä, jotka soveltuvat hyvin siirrettäväksi pilveen.

Web-sovellukset rakentuvat tyypillisesti tietovaraston varaan. Tällöin hyvä käytäntö on erottaa sovelluksen käyttöliittymä, business logiikka ja tietokanta toisistaan.

Grid-arkkitehtuurissa datan prosessointi on erotettu muusta sovelluslogiikasta. Tämä voidaan toteuttaa ns. jonoarkkitehtuurin avulla. Rinnakkain toimivat prosessointiyksiköt hakevat työjonosta töitä sitä mukaa kun ne ovat valmiita käsittelemään uuden työ.

Cloud architectures: Grid application architecture

- Separation of the core application from its data processing nodes



[CloudArchitectures]

Cloud best practices

1. Design for failure
2. Loose coupling
3. Implement “Elasticity”
4. Think Parallel
5. Build security in every layer - Design with security in mind
6. Don't fear constraints - Re-think architectural constraints
7. Leverage different storage options - One DOES NOT fit all

[CloudBestPractices]

Seuraavaksi tarkastellaan muutamia hyviä pilvijärjestelmien arkkitehtuuriratkaisuja (kohdat 1-4).

Kohdat 5-7- kuvaavat muita hyviä toimintatapoja:

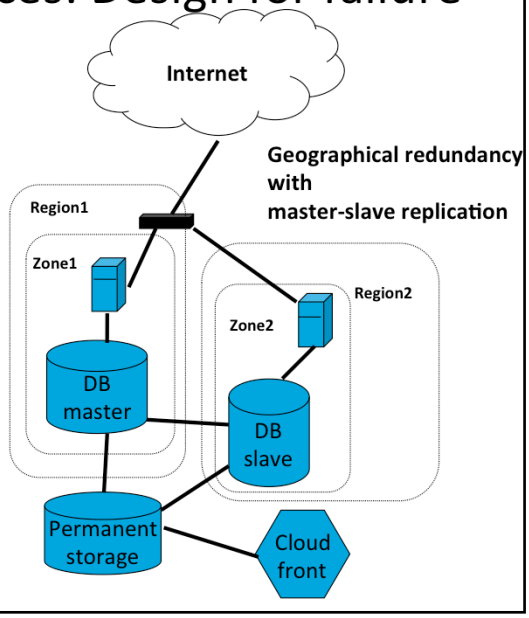
5. Tietoturvan suhteen tulisi noudattaa ns. onion model -mallia, eli tietoturvaan tulee kiinnittää huomiota kaikilla sovelluksen tasoilla (tiedon tallennus, tiedon siirto, käyttöoikeudet, jne.)

6. Vanhat tutut ratkaisut eivät välttämättä päde pilviympäristössä sellaisenaan, vaan jotkin asiat on tehtävä toisin.

7. Esimerkiksi Amazon WS tarjoaa monenlaisia tiedon tallennus paikkoja. Ne on optimoitu eri käyttötarkoituksiin (esim. haut, turvallinen säilytys, jne.). Joista kaikki ei ole suinkaan tarkoitettu pysyväksi säilytys paikaksi. Tämä tulee ottaa huomioon sovelluksen suunnittelussa. Myös sijainnilla on Väliä (saantiviive).

Cloud best practices: Design for failure

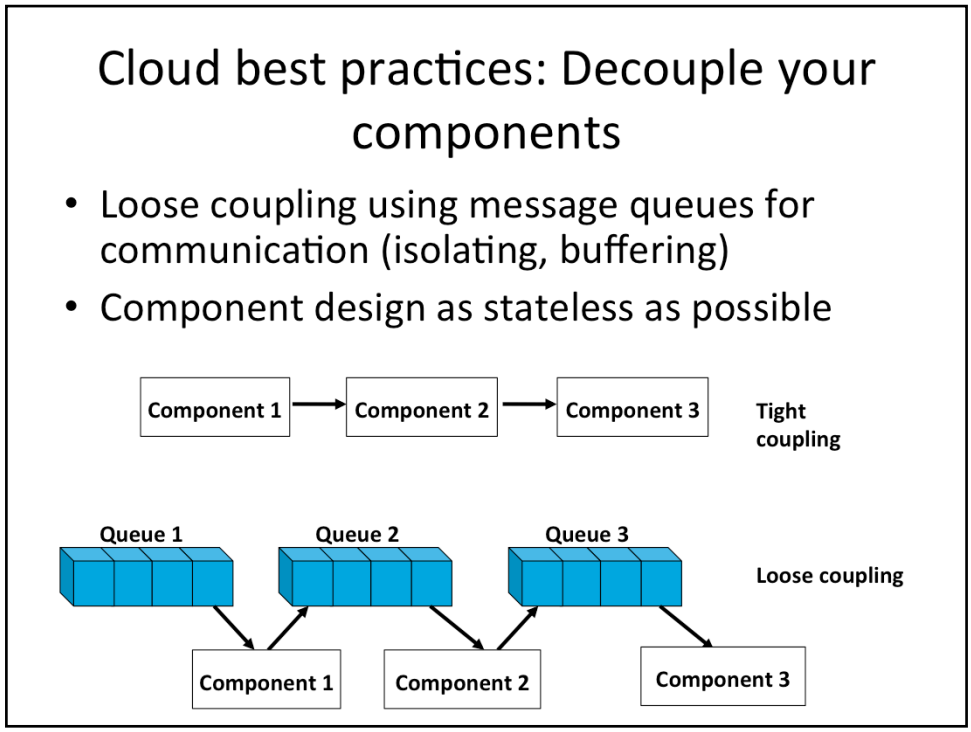
- No single points of failure
  - Replication
  - Monitoring
  - Load balancing
  - Backups
  - Snapshots,
  - …



”Designed to fail” vai ”Design for failure”? Ei tulisi olla yhtä yksittäistä kohtaa, joka virhetilanteessa vaarantaa koko järjestelmän toiminnan. Replikointi on pilviympäristössä yksi tapa välttää edellä mainittu tilanne. Esimerkiksi master ja slave tietokannat voidaan sijoittaa fyysisesti eri puolille maailmaa (maantieteellinen replikointi). Amazon WS ympäristössä virtuaalikoille määritetään aina region, joka vastaa palvelun maantieteellistä sijaintia (Eurooppa, Yhdysvallat jne.)

Ns. snapshot on tietyllä hetkellä tallennettu toimiva konfiguraatio, joka voidaan palauttaa virhetilanteessa.

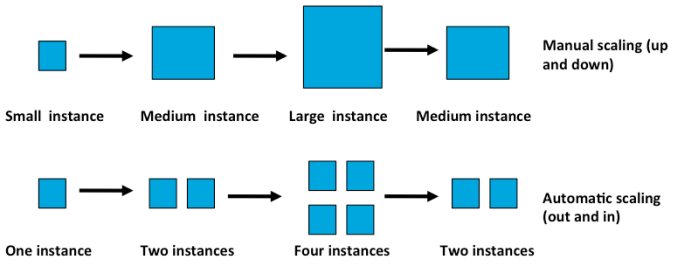
Fyysinen sijainti vaikuttaa myös saantiviiveeseen (latency): prosessoitava tieto säilytetään sielä, missä laskenta tapahtuu ja pysyvä tallennuspaikka puolestaan lähellä asiakasta.



Komponenttien välisiä riippuvuuksia voidaan vähentää käyttämällä viestijonoa. Sen sijaan, että komponentit kommunikoisivat suoraan keskenään, ne kommunikoivat jonon välityksellä. Viestijonon käyttö tukee myös mm. rinnakkaista töiden prosessointia kuten aiemmin esitetyssä Grid-sovellusarkkitehtuurissa.

Cloud best practices: Elasticity

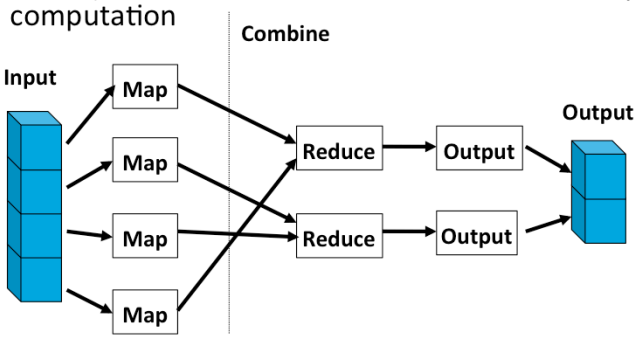
- Scaling (e.g. machine configurations, storage, computing capacity)
- Monitor system metrics
- Use load balancing tools
- Automatize
  - Scale based on variability in usage



Elastisuudella viitataan pilvijärjestelmien yhteydessä tallennus- ja laskentakapasiteetin skaalautuvuuteen kuorman kasvaessa ja pienentyessä. Käytössä olevien resurssien lisäksi tähän liittyy oleellisesti järjestelmän konfiguraation hallinta, monitorointi ja kuormantasaus. Pilvilaskennassa pyrkimys on mahdollistaa käytettävien resurssien skaalautuvuus automaattisesti. Kalvolla on esitetty ratkaisumalli, joka perustuu siihen, että eri kokoisten instanssien sijaan resurssien lukumäärää muutetaan kuorman mukaan. Esimerkiksi laskenta hajautetaan siten, että virtuaalikoneiden määrää voidaan muuttaa tarpeen mukaan. Seuraavalla kalvolla esitetään yksi hajautetun laskennan malli, jossa kaksi yksinkertaista ohjelmaa (map ja reduce) hajautetaan usealle eri (virtuaali)koneelle.

Cloud best practices: Parallel and distributed computing

- E.g., Create job flows using MapReduce
  - Designed for scalable processing of large amount of data
  - Automatic distribution of work load
  - Two simple programs, map(key, value) and reduce(key, values), are distributed in several machines for parallel computation



MapReduce on Googlen kehittämä arkkitehtuuri, jossa kaksi yksinkertaista ohjelmaa (map ja reduce) hajautetaan laskettavaksi rinnakkain useille koneille. Tarkoituksena on skaalautua käsittelemään valtavia datamääriä niin, että kehittäjän ei tarvitse huolehtia hajautukseen liittyvistä yksityiskohdista.

Sisääntulona voi olla mitä tahansa, mistä voidaan tehdä järkeviä avain-arvo pareja. Tieto virtaa sisääntulosta ensin mapin ja sitten reducen läpi ulostuloksi avain-arvo pareina. Avain-arvo-parien käyttö tekee map- ja reduce-ohjelmien rajapinnan yksinkertaiseksi. Myös tiedon hajautus ja yhdistys yksinkertaistuu. Map- ja reduce-funktioita ajetaan rinnakkain useita instansseja useilla koneilla (ensin map ja sitten reduce).

MapReduce

1. **Map**: is run to each key-value pair of the input and it produces a list of preliminary values

$\langle key_{input}, value_{input} \rangle \rightarrow \textbf{map} \rightarrow \langle key_{output}, value_{intermediate} \rangle$

2. **Sort/combine**: values are grouped according to keys

3. **Reduce** is run to a list of values for each key and it produces a list of final values

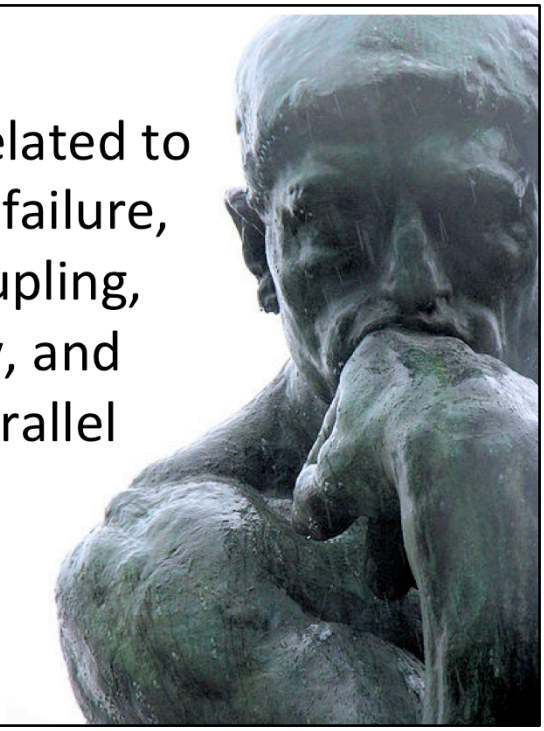
$\langle key_{output}, list(value_{intermediate}) \rangle \rightarrow \textbf{reduce} \rightarrow \langle key_{output}, list(value_{output}) \rangle$

[MapReduce]

MapReduce vaiheet yksinkertaistettuna:

1. map: suoritetaan jokaiselle sisääntulon avain-arvo parille ja se tuottaa väliaikaisia arvoja, joilla on jokin lopullinen avain
2. sort: ohjelmistokehys ryhmittelee arvot avaimien perusteella
3. reduce: reduce funktio suoritetaan ryhmälle arvoja joilla on sama avain ja se tuottaa näistä listan lopullisia arvoja

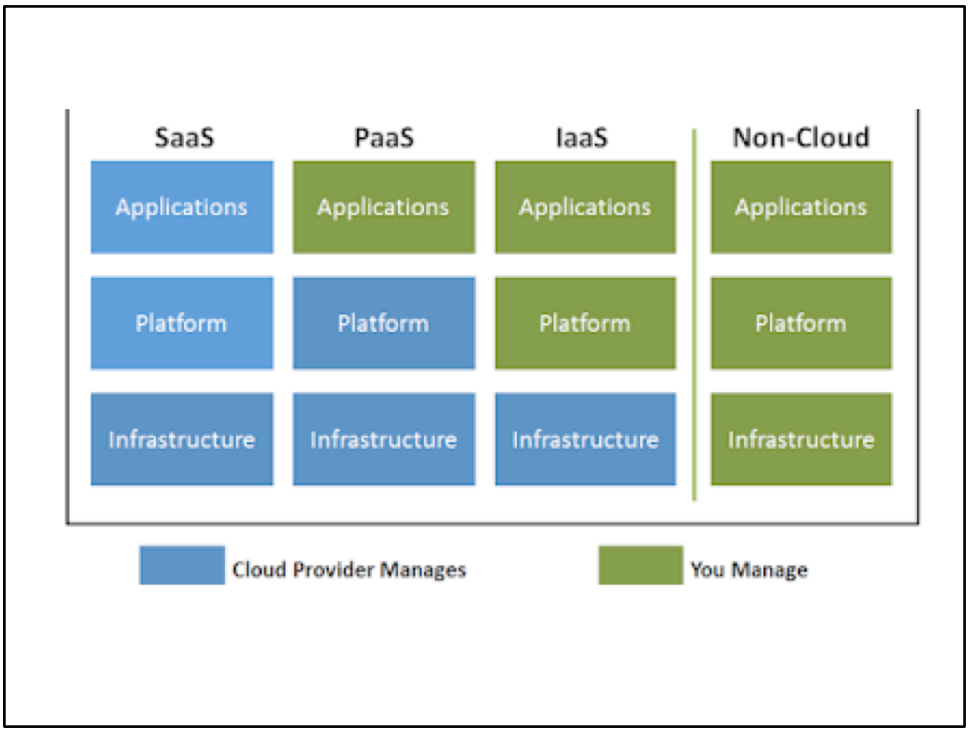
How SOA is related to
Design for failure,
Loose coupling,
Elasticity, and
Think Parallel

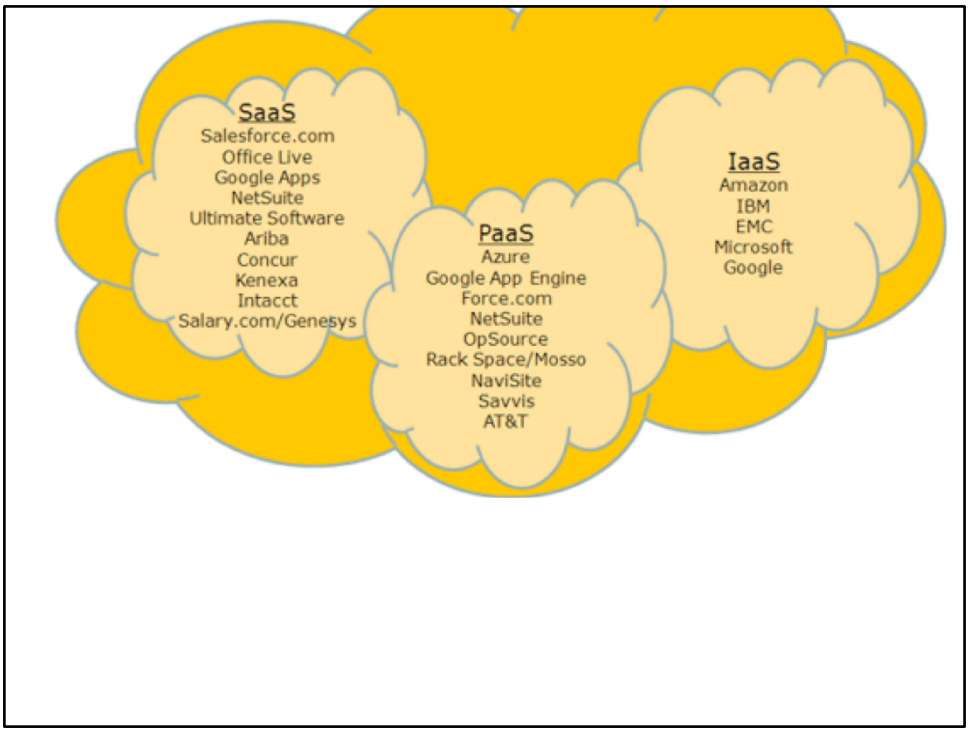


References

- [CloudArchitectures]
  - Cloud Application Architectures: Building Applications and Infrastructure in the Cloud, George Reese, O'Reilly Media, 2009.
- [CloudBestPractices]
  - Architecting for the Cloud: Best Practices: http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
- [Hadoop]
  - Open Source MapReduce, Apache Hadoop: http://hadoop.apache.org/
- [MapReduce]
  - MapReduce: Simplified Data Processing on Large Clusters: http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/mapreduce-osdi04.pdf

Anything as a Service (XaaS)


SaaS
PaaS
IaaS
Non-Cloud
Applications
Applications
Applications
Applications
Platform
Platform
Platform
Platform
Infrastructure
Infrastructure
Infrastructure
Infrastructure
Cloud Provider Manages
You Manage


SaaS
Salesforce.com
Office Live
Google Apps
NetSuite
Ultimate Software
Ariba
Concur
Kenexa
Intacct
Salary.com/Genesys
PaaS
Azure
Google App Engine
Force.com
NetSuite
OpSource
Rack Space/Mosso
NaviSite
Savvis
AT&T
IaaS
Amazon
IBM
EMC
Microsoft
Google

Anything as a Service (XaaS)

- Besides the more traditional IaaS/PaaS/SaaS, various other ”as a Service” concepts have been introduced
- Eg.
  - Backend as a Service
  - Database as a Service
  - Humans as a Service
    - I.e. crowdsourcing
  - IDE as a Service
  - ...

Backend as a Service (BaaS)

- A lot of mobile and web applications need similar features
  - User management
  - Data storage
  - Integration with 3rd party applications
  - Push notifications
  - Versioning, analytics
  - …
- BaaS providers offer these (= the backend) as a service
  - To be used by application developers
    - Saving them the problem of backend implementation (on PaaS or IaaS)
  - SDKs for various environments
    - Web, Android, iOS, Windows Phone, …
    - + often a REST API
  - Eg. Parse, FireBase, Kinvey, …
- BaaS does not provide the application runtime environment

Eri BaaS-tarjoajiin voi tutustua vaikka TTY:llä järjestetyn BaaS-seminaarin esityksistä:
http://www.cs.tut.fi/~taivalsa/kurssit/BaaS2013/

Database as a Service (DBaaS)

- A database on the cloud
- SQL or NoSQL
- Often offered as a part of PaaS
  - Eg. Google App Engine Datastore, Amazon DynamoDB, Microsoft SQL Azure
- Or independent
  - Eg, ScaleDB, BitCan

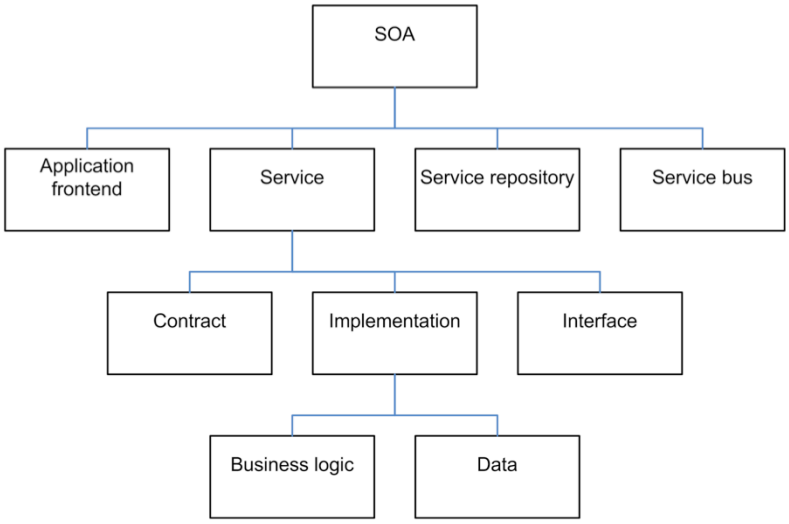
Humans as a Service

- Subdividing work into small tasks to be performed by large group of people
- AKA crowdsourcing
- Either
  - For money
    - Eg. Amazon Mechanical Turk
  - Voluntarily
    - Eg. YLE: merkitse vaarallinen risteys
  - By some other incentive
    - Eg. reCAPTCHA

IDE as a Service

- Integrated development environment as a Service
- Besides offering the application runtime environment (PaaS), offer a full set of tools for developing the applications on the web
  - Code editor
  - Debugger
  - Version control
  - Collaboration
  - …
- Eg. Cloud9, Codebox

SOA?


SOA
Application frontend
Service
Service repository
Service bus
Contract
Implementation
Interface
Business logic
Data

XaaS vs SOA?

- For example, can we se DBaaS as Service in SOA?
- SaaS = SOA?
  - Differences?
  - Similarities?

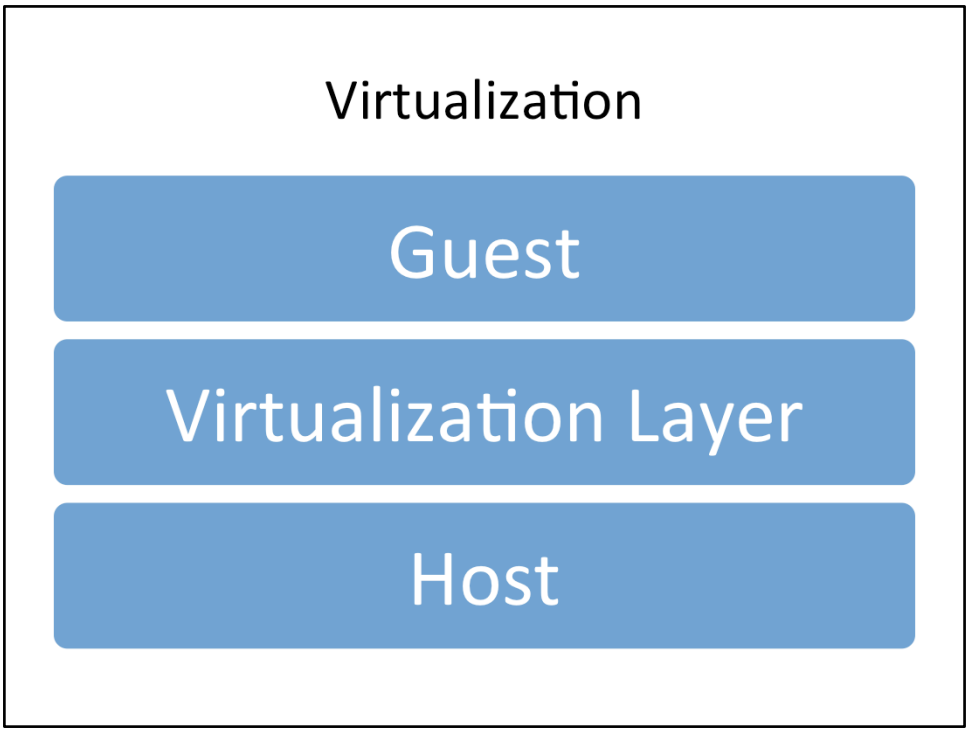
Virtualization

Virtualization

- Virtualization abstracts some elements of computing
  - Hardware, storage, networking, runtime environment, …
- Allows creating *virtual* computing environments
  - Not directly tied to the underlying hardware
- Virtualization is essential in the implementation of IaaS and PaaS providers

Virtualisoinnin avulla voidaan purkaa suora linkki suoritettavan sovelluksen (tai käyttöjärjestelmän tms.) ja sitä suorittavan laitteiston välillä. Näin saavutetaan joustavuutta laitteiston käytössä. Voidaan esimerkiksi suorittaa useampaa käyttöjärjestelmää samaan aikaan yhdessä tietokoneessa.

Virtuaalisointitekniikat ovat erittäin tärkeitä pilvipalveluiden (erityisesti IaaS) toteutuksessa. IaaS-tarjoajat eivät anna käyttäjilleen pääsyä suoraan laitteistoonsa, vaan jonkinlaiseen virtuaaliympäristöön. Tästä on useita hyötyjä, joihin tutustutaan tarkemmin myöhemmillä kalvoilla.



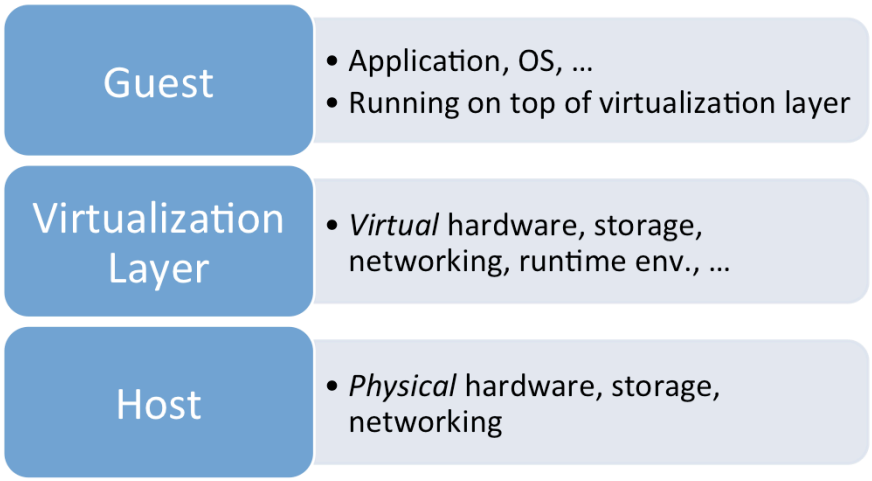
Virtualisointiympäristössä on kolme pääosaa: isäntä (host), virtualisointikerros ja vieras (guest).

Isäntä pitää sisällään fyysisen laitteiston sekä yleensä myös sen päällä ajettavan käyttöjärjestelmän.

Virtualisointikerros tarjoaa virtuaaliympäristön vieraan (guest) käyttöön. Se tarjoaa rajapinnan, johon tulevat kutsut se välittää isännälle muunnettuaan ne isännän ymmärtämään muotoon. Virtualisointikerros voi tarjota myös muita ominaisuuksia liittyen mm. turvallisuuteen.

Vieras (guest) on sovellus, käyttöjärjestelmä tms. jota ajetaan virtuaalikerroksen päällä. Sen sijaan, että se kommunikoisi suoraan isännän kanssa (kuten normaalitilanteessa ilman virtualisointia), kommunikoi se virtualisointikerroksen kanssa. Vieraan ei tosin tarvitse tietää olevansa ajossa virtuaaliympäristössä.

Virtualization



Benefits of virtualization

- Increased security
  - Host can control and filter actions of guest
- Increased flexibility
  - Sharing
  - Aggregation
- Portability
  - The guest can be moved, duplicated, suspended, etc.
- Allows efficient utilization of resources

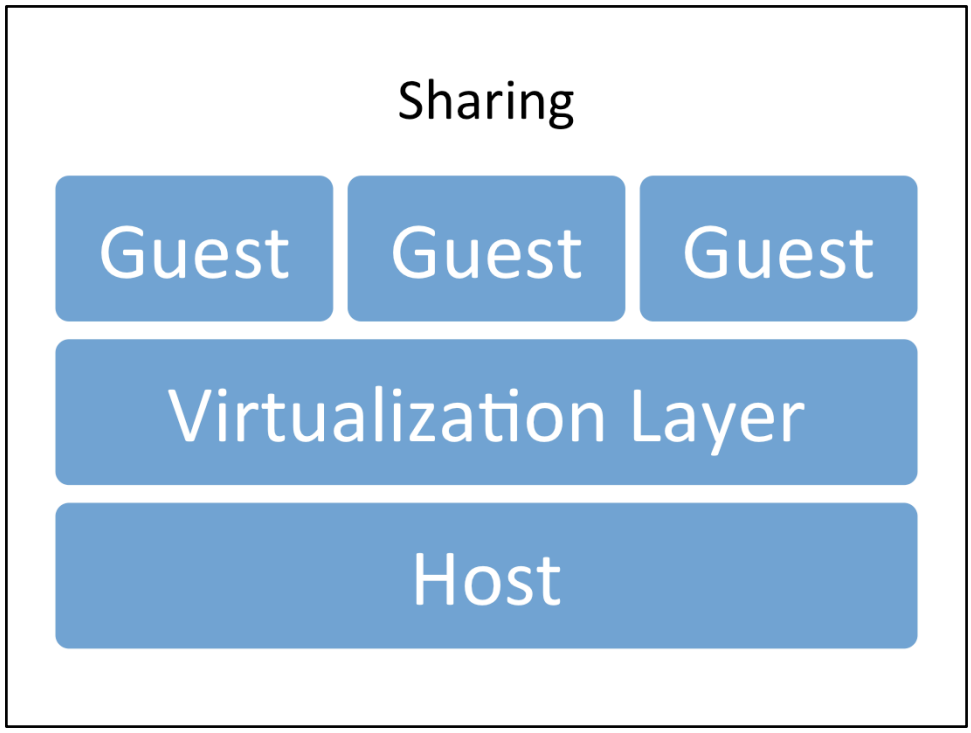
Virtualisoinnilla on monia hyviä puolia.

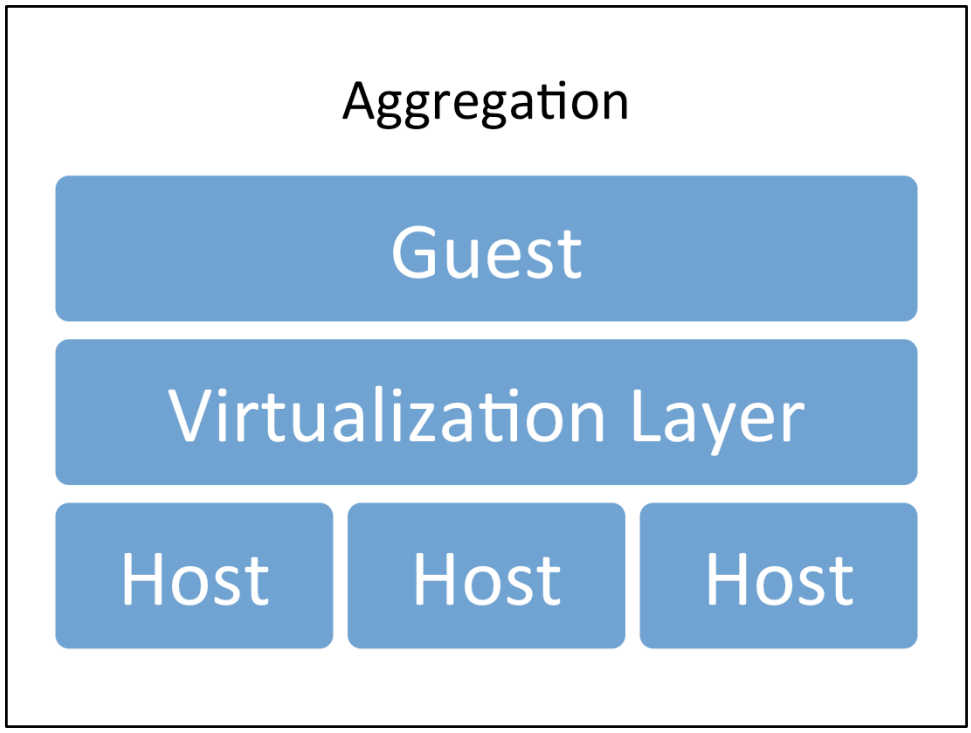
Ensinnäkin sillä saadaan aikaan turvallinen ajoympäristö, jossa virtualisointikerros voi hallita sitä, mitä vieras saa tehdä. Koska kaikki käskyt kulkevat virtualisointikerroksen läpi, voi se esim. rajoittaa tiedostojärjestelmän tai verkon käyttöä.

Toisekseen virtualisoinnilla saadaan paremmin hyödynnettyä resursseja. Yhdessä fyysisessä tietokoneessa voi olla ajossa useampia käyttöjärjestelmiä samaan aikaan. Tai päinvastoin: useampi fyysinen kone voi toteuttaa virtuaaliympäristön, joka näkyy yhtenä suurena tietokoneena.

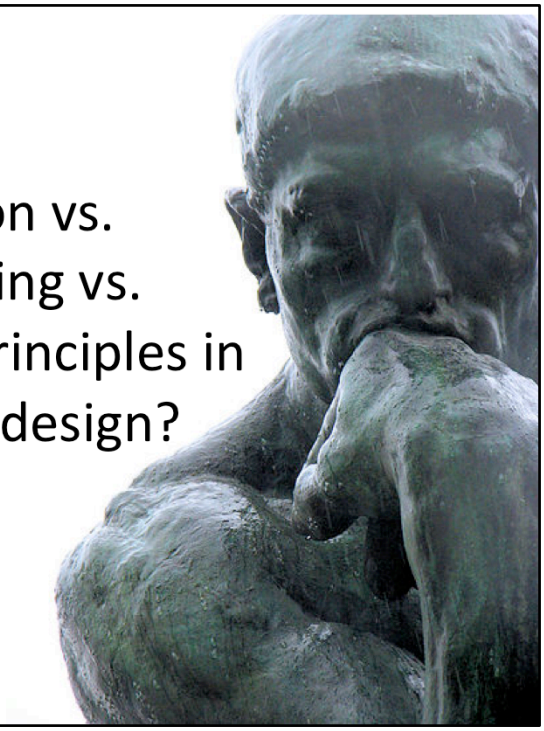
Näin, jos IaaS-tarjoaja haluaa tarjota ympäristöä, jossa on 2 suoritinydintä, 4 gigatavua muistia, 1 Mbit/s verkkoyhteys ja 100 GB levytilaa, ei tarjoajan tarvitse hankkia tietokonetta täsmälleen näillä spekseillä vaan voi luoda tätä vastaavan virtuaaliympäristön.

Virtualisoinnissa on myös se hyvä puoli, että vierassovelluksia voidaan siirrellä fyysiseltä laitteelta toiselle ilman että vieras välttämättä huomaa mitään.


Sharing
Guest
Guest
Guest
Virtualization Layer
Host


Aggregation
Guest
Virtualization Layer
Host
Host
Host

Aggregation vs.
load balancing vs.
applying SOA principles in
architectural design?



Disadvantages of virtualization

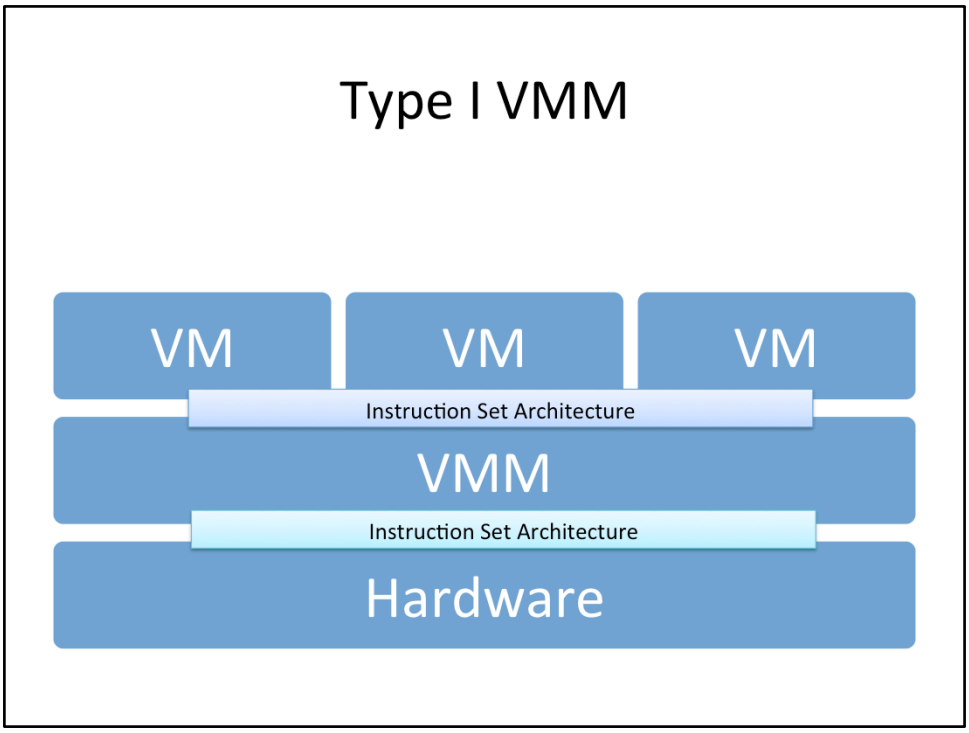
- Performance degradation
  - Virtualization layer brings some performance penalty
  - However, with current technology, the penalty is quite small
- Lack of features
  - Virtualization environment may not provide all the features available in the host

Virtualization techniques

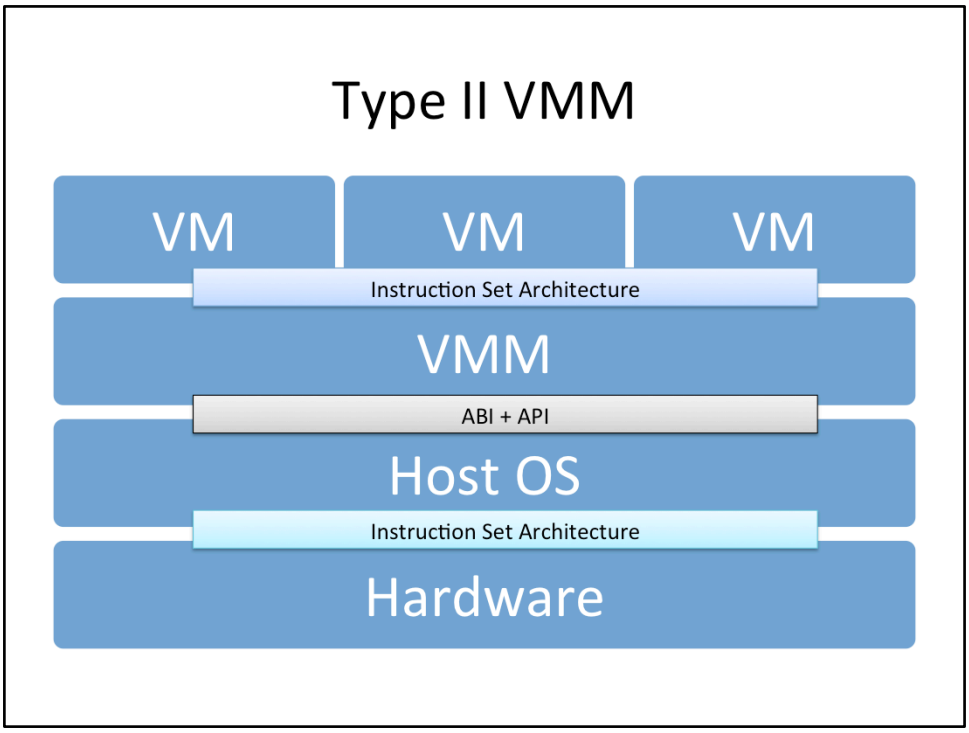
- Execution virtualization
  - Hardware-level virtualization
  - Programming language-level virtualization
  - Application-level virtualization
- Storage virtualization
- Network virtualization

Hardware-level virtualization

- Virtual Machine Monitor (VMM) creates a virtual hardware environment for guest operating system
  - Type I and Type II VMMs
- The guest OS interacts via an Instruction Set Architecture (ISA) provided by the virtualization layer



Instruction Set Architecture (ISA) on laitteiston ja ohjelmiston välinen rajapinta.



Instruction Set Architecture (ISA) on laitteiston ja ohjelmiston välinen rajapinta.

ABI (Application Binary Interface) ja API (Application Programming Interface) ovat käyttöjärjestelmän sovelluksille tarjoamia rajapintoja.

Hardware virtualization techniques

- Hardware-assisted virtualization
  - Hardware provides support for virtualization
  - Reduces performance penalty
- Full virtualization
  - Provides complete emulation of hardware
    - As opposed to partial virtualization
  - Advantage: full isolation
  - Usually achieved by a combination of software and hardware-assisted virtualization
- Paravirtualization
  - Non-transparent: requires modification to the guest operating system

Programming language-level virtualization

- Some programming languages run on top of a *high-level virtual machine* (HLVM)
  - Eg. Java, Python
- Source code is compiled into intermediate presentation called *bytecode*
- Bytecode is executed on a HLVM
  - The HLVM converts bytecode instructions into native instructions
- Same bytecode can be run in multiple environments
  - Only the HLVM implementation is environment-specific
- Useful in PaaS implementations

Virtualisoinnissa ei tarvitse välttämättä emuloida laitteistoa, myös korkeamman tason virtualisointitekniikoita on olemassa. Yksi tällainen on ohjelmointikielitason virtualisointi, jota jotkut ohjelmointikielet (mm. Java ja Python) hyödyntävät. Sen sijaan, että lähdekoodi käännettäisiin suoraan natiiveiksi konekäskyiksi, käännetäänkin se ”tavukoodiksi” (bytecode), jonka suorittamiseen tarvitaan korkean tason virtuaalikone (HLVM). Tavukoodi sisältää käskyjä, jotka HLVM:n on kohtuullisen suoraviivaista muuntaa ajonaikaisesti (tai ns. JIT-käännöksellä) natiiveiksi konekäskyiksi.

PaaS-tarjoajat käyttävät usein tämän tason virtualisointia. PaaS:n käyttäjä voi kirjoittaa ohjelman esimerkiksi Pythonilla ja lähettää tavukoodin PaaS-tarjoajan ajettavaksi. PaaS-tarjoajalla on oma virtuaalikoneensa tavukoodin ajamiseen.

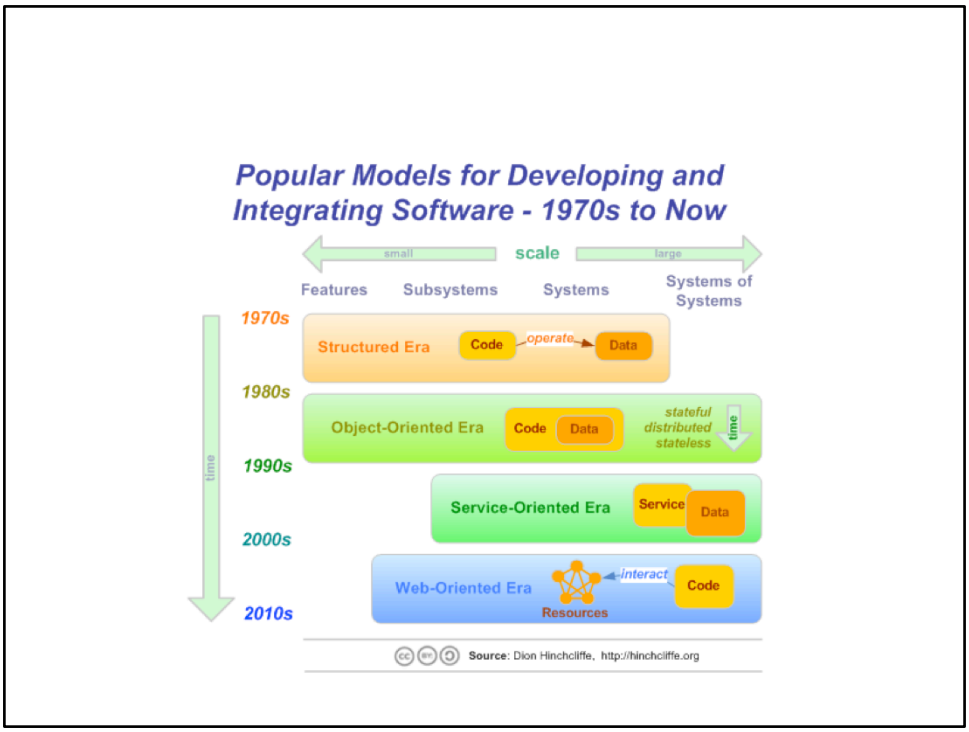
Application-level virtualization

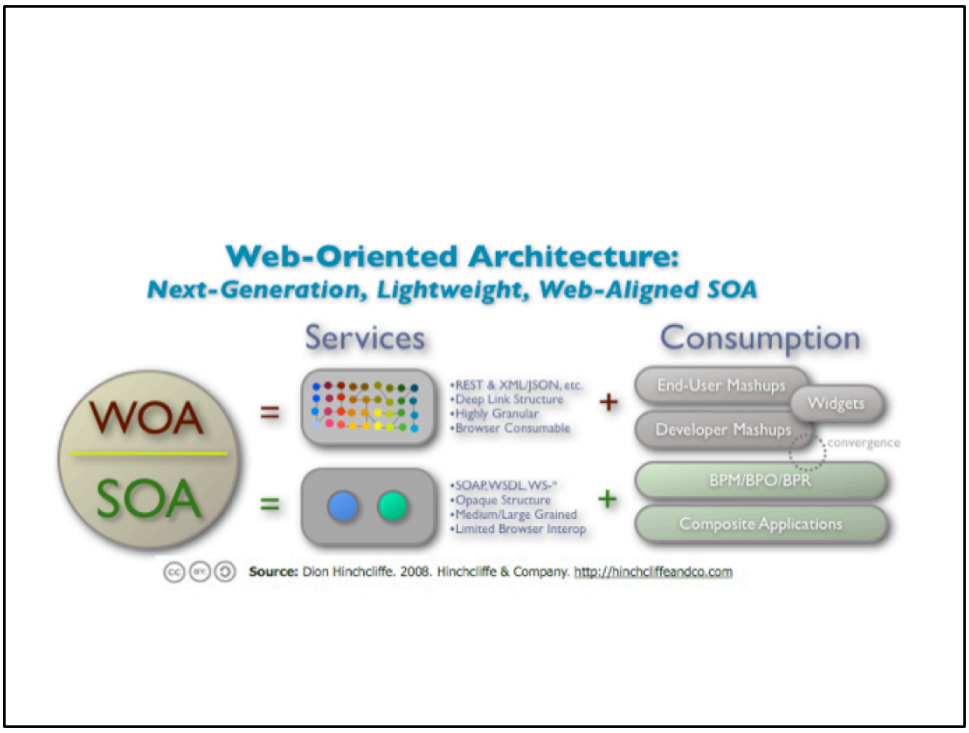
- Allows applications to be run in environments that don't directly support all the features of required by the application
- Eg. Wine – an emulator for running Windows applications in Linux
- Not that relevant in cloud computing

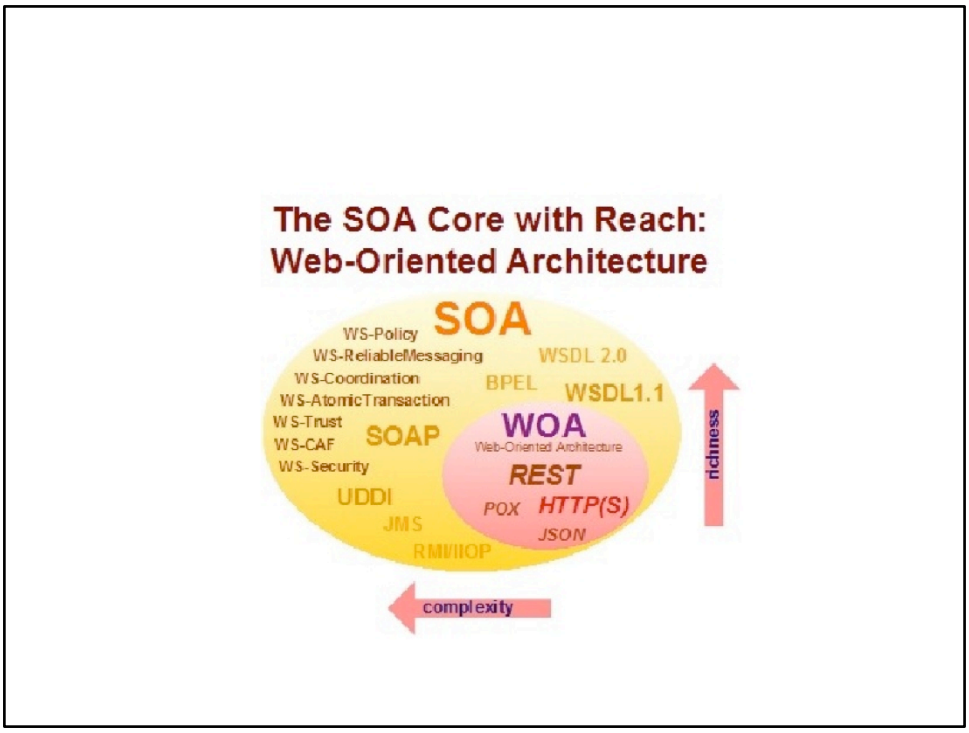
Virtualization and cloud computing

- Hardware virtualization is important in implementing IaaS providers
  - Efficient use of resources
  - More flexibility
- Programming language virtualization is important in implementing PaaS providers
- Virtualization allows user applications to be *sandboxed*
  - Run multiple applications in complete isolation even on the same physical machine

SOA in Context of Web


Popular Models for Developing and Integrating Software - 1970s to Now
small
scale
large
Features
Subsystems
Systems
Systems of Systems
1970s
1980s
1990s
2000s
2010s
time
Structured Era
Code
operate
Data
Object-Oriented Era
Code
Data
stateful
distributed
stateless
time
Service-Oriented Era
Service
Data
Web-Oriented Era
interact
Code
Resources
Source: Dion Hinchcliffe, http://hinchcliffe.org


Web-Oriented Architecture:
Next-Generation, Lightweight, Web-Aligned SOA
Services
Consumption
WOA
SOA
=
=
•REST & XML/JSON, etc.
•Deep Link Structure
•Highly Granular
•Browser Consumable
•SOAP, WSDL, WS-*
•Opaque Structure
•Medium/Large Grained
•Limited Browser Interop
+
+
End-User Mashups
Widgets
Developer Mashups
convergence
BPM/BPO/BPR
Composite Applications
Source: Dion Hinchcliffe. 2008. Hinchcliffe & Company. http://hinchcliffeandco.com


The SOA Core with Reach:
Web-Oriented Architecture
SOA
WS-Policy
WS-ReliableMessaging
WS-Coordination
WS-AtomicTransaction
WS-Trust
WS-CAF
WS-Security
SOAP
UDDI
JMS
RMI/IIOP
BPEL
WSDL 2.0
WSDL1.1
WOA
Web-Oriented Architecture
REST
POX
HTTP(S)
JSON
richness
complexity

Web Services

WS*

SOA and Web services

- *Web services* offer one way to implement SOA
- Web services = WSDL, SOAP, and many other standards
  - Related to interoperability, orchestration, reliability, security, …, …
  - http://en.wikipedia.org/wiki/List_of_web_service_specifications
  - This collection of standards is sometimes also called *WS-**

44

Web-palvelukonsepti tarjoaa yhden tavan toteuttaa SOA. Tämä tapa perustuu ”Web-palvelustandardien” käyttöön: palvelut kuvataan WSDL-kielen avulla ja kommunikointi toteutetaan SOAPin avulla. Näihin kieliin palaamme myöhemmin. On kuitenkin todettava, että ”Web-palvelustandardeihin” liittyy myös paljon ongelmia. Toisaalta nämä standardit kehittyvät jatkuvasti ja toisaalta esimerkiksi palveluiden koordinointiin liittyen ei ”standardeista” olla aivan vielä päästy vastaavaan yhteisymmärrykseen. Sanaan ”standardi” kannattaakin yleisesti suhtautua varauksellisesti Web-palveluihin liittyvistä teknologioista puhuttaessa: kyseessä saattaa olla lähinnä ehdotus eikä varsinaisesti standardi. Ja toisaalta standardista käytetään usein rinnakkain useita eri versioita.

SOA and Web services

- SOA and Web services aim at automatically solving problems related to interoperability, configuration etc.
  - an agreement on the standards partly enables this
    - challenges: a whole range of ”standards”
  - possible problems:
    - genericity/flexibility vs. automation
- A lot of tool support for building Web services and client applications
  - varying features
  - support for Web service languages and recommendations, and their different versions varies

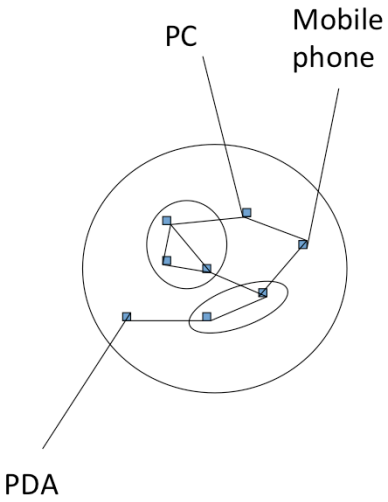
45

SOA ja Web-palvelut pyrkivät periaatteessa ratkaisemaan yhteentoimivuuteen, konfigurointiin jne. liittyvät ongelmat automaattisesti ja vieläpä ajonaikana. Esimerkiksi mikäli kutsuttavaa palvelua ei ole saatavilla tai yhteentoimivuusongelmia ilmenee, tulisi kutsuvan palvelun kyetä joko ratkomaan yhteentoimivuusongelmat tai korvaamaan ko. palvelu toisella vastaavan toiminnallisuuden omaavalla palvelulla. Tämä on kuitenkin vielä useissa tapauksissa kaukana käytännön toteutuksista siitä huolimatta, että useista standardeista onkin päästy jo yksimielisyyteen. Automaattisuuteen liittyy myös omat ongelmansa. Esimerkiksi muutosten tekeminen manuaalisesti voi olla vähintäänkin haasteellista.

Web-palveluiden ja asiakassovellusten tekemiseksi ja osin generoimiseksi on olemassa runsaasti työkalutukea. Esimerkiksi palvelun rajapintakuvaus (WSDL) voidaan generoida automaattisesti palvelun rajapinnan (esim. Java) perusteella. Nämä työkalut kuitenkin poikkeavat toisistaan sekä tarjottujen ominaisuuksien että toteutustapojen suhteen. Esimerkiksi samasta rajapinnasta (esim. Java) eri työkalut generoivat erilaisia WSDL-kuvauksia. Lisäksi eri työkalujen tarjoamatuki eri Web-palvelukielille, niiden eri versiolle ja eri suosituksille vaihtelee.

A vision of Web services

- Software will be assembled from a web of services
- Building applications just-in-time
- Discovering and coordinating services on the Web dynamically
- From distributed systems to de-centralized applications



46

Web-palveluiden alkuperäisen ja tavoiteltavan vision mukaisesti palveluja tulisi voida etsiä ja niitä tulisi voida käyttää dynaamisesti. Tietyn palvelun käyttöön sitominen tulisi siis olla ajonaikainen (dynaaminen) toimenpide, ei staattinen. Sovelluksia tulisi voida muodostaa olemassa olevia palveluita hyödyntäen aina kulloisenkin tarpeen mukaisesti. Nämä yhdessä edellyttävät lisäksi sen, että palveluiden koordinoinnin tulisi myös olla dynaamista.

Ehkäpä yksi oleellisimmista näkökulmista on, että Web-paveluiden myötä siirryttäisiin hajautetuista järjestelmistä ei-keskitettyihin järjestelmiin. Tämä merkitsisi sitä, että tarjolla olisi verkko erilaisia (hallinnan näkökulmasta itsenäisiä ja tasavertaisia) palveluita, joita mikä tahansa sovellus voi käyttää ja mahdollisesti yhdistellä uusiksi palveluiksi.

Nämä edellä esitetyt visiot ovat luonnollisesti vielä kaukana todellisuudesta. Esimerkiksi tietoturvakysymykset ja käyttöoikeudet asettavat reunaehtoja ja vaatimukia, joita ei vielä yleisellä tasolla olla täysin ratkaistu. Yksittäisiä ratkaisuja ja ratkaisuehdotuksia on esitetty, mutta yhtenäistä periaatetta ja käytäntöjä ei vielä ole. Näin ongelmiin palaamme myöhemmin.

Defining Web services

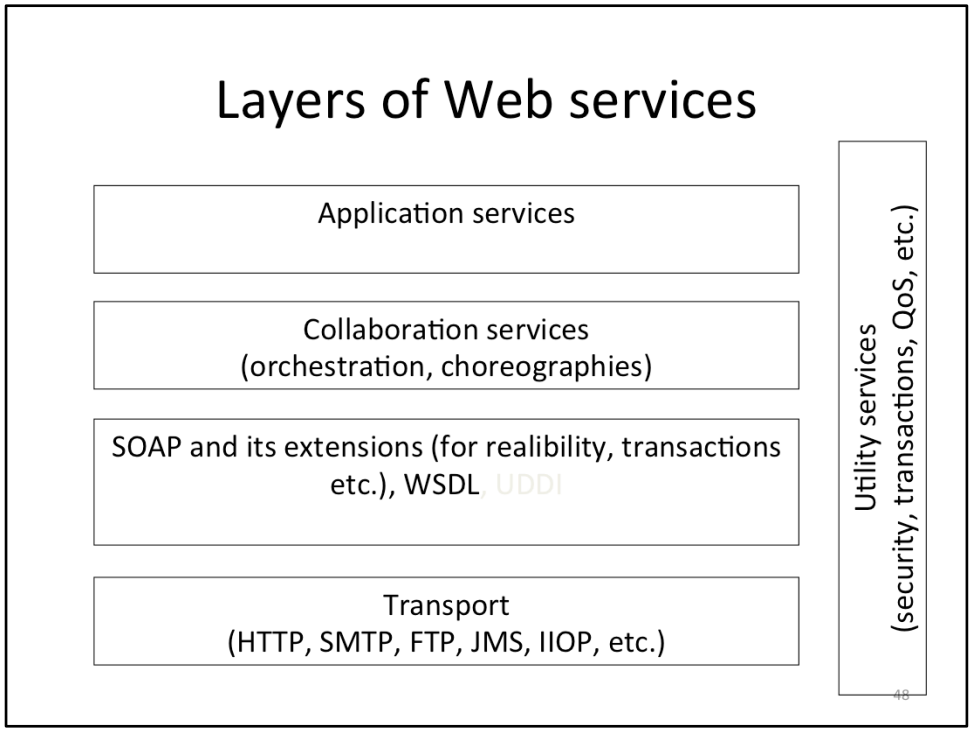
- W3C puts it: ”programmatic interfaces made available for application to application communication are referred to as Web services”
- ...or more precisely: ”A Web service is a collection of functions packaged as a single entity and published to the network for use by other applications”
  -> a hierarcy of Web services: more complicated ones are aggregated from simpler ones
- or perhaps:“XML applications mapped to programs, objects, databases, or business functions”

47

Web-palveluille on esitetty lukuisia määritelmiä. Yksinkertaisimmillaan niiden on sanottu olevan sovelluksia, joihin voidaan ottaa yhteys käyttäen standarditeknologioita (kuten XML ja HTTP). On myös sanottu, että Web-palvelu on käytännössä sama asia kuin SOAP-protokollan käyttö (tästä lisää myöhemmin). Nämä eivät kuitenkaan ole kovin hyviä määritelmiä, sillä ne ovat aivan liian laajoja. W3C puolestaan määrittelee Web-palvelun (vapaasti käännettynä) joukkona ohjelmiin liittyviä rajapintoina, jotka ovat käytettävissä sovellusten välisessä kommunikoinnissa. Tämäkin määritelmä on melko yleinen.

Hieman tarkemmin määriteltynä Web-palvelun on sanottu olevan joukko funktioita, jotka on **pakattu** yhdeksi kokonaisuudeksi ja **julkaistu** verkossa muiden sovellusten käytettäväksi. Tämä määritelmä on jo selvästi parempi. Oleellista tässä on se, että näitä tarjottuja funktioita voidaan yhdistellä ja pakata uusiksi palveluiksi. Se myös implikoi Web-palveluille hyvin oleellisen piirteen: Web-palveluhierarkian. Toisin sanoen yksinkertaisimmista palveluista voidaan koostaa monimutkaisempia palveluita. Toinen oleellinen asia tässä määritelmässä on palveluiden julkaiseminen. Jotta Web-palvelu olisi aidosti vapaasti etsittävissä ja käytettävissä, edellyttää se, että palveluiden käyttäjät voivat etsiä palveluita jostain yleisesti tunnetusta ”markkinapaikasta”.

Viimeinen määritelmä kuvaa Web-palvelut XML-sovelluksina, jotka on sidottu joihinkin ohjelmiin, tietokantoihin tai liiketoimintafunktioihin. Web-palveluissa käytetään XML-pohjaisia kieliä, mutta palveluiden kutsuminen XML-sovelluksiksi voi olla myös harhaanjohtavaa. Tässä määritelmässä oleellista on se, että itse palvelun toiminnallisuus voi mitä vain ja se on voitu toteuttaa millä tahansa halutulla tavalla. Palvelun käyttöä varten tulee kuitenkin toteuttaa käytettäviä standardeja ymmärtävä ja käsittelevä interaktiota tukeva kerros. Web-palvelukonseptin voidaan ajatella olevan myös mekanismi back-end systeemien paketoimiseksi (wrapping). Tällaisia back-end systeemejä voivat olla vaikkapa tietokanta, legacy-systeemi jne.



Kalvolla esitetyn kuvan alimpana kerroksena ovat viestinvälitysprotokollat. Vaikka Web-palveluita usein sanotaan käyttävän HTTP:tä – ja näin käytännössä hyvin usein onkin – itse Web-palvelukonseptia ei ole sidottu tiettyyn viestinvälitysprotokollaan. Yhtä hyvin käytössä voisi olla esimerkiksi SMTP tai FTP. Kuvan seuraavan kerroksen muodostavat Web-palvelustandardit SOAP, WSDL ja UDDI. Näistä ensimmäistä käytetään kommunikointiin sovellusten kesken. WSDL-kieltä puolestaan käytetään kuvaamaan tarjottu Web-palvelu (tarjotut funktiot ja yhteydenottotapa). UDDI on yksi tapa toteuttaa palveluiden ”markkinapaikka” (rekisteri), mutta muitakin vaihtoehtoja on olemassa. Palvelun ”mainostaminen markkinapaikassa” ei palvelun pystytyksen kannalta toki ole välttämätöntä. Mikäli asiakaskunta tietää miten palveluun saa yhteyden ja miten sitä voidaan kutsua, on se tarpeetonta. Näin on usein esimerkiksi kun Web-palvelukonseptia käytetään rajoitetussa ympäristössä kuten yrityksessä. Tällöin mikäli käyttö tapahtuu palomuurien sisäpuolella, ei kommunikoinnin turvallisuuden takaamiseen tarvitse kiinnittää välttämättä huomiota.

Palveluverkosto (esimerkiksi toisiaan käyttävät palvelut) edellyttää palvelujen koordinointia. Koordinointipalvelut on esitetty kuvassa kolmantena kerroksena (collaboration services). Esimerkiksi sekvenssi yksittäisten palvelujen suorittamista operaatiosta voidaan haluta koostaa yhdeksi liiketoimintatransaktioksi. Palveluiden yhdistämiseen käytetään nk. *orkestointi-* ja *koreografiakieliä.* Näistä orkestrointi tarkoittaa palveluiden yhdistämistä yhdestä tietystä liiketoimintaprosessia suorittavasta näkökulmasta, kun taas palvelukoreografia sallii useita samanaikaisia ja tasavertaisia näkymiä liiketoimintaprosessiin. Näihin palataan vielä myöhemmin. Lopuksi kuvan ylimpänä kerroksena ovat itse Web-palvelut.

Kuvan kaikkia eri kerroksia koskee ja tukee joukko muita hyödyllisiä palveluja (utility services). Jotkin käytetyt ratkaisut koskevat kaikkia kerroksia ja voivat siten vaikuttaa käytettäviin formaatteihin, protokolliin ja API-määrittelyihin tai vaikkapa vaikuttaa niiden valintaan. Turvallisuusaspektit ovat esimerkiksi hyvin

Usage examples of Web services

- Software migration and platform integration
  - wrapping legacy systems to act like a Web service
  - ”agreement on disagreement”
- Agile business processes
  - flexible integration
  - business collaborations and agreements
  - e.g., streamlining business operations, invoicing, shipping operations etc.
- Service registries and searching
  - searching goods or services according to given requirements (e.g., best price)
  - coordinating travel tickets or hotel reservations for a given date
- RPCs
  - location transparency!
- And many, many more…

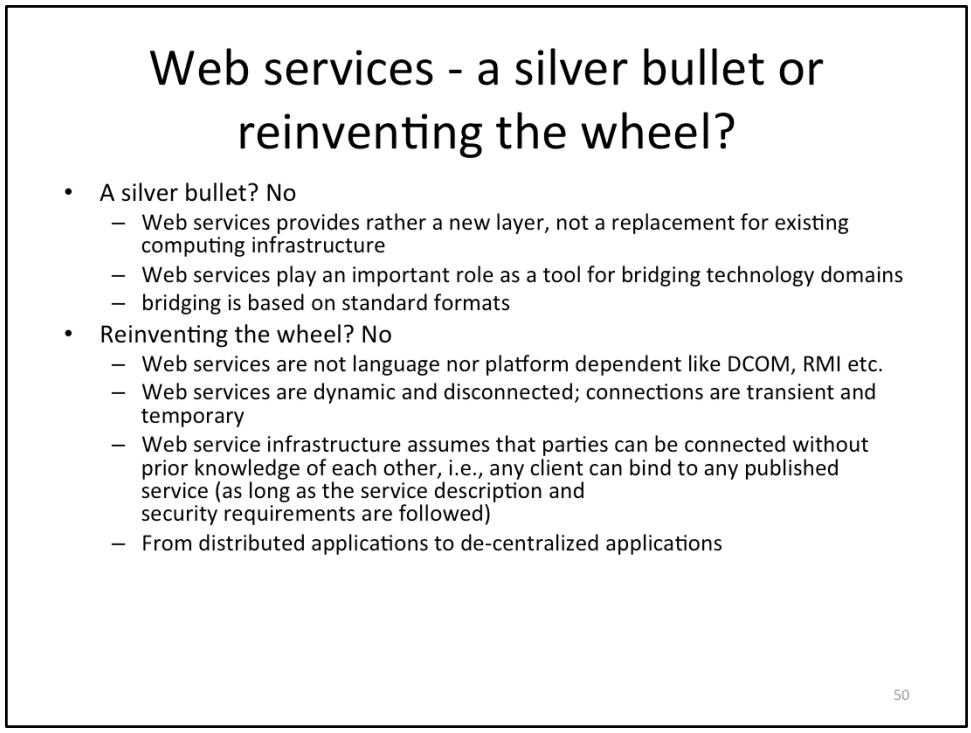
49

Web-palvelut voivat käytännössä olla mitä tahansa. Paketoimalla vanha legacy-systeemi Web-palveluksi sovitaan käytännössä erimielisyydestä: legacy-systeemien logiikka tai toteutusta ei tarvitse muuttaa ja ne voivat olla hyvinkin eri tavoin toteutettuja.

Erilaiset ketterät liiketomintaprosessit ovat myös potentiaalisia Web-palvelukonseptin käyttökohteita. Esimerkiksi liiketoimintaoperaatiot (laskutus, tilaukset jne.) voidaan tarjota Web-palveluina. Liiketoimintasopimuksien ja liiketoimintaprosessien määritykset ovat erityisesti painotettuja RosettaNet ja ebXML –kosepteissa, jotka ovat tietyssä mielessä vaihtoehtoisia näkökulmia Web-palveluihin. Näihin palataan myöhemmin.

Yksi palvelun muoto voi olla vaikkapa muiden palveluiden etsintään tarkoitettu rekisteri. Rekisteristä voidaan etsiä palveluita annetuin kriteerein (esim. halvin matka). Palvelu voi myös hyödyntää muita palveluita. Esimerkiksi matkanjärjestämispalvelu voi käyttää hyväkseen sekä palvelua, jonka avulla voidaan etsiä halvimmat lennot kohteeseen annetulla aikavälillä, että palvelua, joka etsii sopivimman hotellin (annettujen kriteerien mukaisesti) matkakohteessa.

Web-palveluja käytetään – tietyssä mielessä vastoin sen alkuperäistä käyttötarkoitusta ja visiota – etäkutsujen toteuttamiseen. Se onkin tällä hetkellä yleisin käyttömuoto. Tämä voidaan tehdä siten, että kutsun muoto on kutsuttavan ohjelman sijainnista riippumaton eli se ei siis näy kutsun muodosta (location transparency). Etäkutsujen toteuttamiseen on kuitenkin jo olemassa useita eri menetelmiä.



Web-palvelu ei tarjoa mitään mullistavan uutta. Web-palvelua voidaan ajatella ylimääräisenä kerroksena, joka mahdollistaa sovellusten välisen interaktion. Se ei korvaa eikä sen ole tarkoitus korvata olemassa olevia tekniikoita (esim. hajautustekniikat) eikä olemassa olevia ohjelmistoja. Koska kyseessä on kevyt XML-pohjainen integrointi käyttäen yleisesti hyväksyttyjä standardeja, antaa Web-palvelukonsepti mahdollisuuden silloittaa eri teknologioita. Esimerkiksi palvelu voi olla toteutettu .NET ympäristössä kun taas sen asiakas voi J2EE-toteutus.

Web-palvelu ei kuitenkaan ole myöskään pyörän uudelleen keksimistä. Web-palveluissa ei ole kyse niinkään hajautusteknologiasta (kuten CORBA, DCOM, RMI) vaan siinä pyritään ei-keskitettyyn järjestelmään (ainakin periaatteessa). Lisäksi Web-palvelut eivät ole ohjelmointikieli- tai alustariippuvaisia kuten esimerkiksi RMI ja DCOM. Edelleen voidaan sanoa, että yhteydet on tarkoitettu transienteiksi: palveluihin kytkeydytään dynaamisesti aina tarpeen mukaan. Palvelun käyttäjän ei tarvitse tietää palvelun toteutuksen yksityiskohtia vaan sille riittää ainoastaan palvelun kuvaus. Web-palveluissa on myös oleellista se, että palvelun käyttäjän ja palvelun välillä ei tarvitse olla ennalta määriteltyä sopimusta (tämä on kuitenkin mahdollista ebXML-konseptissa) vaan periaatteessa mikä sovellus tahansa voi käyttää julkaistua palvelua, edellyttäen että palvelun kuvausta ja mahdollisia turvallisuusvaatimuksia noudatetaan. Tämä luonnollisesti pätee annetussa ympäristössä: mikäli kyseessä on esimerkiksi yrityksen sisäisessä verkossa tarjottu

Formats and protocols

- Web Service Description Language (WSDL)
  - a format for describing a Web service interface, data, message types, interactions patterns, and protocol mappings
- Simple Object Access Protocol (SOAP)
  - a message format for invoking a Web service
  - binds the WSDL description

51

Edellä esitetty Provider-Requestor-Broker -roolijako kuvaa erilaiset Web-palvelujen käsitteet ja konseptit. Se ei vielä ota kantaa siihen, mitkä ovat käytetyt formaatit ja APIt. Tosin myös siitä on päästy yleisesti yhteisymmärrykseen. Palvelujen kuvaukset ja viestinvälitys suoritetaan käyttäen WSDL ja SOAP -formaatteja (alunperin IBM:n ja Microsoftin ym. yhteisenä ehdotuksena). SOAP ja WSDL ovatkin jo vakiinnuttaneet asemansa. Palvelurekisteri voidaan toteuttaa esimerkiksi UDDI-rekisterinä, mutta sen rinnalle on tullut muitakin kandidaattiteknologioita pääosin UDDI:n tietynlaisen rajoittuneisuuden vuoksi.

WSDL ja SOAP -spesifikaatioit kehittyvät edelleen: uuden versiot seuraavat toisiaan ja niihin liittyviä muita suosituksia ja ehdotuksia (W3C) tehdään jatkuvasti. Eri versioiden käyttö luonnollisestikin johtaa yhteentoimivuusongelmiin. Lisäksi nämä spesifikaatiot sisältävät optionaalisia sääntöjä ja kaikki toteutukset eivät välttämättä toteuta samoja optionaalisia piirteitä. Tämä saattaa myös aiheuttaa yhteentoimivuusongelmia, vaikka eri osapuolet käyttäisivätkin samoja versioita. Web Service Interoperability (WS-I) yhteisö pyrkiikin pureutumaan näihin ongelmiin antamalla suosituksia näiden spesifikaatioiden käyttötavoista ja niiden eri versioiden yhteiskäytöstä. Vaikka tavoite onkin varsin hyvä, on tehtävä silti varsin haasteellinen. WS-I yhteisön roolia käsitellään tällä kurssilla myöhemmin.

What tomorrow?

WSDL and SOAP in Practice,
i.e., programming (finally)

52