

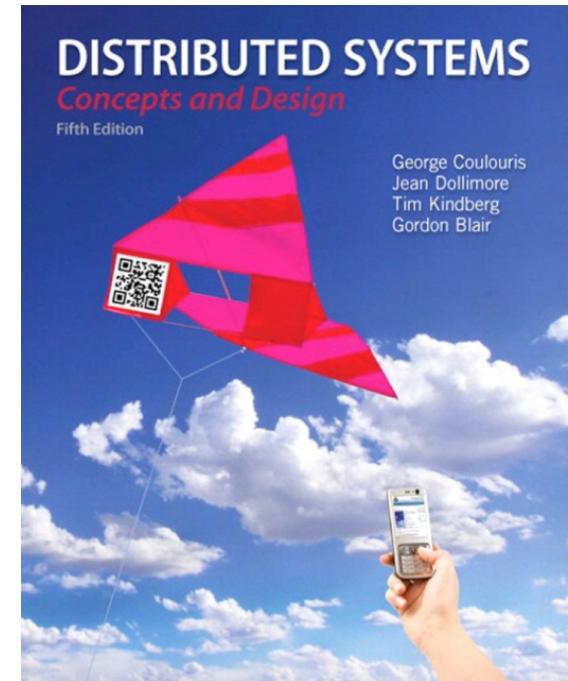
Sisteme de Programe pentru Retele de calculatoare

Introducere

Prima definitie

Un **sistem distribuit** este unul în care componente hardware și software localizate în calculatoare (diferite) conectate în rețea comunică și își coordonează acțiunile doar prin transfer de mesaje.

Felul în care sistemele distribuite sunt construite depinde de **cerințele aplicatiilor**



DISTRIBUTED SYSTEMS. Concepts and Design, Fifth Edition
George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair

WWW - Cerinte

Functie: Descarcare si prezentare informatii din pagini Web
Continut pagini - **texte** si **multimedia**

Cerinte

- **Interfata utilizator simpla si intuitiva** (usor de folosit)
- **Scalabilitate** - numar mare de **utilizatori** si de **pagini**
- **Disponibilitate** – functionare corecta chiar in prezenta **defectelor**
- **Performanta** rezonabila: timp de raspuns in limite acceptabile pentru perceptia umana (0.1 sec – 1 sec)
- **Confidentialitatea** si **integritatea** datelor

Soluția

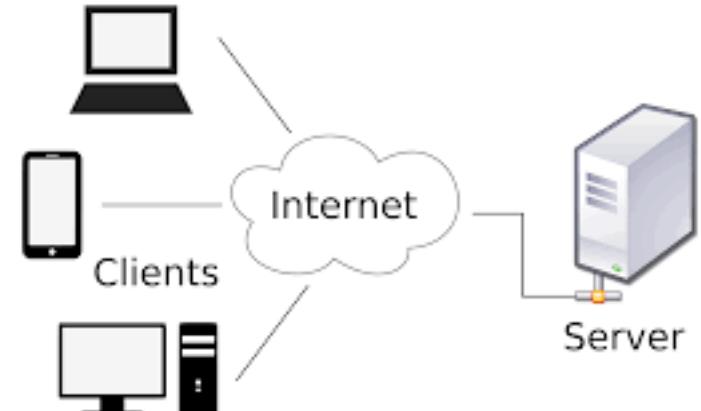
Sistem distribuit cu doua tipuri de componente

clientul (browser) - interfața cu utilizatorul,

serverul - gazduirea și accesarea resurselor.

Bazat pe arhitectura **REST** (Representational State Transfer)

- model introdus în 2000 de Roy Fielding în teza sa de doctorat
- arhitectura **client-server** concepută pentru sisteme distribuite hipermedia, în particular Web



Roy Fielding's dissertation, "Architectural Styles and the Design of Networkbased Software Architectures."

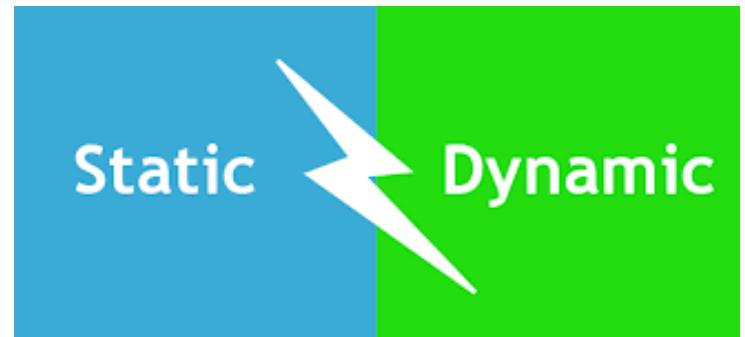
Continutul paginilor

Web-ul manevreaza **texte si continut multimedia**

- HTML (limbaj de marcaje) faciliteaza **afisarea** continutului la browser, **nu si intelegerea** continutului

Continutul furnizat de **server** poate fi

- **static** – preluat ca atare din fisiere
- **dinamic** – obtinut prin executia unui **program**
 - datele de **intrare** ale programului pot fi transmise de client prin **cererea** catre server

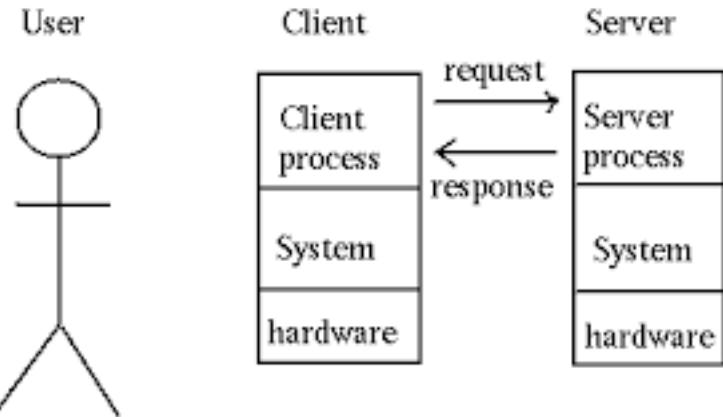


Modelul REST

Model

- sistem alcătuit dintr-o multime de **resurse** localizate prin URL-uri

Funcționare



- la o cerere a clientului (care include URL-ul unei resurse), sistemul returneaza o **reprezentare** a resursei
- la receptia reprezentarii, clientul **trece** intr-o noua **stare** a aplicatiei, in care are acces la alte URL-uri
- URL-urile interconecteaza reprezentarile resurselor, facilitand trecerea de la o stare la alta
- comunicarea client/server este reglementata de protocolul HTTP

Cereri si raspunsuri

Clientul poate cere descarcarea unei pagini, printr-o comanda GET care include URL-ul unei resurse



Serverul returneaza o **reprezentare** a resursei (pagina HTML)

Protocolul HTTP

HTTP este un protocol “stateless” ???

- serverul trateaza cererile **independent** una de alta
- dar **clientul** poate **gestiona** succesiunea cererilor trimise de el
- cererea si raspunsul **contin toate informatiile** necesare receptorului ca sa o poata intelege - sunt **auto-descriptive**

Avantaje ???

- **scalabilitatea** – serverul nu memoreaza starea si elibereaza rapid resursele folosite pentru o cerere
- creste **vizibilitatea** – un sistem de monitorizare gaseste intr-un singur mesaj toate informatiile

Dezavantaj ???

- multe informatii in fiecare mesaj → scad **performanta**

Conecțivitatea și interoperabilitatea

- *conectivitatea* globală ???
 - multe resurse Web răspândite pe scara largă mai multor servere în Internet
 - accesibile prin *orice browser*
- *interoperabilitatea* client / server obținuta prin ??? adoptare de standarde (TCP, HTTP ...)
 - ex. metode HTTP standard pentru acces la resurse
 - GET, POST, PUT, DELETE



Performanta

- *performanta* rezonabila: timp de raspuns 0.1 sec – 1 sec
- obtinut prin ???
 - replicarea serverelor
 - reduce timpului de **asteptare** la server
- distributia lor geografica
 - reduce timpul de transport al cererii si raspunsului
- probleme ???
 - consistentă replicilor pentru pagini dinamice



Disponibilitatea si securitatea

Disponibilitatea obtinuta prin: ???



CRITICAL WEB RE-DESIGN FAULTS

- replicare
 - defectele sunt **partiale** – nu toate replicile sunt defecte
- corectarea erorilor prin retransmitere
- model HTTP simplu, “fara stare” → eroarea nu se propaga de la o cerere la alta

Confidentialitatea & integritatea datelor obtinute prin ???

- mecanisme, algoritmi, protocoale de securitate
- *controlul accesului* la anumite resurse



Aplicatii bazate pe Web

- Folosesc Web-ul ca suport
 - browser: interfata utilizator uniforma si usor de intretinut
 - aplicatiile ruleaza “in spatele” serverului Web
- Aplicatia include componente (programe) distribuite, care ofera **servicii** bine definite
- Se folosesc doua ***arhitecturi*** consacrate
 - **REST**, REpresentational State Transfer
 - **SOA**, Service Oriented Architecture

Paradigma REST in aplicatii Web

- Servicii obtinute prin **operatii** (metode HTTP) asupra resurselor
 - PUT – creare resursa
 - DELETE – stergere resursa
 - GET – citire resursa
 - POST – modificare resursa
- cererile (**parametrii**) si **raspunsurile** pot fi reprezentate in **XML** (sau **HTML**) – **diferenta?**
 - XML descrie **date structurate**, procesabile de catre programe
 - HTML **nu suporta** comunicarea (semantică) intre programe
- **procesarea unei aceleiasi operatii difera** de la o resursa la alta, **serviciile** oferite fiind diverse
 - ex. amazon.com: comanda o carte, verifica starea comenzii curente etc.

Un exemplu – Depozit de mobila

Compania **Mobi** ofera un serviciu Web care permite clientilor

- sa obtina lista articolelor de mobilă
- sa obtina detalii despre un anumit articol
- sa faca o comanda

Serverul Web face disponibil un **URL** pentru **lista articolelor**

<http://www.Mobi.com/articole>

Pagina returnata la o invocare contine **lista articolelor**, intr-un format convenit cu clientul (de ex XML – clientul poate fi un program)

Pentru fiecare **produs** din lista, se specifica un **ID** si un **URL**, de ex

`<Articol id="00345" xlink:href="http://www.Mobi.com/articole/00345"/>`

Clientul poate folosi URL-ul pentru a obtine detalii despre acel articol

Descrierea articolului poate contine legaturi la alte detalii etc.

Pentru a primi o **comanda**, serverul face disponibil un **URL**

- Clientul **creeaza un document** conform cu o **schema** facuta cunoscuta de compania Mobi (de ex o schema XML – clientul poate fi un program)
- Clientul **trimitre** documentul cu o metoda POST
- Serviciul Web raspunde trimitand un nou **URL** asociat cu comanda clientului
- Comanda devine astfel o **resursa partajata** intre client si furnizor, in forma unui **serviciu Web**
- Folosind **URL**-ul comenzi, clientul o poate regasi date, edita etc.
- Obs.: **resursele sunt categorisite** in functie de metodele pe care clientul le poate apela: GET pentru citire si POST, PUT, DELETE pentru modificare

REST - Model orientat pe resurse

- paradigma **cerere / raspuns** si absenta **starii** usureaza gestiunea relatiilor intre servicii Web
 - trateaza cererile independent una de alta
 - raspunde usor cererilor venite de la clienti diferiti
 - un serviciu Web poate fi replicat pe mai multe servere
- dezavantaj - modelul este **orientat pe resurse (date)**
 - clientii primesc **intreaga stare** a resursei nu o parte a ei (ca in cazul invocarii unor operatii)
- totusi, 80% din cererile de servicii Web la amazon.com sunt facute prin interfata REST

Referinta: Roger L. Costello Building Web Services the REST Way
<http://www.xfront.com/REST-Web-Services.html>

Gasiti aici descrierile XML ale continutului comenziilor si raspunsurilor din exemplul dat

SOA

- Un **Serviciu** software este construit ca o unitate de program de sine statatoare
- Fiecare serviciu este o colectie de **operatii** ce pot fi invocate, datele de **intrare** si **rezultatele** diferind de la o operatie la alta
- Serviciul poate comunica cu alte servicii software fara o cunoastere **prealabila** a acestora – **cum e posibil?**
 - Prin **mecanisme** de publicare, subsciere, notificare, descoperire (publish - subscribe - notify / discovery)
- Paradigma serviciilor software este adoptata in **arhitecturi** si **tehnologii** diferite
 - RPC, Java RMI, CORBA, DCOM
 - SOAP

e-Servicii

- Dezvoltate in mediul privat **e-business** si **e-commerce** si public **e-government**
- e-Serviciile furnizeaza **bunuri care nu sunt tangibile**
 - Exemplu: serviciul **FedEx** online de urmarire a pachetelor
- Ele raspund cerintelor si preferintelor utilizatorilor
 - presupun o **intelegerere** intre furnizor si consumator
- Se bazeaza pe **folosirea altor servicii** mai simple – **Exemple ?**
 - de **monitorizare** a activitatii utilizatorilor si a contextului
 - **vizualizare** obiecte multimedia, analiza imaginilor,
 - baze de date,
 - motoare de **cautare**,
 - **comunicare** in timp real,
 - suport al **activitatilor colaborative**,
 - control al accesului, **securitate**
 - **administrare** etc.

e-Business si e-commerce

- e-Business
 - cuprinde gestiunea lantului de **aprovizionare**, prelucrearea comenzilor, gestiunea serviciilor pentru **clienti**, cooperarea cu **partenerii** si altele.
- e-Commerce
 - se refera la tranzactiile de **vanzare/cumparare** a produselor comercializate si intarirea relatiilor cu clientii si furnizorii
- **Cerinte** (se adauga celor de la Web si aplicatii Web)
 - scalabilitate: suporta traficul de **varf**
 - **atomicitatea** tranzactiilor
 - organizarea datelor: **cautarea** usoara in catalogul de produse
 - **administrare** simpla – modificari facute de **ne-specialisti** IT

Exemplu: Amazon.com

Contextul

• dimensiunea

- platforma ofera servicii pentru mai multe site-uri web
- implementata pe o infrastructura de zeci de mii de servere si componente de retea din mai multe centre de date
- scalabilitatea suporta cresterea continua - este o proprietate importanta

• modul de operare

- infrastructura cuprinde milioane de componente
- in orice moment numarul de componente defecte este semnificativ
- tolerarea defectelor este modul normal de operare

Referinta: G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels.

Dynamo: Amazon's Highly Available Key-value Store

Cerinte operationale

Cerinte operationale stricte (care se regasesc si in alte platforme)

- **siguranta** (reliability) – cea mai mica intrerupere a serviciilor scade increderea clientilor si are impact financiar mare
- **tratarea defectarilor** – problema dificila deoarece sistemul este larg distribuit geografic

Solutii adoptate de Amazon

Obiectivul – gestiunea corecta a starii aplicatiei chiar in prezenta defectelor

- de ex. clientul sa poata vedea si adauga oricand obiecte in cosul de cumparaturi
 - serviciul de cumparaturi sa poata, oricand, citi si scrie in depozitul de date
 - datele sa fie disponibile in mai multe centre

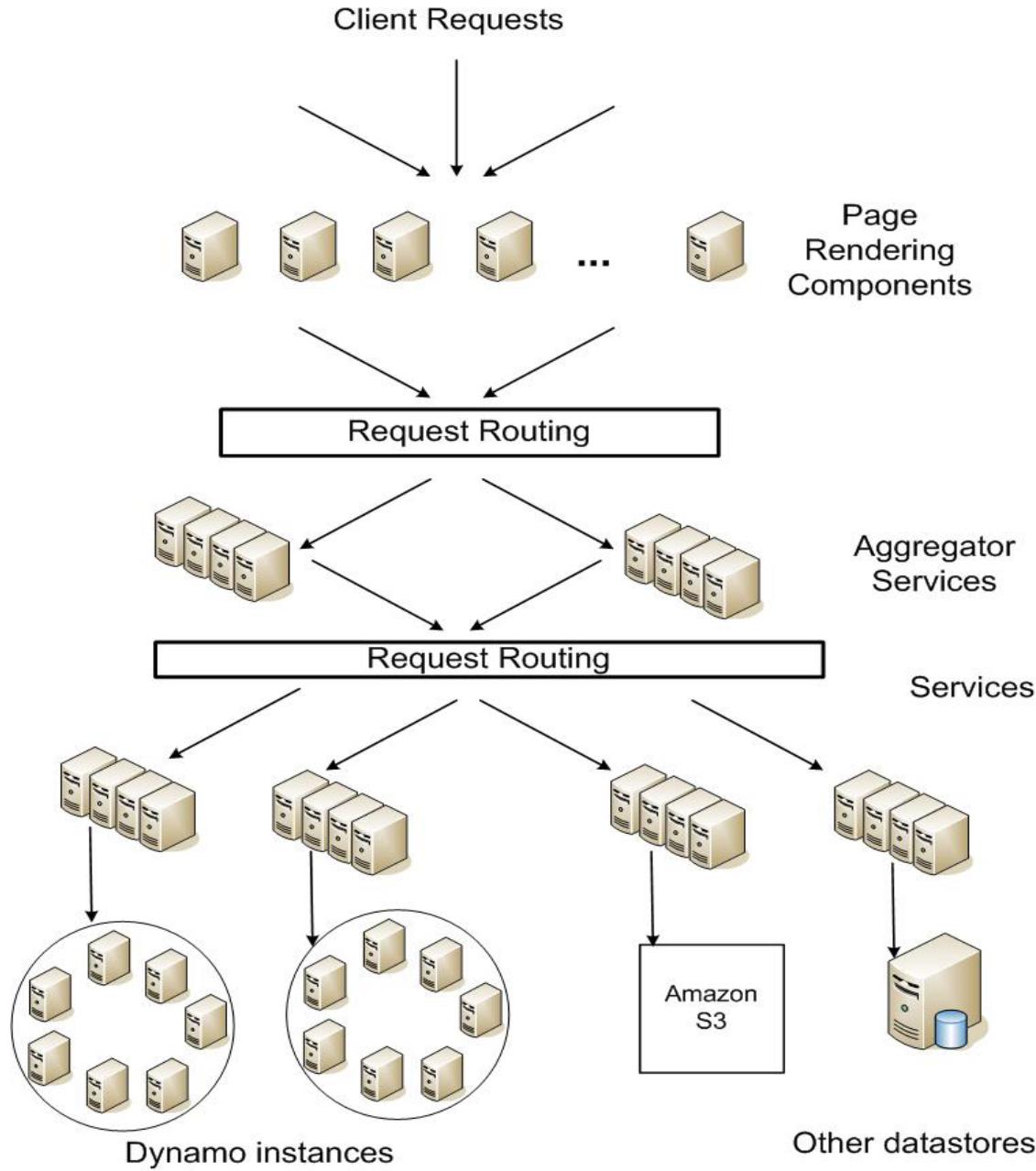
Solutii

- tehnologii de depozitare distribuita a datelor
 - Dynamo – folosit pentru scalabilitate si disponibilitate ridicata
 - Amazon S3 (Simple Storage Service)

ATENTIE!

Modelele si metodele mentionate in aceasta prezentare vor fi studiate in detaliu pe parcursul semestrului !

Arhitectura distribuită orientată pe servicii



O **cerere client** poate fi tradusa in cereri catre peste 150 de servicii

Un **serviciu** poate utiliza diferite **depozite de date** dintr-o zona accesibila

Unele servicii fac **agregarea** raspunsurilor furnizate de alte servicii

Modelul de date

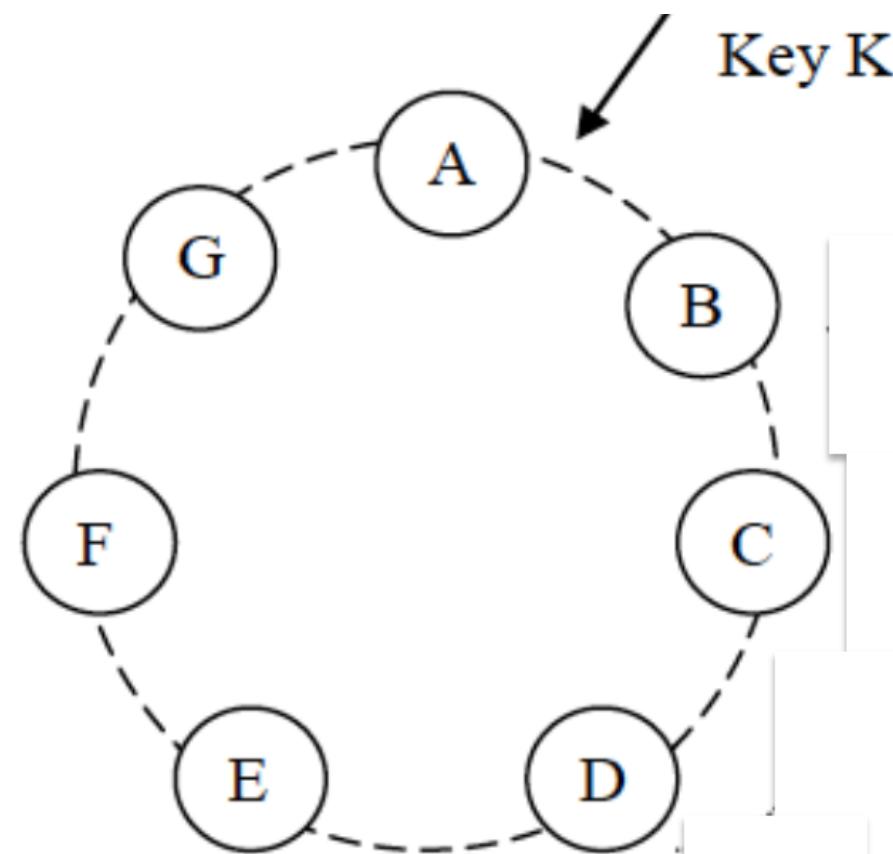
Tipic, sistemele de productie folosesc **modelul relational**

Amazon are un model simplu de date - **cheie-valoare**

- citiri si scrieri ale unor date identificate unic printr-o cheie
- **suficiente** pentru serviciile oferite, **de ex. ???**
 - lista celor mai buni vanzatori,
 - cosul de cumparaturi,
 - preferintele clientului,
 - gestiunea sesiunii,
 - nivelul vanzarilor,
 - catalogul de produse

Volume mari de date care trebuie partitionate

- datele sunt repartizate dinamic unui set de noduri (de stocare)
- foloseste o schema de hashing consistent cu spatiul valorilor organizat logic in inel
- fiecare nod are asociata o valoare din spatiu (pozitia nodului pe inel)
- fiecare data are asociata o valoare (din acelasi spatiu) egala cu hash-ul cheii (pozitia datei pe inel)
- data este stocata in nodul cu cea mai mica pozitie \geq decat pozitia datei (cheia K in nodul B)

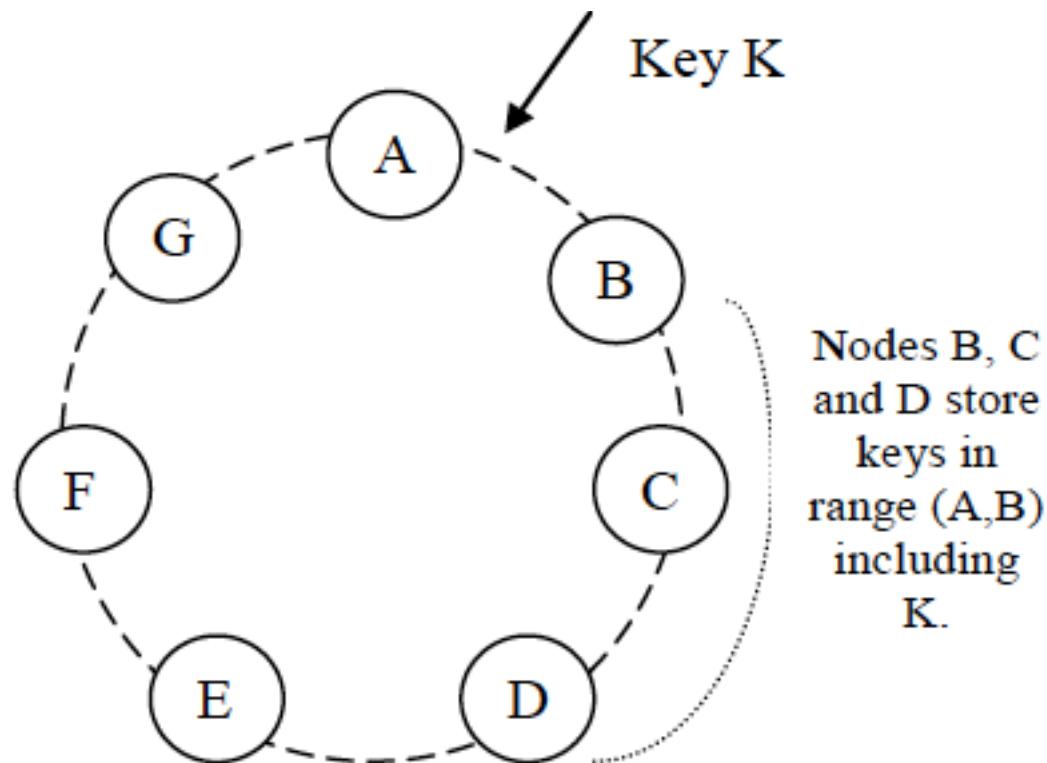


Replicarea

- modelul faciliteaza
 - o repartizare echilibrata a datelor pe diferite noduri
 - regasirea usoara a datelor in functie de cheie
 - adaugarea si eliminarea usoara a nodurilor

Replicarea

- fiecare data este replicata pe N noduri (N configurabil)
- schema: fiecare cheie este asignata unui coordonator (ex. K assign. nod B) care memoreaza datele corespunzatoare cheii si gestioneaza replicarea lor in alte noduri (ex. C si D)



Modelul de consistenta: eventual consistency

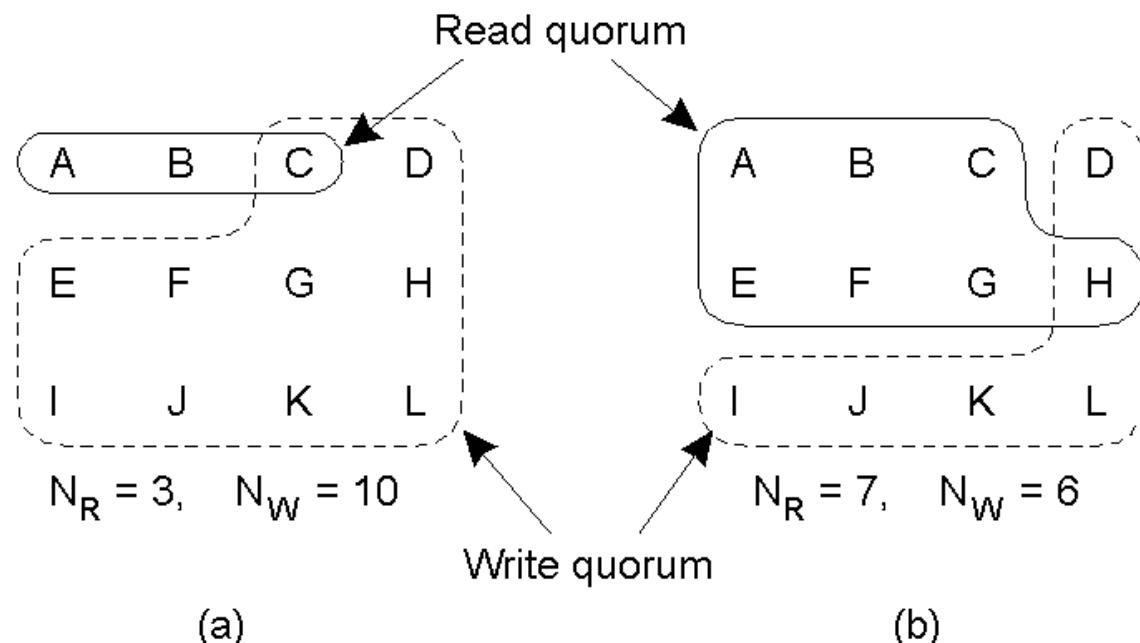
- nodurile de stocare a datelor sunt situate (de regula) in sisteme diferite – **de ce?**
 - modificarea unei valori se face pe o replica (usor) accesibila
 - modificarea este propagata asincron **celorlalte** replici
 - dupa un timp, daca nu sunt alte modificari, replicile vor detine aceeasi valoare
 - modelul = consistenta “**in cele din urma**”
 - in caz de **defect** actualizarea poate dura **un timp indelungat**
 - dar, un apel **PUT** se intoarce la apelant **inainte** de actualizarea tuturor replicilor (operatiile “Add to Cart” nu trebuie rejectate sau uitate !)
- un apel **GET** ulterior poate accesa o replica **neactualizata**

Pastrarea consistentei

- Solutia = **versionarea** obiectelor
- pentru **eficienta** – modificarea se face pe cea mai recenta **replica accesibila** iar rezultatul este tratat ca o noua **versiune** a obiectului
 - pentru a descrie **dependentele cauzale** intre versiuni se folosesc **ceasuri vectoriale**
- **reconcilierea** cu celelalte versiuni se face de catre **coordonator** la un apel **GET**
 - conditionata de relatiile dintre replici !
 - daca replicile pastreaza **mai multe versiuni** independente cauzal, reconcilierea se faca la nivelul aplicatiei

Tratarea defectelor tranzitorii

- executia **PUT** si **GET** se face conform **modelului de cvorum**:
 - pentru N replici, operatia se termina cand
 - minimum N_W noduri participa la operatia de scriere (**PUT**) sau
 - minimum N_R noduri participa la operatia de citire (**GET**)
 - conditia este $N_R + N_W > N$
- asigura ca grupul nodurilor citite include cel putin un nod care are ultima versiune a obiectului



Tratarea defectelor permanente

- doua noduri pot schimba intre ele informatii despre seturile de chei comune
- foloseste un protocol de sincronizare descentralizata a replicilor bazat pe Merkle trees
 - frunzele sunt hash-uri de chei
 - celelalte noduri sunt hash-uri ale fiilor
- doua noduri schimba intre ele radacinile arborilor Merkle corespunzatoare unor seturi de chei comune
- determina diferentele (prin parcurgerea arborilor) si fac sincronizarea

Alte functionalitati

- detectia nodurilor defecte
- adaugarea si eliminarea dinamica a nodurilor de memorie

Modelul de performanta

- Serviciul pentru **cosul de cumparaturi** – raspunde intr-o zi la **zeci de milioane de cereri** rezultate din peste 3 milioane de checkout-uri
- Serviciul de **gestiune a sesiunii** gestioneaza **mii de sesiuni active simultan**
- **Politica firmei** este de a oferi servicii bune unui **numar cat mai mare** de clienti
 - ex.: un serviciu garanteaza un timp de raspuns sub 300 msec pentru **99.9% din cereri**, la o incarcare de 500 de cereri pe secunda
 - procentul este determinat pe baza unei **analize de cost**
 - un procent mai mare ar determina o crestere semnifiativa a costului
 - pentru politica specificata - media / mediana / dispersia, folosite curent in alte sisteme, **nu sunt potrivite**

Cerintele altor categorii de servicii

- e-Banking:
 - Servicii electronice *robuste* si cat mai *complete*
 - *fiabilitate* (situri de rezerva)
 - *confidentialitate* si *protectie*
- e-Government
 - *integrarea* serviciilor conform unor modele de e-guvernare: G2C (Government to Citizens), G2B (Government to Businesses) etc.
- e-Health
 - toleranta la *erori* pentru transmisii audio/video, nu pentru EKG
 - mai multe *canale* de transmisie pentru domiciliu sau ambulanta
 - transmisie *rapida* in situatii de urgență
 - *protectia* si *confidentialitatea* datelor despre pacienti
- m-Services
 - adaptarea serviciilor la *context* (ex. timp, locatie)
 - adaptarea la *resursele* reduse ale mobilelor
 - mentinerea *continuitatii* serviciului (in prezenta deconectarilor)

Cerinte comune aplicatiilor

- utilizare simplă
- interoperabilitate
- performanță ridicată
- scalabilitate
- disponibilitate - **Livrarea** serviciilor *oricand, oriunde, și pe orice echipament*, inclusiv pe mobile
- toleranță la defectări
- satisfacerea unor constrângeri de timp real
- controlul accesului
- confidentialitate, integritate, de încredere
- deschidere la extinderi sau re-implementări
- **eficiență** economică (cost redus)
- **adaptarea** la profilul clientilor

Alte arhitecturi distribuite

Cloud = Colectie (nor) de resurse (hardware, software ...) distribuite in **Centre de date**, functionand dupa modelul client-server, pentru a furniza servicii de **infrastructura, platforma, software** etc., la cerere si contra cost

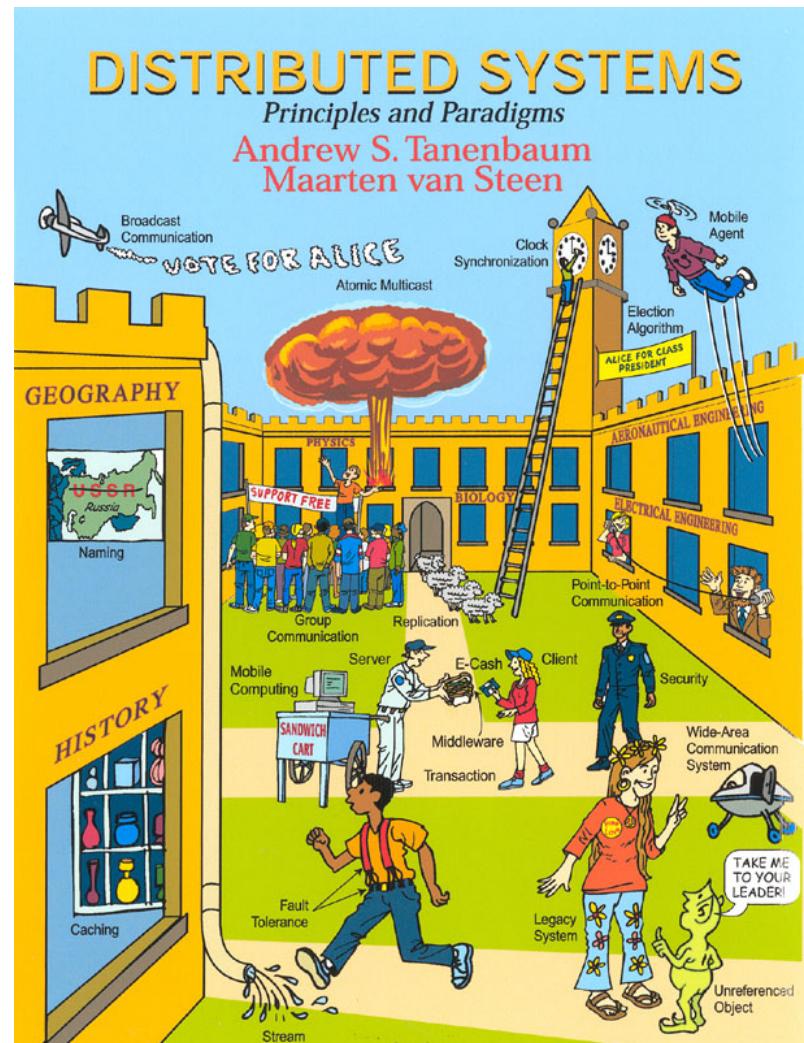
Grid = Colectie de resurse puse la dispozitie si folosite in comun de membrii unei Organizatii Virtuale (VO)

Retele Peer-to-Peer = colectie de resurse distribuite, controlate descentralizat de componente care sunt **in acelasi timp** clienti si servere

Sisteme distribuite - A doua definitie

- Un sistem distribuit este o colectie de calculatoare independente care apar utilizatorilor ca un singur sistem coherent.

Tanenbaum, van Steen
Prentice Hall 2007



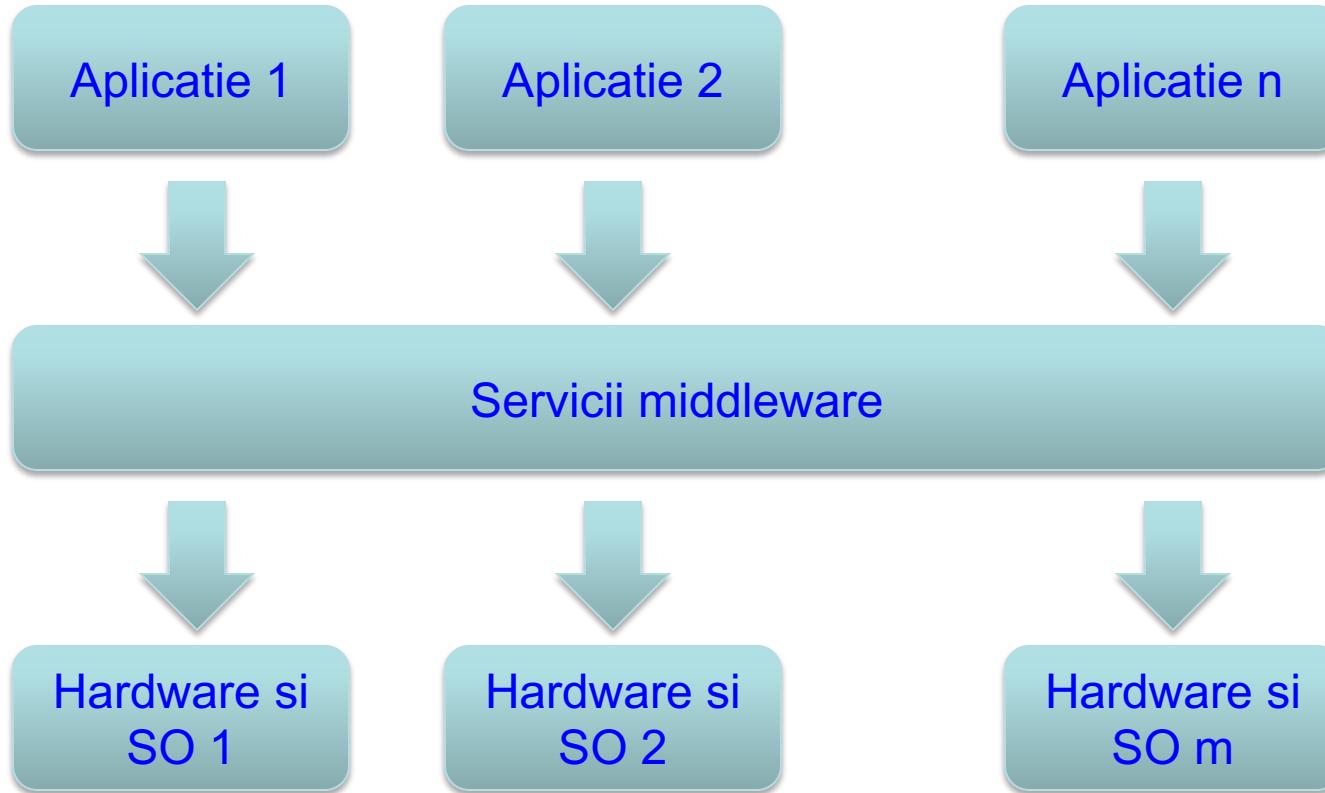
Avantajele sistemelor distribuite

- reducerea **timpului** de executie al aplicatiilor prin paralelizare si distributie,
- cresterea tolerantei la **defectari**
 - in SD defectele sunt partiale!
- **specializarea** functionala
 - exploatarea mai buna a particularitatilor componentelor
- raportul **cost / performanta** mai redus,
- usurinta de **acces** la resurse indepartate,
- cresterea **incrementala** a sistemului.

O arhitectura a sistemelor distribuite

- Intr-un sistem distribuit, componentele **aplicatiei**, situate in sisteme diferite inter-opereaza prin intermediul retelei
 - un **program** situat intr-unul din sisteme **are acces** la programe si date din alte sisteme
- **Platformele** (hardware si sistem de operare ale) sistemelor individuale **nu sunt construite** pentru calculul distribuit
 - ele folosesc produse hardware si software “de gata”, comercializate publicului larg - **Commercial off-the-shelf** (COTS)
- **Solutie:**
 - includerea, intre nivelul aplicatiilor si cel al platformei, a unui nivel suplimentar numit **middleware**, care ofera serviciile cerute de aplicatiile distribuite, prin interfete (API-uri) standard.

Solutia middleware



Avantajele middleware

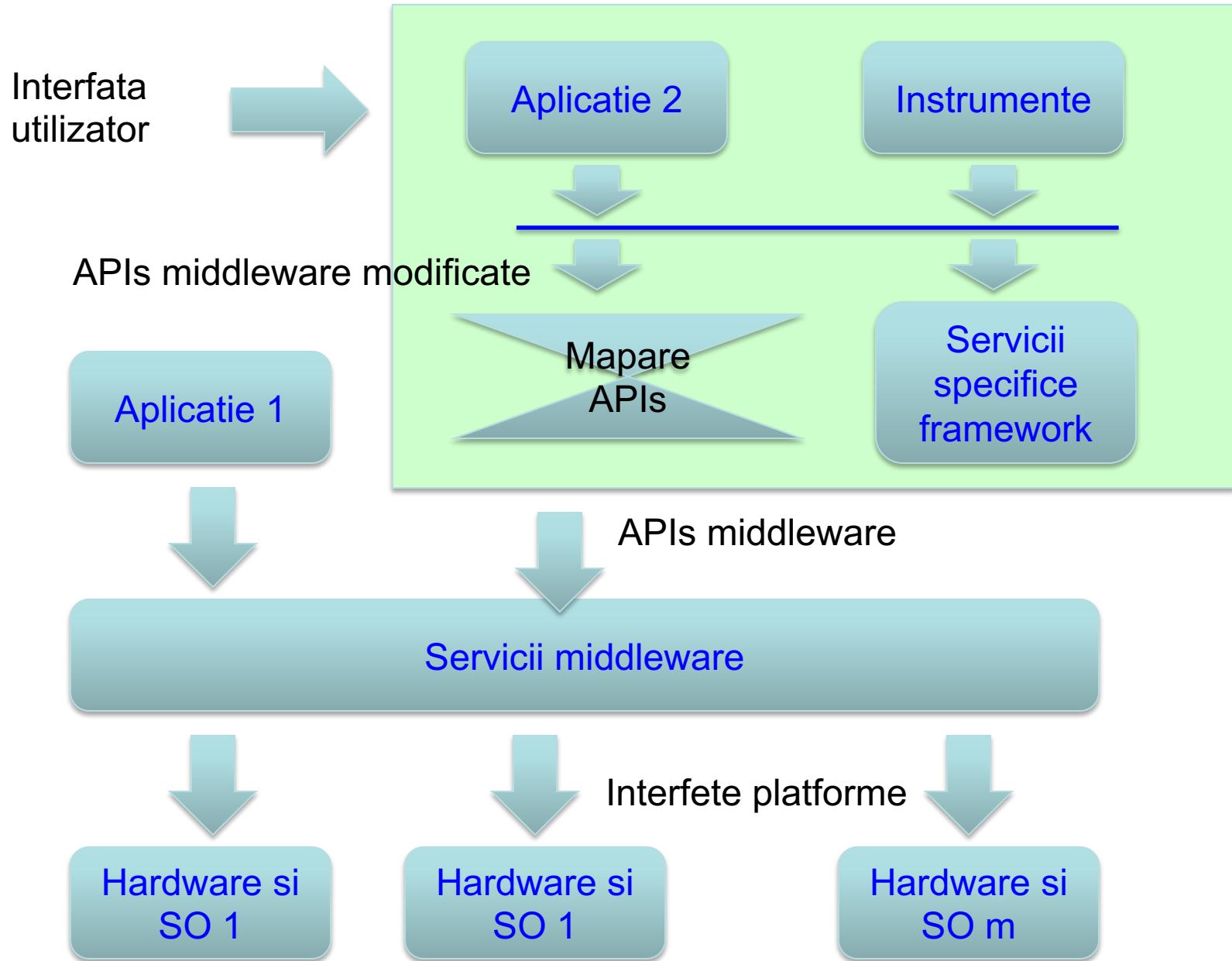
- ofera API-uri de nivel ridicat, independente de platforma (hardware si SO)
- permit dezvoltatorului de aplicatii sa se concentreze pe logica acestora
 - ii scuteste de rezolvarea problemelor de comunicare la distanta intre componente, acces concurrent la resurse, tolerarea defectarilor, controlul accesului etc.
- suporta interoperabilitatea
- permite extinderea treptata a sistemului
- suporta portabilitatea

Categorii de servicii middleware

- **Comunicarea** intre procese
 - transmitere mesaje si apeluri la distanta (RPC, RMI, SOAP)
- **Sincronizarea** proceselor
 - tratarea concurentei
 - tranzactii distribuite
- **Replicarea** si **consistenta** datelor
- Tratarea defectelor si **toleranta** la defectari
- **Mobilitatea**
- **Securitatea**

Frameworks

- Serviciile sunt **re-utilizabile**
- efectul in cazul middleware:
 - efortul dezvoltatorului de aplicatii se muta de la **dezvoltarea** unor componente noi, la **integrarea** unora existente
- **Cadrele de lucru** (frameworks) faciliteaza dezvoltarea aplicatiilor
- Un cadru de lucru include
 - mapari pe serviciile **middleware**
 - **servicii** proprii framework-ului
 - instrumente pentru folosirea framework-ului
 - **aplicatii**



END