1. Sisteme distribuite (SD): caracteristici fundamentale

• Sistem distribuit (SD): sistem ale carui componente se afla pe calculatoare interconectate in retea, comunica si se coordoneaza prin transfer de mesaje.

Un sistem distribuit este o colectie de calculatoare independente care apar utilizatorilor sistemului ca un singur calculator

• Caracteristici: concurenta componentelor, lipsa unui ceas global, erori (defecte, caderi, engl.failures) independente ale componentelor

2. Premize pentru dezvoltarea unor aplicații distribuite;

– Aplicațiile centralizate prezintă dezavantaje importante:
– Dependența de evoluția în timp a structurii de calcul,
– Mentenanța greoaie, determinată de structura ne-modulara a aplicației,
– Performanță penalizată de utilizarea în comun a resurselor de calcul

3.Obiectivele ale SD

*Extensibilitate – Măsură în care funcționalitatea și performanțele sistemului pot fi modificate fără a fi necesară modificarea proiectării acestuia.
Atribute: modificare ușoară și evoluție incrementală
*Integritate – Măsură în care sistemul tolerează defectele, erorile sau anomaliile care apar în timpul funcționării sale, menținîndu-se într-o stare corectă.
*Performanțe

4. Modelul arhitectural Kruchten „4+1”

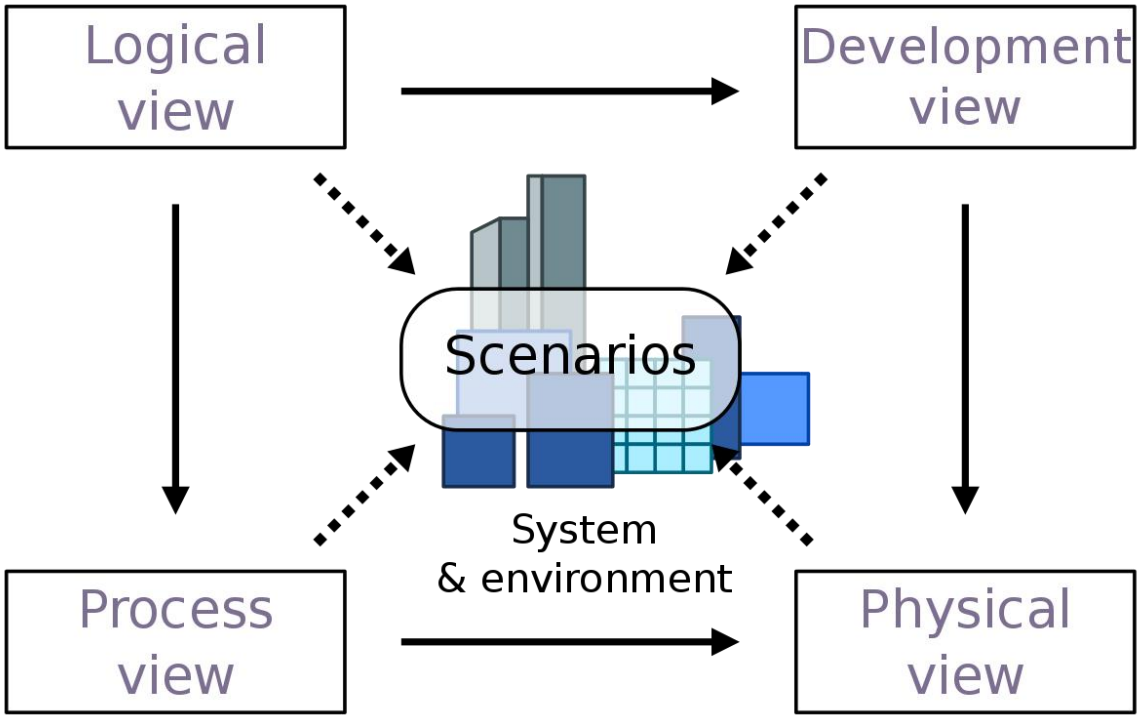
4 + 1 este un model de vedere utilizat pentru „descrierea arhitecturii sistemelor cu software intensiv bazat pe utilizarea mai multor vizualizări simultane”. [1] Părerile sunt utilizate pentru a descrie sistemul din punctul de vedere al diferitor părți interesate, cum ar fi utilizatorii finali, dezvoltatorii, inginerul de sistem și managerii de proiecte. Cele patru puncte de vedere ale modelului sunt viziuni logice, de dezvoltare, proces și fizice. În plus, sunt utilizate cazuri sau scenarii de utilizare selectate pentru a ilustra arhitectura care servește drept vizualizare „plus una”. Prin urmare, modelul conține 4 + 1 vizualizări: [1]

- ***Vizualizare logică*** : ***Vizualizarea*** logică este preocupată de funcționalitatea pe care sistemul le oferă utilizatorilor finali. Diagramele UML sunt utilizate pentru a reprezenta vizualizarea logică și includ diagrame de clasă și diagrame de stare . [2]
- ***Vizualizare proces*** : ***Vizualizarea*** procesului tratează aspectele dinamice ale sistemului, explică procesele sistemului și modul în care acestea comunică și se concentrează pe comportamentul în timp de rulare al sistemului. Vizualizarea procesului abordează concurgența, distribuția, integratorul, performanța și scalabilitatea, etc. Diagramele UML pentru a reprezenta vizualizarea procesului includ diagrama de activitate . [2]
- ***Vizualizare dezvoltare*** : ***Vederea de*** dezvoltare ilustrează un sistem din perspectiva programatorului și este preocupată de gestionarea software. Această vedere este cunoscută și sub denumirea de vedere de implementare. Utilizează diagrama componentelor UMLpentru a descrie componentele sistemului. Diagramele UML utilizate pentru a reprezenta vederea dezvoltării includ diagrama Pachetului . [2]
- ***Vizualizare fizică*** : ***vederea*** fizică prezintă sistemul din punctul de vedere al unui inginer de sistem. Este preocupat de topologia componentelor software de pe stratul fizic, precum și de conexiunile fizice dintre aceste componente. Această vedere este, de asemenea, cunoscută sub numele de vedere de implementare. Diagramele UML utilizate pentru a reprezenta vederea fizică includ diagrama de implementare . [2]

- ***Scenarii*** : Descrierea unei arhitecturi este ilustrată folosind un set mic de cazuri de utilizare , sau scenarii, care devin o a cincea vedere. Scenariile descriu secvențe de interacțiuni între obiecte și între procese. Sunt utilizate pentru identificarea elementelor arhitecturale și pentru ilustrarea și validarea designului arhitecturii. De asemenea, servesc ca punct de plecare pentru testele unui prototip de arhitectură. Acest punctvedere esteasemeneacunoscut sub numele de **vizualizarea cazurilor de utilizare** .

Modelul de vizualizare 4 + 1 este generic și nu se limitează la nicio notare, instrument sau metodă de proiectare. Citându-l pe Kruchten,

Modelul de vizualizare „4 + 1” este mai degrabă „generic”: se pot utiliza și alte notări și instrumente, se pot utiliza și alte metode de proiectare, în special pentru descompuneri logice și de proces, dar le-am indicat pe cele pe care le-am folosit cu succes.



5. Standardul de arhitecturare ANSI/IEEE-1471

IEEE 1471 este denumirea scurtă pentru un standard cunoscut oficial ca ANSI / IEEE 1471-2000, *Practică recomandată pentru arhitectură Descrierea sistemelor software-intensive.* În cadrul Institutului de Ingineri Electrici și Electronici (IEEE), aceasta este o „practică recomandată”, cea mai puțin normativă a standardelor sale. În 2007, acest standard a fost adoptat de ISO / IEC JTC1 / SC7 ca ISO / IEC 42010: 2007 , *Ingineria sistemelor și a software-ului - Practică recomandată pentru descrierea arhitecturală a sistemelor intensiv software* . [1]

A fost mult timp recunoscut [*de cine?*] că „arhitectura” are o influență puternică asupra ciclului de viață al unui sistem. Cu toate acestea, până relativ recent, [*când?*] problemele hardware au avut tendința să domine gândirea arhitecturală, iar aspectele software, atunci când au fost luate în considerare, au fost adesea primele compromise sub presiunile dezvoltării. [1] IEEE 1471 a fost creat pentru a oferi o bază de gândire despre arhitectura sistemelor cu software intensiv.

Contribuțiile IEEE 1471 pot fi rezumate după cum urmează (în această listă, articolele cu *caractere italice* sunt termeni definiți și folosiți în standard):

- Oferă definiții și un meta-model pentru descrierea *arhitecturii*
- Aceasta afirmă că o *arhitectură* ar trebui să abordeze *preocupările părților interesate* ale unui sistem
- Acesta afirmă că *descrierea arhitecturii* este, în mod inerent , viziune multiplă , nici o *viziune* unică nu surprinde în mod adecvat toate problemele părților interesate
- Acesta specifică noțiunile de *vedere* și *punct de vedere* , unde un *punct de vedere* identifică setul de *preocupări* și *reprezentările* / *tehnicile de modelare* etc. utilizate pentru a descrie *arhitectura* pentru a aborda acele *preocupări,* iar o *vedere* este rezultatul aplicării unui punct de vedere la un anumit sistem.
- Stabilește cerințele de conținut pentru descrierile arhitecturii și ideea că o *descriere arhitecturală conformă* are o corespondență 1 la 1 între *punctele de vedere* și *opiniile* sale .
- Oferă îndrumări pentru captarea *rațiunilor de arhitectură* și identificarea inconsistențelor / problemelor nerezolvate între *vizualizările* dintr-o *descriere a arhitecturii*

IEEE 1471 oferă anexe informative care se referă la conceptele sale cu conceptele de arhitectură din alte standarde, inclusiv RM-ODP și IEEE 12207 .

6. Stiluri arhitecturale

- a. layered
- **Stratul de prezentare** conţine toate clasele responsabile de prezentarea interfeţei de utilizator către utilizatorul final sau de a trimite răspunsul înapoi la client (în cazul în care vom opera în profunzime).
- **Stratul de aplicaţie** conţine toată logica cerută de aplicaţie pentru a îndeplini cerinţele sale funcţionale şi, în acelaşi timp, nu face parte din regulile domeniului. În majoritatea sistemelor cu care am lucrat, stratul de aplicaţie a constat în servicii care orchestrează obiectele de domeniu pentru a îndeplini un scenariu de caz de utilizare.
- **Stratul de domeniu** reprezintă domeniul de bază, format în cea mai mare parte din entităţi de domeniu şi, în unele cazuri, servicii. Regulile de afaceri, cum ar fi invariante şi algoritmi, ar trebui să rămână toate în acest strat.
- **Stratul de infrastructură (cunoscut şi sub denumirea de stratul de persistenţă)** conţine toate clasele responsabile de realizarea lucrurilor tehnice, cum ar fi persistenţa datelor din baza de date, cum ar fi DAO-uri, depozite sau orice altceva utilizaţi.

b.data flow

În arhitectura fluxului de date, întregul sistem software este văzut ca o serie de transformări pe bucăți consecutive sau set de date de intrare, unde datele și operațiunile sunt independente unele de altele. În această abordare, datele intră în sistem și apoi trec prin module, simultan, până când sunt atribuite la o anumită destinație finală (ieșire sau un depozit de date).

Obiectivul principal al acestei abordări este atingerea calităților de refolosire și modificare. Este potrivit pentru aplicațiile care implică o serie bine definită de transformări sau calcule independente de date pe intrare și ieșire ordonată, cum ar fi

compilatoare și aplicații de prelucrare a datelor de afaceri. Există trei tipuri de secvențe de execuție între module -

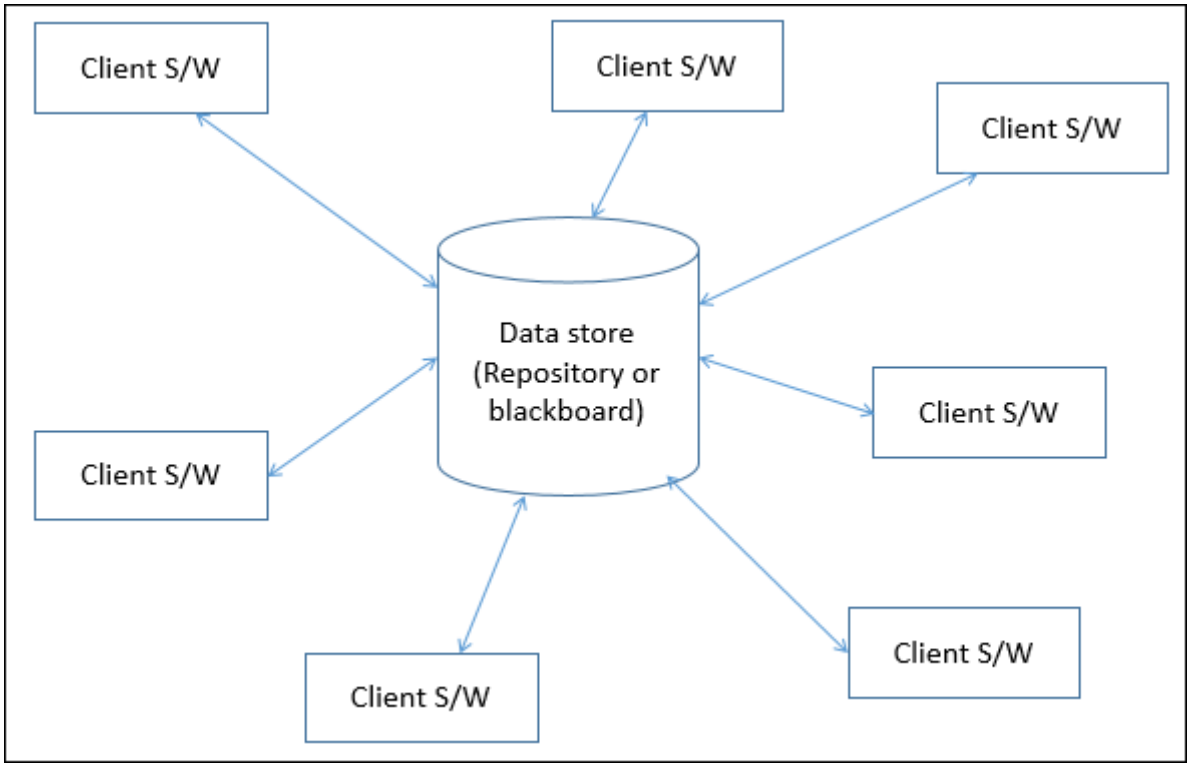
- Secvențial lot
- Modul de conductă și filtru sau nesecvențial
- Controlul procesului

c.data-centered

În arhitectura centrată pe date, datele sunt centralizate și accesate frecvent de alte componente, care modifică datele. Scopul principal al acestui stil este de a realiza integritatea datelor. Arhitectura centrată pe date constă din diferite componente care comunică prin intermediul depozitelor de date partajate. Componentele accesează o structură de date partajată și sunt relativ independente, prin aceea că interacționează numai prin intermediul depozitului de date.

Cele mai cunoscute exemple de arhitectură centrată pe date este o arhitectură a bazelor de date, în care schema bazei de date comune este creată cu protocolul de definire a datelor - de exemplu, un set de tabele înrudite cu câmpuri și tipuri de date într-un RDBMS.

Un alt exemplu de arhitecturi centrate pe date este arhitectura web care are o schemă de date comună (adică meta-structură a Web-ului) și urmează modelul și procesele de date hipermedia comunicând prin utilizarea serviciilor de date partajate pe web.



d.adaptation

Arhitectura adaptivă este un sistem care își schimbă structura, comportamentul sau resursele în funcție de cerere

e.distribution

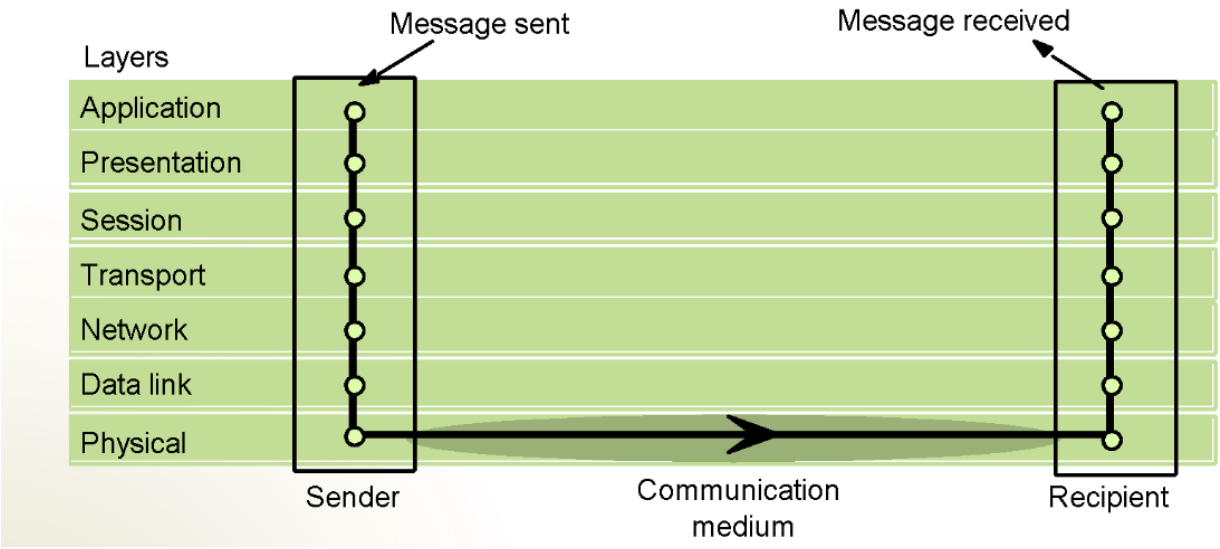
În arhitectura distribuită, componentele sunt prezentate pe diferite platforme și mai multe componente pot coopera între ele printr-o rețea de comunicații pentru a atinge un obiectiv sau un obiectiv specific.

- În această arhitectură, procesarea informațiilor nu se limitează la o singură mașină, ci este distribuită pe mai multe computere independente.
- Un sistem distribuit poate fi demonstrat prin arhitectura client-server care constituie baza pentru arhitecturi cu mai multe niveluri; alternative sunt arhitectura de broker precum CORBA și Arhitectura orientată către servicii (SOA).
- Există mai multe cadre tehnologice pentru a sprijini arhitecturi distribuite, inclusiv .NET, J2EE, CORBA, servicii Web .NET, servicii web Java AXIS și servicii Globus Grid.
- Middleware este o infrastructură care sprijină în mod corespunzător dezvoltarea și execuția aplicațiilor distribuite. Oferă un tampon între aplicații și rețea.
- Acesta se află în mijlocul sistemului și gestionează sau susține diferitele componente ale unui sistem distribuit. Exemple sunt monitoarele de procesare a tranzacțiilor, convertoarele de date și controlerele de comunicare etc.

7. TCP/UDP

1. CP garantează livrarea pachetelor de date într-o formă, secvență și fără pierderi, UDP nu garantează nimic.
2. Pachetele cu numere TCP în timpul transmisiei, dar UDP nu
3. TCP funcționează în modul duplex complet, într-un singur pachet puteți trimite informații și confirma primirea pachetului anterior.
4. TCP necesită o conexiune prestabilită, UDP nu necesită o conexiune, ci doar un flux de date.
5. UDP oferă o rată de date mai mare.
6. TCP este mai fiabil și controlează procesul de schimb de date.
7. UDP este de preferat pentru programele care redă streaming video, fonetii video și telefonie, jocuri de rețea.
8. UPD nu conține funcții de recuperare a datelor

Ierarhii de servicii in comunicatii



Socket

Socket (eng. *Socket* - conector) - numele interfeței software pentru schimbul de date între procese . Procesele în timpul unui astfel de schimb pot fi realizate atât pe un computer , cât și pe diverse computere conectate de o rețea . Un soclu este un obiect abstract care reprezintă punctul final al unei conexiuni.

Este necesar să se facă distincția între **soclurile client** și **server** . Soclurile clienților pot fi comparate aproximativ cu dispozitivele finale ale rețelei de telefonie , iar soclurile serverului cu comutatoare . Aplicația client (de exemplu, browserul) folosește numai socluri client, iar aplicația server (de exemplu, serverul web către care browserul trimite solicitări) folosește atât socluri client cât și server.

Interfață priză pentru prima dată a apărut în BSD Unix . Interfața de programare a soclurilor este descrisă în standardul POSIX .1 și este acceptată într-un grad sau altul de *toate* sistemele de operare moderne .

Principii de priză [editare | edit cod]

Adresele și porturile sunt utilizate pentru a comunica între mașini folosind stiva de protocol TCP / IP . Adresa este o structură pe 32 de biți pentru IPv4 , 128 biți pentru IPv6 . Numărul portului este un număr întreg în intervalul de la 0 la 65535 (pentru protocolul TCP).

Această pereche definește soclul („soclu” corespunzător adresei și portului).

În timpul procesului de schimb, de regulă, sunt folosite două prize - priza expeditorului și priza receptorului. De exemplu, când accesați serverul pe un port HTTP , socket-ul va arăta astfel: 194.106.118.30:80, iar răspunsul va merge la mmm.nnn.ppp.qqq: xxxxx.

Fiecare proces poate crea o priză „de ascultare” (soclu server) și le poate *lega* la un port al sistemului de operare (pe UNIX, procesele neprivilegiate nu pot utiliza porturi mai mici de 1024).

Procesul de ascultare este de obicei într-o buclă de așteptare, adică se trezește când apare o nouă conexiune. În același timp, rămâne posibil să verificați conexiunile în acest moment, să setați un interval de timp pentru operație etc.

Fiecare priză are propria sa adresă. Sistemele UNIX pot suporta mai multe tipuri de adrese, dar sunt necesare o adresă INET și o adresă UNIX . Dacă atașați socket-ul la o adresă UNIX, va fi creat un fișier special (fișier *socket*) pe calea specificată prin care orice proces local poate fi informat citind / scrierea de pe acesta (consultați Socket Domain Unix). Soclurile de tip INET sunt accesibile din rețea și necesită alocarea unui număr de port.

De obicei, clientul se „conectează” în mod explicit la ascultător, după care orice citire sau scriere prin descriptorul fișierului va transfera date între acesta și server.

- **Unicast** : de la o sursă la o destinație, adică unu la unu
- **Broadcast** : de la o sursă la toate destinațiile posibile, adică One-to-All
- **Multicast** : de la o sursă la destinații multiple, care afirmă un interes în primirea traficului, adică One-to-Many