Proiectarea arhitecturilor sistemelor software

Student: Voicu Eduard-Robert

Grupa: 442A

Capitol: 3,4

Student: Mihai Marian

Grupa: 441A

Capitol: 1,2

CUPRINS:

1. Introducere arhitecturi software

1.1. Introducere generala asupra arhitecturilor software

Termenul *arhitectura software* denota nivelul inalt al structurii unui sistem software. Poate fi definit ca un set de structuri necesare pentru a motiva sistemul software, care cuprinde elemente software, si relatiile dintre ele, si proprietatile elementelor cat si a relatiilor. Termenul arhitecturi software denota, de asemenea, si un set de practici folosite pentru a selecta, defini sau concepe o arhitectura software.[2]

Arhitecturile software se bazeaza pe trei principii:

1. Structura high-level a unui sistem software.
2. Disciplina de a crea o astfel de structura high-level.
3. Documentatia unei astfel de structuri high-level.

Una din cele mai puternice mecanisme pentru descrierea unei arhitecturi este descompunerea ierarhica asa cum se arata in Fig.1 de mai
jos.

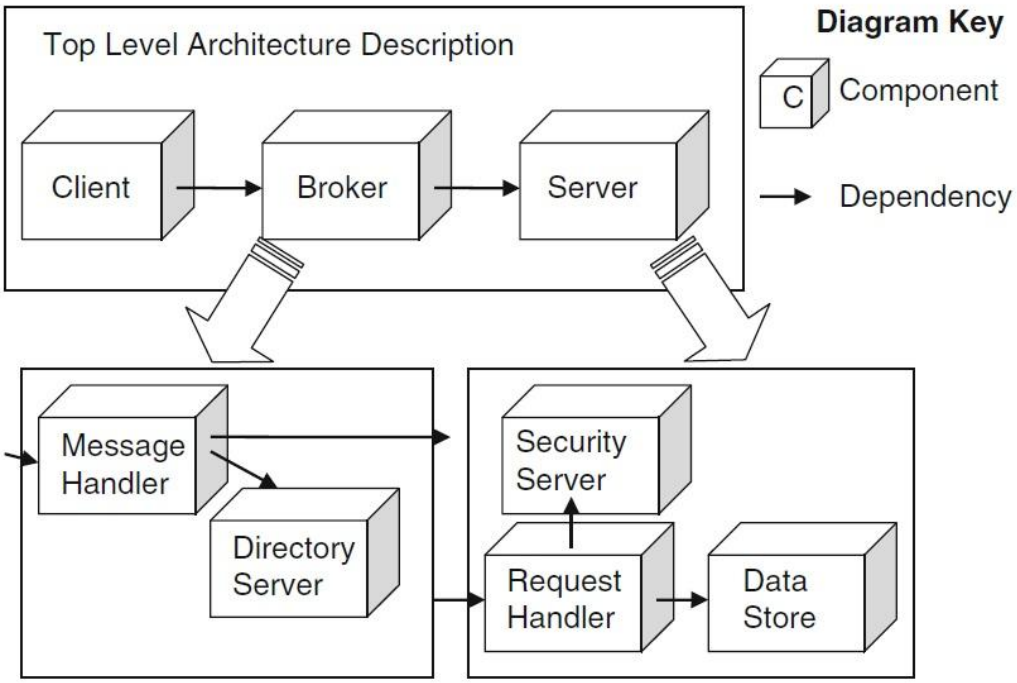

Fig.1 Descrierea unei arhitecturi ierarhice [1]

1.1.1. Caracteristicile unei arhitecturi software[2]

Arhitecturile software prezinta urmatoarele caracteristici:

- **Multitudinea de stakeholders.**

Un stakeholder este o persoana, un grup de persoane, o organizatie sau un sistem care afecteaza sau este afectat de catre actiunile organizatiei.

Sistemele software trebuie sa raspunda unei varietati de stakeholders cum ar fi manageri de bussines, utilizatorii sau operatorii. Toti acesti stakeholders au propriile lor preocupari in ceea ce priveste sistemul. Punerea in balanta a acestor preocupari si demonstrand cum sunt ele adresate face parte din arhitectura software a sistemului. [2]

- **Separarea preocuparilor**.

Modul stabilit de arhitecti pentru a reduce complexitatea este aceea de a separa preocuparile care conduc design-ul. Documentatia arhitecturilor software arata ca toate preocuparile stakehold-erilor sunt adresate de modelul si descrierea arhitecturii din puncte de vedere diferite. Aceste puncte, diferite, de vedere se numesc vederi arhitecturale.

- **Quality-driven.**

Abordarile clasice ale design-ului software au fost conduse de catre functionalitati necesare si de catre flow-ul datelor din sistem, dar intelegerea curenta asupra design-ului software este strans legata de atributele calitatii(toleranta la defecte, compatibilitate inversa, extensibilitate, incredere, mentenanta, securitate, uzabilitate).

- **Stiluri recurente** .

Ca si arhitectura cladirilor, disciplina arhitecturii software a dezvoltat moduri standard de adresare a preocuparilor recurente. Aceste moduri standard au diferite nume si nivele de abstractizare diferite. Termenii comuni pentru solutiile recurente sunt: stilul architectural, strategia arhitecturala sau tactica arhitecturala.

- **Integritate conceptuala.**

Termenul de integritate conceptuala a fost introdus de catre Fred Brooks in “The Mythical Man-Month” pentru a denota ideea ca arhitectura sistemelor software reprezinta o viziune globala a ceea ce trebuie facut si modul cum trebuie facut. Aceasta viziune trebuie sa fie separata de implementare.Arhitectul isi asuma rolul de “keeper of the vision”, si trebuind sa se asigure ca adaugari la sistem trebuie sa fie liniare cu arhitectura , prezervand integritatea conceptuala.

1.1.2. Motivare.

Arhitectura software reprezinta abstractizarea intelectuala a capabilitatii de a intelege un sistem complex. Aceasta abstractizare da un numar de beneficii:

1. *Da o baza a analizei comportamentului sistemului software inainte de a fi creat.* Abilitatea de a verifica ca viitorul sistem software indeplineste cerintele stakehold-erilor fara a fi nevoie de construirea propriu zisa a sistemului reprezinta o economie financiara substantiala si o atenuare a riscului. [2]

2. *Prevede o baza pentru reutilizare a elementelor si deciziilor.* O arhitectura software completa

1.2. Arhitecturi SOA – Service Oriented Arhitecture

SOA (Service Oriented Architecture) este un tip de arhitectura software care presupune distribuirea functionalitatii aplicatiei in unitati mai mici, distincte - numite servicii - care pot fi distribuite intr-o retea si pot fi utilizate impreuna pentru a crea aplicatii destinate afacerilor. Capacitatea mare cu care pot fi reutilizate aceste servicii in aplicatii diferite este o caracteristica a arhitecturilor software bazate pe servicii. Aceste servicii comunica intre ele trimitand informatii de la un serviciu la altul. SOA este deseori vazuta ca o evolutie a programarii distribuite si a programarii modulare.[1]

SOA este o arhitectura flexibila si standardizata ce contribuie la o mai buna conectare a diverselor aplicatiilor si faciliteaza schimbul de informatii intre acestea. SOA unifica procesele de business structurand marile aplicatii intr-o colectie de module mai mici numite servicii. Aceste aplicatii pot fi folosite de diverse grupuri de oameni atat din cadrul companiei cat si din afara ei.

Serviciile sunt unitati functionale neasociate care nu au apeluri unele catre altele inglobate in ele. In mod tipic sunt implementate functionalitati pe care majoritatea oamenilor le-ar recunoaste ca si serviciu cum ar fi de exemplu completarea unei aplicatii online pentru un cont, vizualizarea unui formular de banca sau a unui extras de cont online sau efectuarea unei comenzi de bilet de avion online.

SOA poate sa usureze integrarea diverselor medii care sa gasesc in multe organizatii. SOA faciliteaza colaborarea si distribuirea de informatii in cadrul intregii organizatii si cu partenerii externi. Prin expunerea proceselor de business, SOA va ajuta sa va concentrati asupra celor mai bune metode de a va imbunatati operatiunile. SOA va furnizeaza capacitatea de a sustine un model de afacere care depaseste granitele organizatiei. SOA imbunatateste colaborearea, faciliteaza procesele de afaceri complete si imbunatateste eficacitatea operationala.

1.3. Arhitecturi distribuite

Sistemele distribuite create pana in prezent detin o arhitectura mare. Cu toate aceseta, ele au in comun o serie de caracteristici si impartasesc unele probleme comunein dezvoltarea lor. Caracteristicile comune si aspectele de proiectare a sistemelor distribuite pot fi prezentate sub forma unor modele descriptive. Fiecare model va reprezenta o descriere abstracta, simplificata dar consistenta a aspectelor relevante ale proiectarii sistemelor distribuite.[1]

In functie de semnificatia definitiei de componenta, arhitectura sistemelor poate fi definita intr-un sens restrans si intr-un sens mai larg. Proiectarea arhitecturii unui program poate viza, in sens restrins, componentele programului, respectiv modulele acestuia, insa ea poate fi extinsa prin includerea bazei de date si a componentei middleware care permite configurarea comunicarii intr-un sistem client/server.

Proprietatile modulelor componentelor sunt caracteristicile care permit intelegerea modului in care ele interactioneaza si modul de apelare a unui modul din alt modul sau mecanismul de accesare a bazei de date de catre modulele software-ului. Proiectarea arhitecturala a programului nu ia in considerare proprietatile componentelor, cum ar fi detaliile unui algoritm specifice unui modul.

Legaturile dintre componente se pot realiza fie la apelarea unei proceduri, cu transmiterea eventuala a datelor necesare executiei procedurii respective, fie la protocolul de accesare a bazei de date de catre procedurile de program.[1]

Principalul obiectiv urmarit in cadrul proiectarii arhitecturale vizeaza conceperea unei structuri a sistemului care sa corespunda cerintelor prezente si celor viitoare, astfel incat sistemul sa fie sigur in functionare, adaptabil, usor de gestionat, eficient. O buna proiectare arhitecturala se va traduce intr-un sistem usor de implementat, testat si modificat.

Maretia sistemelor distribuite implementate pana in prezent detin o varietate mare a arhitecturilor, dar care pot totusi fi incadrate in cateva modele arhitecturale. Un model arhitectural defineste modul in care interactioneaza intre ele componentele unui sistem, precum si localizarea (maparea) lor intr-o retea de calculatoare. Arhitectura unui sistem distribuit are rolul de a simplifica si abstractiza (in sensul de a evidentia caracteristicile esentiale ale sistemului) functiile componentelor sistemului.

1.4. Arhitecturi de aplicatie mobile

Aplicatiile mobile reprezinta o provocare pentru lumea dezvoltatorilor de software, desi primele incercari au dat rezultate inca de acum cativa zeci de ani. Evolutia tehnologica a mutat insa aceasta industrie dintr-o zona exclusivista catre una pur comerciala, care suscita interesul marilor companii producatoare de instrumente hardware si software din domeniu. [2]

Ideea de a putea utiliza dispozitive si aplicatii care sa se bazeze pe tehnologii fara fir a atras dezvoltatorii inca de la aparitia retelei ARPANET, in anul 1969, ca precursor al INTERNET-ului. Existenta unei infrastructuri radio, care s-a cristalizat incepand cu anii 1920, a condus la dorinta de a combina cele doua tipuri de tehnologii pentru a genera una capabila sa ruleze aplicatii disponibile pe medii wireless radio. Primele incercari de acest gen au fost realizate in anul 1974, cand Motorola a produs primul pager comercial(Motorola Pageboy 1), de dimensiunea unui pachet de tigari. Dispozitivul in cauza nu facea decat sa emita un semnal sonor care il instiinta pe proprietar ca trebuie sa sune de urgenta la serviciu. Urmatorii pasi au constat in producerea pagerelelor de tip tone-voice, care transmiteau si un mesaj vocal destinatarului, iar mai apoi in producerea pagerelor digitale cu afisaj pe un ecra incorporat. Tehnologia pagerelor digitale a fost dominata pe piata pana la inceputul anilor 1990, cand aparitia telefoanelor mobile a dus la scaderea dramatica a vanzarilor de dispozitive de tip pager.[1]

In acest timp, evolutia INTERNETULUI a dus la definirea limbajului HTML ca standard pentru serviciile furnizate de aplicatiile web. Intalnirea dintre tehnologia deja existenta pe clasicul web si noile dispozitive mobile nu a fost tocmai fericita deoarece acestea din urma au o serie de caracteristici care le fac inadecvate pentru modelul INTERNET:

- puterea generata de baterie este relative scazuta, ceea ce le impiedica sa apeleze la procesoare mai puternice;
- puterea de calcul este limitata;
- capacitatea locala de stocare este nesemnificativa;

- posibilitatile de afisare sunt destul de reduse atat din punct de vedere al dimensiunii, cat si al performantelor calitative.

Pe masura evolutiei tehnologice, aceste limitari vor fi sigur ameliorate, insa ele au impus initial apelarea la un nou standard care sa fie potrivit pentru constrangerile dispozitivelor mobile. Acest standard, s-a numit HDML(Handled Device Markup Language) si nu a avut succesul scontat, insa a constituit punctul de pornire in realizarea si impunerea WAP(Wireless Application Protocol). Dezvoltate in urma colaborarii dintre Nokia, Ericson, Motorola, si Unwired Planet, specificatiile WAP nu include WML(Wireless Makeup Language) si WMLScript(un limbaj de scripting asemanator cu javaScript).

2. Arhitecturile SOA

Arhitecturile bazate pe servicii si serviciile Web reprezinta ultimul pas in dezvoltarea tehnologiilor middleware. Aceste tehnologii rezolva problema inter-operabilitatii si ofera baza pentru dezvoltarea de aplicatii Internet de mari dimensiuni.[5]

Tehnologiile middleware care permit integrarea aplicatiilor sunt folosite in diferite scopuri, de la interconectarea componentelor unei aplicatii desktop sau Web, pana la dezvoltarea de sistem software care se intind peste Internet. Tehnologiile traditionale precum serverele de aplicatii J2EE sau cele bazate pe mesaje reprezinta solutii ideale pentru dezvoltarea de sistem software care ruleaza in cadrul unei singure organizatii. Totusi, ele sunt destul de limitate cand se pune problema interconectarii sistemelor software folosite de organizatii diferite, care sunt conectate prin Internet. Serviciile Web si arhitecturile bazate pe servicii sunt proiectate tocmai pentru a satisface aceste nevoi.[5]

In multe feluri tehnologiile bazate pe servicii si serviciile Web nu reprezinta o noutate. La fel ca restul tehnologiilor si arhitecturilor care ofera suport pentru calcul distribuit, principalul scop al tehnologiilor bazate pe servicii este acela de a oferi suport pentru un sistem software de a invoca functionalitatii oferite de un alt sistem software (asa cum J2EE permite clientilor Java sa apeleze metode implementate de componente J2EE).

Principala diferenta consta in faptul ca accentul in cazul arhitecturilor bazate pe servicii se pune pe interoperabilitate si rezolvarea problemelor ridicate de folosirea de platforme si limbaje diferite. Desi se poate dezvolta o arhitectura bazata pe servicii utilizand orice tehnologie care permite calcul distribuit, numai serviciile Web ofera un grad de interoperabilitate nelimitat.

Necesitatea pentru interoperabilitate izvoraste din faptul ca majoritatea companiilor mari detin un mix de sisteme software in ceea ce priveste limbajele de programare si platformele utilizate. In conditiile in care reimplementarea acestor sisteme pe o singura platforma este mult prea costisitoare si prea riscant este evidenta necesitatea pentru tehnologii middleware care sa permita comunicare intre aceste sisteme software. Nu de putine se intampla ca sistemele software care trebuie sa comunice sa fie dezvoltate pentru platforme incompatibile, de aceea este nevoie de interoperabilitate.

Serviciile Web si arhitecturile bazate pe servicii reprezinta o solutie care permite integrarea diferitelor tehnologii.

2.1. Sisteme bazate pe servicii.[5]

Nevoie de integrare a sistemelor de business a existat din totdeauna. Aceasta integrare realizandu-seprin intermediul documentelor fizice schimbate de diferite organizatii: facturi, chitante sau ordine. In prezent insa datorita Internetului si a serviciilor Web se tinde tot mai mult inspre inlocuirea documentelor fizice cu cele electronice.

Principiile care stau la baza arhitecturilor bazate pe servicii nu sunt noi. Unele principii, ca de exemplu, reducerea intarzierilor introduse de comunicarea pe retea transmitand cat mai multa informatie intr-un singur apel, sunt deja folosite in sistemele software distribuite care ofera performante ridicate.

Ce este diferit despre sistemele orientate spre servicii si serviciile Web este faptul ca aceste aplicatii si servere este de asteptat ca in prezent sa fie accesate de organizatii din afara si de diferiti indivizi peste Internet.

In figura de mai jos se arata o aplicatie tipica de retail bazata pe Internet. Clientii vad o singura aplicatie globala care ii lasa sa comande carti, muzica si sa faca plati.

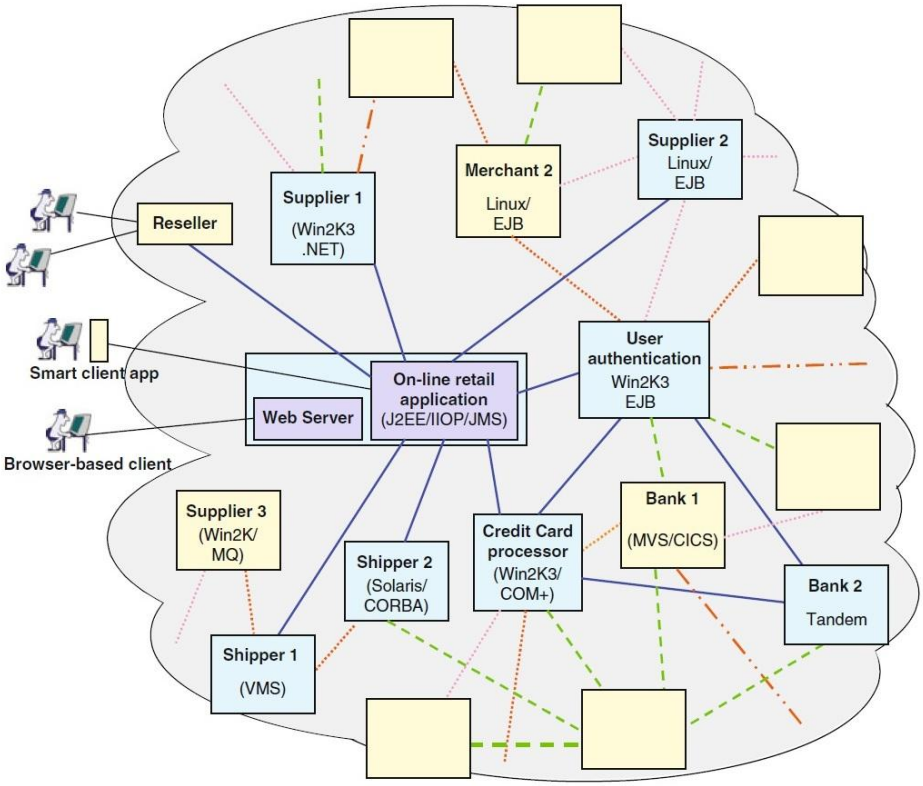

Fig.2 Aplicatie retail [5]

In realitate aceasta aplicatie consta in mici nuclee logice furnizate de diferiti parteneri de afaceri care la randul lor detin servicii oferite de catre alti provideri, si totul ruleaza pe un mix divers de platforme si middleware.

Clientii pot accesa aceasta aplicatie folosind browserul. Aceasta aplicatie ar putea fi construita utilizand oricare din tehnologiile middleware.

Principiile care stau la baza arhitecturilor bazate pe servicii nu sunt noi. Unele principii, ca de exemplu, reducerea intarzierilor introduse de comunicarea pe retea transmitand cat mai multa informatie intr-un singur apel, sunt deja folosite in sistemele software distribuite care ofera performante ridicate.

Principiile de baza ale arhitecturilor bazate pe servicii sunt:

- granitele sunt explicite;

- serviciile sunt autonome;

- se partajeaza scheme si contracte, si nu implementari;

- compatibilitatea este bazata pe reguli (policy).

In continuare vor fi detaliate fiecare din cele patru principii de baza ale unei arhitecturi bazate pe servicii.

2.1.1. Granite explicite

Serviciile sunt aplicatii de sine statatoare si nu doar cod care poate fi apelat de o aplicatie client. Accesarea unui serviciu necesita cel putin traversarea granitelor care separa procesele, si cel mai probabil traversarea retelei si utilizarea autentificarii inter-domenii. Fiecare granita care trebuie traversata (proces, masina, securitate) reduce performanta, creste complexitatea si creste probabilitatea de defectare. Foarte important este faptul ca ele trebuie recunoscute si tratate in faza de proiectare.[5]

Programatorii si furnizorii de servicii pot de asemenea sa fie separati din punct de vedere geografic, ceeace adauga o nou granita, care se reflecta in cresterea costului de dezvoltare si reducerea robustetii. Raspunsul la aceste provocari consta in pastrarea simplitatii atat in ceea ce priveste definirea serviciului cat si in ceea ce priveste suportul pentru standardele in domeniul serviciilor Web..[5]

2.1.2. Serviciile sunt autonome

Serviciile sunt aplicatii independente si autonome si nu clase sau componente strans legate de o anumita aplicatie. Serviciile sunt proiectate pentru a fi instalate pe o retea, posibil Internet, acolo unde ele pot fi cu usurinta integrate intr-o aplicatie in care este nevoie de ele. Serviciile nu trebuie sa cunoasca nimic despre clienti si trebuie sa accepte cereri venite de oriunde, atata vreme cat mesajele receptionate respecta formatul recunoscut de serviciu, iar cerintele in ceea ce priveste securitatea sunt respectate.

Serviciile pot fi instalate si gestionate independent de alte servicii si de posibilele aplicatii client, iar proprietarii serviciilor por modifica interfata si functionalitatea unui serviciu in orice moment. Compatibilitatea cu versiunile anterioare este o problema importanta pentru orice sistem de calcul distribuit, cu atat mai mult in cazul serviciilor Web care sunt deschise prin definitie.Solutia la aceasta problema este rezolvata partial de simplitatea si extensibilitatea unui serviciu. Tot ceea ce stiu clientii despre un serviciu este cel fel de mesaje accepta, respectiva returneaza, respectivulserviciu.

Astfel, un serviciu poate fi modificat atata vreme cat mesajele anterioare sunt in continuare acceptate ca fiind mesaje valide. Pot fi extinse chiar si mesajele care reprezinta cereri, respectiv raspunsuri, atata vreme cat este mentinuta compatibilitatea cu mesajele anterioare.

Intrucat serviciile sunt aplicatii autonome, ele trebuie sa isi asigure singure securitatea, trebuie sa se protejeze impotriva apelurilor rau intentionate. Sistemele software care sunt instalate intr-o retea inchisa pot sa ignore in buna masura problemele de securitate, este suficient sa se bazeze pe firewallsau pe protocoale de comunicare securizate cum este protocolul SSL. In timp ce serviciile accesibile in Internet trebuie sa ia masuri de securitate mult mai stricte.[5]

2.1.3. Sunt partajate scheme si contracte, si nu implementari

Construirea de sistem software complexe robuste si fiabile este dificil de realizat. Construirea de astfel de sistem software din componente dezvoltate in limbaje de programare diferite este si mai dificila. Totusi serviciile reusesc sa ofere o solutie viabila datorita simplitatii lor. Serviciile nu sunt obiecte distribuite care folosesc mostenire, nu suporta evenimente, proprietati sau apeluri care folosesc o sesiune. Serviciile sunt doar aplicatii care primesc si trimit mesaje. Clientii si serviciile nu partajeaza altceva decat definitia mesajelor si cu siguranta nu partajeaza cod.

Tot ceea ce o aplicatie client trebuie sa stie despre un serviciu este contractul si anume: structura mesajelor acceptate si returnate ca raspuns si ordinea in care mesajele trebuie trimise. Clientii pot folosi aceste contracte pentru a compune mesaje si a trimite cereri catre un serviciu, in timp ce serviciul poate folosi contractul pentru a valida faptul ca mesajul receptionat este in formatul corect.

2.1.4. Compatibilitatea este bazata pe reguli(policy).

Clientii trebuie sa fie compatibili cu serviciul pe care vor sa il foloseasca. Compatibilitatea in acest context nu inseamna doar ca, clientii ar trebui sa inteleaga formatul mesajelor ci si ca ei ar trebuie sa fie compatibili cu alte cerinte importante, cum ar fie faptul ca informatia trebuie criptata sau ca trebuie verificat faptul ca nu s-a pierdut informatie in timpul transmisiei. In ceea ce priveste serviciile, cerintele non-functionale sunt definite prin intermediul regulilor, care nu sunt scrise ca si parte a documentatiei unui serviciu.Regulile sunt un set de declaratii care pot fi interpretate de o aplicatie si care ofera informatii despre cerinte precum securitatea si fiabilitatea. Regulile pot fi inglobate in contractul unui serviciu sau pot fi stocate intr-o baza de date specializata, de unde pot fi obtinute dinamic in timpul rularii.Contractele bazate pe reguli pot fi privite ca si o parte a documentatiei serviciului, dar in acelasi timp ele pot fi folosite de utilitare pentru a genera cod compatibil atat pentru partea de client cat si pentru cea de serviciu. De exemplu, o regula de securitate poate fi folosita pentru a genera cod care va verifica, ca mesajele primite sunt criptate, va decripta mesajele si le va trimite catre aplicatia serviciu.Separarea regulilor de contracte permite aplicatiilor client sa se adapteze dinamic pentru a respecta cerintele unui anumit serviciu. Acest lucru se va dovedi foarte util pe masura ce serviciile sunt standardizate si sunt oferite de diferiti furnizori.[5]

3. Arhitecturi distribuite

Intr-un sistem distribuit hardware-ul este important insa software-ul reprezinta elementul determinant; de componenta software depinde cum va arata un sistem distribuit. De aceea, discutia noastra privind arhitectura sistemelor distribuite se va axa pe arhitectura software.

Initial, prin arhitectura software se facea referire la structurarea software-ului pe niveluri sau module, cea mai cunoscuta fiind structura ierarhica pe module. Recent, acelasi termen este descris in termenii serviciilor oferite si solicitate intre procesele localizate pe acelasi calculator sau pe calculatoare diferite. Prin urmare, noua orientare, catre procese si servicii, poate fi exprimata prin intermediul nivelurilor de servicii. Ea este prezentata schematic in figura 3.[3]

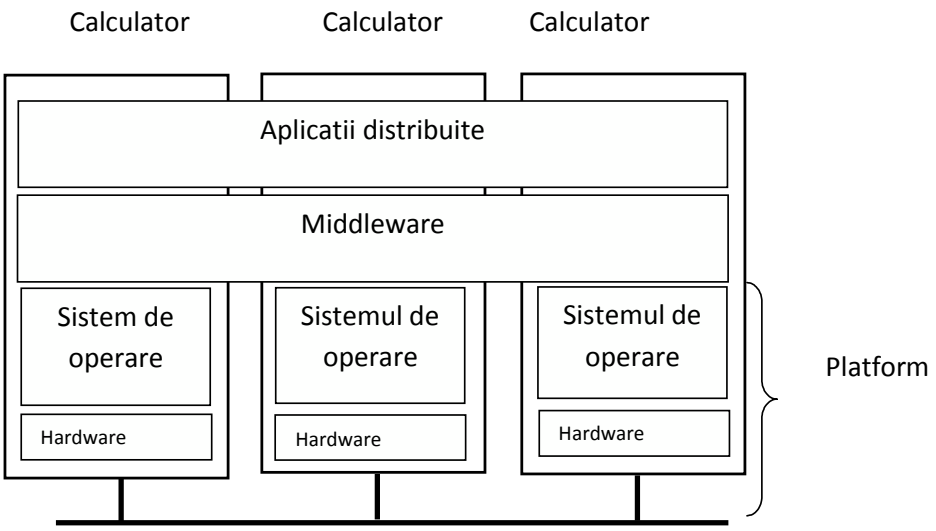

Fig. 3. Structura generala a unui sistem distribuit.[3]

Dupa cum se poate observa din figura 2.1, structura generala a unui sistem distribuit presupune trei niveluri (straturi): platforma, middleware si programele de aplicatii distribuite. Fiecare nivel ofera servicii nivelului superior. Astfel, aplicatiile distribuite apeleaza la serviciile oferite de componenta middleware care, la randul sau, beneficiaza de serviciile oferite de sistemele de operare.

3.1. Platformele hardware si software in sistemele distribuite.

Componenta hardware si nivelul cel mai de jos al software-ului sunt adesea referite impreuna prin termenul platforma. In practica ele pot fi referite separat prin platforma hardware si platforma software. Acest nivel ofera servicii nivelurilor situate deasupra sa, servicii care sunt implementate in mod independent pe fiecare calculator. El ofera interfata de programare nivelului care faciliteaza comunicarea si coordonarea dintre procese. Printre cele mai cunoscute exemple de platforme se regasesc: Intel x86/Windows, Sun SPARC/SunOS, PowerPC/MacOS, Intel x86/Linux.

Referindu-ne la arhitectura hardware, ea specifica modul in care sunt conectate calculatoarele, mai concret procesoarele. Orice sistem distribuit presupune existenta a multiple procesoare, dar care pot fi organizate in cateva moduri diferite in ce priveste interconectarea si comunicarea dintre ele. Desi in literatura de specialitate au fost prezentate numeroase scheme de clasificare a sistemelor bazate pe multiple procesoare, nici una nu a primit o recunoastere larga. In continuare vom face o succinta prezentare a catorva clasificari.

Din punctul de vedere al partajarii sau nu a memoriei, exista doua tipuri de sisteme: multiprocesoare, respectiv cele in care mai multe procesoare partajeaza memoria, si multi calculatoare, respectiv cele care care nu o partajeaza. In cazul sistemelor multi-procesoare, toate procesoarele partajeaza o singura zona fizica de memorie. Astfel, daca oricare procesor scrie valoarea 44 la adresa 1000, atunci ulterior oricare alt procesor care va citi valoarea continuta la adresa respectiva va prelua valoarea 44. In contrast, intr-un sistem multi-calculatoare, fiecare procesor dispune de propria memorie. Daca un procesor va scrie valoarea 44 la adresa 1000 a propriei memorii, ulterior un alt procesor care va citi valoarea continuta la adresa 1000 va obtine probabil o alta valoare. Cel mai comun exemplu de sistem multi-calculatoare il reprezinta o colectie de calculatoare conectate la retea.[3]

O alta clasificare grupeaza sistemele distribuite in sisteme eterogene si sisteme omogene. Aceasta clasificare este aplicata doar in cazul sistemelor multi-calculatoare. Intr-un sistem omogen, toate procesoarele sunt la fel si, in general, acceseaza zone fizice de memorie diferite dar de aceeasi capacitate, iar in cadrul retelei se utilizeaza aceeasi tehnologie. De regula, acest tip de sisteme sunt utilizate mai mult ca sisteme paralele (adica lucreaza pentru aceeasi problema). In schimb, sistemele eterogene pot contine calculatoare de tipuri diferite, interconectate prin retele de diferite tipuri. De exemplu, un sistem distribuit poate fi construit pe baza unei colectii de retele locale care utilizeaza tehnologii diferite, ele putand fi interconectate printr-un backbone bazat pe tehnologia FDDI.

Atunci cand vorbim despre platforma software, cel mai adesea se face referire la sistemul de operare. De fapt, sistemele distribuite se aseamana in buna masura cu sistemele de operare traditionale. Ele gestioneaza resursele hardware permitand mai multor utilizatori si aplicatii sa le partajeze. Prin resurse hardware se face referire la procesoare, memorie, echipamente periferice si retea. De asemenea, sistemele distribuite ascund complexitatea si eterogenitatea resurselor hardware.

Sistemele de operare pentru sistemele distribuite pot fi impartite in doua categorii: sisteme strans- cuplate(tightly-coupled) si sisteme slab-cuplate (loosely-coupled). In cazul sistemelor strans-cuplate, sistemul de operare incearca sa mentina o singura imagine, globala, asupra resurselor hardware, in timp ce sistemele slab-cuplate pot fi vazute ca o colectie de calculatoare, fiecare cu propriul sistem de operare, dar care colaboreaza.

Distinctia intre sistemele strans-cuplate si cele slab-cuplate este legata de clasificarile prezentateanterior pentru componenta hardware. Astfel, sistemele strans-cuplate, referite si ca sisteme de operare distribuite, sunt utilizate in sistemele multi-procesoare si sistemele omogene. Principala sarcina a unui sistem distribuit rezida in ascunderea complexitatii gestiunii resurselor hardware astfel incat ele sa poata fi partajate de multiple procese. In schimb, sistemele slab-

cuplate, referite adesea ca sisteme de operare de retea (NOS – Network Operating System), sunt utilizate in cazul sistemelor eterogene. Distinctia dintre un NOS si un sistem de operare traditional consta in faptul ca, pe langa gestiunea resurselor, el asigura disponibilitatea serviciilor locale clientilor aflati la distanta.

3.2. Modele arhitecturale pentru sistemele distribuite.

Dupa cum aratam in primul paragraf, una din activitatile specifice dezvoltarii sistemelor distribuite consta in proiectarea arhitecturii sistemului, respectiv diviziunea responsabilitatilor intre componentele sistemului si plasarea lor pe calculatoarele din retea. In acest sens, exista mai multe modele arhitecturale. Asupra lor ne vom opri in continuare.

Modelul client/server. Aceasta arhitectura este de departe cea mai cunoscuta si mai utilizata la dezvoltarea sistemelor distribuite.In fapt, ea presupune impartirea sarcinilor aplicatiei in procese client si procese server care interactioneaza intre ele prin schimbul de mesaje in vederea realizarii unei activitati.[3] Acest model va fi discutat pe larg in paragraful urmator.

3.2.1. Servicii furnizate de mai multe servere.

Serviciile pot fi implementate sub forma mai multor procese server rezidente pe diferite calculatoare, care vor interactiona in functie de necesitati in vederea furnizarii serviciului cerut de un proces client. Setul de obiecte care sta la baza serviciului respectiv poate fi partitionat si distribuit pe mai multe servere. De asemenea, este posibil ca mai multe servere sa intretina copii ale obiectelor respective (este vorba despre replicare), cu scopul imbunatatirii tolerantei la erori, a perfomantelor de accesare si a disponibilitatii. De exemplu, serviciul web furnizat de altavista.digital.com este partitionat pe mai multe servere care contin replici ale bazei de date.

3.2.2. Servere proxy si tehnica de caching.

Cache reprezinta tehnica de stocare a obiectelor de date recent utilizate mai aproape de locul de utilizare. Atunci cand un obiect este receptionat de un calculator, el va fi adaugat in zona de stocare cache, inlocuind eventual alte obiecte care exista deja in cache. La solicitatarea unui obiect de catre un proces client, serviciul de caching va cauta mai intai in cache pentru a pune la dispozitie obiectul solicitat, numai daca exista o copie actualizata a acestuia; altfel, o copie actualizata va fi incarcata de pe server[3]. Zonele cache pot fi dispuse pe fiecare client sau ele pot fi localizate pe un server proxy partajat de mai multi clienti.

Tehnica cache este utilizata pe scara larga in practica. Browserele Web intretin pe fiecare client un cache cu cele mai recente pagini Web vizitate si alte resurse Web. Ele utilizeaza o cerere HTTP speciala pentru a verifica daca paginile din cache sunt corespunzatoare cu cele originale de pe server inainte de a le afisa (este vorba de actualizarea lor). Serverele proxy Web ofera clientilor o zona de stocare cache partajabila ce contine resursele Web ale unui singur site sau a mai multor site-uri. In acest mod, se obtine o crestere a disponibilitatii si performantelor serviciilor Web prin reducerea incarcarii retelei si a serverului Web.[3]

3.2.3. Procese perechi.

In aceasta arhitectura toate procesele joaca roluri similare, interactionand in mod colaborativ ca perechi in vederea realizarii unei activitati sau prelucrari distribuite, fara a se face distinctia intre client si server. Codul corespunzator proceselor perechi va avea rolul de a mentine consistenta resurselor de la nivelul aplicatiei si de a sincroniza actiunile de la nivelul aplicatiei daca este necesar.

Eliminarea proceselor server determina reducerea intarzierilor aferente comunicarii inter-procese pentru accesarea obiectelor locale. De exemplu, o aplicatie poate fi conceputa astfel incat sa permita utilizatorilor sa afiseze si sa modifice interactiv o schita (de exemplu schita unui proiect pentru un autoturism realizata cu un program special, de exemplu AUTOCAD) care este partajata. Aplicatia poate fi implementata sub forma unor procese aplicatie plasate pe fiecare nod care se va baza pe straturile middleware pentru a realiza notificarea evenimentelor si comunicarea in cadrul grupului pentru a instiinta toate procesele aplicatiei despre eventuala modificare a schitei. Acest model ofera o comunicare interactiva mai buna (cu timpi de raspuns mai buni) pentru utilizatorii unui obiect distribuit partajat decat in cazul unei arhitecturi bazate pe server.[3]

4. Arhitecuri de aplicatie – arhitecturi de aplicatie mobile

4.1. Componente generale ale unei arhitecturi pentru aplicatii mobile

Aplicatiile mobile au in prezent o multitudine de proprietati, cele mai utilizate referindu-se la posibilitatea de a accesa INTERNET-ul de pe dispozitive mobile, la servicii de autentificare la distanta, precum si la o serie de servicii grupate sub titulatura Rich Voice, care cuprind de fapt transmiterea informatiilor de tip text, audio si video.[4]

Principalele componente si entitati care intra in alcatuirea unei arhitecturi mobile sunt urmatoarele:

- operatorul de servicii pentru transmisii wireless;
- furnizori de echipamente hardware;
- furnizorii de medii de dezvoltare software specializate;
- dezvoltatorii de aplicatii;
- integratorii de sisteme;

Operatorul de servicii pentru transmisiuni wireless are rolul de a asigura functionarea infastructurii de realizare a comunicatiilor fara fir, fiind echivalentul furnizorilor de servicii INTERNET pentru aplicatiile web. Ca factor de baza al arhitecturii generale, operatorul poate impulsiona dezvoltarea pietei aplicatiilor mobile prin reducerea costurilor de conectare si prin latimea de banda din ce in ce mai mare care va fi disponibila pentru utilizatori.

Furnizorii de echipamente hardware au rolul de a asigura o parte principala a infrastructurii fizice, prin dezvoltarea echipamentelor terminalelor mobile care sa utilizeze aplicatiile mobile. Cresterea puterii de procesare, a memoriei si a duratei de viata a bateriilor dispozitivelor mobile vor fi un alt factor determinant pentru evolutia industriei aplicatiilor mobile.[4]

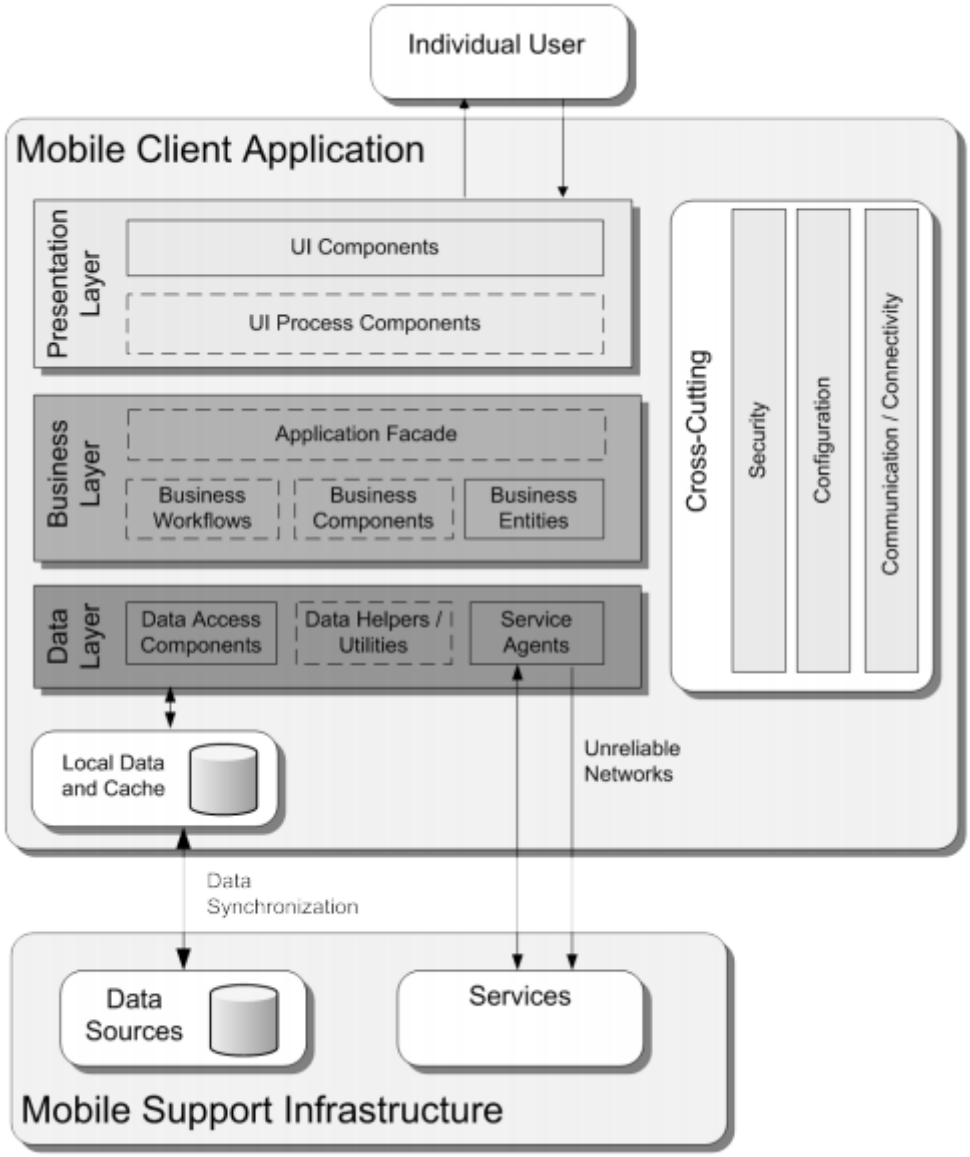

Fig.4 ilustreaza arhitectura de aplicatii mobile common rich cu componente grupate in arii de importanta.[4]

Rolul cel mai important in elaborarea aplicatiilor mobile il are componenta software , care este sustinuta de furnizorii de medii de dezvoltare specializate si de dezvoltatorii de aplicatii.In prezent, diversele standarde de comunicatie si medii de dezvoltare din ce in ce mai productive constituie un atu al industriei din domeniu. WAP, XML, VXML, .NET, J2SE, J2EE asigura transferul de date, legatura cu baza de date distribuite, accesul la rapoarte sintetice, transmiterea de informatii multimedia.

Arhitectura generala, simplificata, care sta la baza rularii aplicatiilor mobile este reprezentata conform figurii urmatoare:

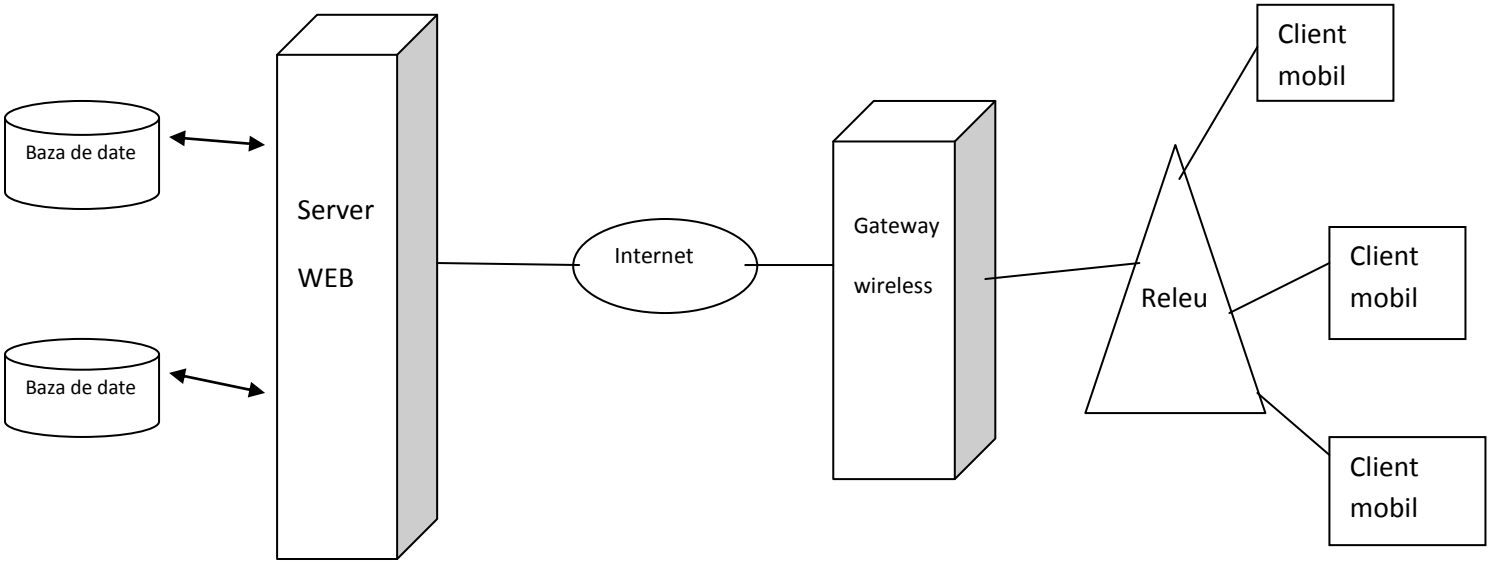

Fig. 4.1 Arhitectura generala a infrastructurii aplicatiilor mobile.

Avantajele unei astfel de arhitecturi sunt:

- procesarea operatiunilor se realizeaza pe serverul web, fara a solicita capacitatea de procesare a terminalelor mobile;
- integrarea cu aplicatiile deja existente este relativ usoara, avand in vedere ca trebuie adaptata doar comunicatia wireless;
- actualizarea datelor se poate realiza cu aceeasi viteza cu cea a INTERNET-ului clasic.

Dezavantajele arhitecturii tind sa fie ameliorate in viitorul prin marirea latimii de banda si prin realizarea de dispozitive mobile capabile sa realizeze operatiuni complexe si sa prezinte date in formate grafice avansate.

4.2. Tipuri specifice de arhitecturi specifice

Succesul aplicatiilor mobile depinde in mare masura de disponibilitatea conexiunii dintre terminalul mobil si serverul de prezentare sau serverul de aplicatii. Desi in literatura de specialitate sunt prezentate numeroase tipuri de arhitecturi, viitoarele aplicatii mobile se vor baza pe una dintre urmatoarele arhitecturi specifice:

- arhitectura in 3 straturi cu conexiune continua;
- arhitectura in 3 straturi cu conexiune intrerupta;

4.2.1. Arhitectura in 3 straturi cu conexiune continua.

Arhitectura presupune existenta unei legaturi neintrerupte intre clientul si server, toate celelalte operatiunile tranzactionale se desfasasoara in serverul de aplicatii.

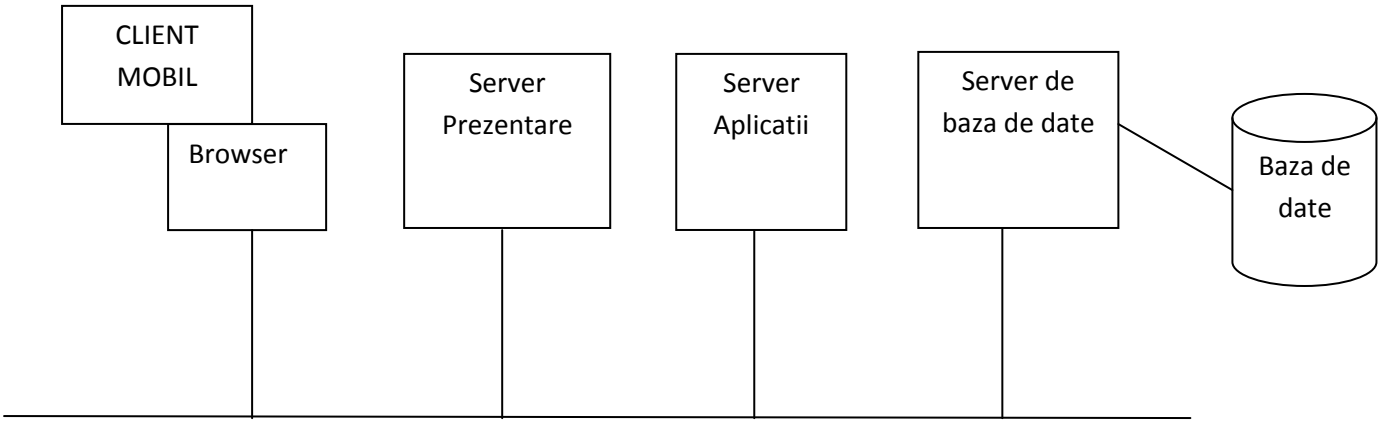

Fig. 4.2 Arhitectura bazata pe conexiune continua

Aceasta arhitectura detine urmatoarele avantaje:

- operatiunile tranzactionale sunt realizate pe serverul de aplicatii sau pe serverul de baza de date si ca atare aplicatiile pot beneficia din plin de puterea de calcul existenta pe aceste componente hardware;
- tranzactiile realizate de un client care face parte din aceasta arhitectura sunt receptate imediat de toti ceilalti clienti, prin intermediul serverului de prezentare si al conexiunii continue;
- in principiu, costul clientului mobil este redus la costul unui browser;

De asemenea, arhitectura are si unele dezavantaje:

- costul folosirii aplicatiei poate creste direct proportional cu durata de utilizare si cu pretul conexiunii pe unitatea de timp;
- in cazul in care conexiunea devine indisponibila, clientul mobil nu va avea nici o posibilitate de a beneficia de datele existente pe partea de server;

4.2.2. Arhitectura cu conexiune intrerupta.

Aceasta arhitectura, arhitectura cu conexiune intrerupta, determina existenta unui client mai complex fata de cel din arhitectura precedenta.

In cazul conexiunii intrerupte, lucrul pe client reprezinta folosirea datelor din baza de date si realizeaza tranzactiile local. Salvarea tranzactiilor in baza de date centrala se realizeaza in momentul restabilirii conexiunii.

Avantajele unei astfel de conexiuni sunt:

- clientul mobil este partial functional chiar in conditiile in care conexiunea este temporar intrerupta;
- in cazul in care costul unei conexiuni neintrerupte este mare, se poate utiliza aplicatia in mod deconectat, urmand a se folosi conexiunea doar pentru salvarea tranzactiilor pe server sau pentru actualizarea perioadica a bazei locale.

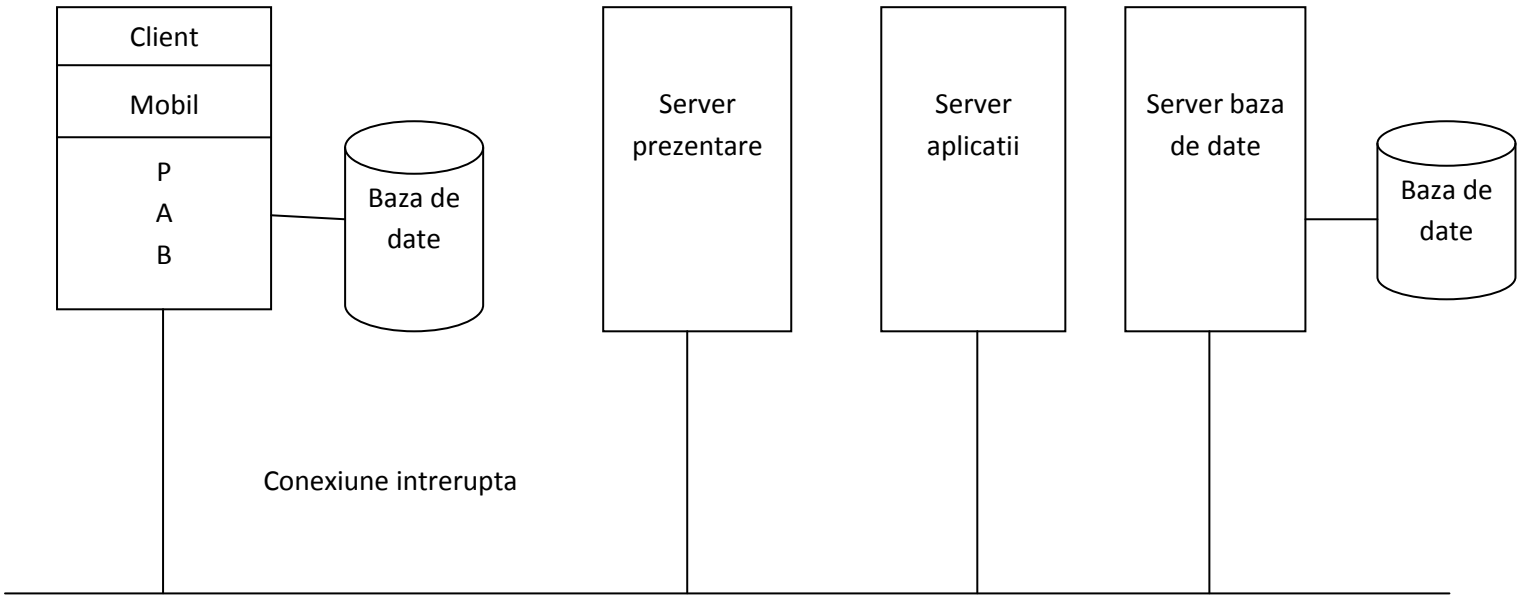

Figura 4.3 Arhitectura bazata pe conexiune intrerupta.

Arhitectura bazata pe conexiune intrerupta are urmatoarele dezavantaje:

- clientul mobil devine mai scump decat in arhitectura precedenta datorita incorporarii de elemente software capabile sa-i asigure o autonomie temporara;
- serverul trebuie proiectat astfel incat sa fie capabil sa prelucreze seturi de tranzactii efectuate off-line pe diversi clienti deconectati;
- exista posibilitatea existentei la un moment dat a mai multor imagini diferite a bazei de date pe care le au clientii deconectati;

5. Bibliografie

[1] "An introduction to Software Arhitecture" by David Garlan si Mary Shaw in 1994 editura: World Scientific Publishing Company.

[2] Arhitecturi software: http://en.wikipedia.org/wiki/Software_architecture. http://en.wikipedia.org/wiki/Software_architecture#Software_architecture_characteristics

[3] "Software Arhitecture in Practice(2nd Edition)" by Len Bass, Paul Clements, Rick Kazman in 2003 editura: SEI Series in Software Engenieering.

[4] "Mobile Applications: Arhitecture, Design, and Development:Arhitecture, Design and Development" by Valentino Lee, Heather Schneider, Robbie Schell in 2004, editura: Prentice Hall.

[5] "Essential Software Architecture" by Ian Gorton in 2006, editura: Springer.

Fig.4.1,4.2,4.3 sunt realizate in aplicatia Paint oferita de Microsoft.