MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA DE CALCULATOARE

AWARE-HOSPITAL: O ARHITECTURĂ PENTRU MEDIEREA CONŞTIENŢEI SOCIALE ÎNTR-UN SPITAL

PROIECT DE DIPLOMĂ

Autor: **Adrian DOLHA**
Coordonator: **S. L. Ing. Cosmina IVAN**

2009

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA DE CALCULATOARE

VIZAT DECAN,
Prof. Dr. Ing. Sergiu NEDEVSCHI

VIZAT ŞEF CATEDRA,
Prof. Dr. Ing. Rodica POTOLEA

Autor : **Adrian DOLHA**

AWARE-HOSPITAL: O ARHITECTURĂ PENTRU MEDIEREA CONŞTIENŢEI SOCIALE ÎNTR-UN SPITAL

1. **Enuntul temei:** *Proiectarea unei arhitecturi pentru medierea conştienţei sociale în funcţie de context, având la bază frameworkul JCAF. Se vor implementa două aplicaţii corespunzătoare diferitelor nivele ale arhitecturii AWARE-HOSPITAL. Aplicaţia HospitalServer implementează serviciul conştienţei contextului. Acest serviciu este utilizat de aplicaţia mobilă HospitalPhone, care afişează informaţii privind contextul de lucru al clinicienilor dintr-un spital.*

2. **Continutul proiectului:** *Introducere, Studiu bibliografic şi concepte introductive, Analiza şi proiectarea arhitecturii AWARE-HOSPITAL, Implementarea,Utilizarea sistemului, Concluzii şi direcţii de dezvoltare, Bibliografie, Anexe*

3. **Locul documentatiei:** Universitatea Tehnică din Cluj Napoca

4. **Data emiterii temei:** 1 noiembrie 2008

5. **Data predării:** 19 iunie 2009

Semnătura autorului _____
Semnătura coordonatorului _____

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA DE CALCULATOARE

Declaraţia autorului,

Subsemnatul Adrian DOLHA, student al Facultăţii de Automatică şi Calculatoare, Universitatea Tehnică din Cluj-Napoca, declar că ideile, analiza, proiectarea, implementarea, rezultatele şi concluziile cuprinse în acest proiect de diploma constituie efortul meu propriu, mai puţin acele elemente ce nu îmi aparţin, pe care le indic şi recunosc ca atare.

Declar de asemenea ca lucrarea de faţă este originală şi nu a mai fost niciodată prezentată sau depusă în alte locuri sau alte instituţii.

Data: 19 iunie 2009

Autor: **Adrian DOLHA**
Semnătura:_____

CUPRINS

Capitolul 1 Introducere

Conştienţa contextului este un concept prin care aplicaţiile pot să descopere şi să utilizeze informaţia contextuală. Cu toate că subiectul acesta a fost introdus de ceva vreme, rămâne un subiect de cercetare vast şi interesant. Conştienţa contextului are aplicaţie mai ales în domeniul medical, motiv pentru care sistemele medicale vor integra noi paradigme de calcul în anii care urmează.

Conştienţa contextului reprezintă o temă de cercetare care se referă la domeniul sănătăţii ca la un domeniu interesant şi bogat pentru dezvoltarea aplicaţiilor. Conştienţa contextului poate fi utilizată pentru realizarea conştienţei sociale într-un grup de indivizi ca şi membrii din personalul unui spital. Este foarte important ca membrii grupului să dispună de conştienţa socială despre ceilalţi colegi de lucru, pentru realizarea comunicării şi cooperării optime între membrii grupului. Această sarcină se poate dovedi destul de dificilă, dacă persoanele sunt mobile şi răspândite în locuri diferite.

AWARE este o arhitectură generală pentru medierea conştienţei sociale într-un spital, iar AwarePhone este o aplicaţie mobilă construită pe baza acestei arhitecturi. Infrastructura contextului care stă la baza arhitecturii AWARE este implementată de frameworkul JCAF. Pornind de la această arhitectură şi de la aplicaţia AwarePhone, lucrarea de faţă prezintă proiectarea arhitecturii AWARE-HOSPITAL şi implementarea aplicaţiilor HospitalServer şi HospitalPhone, pentru ilustrarea acestei arhitecturi.

AWARE-HOSPITAL reprezintă, la fel ca AWARE, o platformă generală pentru medierea conştienţei sociale în funcţie de context într-un spital. HospitalServer este o aplicaţie Java pentru implementarea serviciului conştienţei contextului, iar HospitalPhone este o aplicaţie mobilă care foloseşte serviciul conştienţei oferit de aplicaţia HospitalServer. Ambele aplicaţii implementează diferite nivele ale arhitecturii AWARE-HOSPITAL.

1.1 Motivaţia

Într-un mediu colaborativ precum spitalul este util şi important ca personalul să aibă cunoştinţă despre contextul de lucru al celorlalţi colegi, pentru a putea coopera cu ei în condiţii optime şi cu cât mai puţine întreruperi şi distrageri de la activităţile lor curente. De exemplu, ar fi extrem de util pentru un medic să cunoască în orice moment locaţia celorlalţi colegi de lucru şi activitatea în care sunt implicaţi la acel moment. Pentru îndeplinirea acestor sarcini este necesară dezvoltarea unei aplicaţii conştiente de context.

Având în vedere caracterul mobil al personalului este intuitiv ca aplicaţiile să ruleze pe dispozitive mobile ca şi telefoanele celulare, motiv pentru care AwarePhone este o aplicaţie mobilă de acest gen.

Arhitectura AWARE referită mai sus este o arhitectură client-server, dar frameworkul JCAF de la baza infrastructurii contextului a fost conceput în mod special pentru un model distribuit peer-to-peer. De asemenea, serviciul contextului oferit de frameworkul JCAF este disponibil doar prin Java RMI, tehnologie prea complexă pentru dispozitivele mobile. Din aceste motive arhitectura AWARE-HOSPITAL este proiectată pe baza unui model peer-to-peer hibrid. Totodată, arhitectura are la bază un framework JCAF extins care să furnizeze serviciul contextului printr-o comunicaţie la nivelul TCP, prin intermediul socketurilor.

Începând cu MIDP 2.0, aplicaţiile mobile dispun de interfaţă de comunicaţie prin intermediul socketurilor ceea ce aduce o serie de avantaje având în vedere resursele limitate şi

puterea redusă de procesare a dispozitivelor mobile. Astfel, HospitalPhone este o aplicaţie mobilă care utilizează avantajele oferite de MISP 2.

1.2 Obiectivele lucrării

Lucrarea de faţă are ca scop proiectarea unei arhitecturi pentru medierea conştienţei sociale în funcţie de context într-un spital şi construirea aplicaţiilor HospitalServer şi HospitalPhone pentru ilustrarea acestei arhitecturi.

Arhitectura AWARE-HOSPITAL îndeplineşte următoarele cerinţe:

- suportă medierea conştienţei sociale în funcţie de context;
- are la bază un model peer-to-peer hibrid;
- este construită pe mai multe nivele, fiecare nivel îndeplinind un serviciu specific, într-un mod cât mai independent faţă de celelalte nivele;
- oferă anumite servicii precum: serviciul contextului, serviciul conştienţei, serviciul mesageriei, serviciul administrării personalului;
- infrastructura contextului este implementată de frameworkul JCAF extins care oferă serviciul contextului prin intermediul comunicaţiei la nivelul socketurilor.

HospitalServer este o aplicaţie Java care urmează să fie instalată pe staţii din diferite săli ale spitalului. Aplicaţia HospitalServer îndeplineşte următoarele cerinţe:

- implementează nivelul arhitecturii AWARE-HOSPITAL corespunzător serviciului conştienţei contextului;
- utilizează serviciul contextului oferit de frameworkul JCAF;
- monitorizează contextul specific unei săli de spital: cine intră, cine iese şi ce activităţi desfăşoară;
- furnizează serviciul conştienţei celorlalte aplicaţii HospitalPhone.

Aplicaţia HospitalPhone este o aplicaţie J2ME care rulează pe telefoanele mobile puse la dispoziţia membrilor din personalul spitalului. Aplicaţia HospitalPhone îndeplineşte următoarele cerinţe:

- implementează nivelul aplicaţie al arhitecturii AWARE-HOSPITAL;
- foloseşte serviciul conştienţei furnizat de HospitalServer;
- comunică prin intermediul socketurilor cu aplicaţia HospitalServer pe baza protocolului specific HPP, pentru preluarea informaţiilor contextuale;
- oferă utilizatorului conştienţa socială a colegilor de echipă privind următoarele informaţii contextuale: activitatea, locaţia şi statusul personalului;
- pune la dispoziţia utilizatorului o interfaţă pentru transmiterea şi recepţionarea mesajelor.

1.3 Structura lucrării

Lucrarea este organizată în mai multe părţi. În al doilea capitol sunt descrise conceptele teoretice care au stat la baza acestei lucrări. Capitolul începe cu un studiu asupra conşienţei contextului, continuă cu abordările pe această temă în domeniul medical, apoi este prezentat conceptul conştienţei sociale cu un studiu de caz într-un spital, iar în ultima parte este descrisă arhitectura AWARE pe baza căreia s-a dezvoltat AWARE-HOSPITAL.

Al treilea capitol prezintă proiectarea şi analiza arhitecturii AWARE-HOSPITAL. Este descris frameworkul JCAF care a stat la baza arhitecturii, apoi componentele arhitecturii şi rolul lor, iar la urmă este prezentat modelul peer-to-peer hibrid al arhitecturii.

Al patrulea capitol cuprinde detaliile de implementare ale aplicaţiilor HospitalServer şi HospitalPhone. În prima parte sunt prezentate principalele tehnologii folosite pentru implementare. În a doua parte se descriu principiile procesului de dezvoltare a unei aplicaţii mobile J2ME. A treia parte cuprinde detaliile de implementare pentru aplicaţia HospitalServer, iar a patra parte pentru HospitalPhone. Deoarece aplicaţiile fac parte dintr-o reţea peer-to-peer, în ultima parte este descrisă modalitatea în care poate fi configurată reţeaua utilizând fişierele de configurare. Tot în ultima parte este descris protocolul de comunicaţie HPP (HospitalPhone Protocol), prin care comunică aplicaţiile HospitalServe şi HospitalPhone.

Capitolul cinci se ocupă de evaluarea şi utilizarea sistemului AWARE-HOSPITAL. Capitolul şase prezintă concluziile lucrării de faţă şi principalele direcţii de dezvoltare care ar putea fi urmate în viitor. În sfârşit, ultimul capitol cuprinde referinţele bibiliografice utilizate pentru realizarea acestei lucrări.

Capitolul 2 STUDIU BIBLIOGRAFIC ŞI CONCEPTE INTRODUCTIVE

2.1 Conştienţa contextului

Abordările în ştiinţa calculatoarelor din ultimii 50 de ani poate fi legată de relaţia dintre calculatoare şi oameni. La început, mai mulţi oameni împărţeau un singur calculator, apoi ideea ca fiecare utilizator să aibă un singur calculator a schimbat modul în care oamenii au folosit sistemele de calcul. În ultima decadă, aceasta s-a schimbat mai departe într-o relaţie în care un singur om are mai multe calculatoare, sau cel puţin dispozitive cu posibilităţi procesare. Aceasta a introdus era calculului omniprezent. [2]

Pornind de la o interacţiune om calculator calculul omniprezent descrie fenomenul interacţiunii în context cu artefacte şi medii care sunt dotate cu capacităţi de procesare şi comunicaţie. Cheia pentru a înţelege astfel de sisteme şi utilizarea lor este observaţia că oamenii interacţionează implicit în context cu mediile lor. Sarcina de a face disponibilă informaţia legată de context componentelor din sistemele de calcul a devenit o condişie prealabilă în calculul omniprezent. [2]

Conştienţa contextului joacă un rol central în cercetarea având ca temă calculul omniprezent. O astfel de cercetare ridică întrebări precum achiziţia contextului, reprezentarea contextului, distribuşia şi abstractizarea sau concepte de programare, suport de dezvoltare, şi implicaţii în interacţiunea om calculator în general.[2]

Pornind de la acestea, sistemele conştiente de context oferă noi oportunităţi pentru dezvoltatorii de aplicaţii şi pentru utilizatori prin colectarea informaţiei privind contextul şi adaptarea comportamentului sistemelor în conformitate cu această informaţie. În mod deosebit în combinaţie cu dispozitivele mobile aceste mecanisme sunt de o mare importanţă.

2.1.1 Definiţia contextului

În calculul omniprezent termenul de context joacă un rol central şi este diferit faţă de înţelegerea contextului din alte domenii, dar nu a fost stabilită nici o definiţie generală până în momentul de faţă. Nu este chiar atât de evident să defineşti contextul, iar în continuare ne vom concentra pe definiţia dată de Dey.[5]

Anind Dey a lucrat în vederea oferirii unui suport arhitectural pentru aplicaţiile conştiente de context si a ajuns la următoarea definiţie generală a contextului:

”Contextul este orice informaţie care poate fi folosită să caracterizeze situaţia unei entităţi. O entitate este o persoană, loc, sau obiect care este considerat relevant în ce priveşte interacţiunea dintre utilizator şi aplicaţie, inclusiv utilizatorul şi aplicaţia înşişi.”[5]

Punctul central al acestei definiţii centrată pe aplicaţie este că informaţia care descrie situaţia unei entităţi este contextul. Această definiţie leagă contextul întotdeauna de o entitate. Entitatea însăşi este privită ca ceva relevant interacţiunii dintre utilizator şi aplicaţie.

Considerând aplicaţia, şi implicit starea unei aplicaţii, drept o entitate care este caracterizată de context este introdusă o buclă de reacţie. O schimbare în aplicaţie duce în mod inevitabil la o schimbare în context, probabil ca o reacţie la schimbarea unei situaţii. Pentru anumite tipuri de aplicaţii şi scenarii aceasta este o modalitate elegantă de a influenţa contextul, totuşi în alte domenii acest ciclu poate creşte complexitatea modelului şi eventual a

implemenntării.[1]

Următoarea definiţie pentru sisteme conştiente de context introduce obiectivul utilizatorului ca un concept, care este deasemenea context (deoarece descrie situaţia utilizatorului), dar este folosit să determine relevanţa informaţiei şi a serviciilor. Aceasta ilustrează faptul că este greu să priveşti contextul în izolare.

”Un sistem este conştient de context dacă foloseşte contextul pentru a furniza informaţie relevantă şi/sau servicii utilizatorului, unde relevanţa depinde de obiectivul utilizatorului.”[1]

Accentul sistemelor conştiente de context cade pe sisteme care furnizează informaţii şi servicii dependente de context. În munca lor includ aplicaţii care captează şi etichetează informaţia în fundal şi sisteme unde contextul este utilizat pentru a descrie interacţiunea şi interfaţa, în timp ce furnizează mereu aceeaşi informaţie.[3]

Astfel, scopul principal al calculului conştient de context este colectarea informaţiei despre contextul curent (mediul înconjurător sau situaţia utilizatorului şi dispozitivului) şi să ofere informaţie relevantă şi servicii în confromitate. Contextul poate fi împărţit mai departe în primar şi secundar. Contextul primar sau de nivel jos se referă la caracteristicile mediului şi poate fi obţinut direct prin senzori, e.g locaţia, timpul, obiectele din apropiere, lăţimea de bandă a reţelei, orientarea, nivelul luminii, sunetul, şi temperatura. [3]

Senzorii pot măsura fie parametri fizici din mediu sau informaţie logică obţinută de la gazdă (timpul curent, celula GSM, acţiunea selectată, etc.). Senzorii sunt catalogaţi fizici sau logici în funcţie de aceste caracteristici. Nivelul secundar sau de nivel înalt se referă la un context mai abstractizat, derivat din contextul primar: situaţia socială a utlizatorului, activitatea, starea mentală.

2.1.2 Achiziţia contextului

Sistemele conştiente de context pot fi implementate în diferite moduri. Abordarea depinde de cerinţele speciale precum locaţia senzorilor (local sau remote), numărul posibil de utilizatori, resursele disponibile ale dispozitivelor utilizate sau facilităţi pentru extinderea viitoare a sistemului. În plus, metoda de achiziţionare a contextului este foarte importantă în proiectarea sistemelor conştiente de context pentru că predefineşte într-o anumită măsură stilul arhitectural al sistemului.[3]

În continuare, descriem succint tipurile de context primar care poate fi extras utilizând senzori:

- **Locaţia**: În recunoaşterea locaţiei există practic două abordări: fie dispozitivul portabil poate sa îşi localizeze singur poziţia sau sistemul poate localiza dispozitivul sau persoana. Există diverse abordări pentru localizarea în interior sau în exterior, dar nu există metode uniforme pentru ambele situaţii. Pentru exterior alegerea evidentă este GPS, dar în interior trebuie folosită altă tehnică. De obicei, e uşor să setezi o reţea de unde ultrasonice sau radio.
- **Timpul**: Putem obţine timpul cu uşurinţă din ceasul calculatorului. De multe ori locaţia este însoţită de mărci de timp, astfel putându-se urmări mişcările utilizatorului.
- **Lăţimea de bandă**: Este important, dar dificil să realizezi adaptarea la schimbările lăţimii de bandă.
- **Orientarea**: Schimbarea orientării poate fi recunoscută prin senzori de mişcare, sau se poate folosi o cameră pentru a urmări orientarea utilizatorului. De exemplu, Newton MessagePad foloseşte un senzor simplu de orientare, bazat pe două comutatoare de mercur.

- **Sunetul**: Sunetul poate fi înregistrat de microfoane.
- Altele: nivelul luminii, înclinaţia şi vibraţia, temperatura, presiunea, orare, etc.

În extragerea contextului primar în mod normal datele obţinute de la senzori sunt preprocesate. Unităţile de derivare a contextului de nivel înalt nu au acces direct la senzori, motiv pentru care informaţia senzorială trebuie să fie reprezentată într-o formă inteligibilă, independent de senzori. Un alt motiv pentru preprocesare este supraîncărcarea. Sarcinile tipice ale preprocesării sunt următoarele:

- **normalizarea** (scalarea măsurătorilor senzorilor a.î. să suporte o metrică uniformă),
- **prelucrarea datelor** (diferenţele minore în valorile variabilelor sunt netezite, e.g. dimensionalitatea este redusă sau data coninuă este discretizată),
- **transformări logice** (creare de noi variabile e.g. prin combinarea diferitelor măsurători senzoriale).

În timp ce contextul primar e dedus relativ uşor, deducerea contextului secundar reprezintă o sarcină mai dificilă. Inferarea informaţiei de nivel înalt precum situaţia socială sau starea mentală e o misiune destul de dificilă. Există trei abordări. În primul rând, se poate folosi o tehnică bazată pe tehnologia camerei sau procesarea imaginii. Această abordare oferă informaţie preţioasă despre acşiunile utilizatorului sau starea lui mentală (expresii, gesturi, mişcări), dar aceste caracteristici sunt interpretate drept context primar şi e necesară o prelucrare ulterioară. În plus, extragerea acestor caracteristici contextuale din imagini introduce costuri de calcul şi e o sarcină dificilă.[3]

În al doilea rând, putem consulta calendarul utilizatorului pentru a afla ce ar trebui să facă utilizatorul la un moment dat.Totuşi, acaeastă metodă este potrivită doar pentru a bănui activitatea utilizatorului, dar nu este o metodă de încredere pentru a deduce locaţia utilizatorului sau obiectele din apropiere.

În al treilea rând, există diverse tehnici de inteligenţă artificială, care pot fi utilizate să combine diverse caracteristici ale contextului primar pentru inferarea contextului de nivel înalt.

2.1.3 Modelarea contextului

Este nevoie de un model al contextului pentru a defini şi stoca datele context într-o formă procesabilă de maşină. Dezvoltarea unei ontologii flexibile şi utilizabile care să acopere o gamă largă de contexte posibile este o sarcină provocatoare. Strang şi Linnhoff-Popien au rezumat cele mai relevante abordări în modelarea contextului, care sunt bazate pe structuri de date folosite pentru reprezentarea şi schimbarea informaţiei contextuale în sistemul respectiv.[5]

- **Modele Cheie-Valoare**. Aceste modele reprezintă cea mai simplă structură de date pentru modelarea contextului. Sunt utilizate frecvent în diverse frameworkuri de servicii, unde perechile cheie-valoare sunt folosite pentru a descrie caracterisiticile unui serviciu. Descoperirea unui serviciu este aplicată apoi folosind algoritmi de potrivire care folosesc perechi cheie-valoare.
- **Modele cu scheme de marcare**. Toate aceste modele folosesc o structură de date ierarhică constând din etichete de marcare cu atribute şi conţinut. Profilurile reprezintă modele tipice cu scheme de marcare. Exemple tipice pentru astfel de profiluri sunt Composite Capabilities/Preference Profile si User Agent Profile, care sunt codate în RDF/S.
- **Modele grafice**. Limbajul unificat de modelare (UML) este de asemenea potrivit pentru modelarea contextului. Există diferite abordări unde aspectele contextuale sunt

modelate folosind UML, e.g., Sheng şi Benatallah(2005). Altă abordare a modelării include o extensie a Modelării Rolului Obiectelor (ORM) prin informaţia contextului prezentată în Hendricksen (2003).

- **Modele orientate pe obiecte**. Modelarea contextului prin utilizarea tehnicilor orientate pe obiect oferă utilizarea tuturor beneficiilor precum încapsularea, reutilizabilitatea, moştenirea). Abordările existente folosesc diferite pentru a reprezenta diferite tipuri de context (precum temperatura, locatia, etc.) şi încapsulează detaliile procesării şi reprezentării contextului. Accesul la context şi procesarea logică a contextului este furnizată prin interfeţe bine definite. Hydrogen (2002) foloseşte un astfel de model orientat pe obiecte.
- **Modele bazate pe logică**. Acest tip de modele oferă un nivel înalt de formalitate. Faptele, expresiile şi regulile sunt folosite pentru a modela contextul. Un sistem bazat pe inferenţe logice este folosit ulterior pentru a administra termenii menţionaţi şi pentru a permite să adauge, actualizeze sau să şteargă fapte noi. Procesul de inferenţă poate fi utilizat pentru a deriva noi fapte bazat pe regulile existente în sistem. Informaţia contextuală trebuie să fie reprezentată într-o manieră formală ca şi fapte. Una din primele abordări de acest gen a fost publicată de McCarthy şi Buvac (1997).
- **Modele bazate pe ontologii**. Ontologiile reprezintă o descriere a conceptelor şi relaţiilor. De aceea, ontologiile sunt un instrument foarte promiţător pentru modelarea informaţiei contextuale datorită expresivităţii lor şi posibilităţii de aplicare a tehnicilor ontologice de deducţie. Diferite frameworkuri conştiente de context folosesc ontologii.

Concluzia evaluării prezentată de Strang şi Linnhoff-Popien (2004), bazată pe cele şase cerinţe, arată că ontologiile sunt cele mai expresive modele şi îndeplinesc majoritatea cerinţelor. Korpipaa (2003) prezintă nişte scopuri şi cerinţe în proiectarea ontologiei contextului:

- **simplitate**: expresiile şi relaţiile folosite trebuie să fie cât de simple posibil pentru a simplifica munca dezvoltatorilor aplicaţiei
- **flexibilitate şi extensibilitate**: ontologia ar trebui să suporte simpla adăugare a elementelor noi de context şi relaţii
- **generalitate**: ontologia nu ar trebui limitată la un tip special de context atomic ci mai degrabă să suporte difgerite tipuri de contextual
- **expresivitate**: ontologia trebuie să permită descrierea a cât mai multe stări posibile ale contextului prin detalii arbitrare.[5]

Numeroase unelte sunt disponibile pentru a defini reprezentări declarative şi pentru a publica ontologii dezvoltate de World Wide Web Consortium, e.g., Limbajul de descriere a resurselor (RDF) şi Limbajul ontologiilor Web (OWL). Un singur atom context poate fi descris printr-o serie de atribute. Cele mai evidente sunt:

- **Tipul contextului**. Tipul contextului se referă la categoria contextului precum temperatura, timpul, viteza, etc. Acest tip de informaţie poate fi folosit ca parametru pentru interogarea contextului sau pentru subscriere, e.g., subscribeToChanges(”temperature”). Este important să se folosească nume semnificative, deoarece odată cu creşterea sistemului, unele nume ar putea să nu mai fie unice. De exemplu, tipul poziţie ar putea aparţine unui dispozitiv mobil sau unei persoane. O soluţie este utilizarea numelor în cascadă.
- **Valoarea contextului**. Valoarea contextului înseamnă valoarea primă obţinută de senzor. Unitatea depinde de tipul contextului şi de senzorul aplicat, e.g., grade Celsius, mile pe oră, etc.

În cele mai multe cazuri, tipul contextului şi valoarea contextului nu sunt informaţii suficiente pentru a construi un sistem conştient de context care să fie funţionabil. Atribute adiţionale folositoare ar putea fi:

- **Mostră de timp**. Acest atribut descrie momentul în care a fost captat contextul. Este nevoie de aceste mostre pentru a crea, de exemplu, o istorie a contextului şi pentru a trata conflictele contextuale.
- **Sursa**. Un câmpa care conţine informaţii privind modul în care informaţia a fost colectată. În cazul un senzor hardware ar putea reţine ID-ul senzorului.
- **Confidenţa**. Acest atribut descrie gradul de siguranţă al acestui tip de context. Nu toate sursele oferă date acurateţe în informaţii.

Parte a unui model flexibil al contextului este un vocabular extensibil pentru a lucra cu descrieri abstracte mai degrabă decât cu date tehnice. În acest mod simplifică descrierea diferiţilor atomi context şi instanţe ale contextului.[5]

2.1.4 Distribuirea contextului

Într-un mediu de calcul omniprezent contextul este distribuit intrinsec. Artefactele în lmea reală sunt fizic distribuiţi. Luând în considerare definiţia originală a contextului (”contextul este ceea ce înconjoară altceva”) şi modelul artefactelor conştiente devine evident că distribuţia joacă un rol important. Distribuţia apare la nivele diferite, prima dată la un nivel conceptual unde informaţia –privită ca şi context- este distribuită, şi în al doilea rând la un nivel de implementare unde sistemele componente, operând cu contextul sunt de asemenea distribuite.

În implementarea sistemelor conştiente de context trebuie considerate trei zone funcţionale: achiziţia contextului, sinteza contextului, şi folosirea contextului. Cum artefactele în lumea reală sunt distribuite la fel este şi achiziţia şi folosirea contextului în calculul omniprezent. Pentru sinteza sau fuziunea contextelor nu este necesară o localizare explicită, cu toate acestea intrarea proceselor (diferite surse ale contextului) şi ieşirea sunt ancorate în artefacte şi drept urmare în spaţiu, deci acest proces este distribuit în general.[2]

Pe lângă distribuţia spaţială a contextului mai este şi o distribuţie în timp, deoarece contextele sunt observate ca şi stări decât ca evenimente aşadar timpul şi în particular intervalele joacă un rol important.

O presupunere fundamentală este că relaţia spaţială şi temporală dintre sursele de context şi utilizatorii contextului este semnificativă şi de valoare pentru dezvoltarea aplicaţiilor.

Privită din perspectiva raţionamentului anterior şi din analiza sistemelor conştiente de context implementate, poate fi observat că aspectele distribuţiei sunt centrale în realizarea unor astfel de sisteme. Pentru a uşura proiectarea sistemelor conştiente de context în calculul omniprezent la un nivel conceptual este recomandat un model de distribuţie care să incorporeze toate cerinţele. În plus pentru a simplifica implementarea şi a suporta o dezvoltare rapidă este de dorit o mapare din model în implementare. Alte preocupări sunt suportul la nivelul sistemului (e.g. middleware pentru context) şi funcţionalitatea oferită de limbajele de programare pentru a face sarcinile de distribuţie transparente utilizatorului.[2]

Principalele scopuri în proiectarea unei platforme care să uşureze dezvoltarea aplicaţiilor conştiente de context în medii distribuite sunt să permită:

- Partajarea uşoară a informaţiei contextului care este creat într-un anume artefact cu celelalte părşi componente ale sistemului.
- Observarea şi obţinerea accesului la informaţia contextului care este în jurul unei aplicaţii.

- Distribuţia spaţială şi temporară a informaţiei contextului în sistem în conformitate cu înţelegerea umană a contextului, dar într-un mod transparent dezvoltatorului.
- Furnizarea unei API minimale şi deschisă.[2]

Considerarea contextului drept ceea ce este în preajma unui artefact urmează logic aceleaşi principii ca şi modelarea contextului într-o abordare bottom-up. Informaţia contextului care este creată la un anumit punct în spaţiu şi timp – la un artefact – este vizibilă şi utilizabilă într-un anume interval de timp şi o anume distanţă – în vecinătatea artefactului.

2.1.5 Principii de proiectare a sistemelor conştiente de context

În continuare descriem principiile de proiectare. Sistemele conştiente de context pot fi implementate în mai multe moduri, în funcţie de anumite condiţii speciale. În funcţie de modul în care este colectat contextul există trei modalităţi prezentate de Chen (2004) de a obţine informaţie contextuală:

- **Accesul direct la senzori.** Această abordare este utilizată des în dispozitive cu senzori construiţi local. Programul client adună informaţia dorită direct de la aceşti senzori, i.e, nu exista strat adiţional pentru obşinerea şi procesarea datei senzoriale. Driverele pentru senzori sunt cablate în aplicaţie, deci această metodă se potriveşte doar în situaţii rare. De aceea, nu este potrivită pentru sistemele distribuite datorită naturii accesului direct care duce lipsa unei componente capabilă să administreze accesul concurent la senzori.
- **Infrastructura middleware**. Proiectarea modernă a programelor foloseşte metode pentru încapsulare şi separrare e.g., logica business şi grafica interfeţei utilizatorului. Abordarea bazată pe middleware introduce o arhitectură pe nivele a sistemelor conştiente de context cu intenţia ascunderii detaliilor de nivel jos.
- **Serverul de context.** Următorul pas logic este să permiţi clienţilor multipli accesul la surse aflate la distanţă. Această abordare distribuită extinde arhitectura bazată pe middleware prin introducerea unei componente care să administreze accesele remote. Obţinerea datelor senzoriale este mutată la acest server de context pentru a facilita accesele multiple concurente. Pe lângă reutilizarea senzorilor, utilizarea unui astfel de server are avantajul eliberării clienţilor de operaţii care cer intensiv resurse. După cum majoritatea dispozitivelor utilizate în sistemele conştiente de context sunt mecanisme mobile cu limitări ale puterii de calcul, spaţiu pe disc, etc., acesta devine un aspect important. În schimb trebuie luate în considerare protocoalele adecvate, performanţele reţelei, parametri calităţii serviciului, în proiectarea unui sistem conştient de context bazat pe o arhitectură client-server.[5]

Într-o manieră similară, Winograd (2001) descrie trei modele diferite de administrare a contextului pentru coordonarea proceselor şi componentelor multiple:

- **Widgeturile**. Derivat din elemente omonime GUI un widget este o componentă software care furnizează o interfaţă publică pentru un senzor hardware (Dey şi Abowd). Widgeturile ascund detalii de nivel jos ale achizişionării datelor şi uşurează dezvoltarea aplicaţiilor datorită reutilizabilităţii lor. Datorită încapsulării din widgeturi este posibilă interschimbarea widgeturilor care furnizează acelaşi tip de context (e.g. schimbarea unui widget de frecvenţe radio printr-un widget dat de o cameră video pentru a colecta date referitoare la locaţie). Widgeturile sunt controlate în mod normal de un anumit tip de widget manager. Abordarea prin widgeturi introduce o cuplare strânsă şi creşte eficienţa, dar nu este robustă la căderea componentelor.

- **Servicii reţea**. Această abordare mai flexibilă, argumentată de exemplu în Hong şi Landay (2001), seamănă cu arhitectura serverului de context. În schimbul unui manager global pentru widgeturilor sunt folosite tehnici de descoperire pentru a găsi serviciile reţelei. Această abordare bazată pe servicii nu este la fel de eficientă ca şi o arhitectură widget datorită componentelor complexe bazate pe reţea, dar furnizează robusteţe.
- **Modelul tablei negre**. În contrast cu perspectiva centrată pe proces a widgeturilor şi a modelului orientat pe servicii, modelul tablei negre reprezintă o perspectivă centrată pe date. În această abordare asimetrică procesele postează mesaje într-un mediu partajat, aşa numita tablă neagră, şi subscriu la ea ca să fie notificate când apare un eveniment specific. Avantajele acestui model sunt simplitatea adaugării noilor surse de context şi uşurinţa configurării. Un dezavantaj constă în necesitatea unui server centralizat pentru a găzdui tabla neagră şi lacuna în eficienţa comunicaţiei deoarece sunt necesare două hopuri pe comunicaţie.[5]

Mai multe sisteme conştiente de context pe mai multe straturi au evoluat în ultima vreme. Multe din ele au diferenţe de ordin funcţional, locaţia şi numele nivelului, folosirea unor agenţi opţionali sau alte concepte arhitecturale. Pe lângă aceste adaptări şi modificări, este identificabilă o arhitectură comună în aplicaţiile conştiente de context. Este necesară o separare a detectării şi folosirii contextului pentru a îmbunătăţii extindabilitatea şi refolosirea sistemelor. Următoarea arhitectură conceptuală stratificată, ilustrată în figura 2.1, accentuează nivele pentru detectarea şi folosirea contextului prin adăugarea funcţionalităţilor de interpretare şi raţionare (Ailisto, 2002; Dey şi Abowd, 2001).[14]

Primul nivel constă dintr-o colecţie de diferiţi senzori. Este notabil ca prin senzori se face referinţă la orice sursă de date care poate furniza informaţie contextuală utilizabilă. În funcţie de modul de capturare a contextului, senzorii pot fi: fizici, virtuali sau logici.

Al doilea nivel este responsabil pentru obţinerea informaţiei primare a contextului. Utilizează drivere potrivite pentru senzorii fizici şi APIuri pentru senzorii virtuali sau logici. Funcţionalitatea interogărilor este adesea implementată în componente software refolosibile care fac detaliile de nivel jos ale accesului hardware transparente prin furnizarea unor metode abstracte precum getPosition(). Prin utilizarea interfeţelor pentru componente responsabile pentru tipuri egale de context aceste componente devin interschimbabile. De aceea, este posibil sa înlocuieşti un sistem RFID printr-un sistem GPS fără modificări majore în nivelul curent şi cele superioare.[14]

Nivelul de preprocesare nu este implementat în fiecare sistem conştient de context, dar poate oferi informaţie utilă dacă data în forma primară este nerafinată. Nivelul de preprocesare este responsabil pentru inferare şi interpretare a informaţiei contextuale. Senzorii din nivelul inferior returnează adesea informaţie tehnică care nu sunt potrivite pentru folosirea de către dezvoltatorii de aplicaţii. De aceea acest nivel ridică rezultatele la un nivel de abstractizare mai înalt. Transformarea presupune extracţia şi cuantizarea operaţiilor. De exemplu, poziţia exactă GPS a unei persoane poate să nu fie importantă pentru o aplicaţie, dar numele unei camere ar putea să fie.[14]

În sistemele conştiente de context constând din mai multe surse de context, atomii simpli de context pot fi combinaţi la un nivel mai înalt de informaţie la acest nivel. Procesul este cunoscut drept ”agregare” sau ”compoziţie” . O singură valoare a senzorului adesea nu este importantă pentru o aplicaţie, dar informaţia combinată poate fi mai preţioasă şi mai precisă. În această idee, un sistem este capabil să determine dacă un client se găseşte în interior sau în exterior prin analiza diferitelor date fizice precum temperatura şi lumina sau dacă o persoană este

la o întâlnire prin captarea nivelului de zgomot şi a locaţiei. Pentru a face această analiză să funcţioneze corect sunt implicate o multitudine de metode statistice şi adesea este cerută o anumită fază de antrenare.

Evident, această funţionalitate de abstractizare ar putea fi de asemenea implementată

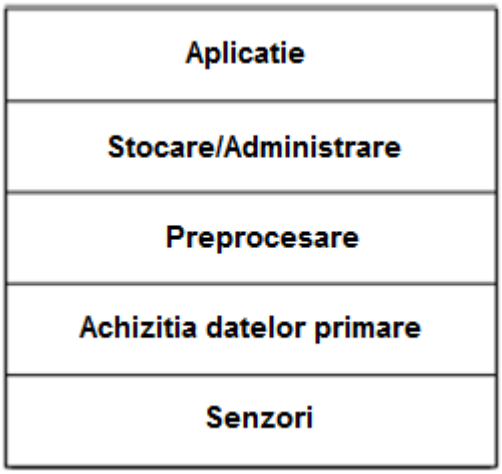


Figura 2.1 Nivelele conceptuale ale sistemelor conştiente de contextă[5]

direct de către aplicaţie. Din anumite motive este recomandat totuşi ca această sarcină să fie încapsulată şi transferată serverului de context. Încapsularea promovează refolosirea şi, prin urmare, uşurează dezvoltarea aplicaţiilor client. Prin facerea acestor agregatori accesibili de la distanţă performanţele reţelei cresc (deoarece clienţii trebuie să trimită doar o cerere pentru a obţine date de nivel înalt decât să se conecteze la diferiţi senzori) şi sunt scutite resursele limitate ale clientului. [14]

Problema detectării conflictelor care ar putea să apară atunci când se folosesc diferite surse de date trebuie de asemenea să fie rezolvată la acest nivel. De exemplu, când un sistem este notificat de locaţia unei persoane prin coordonatele telefonului mobil şi prin camera care urmăreşte acea persoană, ar putea fi dificil să se decidă care informaţie să se folosească. De multe ori acest conflict este rezolvat prin folosirea datelor adiţionale precum mostre de timp şi informaţii de rezoluţie.

Al patrulea nivel, Stocare şi administrare, organizează data colectată şi o pune la dispoziţia clientului printr-o interfaţă publică. Clienţii poat avea acces prin două modalităţi diferite, sincron şi asincron. În modul sincron clientul interoghează serverul despre schimbări prin invocări de la distanţă. De aceea, trimite mesaje cerând anumite date oferite şi aşteaptă până când primeşte răspunsul de la server. Modul asicnron funţionează prin subscriere. Fiecare client subscrie la evenimente specifice în care este interesat. La apariţia unui astfel de eveniment, clientul este notificat pur şi simplu sau este invocată direct o metodă client. De cele mai multe ori este mai potrivit modul asincron datorită schimbăriloe rapide în context. Tehnica presupune folosirea intensivă a resurselor pe măsură ce data contextuală trebuie să fie interogată des şi aplicaţia trebuie să descopere schimbările singură, folosind un fel de istorie a contextului.[14]

Clientul este realizat în al cincilea nivel, nivelul aplicaţie. Reacţia efectivă la diferite evenimente şi instanţe ale contextului este implementată la acest nivel. Uneori obţinerea informaţiei şi administrarea contextului specific aplicaţiei şi raţionarea este încapsulată în forma unor agenţi, care comunică cu serverul de context şi acţionează ca un strat adiţional între nivelul de preprocesare şi cel al aplicaţiei (Chen, 2004).

2.1.6 Frameworkuri existente

Sistemele conştiente de context capabile să se ocupe cu tipuri speciale de context sunt potrivite pentru condiţii specifice, e.g., scenariul dintr-un spital. Aceste sisteme pot fi optimizate pentru situaţiile în care sunt folosite Nu trevuie să fie flexibile şi extensibile. Pentru a simplifica efectiv dezvoltarea aplicaţiilor conştiente de context este necesar un framework abstract. O astfel de infrastructură generică nu doar că furnizează clientului acces la data contextuală, dar permite înregistrarea a noi surse heterogene distribuite. În continuare prezentăm câteva din aceste frameworkuri care diferă în funcţie de criteriile de proiectare (arhitectura, descoperirea resurselor, modelul contextului, securitate, etc.).[5]

Cea mai folosită abordare de proiectare este o infrastructură ierarhică cu una sau mai multe componente centrale folosind o arhitectură pe nivele precum cea prezentată anterior. Această abordare este utilă pentru a preveni constrângerile legate de memorie şi procesor ale dispozitivelor mobile, dar sacrifică prin aceasta robusteţea din pricina unui singur punct de cădere. Frameworkul de Administrare a Contextului (CMS) prezentat de Korpipaa conţine patru entităţi principale: managerul contextului, serverele de resurse, serviciile de recunoaştere a contextului şi aplicaţia. Serverele de resurse şi serviciile de recunoaştere a contextului sunt componente distribuite, iar managerul contextului reprezintă un server centralizat care administrează o tablă neagră. Stochează informaţia contextuală şi o distribuie mai departe clienţilor.[5]

SOCAM (Middleware Con;tient de Context si Orientat pe Servicii) este o altă arhitectură pentru construirea şi prototipizarea rapidă a serviciilor mobile conştiente de context. Foloseşte de asemnea un server central, numit interpretor de context, care obţine data contextuală prin furnizorii distribuiţi de context şi o oferă clienţilor într-o formă procesată.

O altă abordare a unui middleware centralizat şi extensibil proiectat pentru aplicaţii mobile conştiente de context este reprezentată de proiectul Sub-Structura Conştientă de Context (CASS) prezentată de Fahy şi Clarke (2004). Conţine un interpretor, un receptor de context, un motor de reguli şi un monitor pentru senzori. Monitorul ascultă actualizări ale senzorilor, care sunt localizaţi pe calculatoare distribuite numite noduri senzoriale. Informaţia obţinută astfel este stocată într-o bază de date de către monitor. Receptorul este responsabil pentru receptarea datei contextuale stocată. Ambele clase pot folosi serviciul unui interpretor. Notificatorul de schimbări este o componentă cu capacităţi de comunicaţie care anunţă clienţii asupra modificărilor în context.

Arhitectura Broker de Context (CoBrA) este o arhitectură bazată pe agenţi pentru suporta calculul conştient de context în aşa-numitele spaţiile inteligente. Spaţiile inteligente sunt spaţii fizice (camere, autovehicule, birouri) care sunt populate cu sisteme inteligente ce furnizează servicii utilizatorilor.

Proiectul Context Toolkit (Dey şi Abowd) este un alt framework conştient de context care utilizează o arhitectură peer-to-peer, dar tot mai are nevoie de un agent de descoperire centralizat unde sunt înregistrate unităţile de senzori (widgeturile), interpretorii şi agregatorii.[5]

Hydrogen (Hofer, 2002) este un alt framework cu o arhitectură stratificată, care foloseşte o metodă de achiziţie specializată pentru dispozitive mobile. În timp ce o componentă centrală este esenţială în majoritatea sistemelor distribuite conştiente de context, Hydrogen încearcă să evite această dependenţă. Distinge între un context local şi remote. Contextul remote este informaţia despre care a re cunoştinţă un alt dispozitiv, contextul local este informaţia de care are cunoştinţă propriul dispozitiv. Când dispozitvele sunt în proximitatea fizică pot să facă schimb de

context într-o manieră peer-to-peer prin WLAN, Bluetooth, etc. Acest schimb de context este numit partajarea contextului.[14]

Figura 2.2 oferă o comparaţie a frameworkurilor prezentate anterior, în funţie de diferite aspecte de proiectare. Se observă că stilul arhitectural al sistemelor conştiente de context este determinat în primul rând de metoda folosită pentru achiziţionarea contextului.

Architecture	*Sensing*	*Context model*	*Context processing*	*Resource discovery*	*Historical context data*	*Security and privacy*	
CASS	Centralised middleware	Sensor nodes	Relational data model	Inference engine and knowledge base	n.a.	Available	n.a.
CoBra	Agent based	Context acquisition module	Ontologies (OWL)	Inference engine and knowledge base	n.a.	Available	Rei policy language
Context Management Framework	Blackboard based	Resource servers	Ontologies (RDF)	Context recognition service	Resource servers + subscription mechanism	n.a.	n.a.
Context Toolkit	Widget based	Context widgets	Attribute-value tuples	Context interpretation and aggregation	Discoverer component	Available	Context ownership
CORTEX	Sentient object model	Context component framework	Relational data model	Service discovery framework	Resource management component framework	Available	n.a.
Gaia	MVC (extended)	Context providers	4-ary predicates (DAML + OIL)	Context-service module (first-order logic)	Discovery service	Available	Supported (e.g., secure tracking, location privacy, access control)
Hydrogen	Three layered architecture	Adapters for various context types	Object-oriented	Interpretation and aggregation of raw data only	n.a.	n.a.	n.a.
SOCAM	Distributed with centralised server	Context providers	Ontologies (OWL)	Context reasoning engine	Service locating service	Available	n.a.

Figura 2.2 Comparaţie între diferite frameworkuri pentru conştienţa contextului[5]

2.2 Conştienţa contextului în domeniul medical

2.2.1 Introducere

În următorii ani sistemele medicale vor integra noi paradigme de calcul. Evaluarea conştienţei contextului este un domeniu de cercetare care se referă la sănătate ca la o arie larga şi interesantă de dezvoltare.

Odată cu adoptarea unor noi tehnologii spitalele vor simţi imbunătăţiri. Chiar dacă e greu să prezici cum va arăta spitalul viitorului, aspecte precum conştienţa contextului îi vor ajuta pe profesioniştii din domeniul sănătaţii să plaseze o parte din activităţile lor calculatoarelor.

Chiar dacă a crescut numărul uneltelor computerizate ce sunt utilizate in spital(sisteme puternice de informare, conectarea rezultatelor de laborator,etc.), aceste unelte nu sunt suficiente şi noile tehnologii ar trebui să suporte o nouă viziune de abordare a spitalului din viitor. Viitorul spital inteligent va fi destul de diferit de cel actual. Introducerea unor noi unelte duce la apariţia unor probleme de cercetare provocatoare. Aceasta evoluţie cere tehnologii noi şi arhitecturi noi care să implementeze sisteme sigure şi de incredere. Se impune identificarea şi evaluarea a ceea ce poate fi făcut, în ce scop şi cum poate fi implementat. Ar putea induce chiar probleme sociale sau politice în relaţie cu problemele de securitate sau acceptare a unor astfel de sistem.[6]

Comunicarea intre profesioniştii din sănătate presupune o mare parte a activitatii lor de zi cu zi. Aceasta comunicare, directă sau indirectă, începând de la rezultatele laboratoarelor până la consultaţii complexe şi sfaturi, este utilă şi importantă, dar in acelaşi timp introduce multe întreruperi. Cooperarea intre aceşti profesionişti este indispensabilă, dar transmisia informaţiei între ei încă introduce probleme in comunicaţie. Au fost descrise erori, care uneori duc la moartea pacientului, in spitale din SUA şi din întreaga lume.[6]

Comunicarea între cadrele spitalelor poate fi înlesnită prin utilizarea unor platforme computerizate (e.g. platforme de intermediere GP, platforme de coordonare la domiciliu, etc.). Aceste unelte de coordonare vor integra noi tool-uri mobile şi vor propune noi abilităţi de cumunicare. In particular, tehnologiile curente permit introducerea conceptului de conştienţa contextului în activităţile zilnice.

Conştienţa contextului este un concept care a fost descris de ceva vreme, dar tehnologiile curente (tehnologii wireless, unelte mobile, senzori, artifacte inteligente, etc.) sunt disponibile acum pentru a permite dezvoltarea unor astfel de aplicaţii. Aceste unelte pot ajuta cadrele medicale să işi îndeplinească misiunea şi pot aduce imbunătăţiri pentru îngrijirea pacienţilor. Totodată noile tehnologii au impact asupra comunicaţiilor între agenţi. Coiera prezintă un framework pentru proiectarea interacţiunilor dintre oameni şi agenţii de calcul dintr-o organizaţie. El descrie impactul unei clase noi de interacţiune (în cazul acesta o clasă care include aplicaţii conştiente de context) într-o organizaţie. Introducerea unei clase noi de interacţiuni va influenţa nivelul interacţiunii şi comunicaţiei prin agenţi în termeni de cost şi beneficii ale indivizilor. [6]

Sistemele medicale ar putea să integreze calculul conştienţei contextului, nu numai să exploreze unelte noi, dar şi pentru a propune sisteme utile şi acceptabile.

Unităţile de îngrijire intensivă (ICU) conţin situaţii medicale complexe şi sunt un domeniu interesant pentru aceste sisteme. Un număr de cercetători au subliniat acest context al mediului de lucru ca fiind relevant în evaluarea uneltelor complexe care asistă cooperarea între colegii de lucru.

În continuare prezentăm un rezumat al principalelor proiecte de cercetare din domeniul

medical care au ca temă conştienţa contextului, precum şi a echipelor de cercetare care s-au dedicat studiului în acest domeniu.

2.2.2 Proiecte medicale in domeniul conştienţei contextului

Utilizarea calculatoarelor de mână a cunoscut o creştere simţitoare în multe din aplicaţiile medicale-Medline, de exemplu, foloseşte calculatoarele de mână ca un nou termen de indexare a tezaurului MeSH. Mulţi cercetători se concentrează pentru imbunătăţirea condiţiilor de mobilitate a profesioniştilor din domeniul sănătăţii.

Alte paradigme noi de calcul suportă de asemenea cercetarea în acest domeniu. De fapt, există mai multe teme de cercetare care impartaşesc concepte similare: conştienţa contextului, calcul omniprezent, calcul integrat, calcul penetrant, etc. Deşi aceşti termeni sunt bine cunoscuti in ştinţa calcularoarelor, utilizarea lor in aplicaţiile medicale este destul de rară. Totuşi, această temă este destul de atractivă ca să stimuleze workshop-uri sau topice ca parte a unor conferinţe internaţionale (e.g. "Pervasive Computing in Health Care", Pervasive Computing 2002; "Building Bridges: Interdisciplinary Context Sensitive Computing and Mobile Computing in Medical Context", Glasgow 2002; "Ubiquitous Computing for Pervasive Health Care Applications", Ubicomp 2003; and CFP "Pervasive Health Care" in a special issue of the *IEEE EMBS* journal (2003)). [15]

Deşi majoritatea lucrărilor în conştienţa contextului indică domeniul medical drept un câmp de cercetare important si promiţător, cu toate acestea majoritatea laboratoarelor identificate sunt implicate în cercetarea ştiinţei calculatoarelor. Domeniul medical apare eventual ca o zonă potenţială şi interesantă de testare a uneltelor şi framework-urilor lor.

Doar puţine laboratoare au fost indicate în literatură ca fiind implicate în conştienţa socială dedicată cercetării din câmpul medical. *Centre for Pervasive Healthcare*, din Danemarca, este complet dedicat acestei cercetări şi *Univeristatea din Irvine* analizează aspecte sociale legate de aplicarea conştienţei contextului in domeniul medical.[15]

În continuare vor fi detaliate câteva aplicaţii medicale conştiente de context propuse în literatură. Întâi va fi descris proiectul conştienţei contextului şi apoi se vor folosi pricncipalele caracteristici ale conştienţei contextului pentru a vedea cum tratează contextul aceste proiecte.[15]

- *Sistemul de comunicare Vocera- experimentare in spitalul St. Vincent, Birmingham, USA.*
  Sistemul de comunicare Vocera este un sistem de comunicare bazat pe insigne şi destinat utilizatorilor mobili. Este o insignă portabilă, cu un buton "apasă pentru apell", un ecran mic pentru text si capacităţi versatile de apelare voce bazat pe recunoaşterea vocii.
  Permite conversaţia hand free: raspuns hand free la apeluri şi mesaj vocal daca nu este nici un răspuns. E securizat biometric cu verificarea vorbitorului. Transmite informaţia la utilizator şi evită necesitatea utilizării unui dispozitiv de la distanţă.

- *Spitalul viitorului-Centre for pervasive HealthCare, Denmark.* Este propus un prototip context-aware care include:
  - Un pat de spital conştient de context cu un display preconstruit care poate fi utilizat de pacient (e.g. vizionarea televizorului) şi de clinicieni pentru accesarea datelor medicale. Patul ştie cine îl utilizează şi cine sau ce este langă el.
  - Un container de medicamente care este conştient de context, care este conştient de

pacienţi şi atunci când este în aproprierea unui pacient luminează containerul corespunzător cu numele pacientului.

- Un fişier electronic al pacientului(EPR) care este conştient de context.

Patul cunoaşte sora medicală, pacientul şi tava cu medicatţile astfel încât afişează informaţii relevante în acord cu contextul şi alte informaţii privind prescriptii medicale sau dosarul pacientului.

- *Software inteligent pentru spital-University of Cambridge, UK.* După un studiu al nevoilor departamentului de accidente şi urgenţe al spitalului Royal din Londra, autorii au propus un scenariu de utilizare (consultaţii de la distanţă, urmărirea pacienţilor şi echipamentelor, notificaţii ale datelor pacientului şi informaţii referitoare la conştienţa contextului) şi au implementat un prototip experimental. Prototipul permite localizarea unui membru al echipei şi posibilitatea de a iniţia o conferinţă audio-video din cel mai apropiat punct.
  Clinicienii sunt localizaţi, notificaţi de apel şi pot lua cunoştinţă de apel prin intermediul insignei active. Acest prototip este prezentat pentru demonstrarea platformei MiddleWare QoS Dream, pentru multimedia streaming reconfigurabil şi pentru programarea bazată pe evenimente.
- *Comunicatia mobila context-aware – Cicese, Mexico.* Ideea proiectului este de a împuternici anumite dispozitive mobile ca să recunoască un anumit context în care lucrează cadrele medicale dintr-un anumit spital. In particular, autorii propun o extensie IM (Mesagerie instantă) pentru a adăuga conştienţa contextului ca parte a mesajului precum circumstanţe care trebuie satisfăcute înainte ca sistemul să trimită un mesaj.
  Elementele de context includ locaţia, timpul la livrare, starea, artefactul(în special dispozitivul), etc. Este propus un prototip care oferă un serviciu de mesagerie contextual, spre exemplu, "un mesaj pentru camera 266 adresat oricărui fizician, trmiterea mesajului astăzi dupa ora 14:00".

- *MobileWard-Mobile Electronic Patient Record, Aalborg University, Denmark.* Este un prototip destinat procedurilor de dimineaţă pe secţia unui spital, capabil să afişeze lista pacienţilor şi informaţii despre pacienţi. Dispozitivul prezintă informaţii şi funcţionalităţi în acord cu poziţia asistentei şi cu momentul zilei. Proiectul simulează evenimente contextuale legate de locaţia cadrelor medicale. Pacienţii sunt aleşi din cadrul unei liste sau prin activarea un barcod la patul pacientului.

- *Reprezentarea contextului pentru un sistem de informatii al unui spital care foloseste PDAs pentru a livra informatii legate de pacient doctorului sau- Institute of design, Chicago.*
  Echipa prezintă un exemplu simplu în cadrul unui spital pentru a arăta cum poate fi descris contextul prin combinarea informaţiei contextuale din ierarhia unui mediu (e.g. Camera unui spital) persoană (pacient), şi activitate (statusul unei conversaţii doctor-pacient).[15]

Proiectele de mai sus sunt bazate pe spital, dar există şi alte proiecte specifice domeniul medical extins. Fishking si Wang propun asistarea administrării medicaţiei la domiciliu: un pad este utilizat pentru a detecta cand o sticluta de medicament este ridicată şi pusă jos, care sticluţe au fost mişcate şi câte medicamente au fost luate. Informaţii legate de numărul de pilule care trebuie luate, precum şi sugestii post-medicaţie sunt afişate în spatele acestui pad.

Floekermeier si Siegemund propun pachete inteligente de blister, echipate cu senzori. care

detecteaza retragerea pastilelor. În cazul în care pacientul uită sau nu vrea să işi ia medicamentaţia este avertizat printr-un mesaj pe telefonul său mobil.[15]

Korhonen propune o alarmă socială bazată pe senzori portabili şi pe monitorizare inteligentă. Helal propune combinarea telefoanelor inteligente Java cu casele inteligente care să ofere asistenţa pentru pacienţii in vârstă (sistem general de avertizare, conştienţă contextuală sporită, asistarea ingrijirii mobile a pacientului, notificarea unui eveniment, etc.).

Evaluare conştienţei contextului este încă provocatoare. Există trei aspecte majore ale unor astfel de provocari. În primul rand nu au fost găsite recomandări privind nevoile funcţionale ale contextului. În al doilea rând este vorba despre distanţa care exista între cercetările fundamentale asupra reprezentării contextului şi actualele prototipuri conştiente de context. În al treilea rând este vorba despre dificultăţile care apar in construirea unor sisteme computerizate pentru medierea perspectivelor umane.[15]

Majoritatea aplicaţiilor sunt prototipuri şi aplicaţii reale sunt încă greu de găsit. Aceasta din cauză că e greu să administrezi un sistem complex distribuit pentru conştienţa contextului. Analizând mai atent contextul, se poate remarca:

- se pot găsi explicaţii asupra modului in care sunt administrate situaţiile, dar e mai greu să ştii de ce au fost alese aceste setări şi cum întâlnesc acestea nevoile funcţionale. Luând în considerare activitaţile medicale in spitale, lucrările in domeniu dau unele recomandari generale (de exemplu, Bardram menţionează ca e util conceptul conştienşei contextului în particular pentru intefaţa de navigare a utilizatorului).
- Caracteristicile folosite sunt de cele mai multe ori prea simple-locaţia, timpul şi identitatea cadrelor medicale sunt cel mai des folosite atribute.
- Nu este utilizată nici o organizare a contextului deşi este recomandată în studiile fundamentale din domeniul conştienşei contextului
- cunoştinţa activităţii este utilizată în mod infrecvent. [15]

2.3 Conştienţa socială

Menţinerea unei conştienţe socială a contextului în care lucrează colegii de echipă este crucial în vederea unei bune cooperări. Pentru angajaţii care nu sunt colocaţi şi sunt mobili este destul de greu să menţii o conştientizare socială.

Mai departe prezentăm conştienţa socială şi conceptul de mediere a conştienţei sociale prin context, precum şi un studiu de caz în cadrul unui spital prin care se urmăreşte ilustrarea acestor concepte

2.3.1 Cooperarea şi comunicarea între colegii de echipă

Inerente muncii cooperative sunt niste cheltuieli suplimentare în stabilirea eforturilor de coordonare şi cooperare, care implică întreruperi şi disturbări ale colegilor de lucru. Spre exemplu, dacă o asistentă are nevoie de prezenţa unui fizician în cazul unui pacient grav bolnav cel mai adesea îl va suna, ceea ce va duce la întreruperea activităţii acelui medic.

Partajarea şi distribuirea contextului curent de lucru este un lucru esenţial pentru iniţierea unei conversaţii proprii între parteneri care cooperează. Această conştientizare socială ajută în angajarea într-un efort cooperativ. De exemplu, asistenta îşi petrece o parte din timp pentru menţinerea unei imagini mentale a locaţiilor celor mai importanţi fizicieni, a activităţilor lor şi a orarului lor pentru acea zi. Această imagine o ajută să decidă pe cine să contacteze, când, cum şi

unde.[6]

Conceptul medierii conştienţei sociale în funţie de context are ca scop minimizarea întreruperilor nedorite între colegi de muncă mobili şi răspândiţi prin asigurarea unei conştientizări sociale, realizată prin utilizarea unor sisteme conştiente de context.

Întreruperile sunt inerente într-un mediu cooperativ. Studiile muncii organizaţionale şi de birou arată că 90% din conversaţiile scurte sunt neplanificate şi prin urmare sunt potenţial întreruptive. Exista o asimetrie de bază între iniţiatorul şi receptorul unei conversaţii neplanificate: întimp ce iniţiatorul beneficiază de pe urma conversaţiei receptorul este întrerupt din activitateacurentă. Studiile arată că doar 55% din cei care sunt întrerupţi îşi continuă activitatea. O strategie comună pentru receptorii unei cereri de comunicare este să blocheze apelurile. În unele cazuri – mai ales in spitale - aceasta nu este o opţiune. Din perspectiva iniţiatorului, o problemă cheie este localizarea şi receptarea atenţiei persoanei cu care vrea să converseze. Studiile arată că 60% din telefoanele de la locul de muncă eşuează să ajungă la recipienţi pentru că persoanele respective ori nu sunt prezente, ori sunt angajate într-o altă conversaţie.[6]

În cazul spitalelor este arătat că întreruperile, conversaţiile ad hoc, şi crizele acute de administrare sunt probleme esenţiale în cadrul spitalelor moderne. Conştientizarea socială ajută la minimizarea întreruperilor şi disturbărilor în lucrul cooperativ, fenomen denumit "indiscreţie adecvată". Acest termen este opus întreruperilor neintenţionate. Conştientizarea socială este un mecanism de ajustare a întreruperilor la natura şi urgenţa sarcinilor curente. Este o activitate intens activă şi coordonată.

Heath si Luff folosesc conceptele de monitorizare şi afişare. Monitorizarea este folosită pentru a descrie acţiunea în care un actor monitorizează activ activitatea unei alte persoane din încăpere. Actţunea de monitorizare nu disturbă persoana monitorizată. Monitorizarea este un proces foarte concentrat şi selectiv. Nu este monitorizată decât partea de context relevantă pentru un actor specific. Actorul combină contextul în care lucrează cu activitatea observată pentru a crea o conştientizare socială despre statusul şi activitatea altor participanţi din încăpere.[6]

Termenul opus monitorizării este numit afişare. Afişarea descrie semnale implicite sau explicite pe care un actor dat le foloseşte pentru a arăta aspecte specifice ale situaţiei sale curente, care ar putea fi folositoare sau relevante pentru alţi actori din context. Ca şi monitorizarea, această activitate este una de natură selectivă, în care actorul decide ce doreşte să afişeze.

Cand persoane aflate în încăperi diferite cooperează într-o manieră distribuită, conştientizarea socială nu poate fi obţinuta direct, ci prin intermediul unor artefacte sociale. În cazul spitalelor sunt folosite table albe pentru a comunica informaţii întregului personal relevant dintr-o secţie. Prin combinarea cunoştinţelor despre o încăpere de lucru cu anumite indicii din mediu oamenii pot afla anumite lucruri despre contextul coechipierilor lor. Pot fi obţinute informaţii relevante prin consultarea calendarelor online, verificarea camerelor de întâlnire, sau verificarea sălii de operaţie.[6]

Una din deficienţele sistemelor de mediere a conştienţei sociale în funcţie context este că oferă doar un indiciu de context, precum locaţia fizică. De exemplu, dacă un chirurg este situat într-o sală de operaţie ar putea fi acolo să opereze, să ia un instrument, să ceară un sfat, sau altceva complet diferit. De aceea este important ca utilizatorii unui sistem de acest gen să fie caracterizaţi de mai multe indicii de context şi nu doar unul. Totodată este important ca oamenii să poată afişa indicii de context despre ei inşişi.

2.3.2 Un studiu de caz in cadrul unui spital

Cooperarea, coordonarea, întreruperile, mobilitatea şi conştientizarea socială sunt termeni cheie ai muncii într-un spital. Datorită naturii specializate a muncii medicale în spitale, cooperarea şi prin urmare coordonarea joacă un rol central în obţinerea unui flux de lucru. Mai mult, cadrele medicale sunt mobile, vizitând mai multe încăperi şi etaje pe parcursul unei zile.

Una din observatii este că se pierde timp mult în localizarea coechipierilor. Astfel că întreruperile sunt frecvente din cauza utilizării extensive a pagerelor şi telefoanelor mobile. O asistentă este deranjată de un medic care cere date despre un pacient, de o colegă asistentă care are nevoie de ajutor, sau de un pacient care sună alarma.[6]

În continuare vor fi prezentate trei situaţii de lucru frecvente din cadrul departamentului. Cele trei situaţii întâlnite sunt următoarele: ucenicia, colaborarea între profesii şi coordonarea temporală.

Ucenicia reprezintă situaţia în care un medic mai tânăr şi neexpirementat are nevoie de sfatul unui medic mai experimentat. Următorul exemplu ilustrează această situaţie:

"*Un medic tânăr tratează un pacient şi verifică o rană deschisă ca să vadă dacă este posibil un transplant de piele. Rana a sângerat mult şi este pătată spre magini. Pe de o parte ar fi mai bine să se aştepte până când rana va arăta mai bine, dar pe de altă parte transplantul ar putea să protejeze rana şi să ajute la vindecarea ei mai rapidă. Medicul mai tânăr începe să caute un medic mai experimentat şi în cele din urma găseşte unul cu un etaj mai sus şi se uită împreună la rană.*"[6]

Situaţia este una destul de tipică şi de multe ori este dificil să găseşti un medic mai experimentat prin preajmă. S-ar putea întâmpla pe deasupra să fie şi ocupat cu alte lucruri şi să nu poata fi deranjat din lucrul său. Dacă medicul mai tânăr ar fi dispus de o conştientizare socială a statusurilor şi localizărilor medicilor mai experimentaţi, l-ar fi ajutat sa aleagă pe cel mai apropiat şi mai puţin ocupat.

Colaborarea intre profesii este întâlnită frecvent datorită gradului înalt de specializare medicală. Următoarele observaţii au fost făcute despre cooperarea dintre un medic şi o asistenta:

"Un pacient vrea să fie transferat la un spital mai apropiat de locuinţa sa. Pacientul cere asistentei să îi spună când ar putea să plece. Asistenta îl sună pe medic pe pager şi aşteaptă. Nu se intamplă nimic. După ce verifică şi calendarul îşi dă seama că doctorul face vizite în acea zi. Întreabă alt medic să vadă dacă ştie despre ce vizite e vorba. Acesta îi spune că l-a văzut în acea dimineaţă şi că va coborâ în câteva minute. Asistenta îşi continuă activitatea şi mai târziu se întâlneşte cu medicul respectiv şi merg în vizită la pacient."[6]

Când diferite profesii au nevoie să coopereze, situaţia e asemănătoare cu cea a uceniciei, adică e nevoie de un ajutor urgent. În multe cazuri, totuşi, asistenta doreşte doar să reţină doctorul în acea dimineaţă, nu neapărat în acel moment. Aceasta sugerează ideea posibilităţii unor notiţe în genul "Contactează-mă când ai un moment".

Majoritatea sarcinilor dintr-un spital trebuie să fie efectuate într-o anumită ordine temporală. Pacientul trebuie să ajungă la spital înainte să poată fi tratat, pacientul trebuie anesteziat înainte de operaţie, etc. Coordonarea temporală caracterizează situaţii în care medicii şi asistentele aşteaptă ceva să se întâmple.

"*În timpul unei operaţii, unele noduri limfatice sunt îndepărtate de la un pacient pentru că este prezent un cancer malign in acele noduri. După ce au fost îndepărtate acele noduri chirurgul ia cinci mostre din părţile rănii şi le trimite la laborator. Laboratorul trebuie să testeze mostrele ca să vadă dacă muchiile sunt libere. Dacă rezultatele sunt bune atunci rana poate fi inchisă, altfel trebuie înlăturată mai multă piele din rană. În timp ce laboratorul face analizele medicul este*

ocupat cu altceva, dar verifică în mod constant să vadă dacă au sosit analizele. ”[6]

În acest caz este nevoie de o notificare atunci când se intamplă ceva. În cazul de mai sus este important ca medicul să reacţioneze la mesaj imediat, dar în alte cazuri s-ar putea să nu fie atât de urgent. Acest lucru sugerează ideea unui mecanism de prioritizare a mesajelor astfel încât mesajele mai puţin importante să nu îl întrerupă pe medic din activitatea sa.

2.3.3 Medierea conştienţei sociale in funcţie de context

Pentru medierea conştientizării sociale este folosit contextul. În cele trei situaţii prezentate mai sus, medicii şi asistentele duceau lipsa unei conştientizari a celorlalte persoane în contextul de lucru. Pentru că oamenii nu erau colocaţi, a fost imposibil pentru clinicieni să afiseze sau să monitorizeze activităţile între ei. O modalitate în care medicii şi asistentele au incercat să adreseze această situaţie a fost folosirea contextului pentru conştientizarea celorlalţi coechipieri din contextul de lucru. Calendarele medicilor erau printate şi afişate lângă o listă cu descrierea pacienţilor şi a paturilor în care pot fi găsiţi. Totuşi, pacienţii erau mutaţi frecvent şi doctorii îşi schimbau destul de des orarul, ceea ce ducea la o informaţie neactualizată.

2.4 O arhitectura existentă pentru medierea conştienţei sociale

Reflectând asupra celor enunţate în secţiunea anterioară se pune problema cum ar trebui să fie un suport tehnic mai general pentru conştienţa socială. În primul rând ar trebui să furnizeze suport general pentru medierea conştienţei sociale în funcţie de context, permiţând utilizatorilor să afişeze şi să monitorizeze informaţiile relevant. În al doilea rând ar trebui să permită utilizarea dispozitivelor multiple şi tehnologii diferite.

În vederea îndeplinirii cerinţelor anterioare colectivul de ecercetare la *Centre for Pervasive Healthcare* din Danemarca au proiectat arhitectura AWARE.

Arhitectura AWARE a fost proiectată pentru medierea conştienţei sociale în medii mobile şi distribuite şi a fost realizată în urma unei cercetări calitative a muncii cooperative din spitale. Chiar dacă arhitectura porneşte de la colaborarea mobilă din cadrul spitalelor, ideea este să se creeze platformă generică potrivită şi în alte încăperi caracterizate prin cooperarea mobilă la distanţă. Arhitectura a fost proiectată şi implementată ca un framework pentru dezvoltarea unor aplicaţii conştiente de context care să medieze conştienţa socială. Pe deasupra aplicaţia AwarePhone a fost construită pe baza acestei arhitecturi.

2.4.1 Principalele componente ale arhitecturii

Arhitectura AWARE este descrisă în figura 2.2 şi este organizată în patru nivele. Nivelul *Client* conţine aplicaţii utilizator care folosesc frameworkul şi serviciile oferite de arhitectură. Nivelul *Conştienţei* conţine *Serviciul Conştienţei*, care menţine o conştienţă a oamenilor, cum sunt legaţi din punct de vedere social şi cum pot fi contactaţi. *Serviciul Conştienţei* este descris mai amănunţit în secţiunea următoare.[8]

Nivelul *Conştienţei* conţine un serviciu de *Mesagerie*, care este un simplu borker de mesaje suportând cerinţa să fie capabil să posteze mesaje colegilor. Această funcţionalitate simplă este fundamentală pentru o comunicare şi o coordonare simplă, contribuind la menţinerea conştienţei sociale. Strict vorbind, serviciul de *Mesagerie* este o parte opţională a arhitecturii pentru că ar putea fi înlocuit de un serviciu IM, un serviciu SMS/MMS, sau un alt sistem mai

ealborat de cooperare. Totuşi, în proiectarea aplicaţiilor client pentru arhitectura AWARE, s-a dovedit a fi de mare folos integrarea unui serviciu de mesagerie în arhitectură, deoarece permite folosirea funţionalitatea mesageriei direct în aplicaţiile client, precum AwarePhone. Nivelul *Context* şi nivelele *Actuator*, respectiv *Monitor* fac parte din infrastructura contextului pe baza căruia a fost construită arhitectura AWARE, adică frameworkul JCAF.

2.4.2 Serviciul pentru conştienţa contextului

Serviciul *Conştienţei* are două mari responsabilităţi. Prima din aceste responsabilităţi este să menţină informaţia despre utilizatorii înscrişi în sistemul AWARE, despre care vor să păstreze o conştienţă socială, şi să notifice schimbările apărute în contextul utilizatorului pe baza evenimentelor. Acest lucru este realizat prin colaborarea dintre *Managerul Contactelor, Ascultătorul Entităţilor,* şi *Ascultătorul Mesajelor.* A doua responsabilitate care revine *Serviciului Conştienţei* este să intermedieze conexiunile clienţilor, folosind diferite protocoale, să răspndă la cererile lor şi să ştie cum să îi notifice asupra evenimentelor relevante. Aceasta se realizează cu ajutorul componentei *Accesul Conştienţei.* Managerul contactelor păstrează urma contactelor pentru fiecare utilizator, i.e. despre cine vrea să aibă conştineţă socială acest utilizator.

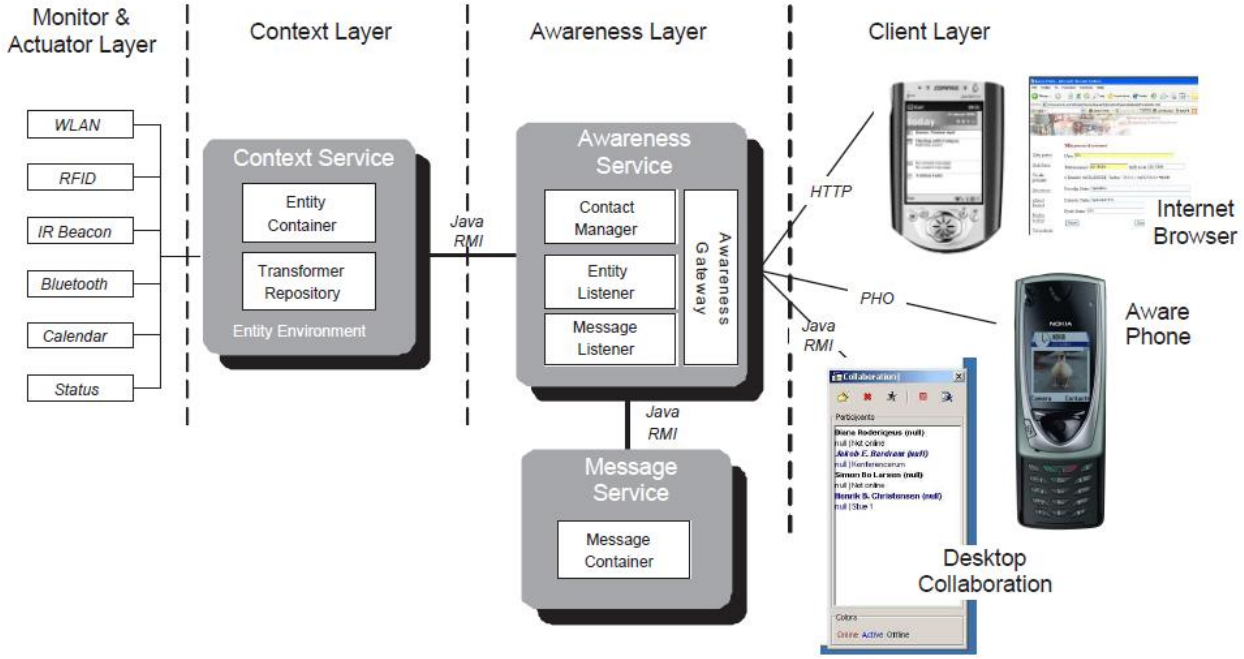


Figura 2.3 Arhitectura sistemului AWARE. [6]

Aceasta este o listă simplă a celorlalţi utilizatori din sistemul AWARE, asemănător cu lista contactelor dintr-un sistem IM.[9]

O calitate importantă a arhitecturii AWARE este infrastructura sa bazată pe evenimente. De aici, atunci când apar modificări în contextul unui utilizator sunt notificaţi clienţii relevanţi. De exemplu, dacă persoana A are persoana B în lista de contacte, atunci A este notificat dacă persoana B se mută dintr-o locaţie în alta, sau altă informaţie contextuală este modificată.

În mod similar, dacă persoana X trimite un mesaj persoanei Y, şi persoana Y citeşte mesajul, atunci această informaţie este transmisă dispozitivului persoanei X. Aplicaşiile care rulează pe dispozitiv pot astfel să trateze evenimentele în mod adecvat, i.e. actualizând interfaţa

utilizatorilor sau notificând clienţii. Această calitate a arhitecturii bazată pe evenimente este implementată de *Ascultătorul Entităţilor,* care ascultă modificări în informaţia contextuală din serviciul contextului şi de către *Ascultătorul Mesajelor,* care ascultă mesaje noi sau schimbarea statusului corespunzător mesajelor.[8]

Pentru a putea suporta diferite dispozitive client, Accesul Conştienţei este capabil să transforme cereri formulate într-un protocol specific Serviciului Conştienţei şi să traducă într-un API AWARE intern pentru a răspunde clienţilor. Acest API este o colecţie din Serviciul Contextului, Serviciul Conştienţei şi Serviciul Mesageriei. Câteva exemple de cereri:

- **getAllContacts()** returnează toate contactele pentru un utilizator folosind Serviciul Conştienţei.
- **SendMessage()** trimite un mesaj folosind Serviciul Mesageriei.
- **SetContextItem()** schimbă informaşia contextuală din Serviciul Contextului, e.g. statusul unui utilizator.

O responsabilitate importantă care revine Accesului Conştienşei este să ştie cum să contacteze un utilizator, cunoscându-i dispozitivul curent (e.g. telefon mobil), protocolul de comunicaţie şi informaţie tehnică relevantă despre adresa fizică şi numărul portului de comunicaţie. Acesta este folosit pentru notificarea prin evenimente. *Accesul Conştienţei* este proiectat pe baza şablonului de proiectare ”Strategy” având *Convertoarele de Protocol* drept pluginuri pentru acces. Aceste convertoare de protocol mapează între un protocol (e.g. HTTP) şi API folosit de serviciul care rulează în arhitectura AWARE.[10]

Protocolul PHO permite clienţilor să comunice cu serverul printr-o interfaţă socket TCP/IP. Acest protocol simplu a fost proiectat şi folosit în locul e.g. XML peste HTTP (SOAP) sau orice alt tip de interfaţă RPC/RMI din cauza limitărilor lăţimii de bandă şi puterii de procesare pe dispozitivele folosite.

Capitolul 3 ANALIZA ŞI PROIECTAREA ARHITECTURII AWARE-HOSPITAL

În acest capitol prezentăm principiile de proiectare ale arhitecturii AWARE-HOSPITAL. Pornind de la arhitectura AWARE descrisă anterior, AWARE-HOSPITAL reprezintă o altă platformă generică pentru medierea conştienţei sociale în funcţie de context, proiectată în mod special pentru îmbunătăţirea cooperării şi comunicării între clinicienii din cadrul unui spital. Cu toate că oferă suport pentru medierea conştienţei sociale ca şi platforma AWARE, arhitectura AWARE-HOSPITAL reprezintă o abordare diferită din multe privinţe, în mod deosebit prin aceea că foloseşte un model peer-to-peer în loc de unul client-server. La fel ca şi arhitectura AWARE, AWARE-HOSPITAL are la bază frameworkul JCAF, dar un framework extins care permite şi conectarea prin interfaţa socketurilor, nu doar prin Java RMI.

În continuare vor fi descrise câteva şabloane de proiectare utilizate în proiectarea arhitecturii, apoi va fi prezentat mai amănunţit frameworkul JCAF. În partea a treia se prezintă principiile de proiectare a arhitecturii AWARE-HOSPITAL şi principalele componente, iar în ultima secţiune este descris modelul hibrid peer-to-peer cu rolul fiecărei perechi.

3.1 Frameworkul JCAF(Java Context-Awareness Framework)

JCAF este o infrastructură conştientă de context bazată pe limbajul de programare Java. În plus, oferă un API de programare pentru crearea aplicaţiilor conştiente de context. Capitolul prezintă principiile de proiectare ale frameworkului, infrastructura Runtime, modelarea contextului, distribuirea contextului şi, în final, clienţii contextului.

3.1.1 Introducere

Ideea calculului conştient de context a fost un concept introdus în munca de cercetare a calculului omniprezent şi a fost subiectul unei cercetări intensive începând de atunci. Contextul se referă la situaţia fizică şi socială în care sunt integrate dispozitivele de calcul. Scopul calculului conştient de context este să capteze şi să utilizeze informaţia refiritoare la contextul unui dispozitiv pentru a furniza servicii care sunt potrivite într-o anumită situaţie.

Obiectivul în care a fost contruit JCAF este să furnizeze un framework simplu cu un set de interfeţe care să fie expresive, compacte şi mici. JCAF este un framework simplu şi robust , care poate fi extins într-un suport specializat pentru dezvoltarea aplicaţiilor conştiente de context. De aici, pe o scară largă frameworkul a fost destinat cercetătorilor, programatorilor, şi studenţilor în scopuri experimentale.[11]

3.1.2 Principii de proiectare

JCAF este o infrastructură generală, robustă, bazată pe evenimente, expresivă şi orientată pe servicii creată ăentru dezvoltarea unor aplicaţii conştiente de context. JCAF se conformează cerinţelor pe care trebuie să le îndeplinească un framework conştient de context şi mai departe vor fi subliniate doar principiile de proiectare.[7]

Practic, JCAF este împărţit în două părţi: o *Infrastructură Runtime Conştientă de Context* şi un *Framework de Programare Conştient de Context* (sau API). Principiile de proiectare ale

arhitecturii sunt următoarele:

- *Servicii Distribuite şi Cooperative* – Un serviciu contextual poate fi dedicat unui scop anume, precum administrarea contextului dintr-o casă privată. Majoritatea administrării contextului este specifică acestei case, dar ocazional s-ar putea să devină relevant pentru serviciile care rulează în alte încăperi. De aceea, o infrastructură conştientă de context ar trebui să fie distribuită şi slab cuplată, cu mijloace de cooperare într-o manieră peer-to-peer sau ierarhică.
- *Infrastructură bazată pe Evenimente* – Principala calitate a aplicaţiilor conştiente de context este abilitatea lor să reacţioneze la schimbări din mediul lor. De aici, aplicaţiile ar trebui să fie în stare să subscrie la evenimentele contextului relevant şi să fie notificate atunci când apar astfel de evenimente.
- *Securitate şi Izolare* – Data contextuală, folosită e.g. într-o locaţie medicală, trebuie să fie protejată, supusă controlării accesului, şi nedescoperită clienţilor neautorizaţi. În plus, stabilirea credibilităţii şi sursei contextului stă la baza unor anumite tipuri de aplicaţii conştiente de context. Aceste cazuri a putea cere un mecanism de autentificare pentru clienţi, chiar o legătură sigură între clienţi şi servicii. Oricum, se susţine suportarea unei securităţi adecvate într-un mediu omniprezent.
- *Extensibil* – Infrastructura ar trebui să fie extensibilă în mai multe moduri, fără să trebuiască să fie pornită din nou. În primul rând, trebuie să fie posibilă rularea, modificarea şi ştergerea serviciilor contextului. În al doilea rân, infrastructura ar trebui să suport implicarea tipurilor de context suportate prin încărcarea dinamică a definiţiilor contextului, funcţionalităţilor şi mecanismelor de achiziţie. Precum senzori noi de context. [7]

Scopul API este să uşureze proiectarea şi dezvoltarea aplicaţiilor conştiente de context pentru scopuri specifice. Aceasta duce la următoarele principii de bază pentru proiectarea şi programarea API:

- *Abstracţii de modelare libere de semantică* – Tipul informaţiei contextuale care este relevant pentru modelare şi intermediere dferă în funcţie de aplicaţii. De exemplu, într-un spital, paturile, containerul de medicamente sunt informaşii importante pentru clinicieni, dar aceasta este specific spitalelor. De aceea programatorul aplicaţiei ar trebui să fie capabil să modeleze şi să administreze data contextuală specifică diferitelor setări.
- *Calitatea Contextului* – Aplicaţiile sunt preocupate de calitatea informaţiei contextuale, inclusiv incertitudinea. Aplicaţia clinică ce încearcă să găsească datele relevante ale pacientului în timpul unei operaţii poate sugera să indice mai mult decât o singură informaţie medicală, dacă incertitudinea este prea mare. De aceea, măsurătorile calităţii pentru informaţia contextuală trebuie să fie menţinute din măsurători, prin orice transformări şi prin folosirea de către aplicaţie.
- *Suport pentru Activităţi* – Motivul pentru captarea locaţiei sau altor informaţii contextuale în mod tipic nu este pentru folosirea directă în aplicaţii, ci pentru a înlesni inferarea la nivelul activităţilor utilizatorului. De exemplu, se doreşte ca un EPR (fişier electronic al pacientului) să arate medicaţia corectă când asistenta se îndreaptă înspre pacient. De aici, frameworkul trebuie să furnizeze intermediari pentru scrierea codului specific aplicaţiilor, care pot translata modificările contextului în sugestii pentru activităţile pe care le desfăşoară utilizatorul.[7]

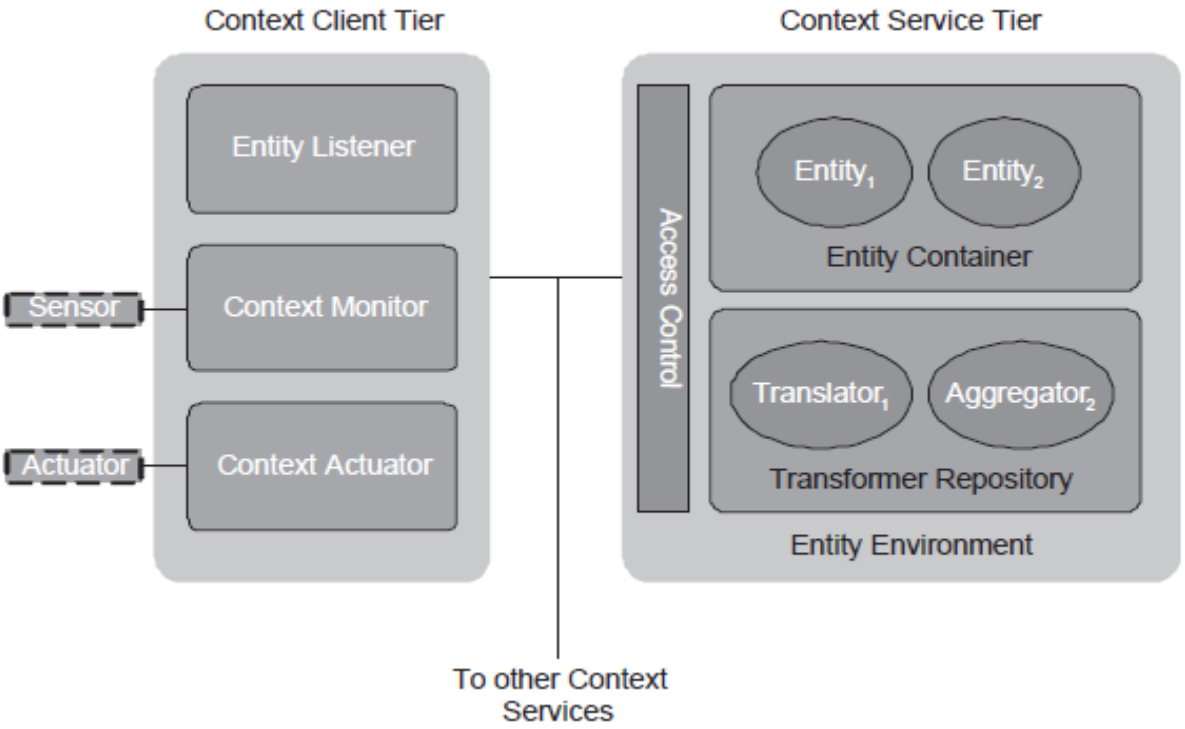


Figura 3.1 Infrastructura Runtime a frameworkului JCAF[7]

3.1.3 Infrastructura Runtime a frameworkului

Infrastructura Rutime a frameworkului JCAF este ilustrată în figura 3.1. Constă dintr-n *Serviciu de Context* care este conectat într-o manieră peer-to-peer, fiecăruia revenindu-i să administreze contextul specific unui mediu. De exemplu, un serviciu de contect poate rula într-o sală de operaţie, intermediind informaţia contextuală specifică acestei încăperi, precum cine este acolo, ce face, cine este pacientul, şi care este statusul operaţiei. O reţea de servicii poate coopera prin interogarea fiecăruia pentru informaţia contextuală.

Fiecare *Serviciu de Context* este un proces de lungă durată analog aplicaţiilor server Java. O *Entitate* cu informaţia *Contextul* ei este administrată de către *Containerul Entităţii* corespunzător serviciului. O entitate este un program Java de dimensiuni reduse, care rulează într-un serviciu de context şi răspunde schimbărilor contextului. Ciclul de viaţă al unei entităţi este controlat de containerul în care este adăugată entitatea. Containerul entităţii intermediază semnatarul la evenimentele contextului şi anunţă clienţii relevanţi despre modificările intervenite.[11]

Transformatorii Contextului sunt programe Java specifice aplicaţiei pe care le poate scrie un programator şi apoi să le adauge în *Depozitul Transformatorilor.* Depozitul transformatorilor poate fi interogat asupra transformatorilor adecvaţi la rulare.

Accesul la serviciul contextului este controlat prin componenta *Controlul Accesului,* care asigură autentificare corectă a cererilor clienţilor. Această componentă constă practic din două părţi şi anume lista de control a accesului, care specifică ce anume pot accesa clienţii şi un mecanism de autentificare.[11]

Clienţii contextului pot accesa entităţile şi informaţia contextuală în două feluri, printr-o schemă cerere-răspuns, solicitând entităţile şi data lor contextuală, sau prin subscriere ca un *Ascultător al Entităţii*, ascultând modificările unor entităţi specifice. JCAF suportă de asemenea subscrieri bazate pe tip, permiţând subscrierea la toate entităţile de un anumit tip.

Există două tipuri speciale de clienţi: *Clienţi Monitor* şi *Clienţi Actuator.* Un monitor este un client proiectat special pentru a solicita informaţia contextuală a unui mediu prin cooperarea cu un anume tip de echipament senzorial, şi să îl asocieze corespunzător cu o entitate. Un actuator contextual este un client proiectat să lucreze împreună cu unul sau mai mulţi actuatori pentru a schimba contextul. Senzorii şi actuatorii nu trebuie să fie neapărat hardware.[11]

3.1.2 Modelarea contextului

JCAF foloseşte o abordare orientată pe obiecte pentru modelarea informaţiei contextuale. Abstracţia de baza pentru modelare este dată de interfeţele şi clasele: **Entity, Context, Relationship şi ContextItem.**

Conceptul principal de modelare este interfaţa Entity. Această interfaţă defineşte o entitate din lumea reală pe care vrem să o modelăm pentru a păstra evidenţa contextului ei. Obiectul care implementează această interfaţă este obiectul principal din serviciul contextului. O entitate este un mic program Java care rulează într-un anume serviciu pe un calculator. Entităţile primesc şi răspund cererilor din partea clienţilor, folosind Java RMI. În plus sunt notificate dacă apar modificări în contextul lor, deoarece Entity extinde interfaţa EntityListener.[12]

Această interfaţă defineşte metode pentru a iniţializa o entitate, pentru a servi solicitările şi ca să înlăture o entitate din serviciul contextului. Acestea sunt cunoscute drept metodele ciclului de viaţă şi sunt apelate în următoarea secvenţă:

- Entitatea este construită, apoi este apelată metodam **init()**.
- Când se schimbă contextul entităţii este apelată metoda **contextChanged()**.
- Entitatea este scoasă afară dintr-un serviciu, apoi distrusă cu metoda **destroy,** apoi este invocat garbage colectorul şi finalizată operaţia.[12]

În plus faţă de metodele ciclului de viaţă, această interfaţă furnizează metoda getEntityConfig(), care poate fi folosită de entitate pentru a primi informaţia de iniţializare, şi metoda **getEntityInfo()**, care permite entităţii să returneze informaţia elementară despre sine, precum autor, versiune, şi copyright. Pentru a implementa această interfaţă se poate scrie o entitate generică ce extinde **dk.pervasive.jcaf.GenericEntity.** [12]

Odată ce s-a construit un obiect entitate se poate accesa contextul prin metoda **getContext().** O entitate are un **Context**, care conţine un set de obiecte **ContextItem**, care sunt indexate după obiectele **Relationship.** O entitate implementează interfaţa ContextItem, de unde rezultă că o entitate poate fi în contextul alteia.

Deoarece informaţia contextuală este transmisă prin Java RMI, entităţile şi contextul lor trebuie să fie serializabile. Pe deasupra trebuie să existe şi contructori fără nici un argument pentru a putea fi folosiţi în deserializare. Toate aceste clase implementează de asemenea interfaţa XMLSerializable care furnizează metoda **toXML().** Aceasta este convenabil în multe situaţii şi a fost incorporată pentru folosirea pe viitor a invocării metodelor pe baza SOAP.[12]

Entităţile dintr-un **EntityContainer** corespunzător serviciului unui context sunt notificate atunci când apar modificări în contextul lor. Containerul entităţii invocă metoda **contextChanged(),** care este implementată de fiecare entitate în parte şi care acţionează în mod corespunzător schimbărilor intervenite. De exemplu, în cazul entităţii unei persoane, metoda ar putea să actulizeze locaţia atunci când apar evenimente.[12]

3.1.3 Administrarea contextului

Interfaţa **ContextService** are metode pentru adăugarea, ştergerea, obţinerea şi setarea entităţilor. Metoda **getEntity()** returnează copia entităţii pe care o are serviciul, pe când **lookupEntity()** contactează alte servicii cunoscute încercând să localizeze obiectul entitate. Metoda lookupEntity() primeşte ca argumente id-ul entităţii pe care să o caute, numărul maxim de hopuri permise pentru căutarea între servicii, şi un **DiscoveryListener**, care este invocat în momentul în care este descoperită entitatea. Metoda este non-blocantă şi se bazează pe notificarea DiscoveryListener-ului dacă este găsită o entitate care să se potrivească.[11]

Integrate în APIurile serviciului pentru context sunt şi APIurile pentru **TrasnformerRepository**, care conţine metode pentru adăugarea şi obţinerea transformatorilor, şi interfaţa **ContextClientHandler**, care conţine metode pentru adăugarea şi autentificarea clienţilor, inclusiv clienţii de tipul monitor sau actuator. Interfaţa **EntityListenerHandler** conţine metode pentru adăugarea, înlăturarea şi accesarea ascultătorilor entităţilor. Obiectul **EntityEnvironment** este împărţit de către toate entităţile dintr-un serviciu şi are metode pentru setarea şi obţinerea atributelor, accesarea informaţiei despre serviciul de contex local, şi pentru accesarea depozitului transformatorilor.[11]

Arhitectura bazată pe evenimente este suportată de interfaţa EntityListener şi de către clasa **ContextEvent** din modelul de programare. Prin implementarea interfeţei EntityListener un client poate subscrie la modificările ce apar în cadrul unei entităţi. Ascultătorii entităţilor pot subscrie la schimbările unei entităşi specifice sau pot subscrie la modificările tuturor entităţilor de un anumit tip. De exemplu, un ascultător de entităţi poate asculta toate entităţile Person.

Clienţii care sunt interesaţi să asculte contextul unei entităţi pot implementa interfaţa EntityListener descrisă mai jos:

```
public interface EntityListener {
    public void contextChanged(ContextEvent event);
}
```

Entităţile sunt conştiente de schimbările care apar în contextul lor, deoarece implementează interfaţa EntityListener. Drept urmare principala parte de procesare a entităţii este metoda contextChenaged(). Această metodă este invocată de către containerul entităţii de fiecare dată când contextul aceste entităţi este modificat. Aceasta e modalitate puternică de implementare a funcţionalităţii intermediind modificările din contextul entităţii şi, în consecinţă, crearea logicii necesare pentru a translata aceste schimbări în activităţi semnificative pentru utilizatorii aplicaţiei.[11]

Obiectul **ContextEvent** este un obiect standard java.util.EventObject care conferă acces la entitate şi la obiectul ContextItem corespunzător care a generat evenimentul. Există şi o interfaţă **RemoteEntiyListener** care permite utilizatorilor să asculte schimbările din contextul entităţii dintr-un serviciu context care rulează un proces la distanţă. Această interfaţă este folosită de asemenea între serviciile context, permiţând unui serviciu context să asculte schimbările contextului entităţii dintr-un alt serviciu de context. În exemplul în care un serviciu contextual este rulat într-un spital, acest serviciu ascultă schimbările în legătură e.g. persoanele care sunt în acea sală de operaţie. De aceea, în frameworkul AWARE construit pe baza JCAF, acest serviciu contextual ascultă schimbările contextului unui chirurg care operează şi poate lua măsurile potrivite.[11]

3.1.4 Clienţii contextului

Frameworkul JCAF poate intermedia achiziţia şi transformarea informaţiei contextuale în două moduri – sincron şi asincron. Un monitor de context poate să aprovizioneze în continuu informaţie contextuală unei entităţi prin folosirea metodei **setContextItem()** din interfaţa serviciului contextual. De exemplu, un monitor pentru locaţie poate să actualizeze locaţia unei entităţi atunci când o descoperă.

Un client care cere informaţie contextuală pentru o entitate primeşte ultima informaţie a locaţiei. Aceasta este numită *administrarea asincronă a contextului,* deoarece clientul şi monitoarele (în general toţi clienţii) lucrează independent unul de altul. Modul asincron este modul prevalent în frameworkul JCAF. Totuşi, unele aplicaţii conştiente de context doresc să aibă informaţie contextuală la zi. Drept urmare, frameworkul JCAF suportă modul sincron, unde un client solicită contextul unei entităţi, iar entitatea solicitî la rândul ei contextul să se împrospăteze.[12]

Activitatea curentă a unui utilizator conform unui calendar este un exemplu unde monitorul de activitate întreabă calendarul despre activitatea la momentul solicitării. Interfaţa ContextMonitor este următoarea:

```
public interface ContextMonitor extends Remote {
    public ContextItem getContextItem(String id) throws RemoteException;
}
```

Un monitor se poate înregistra la un ContextMonitorHandler prin folosirea metodei addContextMonitor(). Când clienţii solicită informaţie contextuală, prin metoda getContext(), apoi sunt apelate obiectele relevante ContextMonitor pentru a solicita informaţia contextuală prin invocarea metodei getContextItem(). Pentru a evita punctele moarte (e.g. dacă sistemul calendar nu răspunde), metoda getContext() începe un thread separat să intermedieze monitoarele şi returnează imediat informaţia contextuală disponibilă la acel moment. Când începe un monitor contextual să raporteze înapoi (care ar putea să dureze ceva timp), atunci clienţii sunt notificaţi folosind metoda contextChanged() în interfaţa EntityListener.[12]

Actuatorii sunt folosiţi pentru a seta informaţia contextuală. Interfaţa pentru Context Actuator este ilustrată mai jos:

```
public interface ContextActuator {
    public void contextItemChanged(ContextEvent event);
}
```

Actuatorii pot să înregistreze la un ContextActuatorHandler (i.e. la un serviciu context) prin specificarea tipului de ContextItem căruia îi aparţine actuatorul. Când un ContextItem este modificat într-un serviciu contextual (i.e. este activată metoda contextChanged()), toţi actuatorii înregistraţi pentru acel tip de ContextItem sunt notificaţi folosind metoda contextItemChanged().

3.2 Descrierea arhitecturii AWARE-HOSPITAL

Arhitectura AWARE-HOSPITAL poate fi privită din două perspective în ceea ce priveşte organizarea ei. În primul rând, AWARE-HOSPITAL este organizată pe mai multe nivele, astfel încât nivelele superioare depind de serviciile oferite de nivelele inferioare. Din acest punct de vedere arhitectura constă din patru nivele.

Primele două nivele sunt reprezentate de fapt prin frameworkul JCAF descris anterior. Astfel, pe primul nivel se situează clienţii monitor şi actuator ai frameworkului, care furnizează

informaţii contextuale şi acţionează asupra contextului. De fapt, în cazul platformei AWARE-HOSPITAL, acest nivel este dat de senzorii pentru locaţie, status şi activitate, dintre care soar primul este un senzor hardware.

Al doilea nivel oferă serviciul contextului şi administrarea entităţilor create şi adăugate în containerul entităţilor. Ascultătorii acestor entităţi sunt notificaţi în momentul în care senzorii din primul nivel efectuează modificări în contextul unei entităţi.

Al treilea nivel, cel al conştienţei contextului, reprezintă partea centrală a arhitecturii deoarece la acest nivel interacţionează aplicaţiile cu serviciul contextului. Practic, este un intermediar între infrastructura contextului şi aplicaţiile care folosesc informaţia contextuală.

Ultimul nivel este nivelul aplicaţie care constă din totalitatea programelor şi aplicaţiilor care beneficiază de serviciile conştienţei contextului oferite de platforma AWARE-HOSPITAL. Exemplele unor astfel de aplicaţii sunt HospitalServer şi HospitalPhone, detaliate în capitolul patru din această lucrare.

În al doilea rând, arhitectura respectă un model peer-to-peer hibrid. Descriere şi rolul fiecărei perechi sunt prezentate în următorul capitol.

În continuare este descrisă organizarea arhitecturii pe nivele. Administrarea contextului prezintă primele două nivele ale arhitecturii, cu referire la frameworkul JCAF. Managerul contactelor, serviciul de mesagerie şi distribuirea contextului detaliază nivelul conştienţei contextului şi modul de acces la context, iar ultima parte descrie clienţii care beneficiază de serviciile conştienţei contextului.

3.2.1 Administrarea contextului

Administrarea contextului este responsabilă pentru monitorizarea datelor contextuale din mediul utilizatorului, pentru stocarea şi distribuirea acestor date serviciului conştienţei. Infrastructura contextului este construită la baza unui framework JCAF extins. Pentru a suporta diferite tehnologii senzoriale şi diferite tipuri de date, administrarea contextului este divizată în două părţi. Astfel, nivelul ”Contextului”, este responsabil pentru intermedierea, transformarea şi distribuirea informaţiei contextuale, iar nivelul ”Monitor şi Actuator” este responsabil pentru achiziţia contextului folosind diverişi senzori şi actuatori.

Un serviciu contextual este un proces de durată rulat e.g. pe un server într-o infrastructură distribuită. Informaţia contextuală despre lumea reală este modelată folosind conceptul de Entitate, care are asociată o anume informaţie Context. Exemple de context sunt persoane, locuri, pacienţi, paturi, etc. Exemple de informaţii contextuale sunt locaţia, nivelul zgomotului, activitatea unei persoane sau persoanele din preajmă. O entitate este reprezentată de un mic program Java care rulează într-un serviciu context şi răspunde la modificările apărute în context. Ciclul de viaţă al unei entităţi este controlat de un container unde entităţile pot fi adăugate. Containerul entităţii intermediază semnatarii contextului şi îi notifică asupra schimbărilor relevante.

O entitate dintr-un serviciu context lucrează împreună cu alte componente pentru a-şi îndeplini sarcinile. De aceea trebuie să aibă posibilitatea să se acceseze reciproc şi să acceseze resursele partajate, precum conexiuni la baza de date sau stuburi RMI la alte procese. Aceasta este îndeplinită prin componenta Entity Environment, la care au acces toate entităţile.

Pe lângă accesul la resurse generale ca inişializarea parametrilor şi facilităţi de logare, mediul entităţii furnizează metode pentru accesarea atributelor cheie-valoare şi a transformatorilor contextului. Depozitul transformatorilor poate fi interogat pentru transformatorii adecvaţi la execuţie.

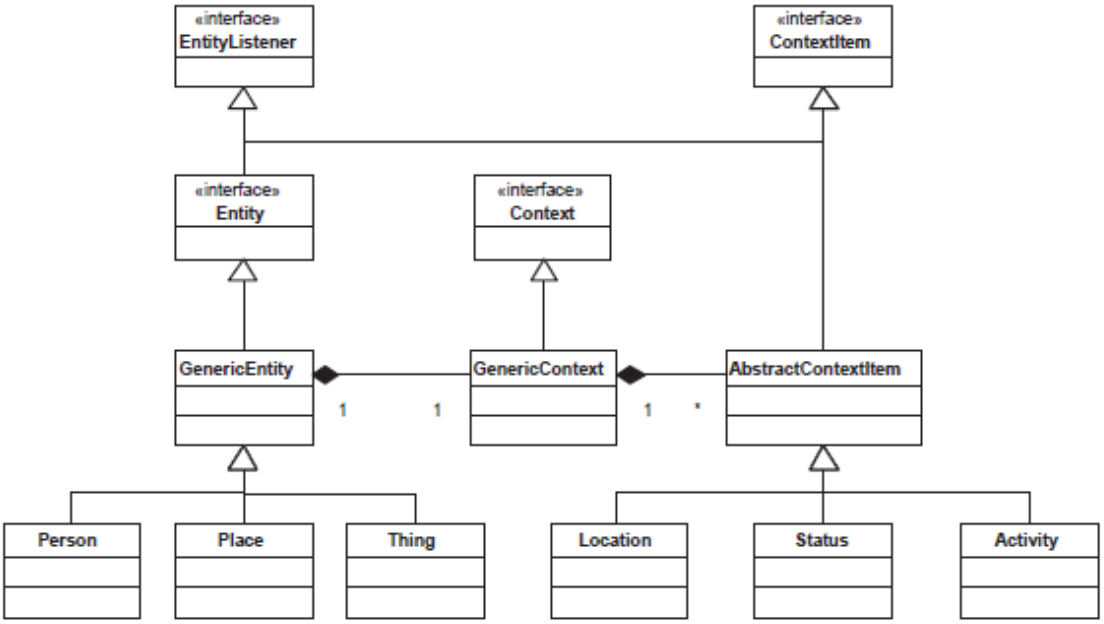


Figura 3.2 Modelul UML al unei Entity, care conţine un Context cu un set de ContextItem.

Clienţii contextului pot accesa entităţile şi informaţia lor contextuală în două moduri. Fie printr-o schemă cerere-răspuns, solicitând entităţi şi data lor contextuală, sau prin subscrierea ca un ascultător al entităţii, ascultând schimbările unor entităţi specifice. JCAF suportă de asemenea subscrieri bazate pe tip , permiţând clienţilor să subscrie la modificările tuturor entităţilor de un anume tip.

Există două tipuri speciale de clienţi: *Monitor şi Actuator.*Un monitor al contextului un proces hardware sau specific contextului, care înregistrează modificările în mediul fizic sau digital. Monitorul adaptează aceste date în funcţie de modelul din serviciul contextual. Exemple de monitoare contextuale sunt senzorii de locaţie pe baza receptoarelor GPS, sau monitoarele WLAN care încearcă să detecteze locaţia pe baza unui echipament WLAN. Alte monitoare pot colecta informaţie despre temperatură, activităţi planificate în calendarul utilizatorului sau statusul afişat într-un sistem IM. Un actuator este un client proiectat să funcţioneze împreună cu actuatorii hardware. Un actuator poate fi proiectat să modifice statusul într-un sistem IM precum în aplicaţia HsopitalPhone construită pe arhitectura AWARE-HOSPITAL.

Conceptul de bază în JCAF este componenta Entity, împreună cu Contextul asociat şi un set de ContextItems. O entitate, un context, şi un context item sunt interfeţe Java, care trebuie implementate de aplicaţiile conştiente de context. Entităţile Person şi Place sunt entităţi de bază şi persoane, precum utilizatorului unui HospitalPhone, şi un loc fizic, ca un birou. Exemple de context sunt contextul unui spital şi cel al unui birou, fiecare cunsocând aspecte specifice despre un spital sau un birou, respectiv. Exemple de context itemi sunt locaţia fizică, activitatea preluată din calendarul activităţilor şi pacientul situat în patul unui spital. Itemii contextului sunt adăugaţi în contextul unei entităţi de către monitoare sau alţi clienţi. Un aspect important este că entităţile în sine sunt itemi (Entity extinde ContextItem în figura 3.2). De aceea, o persoană poate fi adăugată în contextul unei camere, indicând astfel că persoana să găseşte în contextul acelei camere.

Serviciul conştienţei foloseşte infrastructura contextului prin conectarea la un serviciu contextual şi înregistrându-se ca un ascultător pentru entităţile relevante pentru frameworkul AWARE-HOSPITAL. Acestea sunt entităţile de tipul Personal, Medic şi Asistenta. Personal reprezintă clinicienii dintr-un spital , iar Medic şi Asistenta sunt subtipuri ale acestei entităţi.

Serviciul conştienţei este notificat atunci când apar modificări în contextul unui Personal. De exemplu, serviciul conştienţei este notificat când o persoană îşi modifică locaţia, îşi modifică statusul, sau începe o activitate corespunzătoare calendarului de activităţi. Pentru a obţine aceasta, AWARE-HOSPITAL crează şi înregistrează trei monitoare în serviciul contextului: status, locaţie şi activitate. Aceste trei tipuri de monitoare adaugă la rândul lor câte un item în contextul unui Personal, Astfel, cei trei itemi adăugaţi sunt: Status, Location şi Atvity.

3.2.2 Managerul Personalului

Managerul Personalului face parte din serviciul conştienţei contextului, de pe al treilea nivel din arhitectura AWARE-HOSPITAL. Având în vedere că arhitectura are la bază medierea conştienţei sociale din cadrul unui spital, este logic să existe o componentă care sa administreze personalul din acel spital.

Managerul Personalului furnizează accesul şi controlul contextulul pentru clienţii arhitecturii, funcţionând ca un intermediar între clienţi şi infrastructura contextului, oferind transparenţă pentru utilizatori în ceea ce priveşte detaliile infrastructurii.

La nivelul conştienţei contextului, Managerul Personalului se înscrie ca un client la infrastructura JCAF, obţinând astfel acces la serviciul contextului. Astfel, clienţii nu adaugă direct entităţile în serviciul contextului, ci foloseşte serviciile furnizate de către Managerul Personalului.

Personalul unui spital este de mai multe tipuri, în funcţie de rolul pe care îl are acea persoană în cadrul spitalului. Sistemul AWARE-HOSPITAL implementează două subtipuri ale personalului şi anume medicul şi asistenta. Fiecare din cele două subtipuri are roluri diferite într-un spital şi sunt modelate prin atribute specifice. Totuşi, există un set de caracteristici care sunt comune celor două tipuri de personal. Astfel, atât medicul cât şi asistenta sunt caracterizaţi de un nume şi de o specializare.

Personalul spitalului este modelat sub forma entităţilor, care sunt create şi adăugate la serviciul contextului. Fiecărei entităţi îi este asociat un anumit context constând din mai mulţi itemi de context. În prezentarea conceptului conştiinţei sociale a fost descris şi un studiu de caz în cadrul unui spital. Luând în considerare rezultatele acestui studiu, cele mai importante informaşii contextuale pentru un personal sunt date de activitate, status şi locaţie.

În vederea unei bune coordonări şi cooperări într-un spital, personalul spitalului ar trebui să dispună de contextul de lucru al colegilor de lucru. Fiecare din cele trei informaţii contextuale sunt implementate printr-un item care este adăugat în contextul entităţii. Cei trei itemi implementaţi de sistemul AWARE-HOSPITAL sunt: Activity, Status şi Location.

De fiecare dată când este iniţializat un manager de personal, sunt create entităţile corespunzătoare fiecărui medic şi asistentă, iar apoi sunt adăugate în serviciul contextului oferit de infrastrucura contextului.

Managerul spitalului este notificat de fiecare dată când apar modificări în contextul specific unei entităţi, deoarece este înscris ca un ascultător al entităţilor în serviciul contextului. Dacă un medic îşi modifică poziţia şi se mută dintr-o cameră în altă cameră, senzorul asociat efectuează modificările adecvate în contextul entităţii acelui medic şi serviciul contextului notifică Managerul Personalului de evenimentul apărut, care anunţă mai departe clienţii relevanţi

ai arhitecturii.

Activităţile personalului din cadrul spitalului sunt păstrate în calendarul activităţilor şi oferă informaţii asupra programului zilnic al medicilor. În momentul în care un personal tece la o altă activitate conform calendarului, monitorul activităţii notifică serviciul contextului adăugând activitatea nouă în contextul entităţii, iar serviciul contextului anunţă mai departe clienţii relevanţi ai arhitecturii care s-au înscris pentru această informaţie.

Fiecare personal dotat cu un HospitalPhone poate să îşi modifice statusul curent efectuându-se acelaşi proces ca în cazul modificării activităţii sau locaţiei.

Informaţiile privind personalul sunt păstrate într-un fişier XML consultat la iniţializarea managerului contactelor şi acest fişier nu poate fi modificat decât de o singură componentă a arhitecturii şi anume PeerManager.

3.2.3 Serviciul de mesagerie

Un sistem de mediere a conştienţei sociale în funcţie de context trebuie să furnizez şi un mijloc de comunicare asincronă, responsabilitate care îi revine serviciului de mesagerie. Pentru realizarea bunei comunicări între colegii de lucru dintr-un spital, trebuie să aibă posibilitatea să trmită mesaje prioritizate şi să fie notificaţi în momentul în care au fost recepţionate şi citite mesajele.

O componentă principală în serviciul de mesagerie este **PeerManagerul** care joacă rolul unui intermediar sau broker de mesaje. Totodată, PeerManagerul stochează mesajele primite într-o bază de date şi păstrează un istoric al lor.

Serviciul de mesagerie se divedeşte util în multe situaţii din activitatea zilnică a personalului dintr-un spital. Fiecărui mesaj îi poate fi ascociat un nivel de prioritizare, în funcţie de urgenţa acelui mesaj. Sunt situaţii în care nu e nevoie citirea imediată a mesajului şi asistenta nu are nevoie de un răspuns imediat la problema ei, ci este suficient dacă primeşte răspuns pe parcursul acelei dimineţi, nu neapărat în momentul trimiterii mesajului. Pe de altă parte, unele mesaje nu suportă amânare şi trebuie să primească un răspuns cât mai rapid, ceea ce face să aibă un grad de prioritate mai mare.

Serviciul de mesagerie se găseşte la nivelul conştienţei contextului din cadrul arhitecturii AWARE-HOSPITAL şi lucrează împreună cu celelalte componente ale arhitecturii, precum Managerul Personalului. PeerManagerul dispune de lista personalului şi a modului în care poate fi contactat un personal, adică informaţiile referitoare la adresa şi portul unde trebuie să trimită mesajul.

În momentul în care un medic, de exemplu, doreşte să trimită un mesaj altui medic, va scrie mesajul şi îl va transmite PeerManagerului prin intermediul serviciului de mesagerie care rulează în cadrul aplicaţiei HospitalPhone. Un mesaj conţine informaţii privitoare la sursă, la destinaţie, gradul de prioritate şi textul efectiv al mesajului. Odată recepţionat acest mesaj, PeerManagerul verifică destinaţia mesajului şi îl transmite mai departe la aplicaţia HospitalPhone corespunzătoare destinaţiei acestui mesaj. Un pesonal poate să transmită un mesaj folosind şabloane şi poate să îl transmită tuturor persoanelor aflate într-o sală de operaţie, de exemplu.

Comunicaţia se desfăşoară la nivelul TCP/IP ceea ce presupune existenţa unor componente care să transforme un mesaj conform protocolului HPP şi apoi să construiască înapoi acest mesaj.

3.2.4 Distribuirea contextului

Pentru a pune la dispoziţia utilizatorilor sistemului AWARE-HOSPITAL contextul personalului este nevoie de un mecanism de distribuire a contextului. Având în vedere că, de cele mai multe ori, clienţii arhitecturii nu se găsesc în acelaşi loc cu serviciul conştienţei, este nevoie de un protocol de comunicaţie. Accesul contextului este o altă componentă a nivelului conştienţei contextului din organizarea arhitecturii AWARE-HOSPITAL.

Deoarece aplicaţia HospitalPhone rulează pe dispozitive mobile, cu resurse limitate şi putere de procesare mică, a fost definit protocolul de comunicaţie **HPP (HospitalPhone Protocol).** Acest protocol este utilizat şi de serviciul de mesagerie pentru transmiterea mesajelor între colegii de lucru.

Protocolul este utilizat intensiv şi de Managerul Personalului pentru a comunica împreună cu clienţii arhitecturii. Informaţiile privitoare la personalul spitalului, modificările intervenite în context, recepţionarea şi citirea mesajului de către destinatar, fişierele de configurare şi alte informaţii importante sunt transmise pe baza acestui protocol explicat mai amănunţit în continuare la capitolul de implementare.

3.2.5 Clienţii contextului

Principalele aplicaţii dezvoltate până în momentul de faţă şi care sunt clienţi ai contextului oferit de serviciul de conştienţă sunt HospitalServer şi HospitalPhone. Aceste două aplicaţii fac parte din nivelul aplicaţie al arhitecturii AWARE-HOSPITAL.

Pentru a beneficia de serviciile contextului, un client trebuie să se înscrie ca un ascultător al entităţilor relevante pentru el. HospitalPhone rulează pe telefoane mobile şi pune la dispoziţia utilizatorilor două interfeţe: lista personalului şi lista mesajelor. Lista personalului conţine o listă cu numele medicului sau asistentei împreună cu informaţii actualizate ale statusului, locaţiei şi activităţii.

Lista mesajelor conţine toate mesajele primite şi oferă opţiuni pentru citirea mesajului, pentru trimiterea unui răspuns, sau pentru ştergerea mesajului din listă.

Pentru aceasta, HospitalPhone se înscrie ca un ascultător al entităţilor de tipul Personal şi foloseşte ca intermediar Managerul Personalului pentru a avea acces la serviciul contextului oferit de infrastructura JCAF.

HospitalServer implementează nivelul arhitecturii AWARE-HOSPITAL corespunzător conştienţei contextului şi funcţionează şi ca un client al acestei arhitecturi. Totodată monitorizează activitatea din fiecare cameră de spital şi oferă informaţii relevante despre medicii care se găsesc în acea sală şi despre activităţile lor. Deoarece se găseşte pe acelaşi dispozitiv ca şi serviciul conţtienţei nu are nevoie să implementeze protocoale suplimentare.

3.3 Modelul peer-to-peer hibrid

Arhitectura AWARE-HOSPITAL este proiectată ca un model peer-to-peer hibrid, deosebindu-se prin aceasta de arhitectura AWARE, care foloseşte un model client-server în care serverul este singura componentă pe care rulează servciul conştienţei şi serviciul contextului oferit de infrastructura JCAF.

Un astfel de model este favorabil arhitecturii AWARE-HOSPITAL deoarece frameworkul JCAF a fost proiectat pentru servicii de context distribuite care să colaboreze într-o manieră peer-

to-peer, iar arhitectura este construită pe baza infrastructurii contextului oferită de acest framework.

Folosirea unui astfel de model peer-to-peer hibrid este recomandată şi de modul în care este realizată medierea conştiinţei sociale în funcţie de context. Fiecare clinician trebuie să aibă cunoştinţă de contextul de lucru al colegilor săi. Dar contextul este reprezentat de resurse publice accesibile în egală măsură fiecărui participant. Aceasta sugerează ideea folosirii unui model peer-to-peer în care fiecare pereche are un rol egal cu celelalte perechi.

Serviciul conştienţei rulează pe diferite perechi care comunică între ele pentru a menţine informaţia contextuală actualizată. La rândul lor, aceste perechi funcţionează ca nişte servere furnizând informaţia contextuală clienţilor contextului, de exemplu aplicaţia HospitalPhone, de unde rezultă că modelul nu este unul pur peer-to-peer, ci mai degrabă hibrid. Totodată, serviciul de mesagerie este unul de tipul client-server deoarece aplicaţiile HospitalPhone nu comunică direct între ele, ci folosesc ca intermediar şi server un PeerManager.

În cele ce urmează este descris succint modelul peer-to-peer hibrid, apoi este prezentat rolul fiecărei perechi în cadrul arhitecturii, iar în final se descrie Managerul Personalului perechilor, care joacă un rol central în acest model.

3.3.1 Introducere

Modelul peer-to-peer hibrid utilizat de arhitectura AWARE-HOSPITAL este ilustrat în figura 3.3. O reţea peer-to-peer este folosită în general pentru conectarea nodurilor prin conexiuni ad-hoc. O reţea pură peer-to-peer nu are noţiunea de server sau clienţi, ci doar perechi egale care funcţionează simultan ca server şi clienţi. Sistemele peer-to-peer hibride, de exemplu AWARE-HOSPITAL, dispun de o componentă centrală care păstrează informaţii despre perechi şi răspunde cererilor pentru acea informaţie. În cazul de faţă, PeerManagerul reprezintă componenta centrală care deţine informaţiile despre configurarea reţelei şi despre personalul spitalului.

Pe de altă parte, perechile sunt responsabile pentru stocarea şi furnizarea serviciilor disponibile care nu se găsesc pe server. Perechile sunt reprezentate în figura 3.3 prin serviciile contextuale, aplicaţiile HospitalServer şi aplicaţiile HospitalPhone. Toate acestea vor fi prezentate mai detaliat în secţiunea următoare.

Un scop important al reţelelor peer-to-peer este ca toate perechile să furnizeze resurse, precum lăţime de bandă, spaţiu de stocare şi putere de calcul. Pe măsură ce sistemul se măreşte, creşte şi capacitatea sa. Aceasta nu este adevărat pentru sistemele client-server în care adăugarea clienţilor duce la transferuri de date mai lente pentru toţi clienţii.

De exemplu, dacă se întâmplă să cadă serviciul conştienţei dintr-o anume sală de operaţie atunci sistemul nu va mai beneficia de contextul acelei săli, dar va putea să funcţioneze mai departe oferind informaţii contextuale asupra celorlalte încăperi.

3.3.2 Descrierea unei perechi a modelului arhitecturii AWARE-HOSPITAL

Athitectura AWARE-HOSPITAL prezintă trei tipuri de perechi după cum urmează: serviciul contextului, HospitalServer şi HospitalPhone. Toate aceste trei tipuri de perechi sunt conectate la o componentă centrală numită PeerManager.

Nodurile reprezentate de un serviciu context sau de către o aplicaţie HospitalServer sunt aplicaţii care rulează pe staţii în încăperi precum sălile de operaţie sau sala centrală în care se adună întreg personalul cu diverse ocazii. Unele încăperi nu au nevoie de o interfaţă grafică pentru personalul spitalului, drept urmare vor fi instalate doar serviciile contextului pentru a

achiziţiona şi administra contextul corespunzător acelei săli.

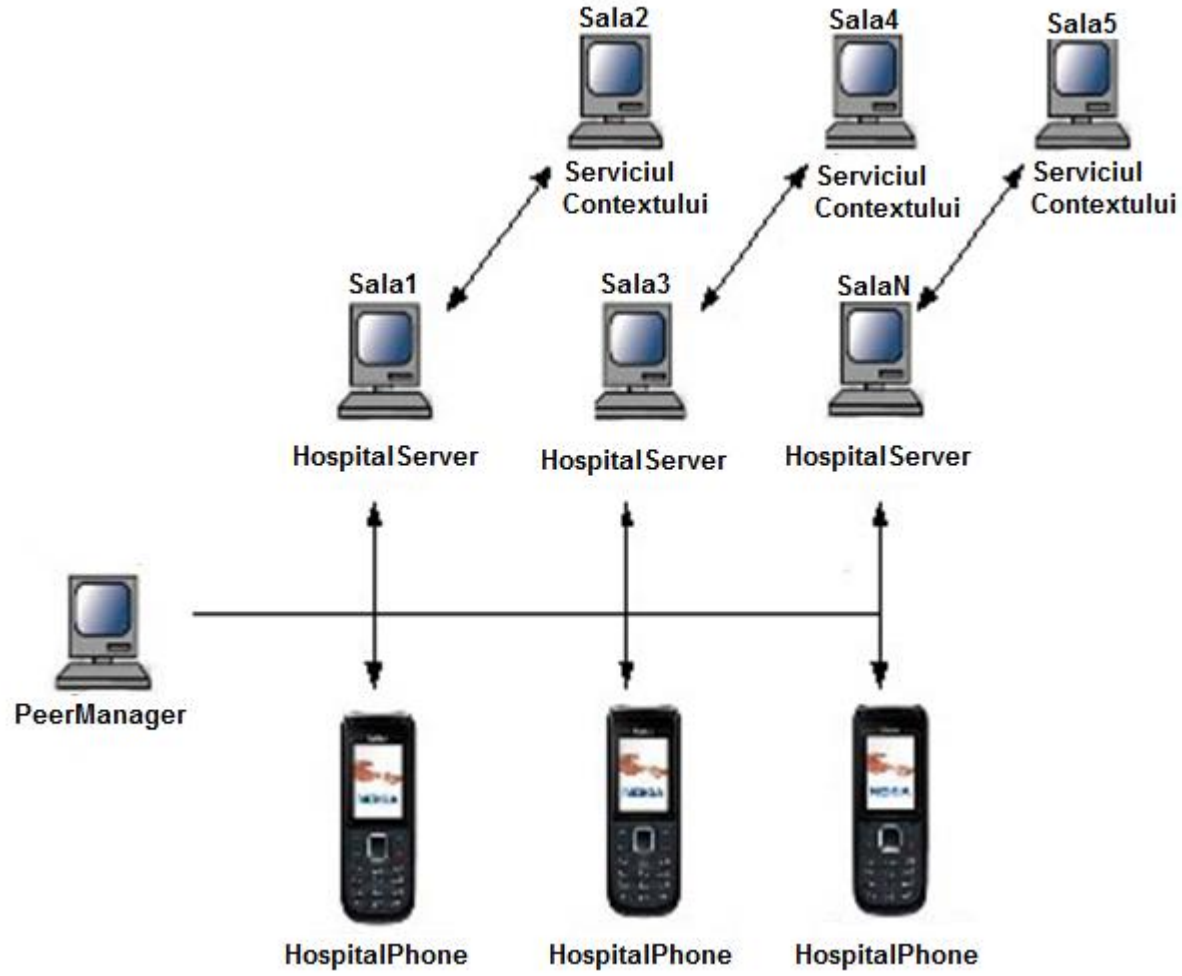


Figura 3.3 Modelul peer-to-peer hibrid al arhitecturii AWARE-HOSPITAL

Nodurile reprezentate de HospitalPhone sunt aplicaşii care rulează pe telefoane mobile şi sunt înmânate fiecărui personal din sistemul AWARE-HOSPITAL care doreşte să aibă cunoştinţă de contextul de lucru al colegilor săi.

Serviciul contextului este implementat de frameworkul JCAF extins, iar perechea care rulează doar acest serviciu contextual furnizează servicii mai departe celorlalte perechi, precum HospitalServer. De exemplu, pentru a ţine evidenţa contextului unei săli cu pacienţi dintr-un spital, a personalului care intră sau care iese este suficient să ruleze un simplu serviciu de context, în care sunt create şi adăugate entităţile corespunzătoare fiecărui membru din personalul spitalului. Clienţii arhitecturii nu acces direct la contextul acestor noduri, deoarece nu este implemntat şi serviciul conştienţei prezent în al treilea nivel al arhitecturii AWARE-HOSPITAL.

Nodurile care rulează serviciul contextului funcţionează totodată şi ca senzori ai locaţiei, informaţie contextuală pe care o transmit mai departe celorlalte perechi la care sunt conectate. Având în vedere că nu lucrează direct cu clienţii arhitecturii şi că sunt controlate de HospitalServer, aceste noduri nu sunt conectate la nodul principal PeerManager după cum sunt conectate celelalte douî tipuri de perechi.

Nodurile HospitalServer sunt aplicaţii care rulează la rândul lor servicii de context, dar

implementează şi nivelul trei al arhitecturii AWARE-HOSPITAL, reprezentat de conştienţa contextului. În plus, dispun şi de o interfaţă prin care utilizatorii aplicaţiilor au cunoştinţă de localizarea personalului spitalului pe încăperi şi de stadiul activităţilor e.g. dintr-sală de operaţie. De asemenea implementează şi serviciu de mesagerie şi comunicare astfel încât medicii care lucrează în sala de operaţie pot să comunice cu alţi colegi printr-un chat şi să trmită mesaje colegilor lor pentru a le cere ajutorul. HospitalServer funcţionează ca un intermediar între serviciul contextului şi clienţii contextului precum HospitalPhone. Nodurile de acest tip se pot lega la alte noduri simple de servicii contextuale, descrise anterior, utilizând informaţia contextuală pe care o furnizează acestea. De asemenea nodurile HospitalServer sunt interconectate între ele şi cu alte noduri de tipul HospitalPhone, descrise mai jos.

Deoarece implementează servicii contextuale, nodurile HospitalServer funcţionează şi ca senzori de locaţie, informaţie contextuală pe care o transmit mai departe celorlalte noduri la care sunt conectate şi care sunt interesate de această informaţie, adică nodurile de tipul HospitalServer sau HospitalPhone.

Toate nodurile sunt conectat la nodul principal PeerManager, de unde primesc informaţiile despre locaţiile celorlalte noduri HospitalServer şi servicii contextuale la care trebuie să se conecteze.

Nodurile de tipul HospitalPhone sunt aplicaţii care rulează pe telefoane mobile, de exemplu Nokia 7650, şi implementează nivelul aplicaţie al arhitecturii AWARE-HOSPITAL, adică sunt clienţi ai arhitecturii, beneficiind de serviciile oferite de HospitalServer. Aceste noduri sunt conectate la nodul PeerManager, de unde primesc informaţiile referitoare la configuraţia reţelei, adică adresele şi porturile la care perechile HospitalPhone îşi oferă serviciile conştienţei contextului.

Aplicaţiile HospitalPhone funcţionează totodată şi ca senzori pentru informaţia contextuală dată de statusul unui personal. Informaţiile achiziţionate la modificare statusului sunt transmise mai departe tuturor nodurilor HospitalServer la care este conectată aplicaţia HospitalPhone unde a intervenit acea modificare.

3.3.3 Managerul perechilor

Managerul Personalului, adică PeerManagerul, reprezintă componenta centrală a modelului peer-to-peer de care este nevoie pentru configurarea celorlalte noduri ale sistemului AWARE-HOSPITAL, precum HospitalServer sau HospitalPhone. Totodată, PeerManagerul joacă rolul important de server în serviciul de mesagerie, intermediind astfel transmiterea mesajelor între colegii de lucru.

Înafara nodurilor simple care rulează doar un serviciu contextual, toate celelalte noduri sunt conectate la acest nod principal, de unde primesc informaţiile de configurare. Principalele informaţii pe care le deţine PeerManagerul constau în două fişiere şi anume fişierul de configurare şi fişierul entităţilor. Era necesar ca fişierul entităţilor să se găsească doar pe o componentă centrală pentru ca să nu poată modifica oricine aceste informaţii despre personalul medical.

Fiecare aplicaţie HospitalServer deţine o copie a acestui fişier urmând să consulte serverul de fiecare dată când se porneşte din nou aplicaţia HospitalServer ca să verifice dacă nu cumva au intervenit modificări.

Lucururile se petrec în mod analog şi cu fişierul de configurare pentru care au copii toate nodurile de tipul HospitalServer sau HospitalPhone. Datele principare pentru configurare diferă pentru fiecare nod. Astfel, nodurile HospitalServer primesc informaţii despre adresele şi

porturile celorlalte noduri la care trebuie să se conecteze, adică noduri simple care oferă doar serviciul contextului sau alte noduri HospitalServer. Pe de altă parte, nodurile HospitalPhone primesc un fişier de configurare cu informaţii referitoare la adresele şi porturile nodurilor HospitalServer.

Totodată, PeerManagerul este brokerul pentru serviciul de mesagerie, responsabil cu administrarea mesajelor şi transmiterea lor mai departe. Este conectat la o bază de date în care păstrează lista tuturor mesajelor împreună cu data şi timpul la care au fost primite acele mesaje.

Capitolul 4 IMPLEMENTAREA

Pentru ilustrarea arhitecturii AWARE-HOSPITAL au fost implementate trei aplicaţii specifice şi anume HospitalServer, HospitalPhone şi PeerManager. Aceste aplicaţii corespund celor trei tipuri de noduri descrise anterior. De fapt, PeerManager este o extindere a aplicaţiei HospitalServer, motiv pentru care aceste aplicaţii vor fi prezentate împreună. Componenta principală de care dispune PeerManagerul în plus faţă de HospitalServer este serviciul de mesagerie.

Aplicaţia HospitalServer a fost dezvoltată utilizând Java SE şi platforma Netbeans, iar HospitalPhone este o aplicaţie mobilă MIDP 2.1 implementată în Java ME utilizând aceeaşi platformă Netbeans şi emulatorul de care dispune pentru simularea aplicaţiei.

HospitalServer utilizează serviciul contextului furnizat de infrastructura JCAF pentru administrarea contextului. Aplicaţia implementează serviciul conştienţei prezent în nivelul al treilea al arhitecturii AWARE-HOSPITAL. De acest serviciu urmează să beneficieze aplicaţia HspitalPhone cu care colaborează pentru asigurarea conştienţei sociale în cadrul sptalului. HospitalServer utilizează comunicaţia la nivel de socketuri şi furnizează un pachet prin care se extinde frameworkul JCAF astfel încât să serviciul contextului să poată fi accesat la nivelul TCP/IP, nu doar Java RMI. Pe deasupra, oferă şi o interfaţă pentru monitorizarea sălilor spitalului, a personalului prezent în ele şi activităţilor desfăşurate în aceste săli.

HospitalPhone comunică la nivel de socketuri cu aplicaţia HospitalServer pentru a beneficia de serviciul conştienţei contextului furnizat de această aplicaţie. Totodată, aplicaţia implementează o versiune mai simplă a frameworkului JCAF care poate rula pe telefoane mobile. De asemenea pune la dispoziţia utilizatorilor o interfaţă pentru conştientizarea contextului de lucru al celorlalţi colegi şi pentru trimiterea sau recepţionarea mesajelor.

În continuare sunt descrise câteva tehnologii utilizate pentru implementarea acestor aplicaţii. Apoi este descrisă implementarea aplicaţiei HospitalServer cu prezentarea funcţionalităţilor aplicaţiei şi a principalelor pachete de clase utilizate pentru realizarea acestor cerinţe. Sunt descrise clasele utilizate pentru modelarea contextului, pentru comunicaţia prin socketuri şi clasele serviciului de mesagerie oferit de PeerManager. În mod analog este descrisă la urmă aplicaţia HospitalPhone cu funcţionalitatea ei şi pachetele utilizate, grupate pe categorii

4.1 Tehnologii si unelte utilizate

4.1.1 Java RMI

În tehnologia Java, apelul procedurilor la distanta se numeşte RMI (Remote Method Invocation) şi este implementată în pachetul java.rmi. Multe din aplicaţiile distribuite se bazează pe modelul client-server. În acest model, un proces, numit *server*, oferă servicii mai multor altor procese, denumite clienţi. Acest model se bazează pe transferul de mesaje : clientul trimite un mesaj catre server, conţinând o cerere de serviciu, iar serverul răspunde clientului, căruia îi trimite de asemenea un mesaj.

Acest model de programare este destul de incomod, deoarece programul trebuie să prevadă operaţii de transfer de mesaje explicite (operaţii *sendreceive*), sincronizarea acestora, tratarea erorilor, etc. De aceea a fost dezvoltată şi o altă metodă de programare distribuită care, deşi foloseşte tot transferul de mesaje între staţii, oferă o interfaţă de programare mai simplă; această metodă se numeşte apelul de procedură la distanţă (RPC – *Remote Procedure Call*).

Ideea acestui mecanism este de a permite unui program să apeleze proceduri localizate în alte staţii (proceduri la distanţă) în acelasi mod cu apelul unor proceduri locale, adică să fie asigurată transparenţa apelurilor la distanţă. Acest mecanism constă în încorporarea în fiecare dintre procesele comunicante (procesul client şi procesul server) a unui nucleu de apeluri (surogat, rădăcină - *stub*) care interceptează apelurile şi răspunsurile, prelucrându-le astfel încât să asigure toate operaţiile necesare de împachetare/despachetare parametri şi rezultate şi de transfer a mesajelor care realizeaza executia procedurii la distanta. (Figura 4.1).

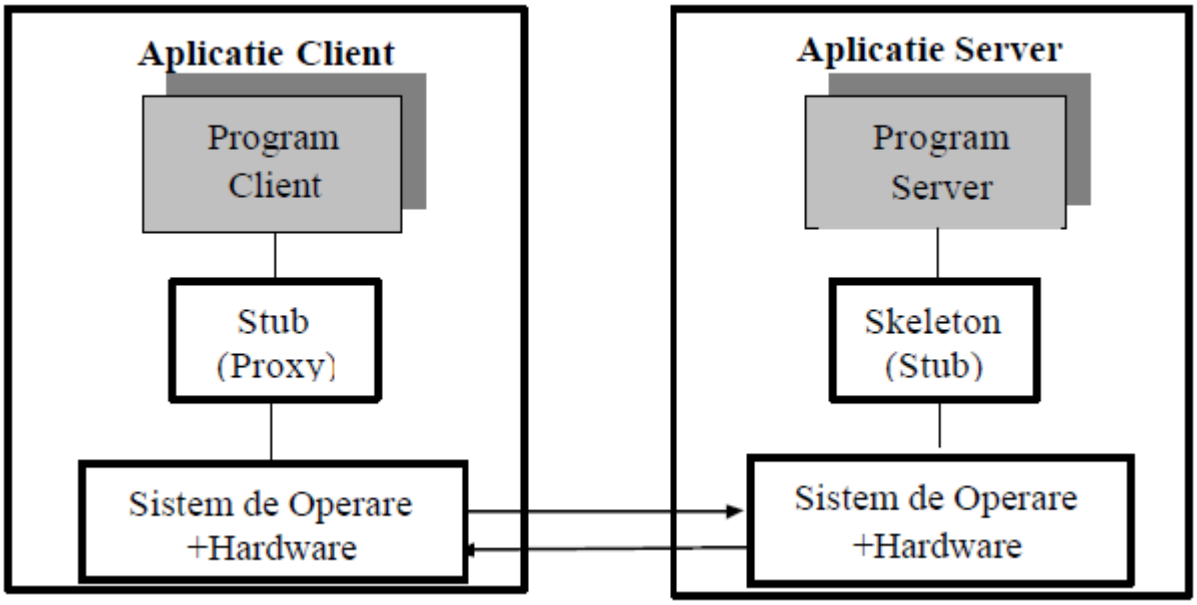


Figura 4.1 Mecanismul de executie RPC[17]

Paşii executaţi la apelul unei proceduri la distanţă pot fi prezentaţi pe scurt astfel:

- Programul client apelează (local) nucleul de apeluri client (surogat - *stub, proxy*), pasându-i acestuia parametrii funcţiei la distanţă.
- Nucleul de apeluri client (*stub*) împachetează parametrii primiţi, construieşte un mesaj pe care îl transmite către nucleul de apeluri al serverului (surogat - *skeleton, stub*) prin intermediul subsistemului IO al sistemului de operare.
- Nucleul de apeluri server (*skeleton*) despachetează parametrii şi apelează local funcţia din server.
- Serverul execută funcţia apelată şi returnează rezultatul nucleului de apeluri server.
- Nucleul de apeluri server (*skeleton*) împachetează rezultatul, construieşte un mesaj, pe care îl transmite către nucleul de apeluri al clientului (*stub*) prin intermediul sistemului de operare şi al reţelei de interconectare.
- Nucleul de apeluri client (*stub*) despachetează rezultatul şi-l returnează programului client apelant.

Mecanisme RPC sunt oferite în mai multe forme; există mecanisme RPC oferite de sistemele de operare (Windows, Unix) şi accesibile în programe prin apeluri de funcţii din bibliotecile de sistem. Alte mecanisme RPC s-au dezvoltat pe baza unor standarde şi cu implementări oferite de diferite firme producătoare de software, deoarece programarea prin componente reutilizabile ca şi programarea distribuită se bazează pe mecanismul RPC.

La nivelul unei aplicaţii distribuite se consideră, de regulă, *client* acel program care oferă unui utilizator (operator) posibilitatea de a lansa diferite operaţii, în general prin intermediul unei

interfeţe grafice, în timp ce serverul (sau serverele) nu au, de regulă interfaţă grafică, deoarece nu sunt sub controlul unui utilizator ci răspund la apeluri ale programelor clienţi. Acest lucru nu înseamnă că un program client nu va putea răspunde unei cereri (deci să îndeplinească funcţie de server), sau că un program server nu va putea apela la servicii de la alte servere (sau chiar de la proprii clienţi). Modul de comunicare între aceste entităţi variază destul de mult în tehnologiile de programare oferite de diferite firme de software.

Tehnologiile care oferă posibilitatea de programare distribuită în reţele de calculatoare asigură o intermediere între aplicaţii şi sistemul de operare definind un nivel intermediar software, cunoscut sub numele de *middleware*. Tehnologiile Microsoft COM, Java RMI, CORBA (*Common Object Request Broker Architecture*, standard propus de OMG - *Object Management Group*) sunt cele mai cunoscute sisteme middleware care permit dezvoltarea aplicaţiilor distribuite folosind componente software. Toate aceste tehnologii se bazează pe mecanismul RPC (*Remote Procedure Call*), implementat în modalităţi specifice.

Folosind facilităţile oferite de un sistem middleware, dezvoltarea aplicaţiilor distribuite este mult mai rapidă şi eficientă, beneficiind de suport pentru crearea si activarea serverelor, împachetarea/despachetarea parametrilor metodelor şi transferul acestora pe reţeaua de interconectare, lansarea şi tratarea excepţiilor şi multe altele. De fapt, este aproape de neconceput dezvoltarea unei aplicaţii distribuite de la zero, mare parte din problemele legate de programarea distribuită fiind deja rezolvate şi oferite ca module reutilizabile de diferitele tehnologii de programare actuale.

Termenii folosiţi în tehnologia RPC diferă în diversele ei abordări, deşi au aceeaşi semnificatie : în Java RMI se folosesc termenii *stub* si *skeleton* pentru nucleul de apel client, respectiv server. O aplicaţie Java RMI constă din doua programe separate: un server şi un client care rulează pe maşini virtuale diferite (posibil şi pe staţii diferite) şi comunică prin intermediul unei interfeţe la distanţă (*remote interface*). În mod tipic, serverul crează obiecte care implementează metodele interfeţei la distanţă (obiecte la distanţă, *remote objects*), face cunoscute referintele acestor obiecte (prin intermediul unui serviciu specializat) si asteapta apelurile clientilor a metodelor la distanţă. În mod tipic, un client obţine referinţa unui obiect la distanţă (tot prin intermediul unui serviciu) şi invocă metodele interfeţei la distanţă implementate de obiectul respectiv.

Aspectele de bază în tehnologia Java RMI: definirea interfetelor la distanta, localizarea obiectelor la distanta, comunicatia clientului cu obiectele la distanta.

4.1.2 J2ME

La ora actuală există o multitudine de dispozitive mobile: telefoane mobile, PDA (Personal Digital Assistants), POS, communicators, telefoane inteligente (smartphones) şi pagere. Caracteristicile comune ale acestor dispozitive mobile sunt: capacitate redusă de calcul; memorie puţină (RAM şi ROM);

- dispozitive de afişare de dimensiuni reduse;
- interfaţă utilizator limitată;
- dimensiuni reduse;
- bandă de transfer limitată.

Aceste caracteristici influenţează modul de proiectare şi realizare a aplicaţiilor mobile. Aplicaţiile mobile sunt aplicaţii concepute să fie executate pe dispozitive mobile care suportă o tehnologie adecvată. Avantajele aplicaţiilor mobile sunt:

- implementarea directă şi rapidă a aplicaţiilor având în vedere compatibilitatea între

protocoalele cunoscute – HTTP, HTTPS şi WAP (Wireless Application Protocol);
- costurile reduse de obţinere a informaţiei şi costurile scăzute în raport cu alte dispozitive cu aceeaşi funcţionalitate;
- portabilitate din punct de vedere al utilizatorului foarte ridicată pentru utilizatorii şi ce care dezvoltă m-aplicaţii;
- absenţa conexiunilor fizice care limitează mobilitatea în timp şi spaţiu a utilizatorului.

Aplicaţiile mobile prezintă şi o serie de dezavantaje precum:
- limitarea introdusă de memoriile folosite de telefoanele mobile;
- limitarea introdusă de suprafaţa ecranului - pot fi afişate în general circa 5 linii de text;
- viteză mai scăzută a ratei de transfer decât în alte conexiuni, de exemplu decât într-o linie închiriată la un ISP (Internet Service Provider), ceea ce conduce la un proces mai dificil de dezvoltare a acestora.[17]

În momentul de faţă dispozitivele mobile, tehnologiile şi aplicaţiile pentru aceste dispozitive cunosc o dezvoltare continuă. Domeniile în care se dezvoltă astfel de aplicaţii sunt variate, precum: comerţ electronic (*mcommerce*); managementul informaţiilor; divertisment (jocuri, aplicaţii multimedia); afaceri. Există mai multe criterii de clasificare aplicaţiilor mobile, cum ar fi criteriul portabilităţii (native, portabile), al distribuirii în spaţiu (distribuite, de tip desktop) ş.a. La ora actuală, cele mai cunoscute sisteme de operare pentru care se dezvoltă aplicaţii mobile native sunt Windows, Symbian şi Palm.[17]

Platforma J2ME (Java 2 Micro Edition) dezvoltată de firma Sun funcţionează pe diverse dispozitive mobile pe care este instalată o maşină virtuală Java (JVM).Cel mai mare avantaj al utilizării platformei Java pentru dezvoltarea aplicaţiilor mobile îl constituie posibilitatea de a realiza cod portabil care poate rula pe platforme diferite. Totuşi, este aproape imposibil de a porta întreaga funcţionalitate a unei aplicaţii pe toate dispozitivele mobile, deoarece acestea au caracteristici care diferă de la un dispozitiv la altul, cum ar fi:
- memoria disponibilă (RAM şi ROM);
- capacitatea de procesare;
- dimensiunea ecranului;
- lăţimea de bandă;
- durata de viată a bateriei.[17]

J2ME este împărţită în configuraţii şi profiluri. Configuraţiile conţin bibliotecile de bază ale limbajului Java pentru o categorie de dispozitive. Deasupra fiecărei configuraţii este un profil. Profilurile definesc biblioteci specifice dispozitivelor mobile pentru interfaţa utilizator, reţea şi stocarea datelor. Fiecare profil are propriul mediu de execuţie şi este realizat pentru un număr de dispozitive mobile similare. La ora actuală există două configuraţii:
- *Connected Limited Device Configuration (CLDC)* – proiectată pentru dispozitive cu restricţii de resurse: telefoane mobile, unele PDA-uri; aceste dispozitive pun la dispoziţie între 128 KB şi 512 KB memorie pentru maşina virtuală Java (KVM);
- *Connected Device Configuration (CDC)* – proiectată pentru dispozitive mobile mai puternice: PDA evoluate, dispozitive de reţea); Maşina virtuală Java pentru aceste dispozitive (JVM) are un set mai mare de clase, de aceea memoria pusă de la dispoziţie de aceste dispozitive ajunge până la 2 MB.[17]

Configuraţia CDC are mai multe funcţii comparativ cu configuraţia CLDC (funcţii matematice, de intrare/ieşire, securitate), de fapt configuraţia CLDC fiind inclusă în configuraţia CDC. Pentru configuraţia CLDC cel mai important profil este *Mobile Information Device Profile*

(MIDP). Pentru configuraţi CDC există două profiluri:

- *Foundation Profile (FP)* – este profilul pe primul nivel al configuraţie CDC şi furnizează suportul de reţea pentru dispozitive mobile
- *Personal Profile (PP)* – destinat dispozitivelor mobile cu interfaţă grafică şi/sau acces la Internet şi include biblioteca Java AWT şi suport pentru applet-uri Web; un subset al PP este *Personal Base Profile (PBP)* destinat dispozitivelor cu acces la reţea şi anumite elemente de interfaţă grafică.[17]

În figura 4.2 este prezentată platforma J2ME cu cele două configuraţii şi profilurile existente. Peste profilurile existente pot fi adăugate profiluri opţionale, specifice fiecărui dispozitiv mobil.

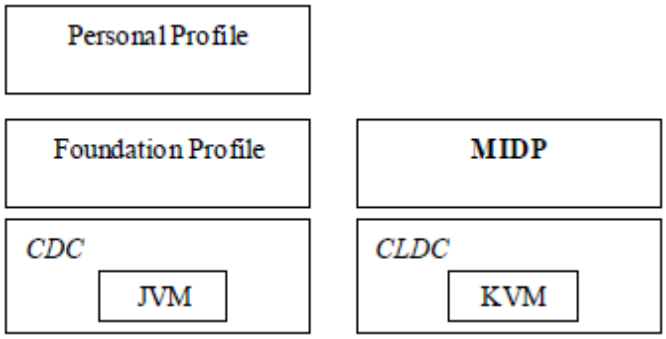


Figura 4.2 Platforma Java 2 Micro Edition[17]

Bibliotecile de clase puse la dispoziţia aplicaţiilor mobile Java sunt limitate la caracteristicile dispozitivelor mobile, spre deosebire de aplicaţiile Java care folosesc J2SE sau J2EE. Astfel, pachetele existente în CLDC şi MIDP sunt:

- java.lang
- java.io
- java.util
- javax.microedition.io
- javax.microedition.lcdui
- javax.microedition.midlet
- javax.microedition.rms

4.1.3 MIDP 2.0

Clasele comune din pachetele J2SE şi J2ME sunt puţine şi la unele dintre acestea lipsesc o funcţii şi date membre faţă de configuraţiile evoluate. De exemplu, CLDC nu oferă suport pentru virgulă mobilă, de aceea clasele *Double* şi *Float* au fost eliminate, precum şi metodele din alte clase care operează asupra acestor tipuri de date. [17]

În momentul de faţă există specificaţiile MIDP 2.0, care sunt o extindere a specificaţiilor MIDP 1.0. Specificaţiile MIDP 2.0 aduc o serie de îmbunătăţesc şi extensii în ceea ce priveşte accesul la reţea, securitatea, interfaţa grafică şi managementul aplicaţiilor.

Noile pachete introduse în MIDP 2.0 sunt:

- javax.microedition.lcdui.game
- javax.microedition.media
- javax.microedition.pki

În ceea ce priveşte reţeaua, MIDP 2.0 pune la dispoziţie noi clase precum suport pentru

HTTPS şi noi clase: *CommConnection*, *UDPDatagramConnection*, *SocketConnection* şi *ServerSocketConnection*.[17]

La capitolul securitate este pus la dispoziţie un manager de securitate bazat pe domenii, semnarea codului aplicaţiilor şi verificarea certificatelor digitale. Sunt aduse elemente de interfaţă grafică noi: *CustomItem, POPUP ChoiceGroup, Spacer şi GameCanvas şi sunt actualizate o serie de*
elemente grafice existente:

- obiectele de tip *Form* şi componentele acestora: *ChoiceGroup*, *Gauge*, *ImageItem*, *Item*, *StringItem* şi *TextField ;*
- *Alert*, *Canvas*, *Choice*, *Command*, *Display*, *Displayable*, *Image*, *List*, *Screen*, *TextBox* şi *Ticker*.

Este oferit suport pentru recepţionarea datelor prin metoda PUSH, modalitatea de descărcare şi instalare a aplicaţiilor fiind standardizată (OTA – Over The Air). Sunt aduse noi capabilităţi în ceea ce priveşte multimedia, grafica şi jocurile. Multe dispozitive mobile de pe piaţa internă care au maşina virtuală Java sunt oferă suport doar pentru specificaţiile MIDP 1.0.

4.1.4 SVG

SVG este o platformă pentru descrierea aplicaţiilor grafice 2D în limbajul XML. Combinaţia dintre SVG şi JavaScript oferă o platformă puternică pentru grafica interactivă 2D, comparabilă cu tehnologiile Flash şi Java. Platforma SVG oferă o grafică XML pentru Web prin trei tipuri de obiecte grafice: **vector graphic shapes** (linii şi curbe), **imagini** şi **text**. Obiectele pot fi grupate, transformate şi reprezentate în mod dinamic şi interactiv.

SVG utilizează standarde XML pentru text, formatele JPEG şi PNG pentru imagini, DOM (*Document Object Model*) pentru scripting şi interactivitate, SMIL (*Synchronized Multimedia Integration Language*) pentru animaţie şi CSS pentru styling.

Platforma SVG are două părţi: un fişier de bază de tip XML şi programare API pentru aplicaţiile grafice 2D. Familiarizarea cu HTML, XML şi programarea orientată spre obiecte (OOP) determină o înţelegere mai clară a utilizării specificaţiilor SVG . Limbajul XML (*eXtensible Markup Language*) este limbajul ce oferă un format pentru stocarea şi transmiterea de date printr-o descriere declarativă. „Gramatica” XML include XHTML (the XML version of HTML), SVG, MathML (the *Mathematics Markup Language*), ChemML (the *Chemistry Markup Language*) şi GML (the *Geography Markup Language*).

Ca limbaj de scripting prin utilizarea lui JavaScript şi SVG Document Object Model (SVG-DOM), SVG este o extensie a lui HTML DOM level 2 familiar dezvoltatorilor de Web. Elementele SVG pot fi caracterizate de animaţie prin utilizarea Synchronized Multimedia Integration Language (SMIL).[13]

Fişierele sau documentele SVG au extensia **.svg** şi sunt editate cu un editor simplu (de exemplu Notepad). Aceste fişiere nu sunt compilate, ci doar interpretate de către browser (*Firefox 1*.5+, *Opera 9* sau *Internet Explorer* cu Adobe SVG Viewer).

Elementele „**root**” pentru documentele XMLsunt acele etichete XML interpretate de browser. De exemplu, <html> pentru XML şi <svg> pentru SVG. „*Namespace*” XML sau atribute "xmlns" realizează o descriere unică de identificare a atributelor SVG cu datele XML. Pentru aplicaţiile SVG sunt necesare următoarele declaraţii:

- xmlns= ”http://www.w3.org/2000/svg”
- xmlns:xlink="http://www.w3.org/1999/xlink"

Elaborarea aplicaţiilor grafice în SVG se poate realiza prin cunoaşterea specificaţiilor

SVG şi utilizarea elementelor şi atributelor conform definiţiilor XML şi SVG. Conţinutul SVG va fi cuprins între etichetele <svg>. Se poate învăţa prin exemple.

4.1.4.1 Programarea în SVG

Elementele (etichetele) esenţiale pentru aplicaţiile grafice sunt: <circle>, <ellipse>, <rect>, <line>, <polyline>, <polygon> şi <path>. Etichete importante sunt şi <g> pentru gruparea elementelor (formelor) şi <use> pentru reutilizarea elementelor definite în secţiunea <defs>. O listă completă privind definirea, sintaxa şi utilizarea elementelor SVG se află la adresa *W3C SVG 1.1 Recommendation* (www.w3.org/TR/SVG11/) şi la adresa *W3Schools SVG site*. Pentru exemplificare prezentăm codul SVG în figura 4.3, cod care realizează generarea unor forme geometrice: *poligon, cerc, dreptunghi*, linie (dreaptă) şi *drum* (path), indicându-se pentru fiecare atributele şi elementele de identificare (referinţa) corespunzătoare. Se va folosi şi elementul <text> pentru desenarea unui text.[13]

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="570" height="470" >
<polygon style="stroke:#24a;stroke-width:1.5;fill:#eefefe"
   points="10,10,400,10,430,230,10,280,10,10" />
<circle style="stroke:#d33;stroke-width:2;fill:#7ce"
   cx="100" cy="80" r="50" />
<rect style="stroke:#2aa;stroke-width:7;fill:#ded;opacity:.8"
   x="170" y="80" height="120" width="220" />
<line style="stroke:#eea;stroke-width:6" x1="30" y1="250" x2="340" y2="60"/>
<path style="fill:#daa;fill-rule:evenodd;stroke:none"
  d="M 230,250 C 360,30 10,255 110,140 z "/>
<circle style="stroke:#d33;stroke-width:2;fill:#7ce"
   cx="370" cy="140" r="30" />
<text x="270" y="50">Exemplul 1</text>
</svg>
```

Figura 4.3 Exemplu de specificaţie SVG

În cazul în care se doreşte ca textul desenat să fie realizat cu un anumit font şi cu o anumită mărime (size), este nevoie să se definească (introdusă înainte de eticheta <polygon>) în secţiunea <defs> o clasă (class="titlu")printr-un stil CSS:

```
<defs>
<style type="text/css"><![CDATA[
.titlu { font-size:30px; font-weight: bold; font-family: batang;
stroke: none; fill: black; text-anchor: middle} ]]></style>
</defs>
```

Elementul <text> va fi în acest caz:

```
<text class="titlu" x="270" y="50">Exemplul 1</text>
```

În exemplul precedent, coordonatele sunt exprimate în pixeli şi valoarea lor este relativă la originea (0,0) considerată implicit la SVG (şi la alte limbaje) în colţul stânga-sus cu direcţia X spre dreapta, iar direcţia Y în jos [9, 13]. În aplicaţii trebuie realizate anumite calcule astfel ca toate coordonatele utilizate să fie relative la această origine. Pentru evitarea acestor calcule se

utilizează *coordonatele omogene* (*x,y,w*), unde (*x,y*) sunt coordonatele carteziene, iar *w* este factorul de multiplicare. În Grafica pe calculator (*Computer Graphics*) se utilizează transformările în 2D (*scalare, translaţie, rotaţie*), prin reprezentarea acestora ca o mulţime de *transformări lineare* 3- imensionale.[13]

Această reprezentare este definită de o matrice simplă 3 × 3, astfel că fiecare transformare are o *matrice de transformare* ce se utilizează în calcule. În felul acesta, transformările sunt reprezentate unitar printr-un *calcul matriceal*: vectorul linie (*x,y,w*) se înmulţeşte matriceal cu matricea transformării. Spre exemplificare, dacă trebuie să mutăm originea implicită (0,0) în punctul de coordonate (250, 250) şi să schimbăm direcţia Y în sus (aşa cum există reprezentarea în matematică), sunt necesare următoarele: a*=1, b=c=0, d=-1, e=f=250*, prin urmare matricea transformării este *matrix (1, 0, 0, -1, 250, 250)*. În SVG acest lucru se realizează prin elementul <g> şi atributul „transform”:

```
<g transform="matrix(1, 0, 0, -1, 250, 250)">
...
elemente svg cu centru in (250,250)si directia Y in sus
...
</g>
```

SVG utilizează *Document Object Model* (http://www.w3.org/TR/SVG/svgdom.html) prin care include tehnici OOP de *JavaScript* în vederea generării de *obiecte, proprietăţi* şi *metode*.

Facilităţile, aspectele dinamice şi interactive oferite de SVG realizează posibilitatea elaborării de *aplicaţii grafice dinamice şi interactive* pentru diverse discipline. Programarea orientată spre obiecte (OOP), evenimente oferite de utilizarea şi programarea acţiunilor mouse-ului, programarea animaţiei cu SMIL (*Synchronized Multimedia Integration Language*), DOM (*Document Object Model*), utilizarea tehnologiei Java şi tehnica „*Drag and Drop*” sunt câteva din aspectele esenţiale ce definesc avantajele platformei SVG.[13]

4.2 Dezvoltarea unei aplicaţii J2ME

4.2.1 Ciclul de viaţă al unui MIDLET

Pentru realizarea aplicaţiilor Java destinate dispozitivelor mobile sunt urmate o serie de etape, printre care:

- proiectare;
- codificare;
- compilare;
- pre-verificare (verificarea modului în care programul utilizează clasele puse la dispoziţie de bibliotecile J2ME);
- împachetare (crearea unei arhive JAR şi a unui fişier de descriere a aplicaţiei, JAD);
- execuţie;
- testare;
- depanare.[17]

Un MIDlet este o aplicaţie Java care se execută în cadrul maşinii virtuale Java. Fiecare MIDlet conţine o clasă principală derivată din *javax.microedition.midlet.MIDlet*. Sistemul de management a aplicaţiilor instalat pe dispozitivul mobil încarcă MIDlet-ul pentru execuţie. Este executat constructorul clasei, după care aplicaţia trece în starea *Întrerupt*.[17]

Activarea MIDlet-ului se realizează prin apelul metodei *startApp*. În figura 4.4 sunt prezentate stările pe care le poate avea un MIDlet, precum şi modalităţile prin care acesta poate

să treacă dintr-o stare în alta. Un MIDlet ajunge în starea *Întrerupt* în momentul în care sunt întreprinse acţiuni externe
aplicaţiei (sună telefonul, primirea unui mesaj) sau prin metodei *notifyPaused.*

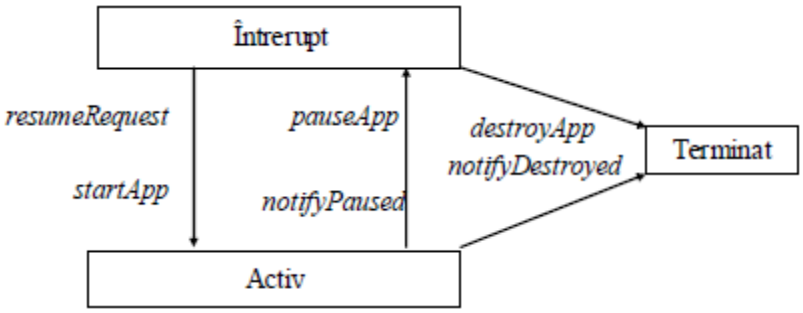


Figura 4.4 Stările unui MIDlet[17]

Metodele *pauseApp* şi *destroyApp* conţin cod pentru eliberarea resurselor în cazul în care aplicaţia este întreruptă sau terminată. Funcţionalitatea MIDlet-ului este asigurată prin intermediul claselor interfeţei grafice. Interfaţa grafică a unui MIDlet este asigurată prin API de nivel scăzut sau de nivel ridicat. API-ul de nivel scăzut presupune utilizarea obiectelor de tip *Canvas*, care permit accesul la ecranul dispozitivului mobil pixel cu pixel.[17]

Tratarea evenimentelor se realizează de către programator la nivelul obiectului de tip Canvas. API-ul pentru interfaţa grafică de nivel ridicat presupune lucrul cu obiecte grafice predefinite, cum ar fi formularele (*Form*) şi elemente de interfaţă care pot fi asociate unui formular (*ChoiceGroup, Gauge, ImageItem, DateField, StringItem* şi *TextField*) sau de tip *Screen* (un singur obiect pe ecran), precum *List, TextBox* şi *Alert*. Tratarea evenimentelor se realizează standard, la nivelul platformei.[17]

Dintre instrumente de dezvoltare a aplicaţiilor mobile Java sunt amintite Borland JBuilder X, care include extensia Mobile Set şi Sun One Studio, Mobile Edition. Java Wireless Toolkit conţine bibliotecile necesare dezvoltării de aplicaţii mobile Java şi un emulator pentru testarea aplicaţiilor.

4.2.2 Utilizarea claselor Display si Displayable

Fiecare MIDlet care afişează ceva pe ecran are instanţă curentă activă a clasei **Displayable.** Este obiectul care reprezintă ce este afişat la un moment dat pe ecran**. Canvas** şi **Form** sunt **s**ubclase ale clasei Displayable.[4]

Ecranul curent se setează prin apelul metodei **setCurrent()** aparţinând instanţei unice Display a MIDletului. Dacă Displayable este un Canvas , apelul setCurrent() apelează metoda **paint()** astfel că data pentru Canvas trebuie să fie gata înainte de apelul acestei metode.

Instanţa Display administrează ecranul şi dispozitivele de intrare ale hardware-ului. Se poate obţine o referinţă la această instanţă prin apelul metodei **Display.getDisplayable().** Metoda cea mai importantă a acestei clase este probabil setCurrent(), dar se pot face şi alte lucruri cu acastă clasă, de exemplu să faci dispozitivul să clipească sau să vibreze. [4]

Clasa Display implementează de asemenea metodele **isColor()** şi **numColors()**, care sunt importante pentru un program care va rula pe un dispozitiv cu diferite limitări ale ecranului. O altă metodă utilă este **setCurrentItem()**, care setează atenţia aplicaţiei pe **Item**-ul solicitat din **Form**. [4]

MIDP 1 şi 2 presupun că dispozitivul are un singur ecran de afişare, dar MIDP 3 permite

desenarea pe mai multe ecrane. Display.getDisplay() funţionează la fel returnând ecranul principal, dar se poat afla celelalte Display-uri disponibile folosind **Display.getDisplays()**. Un MIDlet poate interoga fiecare Display pentru capacităţile sale şi apoi să apeleze setCurrent() în funcţie de situaţie.[4]

4.2.3 Comunicarea in reţea

Cele mai folosite protocoale de dispozitivele mobile pentru transferul datelor sunt HTTP şi SMS. Comunicaţia în reţea pentru Java ME foloseşte o serie de clase **InputStream** şi **OutputStream** care sunt familiare tuturor celor care au programat Java în reţea. Stream-urile de intrare şi de ieşire pot fi obţinute dintr-o conexiune obţinută prin metoda statică **Connector.open()**.[4]

Toate tipurile diferite de protocoale ale conexiunii sunt reprezentate de diferite subinterfeţe ale interfeţei de bază **javax.microedition.io.Connection**. Maşina virtuală alege tipul conexiunii pe baza adresei URL trimisă ca argument pentru **Connector.Open()**. URL-ul trebuie să fie în unul din formatele descrise în RFC2396. În general, un URL constă din numele protocolului (de exemplu HTTP), informaţia de rutare şi argumente adiţionale.[4]

Distincţia de bază dintre protocoale este dacă sunt sincrone sau asincrone, adică dacă este nevoie de un răspuns la fiecare cerere. HTTP este sincron, pe când SMS nu este sincron. În comunicaţia sincronă, un dispozitiv joacă rolul de client, iar alt dispozitiv joacă rolul de client. Pentru a se comporta ca un server, aplicaţia trebuie să dispună de o cale prin care să primească şi să trateze cererile clienţilor. În MIDP, acest lucru este realizat prin componenta **Push Registry.**[4]

4.3 Aplicatia HospitalServer

HospitalServer implementează serviciul conştienţei contextului care este folosit mai departe de alte aplicaţii client precum HospitalPhone. De asemenea joacă rolul unui intermediar între aplicaţiile client ale arhitecturii AWARE-HOSPITAL şi serviciile contextului oferite de către infrastructura contextului.

În continuare se prezintă detaliile de implementare ale aplicaţiei HospitalServer începând cu funcţionalitatea aplicaţiei. Apoi se descrie procesul de iniţializare a aplicaţiei cu toate operaţiile implicate, continuând cu modelarea contextului, cu descrierea claselor de la nivelul socketurilor, iar în final se prezintă serviciul de mesagerie oferit de PeerManager.

4.3.1 Descrierea aplicatiei(Scenarii use case)

În anexa A sunt prezentate mai multe cazuri de utilizare reprezentând principalele funcţionalităţi oferite de aplicaţia HospitalServer. La iniţializare, aplicaţia se conectează la celelalte noduri HospitalServer în funcţie de fişierul de configurare, care conţine informaţiile necesare precum adresa şi portul acestor staţii. Odată stabilită această conexiune la nivel de socketuri se stabileşte o legătură prin care staţiile îşi comunică informaţiile contextuale relevante pentru a-şi actualiza contextul.

După ce se conectează la celelalte noduri, aplicaţia parsează un fişier XML cu datele relevante despre personalul spitalului. În funcţie de aceste date se construiesc entităţile corespunzătoare fiecărui personal şi se adaugă în serviciul contextului asociat aplicaţiei

HospitalServer.

Mai multe detalii despre fişierul de configurare (Config.xml) şi despre fişierul entităţilor (Entities.xml) vor fi furnizate la sfârşitul acestui capitol.

Aplicaţia nu se conectează la noduri HospitalPhone, în schimb crează un socket de tipul server prin care aşteaptă conexiuni din partea acestor aplicaţii pentru a le furniza serviciile conştienţei contextului. În momentul în care se realizează o astfel de legătură se crează un socket pentru a recepţiona mesaje şi pentru a transmite înapoi răspunsul potrivit.

Mesajele care pot fi transmise pe această legătură sunt cereri din partea clienţilor pentru serviciul conştienţei contextului. Astfel, pot fi solicitate cereri pentru trimiterea contextului unei entităţi, informaţii despre personalul spitalului sau apariţia unui nou eveniment generat de schimbarea statusului. Pe de altă parte, HospitalServer transmite răspunsuri la solicitările provenite din partea clienţilor. Pentru realizarea comunicaţiei se foloseşte protocolul HPP, care urmează să fie descris în detaliu la sfârşitul acestui capitol.

Aplicaţia HospitalServer joacă şi rolul de senzori ai contextului achiziţonând informaţii contextuale referitoarea la activitatea sau locaţia personalului. La apariţia unui eveniment de acest gen, aplicaţia generează un eveniment contextual în urma căruia sunt trimise mesaje tuturor clienţilor conectaţi la acest server pentru a-şi actualiza informaţiile contextului. Clienţii contextului pot fi alte noduri HospitalServer sau noduri HospitalPhone.

Având în vedere că aceste aplicaţii urmează să ruleze în sălile spitalelor precum sălile de operaţie, este util pentru utilizatorii lor să beneficieze de o anumită interfaţă pentru serviciul conştienţei contextului. Aplicaţia HospitalServer oferă o interfaţă Java swing prin care utilizatorii pot să afle informaţii despre încăperile din interiorul spitalului şi activităţile care se desfăşoară în aceste săli. Astfel, este prezentată o listă cu sălile spitalului, iar pentru fiecare sală este afişat personalul care se găseşte la acel moment acolo. Interfaţa asociază fiecărei săli din spital, e.g. sălile de operaţie, activitatea care se desfăşoară acolo şi stadiul la care s-a ajuns. Pentru a coopera cu ceilalţi colegi de lucru sau cu celelalte săli de operaţie, aplicaţia HospitalServer prezintă o interfaţă pentru trimiterea şi recepţionarea mesajelor.

Principala componentă adiţională a aplicaţiei PeerManager este reprezentată de serviciul de mesagerie, prin care se realizează transmiterea şi recepţionarea mesajelor.

4.3.2 Iniţializarea aplicaţiei

Principala clasă a aplicaţiei este **JcafClient**, situată în pachetul **application**. De fapt, această clasă împreună cu pachetul **socket** realizează extinderea frameworkului JCAF astfel încât să suporte şi clienţi socket nu doar Java RMI. JcafClient extinde **AbstractContextClient** şi implementează **MessageReceiver**. Extinde AbstractContextClient pentru a putea fi folosită ca un client obişnuit al frameworkului Jcaf şi implementează MessageReceiver pentru a recepţiona mesaje text de la alte săli din spital sau de la membrii personalului.

JcafClient este clasa responsabilă pentru iniţializarea serviciului conştienţei, lucru pe care îl realizează metoda **start()** a acestei clase. Metoda execută mai multe operaţii începând cu parsarea celor două fişiere şi anume fişierul de configurare şi fişierul cu personalul sptalului. Parsarea este realizată ptintr-un procesaor de tip **SAX** . În urma parsării fişierului de configurare se obţine un ArrayList cu obiecte de tipul **ClientInfo**, care urmează să fie parcurs pentru iniţierea unei conexiuni în funcţie de informaţiile conţinute de obiectele ClientInfo construite. Aceste obiecte conţin informaţii despre tipul clientului, nod serviciu context sau altă aplicaţie HospitalServer, adresa şi portul la care se porneşte conexiunea.

Din fişierul cu membrii personalului se construiesc entităţi de tipul **Personal** care sunt

adăugate apoi la serviciul contextului. Pentru construirea acestor entităţi se foloseşte un obiect de tipul **EntityAdmin** care crează instanţe ale clasei abstracte Personal prin generarea unor obiecte de tipul **Medic** sau **Asistenta**, în funcţie de specializarea personalului respectiv. Prin folosirea instanţelor EntityAdmin pentru crearea acestor obiecte se respectă şablonul de proiectare Factory**.**

După ce au fost adăugate aceste entităţi în serviciul contextului pot fi generate evenimente referitoare la contextul acestor entităţi, ca adăugarea sau ştergerea unor itemi de context noi prin modificarea activităţii, statusului sau locaţiei personalului.

Pentru monitorizarea contextului acestor entităţi, meotda start() crează o instanţă a clasei **MedListener** care implementează interfaţa EntityListener. MedListener implementează meotda **contextChanged(ContextEvent event)** prin care este notificată despre evenimentele care apar în contextul entităţilor. Instanţa acestei clase reprezintă un ascultător al entităţilor şi este adăugată la serviciul contextului pentru a deveni astfel activă. MedListener foloseşte funcţionalităţile oferite de frameworkul JCAF şi subscrie tuturor entităţilor de tip Medic sau Asistenta.

La iniţializarea aplicaţiei, JcafClient crează instanţe **TCPServer** şi **TCPServerListener** pentru intermedierea conexiunilor TCP cu aplicaţiile de tipul **HospitalPhone.** TCPServer crează socketul de tipul ServerSocket şi aşteaptă mesaje din partea clienţilor care se conectează la portul pe care rulează serverul. În momentul în care apar mesaje pe aceste conexiuni este anunţat TCPServeListener care se înscrie pentru ascultarea mesajelor provenind de la clienţii HospitalPhone.

Aplicaţiile HospitalServer pot să trmită sau să primească mesaje din alte săli ale spitalului sau de la alţi membrii ai spitalului. Mesajele acestea sunt recepţionate pe aceleaşi conexiuni TCP, iar pentru aceasta se crează o instanţă a clasei **MessageHandler** unde sunt tratate corespunzător aceste mesaje text în funcţie de gradul de prioritate al acestor mesaje.

4.3.3 Modelarea contextului

Pentru modelarea contextului din spital extindem clasa **GenericEntity** implementată de frameworkul JCAF. Principala entitate pe care trebuie să o modelăm în contextul unui spital este personalul acelui spital.

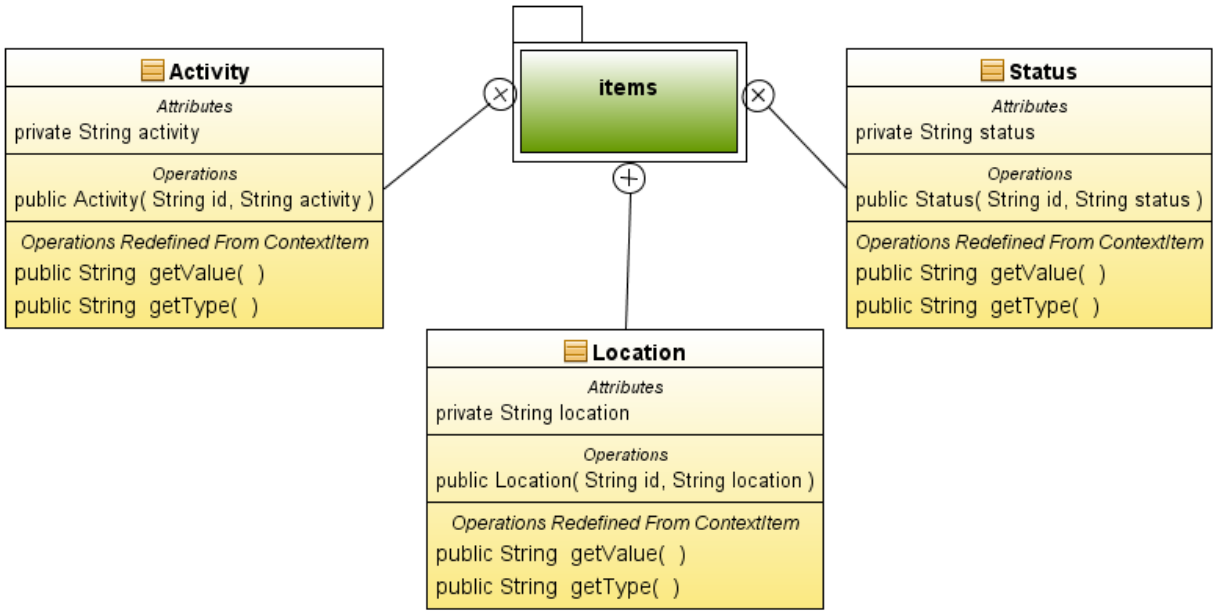


Figura 4.5 Diagrama de clase a pachetului *items*

Pentru informaţia contextuală a acestei entităţi trebuie implementate clase pentru activitate status şi locaţie.Modelarea contextului este implementată de pachetele **items** şi **hospitalentities.** Diagramele UML corespunzătoare celor două pachete sunt ilustrate de figurile 4.5, respectiv 4.6.

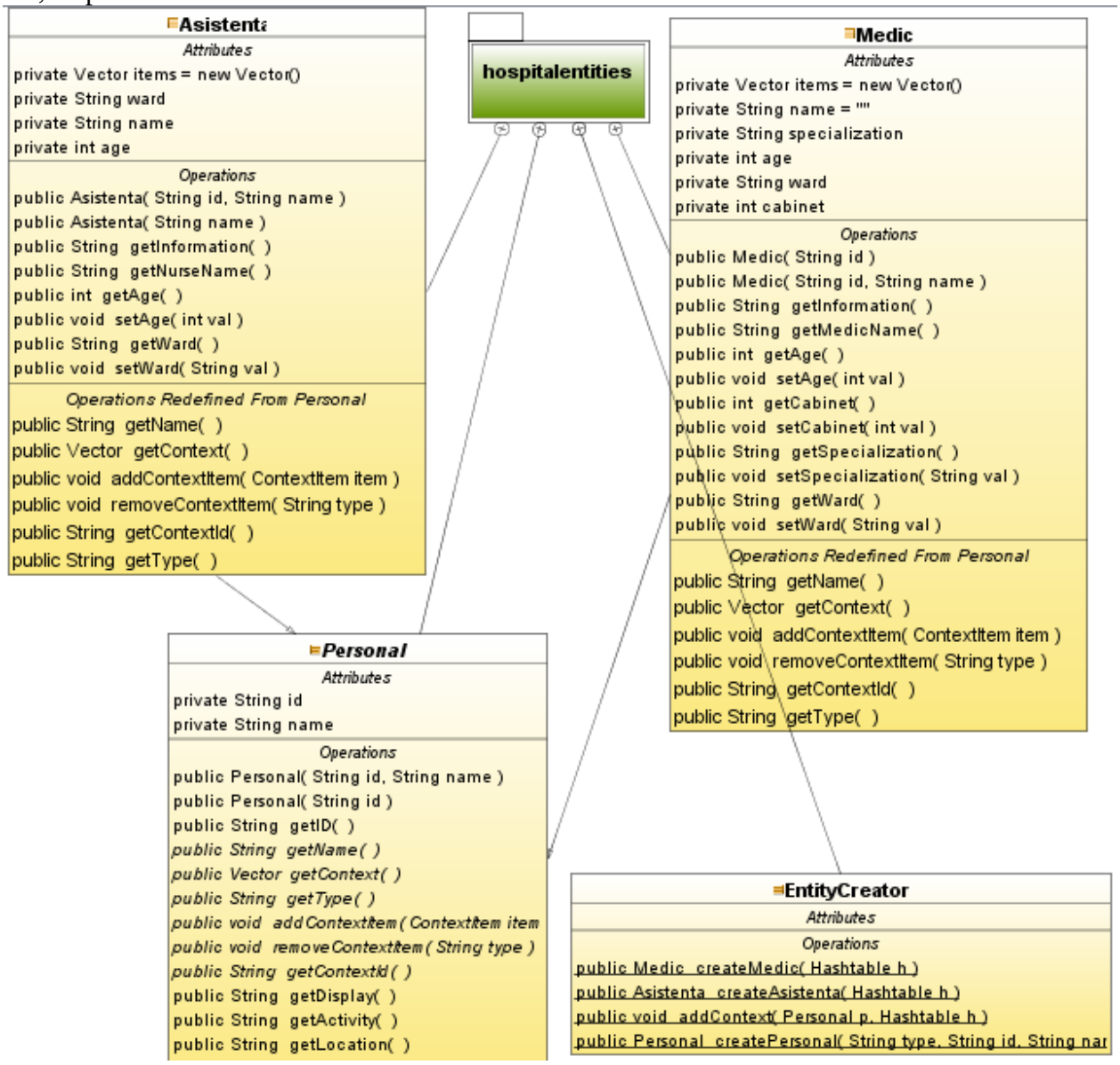


Figura 4.6 Diagrama de clase a pachetului *hospitalentities*

Membrii unui spital pot să fie de mai multe tipuri, în funcţie de specializarea pe care o au în cadrul acelui spital. Aplicaţia HospitalServer modelează două din aceste tipuri corespunzătoare medicilor şi asistentelor.

Pe baza acestor consideraţii este intuitiv să creăm clasa **Personal,** care să descrie caracterisiticile comune ale celor două subtipuri de personal: **Medic** şi **Asistenta.** Clasa Personal este abstractă şi extinde GenericEntity pentru a putea compilată şi adăugată la rameworkul JCAF. Personal are ca atribute numele clinicianului şi id-ul unic de identificare al enităţii. Metodele

implementate de această clasă sunt **getTCPContext()** , care returnează contextul acestui personal în formatul protocolului HPP, şi metoda **contextChanged(ContextEvent event) ,** care adaugă noi itemi la contextul entităţii. La apariţia unei noi activităţi, activitatea veche se pierde şi se păstrează doar cea nouă. În mod analog se petrec lucrurile şi pentru status sau locaţie. Acest lucru este intuitiv deoarece un personal nu poate să facă două lucruri simultan sau să se găsească în două locuri în acelaşi timp.

Pentru informaţii contextuale referitoare la activitate, status sau locaţie sunt implementate metode de tipul getteri sau setteri care sunt abstracte urmând să fie implementate la nivelele inferioare ale claselor Medic sau Asistenta. O altă metodă abstractă importantă este metoda **getTCPFormat(),** care retunează datele despre acest personal în format TCP, adică formatul protocolului HPP.

Clasele Medic şi Asistenta implementează metodele abstracte ale clasei Personal şi adaugă atribute şi metode specifice fiecărui rol. Principala metodă pe care o implementează este metoda **getTCPFormat()** care returnează date despre funcţia lor, numele, id-ul şi contextul specific medicului sau asistentei.

În ceea ce priveşte itemii contextului au trebuit create clase doar pentru activitate şi status, deoarece pentru locaţie se foloseşte clasa oferită de frameworkul JCAF. Cele două clase create sunt **Activity** şi **Status**. Ambele clase implementează AbstractContextItem pentru a putea fi compilate şi adăugate la frameworkul JCAF.

Aceste clase implementează metodele **getValue()** şi **getType()** pentru a returna tipul sau valoarea itemului contextual pe care îl reprezintă.

4.3.4 Administrarea contextului utilizând socketuri

Pachetul **socket** împreună cu clasa JcafClient extind frameworkul JCAF pentru comunicaţia la nivelul TCP/IP. Practic, aici se implementează accesul conştienţei contextului de la nivelul al treilea al arhitecturii AWARE-HOSPITAL. Toate mesajele de nivel jos primite pe conexiunile TCP trebuie să fie aduse la un nivel de abstractizare suficient de bun pentru nivelul aplicaţie al arhitecturii. Acest lucru îl realizează pachetul **socket** al aplicaţiei Hospital Server. Protocolul de comunicaţie la nivelul socketurilor este HPP, care va fi descris mai amănunţit în ultima secţiune a acestui capitol, secţiunea 5.

Aplicaţia HospitalServer porneşte serverul la un anume port care este menţionat şi în fişierul de configurare al aplicaţiei HospitalPhone, aplicaşia client pentru acst server. Odată pornit serverul clienţii se pot conecta folosind adresa staţiei pe care rulează HospitalServer împreună cu portul la care este pornit serverul. Pentru fiecare conexiune acceptată din partea clienţilor se porneşte un thread nou în care sunt tratate corespunzător toate mesajele primite pe această conexiune.

Figura 4.7 conţine diagrama pentru pachetul socket în care sunt ilustrate toate clasele utilizate pentru conexiunea TCP/IP. Clasele necesare pentru abstractizarea procesului de comunicaţie sunt următoarele: **TCPServer**, **TCPServerListener**, **TCPConnectionHandler** şi interfaţa **TCPConnectionHandlerListener**.

TCPServer este clasa corespunzătoare serverului aplicaţiei HospitalServer şi implementează interfaţa Runnable pentru a putea fi pornit un nou thread în care să fie acceptate conexiunile din partea clienţilor. Pentru fiecare conexiune acceptată se crează un obiect TCPConnectionHandler şi se iniţiază un nou thread în care sunt intermediate mesajele de pe această conexiune. TCPServer are un vector în care păstrează toate instanţele TCPConnectionHandler şi de fiecare dată când apare un nou mesaj este notificat ascultătorul

acelei conexiuni.

TCPConnectionHandler este clasa corespunzătoare unei conexiuni socket client. Clasa implementează interfaţa Runnable pentru a putea porni un nou thread în care să intermedieze mesajele conexiunii ca să nu blocheze aplicaţia HospitalServer. Mesajele sunt puse într-o coadă înainte de a fi trimise. Astfel, TCPConnectionHandler are metoda **startWriteQueue()** unde setează această coadă la începutul conexiunii. Pentru a trimite mesaje pe această conexiune se utilizează metoda **queueMessageForSending(byte[] data)** care adaugă mesajul în coada de mesaje de unde urmează să fie luat şi trimis mai departe la destinaţie.

Clasa cea mai importantă a pachetului este clasa **TCPServerListener** unde se realizează practic abstractizarea comunicaţiei TCP. Această clasa implementează interfaţa **TCPConnectionHandlerListener** care conţine metoda principală pentru mesajele conexiunii: **handleConnectionMessage(TCPConnectionHandler handler, InetAddress remoteAddress, int remotePort, byte[] messageBytes).** Serverul TCPServer primeşte la iniţializare şi o instanţă a clasei TCPServerListener pe care o transmite mai departe tuturor instanţelor TCPConnectionHandler astfel încât TCPServerListener ascultă toate mesajele recepţionate de server. În momentul în care apare un nou mesaj este invocată metoda handleConnectionMessage cu parametrii de mai sus. Metoda primeşte ca argumente o referinţă la conexiunea pe care s-a primit mesajul, adresa şi portul acestei conexiuni, şi mesajul sub forma unui şir de bytes.

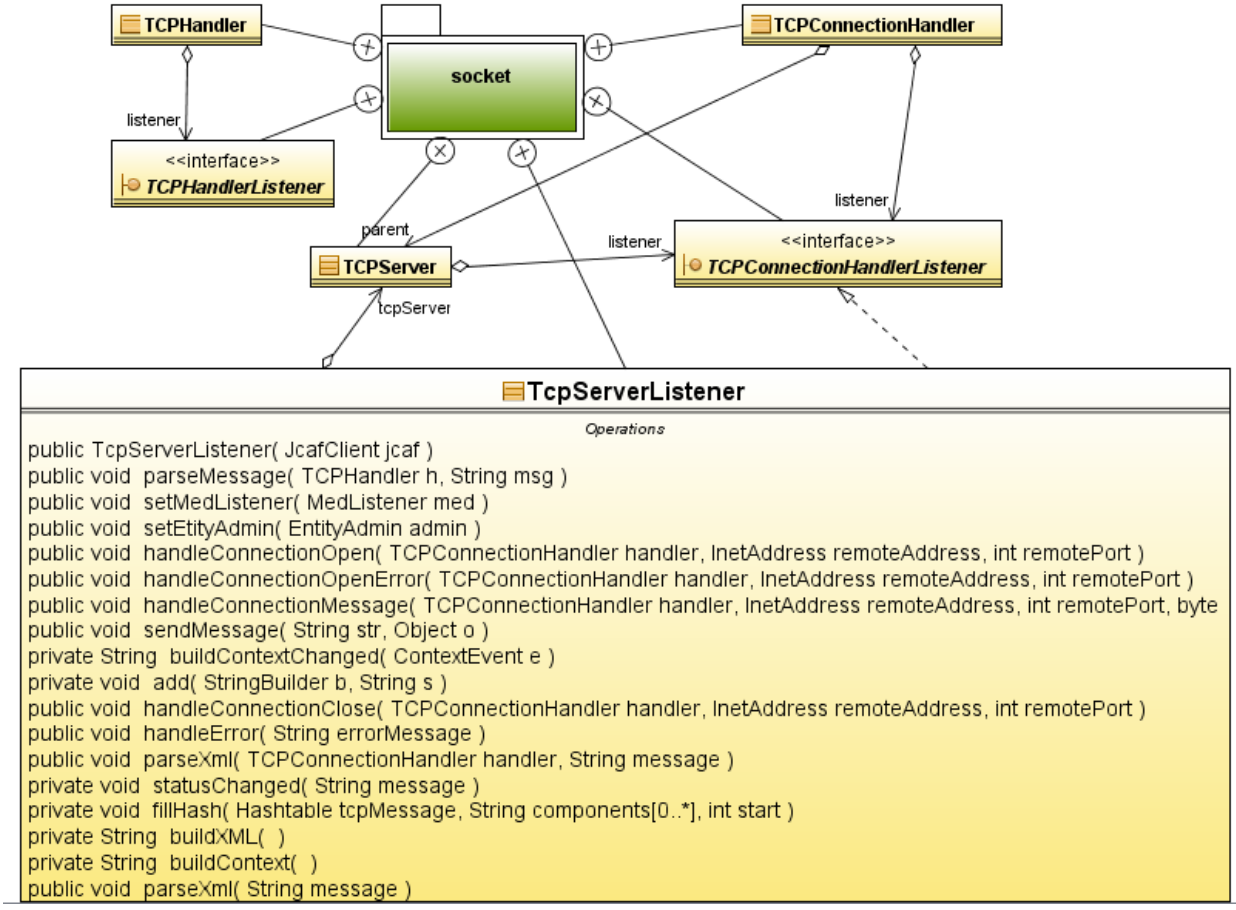


Figura 4.7 Diagrama de clase a pachetului *socket*

Instanţa clasei TCPServerListener foloseşte mai departe referinţa TCPConnectionHandler pentru a răspunde mesajelor în mod corespunzător. Mesajele primite reprezintă cereri pentru contextul entităţilor, informaţii despre personal, mesaje text sau schimbarea statusului unui

clinician. Pentru a afla identitatea clientului HospitalPhone acesta trimite un mesaj cu numele id-ul entităţii pe care o reprezintă. La recepţionarea unui astfel de mesaj ”host name” TCPServerListener adaugă într-o tabelă hash id-ul entităţii împreună cu referinţa TCPConnectionHandler pentru a putea identfica mesajele de pe această conexiune.

Clasa care ascultă schimbările intervenite în context este clasa **MedListener** care implementează interfaţa EntityListener. Deoarece schimbarea statusului presupune modificarea corespunzătoare a contextului entităţii respective, TCPServerListener construieşte un eveniment context şi îl adaugă în serviciul context prin apelarea metodei addContextItem() a clasei JcafClient. Mai departe frameworkul JCAF notifică MedListener despre noul eveniment, deoarece MedListener a subscris la toate modificările contextului entităţilor de tipul Personal.

La rândul ei, clasa MedListener implementează metoda contextChanged(contextEvent) şi la apariţia unui eveniment nou intervenit în contextul aplicaţiei, precum schimbarea locaţiei sau activităţii unui medic, MedListener transformă evenimentul ContextEvent într-o formă a prorocolului HPP şi transmite mai departe mesajul tuturor cleinţilor conectaţi la serverul aplicaţiei HospitalServer.

Metoda care realizează transformarea unui eveniment în formatul protocolului HPP este **contextMessage(ContextEvent)** care se găseşte în clasa StringParser.

4.3.5 Serviciul de mesagerie oferit de PeerManager

În urma analizei arhitecturii AWARE-HOSPITAL s-a evidenţiat beneficiul unui serviciu de mesagerie inclus, motiv pentru care PeerManager implementează un astfel de serviciu. PeerManager conţine aceleaşi clase şi pachete ca aplicaţia HospitalServer, dar are în schimb câteva clase suplimentare în funcţie de serviciile suplimentare pe care le furnizează.

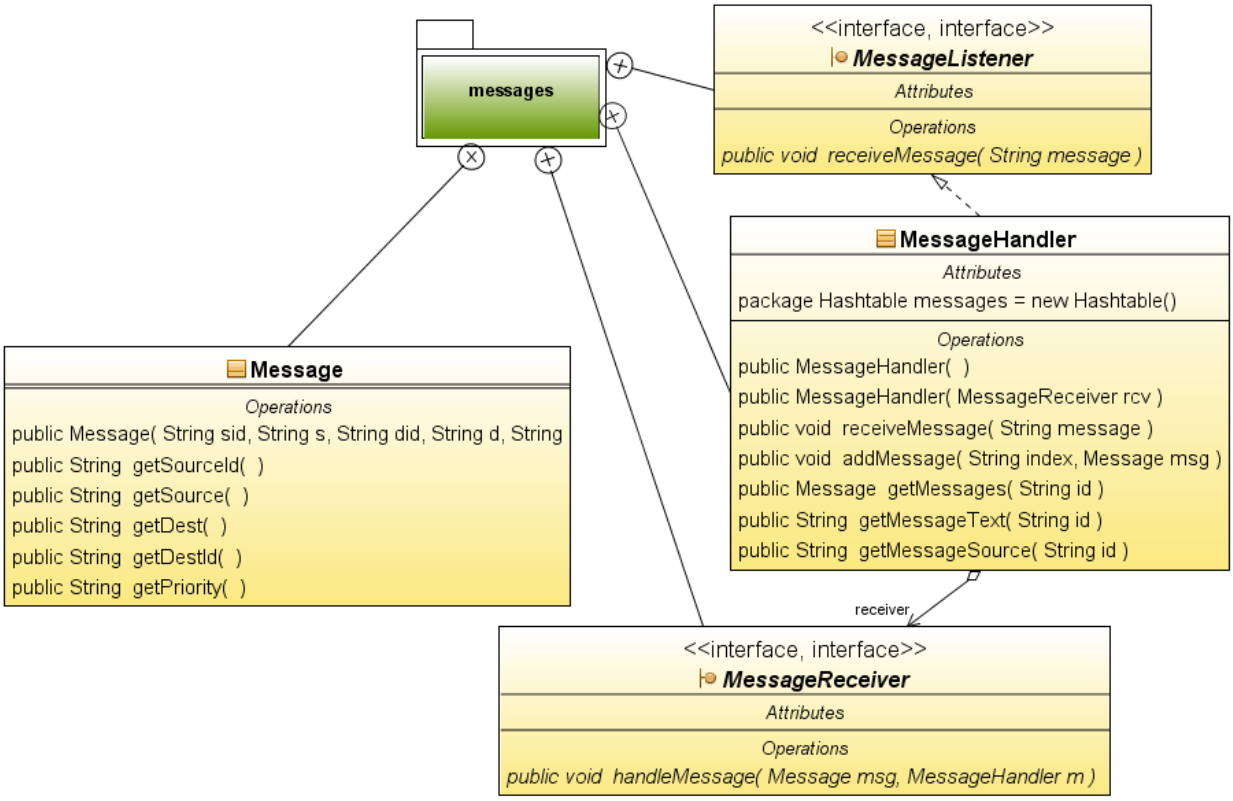


Figura 4.8 Diagrama de clase a pachetului *messages*

Mesajele text sunt recepţionate pe aceeaşi conexiune prin care sunt recepţionate celelalte mesaje pentru serviciul conştienţei contextului. Mesajele se supun de asemenea protocolului HPP şi sunt tranformate în obiecte Message care sunt prelucrate mai departe în mode corespunzător.

Pachetul de clase care implementează serviciul de mesagerie se numeşte **messages** şi este ilustrat mai jos în figura 4.8. Clasele utilizate pentru realizarea serviciului de mesagerie sunt următoarele: **Message**, **MessageHandler** şi interfeţele **MessageListener** şi **MessageReceiver**.

Clasa Message modelează mesajele tect primite şi conţine atribute pentru caracterisiticile mesajelor. Un mesaj este identificat de sursă, destinaţie, prioritate şi corpul mesajului. Sursa şi destinaţia sunt identificate de adresa fizică de unde a provenit mesajul şi de id-ul personalului care a trmis mesajul, respectiv care va recepţiona mesajul.

MessageHandler este clasa care tratează recepţionarea mesajelor şi principala metodă este **handleMessage(String message).** Mesajul primit este transformat într-o instanţă a clasei Message şi transmis mai departe unui MessageReceiver. Clasa MessageHandler implementează interfaţa MessageListener. Instanţa MessageHandler este invocată de TCPServerListener de fiecare dată când recepţionează un nou mesaj text.

JcafClient implementează interfaţa MessageReceiver astfel încât primeşte toate mesajele text şi le transmite mai departe destinaţiei, folosind informaţiile despre identitatea şi adresa destinatarului. Astfel, intermedierea mesajelor între sursă şi destinaţie este o sarcină uşoară care nu presupune completări substanţiale la proiectul aplicaţiei HospitalServer.

4.4 Aplicatia HospitalPhone

Aplicaţia HospitalPhone implementează nivelul aplicaţie al arhitecturii AWARE-HOSPITAL şi beneficiază de serviciul conştienţei contextului oferit de aplicaţia HospitalServer. Implementarea a fost realizată folosind thenologiile Java ME, versiunea MIDP 2.0, utilizând platforma SVG pentru interfaţa utilizator. Pentru simularea aplicaţiei s-a utilizat simulatorul furnizat de platforma Netbeans 6.5 RC2, care a reprezentat totodată şi mediul de dezvoltare utilizat pentru scrierea codului aplicaţiei.

În continuare este descrisă aplicaţia cu principalele funcţionalităţi pe care le pune la dispoziţia utilizatorului. Pentru o vizualizare mai uşoară a acestor funcţionalităţi sunt prezentate câteva cazuri de utilizare. Secţiunea a doua descrie componenta de bază a aplicaţiei şi anume PhoneClient, care este clasa principală prin care se iniţializează aplicaţia şi care controlează celelalte componente ale aplicaţiei.

Secţiunea a treia prezintă modelul folosit pentru reprezentarea contextului, în mare parte preluat de la aplicaţia HospitalServer. Mai departe, secţiunea a patra descrie modul în care este invocat şi utilizat serviciul conştienţei contextului prin socketuri. Apoi este prezentat pe scurt serviciul de mesagerie, iar în ultima parte este explicată interfaţa utilizator.

4.4.1 Descrierea aplicaţiei

Aplicaţia HospitalPhone rulează pe telefoane mobile cu care sunt dotaţi totţi membrii spitalului care fac parte din sistemul de conştienţă socială descris de arhitectura AWARE-HOSPITAL. Scopul arhitecturii AWARE-HOSPITAL a fost medierea conştienţei sociale în funcţie de context, iar aplicaţia HospitalPhone pune la dispoziţia utilizatorilor serviciile furnizate de celelalte componente ale arhitecturii.

Anexa B ilustrează mai multe cazuri de utilizare reprezentând principalele funcţionalităţi oferite de aplicaţia HospitalPhone. La iniţializare, aplicaţia se conectează la nodurile HospitalServer în funcţie de fişierul de configurare, care conţine informaţiile necesare precum adresa şi portul acestor staţii. Dintre aceste noduri, comunicaţia intensivă la nivel de socketuri este realizată doar cu unul dintre noduri care furnizează serviciul conştienţei contextului şi care reprezintă serverul contextului pentru clientul HospitalPhone.

Conexiunile la celelalte noduri HospitalServer sunt realizate deoarece aplicaţia HospitalPhone funcţionează şi ca un senzor al statusului şi ca un client de tipul monitor pentru servciile context implementate pe nodurile HospitalServer respective. Odată stabilită conexiunea la nivel de socket cu serverul se stabileşte o legătură prin care aplicaţia primeşte informaţiile de actualizare a contextului entităţilor asociate personalului spitalului.

Aplicaţia HospitalPhone implementează o versiune simplificată a framewokului JCAF adecvată telefoanelor mobile şi este utilizată pentru administrarea informaţiilor despre clinicienii spitalului. După ce se conectează la server, aplicaţia HospitalPhone parsează un fişier al entităţilor asemănător cu cel folosit de HospitalServer. Din acest fişier extrage informaţiile referitoare la medici şi asistente din care construieşte un vector. Pentru a afla starea contextului acestor entităţi, HospitalPhone interoghează serverul solicitând contextul fiecărei entităţi ascoiate numelui din vectorul personalului.

Utilizând aceeaşi conexiune la server aplicaţia poate să trimită şi mesaje text celorlalţi colegi sau altor săli de spital. Datele transmise prin conexiunea la server reprezintă cereri din partea clienţilor pentru serviciul conştienţei contextului. Astfel, pot fi solicitate cereri pentru trimiterea contextului unei entităţi, informaţii despre personalul spitalului sau apariţia unui nou eveniment generat de schimbarea statusului.

În momentul în care apare un nou eveniment referitor la contextul entităţilor, HospitalServer transformă acel eveniment în format HPP şi îl transmite clientului HospitalPhone prin conexiunea stabilită prin scoketuri. Aplicaţia cleint HospitalPhone recepţionează mesajul HPP şi construieşte evenimentul corespunzător prin care este notificat administratorul contextului despre schimbarea activităţii sau locaţiei unui membru al ersonalului.Toate cererile şi răspunsurile sunt reprezentate în formatul protocolului HPP.

Deoarece aceste aplicaţii vor rula pe telefoanele mobile ale clinicielnilor, este util pentru utilizatorii lor să beneficieze de o anumită interfaţă pentru serviciul conştienţei contextului. Aplicaţia HospitalPhone oferă o interfaţă SVG prin care posesorul telefonului respectiv este informat despre contextul de lucru al celorlalţi colegi. Interfaţa utilizator are două componente principale. Prima componentă prezintă o listă cu personalul spitalului care afişează pentru fiecare membru informaţii contextuale despre activitate, status şi locaţie. Folosind serviciul conştienţei contextului, această listă este actualizată de fiecare dată când apar modificări în contextul unei entităţi. Pentru a coopera cu ceilalţi colegi de lucru sau cu celelalte săli de operaţie, aplicaţia HospitalPhone prezintă o interfaţă pentru trimiterea şi recepţionarea mesajelor. Fiecare mesaj prezintă opţiuni pentru vizualizare sau ştergere, iar în momentul în care destinatarul trimite mesajul este anunţat cel care a trimis acest mesaj pentru a şti că destinatarul a luat cunoştinţă de cererea sa.

4.4.2 Componenta principală

Principala clasă a aplicaţiei este **PhoneClient**, situată în pachetul **application**. De fapt, această clasă împreună cu pachetul **socket** realizează comunicaţia cu serverul HospitalServer. PhoneClient implementează unele din funcţiile serviciului contextului implementat de

infrastructura JCAF. Astfel, păstrează într-un vector informaţii despre personalul spitalului şi are metode pentru adăugarea unor itemi noi de context sau pentru înscrierea ascultătorilor entităţilor.

Metoda prin care se adaugă un nou item în contextul entităţilor este **addContextItem()** care primeşte ca argumente id-ul entităţii şi noul item care se adaugă la context. Ascultătorii entităţii sunt instanţe ale claselor care implementează interfaţa EntityListener cu metoda **contextChanged()**. Clasa **MedListener** reprezintă un astfel de exemplu şi este notificată asupra modificării contextului prin invocarea metodei contextChanged().

PhoneClient conţine tabelă **HashTable** care conţine informaţii despre ascultătorii entităţilor. Fiecare membru al acestei tabele este identificat de o referinţă **TCPHandler** prin care se realizează comunicaţia cu ascultătorul entităţii. Alături de această referinţă, în tabelă se păstrează şi entitatea sau tipul entităţilor pentru care s-au înscris să fie notificaţi.

PhoneClient implementează **Runnable** şi **MessageSender,** pentru a putea rula într-un thread nou şi pentru a trimite mesaje text celorlalte aplicaţii HospitalPhone sau HospitalServer.

PhoneClient este clasa responsabilă pentru iniţializarea aplicaţiei, lucru pe care îl realizează metoda **startClient()** a acestei clase. Metoda execută mai multe operaţii începând cu parsarea celor două fişiere şi anume fişierul de configurare şi fişierul cu personalul sptalului. Parsarea este realizată ptintr-un procesaor de tip **SAX** . În urma parsării fişierului de configurare se obţine un ArrayList cu obiecte de tipul **ClientInfo**, care urmează să fie parcurs pentru iniţierea unei conexiuni în funcţie de informaţiile conţinute de obiectele ClientInfo construite. Aceste obiecte conţin informaţii despre tipul clientului, nod serviciu context sau altă aplicaţie HospitalServer, adresa şi portul la care se porneşte conexiunea.

Din fişierul cu membrii personalului se construiesc entităţi de tipul **Personal** care sunt adăugate apoi într-un vector. Pentru construirea acestor entităţi se foloseşte un obiect de tipul **EntityCreator** care crează instanţe ale clasei abstracte Personal prin generarea unor obiecte de tipul **Medic** sau **Asistenta**, în funcţie de specializarea personalului respectiv. Prin folosirea instanţelor EntityCreator pentru crearea acestor obiecte se respectă şablonul de proiectare Factory**.**

După ce au fost construite aceste entităţi, HospitalPhone interoghează serverul la care este conectat solicitând informaţiile referitoarea la contextul acestor entităţi pentru a asigura integritatea datelor puse la dispoziţia utilizatorilor. Doar la inişializarea aplicaţiei solicită informaţiile contextuale pentru întreg personalul de care dispune serverul, deoarece mai departe serverul îl notifică de fiecare dată când apar schimbări pentru că HospitalPhone se înscrie prin scoketuri ca un ascultător al entităţilor.

Pentru monitorizarea contextului acestor entităţi, metoda startClient() crează o instanţă a clasei **MedListener** care implementează interfaţa EntityListener. MedListener implementează meotda **contextChanged(ContextEvent event)** prin care este notificată despre evenimentele care apar în contextul entităţilor. Instanţa acestei clase reprezintă un ascultător al entităţilor şi este adăugată la serviciul contextului implementat de HospitalServer prin utilizarea conexiunii la server.

La iniţializarea aplicaţiei, PhoneClient crează instanţe **TCPHandler** şi **TCPMessages** pentru intermedierea conexiunilor TCP cu aplicaţiile HospitalServer**.** TCPHandler crează o conexiune la server utilizează această legătură pentru a transmite mesaje serverului şi pentru a recepţiona mesaje din partea acestuia. În momentul în care apar mesaje pe aceste conexiuni este anunţat TCPMessages care se înscrie pentru ascultarea mesajelor provenind de la clienţii HospitalPhone.

Aplicaşiile HospitalPhone pot să trimită sau să primească mesaje din alte săli ale spitalului sau de la alţi membrii ai spitalului. Mesajele acestea sunt recepţionate pe aceleaşi

conexiuni TCP, iar pentru aceasta se crează o instanţă a clasei **MessageHandler** unde sunt tratate corespunzător aceste mesaje text în funcţie de gradul de prioritate.

4.4.3 Modelarea contextului

Pentru modelarea contextului spitalului se folosesc multe din clasele utilizate de aplicaţia HospitalServer. Astfel, HospitalPhone utilizează pachetele **hospitalentities** şi **items** pentru modelarea personalului spitalului şi a contextului său. Fişierul entităţilor conţine descrierea clinicienilor spitalului, din care se construiesc obiecte **Personal** care sunt instanţe ale claselor specifice **Medic** şi **Asistenta**.

Având în vedere că HospitalPhone oferă un nivel de abstractizare al contextului la nivelul aplicaţiei modelul contextului conţine clase suplimentare deoarece clasele oferite de frameworkul JCAF nu au putut fi importate într-o aplicaţie mobilă ce nu le suportă implementarea. Motiv

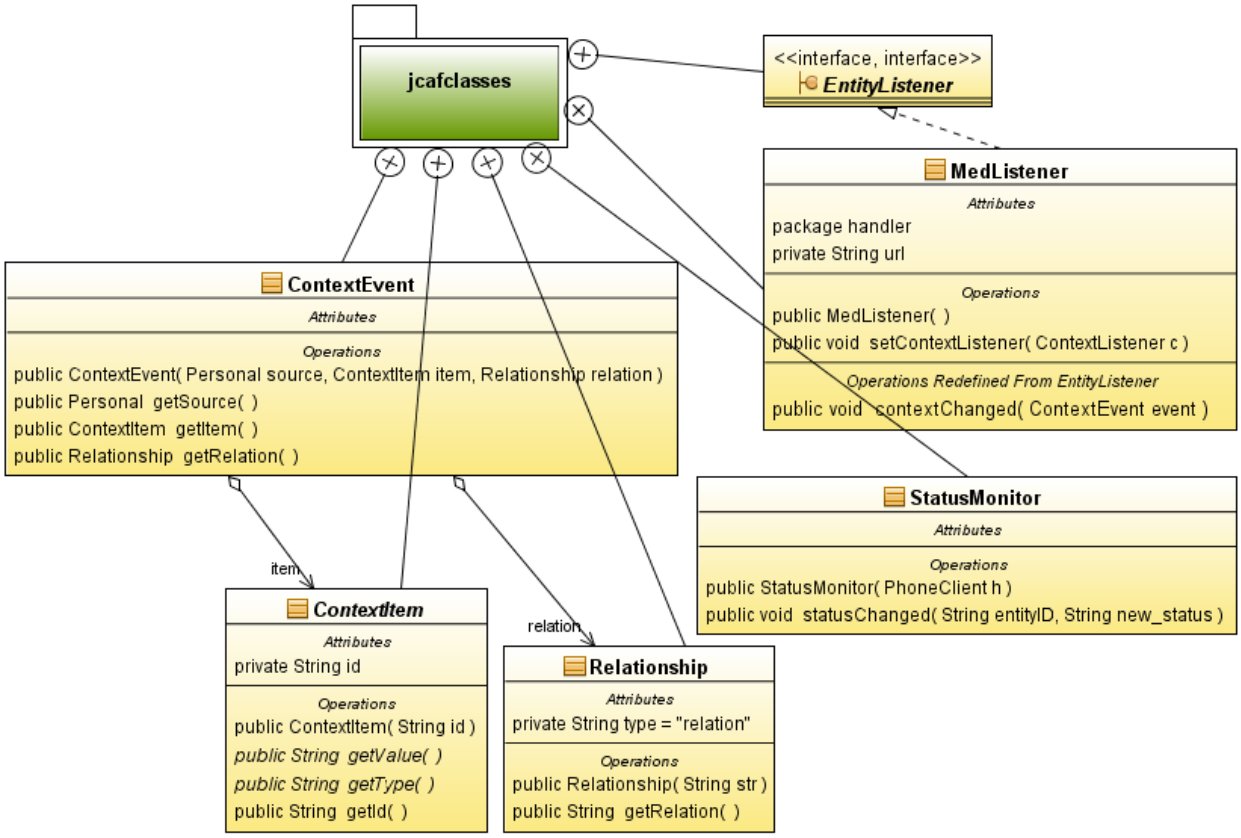


Figura 4.9 Diagrama de clase a pachetului *jcafclasses*

pentru care clasa Personal, de exemplu, nu mai extinde **GenericEntity** , iar itemii contextului ca şi Activity sau Status au trebuit reimplementaţi. Totuşi, clasele nu au suferit modificări substanţiale, păstrându-şi metodele şi atributele, motiv pentru care nu vor fi descrise din nou, deoarece au fost prezentate la implementarea aplicaţiei HospitalServer.

Pentru modelarea contextului a fost nevoie de introducerea câtorva clase splimentare asociate celor din JCAF. Clasele au fost grupate în pachetul **jcafclasses** care este ilustrat în figura 4.9. Clasele noi introduse sunt următoarele: **ContextEvent**, **ContextItem** şi **Relationship.**

ContextEvent este clasa folosită pentru reprezentarea unui eveniment generat de

modificarea contextului. ContextEvent conţine atributele definitorii pentru un astfel de eveniment şi anume: id-ul entităţii care este sursa evenimentului, itemul nou care se adaugă la context şi relaţia în care se află cu entitatea. Evenimentele JCAF pot fi de mai multe tipuri, dar în cazul de faţă este suficientă doar adăugarea unui nou item. Ar mai fi şi stergerea unui item din contextul unei entităţi, dar acest lucru nu este necesar pentru că personalul este caracterizat, după cum s-a menţionat mai sus, doar de o activitate, o locaţie şi un status. Atunci când, de exemplu, un clinician îşi modifică locaţia, noua locaţie se suprascrie peste vechea locaţie devenind actuala locaţie a clinicianului.

ContextItem este clasa abstractă care reprezintă un item al contextului şi este implementată de subtipurile Activity şi Status. Un item al contextului este caracterizat de un id, de o valoare şi de tipul căruia îi aparţine. Id-ul este păstrat în clasa ContextItem, dar metodele pentru returnarea valorii şi tipului contextului sunt abstracte. **Relationship** desemnează pur şi simplu relaţia în care se găseşte itemul faţă de entitatea respectivă, iar în cazul de faţă este reprezentată simplu de un nume, spre deosebire de JCAF unde are mai multe atribute şi metode.

4.4.4 Serviciul conştienţei contextului prin socketuri

Pentru a beneficia de serviciul conştienţei contextului furnizat de HospitalServer, aplicaţia HospitalPhone foloseşte conexiunea pe care a stabilit-o cu serverul la iniţializarea aplicaţiei. Pachetul **socket** împreună cu clasa PhoneClient oferă o serie de servicii pentru realizarea acestei comunicaţii şi pentru abstractizarea detaliilor de nivel jos ale comunicaţiei prin socketuri. Toate mesajele de nivel jos primite pe conexiunile TCP trebuie să fie aduse la un nivel de abstractizare suficient de bun pentru nivelul aplicaţie al arhitecturii. Aplicaţia HospitalPhone utlizează acelaşi protocol HPP definit specific pentru ca aplicaţiile mobile în genul HospitalPhone să beneficieze de serviciul conştienţei contextului.

La iniţializare, aplicaţia HospitalServer porneşte serverul conştienţei contextul la un anume port. După ce a fost pornit serverul, clienţii se pot conecta folosind adresa staţiei pe care rulează HospitalServer împreună cu portul la care este pornit serverul. În urma stabilirii conexiunii aplicaţia HospitalPhone poate să comunice cu serverul solicitând informaţii sau fiind notificată asupra modificărilor apărute în serviciul contextului.

Figura 4.10 conţine diagrama pentru pachetul socket în care sunt ilustrate toate clasele utilizate pentru conexiunea TCP/IP. Clasele necesare pentru abstractizarea procesului de comunicaţie sunt următoarele: **TCPHandler**, **TCPMessages,** împreună cu interfaţa **TCPHandlerListener**. Acest pachet conţine şi câteva clase prin care o aplicaţie mobilă ar putea să funcţioneze ca şi un server pentru alte aplicaţii. În stadiul de faţă al implementării aceste clase nu au fost folosite, dar ar putea fi utilizate în implementări ulterioare pentru serviciul de meagerie sau pentru serviciul contextului care să ruleze pe telefoane mobile într-o formă mai simplă potrivită unor astfel de dispozitive.

PhoneClient păstrează referinţe la toate conexiunile stabilite cu HospitalServer. Totodată PhoneClient stabileşte legătura cu serverul HospitalServer şi crează o instanţă pentru TCPHandler şi TCPMessages, în care sunt intermediate toate mesajele care sunt recepţionate pe această conexiune.

TCPHandler este clasa corespunzătoare unei conexiuni socket cu serverul. Clasa implementează interfaţa Runnable pentru a putea porni un nou thread în care să intermedieze mesajele conexiunii şi în acest fel să nu blocheze aplicaţia HospitalServer. Constructorul acestei clase primeşte ca argument un string cu adresa serverului la care trebuie să se conecteze. Pentru a seta ulterior ascultătorii acestei conexiuni se apelează metoda **setTCPHandlerListener(),** care

primeşte ca argument instanşa unei clase care implemntează interfaţa TCPHandlerListener.

Mesajele sunt puse într-o coadă înainte de a fi trimise. Astfel, TCPHandler are metoda **startWriteQueue()** unde setează această coadă la începutul conexiunii. Pentru a trimite mesaje pe această conexiune se utilizează metoda **queueMessageForSending(byte[] data)** care adaugă mesajul în coada de mesaje de unde urmează să fie luat şi trimis mai departe la destinaţie.

Clasa cea mai importantă a pachetului este clasa **TCPMessages** unde se realizează practic abstractizarea comunicaţiei TCP. Această clasa implementează interfaţa **TCPHandlerListener** care conţine metoda principală pentru mesajele conexiunii: **handleMessage(byte[] messageBytes).** Constructorul acestei clase primeşte ca argumente referinţe la instanţe ale claselor **MedListener** şi **MessageListener,** pe care le invocă pentru a le notifica asupra noilor evenimente intervenite şi pentru mesajele text primite.

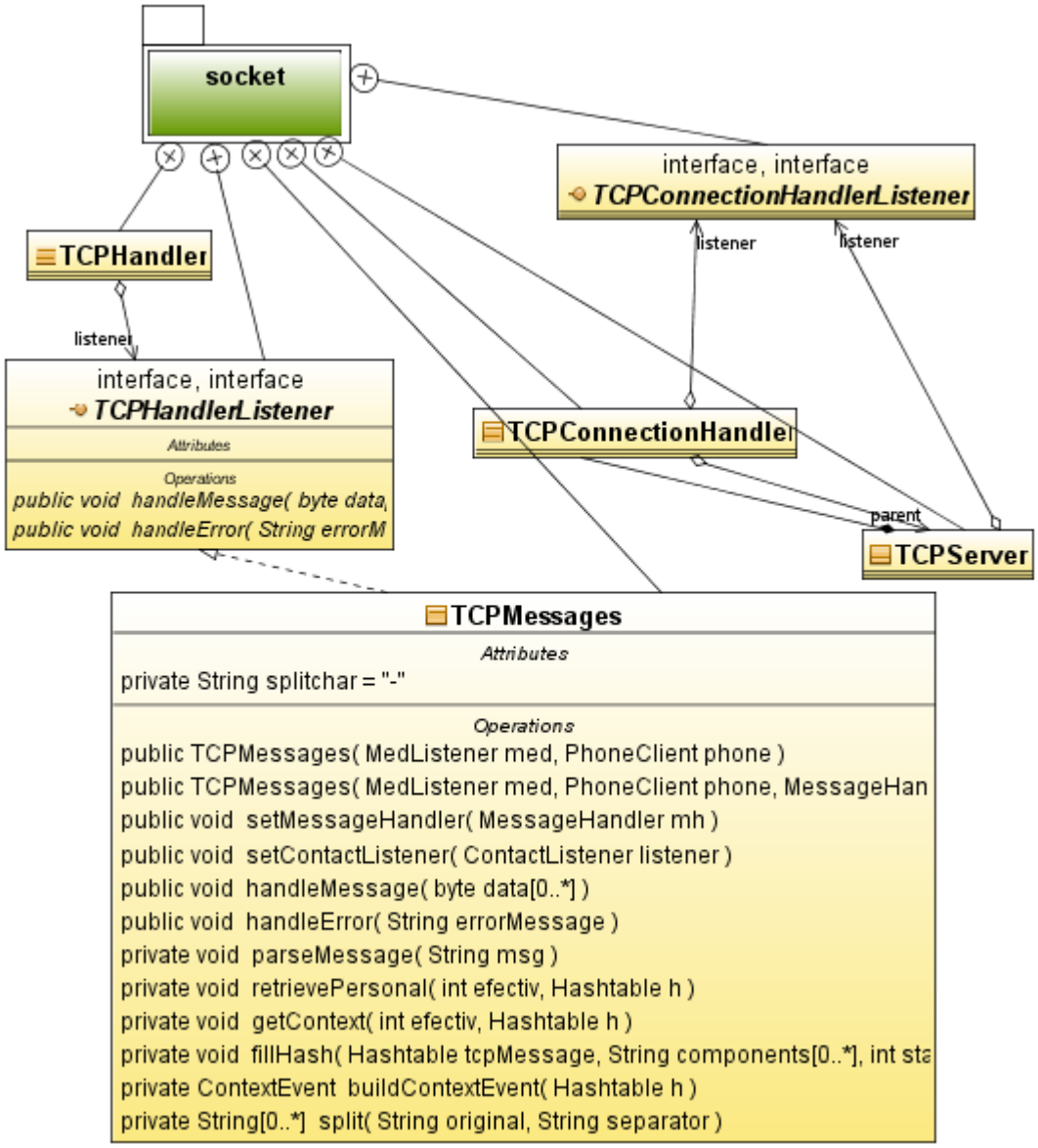


Figura 4.10 Diagrama de clase a pachetului socket aparţinând aplicaţiei HospitalPhone

PhoneClient crează conexiunea la server printr-un obiect de tipul TCPHandler , iar apoi

iniţiază o instanţă a clasei TCPMessages pe care o înscrie să asculte mesajele conexiunii prin apelul metodei setTCPHandlerListener(). Odată realizat acest lucru, de fiecare dată când este recepţionat un mesaj nou se apelează metoda **handleMessage().** Această metodă primeşte ca argument doar mesajul recepţionat sub forma unui şir de bytes care este transformat mai departe în text, care este parsat pentru a putea răspunde cererii sau pentru a prelua informaţiile trimise din partea serverului. Pentru a trimite mesaje serverului, TCPMessages invocă metode ale clasei PhoneClient, care conţine referinţa TCPHandler corespunzătoare conexiunii la server.

Mesajele recepţionate în formatul HPP pot să fie de mai multe tipuri: ”context changed”,”personal info”,”personal context”,”message”, etc. Aceste mesaje sunt parsate şi tratate în mod corespunzător prin metoda **parseMessage().** Un mesaj de tipul ”context changed” reprezintă de fapt un eveniment apărut în contextul unei entităţi. Mesajul text este transmis ca argument metodei **buildContextEvent() ,** iar evenimentul astfel construit este transmis mai departe ca parametru metodei **contextChanged()** implementată de o instanţă a clasei MedListener. Un mesaj de tipul ”personal info” sau ”personal context” conţine informaţii despre personalul spitalului sau despre contextul entităţilor din serviciul contextului. Mesajul text este transmis metodei **retrievePersonal(),** respectiv metodei **getContext().** Mesajele de tipul ”message” reprezintă mesaje text simple care sunt transmise mai departe unei instanţe a clasei MessageHandler.

Clasa care ascultă schimbările intervenite în context este clasa **MedListener** care implementează interfaţa EntityListener. Deoarece schimbarea activităţii sau locaţiei ununi clinician presupune modificarea corespunzătoare a contextului entităţii respective, TCPMessages construieşte un eveniment context şi notifică PhoneClient prin apelarea metodei addContextItem(). Mai departe PhoneClient notifică MedListener despre noul eveniment, deoarece MedListener a subscris la toate modificările contextului entităţilor de tipul Personal.

4.4.5 Serviciul de mesagerie

Pentru recepţionarea sau transmiterea mesajelor, aplicaţiile HospitalPhone folosesc serviciile oferite de către Managerul Personalului. Ca să transmită un mesaj, aplicaţia HospitalPhone construieşte mesajul respectiv conform protocoului HPP şi îl trimite PeerManagerului, care recepţionează mesajul, verifică destinaţia şi trimite mesajul mai departe destinaţiei respective.

Mesajele text sunt recepţionate pe aceeaşi conexiune prin care sunt recepţionate celelalte mesaje pentru serviciul conştienţei contextului. În cazul în care aplicaţia primeşte mesaje, acestea sunt transformate din formatul HPP în obiecte Message care sunt prelucrate mai departe în mod corespunzător.

Clasele utilizate pentru realizarea serviciului de mesagerie sunt următoarele: **Message**, **MessageHandler** şi interfeţele **MessageListener** şi **MessageReceiver**. Aceste clase pun la dispoziţia utilizatorului aproximativ aceleaşi interfeţe ca şi clasele aplicaţiei HospitalServer, dar diferă în implementare.

Clasa Message reprezintă mesajele text primite şi conţine atribute pentru caracterisiticile mesajelor. Un mesaj este identificat de sursă, destinaţie, prioritate şi corpul mesajului. Sursa şi destinaţia sunt identificate de adresa fizică de unde a provenit mesajul şi de id-ul personalului care a trmis mesajul, respectiv care va recepţiona mesajul. O metodă importantă a acestei clase este metoda getTCPFormat() care returnează o instanţă a acestei clase într-un format potrivit protocolului HPP pentru a putea fi transmis mai departe prin conexiunea la server.

MessageHandler este clasa care tratează recepţionarea mesajelor şi principala metodă este

receiveMessage(String message). Mesajul primit este transformat într-o instanţă a clasei Message şi transmis mai departe unui MessageReceiver. Clasa MessageHandler implementează interfaţa MessageListener. Instanţa MessageHandler este invocată de TCPMessages de fiecare dată când recepţionează un nou mesaj text.

Mesajele text primite sunt păstrate de MessageHandler într-un obiect **HashTable**, unde sunt identificate prin numele sursei de la care a fost recepţionat mesajul. MessageHandler implementează metoda **getMessage()** care primeşte ca argument id-ul sursei şi returnează mesajele recepţionate de la acea sursă.

4.4.6 Interfaţa utilizator folosind SVG

Serviciul conştienţei contextului este pus la dispozitia utilizatorului prin intermediul unei interfeţe folosind facilităţile oferite de SVG (Scalable Vector Graphics). SVG aparţine unei familii de specificaţii bazate pe XML şi descrie grafica bidimensională, atât statică precum şi dinamică, adică animaţii. Interfaţa utilizator utilizează SVG deoarece interfaţa telefoanelor mobile este minimală şi limitată.

Interfaţa utilizator conţine două componente principale, având asociate două fişiere svg separate. Prima interfaţă conţine lista personalului spitalului împreună cu informaţii legate de contextul fiecărui clinician. Fiecare membru de personal din listă este afişat împreună cu numele său, cu activitatea, statusul şi locaţia sa. Lista contactelor se actualizează de fiecare dată când apar modificări în contextul unei entităţi. Interfaţa pune la dispoziţia utilizatorului telefonului mobil posibilitatea să îşi modifice statusul curent, modificare de care vor fi notificaţi toţi ceilalţi membrii de personal.

Celaltă componentă de bază a interfeţei utilizator este lista mesajelor recepţionate şi posibilitatea trimiterii mesajelor. Pentru trimiterea mesajelor utilizatorul dispune de o serie de şabloane pe care le folosi. Pentru fiecare mesaj din listă utilizatorul are opţiuni pentru vizualizarea sau ştergerea mesajului. De asemenea, medicul sau asistenta pot trimite mesaje prioritizate.

Interfaţa utilizator este implementată prin intermediul pachetelor de clase **component**, **contact**, **skin** şi **util**. Aceste pachete de clase sunt ilustrate în anexa C. Toate pachetele sunt grupate în folderul view. Pachetul component conţine clase care realizează maparea între interfaţa utilizator şi documentele SVG. Pachetul contact conţine clasele pentru implementarea interfeţelor pentru lista contactelor şi a mesajelor. Documentele SVG corespunzătoare aplicaţiei (list.svg, loadScreen.svg şi messages.svg) sunt păstrate în pachetul skin. Pachetul util conţine clase utile pentru implementarea interfeţei, precum clasa DefaultSVGAnimator pentru animaţie.

Pachetul **component** realizează legătura între interfaţă şi documentul SVG şi conţine următoarele clase importante: **LoadingScreen**, **SVGProgressBar**, **SVGList**, **SVGTextBinding** şi **StatusChange.** LoadingScreen şi SVGProgressBar sunt clasele care implementează interfaţa de încărcare a aplicaţiei HospitalPhone. La pornirea aplicaţiei HospitalPhone apare un ecran cu o bară de progres care ţine până când sunt încărcate toate documentele SVG şi este iniţializată aplicaţia cu toate conexiunile aferente.

Ecranul telefonului poate să afişeze un număr limitat de elemente din lista contactelor. Pentru realizarea mapării între aceste elemente afişate şi corespondentul lor real este utilizată clasa **SVGList.** În momentul în care utilizatorul trece la un alt element din listă se modifică elementele afişate, lucru realizat de implementarea clasei SVGList. Constructorul clasei primeşte ca parametru o referinţă la lista reală cu personalul spitalului. Metodele clasei oferă posibilitatea avansarii în listă cu o poziţie sau revenirea cu o poziţie, modificând în mod corespunzător

elementele afişsate şi documentul SVG asociat. De asemenea se pot accesa elementele listei, poziţia curentă în listă şi valoarea elementului selectat.

SVGList oferă metode pentru actualizarea elementelor afişate. De exemplu, dacă utilizatorul apasă tasta ”down” şi ecranul poate afişa opt elemente din lista contactelor, atunci pe ecran ar trebui să fie afişate şapte contacte din cele prezente, dar devansate cu o poziţie împreună cu un contact nou preluat din listă de la poziţia imediat următoare ultimului element din cele şapte. SVGList realizează actualizare atât pe ecran, cât şi în documentul SVG ascociat interfeţei.

SVGTextBinding este elementul de bază care realizează maparea între un element de pe ecran, de exemplu statusul utilizatorului, şi corespondentul său real din documentul SVG aferent. Metoda **hookSkin()** asociază instanţa SVGTextBinding cu elementul corespunzător din documentul SVG, identificat după numele primit ca parametru al constructorului. Pentru schimbarea valorii afişate se apelează metoda **set()** cu noul text drept argument.

StatusChange este clasa care implementează interfaţa pentru modificarea statusului utilizatorului. StatusChange implementează **KeyListener** şi **CommandListener** pentru a prelua valoarea nouă tastată şi pentru a actualiza valoarea statusului utilizatorului.

Pachetul **contact** este folosit pentru realizarea interfeţei utilizator propriu-zisă şi are următoarele clase: **ContactDetails**, **ContactDetailsForm**, **ContactListSource**, **ContactListScreen**, **ContactMessageForm**, **ContactMessageSend** şi **MessageScreen**. Clasa ContactDetails modelează informaţiile unui clinician aşa cum sunt afişate pe ecran, iar ContextDetailsForm conţine componente SVGTextBinding care fac legătura între atributele instanţei ContactDetails şi elementele corespunzătoare din documentul SVG. ContactDetails conţine informaţii despre numele, specializarea, activitatea, statusul şi locaţia unui membru al personalului.

Componentele de bază ale intefeţei sunt **ContactListScreen** şi **MessageScreen** care folosesc instanţe ale celorlalte clase şi răspunde acţiunilor utilizatorilor, precum apăsarea tastelor sau acţionarea unor comenzi. ContactListScreen foloseşte instanţe SVGList şi ContactDetailsForm pentru realizarea interfeţei utilizator corespunzătoare listei contactelor. La apariţia unui eveniment generat de utilizator, ContactListscreen răspunde în mod corespunzător modificând informaţiile afişate pe ecran.

ContactMessageForm conţine componentele SVGTextBinding corespunzătoare elementelor afişate pe ecran de interfaţa recepţionării mesajelor, de exemplu textul mesajelor primite. În mod analog, **ContactMessageSend** implementează interfaţa pentru trimiterea mesajelor. MessageScreen realizează pentru interfaţa mesajelor ceea ce realizează ContactListScreen pentru interfaţa contactelor, tratând acţiunile utilizatorului interfeţei în mod corespunzător.

4.5. Configurarea şi comunicaţia sistemului

Arhitectura AWARE-HOSPITAL este proiectată utilizând un model peer-to-peer hibrid. Nodurile componente ale reţelei se pot conecta în diferite moduri în funcţie de informaţiile primite de la PeerManager, componenta responsabilă cu configurarea reţelei. Pentru pornirea serviciului contextului, aplicaţiile HospitalServer au nevoie de informaţiile privitoare la membrii personalului din spital. Informaţiile despre configurarea reţelei şi personalul spitalului sunt păstrate în fişiere XML care urmează să fie descrise în cele ce urmează.

Comunicaţia între aplicaţiile HospitalServer şi HospitalPhone se realizează prin intermediul socketurilor, ceea ce presupune definirea unui protocol de comunicare. Deoarece

aplicaţiile HospitalPhone rulează pe telefoane mobile, protocolul trebuie să fie cât mai compact şi mai simplu. Protocolul HPP (HospitalPhone protocol) îndeplineşte aceste cerinţe, fiind destinat în mod special aplicaţiilor mobile.

4.5.1 Fişierul pentru configurarea reţelei

Fişierul de configurare a reţelei se numeşte configXML.xml şi este specificat în format XML. Un exemplu de astfel de fişier pentru aplicaţiile HospitalPhone este următorul:

```
<jcafclients>
   <server name="Peer Server" port="50000">
      localhost
   </server>
   <peer name="OP" port="50000">
   localhost
   </peer>
   <client name="OP1" port="50000">
      localhost
   </client>
   <client name="OP2" port="50000">
      localhost
   </client>
</jcafclients>
```

Fiecare element al fişierului este caracterizat de două atribute specifice reprezentate de numele cu care este identificată staţia respectivă şi portul la care rulează serverul. Corpul fiecărui element conţine adresa staţiei pe care rulează serverul. Există trei tipuri de elemente conform celor trei tipuri de noduri la care se poate conecta aplicaţia HospitalPhone.

Elementul de tipul server este asociat componentei principale PeerManager şi are informaţiile referitoare la adresa şi portul la care rulează serviciile sale. Pe această conexiune aplicaţia HospitalPhone trimite mesaje text şi interoghează PeerManagerul să vadă dacă au intervenit modificări în fişierul de configurare.

Elementul de tipul peer se referă la aplicaţia HospitalServer care furnizează serviciul conştienţei contextului folosit de aplicaţia HospitalPhone pentru medierea conştienţei sociale. În sfârşit, nodurile de tipul client sunt tot aplicaţii HospitalServer, dar conexiunile la aceste noduri nu sunt folosite pentru serviciul de conştienţa contextului, ci pentru notificarea serviciilor contextului despre schimbarea statusului utilizatorului. Numele cu care sunt identificate nodurile HospitalServer sunt id-urile sălilor pe care le monitorizează.

4.5.2 Fişierul cu personalul spitalului

Fişierul Entities.xml conţine descrierea personalului spitalului în format XML. Personalul este reprezentat de medici sau asistente, în funcţie de specializare. Elementele acestui fişier sunt de trei tipuri după cum urmează: ”host”, ”medic” şi ”asistenta”. Anexa D conţine descrierea fişierului Entities.xml cu datele despre clinicienii spitalului.

Fiecare element al fişierului este caracterizat printr-un atribut care reprezintă id-ul entităţii respective. Corpul elementelor, cu excepţia elementului de tipul ”host”, conţine numele corespunzător personalului respectiv.

Elementul ”host” furnizează id-ul entităţii prin care este caracterizat utilizatorul telefonului mobil pe care rulează aplicaţia HospitalPhone respectivă. Unul din motivele pentru

care este necesar acest element îl reprezintă identificarea mesajelor pe care le trimite utilizatorul celorlalţi colegi de lucru.

Elementele ”medic” şi ”asistenta” descriu efectiv personalul spitalului în funcţie de specializarea fiecărui membru şi a numelor sale. Deoarece activitatea, statusul şi locaţia clinicienilor au un caracter dinamic nu sunt prezente în descriere personalului fişierul entităţilor.

4.5.3 Descrierea protocolului HPP

Comunicaţia între aplicaţii are la bază protocolul HPP care este definit pentru aplicaţiile specifice telefoanelor mobile, de exemplu aplicaţia HospitalPhone. Mesajele acestui protocol sunt de un anumit tip şi se compun din unul sau mai multe elemente. Elementele pot să fie caracterizate la rândul lor de unul sau mai multe atribute.

Se folosesc separatorii ”:” şi ”-” pentru a face distincţie între atributele unui element şi între elementele unui mesaj. Elementele mesajului diferă în funcţie de tipul mesajului HPP. Un mesaj HPP poate să aparţină unuia din următoarele tipuri:

- ”personal info”: specifică informaţiile despre personalul spitalului;
- ”personal context”: conţine informaţiile despre contextul fiecărui membru de personal;
- ”context changed”: încapsulează evenimentul generat de schimbarea contextului unei entităţi;
- ”host name”: folosit pentru a trimite numele de identificare al aplicaţiei HospitalPhone
- ”message”: reprezintă un mesaj text caracterizat prin sursă, destinaţie, prioritate şi text.

Având în vedere că formatele mesajelor sunt intuitive şi se aseamănă între ele, vom descrie în continuare doar mesajele de tipul ”context changed”. Aceste mesaje sunt trimise la apariţia unui eveniment generat de modificarea contextului unei entităţi.

Prima dată se adaugă în mesaj tipul mesajului separat de corpul mesajului prin separatorul ”:”. Apoi se transmite id-ul entităţii, tipul itemului care se modifică, valoarea nouă a contextului şi relaţia dintre entitate şi itemul contextului. Mai jos este prezentat un exemplu de mesaj ”context changed”:

#context changed:
entity id:med1:
context item:Status:
item value:away:
relationship:uses#

La recepţionare, aplicaţia parsează textul şi construieşte o tabelă corespunzătoare unui obiect HashTable în care adaugă pe rând atributele mesajului împreună cu valorile acestor atribute.

Capitolul 5 UTILIZAREA SISTEMULUI

Secţiunea aceasta prezintă setările necesare pentru a putea utiliza şi verifica sistemul AWARE-HOSPITAL. Pentru aceasta trebuie instalate anumite unelte şi trebuie urmăriţi câţiva paşi necesari pentru a rula aplicaţile sistemului.

Pentru a putea utiliza sistemul AWARE-HOSPITAL, trebuie rulate cele două aplicaţii: HospitalServer şi HospitalPhone. Aceste aplicaţii trebuie pornite într-o anumită ordine care să nu contravină fişierului de configurare. În mod normal, aplicaţiile HospitalServer trebuie rulate înaintea aplicaţiilor HospitalPhone deoarece furnizează serviciul conştienţei contextului pe care îl utilizează aplicaţiile HospitalPhone de pe telefoanele mobile. În anexa E sunt prezentate câteva imagini surprinse în timpul rulării celor două aplicaţii.

5.1 Rularea aplicaţiilor PeerManager şi HospitalServer

Aplicaţiile sunt furnizate sub forma unor directoare cu acelaşi nume, care conţin fişierele XML necesare rulării aplicaţiilor şi aplicaţiile propriu-zise sub forma unor fişiere cu extensia **.jar**. Înainte de a putea rula aplicaţia, trebuie instalat produsul **JDK**, cel puţin versiunea 6. Acest produs trebuie instalat deoarece HospitalServer şi PeerManager sunt aplicaţii Java.

În primul rând trebuie pornită aplicaţia PeerManager deoarece toate aplicaţiile HospitalServer şi HospitalPhone folosesc serviciile oferite de această aplicaţie. După ce a fost pornită această aplicaţie, se pot porni şi celelalte aplicaţii HospitalServer. Aplicaţiile sunt fişiere cu extensia **.jar**, care pot fi rulate prin comanda **java -jar PeerManager.jar,** respectiv **java -jar HospitalServer.jar .** După ce au fost pornite aplicaţiile HospitalServer, serviciul conştienţei contextului oferit de aceste aplicaţii devine disponibil aplicaţiilor de tipul HospitalPhone.

5.2 Rularea unei aplicaţii mobile HospitalPhone

Aplicaţia HospitalPhone este disponibilă sub forma unui director cu acelaşi nume corespunzător proiectului dezvoltat în NetBeans. În mod normal aplicaţia poate fi rulată pe telefoanele mobile care implementează MIDP 2.0 şi suportă aplicaţii Java. Pentru aceasta, aplicaţia a fost încărcată pe un server de aplicaţii mobile şi poate fi descărcată pe telefoane de la adresa **http://pcs.cruz-network.net/code.php** introducând codul de identificare 612304. De asemenea, fişierele jar şi jad necesare rulării aplicaţiei pot fi descărcate pe telefoane utilizând cablul de date. Fişierele HospitalPhone.jar, respectiv HospitalPhone.jad, se găsesc în **HospitalPhone/dist/**.

Aplicaţia mai poate fi rulată şi din NetBeans folosind emulatorul pentru telefoane mobile de care dispune. Pentru aceasta, trebuie instalat NetBeans, versiunea 6.5, în care a fost realizat proiectul. Nu e nevoie de nici un alt produs, deoarece NetBeans integrează emulatorul necesar simulării aplicaţiei HospitalPhone.

Capitolul 6 CONCLUZII ŞI DIRECŢII DE DEZVOLTARE

6.1 Concluzii

Lucrarea de faţă a prezentat proiectarea arhitecturii AWARE-HOSPITAL pentru medierea conştienţei sociale într-un spital, arhitectură construită pe diferite nivele care reprezintă diferite servicii. Arhitectura a fost proiectată pornind de la arhitectura AWARE existentă, dar AWARE-HOSPITAL foloseşte un model peer-to-peer hibrid în locul unui model client-server ca şi cel utilizat de AWARE.

Infrastructura contextului arhitecturii AWARE-HOSPITAL a fost implementată utilizând frameworkul JCAF. Deoarece frameworkul JCAF era disponibil doar prin intermediul Java RMI, acest framework a fost extins pentru comunicaţia prin socketuri, mai potrivită aplicaţiilor mobile utilizate pentru medierea conştiinţei sociale.

Pentru ilustrarea arhitecturii construite au fost implementate şi testate două aplicaţii: HospitalServer şi HospitalPhone. Ambele aplicaţii au funcţionat corect şi au îndeplinit cerinţele definite în introducerea lucrării. HospitalServer a fost implementată utilizând tehnologia Java şi a fost rulată pe staţii locale pentru a verifica serviciul conştienţei contextului.

Pentru medierea conştienţei sociale a fost implementată aplicaţia HospitalPhone folosind J2ME şi MIDP 2.0. Aplicaţia a fost testată utilizând emulatorul platformei NetBeans şi a răspuns cerinţelor de funcţionare corespunzătoare. Au fost pornite mai multe aplicaţii HospitalServer şi HospitalPhone care au comunicat între ele pentru realizarea conştienţei sociale. Aplicaţiile HospitalPhone au prezentat pe ecranele utilizatorilor informaţiile contextuale referitoare la activitatea, statusul şi locaţia colegilor de lucru.

În final, putem spune că proiectul a urmărit să proiecteze şi să exemplifice o platformă generică pentru medierea conştienţei sociale într-un spital, lucru care a fost realizat prin proiectarea arhitecturii AWARE-HOSPITAL şi construirea celor două aplicaţii: HospitalServer şi HospitalPhone.

6.2 Direcţii de dezvoltare

În viitor, s-ar putea încerca folosirea sistemului AWARE-HOSPITAL şi în alte medii diferite de cel al spitalului. Pentru aceasta, sistemul ar putea fi îmbunătăţit prin furnizarea unui mecanism de autentificare şi securitate, mecanism care nu este implementat în momentul de faţă. Unele din funcţiile necesare securităţii sistemului ar putea fi folosite din frameworkul JCAF.

De asemenea, s-ar putea păstra o istorie a contextului în cazul în care cineva ar fi interesat de monitorizarea participanţilor sistemului de conştienţă socială, lucru care a fost evitat în cadrul unui spital. Locaţia medicilor este monitorizată doar pentru anumite încăperi specifice, de exemplu sălile de operaţie, dar în alte situaţii s-ar putea dori monitorizarea tuturor încăperilor vizitate de membrii grupului.

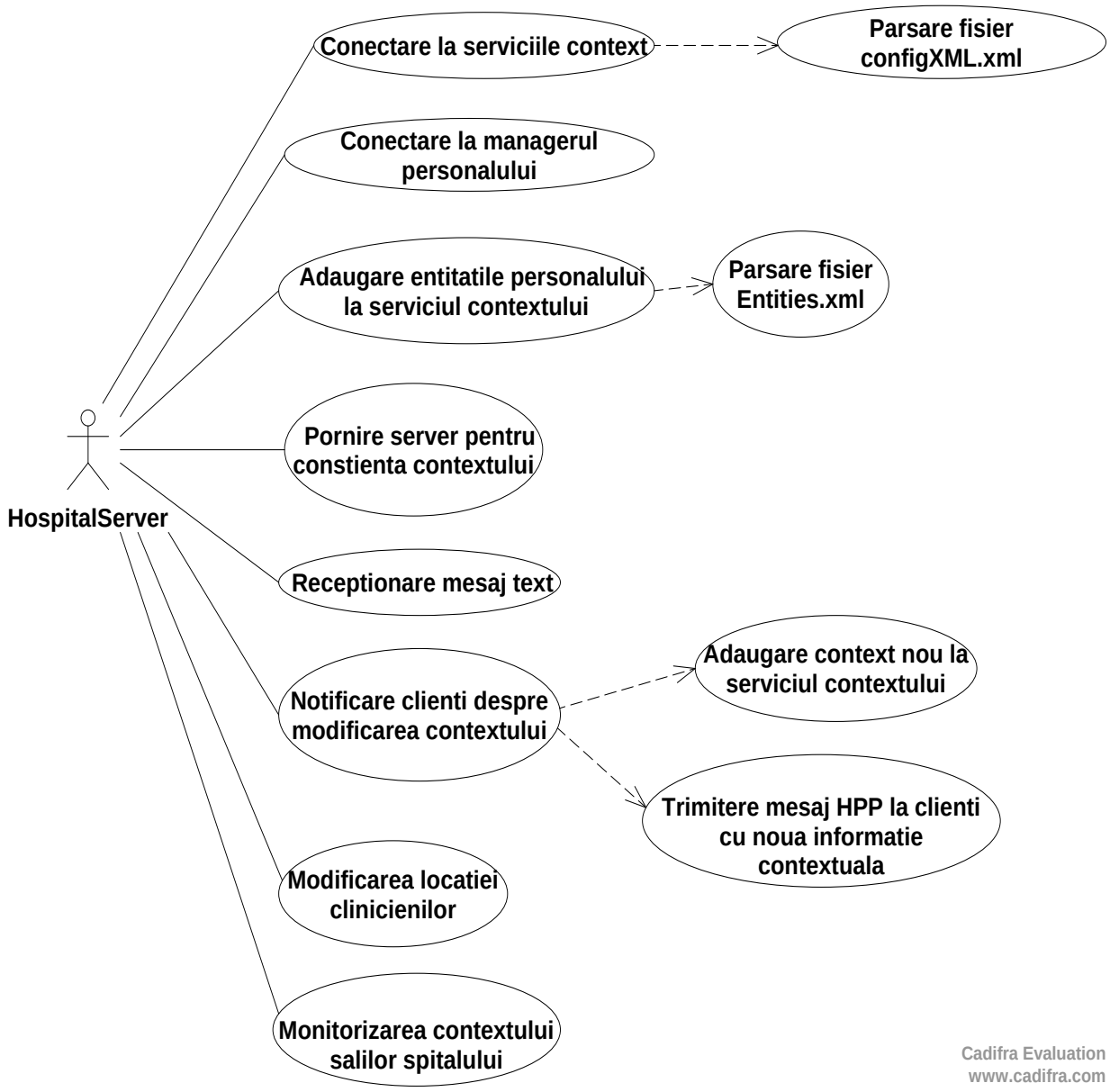
PeerManager funţionează ca un intermediar pentru mesajele dintre clinicieni, dar în viitor s-ar putea folosi direct telefoanele mobile pentru serviciul de mesagerie, utilizând clasele puse la dispoziţie de pachetul sockets. HospitalPhone implementează un serviciu context asemănător frameworkului JCAF, dar de o funcţionalitate limitată, iar acest serviciu ar putea fi folosit în viitor pentru a dezvolta aplicaţii mobile care să ofere direct serviciul conştienţei contextului.

Bibliografie

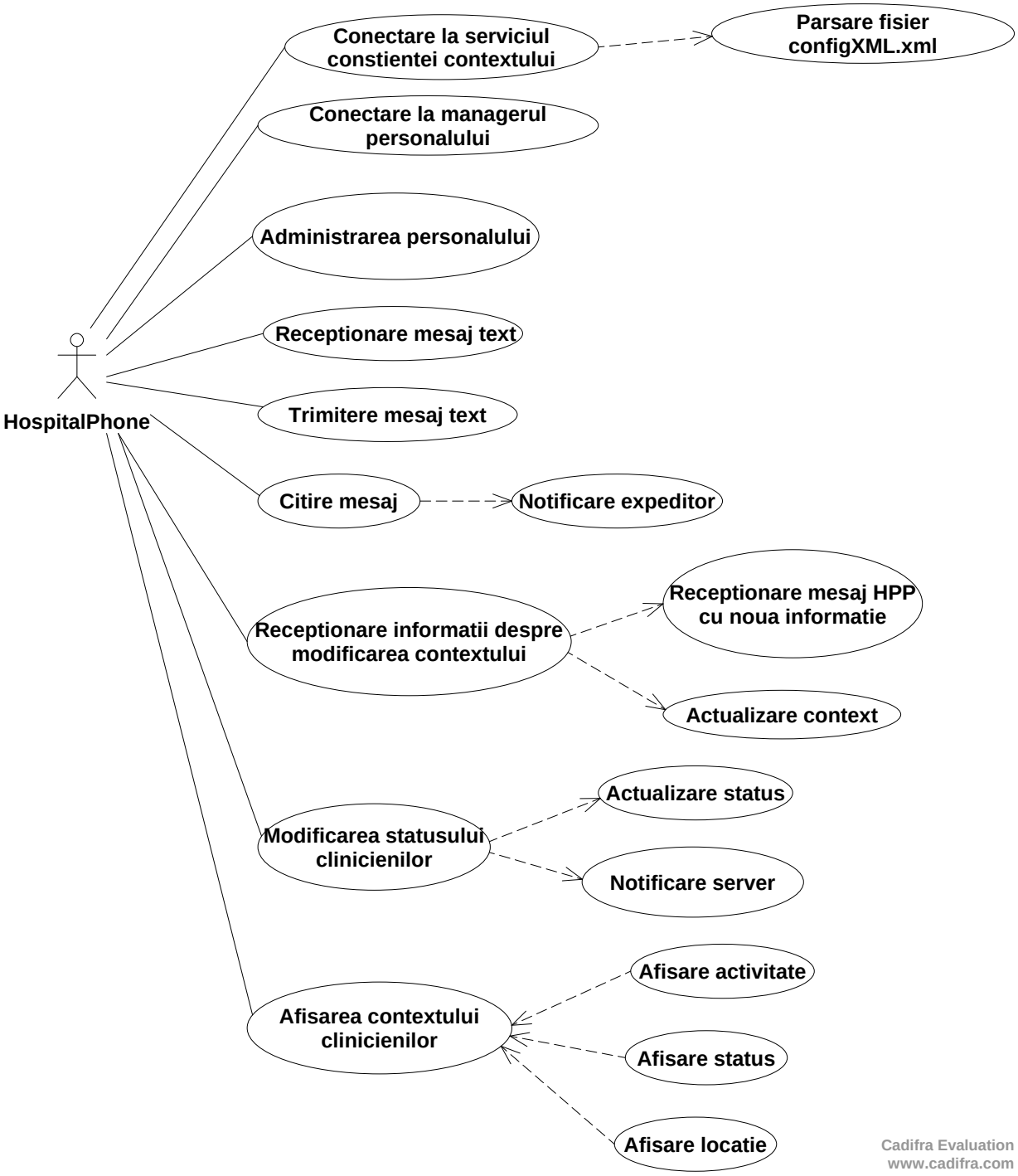
[1] **Anind K. Dey,** *Understanding and using constext*,2002.
[2] **Albrecht Scmidt,** *Ubiquitous computing – Computing in Context*, 2002.
[3] **Arnoud ten Hoedt,** *Context-Aware Mobile Health Application*, 2006.
[4] **Carol Hamer,** *Creating Mobile Games:Using Java ME Platform to Put the Fun into Your Mobile Device and Cell Phone*, ed. Springer-Verlag New York, 2007.
[5] **Guanling Chen, David Kotz,** *A survey of Context-Aware Mobile Computing Research*, 2000.
[6] **Jakob E. Bardram and Thomas R. Hansen,** *The AWARE architecture: supporting context-mediated social awareness in mobile cooperation*, 2004.
[7] **J. E. Bardram,** *The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. Technical Report, Centre for Pervasive Computing, Aarhus, Denmark*, 2003.
[8] **Jakob E. Bardram, Thomas R. Hansen and Mads Soegaard,** *Applying Mobile and Pervasive Computer Technology to Enhance Coordination of Work in an Surgical Ward*, 2007.
[9] **Jakob E. Bardram and Thomas R, Hansen.AwareMedia** – *A Shared Interactive Display Supporting Social, Temporal, and Spatial Awareness in Surgery*, 2006.
[10] **J. E. Bardram**,*Applications of ContextAware Computing in Hospital Work – Examples and Design Principles*,2004.
[11] **Jackob E. Bardram,** *Tutorial for the Java Context Awareness Framework(JCAF), version 1.5*, 2005.
[12] **Jackob E. Bardram,** *Design, Implemetation and Evaluation of the Java Context Awareness Framework(JCAF)*, 2005.
[13] **Marin Vlada,** *Limbajul SVG pentru grafică 2D în XML şi aplicaţii*, 2007.
[14] **Mathias Baldauf, Schahram Dustdar and Florian Rosenberg,** *A survey on context-aware systems*, 2007.
[15] **Nathalie Bricon-Souf and Conrad R. Newman,** *Context-awareness in healthcare: A review*, 2006.
[16] **Patrick Floreen, Greger Linden,** *Context-Aware Methods Course on Context-Aware Computing*, 2003.
[17] **Paul Pocatilu, Cristian Toma,** *The Development of Mobile Applications with Java*, 2004.

Lista figurilor

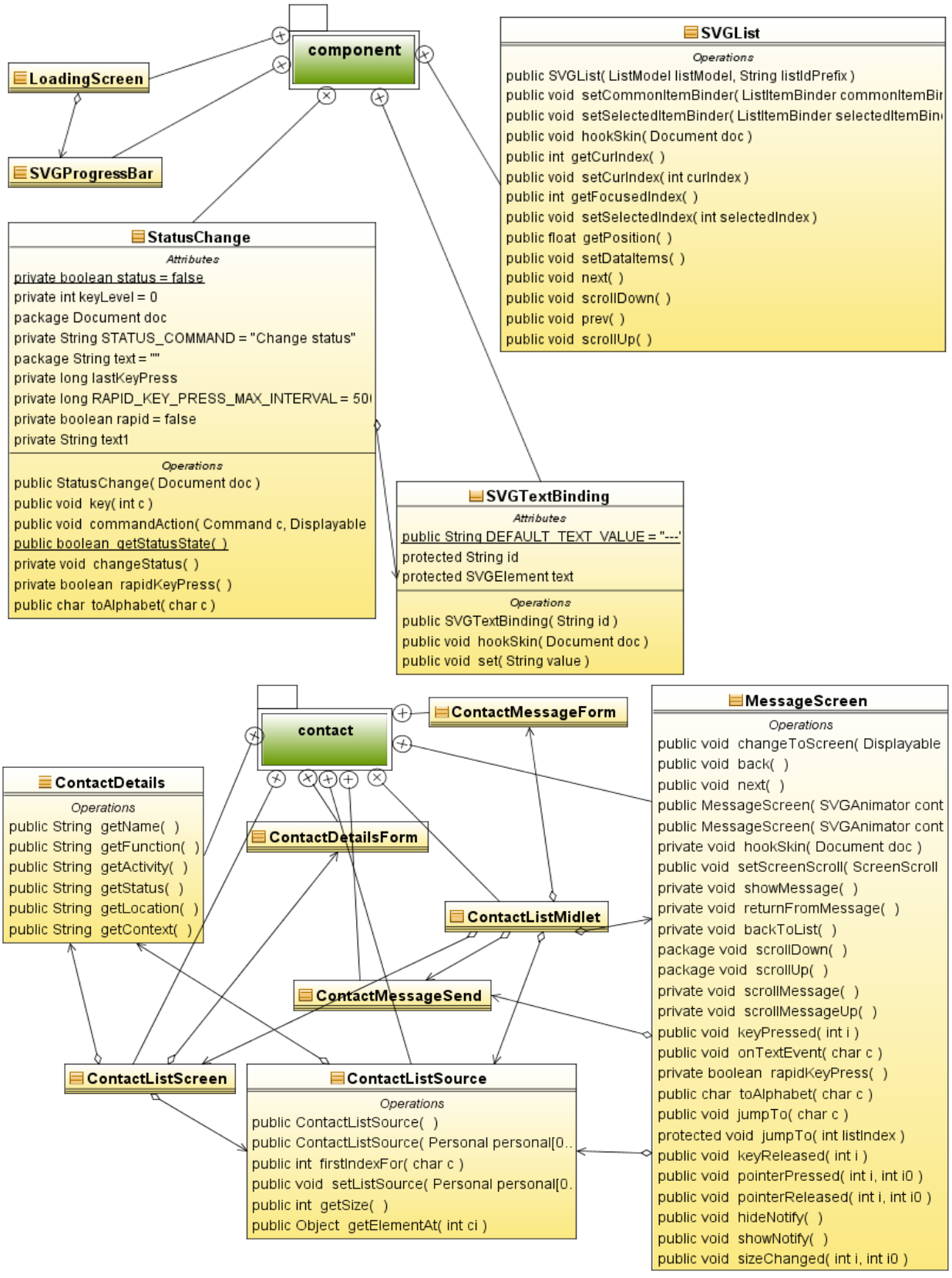
A. Cazuri de utilizare aplicaţia HospitalServer



B. Cazuri de utilizare aplicaţia HospitalPhone



C.Diagramele de clase ale pachetelor *component* si *contact*



D. Fişierul Entities.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entities nr="12">
   <host id="med1">
   </host>
   <medic id="med1">
      Maximilian
   </medic>
   <medic id="med2">
      Mircioiu
   </medic>
   <asistenta id="asist1">
      Iulia
   </asistenta>
   <asistenta id="asist2">
      Alina
   </asistenta>
   <medic id="med3">
      Popescu
   </medic>
   <medic id="med4">
      Florian
   </medic>
   <asistenta id="asist3">
      Livia
   </asistenta>
   <asistenta id="asist4">
      Lavinia
   </asistenta>
   <medic id="med5">
      Diculescu
   </medic>
   <medic id="med6">
      Ciortea
   </medic>
   <asistenta id="asist5">
      Carmen
   </asistenta>
   <asistenta id="asist6">
      Florina
   </asistenta>
</entities>
```

E. Imagini interfeţe utilizator – aplicaţiile HospitalServer şi HospitalPhone