

UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
SECȚIA CALCULATOARE

LUCRARE DE DIPLOMĂ

ADAPTAREA ROBOȚILOR
ÎNTR-UN SISTEM SENZITIV LA CONTEXT
PENTRU O CASĂ INTELIGENTĂ

MIHAI GABRIEL RAȚIU

2009

**UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
SECȚIA CALCULATOARE**

VIZAT DECAN
Prof.Dr.Ing. Sergiu NEDEVSCI
POTOLEA

VIZAT ȘEF CATEDRĂ
Prof.Dr.Ing. Rodica

**ADAPTAREA ROBOȚILOR ÎNTR-UN SISTEM
SENZITIV LA CONTEXT PENTRU O CASĂ INTELIGENTĂ**

Absolvent: **Mihai Gabriel Rațiu**

1. CONȚINUTUL PROIECTULUI:

A. Piese Scrise

Introducere

Studiu bibliografic

Fundamentare teoretică

Specificațiile și arhitectura sistemului

Proiectarea de detaliu

Utilizarea sistemului

Punerea în funcțiune și rezultate experimentale

Concluzii

Bibliografie

B. Anexe

CD conținând piesele scrise, codul sursă și uneltele folosite

2. LOCUL DOCUMENTAȚIEI: Universitatea Tehnică Cluj-Napoca

3. CONSULTANȚI: Asist. Ing. Lucia Văcariu

4. DATA EMITERII TEMEI: 1.10.2008

5. TERMEN DE PREDARE: 19.06.2009

CONDUCĂTOR DE PROIECT

Asist. Ing. Lucia Văcariu

SEMNĂTURĂ ABSOLVENT

Mihai Gabriel Rațiu

Cuprins

1	INTRODUCERE	1
2	STUDIU BIBLIOGRAFIC.....	3
2.1	Case inteligente	3
2.2	Sisteme senzitive la context	5
2.2.1	Concepte de bază.....	5
2.2.1.1	Context	5
2.2.1.2	Senzitivitate la context	5
2.2.1.3	Sistem Senzitiv la Context	5
2.2.2	Paradigma Ubiquitous Computing	6
2.2.2.1	Definiții	6
2.2.2.2	Caracteristici.....	6
2.2.3	Ontologie	7
2.2.3.1	Definiții	7
2.2.3.2	Caracteristici.....	7
2.2.4	Agenți senzitivi la context.....	8
2.2.4.1	Paradigma agentului	8
2.2.4.2	Agenți BDI	8
2.3	Arhitecturi de sisteme senzitive la context.....	9
2.3.1	Arhitecturi de sisteme senzitive la context locale	9
2.3.1.1	Modelul arhitectural stratificat „Context Stack”	9
2.3.2	Arhitecturi de sisteme distribuite senzitive la context	11
2.3.2.1	Serviciile Web Semantice	12
2.3.2.2	Arhitectura Serviciilor Web Semantice.....	13
2.4	Medii de dezvoltare ale aplicațiilor senzitive la context	15
2.5	Aplicații în domeniul caselor inteligente	16
2.6	Dificultăți și riscuri întâmpinate în crearea aplicațiilor senzitive la context.....	18
2.7	Proiecte ample de cercetare în desfășurare	20
3	FUNDAMENTARE TEORETICĂ	22
3.1	Modelarea formală a contextelor.....	22
3.1.1	Agenți bazați pe cunoștințe	22
3.1.1.1	Baze de date de cunoștințe	22
3.1.2	Reprezentarea cunoștințelor	23
3.1.3	Folosirea ontologiilor în modelarea contextelor	24
3.1.3.1	Modelarea contextului cu ontologii.....	24
3.1.3.2	Proiectarea ontologiilor de context	25
3.1.3.3	Limbajul OWL	28
3.1.4	Extensii probabilistice ale ontologiilor	29
3.2	Metode folosite pentru modelarea bazelor de cunoștințe.....	32
3.2.1	Metodologii existente pentru crearea bazelor de cunoștințe	32
3.2.2	Unelte de dezvoltare a ontologiilor	33
3.2.2.1	Protégé 4.....	34
3.3	Metode folosite pentru dezvoltarea agenților.....	35
3.3.1	Platforme de dezvoltare a agenților.....	35
3.3.2	Platforma JADE	37

3.3.3	Mediul JADEx 0.96.....	38
3.3.3.1	Modelul BDI definit în Jadex.....	38
3.3.3.2	Definirea agenților în Jadex	40
4	SPECIFICAȚII ȘI ARHITECTURĂ SISTEM.....	41
4.1	Specificațiile sistemului	41
4.1.1	Controlul casei inteligente.....	41
4.1.1.1	Coordonarea acțiunilor	41
4.1.1.2	Executarea operațiilor	41
4.1.1.3	Interacțiunea cu utilizatorii.....	41
4.1.2	Monitorizarea casei inteligente	42
4.1.2.1	Achiziția datelor de context.....	42
4.1.2.2	Procesarea datelor de context.....	42
4.1.2.3	Propagarea datelor de context	42
4.1.3	Adaptarea roboților la casa inteligentă.....	43
4.1.3.1	Reprezentarea adecvată a roboților	43
4.1.3.2	Comunicarea dintre roboți.....	43
4.1.3.3	Serviciile roboților din casa inteligentă.....	43
4.2	Arhitectura sistemului	44
4.2.1	Nivelul Senzorial.....	44
4.2.2	Nivelul Dispozitivelor	44
4.2.3	Nivelul Roboților.....	45
4.2.4	Nivelul Decizional.....	45
4.2.5	Nivelul de control și monitorizare.....	46
4.2.6	Nivelul fizic.....	46
5	PROIECTARE DE DETALIU	47
5.1	Analiza cerințelor	47
5.2	Proiectarea arhitecturii	48
5.2.1	Sistemul de roboți.....	48
5.2.1.1	Agentul Robot Manager	49
5.2.1.2	Agentul Robot Supraveghetor.....	50
5.2.1.3	Agentul Robot Misiune	51
5.2.1.4	Agentul Robot Divertisment	54
5.2.2	Sistemul de senzori.....	54
5.2.3	Sistemul de dispozitive.....	55
5.2.4	Sistemul de control.....	56
5.2.5	Sistemul de monitorizare.....	56
5.2.6	Proiectarea bazei de cunoștințe	57
5.3	Detalii de implementare	57
5.3.1	Caracteristici generale legate de implementarea sistemului	57
5.3.1.1	Pachetele aplicației	58
5.3.2	Implementarea agenților.....	58
5.3.2.1	Implementarea fișierelor de descriere a agenților	58
5.3.2.2	Implementarea planurilor	60
5.3.3	Interfețe Utilizator	61
5.4	Scenariu de testare.....	61
6	UTILIZAREA SISTEMULUI	62
7	PUNERE ÎN FUNCȚIUNE ȘI REZULTATE EXPERIMENTALE	64

7.1	Tehnologii folosite	64
7.1.1	Cerințe Hardware	64
7.1.2	Cerințe Software.....	64
7.1.3	Instalarea aplicațiilor folosite	64
7.1.4	Configurarea proiectului	65
7.1.5	Pornirea Platformei de Agenți.....	65
7.1.6	Pornirea agentului pentru baza de cunoștințe a sistemului	66
7.1.7	Pornirea agentului manager de senzori	66
7.1.8	Pornirea agentului manager de dispozitive	66
7.1.9	Pornirea agentului de monitorizare	66
7.1.10	Pornirea agentului manager pentru roboți.....	67
7.2	Probleme întâmpinate și modul de rezolvare	67
7.2.1	Probleme de pornire a platformei de agenți	67
7.2.2	Probleme de pornire a agenților	67
7.3	Rezultate experimentale	67
7.3.1	Crearea roboților	67
7.3.2	Adaptarea roboților	70
8	CONCLUZII.....	72
8.1	Realizări	72
8.2	Direcții de dezvoltare	73
9	BIBLIOGRAFIE	75

Lista de Tabele

Tabel 2.1 Clasificarea roboților utilizați în locuințe	4
Tabel 3.1 Comparatie între SOCAM și CoOL privind modelarea ontologiilor de context	27
Tabel 3.2 O parte din regulile de argumentare OWL.....	29

Lista de Figuri

Figura 2.1 Tendința în vânzarea roboților de casă pe piața japoneză	4
Figura 2.2 Modelul arhitectural Context Stack	10
Figura 2.3 Comparație între SemanticURS și sistemele tradiționale	12
Figura 2.4 Clasele componente ale unei ontologii de tip OWL-S	13
Figura 2.5 Arhitectura detaliată a Semantic URS	14
Figura 2.6 Procedura de integrare automată a unui agent SemanticURS	15
Figura 2.7 Diagrama de interacțiune între modulele componente ale JCAF	15
Figura 2.8 Arhitectura generală cu diagrame de colaborare a JCAF	16
Figura 2.9 Arhitectura de tip OSGi pentru o casă senzitivă la context	17
Figura 3.1 Cazul general de programare a unui agent bazat pe cunoștințe	23
Figura 3.2 Exemplu de modelare a unei ontologii superioare	24
Figura 3.3 Definire ontologii specifice reprezentate prin rețele Bayesiene	26
Figura 3.4 Model de proiectare a ontologiilor contextuale, folosind SOCAM	26
Figura 3.5 (a) Schema relațională și (b) Modelul probabilistic pentru o casă inteligentă	30
Figura 3.6 Arhitectura SOCAM cu ontologii specifice și modele probabilistice	31
Figura 3.7 Procesul de dezvoltare a unei baze de cunoștințe folosind Methontology	32
Figura 3.8 Arhitectura generală a unei platforme de agenți și comunicarea între platforme ...	36
Figura 3.9 Platforma, depozitele și comunicațiile între platforme JADE	38
Figura 3.10 Diagrama de colaborare între componentele unui agent Jadex	39
Figura 4.1 Diagrama arhitecturală a sistemului	45
Figura 5.1 Diagrama de interacțiune între actorii și sistemul casei inteligente	48
Figura 5.2 Diagrama de obiective a sub-sistemului de roboți	49
Figura 5.3 Obiectivele robotului transportor în casa inteligentă	52
Figura 5.4 Sistemul de senzori al casei inteligente	55
Figura 5.5 Sistemul de dispozitive al casei inteligente	56
Figura 5.6 Obiectivele sistemului de control al casei inteligente	56
Figura 5.7 Antetul fișierului de descriere a agenților ADF	59
Figura 5.8 Schema de definire a unui agent Jadex	59
Figura 5.9 Definirea planului de mișcare pentru robotul transportor	60
Figura 6.1 Reprezentarea robotului transportor	62
Figura 6.2 Panoul de control al simulatorului	62
Figura 6.3 Consola agentului pentru baza de cunoștințe	63
Figura 7.1 Pornirea platformei de agenți din Eclipse	65
Figura 7.2 Platforma de agenți Jadex	66
Figura 7.3 Clasele ontologice specifice agenților roboți	68
Figura 7.4 Proprietăți individuale specificate în modelul ontologic - robot supervizor	68
Figura 7.5 Mesajul trimis către agentul bazei de cunoștințe pentru identificarea roboților	69
Figura 7.6 Mesajul cu conținutul instanțelor de roboți trimis de agentul bazei de cunoștințe	69
Figura 7.7 Mesaje ACL transmise între agentul robot transportor și sistemul casei inteligente	70
Figura 7.8 Consola agentului Manager pentru roboții casei inteligente și planurile executate	71

1 INTRODUCERE

Acest capitol prezintă motivațiile temei, încadrarea temei în domeniile pe care le acoperă, problemele apărute în elaborarea sistemelor senzitive la context pentru case inteligente, abordarea acestor probleme în elaborarea sistemului dezbătut în lucrarea curentă.

Cercetările curente în domeniul sistemelor senzitive la context adaugă unele soluții problemelor legate de adaptarea contextuală a serviciilor, dispozitivelor electronice sau sistemelor la mediile fizice sau computaționale. Tema de față propune o soluție pentru crearea unui sistem senzitiv la context pentru o casă inteligentă și o soluție de adaptare a roboților într-un astfel de mediu.

Tehnologiile pentru case inteligente se referă la un domeniu colectiv care cuprinde tehnologii informaționale și comunicaționale care pot fi folosite în locuințe, unde diferite componente comunică printr-o rețea locală. Tehnologia este folosită pentru a supraveghea, atenționa sau facilita diferite acțiuni care se petrec în locuință, actorii principali fiind locuitorii acesteia. Tehnologiile pentru case inteligente dăruiesc o flexibilitate și funcționalitate total diferită față de instalațiile convenționale, datorită integrării unităților programabile care comunică prin mesaje de rețea.

Crearea sistemelor senzitive la context, distribuite și omniprezente se confruntă cu un număr mare de probleme care se ridică în momentul proiectării lor, începând de la problemele specifice care apar în momentul construirii unui sistem distribuit, probleme care au legătură cu integrarea diferitelor componente, asigurarea interoperabilității între dispozitive de calcul eterogene, scalabilitatea, extensibilitatea, securitatea sau toleranța la erori. Pe lângă problemele enumerate se mai adaugă problemele specifice tehnologiilor senzitive la context care adaugă problemele legate de procesarea datelor de context achiziționate, care de cele mai multe ori conțin zgomote sau erori care trebuie identificate și filtrate, iar din aceste date venite de la nivelul fizic al aplicației este necesară deducerea de noi contexte bazate pe procedee preluate din domeniul inteligenței artificiale.

Unul din cele mai interesante puncte de cercetare în domeniul sistemelor senzitive la context pentru case inteligente este centrat pe descoperirea intențiilor locatarilor casei și reacția sistemului care va furniza un răspuns optim pentru a satisface nevoile și doleanțele locatarilor. Chiar dacă în ultimii ani s-au construit multe sisteme senzitive la context, lipsește încă un model general care să fie stabilit ca un ghid al proiectării. Acest lucru a determinat duplicarea efortului construirii unui astfel de sistem, precum și mari probleme de interoperabilitate între cei care au construit astfel de sisteme. Soluțiile curente de modelare a informațiilor contextuale sunt bazate pe diferite reprezentări, de exemplu XML, grafuri, diagrame UML și alte soluții împrumutate din alte domenii ale științelor informatice. Tendința generală abordată în acest caz este folosirea ontologiilor. Ontologiile furnizează un limbaj comun pentru entitățile implicate în sistem, care trebuie să colaboreze pentru a atinge anumite obiective legate de adaptarea informațiilor de context. Alte avantaje ale folosirii ontologiilor sunt crearea de inferențe și argumentări logice, re folosirea contextelor, prin crearea de ontologii comune domeniului și ontologii specifice.

Un alt aspect important în crearea unor astfel de sisteme este configurarea contextelor, care se referă la faptul că sistemul se bazează pe niște reguli fixe, neconfigurabile care guvernează răspunsul său la schimbări succesive de context, fără posibilitatea de reconfigurare dinamică a acestora. Odată ce sistemul detectează contextul curent, trebuie să decidă ce tip de comportament să aplice pentru a satisface nevoile locatarilor. Astfel roboții inteligenți, chiar dacă ar putea învăța din situațiile trecute pentru a repeta comportamentul, trebuie să ia în considerare și situația în care se așteaptă un alt tip de comportament pentru situațiile noi sau întâlnite în prealabil. În această lucrare se va demonstra că pentru adaptarea

roboților la sistemul senzitiv la context al casei nu este de ajuns folosirea cunoștințelor despre context și regulile predefinite sau cele deduse ulterior, toate acestea nu vor conduce de fiecare dată la răspunsul optim care să satisfacă preferințele utilizatorului. Dar de multe ori acest răspuns al sistemului se poate apropia de cerințe.

Pentru satisfacerea cerințelor exprimate în problema propusă este necesar ca în primul rând din soluția de proiectare a arhitecturii să rezulte o arhitectură flexibilă, care permite adaptarea a noi componente. Astfel soluția propune construirea unui sistem multi-agent distribuit descentralizat bazat pe agent, fiecare componentă a sistemului fiind un agent, iar comunicarea între componente fiind realizată prin comunicarea de mesaje între agenți. Folosirea agenților aduce o generalizare mare a problemei, deoarece prin agenți se pot reprezenta diferite tipuri de entități sau actori, printre care și roboți, agenții putând fi configurați să emuleze comportamentul unui robot din sistemul senzitiv la context.

Cerințele și direcțiile de proiectare și cercetare abordate în această lucrare sunt următoarele:

- *Studiul conceptelor de bază folosite în domeniul sensibilității la context și a roboților* – Lucrarea furnizează un dicționar cu explicații și amănunte legate de conceptele de bază privind sensibilitatea la context, sisteme sensitive la context și roboți;
- *Studiul metodelor de modelare a contextelor și folosirea ontologiilor* – Se face o analiză a avantajelor și dezavantajelor proiectării contextelor cu ontologii;
- *Studiul diferitelor arhitecturi existente de sisteme sensitive la context și o comparație între acestea* – Se studiază diferite modele arhitecturale pentru sisteme sensitive la context. De asemenea se valorifică aportul adus de proiectarea cu agenți la infrastructura sistemului;
- *Studiul metodelor și metodologiilor de realizare a adaptării agenților la un sistem senzitiv la context* – Pentru a îndeplini obiectivele lucrării este necesară studierea diferitelor metode de dezvoltare a agenților, unele folosite pentru crearea agenților precum și platforme de simulare a sistemelor multi-agent. Se face și un studiu de fezabilitate pentru toate aceste platforme, în încercarea de a alege cea mai bună metodă pentru atingerea scopului lucrării;
- *Proiectarea arhitecturii sensitive la context în compatibilitate cu adaptarea roboților* – În partea de proiectare a arhitecturii trebuie luate în considerare tipurile de agenți folosiți pentru modelarea entităților din sistemul senzitiv la context al casei inteligente, tipurile de agenți folosiți pentru roboți, modul de colaborare între ei, cunoștințele împărțite, obiectivele și planurile executate pentru atingerea obiectivelor și dependențele dintre ei;
- *Implementarea unei aplicații care să simuleze adaptarea roboților în sistemul senzitiv la context al casei inteligente* – Construirea unui sistem multi-agent este un proces complex care include provocările și problemele referitoare la sisteme distribuite, probleme rezultate din flexibilitatea și comunicarea la distanță între componente și simularea adaptării într-o aplicație vizuală demonstrativă;
- *Descoperirea și tratarea provocărilor existente și a dezvoltărilor ulterioare pentru sistemul senzitiv la context cu roboți* – Prezentarea unor direcții de dezvoltare ulterioară a sistemului.

Implementarea unei adaptări complete a roboților este în afara scopurilor lucrării deoarece acest lucru implică cercetări și expertize complexe în domeniul electronicii, ciberneticii, psihologiei și alte domenii auxiliare care pot interveni în proiectarea roboților inteligenți.

2 STUDIUL BIBLIOGRAFIC

2.1 Case inteligente

Ultimii ani au fost martorii avansării rapide a tehnologiilor ce privesc casele inteligente, printre care maturizarea științelor privind rețelele, totodată cu creșterea varietății protocoalelor de comunicații prin cablu și wireless (fără fir), precum și pe sisteme senzor-actuator omniprezente. Utilizatorii locali au dorit servicii din ce în ce mai personalizate pentru casele lor, precum automatizări, securitate, servicii de monitorizare, divertisment și medicale. În particular, multe proiecte ce refereau case inteligente au avut ca țintă utilizatori din categoria persoanelor în vârstă și a persoanelor cu dizabilități, cu scopul de a le ușura traiul și pentru a le oferi cât mai multă independență prin asistarea lor în viața de zi cu zi [10]. O componentă esențială în tehnica computațională ce privește casele inteligente și nu numai este **senzitivitatea la context**. Serviciile în medii omniprezente și mobile trebuie să fie sensibile la context pentru a se putea adapta la schimbări rapide de situație. Prin context ne referim la orice informație fizică sau conceptuală legată de mediul executiv al unui serviciu.

Datorită progresului în comunicațiile prin rețea, mulți cercetători au propus sisteme de roboți bazate pe Internet, care au ca scop controlul de la distanță și monitorizarea dispozitivelor și senzorilor prin rețea. Folosind avantajele date de Internet, aceste sisteme permit utilizatorilor din toate colțurile lumii să viziteze muzee, să navigheze pe mare, și multe alte facilități, toate acestea fără a fi necesară prezența la fața locului. Aceste facilități au un foarte mare potențial pentru industrie, educație, timp liber, securitate.

Datorită creșterii rapide a tehnologiilor computaționale de tip pervaziv a apărut o nouă paradigmă numită **„ubiquitous computing technology”** (omniprezența dispozitivelor de calcul), care este o extensie a paradigmei „desktop computing” (dispozitive de calcul locale), în care informația procesată a fost integrată complet în obiectele și activitatea de zi cu zi, în care utilizatorul poate folosi simultan mai multe sisteme, fără a fi conștient că acționează în acest fel. Paradigma mai este numită și „everyware” ca un joc de cuvinte inspirat din paradigmele existente și deja celebre „software”, „hardware” ș.a.m.d.

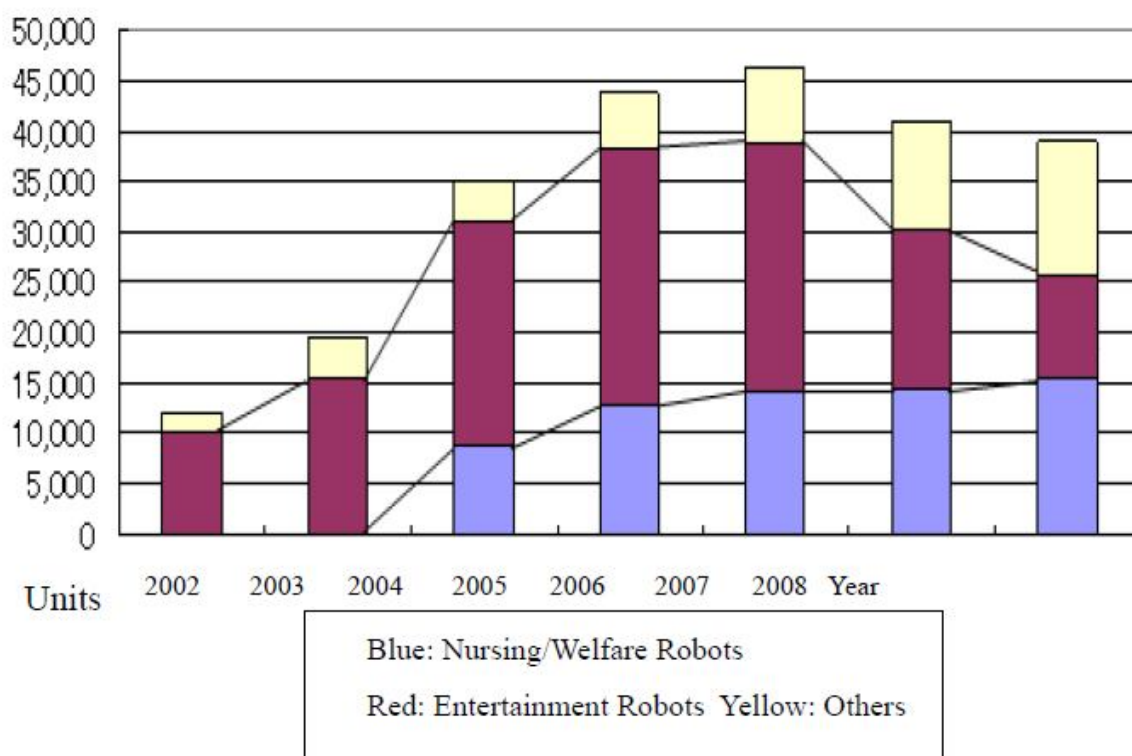
Ubiquitous Robotic Companion (URC) este un concept prin care roboții care întreprind servicii („service robots”) [06], furnizează utilizatorilor serviciile dorite, oricând și oriunde în mediile computaționale pervazive. Pentru a îndeplini viziunea URC, una din cerințele esențiale ale sistemelor de roboți este să suporte omniprezența serviciilor. Acest lucru înseamnă că robotul trebuie să fie folosibil chiar dacă apar modificări în sistemul pervaziv, care trebuie să permită ca roboții să opereze cu senzorii și dispozitivele existente, nu să fie preprogramați pentru ele.

Conceptul de „service-robot” [13] include toți roboții care vin în ajutorul locatarilor unei case. O clasificare a acestui tip de roboți poate fi observată în Tabelul 2.1. Astfel în ultimii ani s-au creat cu succes roboți care execută acțiuni precum spălarea hainelor, aspirare în casă, comunicare, precum și susținerea și ajutarea persoanelor vârstnice.

Tabel 2.1 Clasificarea roboților utilizați în locuințe

Arie de interes	Roboți existenți	Aplicabilitate
Roboți pentru comunicare	NEC “PaPeRo”	Securitatea domiciliului
	Hitachi “EMIEW/EMIW2”	Monitorizare la distanță
	AIST “PARO”	Terminal de informare
	ART “Robovie”	Îngrijire medicală / dădacă
Roboți pentru ajutor casnic	Fuji Heavy Industry	Ajutor casnic
	iRobot “Roomba”	Aerisirea și reglarea temperaturii încăperilor
	Toshiba “ApriPoko”	Curățenie
	Mitsubishi Heavy Industry “Wakamaru”	Securitate și monitorizare
Roboți folosiți pentru munci mai grele	Matsushita Electric Works “HOSPI”	Servicii în instituții publice (școli, spitale, etc.)
	Secom “Secom Robot X”	Dădăcire
	Cyberdyne	Îngrijire medicală
	Toyota IRT	Securitate

În Japonia, conform companiei Seed Planning Company, cota de piață pentru roboții de tip „service-robot” a crescut de la 1,9 miliarde yen la 6,7 miliarde yen în 2006. În 2007 deoarece marile companii s-au retras de la a mai comercializa roboți de amuzament precum roboți de companie, vânzările au scăzut per total, însă, vânzarea roboților de menaj și a celor dădacă a continuat să crească așa cum se poate vedea și în Figura 2.1.



(Source: Seed Planning Inc.)

Figura 2.1 Tendința în vânzarea roboților de casă pe piața japoneză

2.2 Sisteme senzitive la context

2.2.1 Concepte de bază

2.2.1.1 Context

În general, contextul reprezintă orice informație care poate fi folosită pentru a caracteriza starea unei entități [16]. O entitate poate să fie la rândul ei o persoană, un loc, o situație sau un obiect care este relevant interacțiunii dintre un utilizator și o aplicație. Aplicațiile senzitive la context caută răspuns pentru circumstanțele în care o entitate se poate afla, și folosesc aceste informații pentru a determina ce situații apar în aceste circumstanțe.

În [15] este dată următoarea definiție pentru un context: Un context se referă la ceea ce furnizează surse adiționale de informație, aduce contribuție la înțelegerea situației, pentru un anumit punct de interes vizat.

Conceptul de context a fost introdus în multe domenii din știința calculatoarelor, acestea incluzând, procesarea limbajelor formale și naturale (gramatici dependente/independente de context), grafică pe calculator, procese decizionale, procesare de informații, dispozitive de calcul omniprezente sau securitate. Scopul utilizării contextului este adăugarea adaptabilității sistemelor și suport în luarea deciziilor.

2.2.1.2 Senzitivitate la context

Senzitivitatea la context este un punct foarte important și esențial în mediile computaționale omniprezente, pentru adaptarea entităților computaționale la schimbările care pot interveni, cum ar fi schimbările necesităților utilizatorilor sau a capacităților tehnice [17]. Fundamentul senzitivității la context este un model de reprezentare cât mai formal al contextului, care este necesar pentru a reprezenta contextul într-o manieră interpretabilă de către calculator. În medii distribuite, este primordial ca această informație contextuală să poată fi interpretată de entitățile computaționale diferite, ceea ce oferă interoperabilitate. Mai mult, este necesară argumentarea cunoștințelor despre context, de exemplu, pentru a rezolva inconsistențele datelor venite de la senzori, sau pentru a deduce context de nivel înalt, din informații vagi și eronate venite de la senzori.

Senzitivitatea la context este un subiect care tratează modul în care dispozitivele pot interfera și înțelege contextul curent bazat pe informațiile preluate de la senzori [18]. Prin înțelegerea contextului curent (o situație, un mediu) în care se află utilizatorul sau dispozitivul, sistemul poate lua în mod inteligent o decizie pentru a face un anumit tip de acțiune sau cel puțin să notifice și să ceară permisiunea utilizatorului pentru a confirma acțiunea pe care o va executa.

Sistemele senzitive la context bazate pe recomandări sunt o componentă esențială în acest tip de medii pervazive. Scopul acestor agenți inteligenți este să determine priorități sau să sugereze acțiunile de care utilizatorul este interesat, luând în calcul contextul interacțiunii.

2.2.1.3 Sistem Senzitiv la Context

Schilt, în 1995, spunea că sistemele senzitive la context sunt construite astfel încât să se adapteze locației sau locațiilor în care sunt folosite, ansamblului de persoane și obiecte, dispozitivelor accesibile, precum și modificărilor acestor entități de-a lungul timpului. Un sistem cu astfel de capacități monitorizează mediul și reacționează la modificările apărute în acel mediu [22].

Senzitivitatea la context permite sistemelor să întreprindă acțiuni automate, reducând excesul de implicare din partea utilizatorilor și furnizând soluții proactive de asistență inteligentă.

Informația perceptuală despre mediu este ingredientul esențial al sistemelor senzitive la context, ceea ce le distinge de sistemele computaționale clasice. Aplicabilitatea acestor tipuri de sisteme este foarte mare, incluzând extragerea informațiilor senzitive la context, sisteme notificatoare, servicii mobile senzitive la context, precum și medii senzitive la context cu potențiale beneficii aduse societății, de la îngrijire automată și proactivă a sănătății, comerț electronic (e-commerce), la sisteme militare și automobile mai sigure [22]. De asemenea, se poate utiliza senzitivitatea la context și în sisteme de securitate de acasă sau din locuri publice.

2.2.2 Paradigma Ubiquitous Computing

2.2.2.1 Definiții

Cuvântul „ubiquitous” (în traducere care se împrășteie peste tot, omniprezent) poate fi transpus înspre domeniul tehnic ca „a fi sau a exista peste tot în același timp”, „întâlnit în mod frecvent”, „răspândit peste tot”, referindu-se la faptul că tehnologia este peste tot și o folosim tot timpul [19].

Din cauza omniprezenței acestor tehnologii tindem să le folosim fără a ne gândi la un anumit dispozitiv, la o anumită unealtă gen calculator sau telefon mobil. În schimb ne gândim doar la acțiunea pe care vrem să o întreprindem, făcând astfel acest tip de tehnologii invizibile din punctul de vedere al utilizatorului.

Cercetătorii de la Research Center for Educational Technology [19], în urma muncii lor și a observațiilor bazate pe experiență și pe cunoștințele existente au dezvoltat o definiție care poate fi aplicată în domeniul învățământului pentru a răspândi această noțiune nouă în rândul studenților: „Mediile computaționale pervazive sunt medii de învățare în care toți studenții au acces la o varietate de servicii și dispozitive digitale, incluzând calculatoare conectate la Internet, precum și dispozitive mobile, oriunde și oricând au nevoie de ele. Noțiunea noastră de ubiquitous computing este mai mult axată pe relația: de la mai mulți-la mai mulți (many-to-many) decât pe unu-la-unu (one-to-one) sau unu la-mai mulți (one-to-many) și include ideea de tehnologie oricând la dispoziția studenților”.

Mark Weiser, cercetător la Xerox, a fost primul care a introdus termenul de ubiquitous computing și a definit termenul astfel: „Ubiquitous computing este metoda prin care se îmbunătățește utilizarea calculatorului, făcând mai multe dispozitive disponibile prin mediul fizic, acestea fiind invizibile pentru utilizator” [20].

Marcia Riley, de la Georgia Institute of Technology, Atlanta, spunea că UC, sau „tehnologia calmă” este o schimbare de paradigmă, în care tehnologia devine practic invizibilă, dar este peste tot prezentă în viețile noastre [20].

2.2.2.2 Caracteristici

Folosind dispozitivele apărute în ultima perioadă utilizatorii tind să comunice prin diferite metode, să fie mult mai activi, să folosească spațiul geografic și temporal în mod diferit, și să dețină mai mult control. Astfel, datorită nevoilor utilizatorilor, UC a fost concepută având următoarele caracteristici:

- Dispozitivele sunt prezente peste tot în jurul nostru;
- Sunt interconectate și comunică unul cu celălalt;
- Nu necesită atenție continuă din partea utilizatorului pentru a funcționa;
- Fiecare dispozitiv tinde să fie cât mai specializat;
- Sunt invizibile, fiind ascunse în mediul în care sunt folosite;

În [20] se dă următoarea caracterizare pentru UC. Astfel paradigma permite tehnologiei să fie:

- globală și locală;
- socială și personală;
- publică și privată;
- invizibilă și vizibilă;
- creatoare de cunoștințe și răspânditoare a informațiilor.

Datorită naturii sale UC este strâns legată de noțiunea de context și sensibilitate la context. Principalul motiv al acestei corelări este eterogenitatea și răspândirea ubicuitară a entităților în acest tip de medii. Aceste două aspecte necesită adaptare la run-time a serviciilor și dispozitivelor, depinzând de locația și rolul lor, adaptarea la rândul ei fiind dependentă în multe situații de folosirea contextului.

Când această paradigmă este aplicată în contextul casei, o casă normală se transformă într-o casă inteligentă. Ceea ce locuitorii ei vor putea observa este doar cum casa și aplicațiile și mediul se ajustează pentru a se adapta cerințelor dinamice ale rezidenților având nevoie de o interacțiune minimală între utilizatori și aplicațiile casei inteligente.

2.2.3 Ontologie

2.2.3.1 Definiții

Termenul „ontologie” își are originile în filosofie și se referă la disciplina care se ocupă cu existența și cu lucrurile care există. În știința calculatoarelor lucrurile care „există” sunt cele care pot fi reprezentate prin date [17].

Diferite definiții pentru ontologiile legate de știința calculatoarelor pot fi găsite în literatură [23]. Ca un exemplu, o definiție dată de J. Voß [24] este: O ontologie este un sistem definit formal de concepte și relațiile dintre aceste concepte. Ontologiile conțin, cel puțin în mod implicit, reguli. Se pot deosebi trei probleme când punem această definiție pentru ontologii în relație cu modelarea contextelor și argumentarea:

- Un model de context este de asemenea un sistem de concepte (entități) și relații care fac din ontologie o unealtă bună pentru modelarea conceptelor;
- Ontologia este „definită” formal, ceea ce este o precondiție pentru ca un computer să o poată interpreta, de exemplu, pentru motive de argumentare;
- Regulile pot fi folosite pentru a implementa argumentarea contextului.

2.2.3.2 Caracteristici

M. Gruninger și J. Lee [25] despart aplicabilitățile ontologiilor în 3 grupuri mari:

- Comunicații și împărțirea cunoștințelor: o ontologie servește ca un vocabular comun pentru agenți diferiți (entități computaționale sau oameni).
- Inferență și argumentare logică: O ontologie poate fi folosită pentru a deduce cunoștințele implicite din cunoștințele explicite, aplicând reguli.
- Refolosirea cunoștințelor: Ontologiile comune pot fi folosite când se construiesc ontologii specifice domeniului.

Elementele tipice ale unei ontologii sunt:

- Conceptele și atributele lor.
- Taxonomii pentru a clasifica conceptele prin generalizare și specificare.
- Relații între concepte.
- Axiome pentru a defini formule care sunt întotdeauna adevărate. Ele sunt folosite pentru a demonstra consistența cunoștințelor modelate de o ontologie și pentru a deduce lucruri ulterioare.
- Faptele sunt instanțe ale conceptelor și relațiile dintre ele.

- Se deosebesc mai multe limbaje diferite ce pot fi folosite pentru a defini ontologii. Dintre acestea amintim Ontolingua [28], LOOM [29] și OWL – Web Ontology Language [26, 27]. OWL este limbajul cel mai folosit pentru a defini ontologii pentru sisteme senzitive la context, incluzând și serviciile Web semantice [26].

2.2.4 Agenți senzitivi la context

Construirea aplicațiilor sistemelor senzitive la context și omniprezente din domeniul locuințelor inteligente a ajuns la un punct comun prin folosirea în implementarea lor a paradigmei sistem multi-agent.

Paradigma agentului inteligent este corelată cu conceptul de sistem senzitiv la context prin faptul că fiecare agent posedă cunoștințe incomplete în sisteme fără control global, cu date descentralizate. Tehnologiile multi-agent sunt un punct comun în construirea unor sisteme care se pot adapta foarte ușor la schimbările de mediu, care sunt capabile să integreze componente eterogene, furnizând flexibilitate în controlarea unor sisteme distribuite complexe, și de scară largă.

2.2.4.1 Paradigma agentului

Cercetătorii în domeniul sistemelor agent au definit în diferite maniere noțiunea de agent, deosebindu-se două dintre aceste abordări în funcție de proprietățile care sunt dorite pentru agentul definit: Noțiunea de agent simplu și noțiunea de agent **BDI – Belief-Desire-Intention** (Convingere – Dorință – Intenție) care extinde agentul simplu adăugând proprietăți.

În descrierea agentului simplu Wooldridge [38] punctează o definiție descriind proprietățile esențiale ale agentului „Un sistem de calcul hardware sau cel mai des software care satisface următoarele proprietăți:

- autonomie: agenții operează fără intervenția directă a omului sau altfel, și au un nivel de control asupra acțiunilor și a stării interne;
- abilități sociale: agentul interacționează cu alți agenți printr-un anumit limbaj de comunicare;
- reacțiune: agenții percep mediul în care se află și reacționează la schimbările care au loc în el;
- proactivitate: agenții nu doar că iau decizii ca răspuns la schimbări, ei sunt capabili să aplice comportamente obiective preluând inițiativa”.

O definiție mai generală a agentului a fost elaborată de Russell și Norvig [39]: „Un agent este orice care poate fi să perceapă mediul prin senzori, și să acționeze în acel mediu prin efectori”. Ei specifică că „noțiunea de agent este prevăzută pentru a fi o unealtă pentru a analiza sistemele, nu o caracterizare absolută care desparte lumea în agenți și non-agenți.”

2.2.4.2 Agenți BDI

Modelul BDI așa cum este descris în [40] este bazat pe 3 concepte: Convingere, Dorință și Intenție (tradus „Belief-Desire-Intention”). Ideea are baze filosofice în conceptul analog din teoria argumentării umane. Convingerile reprezintă cunoștințele agentului despre starea curentă a mediului, starea internă a lui, informații despre alți agenți. În mediile reale, cu un grad mare de incertitudine convingerile agentului sunt incomplete și inexacte.

Convingerile agentului se deosebesc de cunoștințele lui, deoarece acestea nu trebuie neapărat să fie adevărate. La nivelul fiecărui agent din mediu se construiește o viziune parțială asupra mediului în care el operează, convingerile furnizând o abstractizare a entităților relevante.

Dorințele agentului sunt stările de fapt în care agentul vrea să ajungă sau să le folosească. Stările trebuie să fie consistente una față de cealaltă.

Intențiile reprezintă stările deliberative ale agentului, activitățile pe care agentul le-a executat pentru a îndeplini o dorință. În unele situații un agent poate alege mai multe planuri de execuție diferite. Planul selectat împreună cu încrederea de a executa acel plan devin o intenție.

Obiectivul este conceptul central al unei arhitecturi BDI, reprezentând o anumită stare finală în care agentul vrea să ajungă. *Obiectivele* sunt dorințele alese pe care agentul vrea să le îndeplinească, dacă pot să considere că din starea curentă se poate ajunge la aceste stări. Când unele dintre aceste obiective eșuează agentul poate să determine dacă starea poate fi încă atinsă pentru a reîncerca intenția sau trebuie să încerce un alt set de intenții pentru a ajunge în starea dorită. Obiectivele permit modelarea agenților care să acționeze proactiv. *Planurile* reprezintă mijloacele prin care se poate stabili un obiectiv, incluzând secvența de acțiuni care trebuie executate pentru a ajunge la acel obiectiv.

Modelul BDI s-a dovedit a fi extrem de eficient la modelarea unor agenți care rulează în medii dinamice și care operează într-o manieră flexibilă în ciuda informațiilor incomplete despre mediu și ceilalți agenți.

2.3 Arhitecturi de sisteme senzitive la context

2.3.1 Arhitecturi de sisteme senzitive la context locale

Sistemele orientate pe rețea sunt ușor de implementat din cauza standardului ISO Open System Interconnection (OSI), care este un model stratificat în care fiecare nivel este independent și are funcționalitatea sa specifică. În acest fel, în [33] se propune un model stratificat pentru arhitectura unui sistem senzitiv la context, „Context Stack”.

2.3.1.1 Modelul architectural stratificat „Context Stack”

Modelul este un model referință pentru arhitectura sistemelor pervazive. Acest model este similar cu modelul pe 7 nivele ISO-OSI pentru rețele de calculatoare. Modelul stratificat combină elementele funcționale ale sistemelor senzitive la context:

- Nivelul achiziție;
- Nivelul reprezentare;
- Nivelul agregare;
- Nivelul interpretare;
- Nivelul utilizare.

Acestea se combină într-o arhitectură coerentă și generică la care majoritatea sistemelor senzitive la context actuale se pot mapa. Figura 2.2 **Error! Reference source not found.** ilustrează cele 5 nivele ale arhitecturii Context Stack și exemplul concret al unui serviciu senzitiv la context pentru un telefon inteligent (smart phone).

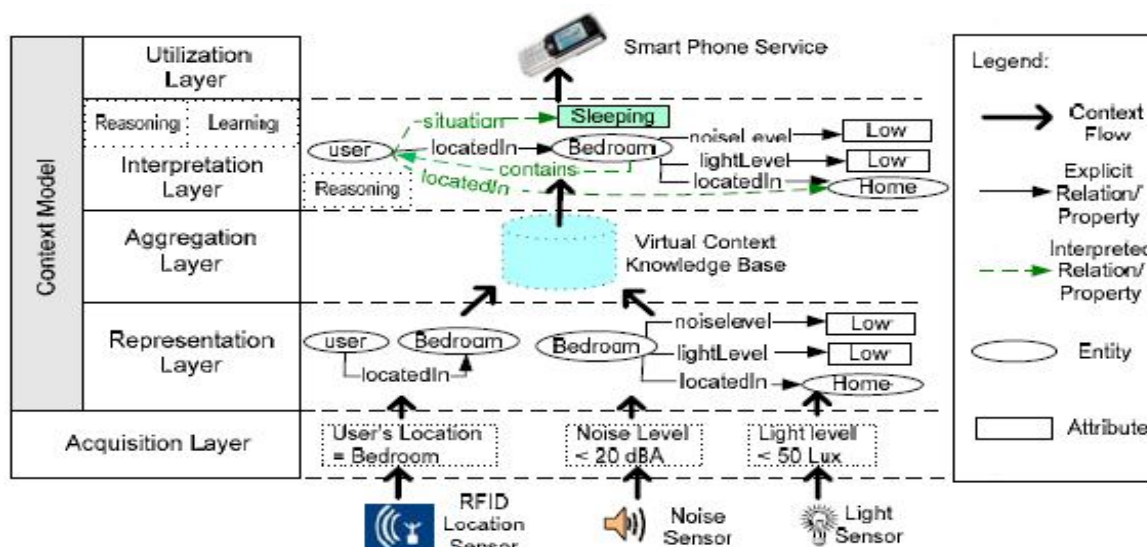


Figura 2.2 Modelul arhitectural Context Stack

Nivelul achiziție

Este nivelul cel mai de jos în care contextul este preluat într-un format neprocesat, fiind preluat de la o multitudine de tipuri de senzori pervazivi. De exemplu, locația în casă a unui locatar poate fi obținută de la un sistem de senzori RFID, care detectează apariția unei modificări în configurația RFID, nivelul luminii fiind detectat de senzorii de lumină, iar nivelul zgomotului este obținut de la senzorii de zgomot. Datele preluate pot să nu ofere context care să poată fi înțeles și utilizat de serviciile sensibile la context, astfel el trebuie trimis la nivelele de mai sus pentru procesări ulterioare.

Nivelul Reprezentare

Începând de la acest nivel, toate nivelele se bazează pe un model de context comun, care formează baza pentru managementul contextului, pentru a facilita o reprezentare expresivă și interoperabilitate între entitățile computaționale eterogene. Modelele de contexte existente variază în funcție de tipul de context pe care îl reprezintă și expresivitatea lor. Așa cum este prezentat în Figura 2.2 s-a folosit un model de context orientat pe entitate-relație, pentru a ilustra rolul modelului în sistemele sensibile la context. În asemenea modele de context, informația contextuală este structurată în jurul unui set de entități fiecare descriind un obiect fizic sau conceptual, precum o persoană sau o activitate, iar toate aceste entități sunt legate cu celelalte entități prin relații.

În nivelul de reprezentare al contextului, datele neprelucrate venite de la senzori sunt reprezentate într-o formă care poate fi înțeleasă, după modelul de context stabilit în prealabil. Acest nivel este un nivel de abstractizare a datelor, care adună datele de la senzori, venite de la diferite surse, și apoi le combină cu semantici care sunt structurate în jurul unui set de entități contextuale și relațiile dintre ele.

Nivelul Agregare

Acest nivel se ocupă de agregarea datelor contextuale, adunând datele de la senzori contextuali distribuiți într-o bază de contexte centralizată. Agregarea contextelor ajută la simplificarea procedurii de interogare a contextului, centralizând astfel entitățile computaționale și furnizează o bază pentru interpretări ulterioare ale cunoștințelor contextuale. Astfel, agregarea contextelor prin natura sa furnizează funcționalitățile unei baze de cunoștințe. O altă funcție a nivelului de agregare este de a salva o arhivă de contexte,

pentru a se putea ulterior să se facă interpretări, bazate pe contextele curente și pe cele din trecut.

Nivelul Interpretare

Când se stabilește un model de context pentru a reprezenta și prelucra contexte, contextele pot fi interpretate prin mai multe metode de argumentare și/sau învățare, pentru a deriva contexte adiționale de nivel înalt. Nivelul de interpretare a contextului adaugă tehnici de argumentare/învățare pentru a deduce contexte implicite de nivel înalt din contextele explicite provenite de la nivelele inferioare și care sunt folosite de serviciile inteligente.

De exemplu, modulul de argumentare bazată pe reguli poate deduce situația curentă a utilizatorului bazându-se pe locația sa și contextul mediului în care se află. Contextul inferat poate sugera că utilizatorul e posibil să doarmă în momentul actual, având în vedere că este ora 11 PM și că el stă în întuneric, e liniște, iar patul este ocupat. Un alt exemplu de interpretare a contextului ar putea fi învățarea și predicția comportamentului. Sistemul inteligent poate învăța șabloane ale acțiunilor întreprinse de utilizator din secvențe trecute de date contextuale, care mai târziu pot fi folosite pentru a prezice următorul eveniment. De exemplu, se poate prezice că odată ce utilizatorul a terminat dușul (închizând încălzitorul electronic de apă) după ora 10:30 PM, el își va verifica emailurile folosind telefonul, pentru ca ulterior să se așeze în pat după ce le-a citit.

Nivelul Utilizare

La nivelul cel mai de sus, serviciile senzitive la context utilizează atât contexte de nivel jos, cât și contexte de nivel înalt pentru a-și ajusta comportamentul. Luând exemplul dat în Figura 2.2, telefonul inteligent (smart phone-ul) preia contextul utilizatorului și decide să redirecționeze toate apelurile telefonice înspre căsuța vocală. De asemenea el ia în considerare contextul prezis pentru a înfăptui unele acțiuni.

2.3.2 Arhitecturi de sisteme distribuite senzitive la context

Au fost câteva încercări de a construi astfel de sisteme de roboți și monitorizare prin Internet, folosind tehnologiile World-Wide-Web și de obiecte distribuite. Sistemele de tip WWW foloseau protocolul HTTP în combinație cu CGI (Common Gateway Interface) sau Java pentru controlul la distanță a senzorilor și actuatorilor. Exemple ale unor astfel de încercări pot fi văzute mai jos:

- University of California's tele - excavation system, Mercury [01]
- Carnegie Mellon University's indoor mobile robot, Xavier [02]
- Ecole Polytechnique Fédérale de Lausanne's maze robot, KhepOnTheWeb [03]
- Roger Williams University's PumaPaint [04]
- Pohang University of Science and Technology's XNMS [05]

Există și implementări realizate cu ajutorul tehnologiilor obiectelor distribuite precum CORBA (Common Object Request Broker Architecture) și Java RMI (Remote Method Invocation) [07]:

- NRSP (network robot service platform);
- DAIR (distributed architecture for Internet robot).

Datorită limitărilor soluțiilor expuse mai sus, s-a propus o nouă metodă și s-a dezvoltat un nou framework pentru sistemele robotice pervazive, acesta fiind bazat pe semantică și numit SemanticURS [06]. Această metodă permite integrarea automată a roboților conectați la rețea în sistemele pervazive, printr-o metodă orientată pe servicii Web. Astfel SemanticURS exploatează serviciile Web semantice (Semantic Web Services), care sunt ultima apariție în domeniul tehnologiilor Web, și o metodă de planificare preluată din inteligența artificială,

pentru a automatiza procesul de interoperare între roboții conectați la rețea și sistemele pervazive.

Astfel serviciile Web, senzorii și dispozitivele din rețea, implementează o interfață unică pentru a se accesa reciproc. Ulterior cunoștințele legate de aceste servicii Web sunt descrise în limbajul ontologic Web pentru servicii (OWL-S), care este un limbaj semantic de descriere a serviciilor Web, și salvează în baza de cunoștințe (KB – **K**nowledge **B**ase) pentru ca un agent robot să poată descoperi în mod automat cunoștințele necesare pentru a compune un plan de servicii fezabil pentru mediul în care este conectat. Ulterior, agentul furnizează servicii interacționând în mod automat cu roboții, senzorii, și dispozitivele prin protocolul SOAP (Simple Object Access Protocol) [64], precum în Figura 2.3.

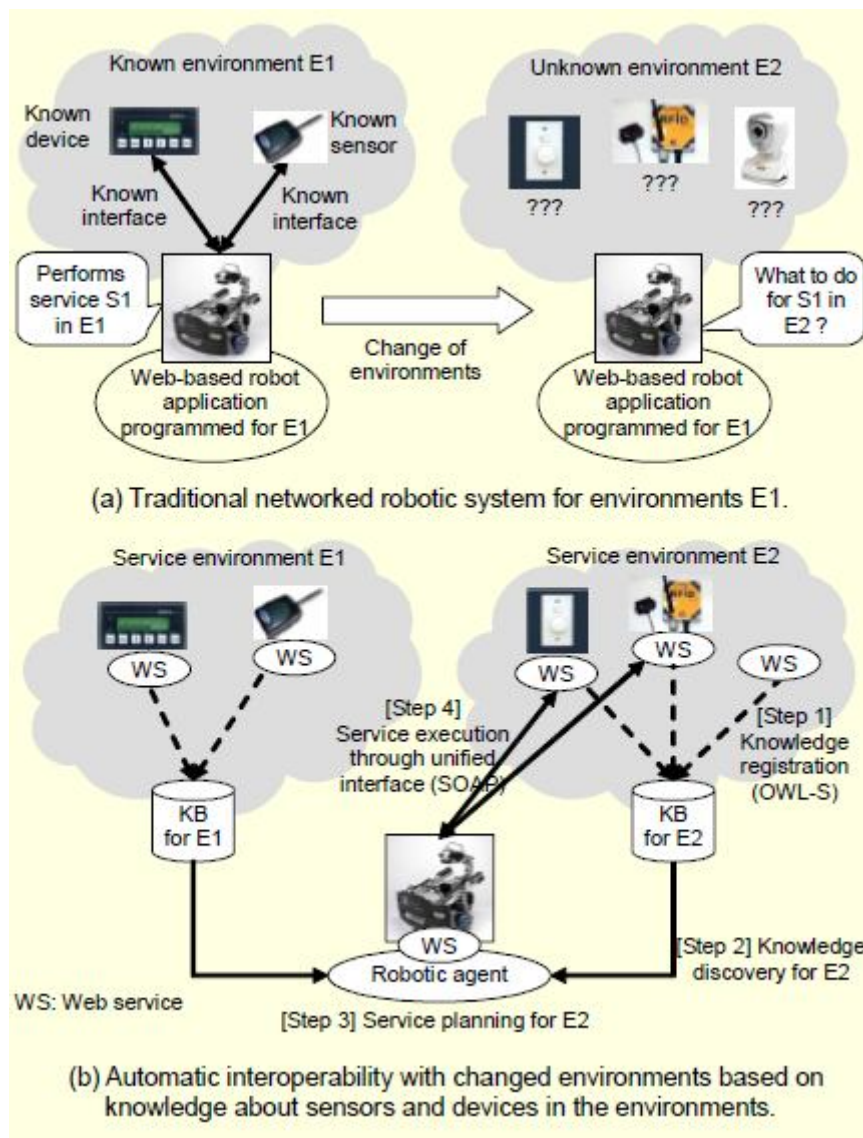


Figura 2.3 Comparație între SemanticURS și sistemele tradiționale

2.3.2.1 Serviciile Web Semantice

Rețeaua World Wide Web inițial concepută pentru ca utilizatorii să schimbe fișiere text și imagini a evoluat spre un furnizor de servicii. Programele accesibile prin Web realizează aceste servicii cu ajutorul tehnologiilor CGI, Java, ActiveX sau prin Web Services. O cerință fundamentală pentru a realiza interoperabilitatea acestor tip de servicii este necesitatea de a realiza aceste servicii interpretabile de computere, de a crea servicii Web

semantice ale căror semantică, precum proprietăți, capacități și interfețe, să fie codificate într-un mod neambiguu, într-o formă inteligibilă de către mașini. Serviciile Web Semantice sunt construite astfel încât să îndeplinească aceste cerințe, descriind serviciile Web folosind ontologii OWL, numite generic OWL-S. OWL-S este structurat în 3 clase esențiale: Service Profile, Service Model, Service Grounding [06] (Figura 2.4).

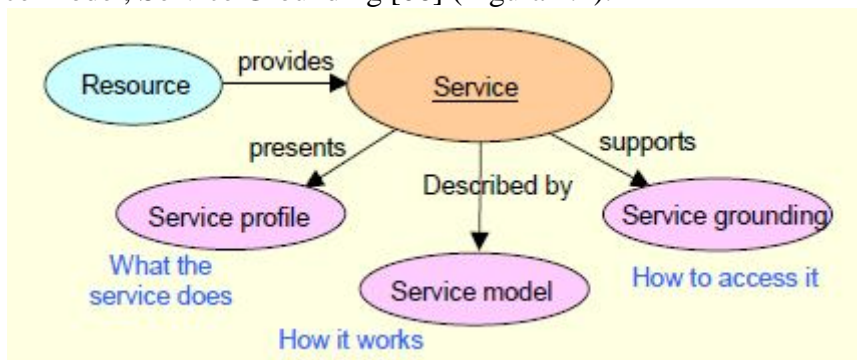


Figura 2.4 Clasele componente ale unei ontologii de tip OWL-S

Service Profile – această componentă a unei ontologii OWL-S specifică ce face serviciul. Ea are rolul de a informa agentul care cere servicii despre serviciile oferite de web service-ul pe care vrea să îl acceseze. Se folosesc următoarele noțiuni pentru a marca proprietățile unui Service Profile: profile, serviceName, textDescription, hasInput, hasOutput, has Precondition, hasEffect, serviceCategory ș.a.m.d.

Service Model – Această componentă specifică cum funcționează serviciul. Se descrie ce se întâmplă când este alocat un serviciu atomic sau compus. Această descriere poate fi folosită de un agent care necesită servicii pentru a realiza o analiză mai amănunțită a serviciului și dacă îi poate satisface nevoile. De asemenea se poate folosi pentru a compune multiple servicii pentru a atinge un anumit obiectiv. Se folosește și pentru monitorizarea execuției unui serviciu. Procesul ontologic furnizează modelului serviciului următoarele procese: ProcessModel, AtomicProcess, CompositeProcess, SimpleProcess, ProcessComponent, ControlConstruct, Sequence, Choice, Repeat-Until.

Service Grounding – Această componentă detaliază cum poate un agent accesa un serviciu. Astfel componenta specifică protocolul de comunicație, formatul mesajelor și alte detalii specifice, precum numărul portului prin care serviciul poate fi accesat. Procesul ontologic al componentei de configurare are interfețele descrise în Web Services Description Language (WSDL): WsdlGrounding, WsdlAtomicProcessGrounding, WsdlOperation, WsdlDocument, WsdlOperationRef, portType.

2.3.2.2 Arhitectura Serviciilor Web Semantice

Așa cum se poate vedea în Figura 2.5 arhitectura SemanticURS [06] este compusă din 3 componente majore: un agent robotic RA (**R**obotic **A**gent), serviciile web ale dispozitivelor DWS (**D**evice **W**eb **S**ervices) și baza de date de cunoștințe a mediului EKR (**E**nviromental **K**nowledge **R**epository).

RA – robotic agent este un agent apelator de servicii și de planificare inteligentă. El are un rol important în integrarea automată în arhitectura SemanticURS. RA este alcătuit dintr-o interfață utilizator, un modul de alcătuire a planului, un modul de descoperire a cunoștințelor, un modul de inferență OWL, un modul de executare a planului și stiva de comunicare pentru executarea serviciilor Web, de exemplu SOAP [08], XML și HTTP. Un utilizator poate introduce o cerere de executare a unui serviciu folosind interfața utilizator și eventual poate inițializa contextele serviciilor. Apoi, cererea este transformată folosind vocabularul OWL-S, în ontologiile procesului OWL-S, pentru ca modulul de alcătuire a

planului să poată înțelege cerința de servicii și să descopere în mod automat cunoștințele necesare pentru a face planificare folosind modulul de descoperire a cunoștințelor. Pentru a căuta baze de date de cunoștințe în EKR pentru a prelua cunoștințele necesare pentru cererea actuală, modulul de descoperire a cunoștințelor creează interogări semantice de descoperire codificate în **Resource Description Framework (RDF)**.

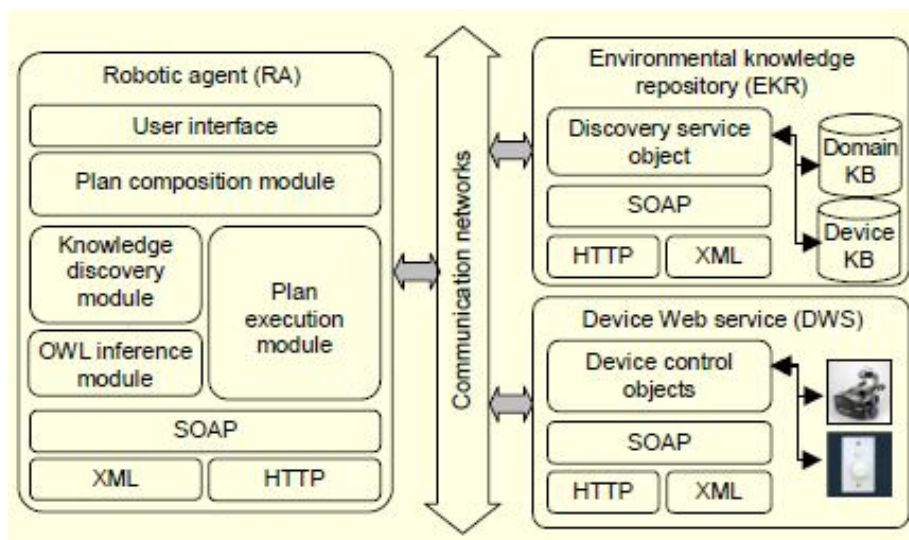


Figura 2.5 Arhitectura detaliată a Semantic URS

RDF folosește limbajul de interogare Data Query Language (RDQL) [09], [10]. După ce planul a fost compus cu succes, modulul de execuție a planului transformă planul într-un format executabil pentru ca ulterior să fie pus ca cerere pe stiva de comunicare a serviciului Web.

EKR – Enviromental Knowledge Repository conține o bază de cunoștințe a domeniilor (Domain Knowledge Base - DoKB) și o bază de cunoștințe a dispozitivelor (Device Knowledge Base - DeKB) utilizate pentru înregistrarea și descoperirea cunoștințelor. DoKB conține cunoștințe OWL-S despre conceptele generale legate de servicii și task-urile compuse care descriu modelele comune ale serviciilor pentru un anumit domeniu. DeKB conține cunoștințe salvate în format OWL-S despre task-urile atomice care reprezintă serviciile oferite de dispozitive și senzori, în medii specifice serviciilor. EKR conține de asemenea și obiectele de descoperire a serviciilor, pentru a manipula în mod automat interogările de descoperire a cunoștințelor cu predicate semantice despre cunoștințele OWL-S formate de RA. EKR conține și stiva de comunicare Web Service, componenta comportându-se ca un Web Service independent, fiind accesată de RA prin SOAP.

DWS – Device Web Service este o implementare specifică a Web Service-urilor pentru dispozitive ubicuitare incluzând roboți, senzori, actuatori ș.a.m.d. Fiecare DWS poate avea obiecte de control pentru unul sau mai multe dispozitive, care pot lucra împreună, de pildă un dispozitiv de aer condiționat și un senzor de temperatură. DWS dispune de asemenea de servicii Web pentru a comunica cu RA.

Proiectarea așa cum este descrisă în Figura 2.6 împarte procesul de integrare automată al unui agent SemanticURS în 3 faze:

- Descoperirea cunoștințelor;
- Planificarea Serviciului;
- Executarea planificării.

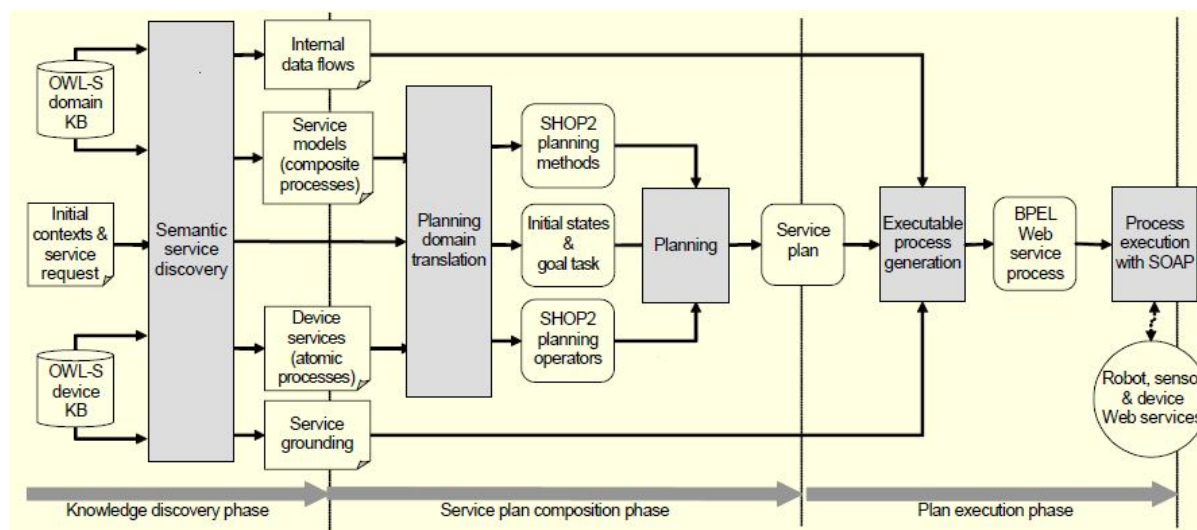


Figura 2.6 Procedura de integrare automată a unui agent SemanticURS

2.4 Medii de dezvoltare ale aplicațiilor sensitive la context

Java Context Awareness Framework (JCAF) [36] este un mediu de dezvoltare inspirat din J2EE desemnat exclusiv pentru a dezvolta aplicații sensitive la context. Scopul acestui mediu de dezvoltare este să suporte servicii distribuite și coordonate, controlul accesului și o infrastructură modulară pentru a permite extensii. Dintre caracteristicile JCAF care îl diferențiază este faptul că este independent de semantică și că pune în calcul calitatea informațiilor despre context, iar evenimentele sunt interceptate de către modulele care detectează schimbările (vezi Figura 2.7). De exemplu, se detectează schimbările de context și ulterior sunt apelate funcții prin care schimbările sunt trimise la aplicații care prelucrează informația.

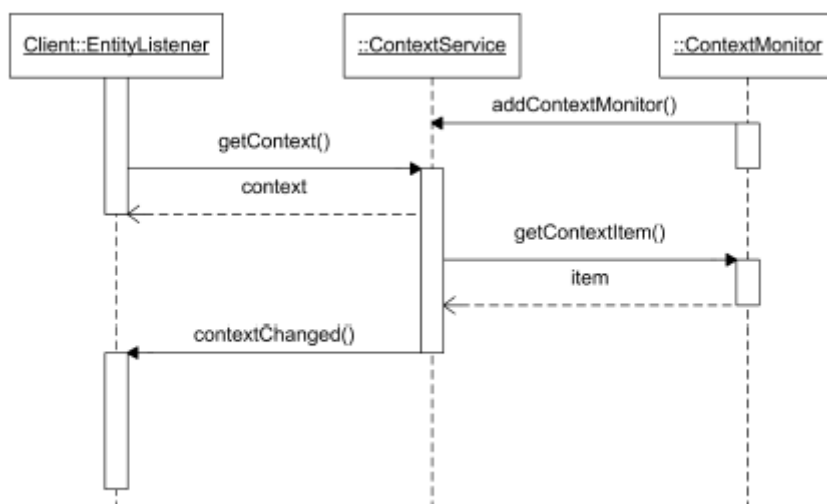


Figura 2.7 Diagrama de interacțiune între modulele componente ale JCAF

Astfel mediul JCAF nu are construit nici un modul inteligent el fiind responsabil doar de controlul traficului, schimbările de mediu și transportul informațiilor către aplicațiile sensitive la context. Nivelul cel mai important al mediului JCAF o constituie serviciile de context, care formează o rețea interconectată de tip punct-la-punct, fiecare serviciu fiind specializat pentru un anumit tip de informație de context, aceste servicii putând comunica

între ele pentru a schimba informații de context. O arhitectură generală poate fi văzută în Figura 2.8, împreună cu arhitectura unui serviciu.

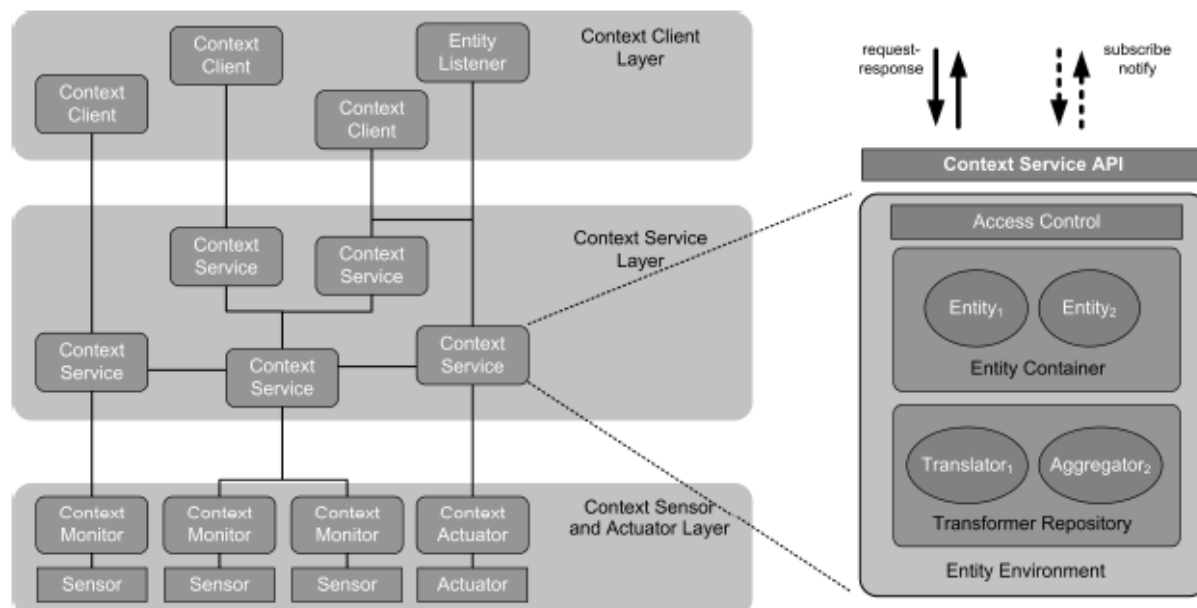


Figura 2.8 Arhitectura generală cu diagrame de colaborare a JCAF

Mai există și alte medii de dezvoltare a aplicațiilor sensibile la context, cu diferite funcționalități. Dintre acestea amintim **Maythofer's context prediction framework** [37] concentrat pe recunoașterea și deducerea unor informații consistente de context, folosind informațiile cu zgomot venite de la senzori. Scopul acestui mediu este de a fi folosit în scop local pe dispozitive cu resurse limitate, fără a fi necesară constituirea unei infrastructuri în prealabil. Se folosesc algoritmi Growing Neural Gas pentru învățare și algoritmi Active LeZi pentru deducerea contextelor.

2.5 Aplicații în domeniul caselor inteligente

În această secțiune va fi prezentată proiectarea și implementarea unei arhitecturi generale de tip service-oriented precum cele descrise anterior, aplicată în cazul unei case inteligente.

Obiectivele proiectării

Proiectarea infrastructurii serviciului adoptă modelul stratificat prezentat în Figura 2.2 – Context Stack, ca și abstracția de proiectare pentru a furniza middleware pentru servicii reutilizabil. Proiectarea respectă următoarele convenții [33]:

1) Arhitectura middleware bazată pe componente are ca rol combinarea elementelor funcționale independente ale sistemelor sensibile la context (achiziția, agregarea, interpretarea, utilizarea și descoperirea) și contopirea acestor elemente într-un middleware coerent și reutilizabil.

2) Proiectarea cercetează infrastructura Semantic Web pentru a colecta informații contextuale. Argumentarea ontologiilor Semantic Web (CONON, COBRA-ONT sau SOUPA), interogări Semantic Web și mecanisme de inferență sunt încorporate în această infrastructură pentru a facilita modelarea contextelor, împărțirea cunoștințelor, interogări semantice, argumentarea contextuală și descoperirea semantică a contextelor.

Argumentarea ontologiilor („Ontologic Reasoning”) este un pas foarte important în dezvoltarea unei aplicații senzitive la context pentru o casă inteligentă, deoarece se ocupă de evaluarea cunoștințelor de context („contextual knowledge”).

Abordările existente pentru modelare sunt așa cum au fost amintite și mai sus **CONON** [32] bazat pe ideea de dezvoltare a unui model de context din ontologii, folosind capacitățile de inferențe logice, refolosirea și împărțirea cunoștințelor. Wand a dezvoltat o ontologie superioară, care captează trăsăturile generale ale entităților contextuale și o colecție de ontologii specifice împreună cu trăsăturile lor în subdomenii. **CANON** folosește un mecanism de serializare a ontologiilor în OWL-DL, care are o echivalență semantică cu logica descriptivă foarte bine definită. Astfel se poate verifica consistența și argumentarea contextelor folosind modulele de inferență dezvoltate în limbaje descriptive. Un sistem mai complex și mai nou este **CoBrA** care furnizează o colecție de concepte ontologice pentru a caracteriza entități precum persoane, locuri sau alte tipuri de obiecte aflate în contextele lor. Sistemul **CoBrA** are o arhitectură de tip broker [32] pentru a furniza suport la momentul rulării pentru sisteme senzitive la context, în mod particular în săli de conferințe inteligente.

Folosirea standardelor existente incluzând OSGi sau UpnP pentru a simplifica programarea componentelor reutilizabile ale serviciilor are ca scop furnizarea unui standard API și implementarea unor servicii senzitive la context utilizabile pentru o casă inteligentă.

Structură OSGi de tip Service pentru o casă inteligentă

Arhitectura de tip OSGi propusă pentru case inteligente este prezentată Figura 2.9 și a fost descrisă în detaliu în [34][35]. Punctul central al arhitecturii este OSGi gateway aflat în reședință, care este responsabil pentru conectarea la Internet a aplicațiilor din casă, terminale mobile, dispozitive senzor/actuator, prin tehnologii de acces broadband. O varietate mare de tehnologii legate de rețelele din locuințe, cum ar fi Ethernet, X.10, IEEE 1394, Lonworks, WLAN sau Bluetooth, pot fi utilizate pentru a conecta diferite tipuri de dispozitive.

Framework-ul OSGi standardizează furnizarea de servicii pentru a le realiza într-o manieră sigură și fiabilă. El constituie un intermediar între diferitele standarde de rețele ce se pot afla în casă.

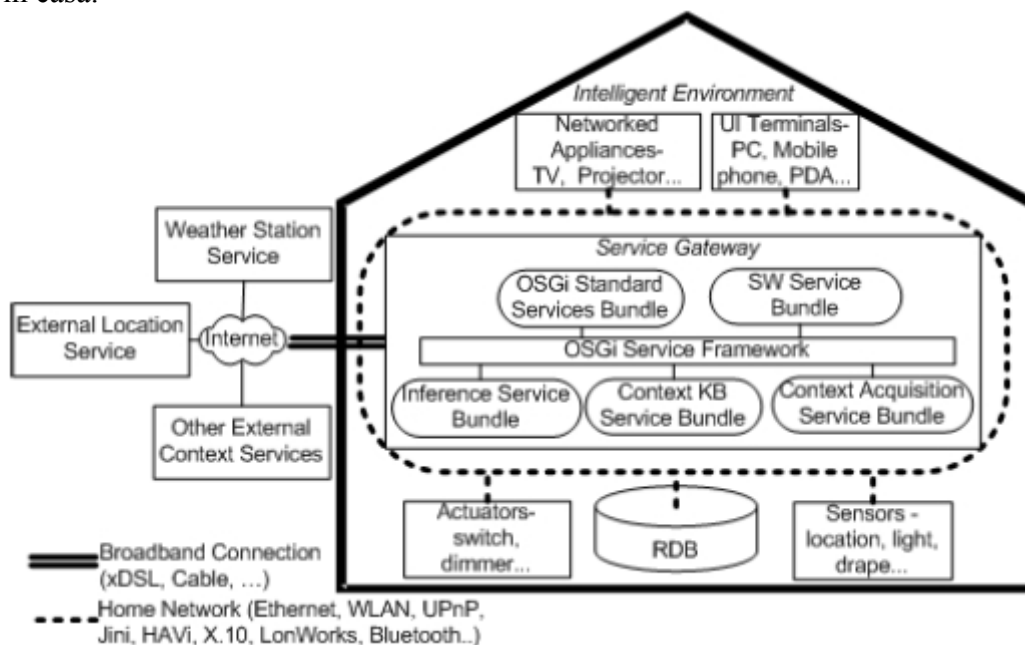


Figura 2.9 Arhitectura de tip OSGi pentru o casă senzitivă la context

Pentru a obține partajarea cunoștințelor și inferențele logice necesare informațiilor contextuale, sunt legate în arhitectura OSGi sub formă de servicii înmănunchiate (bundles) următoarele 4 module funcționale cheie:

Serviciul Web (SW) Semantic – Serviciul SW furnizează API-uri Semantic Web pentru accesarea și manipularea contextelor codificate în OWL, pentru a persista și interoga modelele semantice ale contextelor în baze de date relaționale și pentru a infera contexte adiționale din contextele existente în KB. Implementările curente sunt bazate pe Jena Semantic Web Toolkit, totuși implementări alternative pot fi expuse printr-o interfață generică pentru a fi atașate arhitecturii. SW service bundle furnizează fundamentele următoarelor 3 servicii.

Serviciul de Context Acquisition - Acest serviciu este responsabil pentru colectarea informațiilor despre context de la o gamă variată de surse și transformă descrierile ad-hoc ale contextelor în reprezentări semantice folosind ontologii. Acest serviciu furnizează funcționalitățile nivelului achiziție și nivelul reprezentare semantică al arhitecturii stratificate descrisă în Capitolul 2.3.1.

Serviciul Context KB – Baza de date de cunoștințe despre contexte furnizează un mediu de stocare pentru cunoștințele despre contexte prin folosirea bazelor de date relaționale. Aceasta stochează și manipulează atât ontologii de context statice (descrierile ontologice ale claselor sau tipurilor de entități relaționale) cât și contexte dinamice (instance ale situațiilor sau indivizi reprezentați bazați pe ontologiile de context existente). Astfel are suport pentru argumentare despre ontologiile de context, cât și pentru informațiile contextuale situaționale. Serviciul KB permite arhitecturii serviciului să agreghe și relaționeze contexte dinamice de la surse contextuale distribuite și astfel furnizează funcționalitatea nivelului agregare din arhitectura prezentată în Capitolul 2.3.1.

Serviciul de Context Inference – Acest serviciu este un modul de argumentare care poate fi folosit de serviciile senzitive la context pentru a deduce contexte de nivel înalt din contextele de bază. Serviciul de inferență este bazat pe Jena API, care furnizează inferență bazată pe reguli pentru OWL. Acest serviciu furnizează funcționalitatea nivelului inferență de context al arhitecturii prezentate în Capitolul 2.3.1.

2.6 Dificultăți și riscuri întâmpinate în crearea aplicațiilor senzitive la context

Una dintre cele mai mari dificultăți în crearea aplicațiilor ubiquitous și senzitive la context robuste este dezvoltarea unor algoritmi care pot detecta contexte din date senzoriale ambigue și cu zgomote.

Provocarea pentru astfel de sisteme constă în complexitatea capturării, reprezentării și procesării datelor de context. Pe lângă abilitatea de a obține informațiile despre context aplicațiile trebuie să includă logică adițională pentru a procesa informația și pentru a deduce scopul acestora. Aceasta este probabil cel mai provocator lucru, deoarece este de cele mai multe ori indirect și poate fi dedus prin combinarea unor diferite piese de informație contextuală [41].

În [42] se descriu dificultățile ce apar când se folosesc informațiile de context în aplicații pe calculator:

Informația de context este adeseori achiziționată de la senzori neconvenționali care nu sunt găsiți în viața de zi cu zi: termostat, GPS ș.a.m.d.

Datele preluate de la senzori trebuie să fie abstractizate într-o reprezentare de nivel înalt pentru a deveni folositoare. Luând exemplul unui termostat, temperatura citită în grade trebuie abstractizată în frig, mediu, cald, foarte cald. Dar, o aplicație poate să abstractizeze informațiile venite de la mai mulți senzori, de exemplu, pe lângă temperatură aplicația poate să aibă nevoie de informații despre umiditate pentru a determina nivelul de confort al unei camere.

Informația de context poate fi achiziționată de la surse multiple distribuite și eterogene. Aplicațiile au nevoie de mai mult de un senzor pentru a înțelege un anumit context.

Informația de context este dinamică. O aplicație senzitivă la context are nevoie de multe ori să detecteze modificările în timp real pentru a se adapta la schimbările care au loc în mod frecvent. De exemplu, o aplicație pentru o locație mobilă trebuie să detecteze modificările frecvente ale utilizatorului pentru a furniza acestuia informații relevante legate de locația curentă

Acestor dificultăți li se adaugă riscurile și provocările în procesul creării acestui fel de aplicații:

- ✓ Aplicațiile sensitive la context trebuie să înțeleagă intenția utilizatorului.

Acest proces este crucial în faza de adaptare la context. Odată ce o aplicație senzitivă la context s-a adaptat contextului curent, trebuie să decidă cum să optimizeze acea adaptare. O aplicație trebuie să presupună cât mai aproape de adevăr ce fel de comportament al acesteia este dorit de către utilizator, în contextul curent. Unele tehnici din Inteligența Artificială folosite în aplicațiile sensitive la context permit învățarea din trecut a comportamentelor repetitive sau asemănătoare, totuși, aplicația poate să înțeleagă mai greu atunci când intervine un comportament nou.

- ✓ Aplicațiile sensitive la context sunt multidisciplinare

Construirea unor astfel de sisteme sensitive la context precum casele inteligente, de obicei implică și necesită cunoștințe din știința calculatoarelor, arhitectură, inginerie electrică și mecanică, științe sociale și poate altele. Totodată construirea unui astfel de tip de aplicații necesită experți din diferite domenii, fiind nevoie de o echipă de oameni cu cunoștințe diferite.

- ✓ Costurile sunt ridicate

Unele dispozitive gen biosenzori folosite pentru a construi infrastructuri sensitive la context nu sunt foarte uzuale, astfel ele costă în general mai mult decât orice alt dispozitiv comun. Acest fapt creează o barieră pentru persoanele care doresc să dezvolte astfel de aplicații care necesită dispozitive și infrastructură specifică. Deci, de cele mai multe ori proiectarea unei astfel de aplicații este dictată de ce dispozitive sunt accesibile dezvoltatorilor.

Un fapt foarte important în construirea unui mediu cu resurse limitate este decizia de a distribui diferite componente ale sistemului între client și server. O soluție evidentă, dar sensibilă, este încărcarea majorității proceselor computaționale pe server, dar fără a se calcula atent acest fapt ar putea rezulta în supraîncărcarea serverului. O îmbunătățire semnificativă a performanței se poate realiza dacă agenții pot găsi momentul în care se poate face migrarea de date de la server la client pentru realizarea unei anumite operații.

O altă problemă importantă care trebuie luată în calcul este alegerea sistemului de operare pentru dispozitivele mobile. Deși Linux oferă o performanță mărită, WindowsCE este suportat de o gamă mai variată de dispozitive mobile.

Pe lângă aceste probleme de balansare client/server, calitate/preț mai intervin și alte cerințe minimale pentru o aplicație de acest gen: interoperabilitate, portabilitate, flexibilitate, robustețe. Toate aceste probleme sunt menționate și detaliate în [43].

Construirea sistemelor multi-agent este un proces dificil și complex deoarece integrează în mod direct sau indirect provocările și riscurile existente din sistemele distribuite și concurente și în plus dificultățile rezultate din cerințele de flexibilitate și interacțiune intensivă. Problemele construirii unor astfel de sisteme sunt detaliate în [44].

2.7 Proiecte ample de cercetare în desfășurare

Dintre proiectele ample de cercetare în legătură cu casele inteligente, și mai general cu clădiri inteligente, le amintim pe cele de mai jos.

The Adaptive House (University of Colorado)

Scopul experimentului Adaptive House este să exploreze conceptul unei case autoprogramabilă, eliberându-i pe locuitorii ei de necesitatea efectuării acestei munci. Cercetătorii au pus accent pe faptul că software-ul necesar pentru o casă automată trebuie să fie programat pentru fiecare familie și casă în parte și reactualizat odată cu modificările survenite în stilul lor de viață. Luând în considerare că la vremea aceea pentru multă lume era destul de dificil să își programeze DVD-playerele, se considera ca programarea unei case inteligente ar fi mult sub interesul și capacitățile lor, iar angajarea unui specialist care să facă aceste lucruri ar fi foarte neconvenabilă și costisitoare.

Prototipul a fost instalat în casa unuia dintre cercetători și controlează temperatura camerei, temperatura apei, nivelul luminii, a sunetului, deschiderea ferestrelor și totodată controlează aparatele responsabile pentru încălzire, lumină, curent ș.a.m.d. Sistemul monitoriza acțiunile luate de rezidenți precum stabilirea unei anumite configurații a luminii, sau pornirea unui termostat și căuta șabloane în acel mediu, pentru ca ulterior să poată prezice acele acțiuni. O rețea neuronală este cea responsabilă pentru a învăța acele șabloane pentru ca apoi sistemul să facă acele acțiuni învățate în mod automat.

ComHome (The Interactive Institute, Suedia)

Proiectul ComHome este descris de cercetători ca un model de scară largă construit dintr-un număr de scenarii preluate din camere diferite configurate. Casa este echipată cu tehnologii senzoriale, precum și tehnologii bazate pe control prin voce. În acest context, cercetătorii investighează diferite sfere ale activităților casnice, precum munca la distanță și activitățile sociale, în special comunicările mediate prin video (VMC – Video Mediated Communications), explorând impactul pe care tehnologiile îl pot avea asupra lor.

House_n (Massachusetts Institute of Technology)

House_n este un proiect colaborativ și multi-disciplinar condus de Departamentul de Arhitectură al MIT. Scopurile globale ale acestui proiect includ crearea unor medii care acoperă anumite nevoi pentru persoane din diferite categorii de vârstă, crearea unui mediu dinamic și obișnuit, dezvoltarea unor algoritmi care interpretează datele venite de la senzori pentru a detecta acțiunile îndeplinite de oameni, explorarea impactului tehnologic asupra mediilor de învățare tradiționale, precum și impactul avut de comandarea de la distanță a unor anumite acțiuni. Principalele inițiative ale grupului House_n sunt *The PlaceLab* și *Open Source Building Alliance (OSBA)*.

The PlaceLab este un ansamblu rezidențial aflat în Cambridge, Massachusetts, proiectat pentru a fi un mediu foarte flexibil și multi-disciplinar pentru cercetare și observare pentru studiul științific al omului și șabloanelor lui de interacțiune cu tehnologii noi și cu condițiile noi oferite de mediu. Casa experimentală este ocupată pentru o perioadă de timp de un anumit voluntar care nu are interacțiuni cu cercetările, ele fiind făcute din afară, pentru ca el să poată acționa cât mai natural. Cercetătorii au capacitatea de a monitoriza aproape fiecare aspect al vieții în casă, în particular ce acțiuni ia locuitorul pe durata unei zile. Unii ocupanți sunt studiați și anterior în propriile lor locuințe aceștia purtând o baterie portabilă de senzori dezvoltată de către cercetătorii MIT. Aceste dispozitive monitorizează de la distanță acțiunile

săvârșite de subiecții care participă la experiment, atât înainte de a locui în casă, precum și după aceea. Principalele probleme care se ridică în cadrul experimentului sunt:

- Ce influențează comportamentul oamenilor în propria lor casă?
- Cum se poate eficientiza aportul tehnologic în cadrul traiului în casă pe perioade mai întinse de timp?
- Poate tehnologia să modeleze schimbările de comportament din timpul vieții?
- În ce măsură, monitorizarea și măsurarea activităților din casă pot fi cuantificate în așa manieră încât se pot crea noi aplicații pe calculator utile casei?
- Cum poate tehnologia să fie folosită pentru a facilita controlul caselor din prezent și viitor, să se economisească resurse și să se îmbunătățească sănătatea?
- Ce influențează modul în care oamenii se adaptează la noi medii?
- Care este modul de învățare al oamenilor în contextul casei?
- Ce inovații aduse unei case ar putea îmbunătăți cel mai semnificativ maniera noastră de a trăi viața de zi cu zi?

OSBA – Open Source Building Alliance are scopul de a dezvolta componente cheie ale unui model reactiv cu ajutorul căruia să se poată crea locuințe în care:

- Dezvoltatorii devin integratori și membri ai alianței și oferă soluții adaptate indivizilor;
- Arhitecții concep șabloane de proiectare pentru a crea în mod eficient mii de medii inteligente unice;
- Producătorii aprobă standardele interfețelor și devin furnizori de componente de primul nivel;
- Constructorii devin instalatori și asamblori;
- Clienții sunt cei din centrul procesului primind informații personalizate legate de proiectare, produse și servicii în funcție de deciziile lor.

Dintre implementările realizate de către alianța OSBA amintim:

- OSBA: Integrated Interior Infill (I3);
- Sisteme OSBA care oferă soluții gen „work-at-home”, sisteme acustice integrate, sisteme interactive, elemente transformabile, aplicații și dispozitive conectate în rețea ș.a.m.d.

OPEN Prototype House Initiative

A fost format pentru a crea niște prototipuri de locuințe care testează un nou model pentru design-ul și fabricarea unor locuințe cu reactivitate mare. Această alianță, condusă de MIT House_n Open Source Building Alliance și Bensonwood Homes, va permite partenerilor industriali să colaboreze pentru prototipizarea și crearea unor noi materiale, sisteme, dispozitive utile pentru casele inteligente.

Just-In-Time Persuasive User Interfaces for Motivating Healthy Behaviors

Acest program investighează tehnologiile și proiectarea interfețelor utilizator pentru a crea dispozitive și spații persuasive. Programul dezvoltă o tehnologie care detectează automat un anumit punct de decizie (point-of decision) folosind senzori de mediu. Interfețele utilizator folosesc aceste informații „just-in-time” despre ce acțiune efectuează un anumit individ, și idei preluate din psihologia socială și alte științe sociale, pentru a motiva o schimbare a comportamentului în perioade lungi de timp aplicând o metodă neiritantă, educațională.

Astfel, domeniile în care se aplică această idee sunt conservarea energiei, regim alimentar sănătos, activitate fizică, siguranța personală și la locul de muncă și învățarea. Informațiile pot fi expuse cel mai eficient pe un dispozitiv pervasiv, precum un telefon mobil.

3 FUNDAMENTARE TEORETICĂ

3.1 Modelarea formală a contextelor

3.1.1 Agenții bazați pe cunoștințe

Aplicația curentă are la bază folosirea agenților inteligenți. Precum oamenii cunosc, învață și raționează, cunoștințele și raționamentele sunt aplicate și în cazul agenților bazați pe cunoștințe. Astfel ei își pot însuși comportamente foarte greu de atins într-o altă manieră. De exemplu [39], un agent-reflex poate să găsească o rută validă între 2 orașe doar prin noroc, sau un agent bazat pe rezolvarea problemelor poate rezolva probleme foarte complexe, dar cunoștințele sale sunt limitate și inflexibile. Cunoștințele unui agent pot fi exprimate în diferite forme generale, combinând și recombinați informații pentru diferite scopuri, nu de fiecare dată dintre cele mai folosite, la fel cum un matematician poate dezvolta o nouă teoremă, care nu neapărat trebuie să satisfacă anumite nevoi din acel moment.

Agenții bazați pe cunoștințe joacă un rol crucial în cazul mediilor observabile parțial precum casele inteligente. Aceștia pot combina cunoștințele generale cu informațiile curente pentru a infera aspecte nevizibile ale stării curente, înainte de a decide o anumită acțiune, la fel cum un doctor diagnostichează un pacient pentru a infera boala cu care se confruntă pacientul, care nu este direct observabilă, înainte de a îi prescrie un anumit tratament [39].

Agenții bazați pe cunoștințe sunt foarte flexibili. Aceștia sunt capabili să accepte noi întrebări sub forma unor obiective descrise și pot deveni competenți pentru a le îndeplini doar spunându-li-se sau învățând noi cunoștințe despre mediu, putându-se adapta schimbărilor mediului reactualizând cunoștințele relevante despre acesta.

3.1.1.1 Baze de date de cunoștințe

Componenta centrală și esențială a unui agent bazat pe cunoștințe este baza de cunoștințe sau prescurtat KB (**K**nowledge **B**ase). O bază de cunoștințe este la modul cel mai simplist, un set de propoziții. Propozițiile se folosesc în acest cadru ca și termen tehnic, acestea fiind exprimate într-un limbaj numit limbaj de reprezentare a cunoștințelor, care poate fi un limbaj ontologic, și reprezintă niște adevăruri formale pe care vrem ca agentul să le știe.

Principalele funcționalități ale unei baze de cunoștințe sunt:

- Adaugă propoziții noi în baza de cunoștințe;
- Interoghează cunoștințele existente.

Aceste 2 funcționalități sunt numite în [39] TELL, respectiv ASK. În Figura 3.1 poate fi vizualizată structura generală a programării unui agent bazat pe cunoștințe. Ambele funcționalități pot implica **inferența**, care este derivarea unor noi propoziții din cele existente. Ca un orice alt agent, agenții bazați pe cunoștințe preiau la intrare o informație percepută (în cazul unei case inteligente o informație primită de la senzorul de temperatură, de exemplu) și returnează o acțiune. Agentul folosește baza de cunoștințe pentru a efectua orice operațiune. De fiecare dată când programul prezentat în Figura 3.1 este apelat, se efectuează 3 lucruri. Prima dată, se adaugă informațiile percepute în baza de cunoștințe. Ulterior, este interogată baza de date pentru a cere ca răspuns o acțiune pe care ar trebui să o îndeplinească, în acest proces putând fi implicate mai multe acțiuni de argumentare în legătură cu starea curentă a mediului, despre diferite scenarii posibile ale concretizării acțiunii returnate ș.a.m.d., acest proces decizional fiind cel mai pretențios dintre pașii concreți ai programului. În final, agentul execută acțiunea și informează KB dacă acțiunea a fost concretizată și care au fost consecințele.

```

function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
           t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(^))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action

```

Figura 3.1 Cazul general de programare a unui agent bazat pe cunoștințe

Programul inițial al agentului, înainte de a primi informații de la mediu, este construit adăugând una câte una propozițiile care reprezintă cunoștințele celui care a făcut proiectarea despre mediul în care va acționa agentul. Limbajul folosit pentru a exprima aceste propoziții este un **limbaj declarativ**. O abordare mai veche sugera folosirea unui **limbaj procedural** pentru a exprima aceste propoziții direct prin cod de program. În cazul aplicației de față vom folosi un limbaj declarativ pentru a reprezenta cunoștințelor. El va fi detaliat în secțiunile următoare.

În plus, pe lângă aceste acțiuni din program prin care i se spune unui agent ce să facă, se pot adăuga acestui agent mecanisme care îi permit să învețe de la sine. Aceste cunoștințe pot fi încorporate în baza proprie de cunoștințe putând fi folosite pentru luarea deciziilor, astfel putând construi un agent autonom.

3.1.2 Reprezentarea cunoștințelor

În secțiunea anterioară am prezentat agenții bazați pe cunoștințe. Aceste cunoștințe trebuie reprezentate într-un limbaj declarativ. Aici vom detalia conceptul de ontologie, cu ajutorul căreia se poate organiza mediul și entitățile componente într-o ierarhie generalizată și structurată de clase.

Domeniile complexe, precum controlarea unui robot într-un mediu schimbător, precum o casă inteligentă, necesită o reprezentare generalizată și flexibilă a informațiilor. Conceptele generale ce trebuie reprezentate sunt *Acțiune*, *Timp*, *Obiect Fizic*, *Presupunere*. Reprezentarea acestor concepte este domeniul **ingineriei ontologice**.

Este dăunător să încercăm să reprezentăm o descriere completă a tuturor lucrurilor, existente, dar e binevenit să reprezentăm conceptele de bază, care ulterior să fie detaliate, în funcție de necesități. Colecția generală de concepte reprezentate, se numește **ontologie superioară**, datorită convenției de a reprezenta conceptele generale în partea de sus, în timp ce conceptele specifice completează partea de jos a modelului, ca în Figura 3.2.

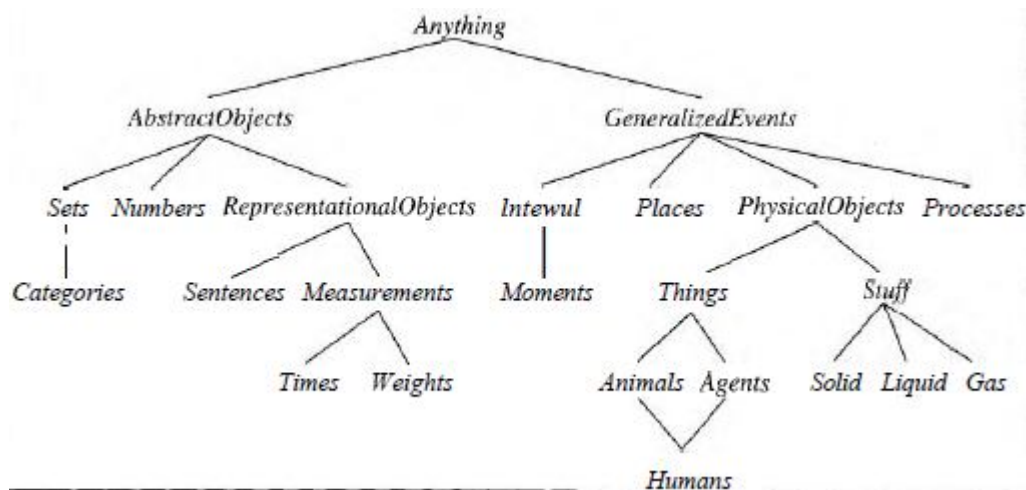


Figura 3.2 Exemplu de modelare a unei ontologii superioare

Pentru orice ontologie specializată este posibil să se facă schimbări pentru a putea conecta această ontologie într-una mai generalizată. Această proprietate conferă reprezentării ontologice a cunoștințelor o flexibilitate foarte mare, fiind exact ceea ce este nevoie pentru modelarea bazei de cunoștințe a unui agent bazat pe cunoștințe care acționează într-o casă inteligentă. O problemă care se pune este dacă toate ontologiile ar putea converge la o **ontologie generalizată** [25]. Acest lucru este un subiect de controversă. Totuși, există niște caracteristici ale ontologiilor de uz general care le face să fie distincte față de ontologiile specializate:

- O ontologie generalizată trebuie să poată fi aplicată în orice domeniu specializat;
- Într-un domeniu suficient de complex cunoștințele diferite trebuie unificate deoarece argumentarea și rezolvarea problemelor poate să implice mai multe domenii simultan.

3.1.3 Folosirea ontologiilor în modelarea contextelor

3.1.3.1 Modelarea contextului cu ontologii

În trecut, integrarea sensibilității la context în sistemele ubiquitous s-a dovedit a fi o muncă complexă și foarte costisitoare datorită lipsei de infrastructura necesară, cum ar fi componente hardware, sisteme de operare, precum și inexistența unui model de contexte comune care să permită reutilizarea [45].

În [21] sunt expuse cerințele de infrastructură și de reutilizare pentru a crea servicii sensibile la context pentru o casă inteligentă:

- Un model comun de reprezentare a contextelor, care poate fi împărtășit de toate dispozitivele și serviciile. Această componentă e necesară deoarece fiecare componentă ca parte din sistemul casei inteligente trebuie să aibă aceeași interpretare a datelor schimbate. Folosirea unui model comun dintr-un singur domeniu sau între mai multe domenii eficientizează schimbul de informații între componente.
- Un set de servicii care execută achiziție de context, descoperire de context și interpretare de context.

Reprezentarea folosind ontologii se dovedește a fi cea mai bună abordare în rezolvarea problemelor de independență a componentelor, împreună cu limbajul de exprimare, sistemul de operare și componentele middleware. Avantajele modelării contextelor folosind ontologii aplicate și în proiectul curent sunt [46]:

- Permit analiza formală a cunoștințelor prin argumentarea contextelor;

- Prin argumentare, concepte de nivel înalt pot fi derivate din concepte de nivel jos;
- Ontologiile pot fi verificate pentru consistență și pot fi validate în funcție de celelalte ontologii existente;
- Pot fi introduse relații inter-ontologice și ierarhii implicite bazate pe reguli.

3.1.3.2 Proiectarea ontologiilor de context

Deși există multe propuneri de proiectare a ontologiilor de context, se pune problema ce model este cel mai potrivit și care aduce cele mai multe avantaje când se creează un sistem senzitiv la context nou. Astfel se ridică următoarele întrebări:

- Cât de bine poate o ontologie să reprezinte concepte, attribute și relații într-o formă exactă și bine definită?
- Cât de capabil este limbajul pentru a crea interogări eficiente pentru argumentare?
- Care este cea mai eficientă proiectare în arhitectura ontologică?
- Cum pot fi extinse aceste ontologii pentru a suporta incertitudinea?
- Care este câștigul de performanță folosind alegerea de proiectare?

Propunerile existente de proiectare, unele fiind în curs de standardizare, susțin un model ierarhic și stratificat.

Contextele pot fi organizate în două categorii majore, context direct și indirect. Contextul direct, achiziționat direct de la un furnizor. Mai departe contextul direct poate fi clasificat în context senzorial și context predefinit. Contextul senzorial este achiziționat de la senzorii casei inteligente sau de la senzorii virtuali, cum ar fi serviciile. Contextul definit este cel mai des predefinit de către utilizator în mod particular. Folosind un mecanism de argumentare a contextelor, se pot infera contexte indirecte, bazate pe cele directe. De exemplu, statutul curent al unei persoane care este la duș într-o casă inteligentă poate fi inferat din locația sa, starea încălzitorului de apă și starea ușii. În Figura 3.3 se pot observa ontologii specifice pentru o casă inteligentă.

Diferitele tipuri de contexte au caracteristici temporale diferite, de exemplu contextul senzorial este dinamic odată cu trecerea timpului, în timp ce contextul predefinit este static. De asemenea, pot interveni inconsistențe sau conflicte între diferitele tipuri de contexte, datorită erorilor de tehnologie sau întârziere în procesare. Doar clasificând contextele variate și știind caracteristicile se poate efectua argumentare de contexte pentru a elimina conflictele și a menține consistența [13].

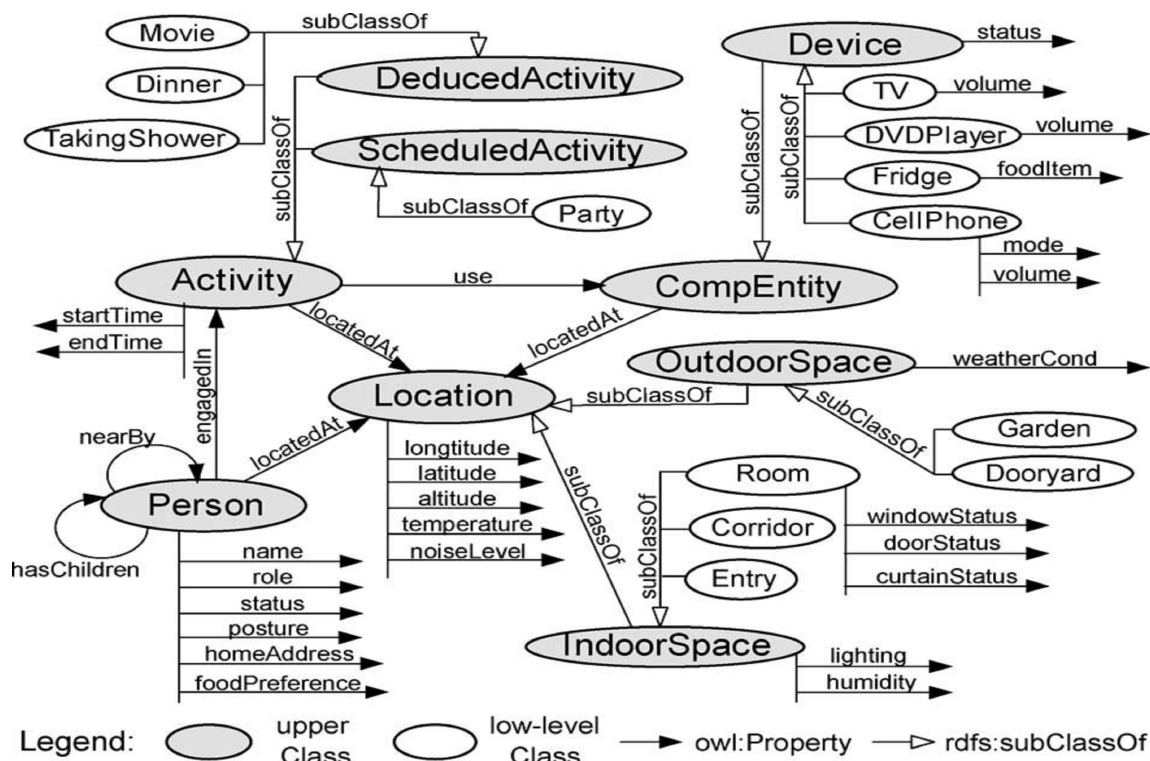


Figura 3.3 Definire ontologii specifice reprezentate prin rețele Bayesiene

Unul dintre cele mai cunoscute modele este **SOCAM** (Service Oriented Context Aware Middleware) detaliat în [21]. Specificațiile acestuia presupun o arhitectură naturală pe două nivele ca Figura 3.4:

- Nivelul superior - pentru ontologii superioare comune, în care sunt încadrate conceptele generale;
- Nivelul inferior - pentru ontologii specifice, care sunt aplicate în câteva domenii inteligente.

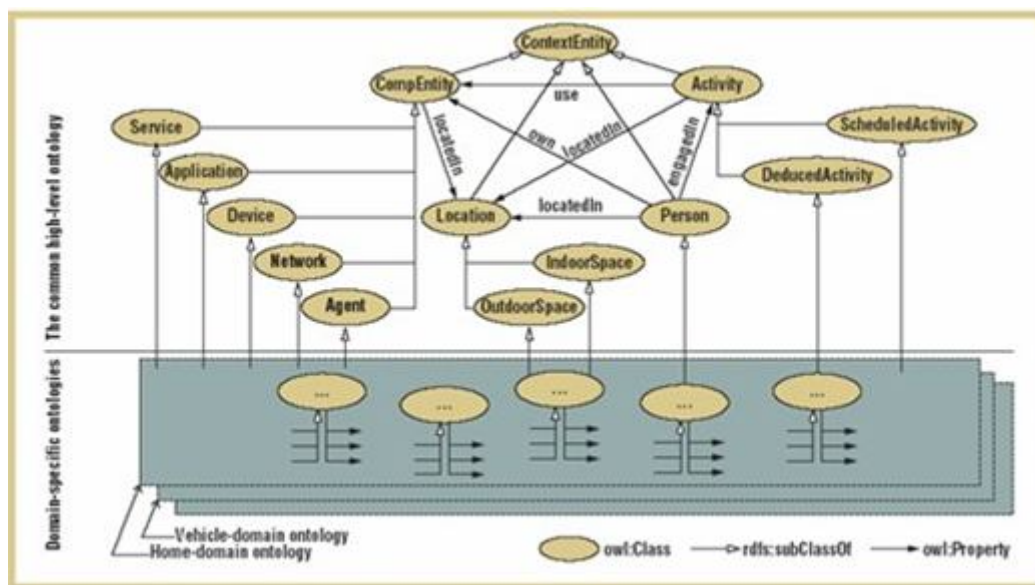


Figura 3.4 Model de proiectare a ontologiilor contextuale, folosind SOCAM

Ontologiile sunt scrise în OWL (Ontology Web Description Language), limbajul pentru ontologii Web propus de W3C. Ontologiile definite în OWL pot fi referite prin serverele Web de către alte ontologii și descărcate de către aplicații care folosesc ontologiile. Faptul că aceste ontologii pot fi accesate prin diferite servere Web, furnizează flexibilitatea și extensibilitatea necesară unui sistem distribuit.

Avantajele folosirii unei astfel de arhitecturi stratificate este faptul că numărul cunoștințelor de context este redus, iar procesarea contextului de către mecanismul de argumentare este limitată, prin reîncărcarea dinamică a ontologiilor de nivel inferior atunci când apar modificări de mediu. Ontologiile inferioare specifice se pot atașa ontologiei generale pentru a crea prin deducție concepte de nivel înalt.

O altă abordare specifică în elaborarea ontologiilor de context este **CoOL**[46]. Precum SOCAM acesta se bazează tot pe o arhitectură pe două nivele:

- Nivelul inferior – CoOL Core. Acest nivel se bazează pe limbajele OWL, DAML + OIL de descriere a ontologiilor, de altfel, oferind extensii pentru folosirea și a altor limbaje descriptive;
- Nivelul superior – CoOL Integration o colecție de extensii de scheme și protocoale.

Autorii argumentează că flexibilitatea alegerii limbajului de reprezentare a ontologiilor favorizează dezvoltatorii de programe care au posibilitatea de a alege pe cel mai potrivit planurilor lor. Mecanismul de argumentare este construit în limbajul F-Logic, în principiu pentru că F-Logic este un limbaj orientat obiect, expresiv, având posibilitatea de extensie la argumentare bazată pe reguli.

Modelul contextual propus este bazat pe modelul ASC (Aspect Scalabilitate Context). Modelul ASC este construit în jurul acestor trei concepte de bază, entitățile fiind instanțe ale informațiilor contextuale. Un aspect agregă una sau mai multe modele scalare și încapsulează obiecte având aceleași semantici. Informația contextuală încapsulează datele propriu-zise împreună cu meta-datele care caracterizează datele propriu zise.

Tabel 3.1 Comparație între SOCAM și CoOL privind modelarea ontologiilor de context

	SOCAM	CoOL
Limbaaj de descriere	OWL	OWL sau alte limbaje (DAML, F-Logic ș.a.m.d.)
Specificații Model	Proiectat pe 2 nivele Modelează contexte și relații cu tipuri de date și obiecte Extensii probabilistice ale predicatelor	Proiectat pe 2 nivele Utilizează modelul ASC Modelul ASC poate fi folosit ca un model de transfer între diferite tipuri de modele de context
Mecanism de argumentare	Argumentare ontologică Argumentare cu reguli	Mecanism de inferență Onto Broker F-Logic ca limbaj de interogare
Concluzii	Fiabilitatea ontologiilor – se folosesc numai ontologiile inferioare în procesul de argumentare. Model foarte bine acceptat în dezvoltarea sistemelor ubiquitous	ASC este un model flexibil ce poate folosi ca vocabular de bază pentru alte modele. ASC se poate folosi cu valori eterogene care descriu aceeași entitate

3.1.3.3 Limbajul OWL

a) Introducere

Limbajul OWL se află între specificațiile *World Wide Web Consortium (W3C)* și este folosit ca o componentă fundamentală a inițiativei Semantic Web [30]. Limbajul se bazează pe Extensible Markup Language (XML), XML Schema[31], Resource Description Framework (RDF) și pe RDF Schema (RDF-S) [09]. Este divizat în 3 sub-limbaje diferite prin expresivitate: OWL-Lite, OWL-DL și OWL-Full. OWL-DL este folosit cel mai frecvent, pentru că este cel mai expresiv, fără a pierde completitudinea și decidabilitatea, spre deosebire de OWL-Full.

Au fost dezvoltate mai multe unelte pentru a putea lucra cu OWL:

- Protégé, care este un editor grafic;
- Jena, care este un Java API pentru a accesa ontologiile OWL;
- Racer, un modul de inferență care poate fi integrat cu Jena și Protégé.

Mai multe detalii despre aceste unelte vor fi furnizate în secțiunile ulterioare.

b) Elementele limbajului OWL

Cele mai importante elemente ale limbajului OWL vor fi descrise aici. O specificare detaliată poate fi găsită în [26]. Cuvântul cheie *Class* este folosit pentru a defini concepte ontologice. Cuvântul cheie *subClassOf* definește concepte specializate, deci poate fi folosit pentru a construi taxonomii. Cuvântul cheie *DataTypeProperty* este folosit pentru a defini atribute și concepte. Scopul variabilei se specifică prin schemă XML. Cuvântul cheie *ObjectProperty* definește relații între concepte. Acestea pot fi marcate ca și tranzitive, simetrice sau funcționale. Două relații pot fi marcate ca una fiind inversa celeilalte. Relațiile pot fi specializate folosind *subPropertyOf* în analogie cu *subClassOf* pentru concepte.

Atât relațiile cât și atributele pot fi modificate prin restricții de cardinalitate și noi valori. De asemenea este posibil să se definească fapte despre conceptele, atributele și relațiile definite anterior prin instanțierea lor. Aceste instanțe se numesc indivizi. Mai mult, este posibil să definim concepte specializate, dependente de valorile relațiilor și atributelor.

OWL furnizează de asemenea mijloace pentru a clasifica ontologiile și pentru a marca conceptele, atributele, relațiile și indivizii în analogie cu elementele altor ontologii. Acest lucru furnizează interoperabilitatea sistemelor care folosesc ontologii diferite.

c) Argumentare folosind OWL

Potrivit semanticii diferitelor elemente ale limbajului se pot deduce cunoștințe în mod explicit folosind faptele deja definite. Conform proprietăților *subClassOf*, *subPropertyOf*, *intersectionOf*, *unionOf* și *disjointWith* se poate calcula ce indivizi aparțin unui concept și vice-versa, ce concepte se mapează unui obiect dat. O listă mai completă de reguli poate fi găsită în [32].

Mai mult, valorile unor atribute și relații pentru un individ pot fi calculate în cazul în care acestea nu sunt explicit precizate, folosind *subPropertyOf*, *inverseOf* și proprietățile de tranzitivitate și simetrie.

Există mai multe metode cu care se face modelarea și argumentarea contextelor. Dintre acestea amintim SOCAM, CoOL, CONON, COBRA-ONT și SOUPA. SOCAM și CoOL fac parte din scopul acestei teme fiind detaliate în capitolul precedent. Toate aceste metode, folosesc semantica ontologică pentru a face argumentare logică. Totuși, beneficiul real al folosirii ontologiilor pentru informații despre context în sistemele pervazive, nu va deveni efectiv înainte ca să se propună un standard general valabil pentru ontologia contextelor.

Tabel 3.2 O parte din regulile de argumentare OWL

Transitive-Property	$(?P \text{ rdf:type owl:TransitiveProperty}) \wedge (?A ?P ?B) \wedge (?B ?P ?C) \Rightarrow (?A ?P ?C)$
subClassOf	$(?a \text{ rdfs:subClassOf } ?b) \wedge (?b \text{ rdfs:subClassOf } ?c) \Rightarrow (?a \text{ rdfs:subClassOf } ?c)$
subProperty-Of	$(?a \text{ rdfs:subPropertyOf } ?b) \wedge (?b \text{ rdfs:subPropertyOf } ?c) \Rightarrow (?a \text{ rdfs:subPropertyOf } ?c)$
disjointWith	$(?C \text{ owl:disjointWith } ?D) \wedge (?X \text{ rdf:type } ?C) \wedge (?Y \text{ rdf:type } ?D) \Rightarrow (?X \text{ owl:differentFrom } ?Y)$
inverseOf	$(?P \text{ owl:inverseOf } ?Q) \wedge (?X ?P ?Y) \Rightarrow (?Y ?Q ?X)$

3.1.4 Extensii probabilistice ale ontologiilor

În domeniul caselor inteligente incertitudinea este un aspect inerent. De exemplu, cel mai evident caz de incertitudine este incorectitudinea și imperfecțiunea datelor preluate de la senzori. Agenții, în aproape nici un caz nu au acces în întregime la toate informațiile din mediu. De asemenea, acești agenți raționali trebuie să poată crea raționamente în incertitudine într-o manieră eficientă. Toate aceste aspecte privind incertitudinea în modelarea contextelor se traduc în interpretare eronată a datelor de context de nivel inferior și implicit a datelor derivate de nivel superior.

Încă de la apariția sa, modelul probabilistic sau modelul rețelelor Bayesiene [39] s-a dovedit a fi o tehnică bine întemeiată pentru reprezentarea și raționarea în incertitudine. În particular o rețea Bayesiană reprezintă o distribuție uniformă a unui set de variabile aleatoare. Cu ajutorul acestui model se poate răspunde la interogări complexe despre oricare dintre aceste variabile. Astfel rețelele Bayesiene furnizează diferite tipuri de raționamente:

- Predicție – Raționare de la cauză la efect;
- Abducție – Inferarea unei cauze din efect;
- Explicare – Evidențele privind cauza unui eveniment reduc posibilitatea consumării unui alt eveniment datorită rezultatelor lor comune. Acest lucru este dificil de modelat în sistemele cu raționamente bazate pe reguli.

Totuși o limitare fundamentală a rețelelor Bayesiene în cazul reprezentării cunoștințelor este faptul că acestea nu pot reprezenta informația structurală și relațională [49]. Deci o rețea Bayesiană nu este potrivită pentru a reprezenta date contextuale corelate și dinamice în mediile computaționale omniprezente.

În [48] se prezintă un model unificat care moștenește avantajele folosirii atât a modelului probabilistic, cât și al ontologiilor, aceste două tehnici fiind integrate. Cu ajutorul acestui model se va putea construi o ontologie care va reprezenta atât structural cât și probabilistic cunoștințele contextuale, iar raționamentele în incertitudine vor fi construite tot cu ajutorul rețelelor Bayesiene.

O mare parte a cercetării în domeniul caselor inteligente s-a adresat problemei de incertitudine. Printre primele eforturi a fost încercarea de a modela incertitudinea informațiilor contextuale folosind atribute precum imperfecțiunea, acuratețea ș.a.m.d. Astfel acest model unificat este compus din două părți esențiale:

- Schema relațională – Reprezintă structura și organizarea informațiilor sub formă de clase, relații și proprietăți.
- Model probabilistic – care adnotează dependențele probabilistice condiționale dintre proprietățile claselor.

Un exemplu de reprezentare a acestor două scheme poate fi văzut în Figura 3.5:

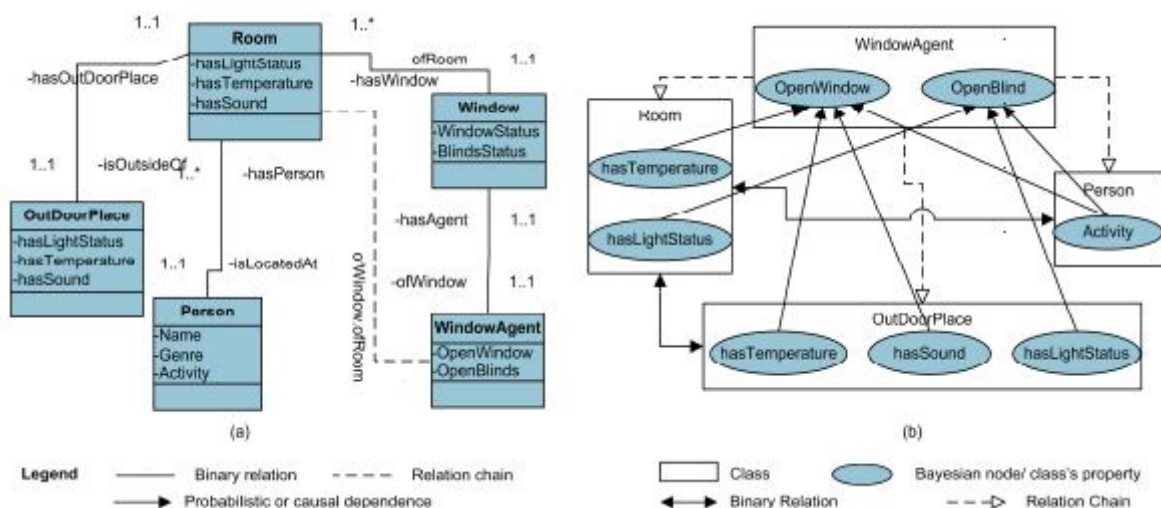


Figura 3.5 (a) Schema relațională și (b) Modelul probabilistic pentru o casă inteligentă

Bazat pe modelul propus mai sus putem construi o ontologie de context pentru a capta cunoștințele preluate dintr-o casă inteligentă. Precum se specifică și în cazul arhitecturii SOCAM, și aici se pot modela ontologii de context folosind acel model stratificat, cu ontologiile generale pe nivelul superior, iar cele specifice pe nivelul inferior.

Acest model are două componente:

- Ontologie generică – este ontologia de nivel superior la fel ca și în cazul SOCAM;
- Ontologie specifică – o colecție de ontologii de nivel inferior care definesc detalii despre conceptele și proprietățile din fiecare sub-domeniu. O ontologie specifică pentru un sub-domeniu este compusă din două părți:
 - Schema relațională – care specifică relațiile și cascaderile între relații pentru sub-domeniu;
 - Modele probabilistice – care reprezintă dependențele probabilistice condiționale dintre proprietățile acelui sub-domeniu.

În Figura 3.6 ontologia generică definește conceptele de „CompEntity”, „Location”, „Person”, „Activity”. Detaliile fiecărui concept generic precum relațiile sau dependențele probabilistice sunt redefinite în ontologii specifice domeniului (pe un nivel inferior), care pot varia de la un domeniu la altul.

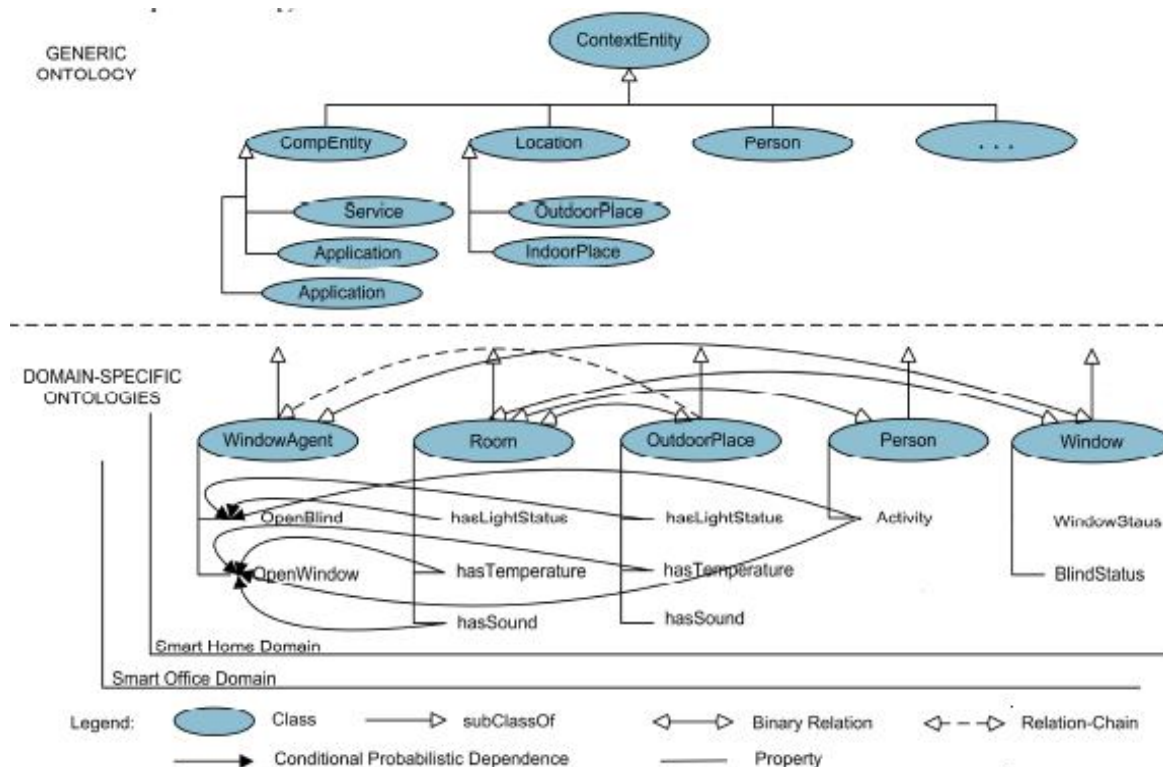


Figura 3.6 Arhitectura SOCAM cu ontologii specifice și modele probabilistice

Contextul ontologic se reprezintă folosind limbajul OWL, căruia i-au fost adăugate câteva elemente pentru a modela conceptele noi, privind relațiile între entitățile probabilistice și modelele probabilistice. Limbajul este numit de cercetători **PROWL (Probabilistic Annotated OWL)**, tradus OWL cu adnotări pentru probabilități).

Folosind această structură de modelare probabilistică a contextelor, se pot face 3 tipuri de raționamente:

- Raționamente din schema de bază a modelului:
 - Raționament bazat pe reguli;
 - Raționament ontologic.
- Raționamente inovative adăugate de modelul extins:
 - Raționament Bayesian.

Raționamentele bazate pe reguli au aceeași logică precum în schema clasică, dar cu mențiunea că în acest model sunt aplicate claselor probabilistice și nu unor clase normale. *Raționamentele ontologice* sunt asemenea cu cele din schema clasică, doar ca ele sunt reprezentate acum în PROWL. Totuși se pot face raționamente tip OWL fără probabilități, folosind OWL, deci nu se va ajunge la o constrângere de limbaj în cazul în care nu se vrea folosirea probabilităților pentru astfel de raționamente.

Raționamentele Bayesiene se fac pe baza rețelelor Bayesiene construite din modelul contextului. În funcție de domeniu se pot construi una sau mai multe rețele derivate care corespund fiecărui model probabilistic. Aceste rețele se construiesc folosind algoritmul Construct-BN [50].

Avantajele acestui model sunt introducerea probabilităților în modelarea inițială a contextelor. Deoarece ontologiile reprezintă conceptele dintr-un domeniu, acest model de context poate îngloba într-o manieră facilă. Folosind această abordare se poate modela un domeniu al bazei de cunoștințe comună pentru casele inteligente fără a fi necesară construcția unei baze de cunoștințe noi de fiecare dată. Deci această modelare suportă scalabilitatea.

3.2 Metode folosite pentru modelarea bazelor de cunoștințe

3.2.1 Metodologii existente pentru crearea bazelor de cunoștințe

Ingenieria proiectării cunoștințelor se ocupă cu structurarea, dezvoltarea și standardizarea bazelor de cunoștințe. Așa cum a fost precizat și în capitolele anterioare ontologiile sunt un model comun și cel mai folosit pentru reprezentarea cunoștințelor și crearea bazelor de cunoștințe.

În [51] se descriu multiple metodologii create de-a lungul timpului pentru a defini baze de cunoștințe folosind ontologii. Metoda folosită de către *Cyc KB* [52] este compusă din 3 faze. Prima fază constă din codificarea manuală a cunoștințelor existente, în care cunoștințele comune sunt preluate manual. A doua și a treia fază se referă la achiziționarea noilor cunoștințe utilizând limbajul natural, respectiv unelte de învățare folosite de dispozitivele de calcul. Diferența dintre aceste două faze este faptul că cunoștințele noi sunt achiziționate de dezvoltatori folosind unelte în faza a doua, respectiv automat de către mașini în faza a treia.

În metoda *KACTUS* [53] ontologiile se construiesc pe baza unei baze de cunoștințe a unei aplicații existente, la care se aplică un proces de abstractizare. Astfel, cu cât mai multe aplicații vom construi, cu atât mai generală va deveni baza de cunoștințe. Cu alte cuvinte, metoda presupune crearea unei baze de cunoștințe pentru o aplicație, pentru ca mai târziu dacă o bază nouă de cunoștințe va fi necesară pentru un domeniu similar, se va generaliza baza de date existentă într-o ontologie și se va adapta ambelor aplicații. Aceasta este o metodă „bottom-up” (de jos în sus).

Metoda *Sensus* [54] este o abordare „top-down” (de sus în jos) pentru a deriva ontologii specifice din ontologii mari. Autorii propun identificarea unui set de termeni cheie care sunt relevanți pentru un anumit domeniu.

Methontology [55] este o metodologie cu ajutorul căreia se pot crea baze de cunoștințe fie de la început, fie printr-un proces de reutilizare. Acest mediu de dezvoltare include mai mulți pași în construirea lor așa cum se poate vedea în Figura 3.7: identificarea procesului de dezvoltare ontologic, un ciclu de viață bazat pe prototipuri evolutive și tehnici particulare pentru executarea fiecărei activități.

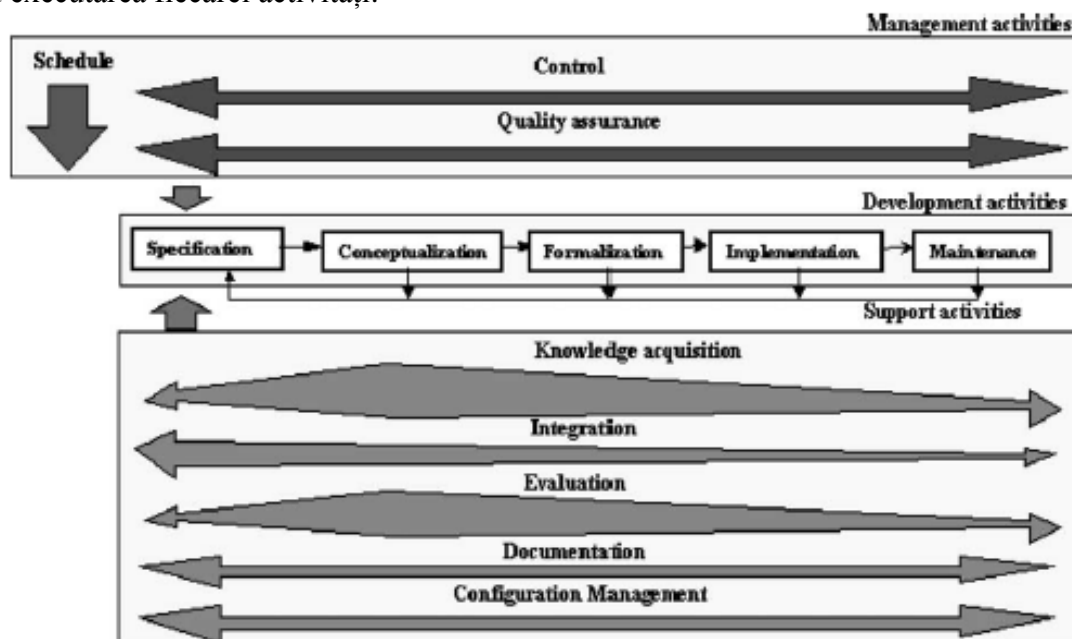


Figura 3.7 Procesul de dezvoltare a unei baze de cunoștințe folosind Methontology

Pe lângă aceste metodologii enumerate mai sus, mai există și altele, niciuna dintre aceste abordări neatingând încă stadiul de maturitate, comparând, de exemplu cu ingineria software. Totuși, se poate considera Methontology cea mai matură abordare în construirea bazelor de cunoștințe, recomandată și de FIPA. Toate aceste metodologii nu sunt unificate, deoarece fiecare grup de cercetători aplică abordarea proprie în procesul creațional. Colaborarea între aceste grupuri pentru a unifica abordările lor ar fi cea mai rezonabilă metodă pentru a ajunge la un proces unificat și matur de creare a ontologiilor specifice aplicate în cazul bazelor de cunoștințe.

3.2.2 Unelte de dezvoltare a ontologiilor

În ultimii ani numărul de unelte pentru construirea de ontologii a crescut exponențial. Aceste unelte au scopul de a oferi suport pentru dezvoltarea de ontologii și pentru folosirea acestora. Astfel vor fi prezentate cele mai importante unelte pentru ca mai apoi să fie descrisă mai amănunțit unealta folosită pentru crearea ontologiilor pentru casa inteligentă dezvoltată în lucrarea de față și anume Protégé.

Ontoligua Server [56] a fost prima unealtă creată pentru dezvoltarea ontologiilor. A fost dezvoltat la Universitatea din Stanford și a apărut în 1990, pentru a facilita dezvoltarea ontologiilor de tip Ontoligua folosite pentru niște aplicații de tip Web. Inițial, Ontoligua Server se comporta ca un editor pentru ontologii pentru ca mai apoi să fie adăugate noi module precum Webster, care era un rezolvitor de ecuații sau OKBC (**O**pen **K**nowledge **B**ase **C**onectivity) care era un modul de conectare la baza de cunoștințe, mai târziu standardizat ca și protocol de comunicație. Editorul funcționa de asemenea ca un translator inter-limbaje, precum LOOM, Prolog, CORBA IDL, CLIPS ș.a.m.d.

Tot din acea generație de unelte a apărut *Ontosaurus* [57] dezvoltat la Institutul de Științe Informaționale de la Universitatea South-California. Acesta avea în componența sa două module: un server de ontologii, care folosea LOOM pentru reprezentarea cunoștințelor și un client Web care avea și suport de traducere din LOOM în Ontoligua, KIF, KRSS și C++.

Aceste unelte din prima generație au dezavantajul de a fi create ca fiind neextensibile, dependente de limbaj, fiind create pentru a fi folosite doar la nivel de proiect de cercetare, nu pentru uz comun.

A doua generație de unelte de dezvoltare de ontologii a apărut recent. Ele au fost construite ca fiind medii robuste integrate în sisteme care aduc suport tehnologic pe toată durata de viață a activității unei ontologii. Acestea au fost concepute prin arhitecturi extensibile, bazate pe componente, unde noi module se pot integra ușor pentru a adăuga întrebări noi. Pe lângă toate acestea, modelele de reprezentare a cunoștințelor sunt independente de limbaj. *Protégé 2000* [65] dezvoltat la Universitatea din Stanford, este un program gratuit, independent de platformă, cu o arhitectură extensibilă. Partea esențială a acestuia o constituie editorul de ontologii, dar se pot importa module pentru a edita ontologii în diferite limbaje (F-Logic, Jess, OIL, XML, Prolog), dispune de acces OKBC, creare și execuție de constrângeri (PAL), fuzionare de ontologii (PROMPT) ș.a.m.d. *WebODE* [66] este succesorul lui ODE și a fost dezvoltat la Universitatea Tehnică din Madrid. Este o unealtă asemănătoare cu Protégé și implementează metodologia Methontology prezentată în secțiunea anterioară. Ontologiile sunt păstrate în baze de date relaționale și de asemenea suportă introducerea de limbaje ontologice diferite (XML, RDF, OIL, DAML+OIL, F-Logic, Jess).

În ultima generație au apărut extensii ale editoarelor enumerate mai sus pentru a edita ontologii și a crea baze de cunoștințe specifice serviciilor Web semantice.

3.2.2.1 Protégé 4

Mediul de dezvoltare a ontologiilor Protégé, ajuns la a 4-a versiune, este un mediu complet și stabil pentru dezvoltarea bazelor de cunoștințe și a ontologiilor oferind facilitatea de a integra o varietate mare de module adiționale, care pot fi dezvoltate și de către utilizatorii de rând, codul sursă al aplicației fiind pus la dispoziție.

Protégé dispune de 2 editoare separate fiecare având un anumit scop. Editorul *Protégé-Frames* facilitează dezvoltarea bazelor de cunoștințe compatibile cu standardul OKBC. Acest editor furnizează interfețe utilizator, care pot fi modificate pentru a modela cunoștințele după bunul plac al utilizatorului, precum și module pentru vizualizare, management, inferență sau raționare. De asemenea este pus la dispoziție și un API Java care poate fi folosit pentru a crea alte aplicații pentru accesarea cunoștințelor modelate cu Protégé Frames.

Editorul Protégé-OWL face posibilă dezvoltarea de ontologii folosind standardul W3C OWL. Acesta oferă uneltele necesare pentru a folosi ontologiile OWL și RDF, suport pentru vizualizarea claselor, expresii OWL și alte facilități. Platforma Protégé oferă flexibilitatea de a integra module adiționale. Aceste module sunt de trei tipuri: module de stocare, module grafice și panouri.

Un modul de stocare este un modul care salvează sau încarcă modele dintr-un format fișier sau baze de date (XML, XML Schema, RDF ș.a.m.d.).

Un modul grafic se ocupă de partea grafică a aplicației, fiind o componentă cu ajutorul căreia se pot vizualiza imagini, sunete sau chiar imagini video.

Panourile pot fi vizualizate în fereastra principală Protégé. Există mai multe subtipuri de panouri în Protégé:

- Panouri de vizualizare:
 - OntoWiz – vizualizare grafică configurabilă sub formă de graf pentru ontologii și relațiile între ele. Se formează o diagramă similară cu UML.
 - TGWiz – vizualizare grafică a metodelor claselor.
 - Jambalaya – o unealtă de răsfoire a ontologiilor care permite editarea acestora.
- Panouri pentru managementul fișierelor sau a proiectului:
 - BeanGenerator – generează clase Java („Java Beans”) dintr-un model de clase ontologice specificate în Protégé.
 - Prompt – permite manipularea a diferite modele care pot fi unificate, comparate, sau compuse.
 - DataGenie – întărește interacțiunea dintre Protégé și baze de date furnizând un modul JDBC de conectare la baze de date.
- Panouri pentru motoare de raționare:
 - QueryTab – pentru interogarea bazei de cunoștințe.
 - PAL – mecanism de editare și evaluare a axiomelor PAL (Protégé Axiom Language – Limbajul Axiomelor Protégé).

Conceptul de ontologie are în Protégé o formă asemănătoare cu cea a claselor din proiectarea orientată-obiect. Clasele Protégé sunt similare cu clasele Java și pot fi așezate într-o structură ierarhică, diferența fiind că aceste clase nu conțin definiții de metode. Acestea pot fi abstracte sau concrete, doar cele concrete putând fi instanțiate. Protégé suportă de asemenea moștenirea multiplă. Atributele sunt definite ca perechi nume-valoare: valori primitive (logic, întreg, flotant, șir de caractere); numele se reprezintă printr-un șir de caractere. Aceste atribute pot fi de asemenea referințe la alte clase sau instanțe de clase. Cu ajutorul atributelor se construiesc relațiile între clase.

3.3 Metode folosite pentru dezvoltarea agenților

Deși numărul de medii de dezvoltare a agenților a crescut mult în anii curenți, medii care furnizează unelte vizuale și interfețe pentru a ușura proiectarea agenților și realizarea a diverse sisteme de agenți, eforturile și investițiile în acest sens sunt considerabil mai mici, decât în dezvoltarea uneltelor menite altor ramuri ale industriei software.

Totuși, metodologiile existente elaborate de diferite laboratoare de cercetare în domeniul sistemelor multi-agent sugerează împărțirea procesului de dezvoltare a agenților în faze similare cu dezvoltarea unui alt produs software. Astfel procesul este împărțit în 5 faze: cerințele primare, cerințele secundare, proiectarea arhitecturii, proiectarea detaliată, implementarea.

Cerințele primare identifică actorii și obiectivele reprezentate de două modele diferite. Se construiesc diagrame care depictează implicările și relațiile dintre actorii modelului, numite dependențe. Aceste dependențe depictează modul în care actorii depind unii de alții pentru a îndeplini obiectivele sau pentru a executa planurile. Diagramele obiectivelor înfățișează analiza obiectivelor și planurilor referitoare la un actor responsabil să le îndeplinească.

Toate aceste modele sunt extinse în faza de cerințe secundare, care modelează sistemul încadrat în mediul în care va acționa. În această fază obiectivele sunt descompuse în sub-obiective.

În faza de proiectare detaliată se construiesc specificațiile obiectivelor, capacităților și a raționamentelor agenților. De asemenea, comunicarea între agenți este specificată în detaliu. În această fază se proiectează sistemul strict dependent de platforma de dezvoltare folosită, fiind adaptată la trăsăturile programării adaptate a agenților.

Faza de implementare are ca obiectiv generarea codului de construire a agenților din specificațiile oferite folosind proiectarea de detaliu.

3.3.1 Platforme de dezvoltare a agenților

Datorită existenței unui număr mare de platforme și unelte de dezvoltare a agenților, fiecare oferind diferite facilități, este binevenită o evaluare a acestora înainte de a alege una care să satisfacă necesitățile sistemului dezvoltat. În principiu, criteriile de evaluare a diferitelor medii de dezvoltare sunt necesitate, popularitate, întreținere, documentație.

Platformele pentru agenți furnizează un mediu în care se pot crea agenți executabili și facilitează și servește la coordonarea agenților prin platformă, de exemplu, timpul de execuție al unui agent, facilitând totodată comunicarea cu alte module necesare în ansamblul sistemului, unele platforme implicând și elemente de securitate.

Fundația pentru agenți fizici inteligenți FIPA [53] (The Foundation for Intelligent Physical Agents) este o organizație care promovează și standardizează tehnologiile folosite în construirea agenților. FIPA a completat un proces de standardizare a platformelor de dezvoltare a agenților. Acest proces include 5 cerințe pe care o platformă trebuie să le îndeplinească: comunicarea între agenți, transportul între agenți, managementul agenților, arhitectura abstractă a platformei, respectiv aplicații care extind platforma.

Arhitectura abstractă a unei platforme precum și comunicarea între mai multe platforme precizată în standardele FIPA poate fi văzută în Figura 3.8..

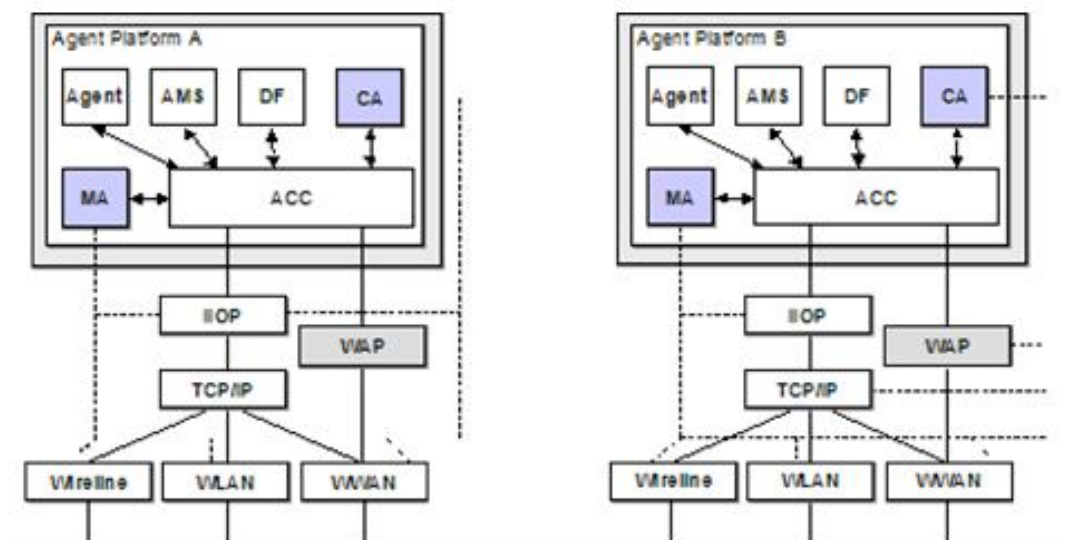


Figura 3.8 Arhitectura generală a unei platforme de agenți și comunicarea între platforme

În [56] sunt analizate diferite platforme după următoarele trăsături:

- Standardizare – standardele comune pentru dezvoltarea agenților sunt FIPA și (OMG) MASIF;
- Comunicare – suport pentru comunicare prin mesaje între platforme;
- Licențiere – ce tip de licențiere necesită;
- Accesibilitate;
- Mobilitate – abilitatea de a migra un agent în stare de execuție;
- Securitate – securitatea intră și inter-platforme;
- Documentare – la nivel utilizator sau dezvoltator .

FIPA-OS [58] este o platformă bazată pe componente, care facilitează dezvoltarea rapidă a agenților conformi FIPA. FIPA-OS suportă majoritatea specificațiilor experimentale FIPA, fiind îmbunătățit în mod regulat ca proiect open-source.

- Standardizare – FIPA;
- Întreținere – Ultima versiune 2.2.0;
- Comunicare – Tehnologiile de comunicație puse la dispoziție sunt ACL, IIOP, RMI, XML;
- Licențiere - licență publică EPL (Emorphia Public License);
- Accesibilitate – orice sistem de operare, necesită minim Java 4;
- Mobilitate – suportă agenți mobili, doar ca prototip;
- Securitate – RMI este criptat cu SSL pentru comunicare securizată inter-platforme;
- Documentare – la nivel utilizator sau dezvoltator.

Jack [60] este un mediu de dezvoltare orientat agent integrat în totalitate cu mediul de dezvoltare Java. JACK furnizează o extensie orientată agent limbajului de programare Java, având un mediu de dezvoltare și furnizând un set de clase utile dezvoltării de aplicații. Codul sursă JACK este compilat mai întâi în cod sursă Java înainte de a fi compilat mai departe în cod-mașină pentru a fi executat.

- Standardizare – FIPA;
- Întreținere – Ultima versiune 3.5;
- Comunicare – Necesită o rețea DCI pentru comunicare, similară cu TCP/IP;
- Licențiere – licență publică;
- Accesibilitate – orice sistem de operare;
- Mobilitate – nu suportă mobilitatea agentilor;

- Securitate – profită de securitatea oferită de JDK;
- Documentare – detaliată.

Zeus [61] are ca scop dezvoltarea rapidă a aplicațiilor multi-agent furnizând un set de unelte și componente care sunt folosite pentru a proiecta, dezvolta și organiza sistemele de agenți. Mai mult, acesta furnizează un mediu care permite efectuarea de statistici și rapoarte, vizualizarea agenților, o interfață bine realizată, permite repararea agenților, furnizează librării de strategii predefinite și multe alte facilități. În schimb, documentația disponibilă este foarte redusă, ceea ce duce la multe dificultăți în crearea aplicațiilor de către începători.

- Standardizare – FIPA;
- Întreținere – Ultima versiune 3.5;
- Comunicare – KQML și ACL;
- Licențiere – licență publică;
- Accesibilitate – orice sistem de operare;
- Mobilitate – criptare prin cheie publică sau privată;
- Securitate – codare MIME;
- Documentare – la nivel utilizator sau dezvoltator.

Platforma cea mai folosită pentru dezvoltarea agenților este **JADE** [59]. Aceasta fiind platforma folosită în proiectul curent, vom detalia specificațiile ei în secțiunea următoare.

3.3.2 Platforma JADE

JADE (**J**ava **A**gent **D**evelopment **F**ramework) a fost dezvoltat pentru a fi compatibil cu specificațiile FIPA pentru sisteme multi-agent inteligente și interoperabile. Scopul acestuia este de a simplifica dezvoltarea, asigurând în același timp cerințele standardului printr-un set de servicii sistem pentru agenți. Acest produs poate fi considerat un middleware care implementează o platformă pentru agenți și un mediu de dezvoltare pentru agenți. Sunt luate în considerare toate aspectele care nu au de a face cu cerințele de construire ale unui agent, și care sunt independente de aplicație, precum transportul de mesaje, codarea mesajelor, sau ciclul de viață al unui agent.

JADE creează depozite multiple pentru agenți, fiecare dintre acestea poate fi pe aceeași mașină sau pe mașini diferite. Împreună, toate aceste containere formează *platforma JADE* precum în Figura 3.9. Fiecare platformă trebuie să aibă un depozit principal, care deține doi agenți speciali:

- **AMS** (**A**gent **M**anagement **S**ystem) – Sistemul de management al agenților este agentul care deține autoritatea în platformă, fiind agentul care creează sau distruge alți agenți, distruge depozite de agenți sau oprește platforma.
- **DF** (**D**irectory **F**acilitator) – Agenda de servicii a agentului care funcționează ca un serviciu după paradigma „pagini aurii”, care publică serviciile agenților din platformă, astfel încât ceilalți agenți care au nevoie de aceste servicii, să le poată găsi.

Platforma JADE dispune de niște servicii care furnizează tehnologii de creare a agenților specifice aplicațiilor multi-agent distribuite:

- Controlul ciclului de viață al agenților;
- Mobilitatea agenților;
- Transport punct-la-punct de mesaje;
- Securitate, toleranță la erori, suport pentru agenți replicați;
- Planificarea mai multor acțiuni simultane;
- Unelte grafice care suportă monitorizarea, controlul și depanarea aplicațiilor: Agenți de monitorizare la distanță, Agenți introspectori ș.a.m.d.

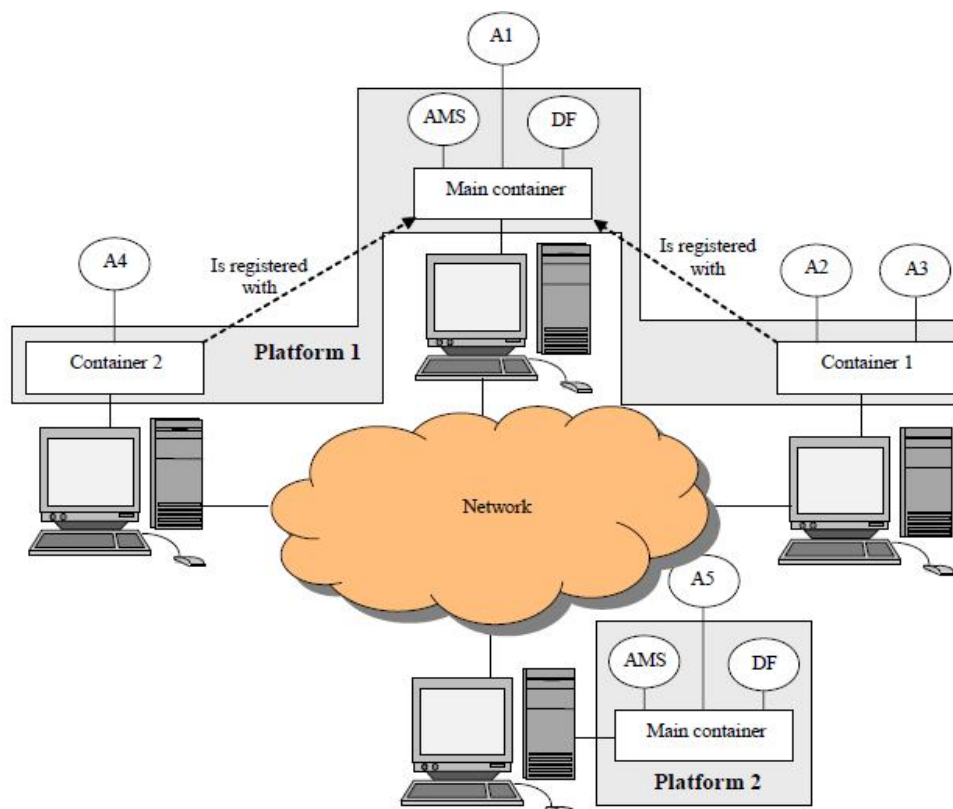


Figura 3.9 Platforma, depozitele și comunicațiile între platforme JADE

JADE furnizează de asemenea o integrare completă cu ontologiile create folosind Protégé. În terminologia JADE ontologiile împreună cu reprezentarea lor se numesc scheme. Aceste scheme se referă la clase Java proiectate pentru a reprezenta structura statică a ontologiilor. Elementele specifice ale unei scheme sunt obiecte care descriu structura conceptelor, acțiunilor și predicatelor permise în mesaje. De asemenea, pentru crearea acestor scheme JADE furnizează un mecanism de transformare a ontologiilor direct din limbajul descriptiv OWL în clase JAVA. Această unealtă face ca integrarea dintre ontologiile descrise în Protégé și JADE să fie simplă.

3.3.3 Mediul JADEX 0.96

Jadex (JADE eXtension) este o aplicație JADE, care este folosită pentru a crea raționamente de tip **BDI (Belief-Desire-Intention – Convingere-Dorință-Intenție)** pentru agenți BDI, implementați folosind platforma JADE [62]. Ultima versiune apărută în 2007 este 0.96 care este o versiune stabilă. Recent, Jadex a ajuns la versiunea 2.0 care este încă în fază beta.

3.3.3.1 Modelul BDI definit în Jadex

Jadex facilitează folosirea modelul BDI pentru agenți introducând presupuneri, obiective și planul ca obiecte foarte importante care pot fi create și manipulate în interiorul unui agent. În Jadex agenții au presupuneri care sunt modelate sub forma obiectelor Java și memorate într-o bază de cunoștințe. Obiectivele reprezintă stările de atins care influențează comportamentul agenților. Pentru a atinge obiectivele, un agent execută planuri care sunt transpuse prin metode procedurale în Java. Arhitectura abstractă prin care se pot vizualiza și componentele descrise mai sus poate fi văzută în Figura 3.10 [63].

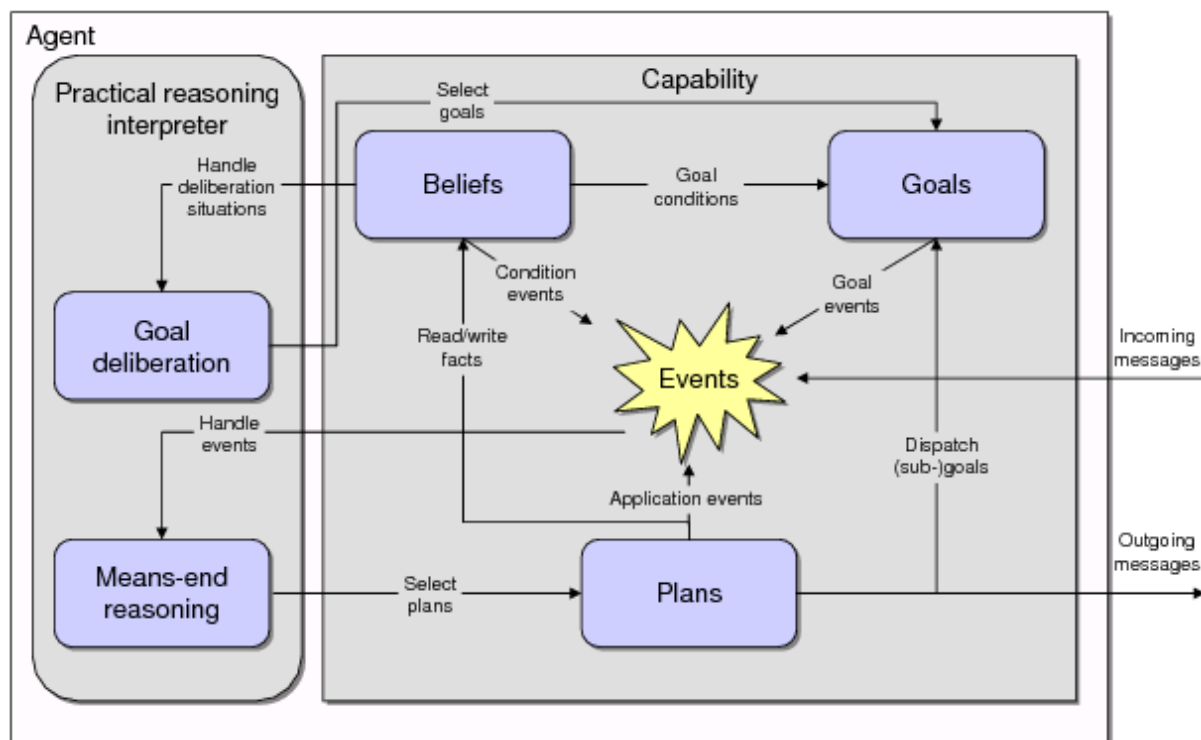


Figura 3.10 Diagrama de colaborare între componentele unui agent Jadex

Așa cum se poate observa raționamentele agenților Jadex se construiesc folosind cele două componente inter-conectate. Pe de o parte agentul reacționează la mesajele venite, evenimente interne și obiective selectând și executând planuri, pe cealaltă parte interpretorul de raționamente procesează obiectivele curente pentru a vedea ce sublan consistent ar trebui urmat.

Cele 3 concepte de bază ale unui agent Jadex sunt presupunerile, obiectivele și planurile, acestea 3 fiind definite de către dezvoltator pentru a defini comportamentul agentului BDI. Realizarea acestor 3 concepte este descrisă pe scurt în continuare.

Baza de presupuneri memorează faptele care se presupun a fi adevărate și este un punct de acces pentru datele conținute de agent. Astfel, această componentă abstractizează cunoștințele și reprezintă o vedere unificată a cunoștințelor agentului. Reprezentarea presupunerilor în Jadex este foarte simplificată și nu suportă niciun mecanism de inferență. Baza de presupuneri conține șiruri de caractere care reprezintă un identificator pentru o presupunere specifică (similar cu numele tabeli dintr-o bază de date relațională). Acești identificatori sunt mapați valorilor presupunerilor, numite de această dată fapte, care pot fi convertite în obiecte Java. Jadex suportă în acest moment două tipuri de astfel de presupuneri: presupuneri singulare și seturi de presupuneri. Deci baza de presupuneri fiind tare tipizată, poate verifica în momentul rulării dacă obiectele care respectă tipurile prezentate mai sus sunt memorate.

Obiectivele sunt tratate în mediul Jadex ca fiind dorințele concrete ale unui agent la un moment dat în timp. Pentru fiecare obiectiv pe care îl are, un agent va înfăptui acțiuni, până în momentul în care consideră obiectivul ca fiind atins, sau că acesta nu poate fi atins, sau că nu se mai dorește atingerea acelui obiectiv. Spre deosebire de alte sisteme în Jadex nu trebuie ca toate obiectivele să fie consistente unul cu celălalt. Pentru a distinge între obiectivele nou propuse și obiectivele pentru care deja se îndeplinesc acțiuni, s-a creat conceptul de ciclu de viață al obiectivului, obiectivele putându-se astfel afla în 3 stări: *activ*, *suspendat* și *opțional*. Când un nou obiectiv este introdus el intră în sistem cu starea opțional. Mecanismul de

deliberare a obiectivelor, este responsabil pentru determinarea tranzițiilor pentru obiectivele active. Unele obiective pot fi valide doar în contexte specifice determinate de presupunerile agentului. Când un obiectiv devine invalid pentru contextul curent, acesta este marcat ca suspendat până la validarea contextului.

Patru tipuri de obiective sunt suportate de către mediul Jadex. Primul tip se referă la obiectivele care definesc o acțiune care trebuie executată, dar care nu neapărat conduce la un rezultat specific. Al doilea tip se referă la obiectivele care definesc un scop sau care au o țintă fără însă a se specifica cum se atinge acel obiectiv. Al treilea tip se referă la obiectivele care reprezintă o nevoie pentru informații noi. Dacă informațiile cerute nu sunt la dispoziție, planurile sunt selectate și executate pentru a aduna informațiile necesare. Al patrulea tip și ultimul reprezintă obiectivele care specifică o stare a agentului care trebuie păstrată odată ce a fost atinsă. Toate obiectivele sunt reprezentate ca obiecte care au câteva atribute. Starea finală poate fi specificată ca o expresie evaluată pentru a verifica dacă obiectivul a fost atins. Structura obiectivelor care sunt în starea activă este memorată într-o bază de obiective. Un agent BDI specificat în Jadex are câteva obiective de nivel înalt, care servesc ca puncte de intrare în baza de obiective. Obiectivele pot fi separate în sub-obiective, formându-se astfel o structură arborescentă de obiective.

Planurile descriu acțiunile concrete pe care un agent le poate executa pentru a îndeplini obiective. Un dezvoltator dezvoltă un plan specificând antetul planului și planul propriu-zis. Antetul conține condițiile sub care planul poate fi executat și este specificat în fișierul de descriere a agentului. Corpul planului este o structură secvențială, care descrie acțiunile care trebuie executate pentru a atinge un obiectiv sau pentru a reacționa la un eveniment. În timpul execuției unui plan se pot defini noi sub-obiective și aștepta ca alte evenimente să apară.

3.3.3.2 Definirea agenților în Jadex

Definirea completă a unui agent se face în fișierul de definire a agentului. Acest fișier este un fișier în format XML care conține toate proprietățile relevante pentru un agent (presupuneri, obiective, planuri). Pe lângă descriptorii XML folosiți pentru componentele agentului, dezvoltatorul poate folosi expresii într-o variantă Java pentru a specifica valori sau parametri. Acest fișier este un fel de descriere a clasei agentului. Din acest fișier se instanțiază agenții precum obiectele se instanțiază din clasele lor. De exemplu, planurile sunt declarate specificând cum să fie instanțiate folosind clasele Java de descriere și un trigger cu care se determină condițiile în care planul se execută. Tot în acest fișier se definește starea inițială a unui agent folosind o configurație inițială. Scrierea corectă a fișierului de definiție a agentului va fi detaliată în secțiunea de implementare a lucrării curente.

4 SPECIFICAȚII ȘI ARHITECTURĂ SISTEM

4.1 Specificațiile sistemului

Principalele specificații ale sistemului sunt controlarea și monitorizarea mediului, dispozitivelor și aplicațiilor din casa inteligentă. Sistemul va avea în componență trei tipuri de roboți care vor trebui adaptați la contextul casei inteligente.

4.1.1 Controlul casei inteligente

Sistemul Smart House va furniza control automat al unor obiecte casnice (aparat de aer condiționat, încălzirea centrală) împreună cu controlul factorilor de confort din casă (luminozitate, umiditate, căldură), sistemul fiind capabil să ia o decizie asupra acțiunilor sau misiunilor care trebuie îndeplinite bazate pe datele contextuale procesate.

Cerințele funcționale ale sistemului dezvoltat curent vor fi prezentate în continuare.

4.1.1.1 Coordonarea acțiunilor

Aplicația va integra planuri de acțiuni predefinite în vederea efectuării anumitor execuții complexe și totodată va permite facilitarea de a crea acțiuni definite de utilizator. Utilizatorul va avea abilitatea de a stabili obiectivele primare ale sistemului, precum maximizarea confortului locuitorilor casei inteligente. Obiectivele care trebuie atinse când se vorbește de coordonarea acțiunilor sunt:

- *Crearea sau anularea unei acțiuni* – Utilizatorul va putea să introducă noi acțiuni sau să replanifice sau să anuleze misiunea curentă. În cazul în care se produc dependențe în executarea unei misiuni planificate, sistemul va putea crea acțiuni care să fie executate înaintea acțiunii specificate.
- *Executarea acțiunilor* – Aplicația va executa în mod automat acțiunile planificate sau va putea să replanifice acțiunile care nu au fost terminate cu succes.
- *Crearea de rapoarte* – Orice acțiune îndeplinită în sistem va fi detaliată într-un raport de execuție.

4.1.1.2 Executarea operațiilor

Sistemul va putea să execute operații de bază ca răspuns la schimbările de context sau acțiuni primitive, ca operații unitare ale unei misiuni. Dintre operațiile care se vor putea desfășura în sistemul dezvoltat identificăm:

- *Manipularea de obiecte* – Roboții vor putea manipula obiecte, prelua și muta diferite obiecte între diferite locații ale casei.
- *Controlul dispozitivelor* – Sistemul poate să își adapteze comportamentul în mod automat, modificând starea diferitelor dispozitive ubiquitouse în acord cu situația curentă sau intervenția directă a utilizatorilor. Cerința primordială a sistemului este să creeze o atmosferă cât mai confortabilă pentru locuitorii casei inteligente, prin controlarea dispozitivelor, în funcție de anumiți parametri preluați ca date de context (de ex. temperatura). Printre parametrii care pot fi manipulați se numără intensitatea luminoasă, volumul muzicii ambientale, temperatura, umiditatea etc.

4.1.1.3 Interacțiunea cu utilizatorii

Utilizatorii trebuie să poată interveni în sistemul casei inteligente, după bunul plac, astfel sistemul necesită o interfață utilizator pentru crearea de acțiuni utilizator sau controlarea

acțiunilor și parametrilor curenți. Informațiile pot fi prezentate la cererea utilizatorului sau automat când se execută o anumită acțiune specifică.

4.1.2 Monitorizarea casei inteligente

4.1.2.1 Achiziția datelor de context

Sistemul va putea extrage informații contextuale de nivel inferior de la un număr mare de furnizori de informație, prin senzori fizici sau programe software. Aceste informații de context care vor fi urmărite sunt:

- *Urmărirea locației unei persoane, robot, sau obiect* – Locația este cea mai importantă dată contextuală care este procesată. Sistemul trebuie să poată detecta prezența membrilor familiei sau a intrușilor, locația acestor persoane în timp-real, precum și locația roboților casei inteligente sau a altor dispozitive relevante.
- *Identificarea persoanelor, obiectelor, roboților* – Persoanele sau roboții trebuie identificați de sistemul casei inteligente, pentru a programa acțiuni specifice fiecărui actor din casa inteligentă în mod separat.
- *Monitorizarea condițiilor de mediu din interiorul sau exteriorul casei inteligente* – Monitorizarea acestui tip de condiții va fi posibilă prin achiziționarea informațiilor de la diferitele tipuri de senzori: luminozitate, umiditate, zgomot, presiune etc..
- *Monitorizarea dispozitivelor și a aplicațiilor casei inteligente* – Deoarece dispozitivele și actuatorii în sistemul casei inteligente pot furniza informații contextuale valoroase, monitorizând care dintre dispozitive este folosit precum și starea lor actuală este foarte importantă pentru îndeplinirea obiectivelor sistemului curent. Astfel sistemul va monitoriza:
 - Starea ușilor – închis / deschis / încuiat;
 - Starea porții de la garaj – închis / deschis;
 - Starea sistemului de încălzire – închis / deschis / nivel de încălzire;
 - Starea sistemului de iluminare – închis / deschis;
 - Starea ferestrelor – închis / deschis ș.a.m.d.

4.1.2.2 Procesarea datelor de context

Raționamentele aplicate pe datele contextuale, inferă date contextuale de nivel superior din date de context senzoriale și atomice, menținând consistența bazei de cunoștințe. Sistemul va îngloba reguli de inferență și de asemenea reguli auto-configurabile, pentru a putea raționa asupra informațiilor contextuale.

Determinarea tipului de activitate înfăptuită de un locuitor al casei inteligente este o problemă complexă pentru sistemul senzitiv la context al casei și este bazată pe achiziția informațiilor atomice de nivel inferior precum locația, monitorizarea condițiilor de mediu sau a dispozitivelor folosite.

Aceste date sunt procesate și în timp sistemul stabilește preferințele personale ale fiecărui locuitor, pentru ca ulterior să determine programul zilnic al fiecărui locuitor. Astfel sistemul va îngloba și funcția de învățare, pentru a deduce și actualiza preferințele utilizatorilor.

4.1.2.3 Propagarea datelor de context

După achiziționarea informațiilor contextuale relevante, datele contextuale trebuie să fie propagate la dispozitive, roboți, aplicații. De asemenea aceste informații necesită să fie adaptate în funcție de tipul dispozitivului, în cazul în care un anumit dispozitiv al casei inteligente necesită acele informații.

4.1.3 Adaptarea roboților la casa inteligentă

Cele trei tipuri de roboți cu care va fi prevăzut sistemul casei inteligente sunt robotul supervisor, robotul care îndeplinește misiuni și robotul de companie, care se ocupă de divertismentul locuitorilor casei.

Acești 3 roboți vor trebui adaptați la sistemul casei inteligente. Provocările construirii adaptabilității roboților la sistemul casei inteligente se rezumă la două motivații-cheie. Prima, un model adecvat sistemului casei inteligente, proiectat pentru a îndeplini misiunile pentru care ei au fost introduși, respectiv, descrierea modelului folosind un limbaj cunoscut sistemului.

4.1.3.1 Reprezentarea adecvată a roboților

Unul din rolurile esențiale ale unei reprezentări este ca aceasta să îndeplinească rolul de substitut al obiectului real care trebuie controlat. O reprezentare bună a roboților face de asemenea posibilă raționarea pe baza modelului, putând astfel executa diferiți algoritmi de raționare pentru a infera cunoștințe noi pentru a reflecta cât mai bine cu putință evoluția ulterioară a lumii în care acționează sistemul.

Elaborarea unui astfel de model a fost tot timpul o provocare pentru oricine a fost interesat de reprezentarea unei părți din lumea reală. Faptul că trebuie să reprezentăm de exemplu elementele reprezentative ale stării în care un membru al familiei, este supărat sau gânditor, pentru a putea alerta robotul de companie să înceapă să își facă datoria poate fi de multe ori o problemă complexă și greu de atins.

4.1.3.2 Comunicarea dintre roboți

În plus față de reprezentarea adecvată a roboților, inteligența sistemului robotic constă de asemenea în interacțiunea acestuia cu mediul casei inteligente. Roboții care execută servicii pentru un mediu casnic inteligent, trebuie să aibă mai multe funcționalități cheie: localizare, navigare, cartografiere, interacțiune om-robot, recunoaștere de obiecte și manipulare de obiecte. Pentru a executa aceste funcționalități roboții trebuie să fie în legătură cu sistemul senzorial al casei, precum și cu ceilalți roboți pentru a prelua informații relevante.

4.1.3.3 Serviciile roboților din casa inteligentă

Scenariile pentru roboți dorite în casa inteligentă tratată în lucrarea de față sunt:

- **Mutarea obiectelor** – Acest scenariu are la bază mutarea obiectelor din camera copiilor când aceștia sunt plecați de acasă, precum și a altor obiecte din diferite camere, în contextul în care robotul este înștiințat că aceste obiecte nu se află la locul lor. Robotul („mission robot”) desemnat pentru a îndeplini această sarcină navighează prin cameră și identifică respectivele obiecte folosind semnale RFID, ridică, transportă și depozitează obiectele în poziția desemnată. Printre obiectivele urmărite din acest scenariu sunt depozitarea jucăriilor în cutii și depozitarea lor în locul lor de depozitare, de exemplu, garaj.
- **Servicii de divertisment pentru copii** – Acest serviciu este un serviciu complex, în care se detectează starea de spirit a copiilor, în cazul în care se detectează că unul dintre aceștia ar fi supărat, robotul de divertisment („pet robot”) intră în acțiune, îndeplinind anumite planuri preluate din baza sa de cunoștințe și ducându-le la finalitate, până când se atinge unul din obiectivele fixate și anume ca starea de spirit a copilului supărat să se schimbe într-una mai veselă.
- **Patrularea prin diferite încăperi ale casei** – Acest scenariu este menit să ajute la supravegherea casei inteligente, să detecteze sosirea sau plecarea unui membru de

acasă, să detecteze eventuali intruși. Robotul („supervisor robot”) menit să înfăptuiască acest scenariu supraveghează buna funcționare a celorlalți roboți și detectează schimbări de mediu pentru a putea alerta sistemul sau ceilalți roboți de noile planuri și obiective care trebuie îndeplinite.

4.2 Arhitectura sistemului

Cum a fost detaliat în capitolele anterioare, dezvoltarea sistemelor multi-agent omniprezente trebuie să ia în considerare o serie de factori:

- Omniprezența dispozitivelor folosite;
- Interoperabilitate;
- Comportament proactiv;
- Scalabilitate;
- Acuratețe;
- Mobilitate;
- Securitate.

Pentru flexibilitate și scalabilitate arhitectura aleasă este o arhitectură orientată pe agenți, distribuită și ierarhică. Este alcătuită din 5 componente legate între ele, precum în Figura 4.1 Diagrama arhitecturală a sistemului.

4.2.1 Nivelul Senzorial

Acest nivel controlează traficul de informații captate de la senzori și le transmite la sistem prin platforma unde urmează să fie prelucrate. Agenții logici software de la acest nivel transformă informațiile preluate de la nivelul fizic, transformându-le în informații inteligibile pentru calculator.

Agenții senzor de la acest nivel achiziționează datele specifice și controlează acuratețea lor fiind capabili să detecteze funcționarea malițioasă a senzorilor sau alte tipuri de erori.

Pentru fiecare tip de senzor din sistem este responsabil un agent logic, care monitorizează schimbările de context și eventualele erori apărute.

Un agent supervisor supraveghează ciclul de viață a agenților logici senzoriali și este responsabil de pornirea acestora și de controlarea lor.

4.2.2 Nivelul Dispozitivelor

Nivelul dispozitivelor are un scop asemănător cu nivelul senzorilor, cu deosebirea că agenții specifici acestui nivel manipulează datele venite de la nivelul fizic legate de celelalte dispozitive senzitive la context acestea fiind achiziționate la acest nivel înainte de a fi trimise la sistem.

Un agent este responsabil pentru fiecare dispozitiv omniprezent, astfel toate datele preluate de la unul dintre aceste dispozitive fiind procesate la acest nivel și trimise mai departe la sistem

De asemenea un agent supervisor este responsabil de ciclul de viață al agenților pentru dispozitive și este responsabil de pornirea și oprirea acestora.

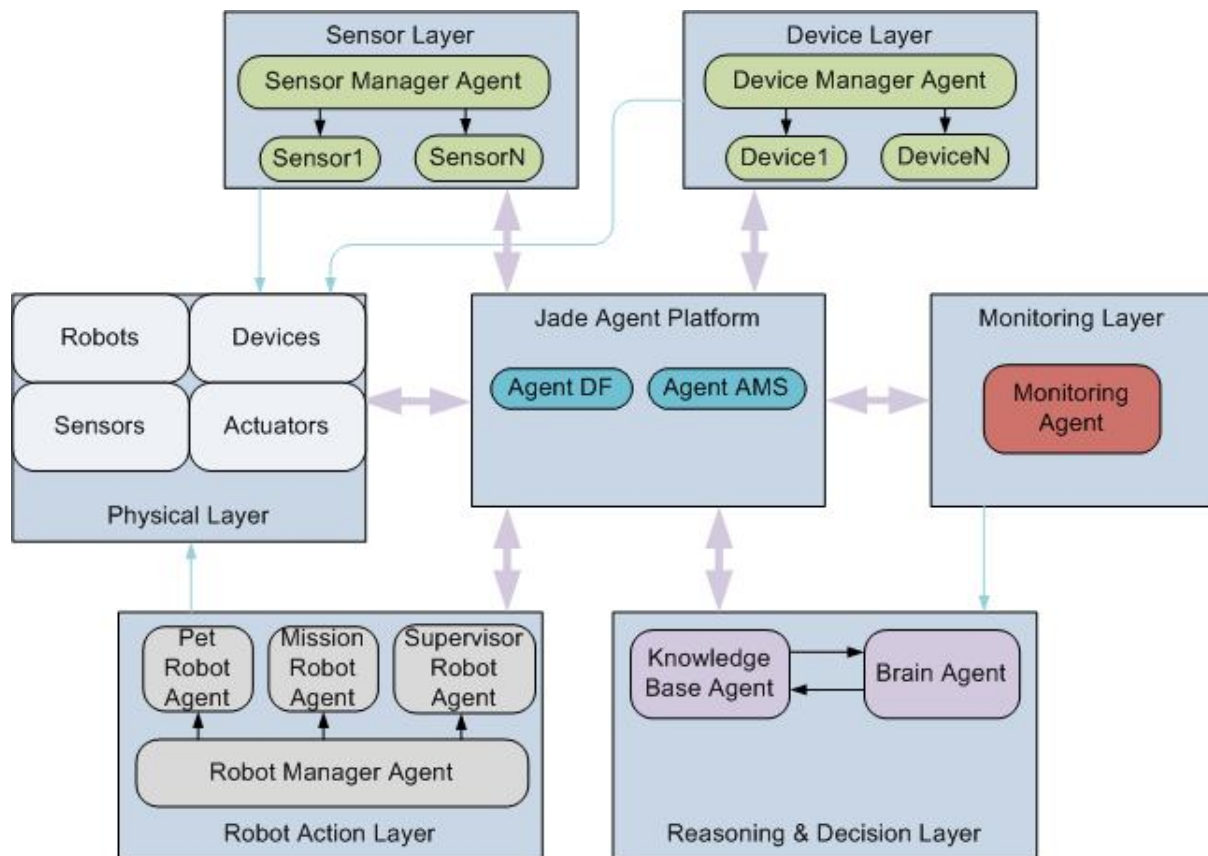


Figura 4.1 Diagrama arhitecturală a sistemului

4.2.3 Nivelul Roboților

În acest nivel se mapează la nivel de agenți roboții din sistem preluând informațiile de la nivelul fizic legate de roboți și introducându-le în sistem. Cei trei roboți monitorizați prin agenți logici sunt:

- Pet Robot – robotul preocupat de divertismentul din casă;
- Mission Robot – mutarea diferitelor obiecte din casă;
- Supervisor Robot – care monitorizează activitatea roboților precum și alte schimbări legate de locuitorii casei.

Agentul logic Robot Manager este responsabil de controlarea agenților logici pentru roboți, fiind sensibil la modificările apărute în activitatea roboților activi din casă.

4.2.4 Nivelul Decizional

La acest nivel, prin introducerea regulilor de inferență, informațiile despre context de nivel înalt sunt deduse din informațiile de context preluate de la diferitele dispozitive ubiquitous. Tot acest nivel este responsabil de crearea, menținerea și actualizarea bazei de cunoștințe.

Astfel baza de cunoștințe furnizează informații actualizate salvate ca urmare a unor diferite acțiuni din sistem și salvează informațiile contextuale specifice fiecărei entități din sistem. Agentul logic responsabil pentru baza de cunoștințe are responsabilitatea de a raționa în privința contextelor existente. Rezultatele raționamentelor făcute de acest agent sunt comunicate celorlalți agenți prin platformă, acestea fiind concretizate prin acțiuni.

Tot acest nivel are și funcția de învățare, învățând preferințele actorilor umani ai casei inteligente.

4.2.5 Nivelul de control și monitorizare

Acest nivel concretizează interacțiunea dintre roboți și oameni, precum și dintre oameni și sistem, furnizând capacități de nivel înalt pentru interacțiunea dintre entitățile din casa inteligentă.

Dintre facilitățile oferite de acest nivel enumerăm: înregistrarea a noi entități fizice în sistem, precum un dispozitiv utilizator, înregistrarea unor noi acțiuni sau misiuni, reprezentarea grafică a mediului casei pentru utilizator.

4.2.6 Nivelul fizic

Acest nivel reprezintă senzorii la nivel fizic, diferite obiecte, dispozitive, actuatori care furnizează informații de cele mai multe ori eronate sau inconsistente despre sistem. Fiecare informație provenită de la o categorie de dispozitive fizice este controlată și prelucrată de un anumit nivel din arhitectura curentă, înainte de a fi trimisă la nivelul de achiziționare de context din sistem.

5 PROIECTARE DE DETALIU

În acest capitol se detaliază procedura de proiectare a sistemului Smart House. Aici se va explicita proiectarea sistemului începând de la nivelul inferior până la proiectarea sistemului în ansamblu. De asemenea se vor prezenta detaliile de arhitectură, diagramele de clase, cazurile de utilizare precum și detaliile de implementare și dificultățile întâmpinate în decursul dezvoltării sistemului.

5.1 Analiza cerințelor

În această fază se organizează actorii și obiectivele majore, se definesc participanții la sistemul curent, precum și legăturile între aceștia. Această fază de analiză se completează răspunzând la câteva întrebări și modelarea răspunsului la acestea în mod vizual în diagrama de interacțiune a sistemului (Figura 5.1).

- Cine sunt actorii principali ai sistemului?

Etapă de identificare a participanților la sistemul casei inteligente a demonstrat că actorii principali care interacționează cu sistemul sunt membrii familiei care au autoritate asupra sistemului, putând face modificări și lua decizii, precum și alți membri care nu aparțin familiei și care pot fi tratați ca invitați sau intruși. Oaspeții au privilegii limitate, în timp ce intrușii nu au privilegii în sistem.

- Care sunt obiectivele lor?

Membrii familiei au următoarele obiective în sistemul casei inteligente:

- A. Managementul Misiunilor – membrii familiei vor putea planifica sau anula anumite misiuni;
- B. Controlul Casei inteligente – reprezintă abilitatea unui actor de a controla aparatele, senzorii, actuatorii sau roboții casei inteligente;
- C. Vizualizarea hărții casei inteligente – reprezintă abilitatea de a vizualiza mediul inteligent al casei incluzând poziția oamenilor, roboților, statusul dispozitivelor, al senzorilor și informații deduse despre context, precum starea emoțională dedusă.

- Cum pot aceștia să le îndeplinească, depinzând sau nu de ceilalți actori ai sistemului?

Pentru a îndeplini starea de fapt dorită de membrii familiei, se pot asigura responsabilitățile și obiectivele agentului de interacțiune cu sistemul casei inteligente. Acesta îndeplinește rolul de agent de proximitate, fiind cel care interacționează cu sistemul cerând o planificare a noilor obiective cerute de către membrul familiei.

Următorul pas în această fază este analiza fiecărui obiectiv și descompunerea lui în sub-obiective cu cât mai atomice posibil, privind din perspectiva unui singur actor, și arătând mijloacele de a le îndeplini, adică alte obiective care ar putea preveni sau ar putea contribui la atingerea obiectivului curent.

După ce s-a răspuns la aceste întrebări, următorul pas este identificarea actorilor secundari care contribuie la sistemul Smart House. Actorii secundari ai sistemului sunt sistemul casei inteligente cu cele două ramificații: sistemul de control și sistemul de monitorizare.

Sistemul de control se bazează pe agenții de management ai senzorilor, dispozitivelor, roboților și actuatorilor pentru execuția de planuri în vederea atingerii obiectivelor.

Sistemul de monitorizare construiește harta mediului casei inteligente furnizând informații personalizate actorilor familiei prin mijlocirea oferită de agentul de interacțiune.

Harta mediului este procesată de către sistemul de control care include mecanisme decizionale pentru a se adapta în mod dinamic la noile informații contextuale preluate.

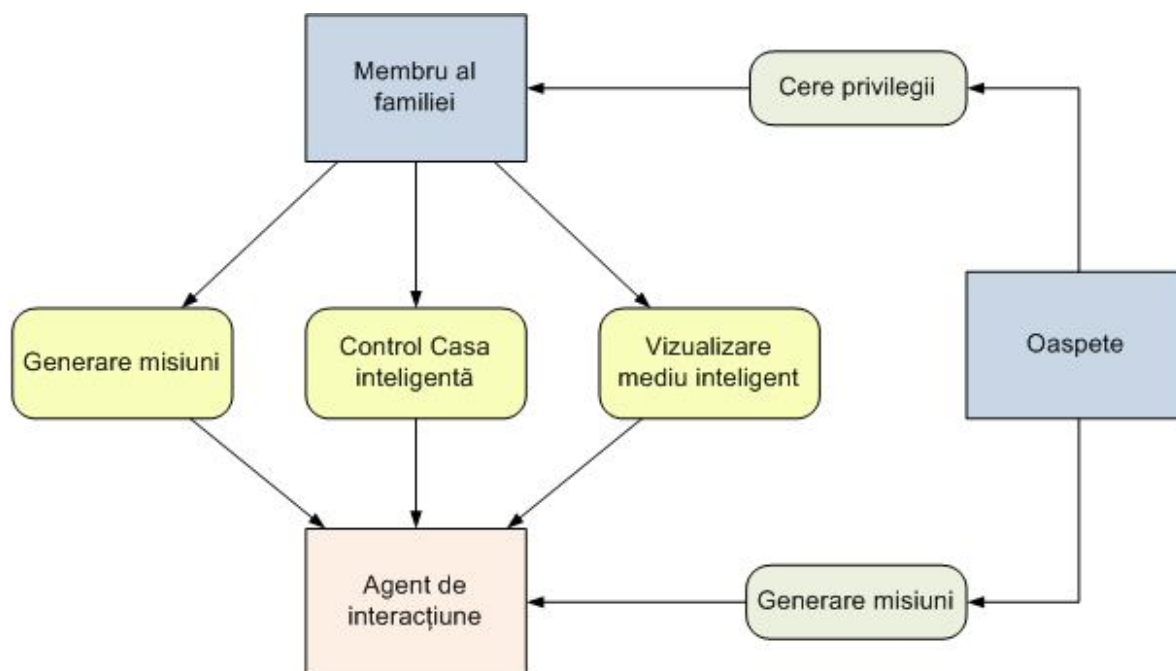


Figura 5.1 Diagrama de interacțiune între actorii și sistemul casei inteligente

5.2 Proiectarea arhitecturii

Proiectarea și implementarea arhitecturii sistemului multi-agent prezentată în capitolul anterior va fi detaliată în această secțiune. Agenții implicați la nivelul fiecărui modul sunt responsabili pentru a atinge anumite obiective, pentru aceasta ei efectuând anumite planuri care sunt cerute de către sistemul de roboți. Anumiți agenți suplimentari sunt introduși pentru a contribui la îndeplinirea unor cerințe funcționale sau non-funcționale.

Mijloacele prin care se pot atinge obiectivele, și anume planurile și presupunerile inițiale, pot fi descompuse până la activități unitare. Proiectarea arhitecturală cuprinde următoarele faze:

- *Descompunerea și detalierea diagramei arhitecturale generală* – această fază implică apariția a noi actori și noi sub-obiective și planuri care au ca scop îndeplinirea cerințelor funcționale ale sistemului;
- *Includerea de noi actori specifici stilului arhitectural;*
- *Includerea de noi actori care să contribuie proactiv la îndeplinirea cerințelor non-funcționale;*
- *Identificarea capacităților.*

5.2.1 Sistemul de roboți

Sistemul SmartHouse dezvoltat fiind echipat cu 3 roboți, un subsistem multi-agent a fost proiectat pentru a se ocupa cu acțiunile specifice roboților. Roboții trebuie să poată să îndeplinească misiuni de manipulare a obiectelor, dar au și capacități de raționare pentru a putea opera în mod autonom, expunând astfel un comportament senzitiv la context.

Agentul manager al roboților are rolul de a iniția misiunea celor trei agenți roboți. Fiecare dintre acești trei agenți are responsabilități bine definite, și poate să execute misiuni

planificate sau chiar și neplanificate, în funcție de contextul curent. Diagrama de obiective majore pentru sub-sistemul de roboți poate fi văzută în Figura 5.2. Fiecare dintre cei trei roboți are planuri și obiective diferite. În continuare vom detalia arhitectura agenților roboți.

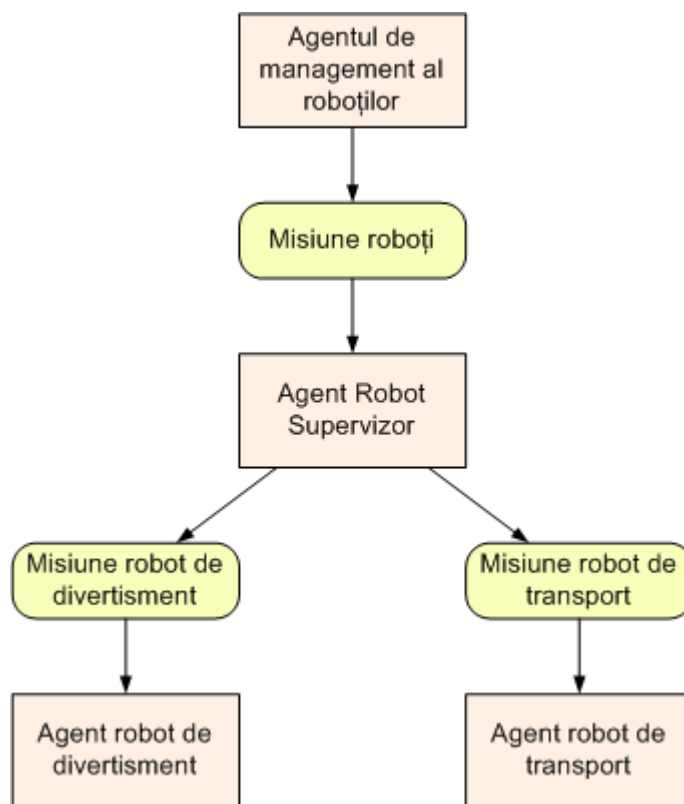


Figura 5.2 Diagrama de obiective a sub-sistemului de roboți

5.2.1.1 Agentul Robot Manager

Misiunea acestui agent este crearea sistemului de agenți roboți al sistemului multi-agent al casei. Atunci când pornește sistemul Smart House îndeplinește următoarele acțiuni:

- *Înregistrare agent la platforma de agenți* – Acest lucru se face printr-un sistem request-response prin care se cere platformei autorizarea pornirii agentului manager.
- *Căutare agenți roboți în baza de cunoștințe* – După ce înregistrarea acestui agent s-a înfăptuit cu succes el este responsabil cu căutarea în baza de cunoștințe a agenților de tip robot pe care trebuie să îi pornească.
- *Pornire agenți roboți* – După ce au fost găsite toate posibilele instanțe de agenți, managerul de roboți caută misiunile fiecăruia dintre roboți și pornește fiecare tip de agent robot cu misiunea asignată.
- *Supraveghere funcționare agenți roboți* – După procesul de pornire a acestor agenți, dacă procesul a fost terminat cu succes, se supraveghează din punct de vedere funcțional software buna funcționare a agenților de tip robot.
- *Oprire agenți roboți* – Dacă se declanșează procesul de oprire a sistemului multi-agent, acest agent este alertat cu acest lucru și oprește toți agenții logici aflați în subordinea sa.

Agentul manager de roboți are asignate următoarele ipoteze, planuri și obiective:

- **Ipoteze** („beliefs”)
 - *Tipul roboților* – aceasta ipoteză este necesară pentru a ști ce tip de cunoștințe să caute în baza de cunoștințe;
 - *Creare robot nou* – această ipoteză specifică tipul de obiect cu care va fi instanțiat robotul;
 - *Distrugere roboți* – această ipoteză permite agentului manager distrugerea celorlalți agenți folosind agentul AMS.
- **Planuri** („plans”)
 - *Plan de găsim a agenților roboți* – Acest plan se concretizează printr-o operație de extragere a tipurilor de roboți din baza de cunoștințe folosind un mecanism de tip request-response între acest agent și agentul responsabil cu baza de cunoștințe;
 - *Plan de distrugere a agenților roboți* – Acest plan se concretizează prin distrugerea agenților roboți creați.
- **Obiective** („goals”)
 - *Creare agent* – Acest obiectiv este o referință spre unul din obiectivele agentului master al platformei numit AMS („Agent Management Service”);
 - *Înregistrare agent* – Acest obiectiv este o referință spre unul din obiectivele agentului de tip DF al platformei de agenți.

După cum s-a văzut mai sus acest agent manager de roboți joacă un rol esențial în crearea sub-sistemului multi-agent pentru roboții casei. El nu este implicat direct în vreuna din misiunile de bază ale sistemului multi-agent al casei, dar se comportă ca un agent master care comunică cu sistemul de control al casei, fiind responsabil pentru managementul celorlalți agenți robotici ai casei.

5.2.1.2 Agentul Robot Supraveghetor

Agentul robot supraveghetor este agentul care mapează la nivel software acțiunile pe care robotul supervisor trebuie să le îndeplinească, el având misiunea de a supraveghea acțiunile celorlalți roboți care participă activ la misiunea casei, efectuând acțiuni în scopul de sporire a confortului locatarilor.

Acțiunile acestui agent robot sunt:

- *Căutare agenți roboți în baza de cunoștințe* – Această operațiune se execută odată cu crearea acestuia de către agentul manager, el căutând instanțele de roboți pe care va trebui să îi supravegheze, căutând toate detaliile (ipoteze, planuri, obiective) despre ei și monitorizând buna funcționare a acestora;
- *Procesarea schimbărilor de context* – Schimbarea sau adăugarea de noi cunoștințe care au impact direct asupra sub-sistemului de roboți vor fi sesizate și procesate de către acest agent;
- *Înregistrarea acțiunilor efectuate de robotul misiune și robotul divertisment* – Planurile și obiectivele care trebuie atinse de către roboții supervizați sunt sesizate de către acest robot;
- *Planificarea de noi acțiuni* – În cazul unor schimbări de cerințe sau alte schimbări de context, robotul supervisor va putea planifica noi acțiuni pe care roboții supervizați să le îndeplinească;
- *Înregistrarea de noi ipoteze* – În cazul în care în baza de ipoteze a subsistemului de roboți apar ipoteze acestea vor fi sesizate de către agent și vor fi procesate;
- *Stabilirea de noi obiective* – Obiectivele noi vor fi planificate de către acest agent pentru a fi atinse.

Acest agent are asignate următoarele ipoteze, planuri și obiective:

- **Ipoteze** („beliefs”)
 - *Locație* – locația inițială în care se află robotul;
 - *Tipul robotului* – tipul robotului;
 - *Început misiune* – timpul în care pornește misiunea;
 - *Sfârșit misiune* – timpul în care se sfârșește misiunea;
 - *Mediul curent al casei inteligente* – această ipoteză se mapează printr-un obiect de tip singleton care conține toate cunoștințele despre casă;
 - *Mediul vizual curent al casei inteligente* – această ipoteză creează mediul casei inteligente în mod vizual.
- **Planuri** („plans”)
 - *Supraveghează ținte* – supraveghează roboții țintă;
 - *Patrulare* – Patrulează încăperile casei.
- **Obiective** („goals”)
 - *Înregistrare DF* – înregistrare robot la registrul de tip „pagini aurii”;
 - *Deregistrare DF* – deînregistrare robot de la serviciul de tip „pagini aurii”;
 - *Redă cunoștințele despre casă roboților care îndeplinesc acțiuni* – Această funcție redă toate cunoștințele despre mediu existente în baza de cunoștințe tuturor actorilor de tip robot.

5.2.1.3 Agentul Robot Misiune

Agentul Robot pentru misiuni este un robot specializat care are ca scop executarea diferitelor munci prin casa inteligentă. Capacitățile de comunicare a acestui robot cu ceilalți agenți roboți și cu mediul Smart House sunt:

- *Cerere de vizualizare a mediului și înregistrare a acestuia* – Robotul de tip misiune cere robotului supervisor informații despre casa inteligentă, coordonate de mișcare, harta casei, și alte informații;
- *Descoperire obiecte utile* – În cazul în care robotul are nevoie de anumite obiecte pentru a duce la capăt o misiune (o cutie, un loc de depozitare), în cazul în care acesta trece pe lângă vreunul dintre aceste obiecte, le identifică și le adaugă în baza proprie de cunoștințe;
- *Cerere de actualizare a mediului casei inteligente* – În cazul în care se produce o schimbare în mediul robotul de tip misiune, cere robotului supervisor reîncărcarea viziunii asupra lumii și implicit a mediului casei inteligente.

Obiectivele majore ale acestui agent robot sunt proiectate astfel încât robotul misiune să poată îndeplini obiectivele majore prezentate în Figura 5.3. Astfel obiectivele majore ale acestui robot de tip misiune sunt:

- *Menține acumulatorul încărcat* – Dacă nivelul acumulatorului scade sub un anumit nivel, celelalte obiective devin ignorate pentru moment, singurul obiectiv al robotului de tip misiune rămânând obiectivul de a menține acumulatorul peste nivelul de urgență. Astfel se spune că acest tip de obiectiv este *inhibant*. Este obiectivul de tip obiectiv de menținut setat pentru robotul misiune;
- *Menține camera copiilor curată* – Dacă în camera copiilor se află o cutie cu jucării, mută această cutie în locul de depozitare cunoscut. Acest obiect este blocant față de celelalte obiective, în cazul în care robotul identifică o cutie care trebuie transportată acesta își setează ca obiectiv mutarea acelei cutii, în pasul următor, indiferent dacă robotul se află într-o altă misiune de depozitare a vreunei cutii. Acest obiectiv are asignat un plan de căutare a cutiilor, înregistrând în baza proprie de cunoștințe noile cutii apărute în mediul casei inteligente;

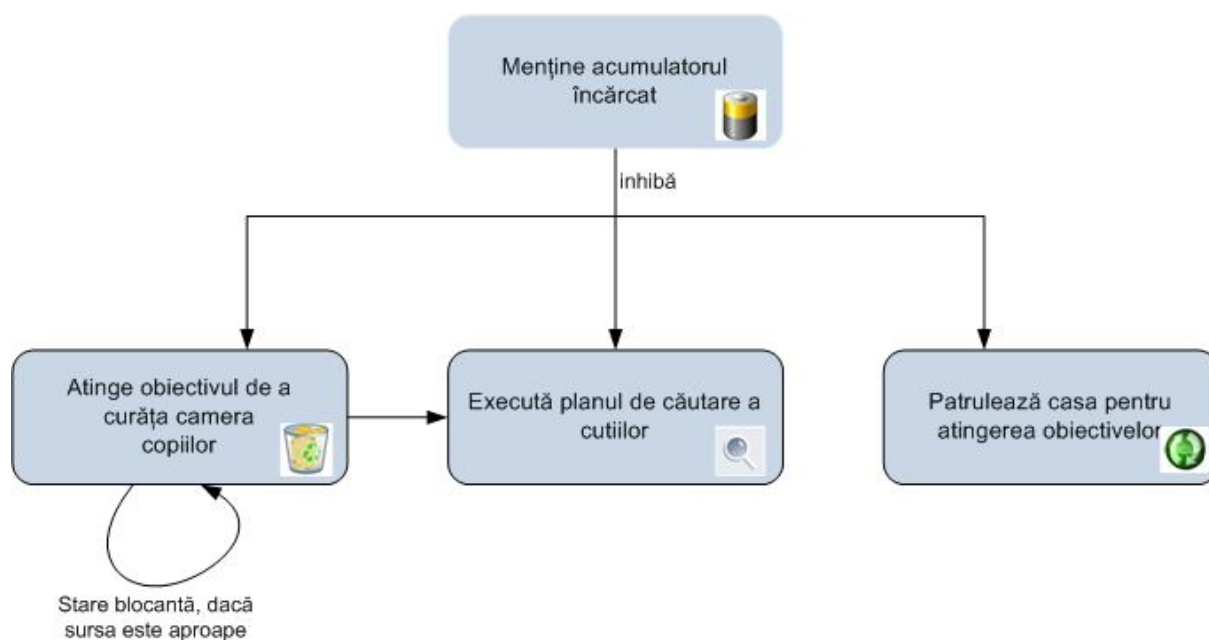


Figura 5.3 Obiectivele robotului transportor în casa inteligentă

- *Patrulează casa pentru atingerea obiectivelor* – Se setează anumite puncte de patrulare ale robotului care sunt urmărite în modul normal când robotul nu se află în misiune de reîncărcare sau în alt tip de misiune. În mod normal dacă apare o schimbare de context, robotul iese din modul de patrulare, îndeplinind obiectivele prioritare.

Ipotezele, obiectivele și planurile acestui tip de robot sunt:

- **Ipoteze** („beliefs”)
 - *Mediul curent* – Robotul transportor are referință spre mediul curent;
 - *Cutiile reperate* – În cazul în care robotul transportor a reperat o cutie el adaugă acest reper în baza de ipoteze;
 - *Locurile de depozitare reperate* – Dacă robotul a trecut prin raza locului de depozitare acesta adaugă acel loc de depozitare ca un fapt cunoscut în baza lui de ipoteze;
 - *Puncte de încărcare* – Robotul reperează cu ajutorul senzorilor de locație punctele de încărcare a acumulatorilor;
 - *Locația* – Locația curentă a robotului. Această ipoteză se actualizează cu fiecare mișcare a robotului;
 - *Raza vizuală* – Această ipoteză reprezintă un factor care determină aria vizuală a robotului. Toate obiectele de interes care se vor afla în interiorul acestei arii vizuale vor fi reperate și salvate ca ipoteze;
 - *Nivelul acumulatorului* – Această ipoteză reprezintă procentul de încărcare a acumulatorului robotului actualizată odată cu mișcarea robotului sau încărcarea lui;
 - *Punctele de patrulare* – Aceste constante reprezintă coordonatele de mișcare ale robotului. Robotul se va mișca între aceste puncte dacă nu are un alt obiectiv mai prioritar de îndeplinit;
 - *Mediul curent în mod vizual* – Această ipoteză se concretizează prin instanțierea unei ferestre de vizualizare a acțiunilor robotului transportor.
- **Planuri** („plans”)

- *Planul de patrulare prin casă* – Patrularea prin casă se concretizează prin executarea unui plan în care robotul iterează în mod ciclic niște sub-planuri prin care robotul se mișcă de la locația sa curentă la locația următoare specificată;
- *Planul de ridicare a unei cutii* – Acest plan se concretizează prin cererea către robotul supervisor de actualizare a mediului și crearea unei noi ipoteze a robotului. Obiectivul final care trebuie îndeplinit ca urmare a executării planului este obiectivul de ridicare a cutiei;
- *Planul de depozitare a unei cutii* – Acest plan se execută având ca obiectiv final depozitarea cutiei în garaj, fiind actualizate ipotezele legate de cutiile transportate de robot, cât și ipotezele legate de mediu;
- *Planul de mișcare într-o altă locație* – Acest plan este folosit ca sub-plan în cazul patrulării și se concretizează prin actualizarea ipotezelor legate de locație, precum și actualizarea mediului casei inteligente;
- *Planul de încărcare a acumulatorului* – Acest plan se concretizează prin stabilirea sub-obiectivului de mișcare a robotului spre locația locului de încărcare și încărcarea acumulatorului până la nivelul stabilit. Acest plan se execută cu scopul de a îndeplini obiectivul de încărcare a bateriei.
- **Obiective („goals”)** – Obiectivele în mediul de dezvoltare a agenților se despart în trei tipuri: obiective de menținut („maintain goals”), obiective de atins („achieve goals”), obiective de efectuat („perform goals”) și obiective de tip interogare („query goals”).
 - **Obiective de menținut**
 - *Menține bateria încărcată* – Acest obiectiv este obiectivul de cea mai înaltă prioritate în sistem, toate celelalte execuții de planuri pentru a îndeplini obiective fiind oprite pentru a se executa planul de mutare a robotului transportor înspre acumulator. Nivelul de energie care trebuie menținut este un nivel mai mare de 20% din valoarea totală a acumulatorului. Nivelul optim de energie este 100%, astfel la fiecare încărcare robotul așteaptă pentru a se atinge acest nivel.
 - **Obiective de atins**
 - *Obiectivul de a curăța camera copiilor* – Este obiectivul esențial pentru robotul de tip transportor, pentru executarea sa fiind create planurile descrise mai sus. Sub-obiectivele fixate, care trebuie îndeplinite pentru a putea atinge acest obiectiv major, sunt obiectivul de a ridica o cutie, obiectivul de a depozita o cutie și obiectivul de mișcare;
 - *Obiectivul de a ridica o cutie* – Obiectivul se atinge prin efectuarea cu succes a planului desemnat pentru acest lucru;
 - *Obiectivul de a depozita o cutie* – Obiectivul se atinge prin executarea planului de depozitare a cutiei;
 - *Obiectivul de a se mișca într-o anumită locație* – Obiectivul este concretizat prin efectuarea planului de schimbare a locației robotului.
 - **Obiective de efectuat**
 - *Patrulare* – Acesta este un obiectiv de prioritate joasă care se execută atunci când nici un alt obiectiv mai prioritar nu trebuie îndeplinit;

- *Efectuează căutarea cutiilor* – În mișcarea sa prin camera copiilor robotul identifică cutiile care trebuie depozitate.
- **Obiective de tip interogare**
 - *Caută stații de încărcare* – Acest obiectiv este un obiectiv de tip interogare, deoarece robotul cere informații despre mediu legate de stațiile de încărcare;
 - *Caută locuri de depozitare* – Robotul cere informații despre mediu prin robotul supervizor, legate de locurile de depozitare din casă.

5.2.1.4 Agentul Robot Divertisment

Agentul robot divertisment este un agent mai complex ,care trebuie să detecteze starea de spirit a copiilor, iar în cazul în care se consideră că trebuie să schimbe starea de spirit a acestora să o facă. Locul acestuia este tot în camera copiilor. Obiectivul său este îndeplinirea acțiunii de înveselire a copiilor. Pentru aceasta robotul de divertisment va executa diferite planuri în funcție de ipotezele formate și de cerințele venite de la robotul supervizor.

5.2.2 Sistemul de senzori

Sub-sistemul de senzori conține agenți logici care furnizează funcții de achiziție de context. În mod mai specific, ei adună informații despre context de nivel inferior, de la anumiți senzori. Agentul senzor de lumină, agentul senzor de temperatură, agentul senzor de umiditate, precum și agentul senzor de zgomot procesează informațiile venite de la senzori, care de cele mai multe ori sunt eronate, conțin zgomote. Se folosesc diferite metode specifice pentru a le interpreta și se comunică rezultatele agentului de monitorizare a sistemului. Agentul manager de senzori are rolul de a detecta și monitoriza agenții de tip senzor, detectând eventualele erori sau funcționalități anormale.

Managerul de senzori are obligația de notifica sistemul de monitorizare a casei prin agentul de monitorizare dacă un comportament anormal a fost detectat și totodată sistemul de control al casei, care ia măsurile specifice în vederea tratării acestui caz. O vedere detaliată despre relația dintre componentele sub-sistemului de senzori este detaliată în Figura 5.4.

Totodată, sistemul de monitorizare trebuie să fie capabil să dicteze acțiuni sistemului de senzori, prin agentul manager, acțiuni legate de funcționarea acestora precum și de parametrii de funcționare.

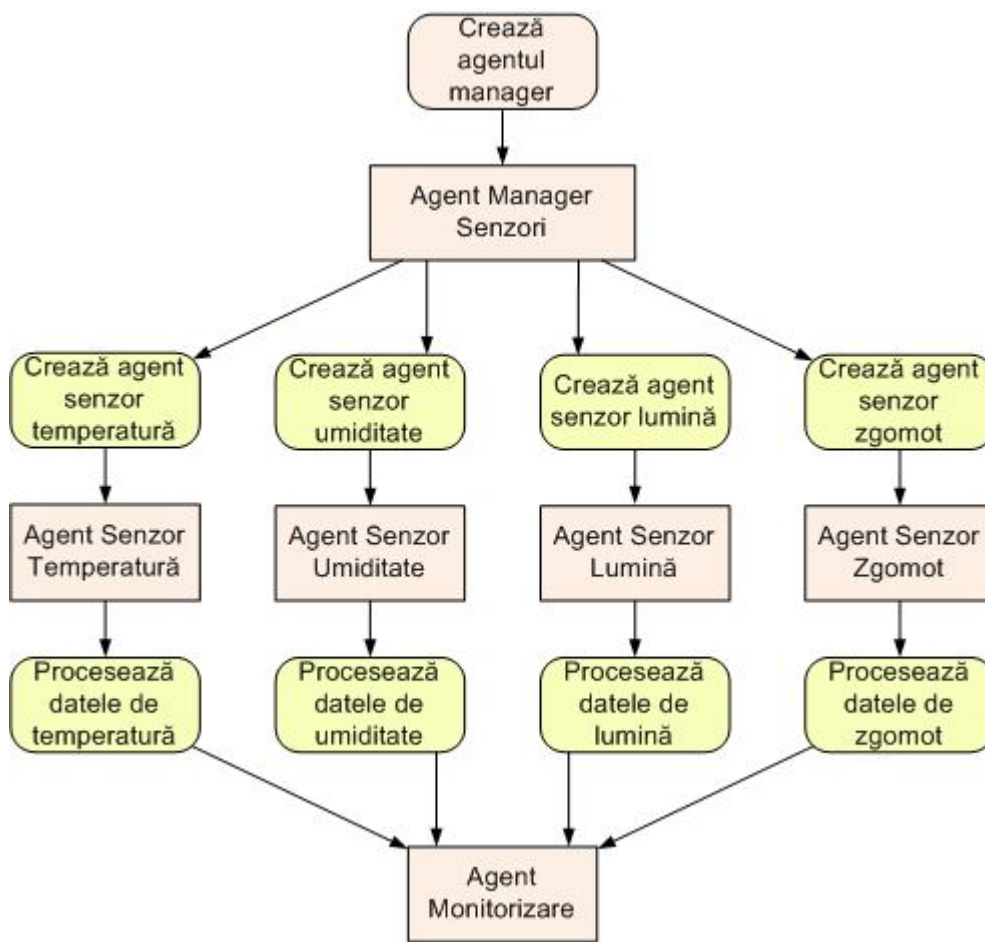


Figura 5.4 Sistemul de senzori al casei inteligente

5.2.3 Sistemul de dispozitive

Sub-sistemul de dispozitive monitorizează informațiile contextuale furnizate de echipamentele interioare ale casei (frigider, televizor ș.a.m.d.), supraveghind care dintre dispozitive sunt în folosință și care este starea lor de funcționare. Pentru fiecare dispozitiv un agent specific este delegat pentru a interpreta semnalele venite de la dispozitive și pentru a executa operații precum pornirea sau oprirea dispozitivului. Diagrama de activitate în sub-sistemul multi-agent pentru dispozitive poate fi vizualizată în Figura 5.5.

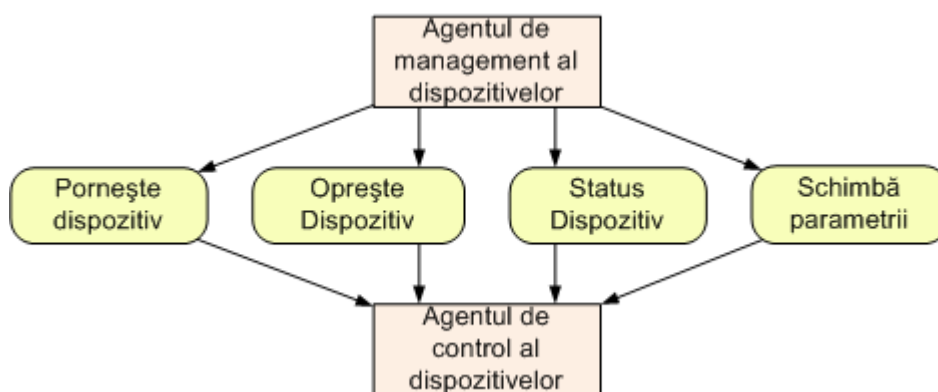


Figura 5.5 Sistemul de dispozitive al casei inteligente

5.2.4 Sistemul de control

Sistemul de control este responsabil de execuția de planuri decizionale bazate pe reguli. Acest sistem este sistemul de raționare a casei inteligente. Din baza de cunoștințe în care se adună cunoștințe de nivel inferior despre casa inteligentă prin mecanismele de raționare se deduc cunoștințe de nivel înalt. Obiectivele majore ale acestui sistem sunt precum cele prezentate în Figura 5.6.



Figura 5.6 Obiectivele sistemului de control al casei inteligente

5.2.5 Sistemul de monitorizare

Sistemul de monitorizare al casei inteligente folosește informațiile relevante precum locația persoanelor sau a roboților, identificarea actorilor sau statutul dispozitivelor sau a senzorilor. Agentul responsabil pentru monitorizarea casei, folosește aceste informații de nivel inferior pentru a le trimite mai departe la sistemul de control al casei, care deduce contexte de nivel superior din aceste informații primite, integrând aceste informații în sistemul de informații al casei.

Sistemul cuprinde câteva interfețe utilizator menite să furnizeze o reprezentare vizuală a informațiilor contextuale furnizate de senzori, dispozitive sau obiecte și de asemenea interactivitate vizuală pentru sistemul luat ca un întreg, prin crearea de noi planuri sau misiuni sau prin eliberarea comenzilor specifice pentru obiectele interioare.

Interfețele utilizator sunt proiectate ca fiind ipoteze pentru anumiți agenți Jadex, acestea instanțiindu-se odată cu înregistrarea cu succes a agentului la platforma de agenți. Aceste interfețe extrag informații contextuale din sistem folosind agenții specifici pentru fiecare domeniu specific. Următoarele funcționalități sunt furnizate de sistemul de monitorizare al casei:

- Reprezentare vizuală a locuitorilor, roboților, obiectelor sau dispozitivelor;
- Reprezentare vizuală a camerelor din casă;

- Reprezentare vizuală a informațiilor contextuale deduse, precum activitatea curentă sau starea emoțională;
- Monitorizarea statutului senzorilor și dispozitivelor interioare (uși, geamuri, senzori, dispozitive electronice ș.a.m.d.);
- Monitorizarea stării emoționale a locuitorilor casei;
- Trimiterea de răspunsuri către sistem specificând starea emoțională a locuitorilor casei;
- Managementul misiunilor.

Ecranele interfețelor utilizator vor putea fi vizualizate în secțiunea de anexe a documentului.

5.2.6 Proiectarea bazei de cunoștințe

Pentru a suporta mecanismele decizionale și de raționare pentru agentul aflat la nivelul de raționamente și decizii ale arhitecturii curente, sistemul se ramifică și trebuie să integreze două sub-sisteme: un mediu persistent pentru reguli și pentru o istorie a informațiilor contextuale, precum și un sistem decizional.

Sistemul de reguli este un motor de inferență care permite integrarea mai multor tipuri de motoare de raționare și inferență, dar aplicabil pentru sistemul curent fiind doar sistemul pentru RDFS și OWL. Sistemul aplică regulile salvate în baza de reguli și monitorizează fiecare schimbare care are loc la nivelul datelor reprezentate prin ontologii. Acest modul suportă inferența a noi reguli și apoi regulile deduse sunt actualizate la nivelul modelului ontologic.

Ontologiile salvează conceptele reprezentate sub formă de perechi entitate-relație. Deoarece modelul ontologic este actualizat cu noi informații sau cu noi valori pentru informațiile deja existente, vechile valori sunt înregistrate în mediul persistent al sistemului, în asociere cu secvența de timp asociată.

Sistemul expert folosit, preluat dintr-un model, are și capacitatea să învețe care ar fi cel mai bun răspuns pe care utilizatorul îl așteaptă de la sistem. Sistemul expert este construit cu ajutorul rețelelor neuronale, care vor fi antrenate cu ajutorul unor exemple și vor fi apoi adaptate bazat pe răspunsul utilizatorului, măsurând nivelul de satisfacție în cazul îndeplinirii unor anumite acțiuni. Totuși această caracteristică este la nivel cercetare și nu intră în scopul acestei lucrări.

5.3 Detalii de implementare

5.3.1 Caracteristici generale legate de implementarea sistemului

Sistemul a fost implementat folosind limbajul Java și mediul de dezvoltare Eclipse pentru crearea proiectului.

S-a folosit mediul de dezvoltare a agenților Jadex care se poate instala ca o extensie Eclipse. După instalarea mediului în Eclipse va apărea în interfață un buton nou cu ajutorul căruia se va putea porni platforma de agenți.

Regulile de dezvoltare a agenților Jadex specifică faptul că descrierea agenților cu toate caracteristicile lor să se facă folosind fișiere de descriere XML. Descrierea unui agent BDI prin fișiere de descriere care să poată fi executat în Jadex va fi detaliată în secțiunea următoare.

Planurile specifice care ajută la îndeplinirea obiectivelor stabilite pentru agenți în particular și pentru sistemul Smart House în general sunt descrise în limbajul Java, în toate planurile extinzând clasa abstractă *Plan* și implementând metoda *body()*.

Pentru crearea ontologiilor s-a folosit limbajul OWL specificat sub formă de schemă RDF, iar editarea acestora s-a făcut folosind mediul de dezvoltare Protégé.

Pentru crearea claselor ontologice în reprezentarea Java s-a folosit o extensie Jadex a mediului Protégé numită Jadex Beanynizer care convertește clasele ontologice în clase Nuggets, toate clasele implementând interfața INuggets.

Nuggets XML este o tehnologie proprietară Jadex referitoare la serializarea și deserializarea între obiecte Java și formatul XML. Are la bază tehnologia de transformare din clase în fișiere XML și invers JibX, și are un modul de encodare, numit bean-encoder disponibil în pachetul JDK care transformă clasele Java specificate după standardul Java Beans în XML standard. Aceste XML-uri sunt trimise apoi agentului destinat din platformă care le deserializează înapoi în obiecte și trimite înapoi un răspuns.

Folosirea unelei Beanynizer facilitează crearea de clase Java după standardul Nuggets fiind ușor de folosit și creând clase de date, nemodificabile, apoi clasele propriu-zise echivalente cu clasele ontologice, care pot fi modificate pentru a adăuga noi atribute utile.

Crearea mediului vizual s-a făcut folosind unealta JformDesigner, care facilitează crearea interfețelor utilizator, programatorul mai trebuind doar să adauge funcționalitățile aferente fiecărui tip de obiect. Pentru crearea mediului vizual s-a folosit tehnologia Java Swing.

5.3.1.1 Pachetele aplicației

Toate pachetele aplicației conțin prefixul *smarthouse* următorul nume relevant fiind modulul arhitectural din care acestea fac parte. Astfel pachetul *smarthouse.robots* definește toate definițiile legate de roboții din sistemul casei inteligente. Celelalte pachete tind să respecte modelul arhitectural specificat prin componente, astfel alte pachete relevante sunt *smarthouse.sensors*, *smarthouse.devices*, *smarthouse.ontology*, *smarthouse.knowledge.base*. Ontologiile și clasele Nuggets generate sunt specificate în pachetul *smarthouse.ontology* iar partea vizuală și de monitorizare a aplicației în pachetul *smarthouse.monitoring*.

5.3.2 Implementarea agenților

Pentru a crea aplicații cu Jadex dezvoltatorul trebuie să creeze două tipuri de fișiere:

- Fișiere de definire a agenților („Agent Definition Files”, prescurtat ADF);
- Clase Java prin care se implementează planurile.

5.3.2.1 Implementarea fișierelor de descriere a agenților

Fișierele ADF de descriere a agenților pot fi privite ca o specificare a tipului unei clase de agenți care vor fi instanțiați. La momentul pornirii unui agent, în primul rând se încarcă fișierul ADF și apoi se inițializează ipotezele, planurile și obiectivele așa cum au fost ele specificate.

Antetul unui ADF este obligatoriu și are o structură bine stabilită care trebuie respectată. Un exemplu de astfel de fișier definit în proiectul curent poate fi văzut în Figura 5.7.

```

<agent xmlns="http://jadex.sourceforge.net/jadex"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                           http://jadex.sourceforge.net/jadex-0.96.xsd"
       name="MissionRobot"
       package="smarthouse.robots.world.mission">

```

Figura 5.7 Antetul fișierului de descriere a agenților ADF

În primul rând tag-ul „agent” specifică faptul că documentul XML respectă definiția de schemă *jadex-0.96.xsd* care permite verificarea faptului că documentul este un fișier valid ADF, nu doar un XML corect formulat. Numele tipului agentului este specificat prin atributul „name” și trebuie să fie același cu numele dat în definirea numelui fișierului, dar înlăturând sufixul „agent.xml”. Este de asemenea folosit ca un nume din oficiu pentru noi instanțe ale aceluiași agent când ADF este încărcat din aplicația grafică Jadex. Definiția pachetului specifică locul în care agentul trebuie să caute prima dată clasele necesare pentru planuri și ipoteze, și trebuie să corespundă cu locul în care fișierul XML este locat. În plus, pachete adiționale pot fi specificate folosind directiva *imports*.

Motorul Jadex necesită anumite proprietăți pentru inițializare, care sunt în mod implicit preluate din directorul *jadex/config/runtime.properties.xml*. În mod normal acest lucru nu este de interes pentru dezvoltatorii de agenți, dar uneori, precum este și cazul de față, a fost necesară o schimbare a comportamentului motorului Jadex.

În Figura 5.8 se prezintă schema de definire a unui agent Jadex folosind fișierul ADF, toate acestea fiind elementele care pot fi specificate în interiorul fișierului ADF. Eticheta *<imports>* este folosită pentru a specifica ce clase și pachete pot fi folosite de expresiile din fișierul ADF. Pentru a modulariza funcționalitatea agentului, agenții pot fi descompuși după capacități. Capacitățile folosite de un agent sunt referențiate cu ajutorul tag-ului *<capabilities>*. Partea esențială a specificării unui agent se referă la definirea ipotezelor, obiectivelor și planurilor agentului, care sunt plasate în tag-urile *<beliefs>*, *<goals>*, respectiv *<plans>*.

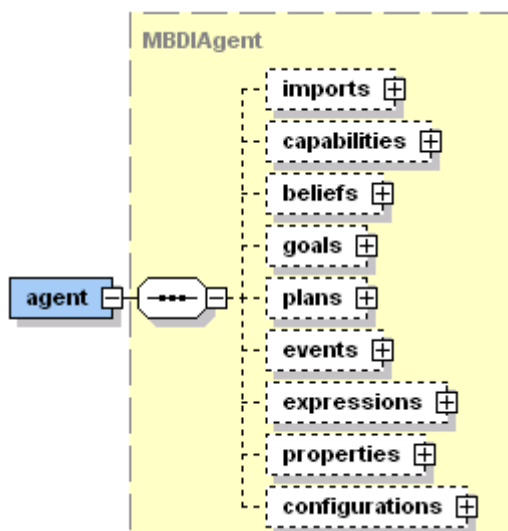


Figura 5.8 Schema de definire a unui agent Jadex

Evenimentele cunoscute de către agent sunt definite în secțiunea *<events>*. Secțiunea *<expressions>* permite specificarea expresiilor și condițiilor care pot fi folosite ca interogări predefinite pentru planuri. Secțiunea *<properties>* este folosită pentru a crea setări legate de pildă, de logarea aplicației sau de depanarea ei. În sfârșit în secțiunea de *<configurations>* configurații predefinite care conțin ipotezele, obiectivele și planurile inițiale sau obiectivele și planurile finale sunt specificate.

Este de notat faptul că ordinea aparițiilor secțiunilor mai sus este fixă, fiind exact așa cum a fost specificată în schema XML. De asemenea se pot omite acele elemente care nu sunt necesare în definirea agenților curenți.

Când fișierul ADF este încărcat, obiecte Java sunt create pentru elementele din XML(ex. ipoteze, obiective, planuri). Interfețele care specifică modelul acestor elemente sunt specificate în pachetul *jadex.model*. Exemple de astfel de interfețe sunt *IMBelief*, *IMGoal*, *IMPlan*. În cele mai multe cazuri, nu trebuie accesate aceste interfețe. Când un agent este executat, instanțe ale modelelor predefinite pentru elementele sale sunt create sub formă de elemente runtime. Acest lucru asigură ca în momentul rulării mai multe instanțe ale aceluiași model să poată fi create în mod simultan.

5.3.2.2 Implementarea planurilor

Planurile reprezintă mijloacele prin care agenții acționează în mediu. Astfel planurile compun librăria de acțiuni specifice agentului care le descrie. Depinzând de situația curentă planurile sunt selectate ca un răspuns la evenimentele sau obiectivele curente. Selecția planului care se execută se face în mod automat de către sistem și reprezintă unul din aspectele esențiale ale infrastructurii BDI. În Jadex planurile sunt împărțite în două componente: antetul planului definit în ADF și etalarea circumstanțelor în care un plan propriu-zis va fi instanțiat și executat.

Implementarea antetului planurilor în fișierul ADF

În ADF poate fi definit un număr arbitrar de planuri în funcție de specificațiile agentului. Pentru fiecare antet câteva atribute sunt necesare pentru definirea validă a planului. Pentru fiecare plan, trebuie definit atributul prin care se specifică numele planului. De asemenea pentru fiecare plan, trebuie definit corpul propriu zis al planului prin clasa Java atașată. Un exemplu de definire a planului poate fi vizualizat în Figura 5.9. Tipul planului definește ce fel de plan a fost specificat. Acesta poate fi plan standard sau plan mobil.

Directiva *<trigger>* specifică condițiile în care planul se va instanția și executa. Directiva *<parameter>* specifică parametrii de intrare pentru planul curent, dacă este nevoie de aceștia. Directiva *<contextcondition>* reprezintă condiții care trebuie validate pe tot parcursul executării planului, în caz contrar planul anulându-se.

```
<plan name="moveto">
    <parameter name="location" class="Location">
        <goalmapping ref="achievemoveto.location"/>
    </parameter>
    <body class="MoveToLocationPlan"/>

    <trigger>
        <goal ref="achievemoveto"/>
    </trigger>
    <contextcondition>$beliefbase.my_chargestate >
0</contextcondition>
</plan>
```

Figura 5.9 Definirea planului de mișcare pentru robotul transportor

Implementarea planului propriu-zis în Java

Planul propriu-zis reprezintă o parte din funcționalitatea agentului și încapsulează o suită de acțiuni. În Jadex planurile sunt scrise în limbajul Java, astfel e foarte ușoară scrierea planurilor având la dispoziție oricare dintre librăriile Java existente și de a specifica planurile în Java Integrated Development Environment (IDE), adică în mediul de dezvoltare preferat.

Conexiunea dintre planul propriu zis și antetul planului este stabilită în descrierea făcută în antetul planului, astfel planurile propriu-zise putând fi reutilizabile în diferite configurații sau planuri specificate. Pentru a spori reutilizabilitatea planurilor, este recomandată folosirea parametrilor planului, în combinație cu mapările planului pentru anumite evenimente sau obiective ale căror succes depind de execuția planului.

Orice plan extinde clasa abstractă *jadex.runtime.AbstractPlan*, în timp ce planurile standard implementează metoda *body()* așa cum a fost specificat mai sus. Metoda *body()* este executată o singură dată de-a lungul apelării unui plan, apoi planul așteptând terminarea execuției evenimentelor descrise prin trigger apelând directiva *waitFor()*.

Dacă planul se termină fără producerea vreunei excepții, este considerat ca și succes. După terminarea cu succes, eșec sau anularea planului se apelează metoda *passed()*, *failed()*, respectiv *aborted()*.

Excepția care indică o anomalie în execuția planului este *PlanFailureException* care se apare de fiecare dată când planul nu se execută cu succes.

Scrierea planurilor permite de asemenea executarea de porțiuni de cod atomice, în cazul în care nu vrem ca prin concurență să producem anomalii de citire sau scriere pentru anumite ipoteze esențiale pentru buna funcționare a agentului BDI în care este specificat planul.

5.3.3 Interfețe Utilizator

Există trei interfețe utilizator disponibile în sistemul curent, prima fiind interfața de monitorizare, cu ajutorul căreia se monitorizează și controlează senzorii și dispozitivele disponibile în casa inteligentă.

Interfața de monitorizare a robotului supervisor reprezintă o hartă 2D a casei și robotul care se plimbă pe harta casei.

Interfața de monitorizare explicită a robotului misionar reprezintă un plan simplu pe care sunt așezate obiectivele identificate de către robotul transportor, precum și robotul transportor împreună cu raza lui de acțiune.

5.4 Scenariu de testare

Scenariul de testare urmărit când s-a dezvoltat sistemul curent este:

Un cuplu de oameni căsătoriți s-au decis să înceapă să folosească sistemul Smart House în reședința lor. Ei au doi copii: Ioana, o fetiță de 14 ani, și pe micuțul Ionuț de doar 4 ani. Pentru a fi ajutați și a eficientiza muncile casnice de zi cu zi familia s-a decis să folosească 3 roboți: Locke, robotul supervisor, Sawyer, robotul transportor și pe Dolly, robotul de divertisment.

Sawyer, robotul transportor are ca scop transportul de obiecte dintr-o cameră în alta și amplasarea lor în locul de depozitare potrivit. Dolly, robotul de divertisment are ca rol supravegherea micuțului casei și menținerea unei stări de bucurie pentru băiețel. Locke este robotul supraveghetor și elaborează comenzi sau monitorizează activitatea celorlalți 2 roboți.

6 UTILIZAREA SISTEMULUI

Aplicația se pornește din mediul Eclipse sau individuală a platformei pentru agenți pentru ca ulterior să se pornească fiecare agent pe rând. Pentru a realiza acest lucru se accesează fișierele .bat existente în directorul sursă al aplicației.

Odată ce toți agenții au fost porniți cu succes vor apărea cele trei interfețe utilizator: interfața de monitorizare și control a senzorilor și dispozitivelor, interfața specifică robotului supervisor și interfața specifică robotului misiune.

Pentru a testa pornirea cu succes a senzorilor casei se verifică consola pentru agenți, pentru a vedea toți agenții porniți.

Simularea roboților începe odată cu pornirea agentului manager de roboți. Pentru ca rezultatele rulării simulatorului pentru roboți să fie edificate, este nevoie ca celelalte procese costisitoare care rulează pe mașina de test să fie oprite.

Pentru a testa adaptarea se urmărește vizualizarea grafică a mișcării robotului precum și faptul că el își îndeplinește cu succes obiectivele și anume:

- Mișcarea robotului transportor este urmărită de către robotul supervisor și mediul se actualizează de fiecare dată;
- Cutiile care intră în raza vizuală a robotului transportor vor fi transportate una câte una la locul de depozitare;
- Robotul își menține starea de patrulare prin casă atât timp cât nu îndeplinește altă misiune;
- Robotul se îndreaptă spre stația de acumulare a energiei atunci când bateria este la nivelul minim acceptat;
- La fiecare mișcare nivelul bateriei robotului scade cu 1%. Acest lucru se monitorizează prin eticheta „energy” atașată robotului;
- Dacă robotul transportă o cutie atunci acest lucru este semnalizat pe interfața grafică prin eticheta „carries box”.

În Figura 6.1 este reprezentarea robotului transportor așa cum este el vizualizat în simulatorul pentru roboți al casei inteligente.



Figura 6.1 Reprezentarea robotului transportor

Pentru a monitoriza numărul de cutii care trebuie să fie preluate de către robotul transportor se vizualizează panoul aflat în partea de jos a imaginii, care are aceeași structură ca în Figura 6.2:



Figura 6.2 Panoul de control al simulatorului

Tot cu ajutorul simulatorului se poate seta starea depozitului de cutii. În cazul în care depozitul este plin robotul este informat de sistemul casei de acest lucru, drept urmare el anulând planul de a depozita cutia în depozitul respectiv.

Pe consola agentului bazei de cunoștințe trebuie să apară anumite loguri prin care se explică ce roboți au fost găsiți în casa inteligentă, aceste informații de context fiind transmise agentului manager de roboți pentru a porni agenții roboți subsecvenți. Cu ajutorul consolei agentului bazei de cunoștințe putem vedea toate instanțele de clasă Robot create, împreună cu proprietățile aferente acestora (Figura 6.3).

```
Resource - SmarthHouse Element Kitchen1
Literal - SmarthHouse PropertyOFF
Literal - SmarthHouse PropertyGasCooker
Literal - SmarthHouse Property67446878487
Jun 13, 2009 4:24:02 PM smarthouse.knowledge.base.ContextPlan body
INFO: Plan body: ContextPlan(name=gui#12)
Jun 13, 2009 4:24:02 PM smarthouse.knowledge.base.ContextPlan body
INFO: Sender: RobotManager@biggiebig:1099/JADE
Jun 13, 2009 4:24:02 PM smarthouse.knowledge.base.ContextPlan body
INFO: Request is getRobots and model namespace is http://www.owl-ontologies.com/Ontolog
Resource - SmarthHouse Element LivingRoom_15
Literal - SmarthHouse PropertySawyer
Resource - SmarthHouse Element LittleJohn_Room
Literal - SmarthHouse PropertyDolly_PetRobot
Resource - SmarthHouse Element Hall_6
Literal - SmarthHouse PropertyLocke_MissionRobot
Jun 13, 2009 4:24:02 PM smarthouse.knowledge.base.ContextPlan body
INFO: Inside getRobots. Found 3 robots to send to RobotManager
```

Figura 6.3 Consola agentului pentru baza de cunoștințe

7 PUNERE ÎN FUNCȚIUNE ȘI REZULTATE EXPERIMENTALE

7.1 Tehnologii folosite

7.1.1 Cerințe Hardware

Sistemul dezvoltat ca urmare a lucrării curente este un sistem software, care implică utilizarea unei mașini de tip PC cu sistemul de operare Windows sau Linux. Pentru rularea aplicațiilor cerințele hardware minimale pentru mașina utilizată sunt:

- Procesor: Frecvență 1GHz, Spațiu de adresare 32-bit (x86);
- Memorie RAM: 512 MB;
- Spațiu hard-disk: minimum 1GB.

7.1.2 Cerințe Software

Produsul curent este dependent de următoarele produse software:

- Java VM Version 6 - Pentru rularea sistemului este nevoie de instalarea mașinii virtuale Java (Java Virtual Machine) pe mașina folosită. Instalarea mașinii virtuale Java poate fi făcută accesând adresa <http://www.java.com/en/download/index.jsp>;
- Platforma Jade – Platforma de agenți Jade poate fi descărcată de la adresa <http://jade.tilab.com/>;
- Jadex – Aplicația pentru agenți Jadex poate fi descărcată de la adresa http://sourceforge.net/project/showfiles.php?group_id=80240&package_id=81901;
- Eclipse Ganymede – Pentru a descărca mediul Eclipse se poate accesa pagina <http://www.eclipse.org/downloads/>.

7.1.3 Instalarea aplicațiilor folosite

Pentru punerea în funcțiune a aplicației este nevoie de instalarea extensiei Jadex pentru Eclipse Ganymede. Pentru efectuarea acestui lucru trebuie executați următorii pași:

- Instalare manuală
 - Se descarcă pachetele de la adresa <http://sra.itc.it/tools/taom4e/download.php?action=register>;
 - Se extrag pachetele din arhiva descărcată;
 - Cele 2 pachete *it.itc.sra.taom4e.model_0.5.0.jar* și *it.itc.sra.taom4e.platform_0.5.0.jar* se copiază în directorul `<dir_eclipse>/eclipse/ plugins` din Eclipse;
 - Se pornește mediul Eclipse.
- Instalare automată
 - Se pornește mediul Eclipse;
 - Se accesează meniul Help/Software Updates...
 - În panoul Available Software se apasă butonul Add Site...
 - Se adaugă <http://sra.itc.it/tools/taom4e/eu.fbk.se.taom4e.updateSite/> în spațiul destinat adresei web;
 - Se apasă OK și apoi Install.

7.1.4 Configurarea proiectului

Pentru a configura proiectul se efectuează următorii pași:

- La pornirea mediului Eclipse se configurează calea spre workspace-ul în care se află proiectul. Astfel se adaugă calea `<dir_workspace> /<nume_workspace> /workspace`. Workspace-ul Eclipse cu sursele și configurația proiectului se află în distribuția aplicației. Numele implicit este *WSBB*;
- Se configurează o nouă librărie utilizator Eclipse din meniul *Project /Properties* apoi secțiunea *Java Build Path ->Add Library...*. Se adaugă toate fișierele cu extensia `.jar` din directorul `<dir_workspace>/<nume_workspace> /workspace/SmartHouse/lib`. După adăugare se atașează această librărie la proiectul curent pentru a compila sursele.

7.1.5 Pornirea Platformei de Agenți

Punerea în funcțiune a platformei pentru agenți Jadex se poate face după ce s-a pornit Eclipse împreună cu pachetele auxiliare care necesită instalarea. Dacă instalarea a avut loc cu succes, în bara de butoane a mediului Eclipse va apărea un buton cu ajutorul căruia se poate porni platforma, locat ca în Figura 7.1:

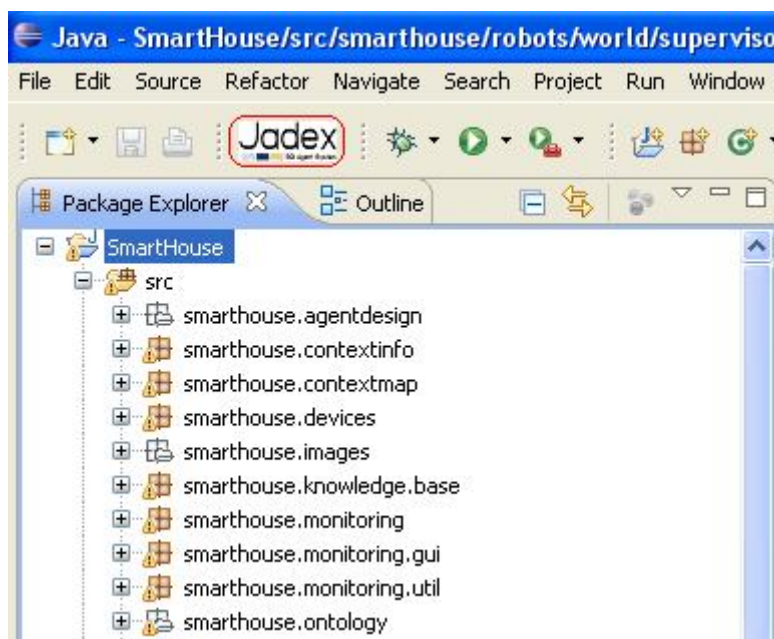


Figura 7.1 Pornirea platformei de agenți din Eclipse

Dacă platforma a pornit cu succes va apărea o nouă fereastră în care vor exista cei trei agenți master din Jadex AMS, DF și RMA. Fereastra are forma din Figura 7.2:

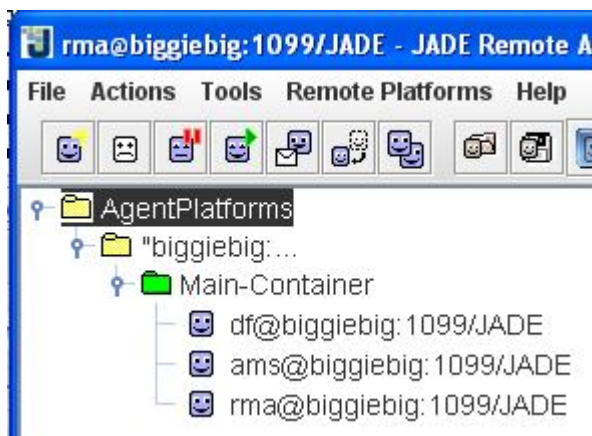


Figura 7.2 Platforma de agenți Jadex

7.1.6 Pornirea agentului pentru baza de cunoștințe a sistemului

Pentru pornirea agentului pentru baza de cunoștințe a sistemului se folosește fișierul cu extensia *.bat* numit *KnowledgeBaseAgent*. Acest fișier se află la calea `<dir_workspace>/<nume_workspace>/workspace/SmartHouse`. Pornirea acestui agent are ca rezultat înregistrarea sa în platforma de agenți. Acest agent este primul care trebuie pornit pentru ca ceilalți agenți să poată interoga acest agent în vederea extragerii informațiilor de context necesare.

7.1.7 Pornirea agentului manager de senzori

Pentru punerea în funcțiune a agentului pentru rețeaua de senzori a sistemului casei inteligente se folosește fișierul cu extensia *.bat* numit *SensorManager*. Acest fișier se află la calea `<dir_workspace>/<nume_workspace>/workspace/SmartHouse`. Pornirea acestui agent are ca rezultat înregistrarea sa în platforma de agenți Jadex. În plus, după stabilirea comunicării cu agentul bazei de cunoștințe se mai înregistrează la platformă prin instanțe specifice diferitele tipuri de senzori definiți pentru sistemul curent. Astfel prin pornirea agentului de manager a senzorilor se creează rețeaua senzorială a casei.

7.1.8 Pornirea agentului manager de dispozitive

Pentru punerea în funcțiune a agentului pentru rețeaua de dispozitive a sistemului casei inteligente se folosește fișierul cu extensia *.bat* numit *DeviceManager*. Acest fișier se află la calea `<dir_workspace>/<nume_workspace>/workspace/SmartHouse`. Pornirea acestui agent are ca rezultat înregistrarea sa în platforma de agenți Jadex. În plus după stabilirea comunicării cu agentul bazei de cunoștințe se mai înregistrează la platformă prin instanțe specifice diferitele tipuri de dispozitive definite pentru sistemul casei inteligente. Astfel prin pornirea agentului de manager a senzorilor se creează rețeaua de dispozitive a casei inteligente.

7.1.9 Pornirea agentului de monitorizare

Agentul de monitorizare se pune în funcțiune prin executarea fișierului cu extensia *.bat* numit *MonitoringAgent*. Fișierul se află la calea `<dir_workspace>/<nume_workspace>/workspace/SmartHouse`. În urma punerii în funcțiune a agentului de monitorizare, se va crea

mediul vizual cu ajutorul căruia vor putea fi monitorizate rețeaua de senzori și rețeaua de dispozitive a aplicației. De asemenea acest agent monitorizează și starea de spirit a locatarilor casei inteligente.

7.1.10 Pornirea agentului manager pentru roboți

Și pentru punerea în funcțiune a agentului manager al sistemului de roboți este pus la dispoziție un fișier cu extensia *.bat* numit *ManagerRobot*. Fișierul se află la calea `<dir_workspace> /<nume_workspace>/workspace/SmartHouse`. În urma punerii în funcțiune a acestui agent responsabil pentru roboții casei inteligente se vor înregistra după comunicarea cu baza de cunoștințe și agenții pentru roboții specifici ai casei și anume robotul supervisor, robotul transportor și robotul de divertisment. Ca urmare a instanțierii roboților specifici, se vor crea și interfețele vizuale specifice celor doi roboți prin care se poate vizualiza activitatea roboților în casa inteligentă.

7.2 Probleme întâmpinate și modul de rezolvare

7.2.1 Probleme de pornire a platformei de agenți

Dacă după pornirea platformei nu apare fereastra de control care asigură pornirea cu succes a platformei de agenți, iar pe consolă apare o excepție numită *AlreadyBoundException* atunci este cazul excepției de înregistrare a aplicației în registrul RMI, datorită faptului că portul implicit pe care se pune în funcțiune RMI și anume 1099 este ocupat.

De asemenea și o oprire forțată a platformei de agenți poate duce la ocuparea de către procesul *javaw* în stare inactivă a portului 1099.

Pentru rezolvarea problemei se distruge procesul *javaw* care ține ocupat portul 1099, folosind o aplicație de management a porturilor numită *Cports*. O altă modalitate de rezolvare a problemei este restartarea platformei Eclipse care distruge toate procesele copil create la închiderea sa.

7.2.2 Probleme de pornire a agenților

Dacă la pornirea agenților apar erori pe consolă, atunci trebuie verificat în primul rând dacă consola pentru agenți funcționează corect. Ulterior se verifică dacă toți agenții dependenți, precum agentul pentru baze de cunoștințe, funcționează corect.

Dacă eroarea de pornire nu este cauzată de unul din cazurile de mai sus atunci se editează fișierul *.bat* specific agentului, pentru a verifica dacă toate căile spre pachetele care se setează ca și Java Classpath sunt corecte. Dacă acele căi nu există se actualizează cu căile valide spre pachetele folosite la pornirea agenților.

Dacă nici acest lucru nu rezolvă problemele de punere în funcțiune, atunci se verifică dacă numele pachetelor corespunzătoare locației agenților și dacă numele agenților există.

7.3 Rezultate experimentale

7.3.1 Crearea roboților

În faza de creare a roboților după crearea instanței de agent manager pentru roboți și înregistrarea lui la platforma de agenți se face un apel la agentul bazei de cunoștințe pentru

găsirea instanțelor de agenți roboți specifici casei inteligente și crearea lor ca agenți logici pentru casa inteligentă.

Crearea agenților roboți se face printr-un apel de tip creare agent spre agentul master al platformei, AMS.

Clasele ontologice specifice roboților create în contextul ontologic al sistemului casei inteligente sunt precum în Figura 7.3:

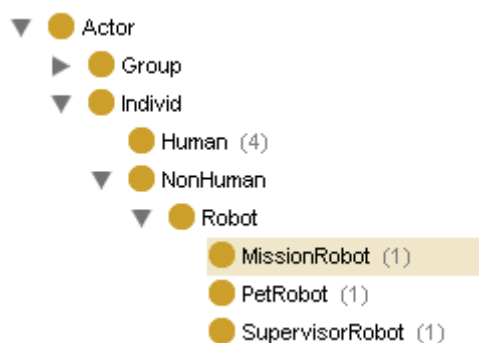


Figura 7.3 Clasele ontologice specifice agenților roboți

De asemenea fiecare robot are specificate anumite proprietăți, care vor fi luate în considerare la crearea instanțelor. Proprietățile robotului supervisor sunt legate de locație, nume, descriere, activitate, precum în Figura 7.4:

INDIVIDUAL EDITOR

For Individual: (instance of Super)

Property	Value
rdfs:comment	Sawyer is the supervisor of the robots. It monitorizes the activity of the other robots of the house

instancellame

name

hasActivity

hasLocation

Figura 7.4 Proprietăți individuale specificate în modelul ontologic - robot supervisor

Crearea efectivă a roboților se poate valida prin vizualizarea mesajelor trimise între agentul manager pentru roboți și agentul responsabil pentru baza de cunoștințe.

Agentul Manager trimite un mesaj tip ACL cu conținutul „getRobots” agentului pentru baza de cunoștințe așa cum poate fi vizualizat în Figura 7.5.

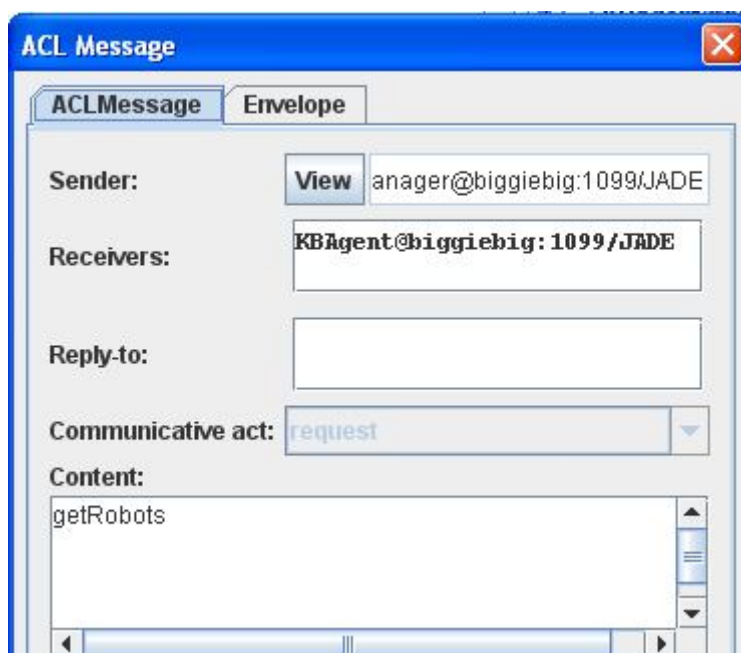


Figura 7.5 Mesajul trimis către agentul bazei de cunoștințe pentru identificarea roboților

Mesajul primit de către agentul manager de roboți este ca în Figura 7.6:

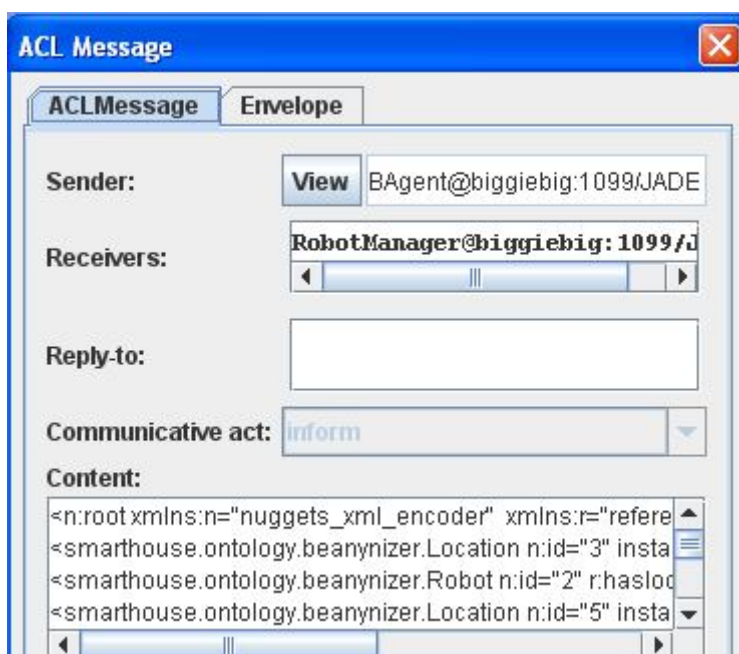


Figura 7.6 Mesajul cu conținutul instanțelor de roboți trimis de agentul bazei de cunoștințe

7.3.2 Adaptarea roboților

Odată ce instanțele de agenți roboți au fost create cu succes, începe adaptarea roboților la contextul casei inteligente. Acest lucru poate fi vizualizat interogând consola de pornire a agenților pentru a vedea planurile executate și mesajele transmise între agenții roboți și sistemul casei inteligente, precum în Figura 7.7.

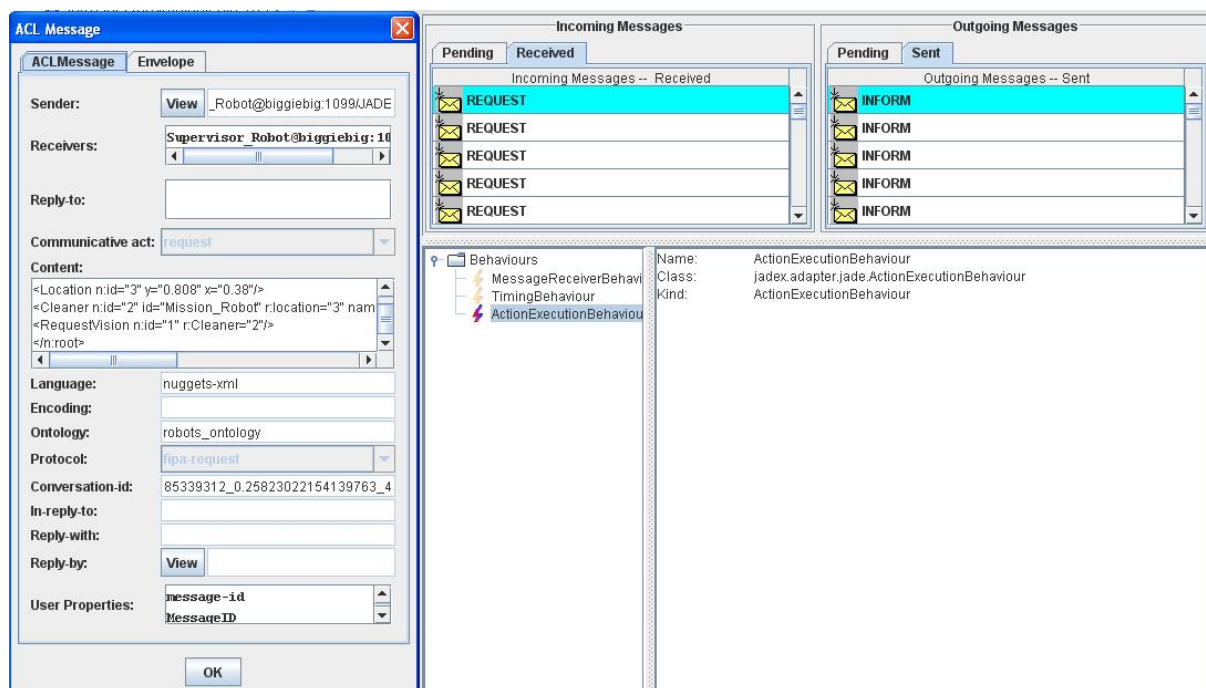


Figura 7.7 Mesaje ACL transmise între agentul robot transportor și sistemul casei inteligente

De asemenea pe consola agentului manager pentru roboți pot fi văzute toate acțiunile efectuate de către roboții din casa inteligentă, cele mai relevante informații fiind planurile care se execută la un moment dat de către aceștia, cel mai interesant de urmărit fiind planurile robotului transportor (Figura 7.8).

```

INFO: -----
Agent container Container-6@JADE-IMTP://biggiebig is ready.
-----

This is Jadex 0.96 - 2007/06/15
Using configuration: D:\WSBB\workspace\SmartHouse\bin\jadex.properties

Patrol Plan Started
Move to location plan started
Using JaninoGenerator for generating delegates.
Clean-up box plan started.
Pick up box plan started
Moving to waste?
Move to location plan started
Remote pickup waste action plan started
Move to location plan started
Pick up Box Plan started
Picked up-waste?
Patrol Plan Started
Move to location plan started
Move to location plan started
Move to location plan started
Move to location plan started
Move to location plan started
Load battery plan started
Patrol Plan Started
Move to location plan started
Move to location plan started
Move to location plan started
Move to location plan started
Patrol Plan Started
Clean-up box plan started.
Move to location plan started
Patrol Plan Started
Move to location plan started
Move to location plan started
Move to location plan started
Move to location plan started
Move to location plan started

```

Figura 7.8 Consola agentului Manager pentru roboții casei inteligente și planurile executate

8 CONCLUZII

8.1 Realizări

Cerințele lucrării exprimate în introducere au fost atinse prin proiectarea adaptării roboților la sistemul senzitiv la context pentru casa inteligentă și prin argumentarea tuturor conceptelor și metodelor folosite, care au condus la validarea proiectării cerinței de adaptare a roboților la sistemului senzitiv la context, multi-agent și distribuit.

În primele capitole ale lucrării s-a abordat problema adaptării în toată complexitatea ei prin prezentarea noțiunilor teoretice legate de conceptele folosite în scopul adaptării roboților. S-au detaliat noțiuni bine stabilite legate de domeniul senzitiv la context, domeniul ubicuu care este o aplicație practică a senzitivității la context, precum și de noțiunile de bază folosite, unele dintre ele fiind împrumutate din domeniul inteligenței artificiale, precum noțiunea de agent, sistem multi-agent, noțiuni legate de incertitudine.

S-au identificat problemele existente și provocările apărute în timpul proiectării unei aplicații senzitive la context, precum și provocările apărute în comunicarea dintre roboți și o aplicație senzitivă la context.

S-a motivat nevoia de a folosi o metodă flexibilă și extensibilă pentru a modela datele contextuale, un limbaj comun înțeles de toți participanții la arhitectură, imperios necesar pentru a putea realiza comunicarea dintre sistemul de roboți și sistemul casei inteligente. Folosirea ontologiilor facilitează de asemenea și deducerea de noi informații prin raționarea asupra datelor contextuale. Din punct de vedere a raționamentelor deduse, folosirea ontologiilor permit ierarhizarea datelor de context, în contexte de nivel superior, generale și reutilizabile, și contexte de nivel inferior, specifice, care conțin validări de consistență. Derivarea de noi contexte de nivel superior din datele de nivel inferior se poate obține folosind tehnicile împrumutate din inteligența artificială legate de algoritmi de raționare și mecanisme de inferență. După aprofundarea metodei de modelare a contextelor s-a studiat și un model derivat care adaugă suport pentru incertitudine în modelul contextual inițial, aprofundarea acestui model ajutând la rezolvarea problemelor de incertitudine deschise în proiectarea adaptării roboților la sistemul casei inteligente.

Au fost studiate și diferite abordări și metodologii de proiectare ale unui sistem senzitiv la context extensibil și flexibil, în care se pot integra foarte ușor noi componente, lucru care facilitează adaptarea rapidă a agenților roboți în structura sistemului. S-a studiat arhitectura stratificată „Context Stack”, care are la bază modelul folosit în construcția rețelelor de calculatoare și avantajele aduse de acesta.

Abordarea problemei centrale a lucrării legată de adaptarea de noi componente, în caz particular, roboți, în sistemul senzitiv la context a condus, printr-o aprofundare a metodologiilor existente pentru crearea de baze de cunoștințe și crearea sistemelor multi-agent, la rezolvarea ei prin folosirea metodologiilor existente cele mai potrivite pentru cazul de față. Astfel studiind diferite metodologii de creare a agenților și obținând o comparație din punct de vedere a standardizării, a întreținerii, a protocoalelor de comunicare folosite, a documentației disponibile s-a concluzionat că folosirea platformei Jade cu extensia Jadex este cea mai potrivită pentru atingerea obiectivelor propuse în scopul temei curente. Astfel s-a ales ca paradigmă de implementare programarea cu agenți, în care toate modulele componente ale arhitecturii reprezintă un agent cu diferite funcționalități inclusiv agenți care funcționează ca rol de interfață utilizator, cum este cazul agentului de monitorizare. Agenții folosiți sunt agenți de tip Ipoteză-Obiectiv-Plan sau **BDI (Belief-Desire-Intention)**, care pornind de la anumite ipoteze, ating obiectivele setate prin executarea de planuri de execuție.

Pentru modelarea bazei de cunoștințe folosită de sistem s-a folosit reprezentarea cunoștințelor de context prin ontologii, fiind aprofundate mai multe metodologii și limbaje de descriere a ontologiilor, precum și modul de integrare și folosire a acestora pentru comunicarea între agenții implicați în arhitectură. După studierea limbajelor în funcție de întreținere, naturalețe și ușurință a scrierii s-a ajuns la concluzia că limbajul OWL este cel mai potrivit pentru scrierea de ontologii specifice pentru sistemul senzitiv la context și pentru adaptarea ontologiilor legate de roboți cu ontologiile sistemului casei inteligente, făcându-se astfel legătura între nivelul roboți și sistemul senzitiv la context al casei inteligente. Pentru crearea și editarea de ontologii s-a folosit editorul Protégé care este întreținut și modern fiind cel mai bine dezvoltat editor pentru ontologii, beneficiind și de multe extensii vizuale, unelte pentru efectuarea de raționamente. Deoarece agenții din sistemul multi-agent creat pentru casa inteligentă comunică prin protocolul Nuggets bazat pe XML-uri, ontologiile au fost transformate în clase Java de tip Java Beans ale căror instanțe de tip obiect vor putea fi serializate în format XML și transmise prin rețea ca mesaje celorlalți agenți conectați la platformă. Astfel s-a folosit extensia Beanynizer a mediului Protégé prin care s-a făcut transformarea sub forma de unu la unu din clase ontologice în clase Java.

Având toate specificațiile formate și uneltele necesare stabilite în prealabil s-a creat o arhitectura descentralizată, bazată pe componente corespunzătoare unui anumit tip de agenți care pot comunica unul cu celălalt. Astfel componenta legată de agenții roboți s-a integrat în mod natural cu restul componentelor arhitecturii, colaborarea între diferiții agenți făcându-se prin mesaje de tip Nuggets. Astfel, prin folosirea uneltelor studiate și construirea arhitecturii s-a rezolvat deja problema principală a temei, prin simplificarea problemei la adaptarea agenților care reprezintă roboții casei inteligente la sistemul senzitiv la context al casei inteligente.

Scenariul propus pentru testarea și simularea adaptării roboților constă în realizarea robotului transportor și robotului superzivor, care au misiuni bine specificate în casa inteligentă, misiuni care sunt îndeplinite folosind informațiile de context oferite de ceilalți agenți, precum agentul responsabil pentru rețeaua senzorială. Astfel au fost specificate ipoteze, obiective și planuri pentru cei doi roboți. Robotul transportor are misiunea de a transporta cutii din camera copiilor până în locul de depozitare aflat în garaj, iar robotul supervisor supraveghează mediul casei, identifică schimbările de context din casă și comandă noi misiuni pentru robotul transportor.

Această lucrare a prezentat toate detaliile necesare pentru proiectarea, construirea și realizarea adaptării roboților prin agenți Jadex la sistemul multi-agent al casei inteligente. De asemenea arhitectura este extensibilă și scalabilă, ulterior putând fi adaptate și alte tipuri de agenți roboți, de exemplu roboții de divertisment. De asemenea arhitectura lasă loc explorării de noi domenii de cercetare din domeniul aplicațiilor ubicue și senzitive la context, pornind de la arhitectura de tip multi-agent a sistemului în conjuncție cu modelarea contextelor sub formă de ontologii și folosirea tehnicilor împrumutate din inteligența artificială precum învățarea, și procesul de luare a deciziilor, toate aceste arii de cercetare având ca obiectiv final sporirea confortului locatarilor unei case inteligente.

8.2 Direcții de dezvoltare

Dezvoltările ulterioare ale aplicației pot lua diferite direcții, dar cele mai importante vor fi subliniate în paragrafele următoare.

Un prim aspect este legat de implementarea unei aplicații complete pentru case inteligente, care implică și roboți antrenați activ în atingerea obiectivelor finale. Aceasta este o chestiune în curs de cercetare și o perspectivă bună de dezvoltare ulterioară. Astfel sistemul ar trebui să integreze componente noi, precum un detector a stării emoționale, care în

consecință ar putea valorifica misiunea robotului de divertisment, integrarea unui sistem-expert complet funcțional, care să poată folosi reguli de inferență complexe, adăugarea nivelului fizic prin care aplicația să fie valorificată, care implică și adăugarea de reguli și mecanisme de deducție pentru a procesa informațiile eronate preluate de la nivel fizic.

Un alt aspect legat de adaptarea roboților sau a altor dispozitive de tip eterogen este interoperabilitatea modelelor de context, care ar facilita adaptarea de noi roboți fără a fi necesară cunoașterea modelului de context curent, deci ar fi înlăturată rigiditatea folosirii unui singur limbaj de reprezentarea a contextelor.

Tot un aspect de luat în considerare este mobilitatea agenților roboți creați în Jadex, adaptarea acestora într-un mediu senzitiv la context nou este o muncă dificilă, acest lucru afectând instalarea aplicației într-un mediu senzitiv la context real.

Alte provocări aflate încă în faza de cercetare sunt:

- Incertitudinea – Adaptarea roboților sau a altor agenți într-un mediu incert, mediu care mapează cel mai bine lumea reală rămâne o provocare pentru cercetătorii în domeniul sistemelor senzitive la context. Nu există încă un model unificat pentru modelarea probabilistică a contextelor.
- Securitatea datelor – Mobilitatea agenților adaugă probleme complexe de securitate, deoarece posibilitatea executării de misiuni la distanță poate duce la breșe mari de securitate. Un agent logic trebuie să fie autentificat înainte să comande de exemplu un robot de la distanță.

9 BIBLIOGRAFIE

- [01] K. Goldberg, S. Gentner, and C. Sutter et al., "The Mercury Project: A Feasibility Study for Internet Robotics," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, 2000, pp. 35-40.
- [02] R. Simmons, "Xavier: An Autonomous Mobile Robot on the Web," *Proc. IEEE/RSJ Conf. on Intelligent Robots and Systems; Robots*, Victoria, B.C. Canada, Oct. 1998, URL: http://www.ri.cmu.edu/pub_files/pub1/simmons_reid_1999_1/simmons_reid_1999_1.pdf.
- [03] P. Saucy and F. Mondada, "Open Access to a Mobile Robot on the Internet," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, 2000, pp. 41-47.
- [04] M.R. Stein, "Interactive Internet Artistry," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, 2000, pp. 28-32.
- [05] M. Choi, J. Hong, and H. Ju, "XML-Based Network Management for IP Networks," *ETRI J.*, vol. 25, no. 6, Dec. 2003, pp. 445-463.
- [06] Young-Guk Ha, Joo-Chan Sohn, Young-Jo Cho, and Hyunsoo Yoon, „Towards a Ubiquitous Robotic Companion: Design and Implementation of Ubiquitous Robotic Service Framework”, Sept. 15, 2005.
- [07] D. Wang, X. Ma, and X. Dai, "Web-Based Robotic Control System with Flexible Framework," *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA 2004)*, New Orleans, LA, Apr. 2004, pp. 3351 - 3356.
- [08] W3C Recommendation, *Web Services Architecture*, W3C, 2004, URL: <http://www.w3c.org/TR/ws-arch/>.
- [09] W3C Recommendation, *RDF Primer*, W3C, 2004, URL:<http://www.w3c.org/TR/rdf-primer/>.
- [10] L. Miller, A. Seaborne, and A. Reggiori, "Three Implementations of SquishQL, a Simple RDF Query Language," *Proc. 1st Int'l Semantic Web Conf. (ISWC 2002)*, LNCS 2342, Springer-Verlag, 2002, pp. 423-435.
- [11] Z. Zenn Bien. Human-friendly Man-Machine Interaction in Smart Home. *Keynote speech at the 3rd International Conference On Smart homes and health Telematic*, Sherbrooke, Québec, Canada, July 4-6, 2005.
- [12] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen, "Gator Tech Smart House: A Programmable Pervasive Space", *IEEE Computer magazine*, March 2005, pp. 64-74.
- [13] Mohamed Ali Feki, Stéphane Renouard, Bessam Abdulrazak, Gérard Chollet, Mounir Mokhtari. Coupling Context Awareness and Multimodality in Smart Homes Concept. *The 10th International Conference on Computers Helping People with Special Needs*. Lecture

Notes in Computer Science 3118 Springer 2004, ISBN 3-540-22334-7. Paris, France, July, 2004.

[14] Kensuke Murai and Akira Okubo, "Government Policy for Next Generation Robots" in *Journal of the Robotics Society of Japan*, Vol. 26. No.5, 2008.

[15] Ghita Kouadri Most'efaoui, Jacques Pasquier-Rocha, Patrick Br'ézillon „Context-Aware Computing: A Guide for the Pervasive Computing Community, Pervasive Services”, *ICPS 2004 IEEE/ACS International Conference*, July 2004.

[16] Dey Anind K. & Gregory D. Abowd, „Towards a Better Understanding of Context and ContextAwareness”, *GVU Technical Report GIT-GVU-00-18*, GIT, 1999.

[17] Ay Feruzan, „Context Modeling and Reasoning using Ontologies”, *University of Technology Berlin*, Berlin , July 2007.

[18] Ronny Haryanto, „Context Awareness in Smart Homes to Support Independent Living”, *Master of Science in Internetworking*, University of Technology, Sydney, 2005

[19] Research Center for Educational Technology, *Ubiquitous Computing – Where it came from*, RCET, 2006, URL: <http://www.rcet.org/ubicomp/what.htm>.

[20] Boehm Barry, „Ubiquitous Computing”, *Center for Systems and Software Engineering, University of Southern California*, 2007, URL: http://sunset.usc.edu/classes/cs599_2002/Week3_c.ppt.

[21] Tao Gua, Hung Keng Punga, Da Qing Zhangb,” A service-oriented middleware for building context-aware services”, *Journal of Network and Computer Applications*, archive Volume 28 , Issue 1, January 2005.

[22] Seng Loke , „Context-aware pervasive systems: architectures for a new breed of applications”, CRC Press, 2006.

[23] T. Wahl: “Konzeption und Realisierung einer Ontologie zur Modellierung und Ableitung von Geschäftsprozessen“, TU Berlin, 2005.

[24] J. Voß: „Begriffssysteme – Ein Vergleich verschiedener Arten von Begriffssystemen und Entwurf des integrierenden Datenmodells“, HU, Berlin, 2004.

[25] M. Gruninger, and J. Lee: „Ontology – Applications and Design”, in *Communications of the ACM*, 2002.

[26] M. K. Smith, C. Welthy, and D. L. McGuinness: “OWL Web Ontology Language Guide”, URL: <http://www.w3.org/TR/owl-guide>.

[27] S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, et al.: „OWL Web Ontology Language Reference”, URL: <http://www.w3.org/TR/owl-ref>.

[28] T. R. Gruber: “Ontolingua: A Mechanism to Support Portable Ontologies”, Stanford University, 1992.

- [29] R. M. MacGregor: "Inside the LOOM description classifier", in ACM SIGART Bulletin, 1991.
- [30] T. Berners-Lee, J. Hendler, et al.: "The Semantic Web", *Scientific American*, 2001.
- [31] Bray, T., J. Paoli, et al.: Extensible Markup Language (XML) 1.0 (Third Edition) - W3C Recommendation. URL: <http://www.w3.org/TR/2004/RECxml-20040204>, 2004.
- [32] X. H. Wang, T. Gu, D. Q. Zhang, and H. K. Pung: „Ontology Based Context Modeling and Reasoning using OWL”, *Pervasive Computing and Communications Workshops*, March 2004.
- [33] Daqing Zhang, Tao Gu, Xiaohang Wang: „Enabling Context-Aware Smart Home with Semantic Web Technologies”, Institute for Infocomm Research, June 2005.
- [34] Daqing Zhang, Xiaohang Wang: „OSGi Based Service Infrastructure for Context Aware Connected Home.” Proceeding of 1st International Conference On Smart Homes and Health Telematics (ICOST2003), IOS Press, Paris, France, 2007.
- [35] T. Gu, H. K. Pung, D. Q. Zhang, "Towards an OSGi-Based Infrastructure for Context-Aware Applications in Smart Homes", *IEEE Pervasive Computing*, Vol. 3, Issue 4, 2004.
- [36] Bardram, Jakob E. The Java Context Awareness Framework (JCAF) A Service Infrastructure and Programming Framework for ContextAware Applications, Third International Conference on Pervasive Computing, vol. 3468 of Lecture Notes in Computer Science, Munich, Germany, May 2005.
- [37] Mayrhofer Rene, An Architecture for Context Prediction, PhD Thesis, University Linz, Austria, 2004.
- [38] Wooldridge, Michael and Nicholas R. Jennings (1995), "Agent Theories, Architectures, and Languages: a Survey", *Wooldridge and Jennings Eds., Intelligent Agents*, Berlin: Springer-Verlag, 1-22.
- [39] Russell, Norvig, „Artificial Intelligence: a Modern Approach”, Russell and Norvig, 1995.
- [40] Rem Collier, Gregory O'Hare, Terry Lowen, Colm Rooney, Beyond Prototyping in the Factory of Agents, *3rd Central and Eastern European Conference on Multiagent Systems (CEEMAS'03)*, Lecture Notes in Computer Science (LNCS), 2691, 2003.
- [41] Kavi Kumar Khedo, „Context-Aware Systems for Mobile and Ubiquitous Networks”, *International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, 2006.
- [42] Salber, Daniel, Anind K. Dey & Gregory D. Abowd. 1999, „The Context Toolkit: Aiding then Development of ContextEnabled Applications”, Pittsburgh, USA, May 1999.

- [43] N. Hristova, G.M.P. O'Hare & T. Lowen, „Agent-based Ubiquitous Systems: 9 Lessons Learnt, In Proceedings of the System Support for Ubiquitous Computing Workshop” , *Fifth Annual Conference on Ubiquitous Computing (UbiComp'2003) Seattle, Washington*, 2003.
- [44] Strang, T. and Linnhoff-Popien, C., „A Context Modeling Survey”, *First International Workshop on Advanced Context Modelling, Reasoning and Management*, UbiComp, 2004.
- [45] Chen, Guanling, David Kotz, „A Survey of Context-Aware Mobile Computing Research”, *Dartmouth Computer Science*, Technical Report, 2000.
- [46] Thomas Strang, Claudia Linnhoff-Popien, Korbinian Frank, „CoOL: A Context Ontology Language to enable Contextual Interoperability”, *Distributed Applications and Interoperable Systems (DAIS2003)*, 2003.
- [47] M. K. Smith, C. Welthy, and D. L. McGuinness: “OWL Web Ontology Language Guide”, URL: <http://www.w3.org/TR/owl-guide>.
- [48] Binh An Truong, Young Koo Lee, Sung Young Lee, „A Unified Context Model: Bringing Probabilistic Models to Context Ontology”, *EUC Workshops*, 2005.
- [49] J. Pearl, "Belief Networks Revisited", *Artificial intelligence in perspective*, p. 49-56, 1994.
- [50] Gregory D. Abowd and Anind K. Dey, "Towards a Better Understanding of Context and Context-Awareness", *Workshop on the what, who, where, when and how of context-awareness at CHI 2000*, Aprilie 2000.
- [51] Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A., „Methodologies, tools, and languages for building ontologies. Where is their meeting point?”, *Data and Knowledge engineering*, 2003.
- [52] D.B. Lenat, R.V. Guha, „Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project”, *Addison-Wesley*, Boston, 1990.
- [53] A. Bernaras, I. Laresgoiti, J. Corera, „Building and reusing ontologies for electrical network applications”, *Proc. European Conference on Artificial Intelligence (ECAI_96)*, Budapest, Hungary, 1996, pp. 298–302.
- [54] B. Swartout, P. Ramesh, K. Knight, T. Russ, „Toward Distributed Use of Large-Scale Ontologies”, *AAAI Symposium on Ontological Engineering*, Stanford, 1997.
- [55] M. Fernandez-Lopez, A. Gomez-Perez, A. Pazos-Sierra, J. Pazos-Sierra, „Building a chemical ontology using METHONTOLOGY and the ontology design environment”, *IEEE Intelligent Systems & their applications*, 1999, p. 37–46.
- [56] Nguyen G., Dang T.T, Hluchy L., Laclavik M., Balogh Z., Budinska I. Agent platform evaluation and comparison, Jun 2002.
- [57] FIPA standards and specification, FIPA, *IEEE Computer Society Standards for Agent Based Technologies*, <http://www.fipa.org/specifications/index.html>.

- [58] FIPA-OS, Agent Platform Development Tool, *IEEE Computer Society Standards for Agent Based Technologies*, <http://fipa-os.sourceforge.net/>.
- [59] ***, „JADE, Tutorial for beginners”, *JADE Board*, <http://jade.tilab.com>.
- [60] ***, JACK, <http://www.agent-software.com/>.
- [61] ***, „Zeus – BT Intelligent Agent Research”, <http://www.opensource.org/>.
- [62] Lars Braubach, „Jadex User Guide”, *Distributed Systems Group University of Hamburg, Germany*, <http://prdownloads.sourceforge.net/jadex/userguide-0.96.pdf>.
- [63] Lars Braubach, „Jadex Tool Guide”, *Distributed Systems Group University of Hamburg, Germany*, <http://prdownloads.sourceforge.net/jadex/toolguide-0.96.pdf>.
- [64] ***, W3C Recommendation, SOAP Version 1.2 Primer, W3C, 2003, URL: <http://www.w3c.org/TR/%20soap12-part0/>.
- [65] N.F. Noy, R.W. Fergerson, M.A. Musen, „The knowledge model of protege-2000: combining interoperability and flexibility”, *12th International Conference in Knowledge Engineering and Knowledge Management (EKA 2000), Lecture Notes in Artificial Intelligence, vol. 1937*, Springer, Berlin, 2000, pp. 17–32.
- [66] J.C. Arpirez, O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez, „WebODE: a scalable ontological engineering workbench”, *First International Conference on Knowledge Capture (KCAP01)*, ACM Press, Victoria, 2001, pp.6–13.