CONSERT

Platformă de management a contextului pentru aplicații de inteligență ambientală

Cod proiect PN-III-P2-2.1-PED-2016-1753
Număr contract 34PED

Etapa I

Proiectarea modelului CONSERT extins. Dezvoltarea platformei CONSERT extinsa, inclusiv IDE

Această etapă descrie prima iterație a unui modul funcțional (Motorul de inferența CONSERT modularizat), funcționalitatea imbunatatita a motorului de inferența CONSERT, funcționalitatea si indicații de utilizare a mediului de dezvoltare (IDE) CONSERT, opțiunile imbunatatite de deployment ale platformei CONSERT si mecanismele de detecție automata a schimbării de context.

Cuprins

1. Proiectul CONSERT - Motivație si Obiective

Inteligența Ambientală (AmI) este astăzi destul de matură pentru a intra în atenția industriei și a directivelor de susținere la nivel european.
Un subdomeniu de bază al AmI este cel al gestiunii contextului și calculul sensibil la context (furnizarea către aplicații și utilizatori a informației potrivite, la momentul potrivit și în modul potrivit). Gestiunea Contextului acoperă multe subiecte de cercetare de la reprezentarea și raționamentul asupra informației și până la arhitecturi de sisteme informatice. O separare a nevoilor și obiectivelor între funcționalitatea de bază a unei aplicații și adaptarea ei contextuală este necesară.

Astfel, obiectivul acestui proiect este **crearea unui middleware open source pentru gestiunea contextului** care să furnizeze opțiuni bogate ce înlesnesc dezvoltarea de aplicații sensibile la context.
Plecând de la dezvoltări create in cadrul laboratorului AIMAS al UPB, proiectul CONSERT isi propune următoarele:

- creșterea performanței și modularizarea motorului de inferență pentru a susține aplicații cu diferite nivele de complexitate,
- crearea unui mediu integrat de dezvoltare (IDE) pentru modelarea contextului spre facilitarea dezvoltării de aplicații contextuale pe baza CONSERT Middleware
- dezvoltarea unor opțiuni de implementare (deployment) a unităților de lucru din CONSERT potrivite pentru aplicații din domeniul IoT (folosind OSGi și Docker pentru a aborda schimbarea dinamică a contextului într-o aplicație)
- dezvoltarea a două aplicații de test: Smart Campus Support pentru monitorizarea activităților studenților și profesorilor într-un campus universitar și Smart Elderly User Support pentru configurarea dinamică adecvată a serviciilor oferite de un mediu de inteligență ambientală care asista utilizatorii in vârsta acasă.

In acest raport se prezintă progresul înregistrat pentru primele trei obiective din cele prezentate mai înainte.
Secțiunea 2 prezintă motorul de inferența CONSERT si imbunatatirile aduse acestuia. Secțiunea 3 descrie middleware-ul CONSERT in ansamblu si progresul de reimplementare a acestuia sub forma de aplicație IoT. Secțiunea 4 arata dezvoltarea IDE-ului CONSERT, iar secțiunea 5 concluzionează raportul.

2. Motorul CONSERT - Imbunatatiri si mod de utilizare

Motorul de inferența CONSERT reprezintă componenta de raționament asupra informației contextuale in middleware-ul CONSERT. Funcționalitatea acestuia este similara celei unui motor de recunoaștere de evenimente complexe (semantic complex event processing - SCEP), dar are elemente distincte inovatoare ce îl fac potrivit pentru tipul de inferențe des întâlnite in aplicații de inteligenta ambientala. Aceste elemente si modul lor de întrebuințare sunt prezentate in continuare.

2.1 Functionalitatile motorului de inferența CONSERT si modul sau de operare

Motorul de inferența CONSERT se bazează pe reprezentarea informației contextuale propusa in [Sorici et al, 2015b]. Pentru metamodelul CONSERT exista atât o implementare sub forma de ontologie, cat si una obiectuala (clase in limbajul de programare JAVA), cele doua fiind transformabile una intr-alta.
Forma ontologica este folosita pentru comunicare: primirea de evenimente de la componente senzoriale si interogări de la aplicații finale pentru situațiile contextuale inferate (cf. Secțiunea 3).

Forma obiectuala este folosita intern pentru efectuarea raționamentelor de către motorul CONSERT (cf. Secțiunea 2.3).

Funcționalitatea motorului si modul sau de operare sunt descrise in detaliu in [Sorici et al, 2015b]. Acest raport prezintă imbunatarile care au fost aduse functionalitatii acestuia si privesc următoarele aspecte:

- Inserarea si tratarea evenimentelor in funcție de tipul lor de achiziție (informație statica, de la senzori, direct din partea unei aplicații finale sau inferata de motor)
- Aplicarea inferențelor pe baza unui sistem de reguli implementat cu ajutorul framework-ului DROOLS (cf. Secțiunea 2.2)
- Prelucrarea explicita a evenimentelor ce admit adnotări de durata temporala. In particular, motorul CONSERT implementează un mecanism de extensie a validității temporale a unei situații contextuale pe baza evenimentelor atomice (cf. Secțiunea 2.3).
- Aplicarea de operatori particulari pentru a deduce in mod automat noi valori pentru adnotările care sunt combinate sau extinse in timpul inferențelor (e.g. grad de încredere, timestamp)

2.2 Framework-ul DROOLS ca tehnologie la baza motorului CONSERT

Drools [Drools, 2017 a] este un sistem de management pentru reguli de business si include un motor de reguli de business, o aplicație de management al regulilor si o extensie pentru interfața aplicației Eclipse folosita pentru dezvoltare. Motorul de reguli pe care se bazează Drools, numit PHREAK [Drools, 2017b] este de fapt o evoluție a clasicului algoritmul RETE [Forgy 1982], folosit (in diverse variante) de aproape toate sistemele expert bazate pe reguli care se afla astăzi in producție. PHREAK, spre deosebire de Rete, aduce îmbunătățiri precum evaluarea întârziată a regulilor, propagare orientată pe set-uri și segmentarea rețelei de noduri pe care se bazează rețeaua. Aceste îmbunătățiri cresc eficiența algoritmului față de varianta clasică, dar poate mai mult decât atât ele permit evaluarea în paralel a regulilor, ceea ce oferă reale posibilități de reducere a latenței de execuție.

Suita de aplicații și platforme asociate cu Drools sunt printre cele mai folosite sisteme de business, unde este nevoie fie de fluxuri operaționale sau de sisteme expert. Nu doar eficiența sa este recunoscută ci și faptul că poate fi relativ simplu de implementat în producție, alături de sisteme și aplicații deja existente într-o anume infrastructură (plecând de la aplicații simple și până la unele complexe, de mare anvergura).
Din punct de vedere al ușurinței cu care Drools poate fi pus în producție, chiar și pe o varietate mai mare de sisteme, faptul că este implementat folosind limbajul *Java* și că permite împachetarea motorului de reguli (cu orice aplicații auxiliare ar fi necesare) într-un container de tip *Docker* reprezintă un real avantaj. Astfel, dezvoltatorii de aplicații pot lansa și folosi o aplicație integrată cu Drools atât în cazul în care au resurse limitate cât și în cazurile în care aplicația lor scalează către o utilizare de mari proporții.
Pe lângă motorul de reguli implementat cu algoritmi de inferență eficienți, Drools permite funcționarea în două moduri:

- *Cloud* - în acest mod, regulile sunt evaluate plecând de la o bază de cunoștințe în care faptele conținute reprezintă „starea” sistemului la un anumit moment în timp
- *Streaming* - în acest mod, faptele se pot introduce treptat, ele având și o caracteristică temporală ce poate fi utilizată de reguli (folosind, de exemplu, principii de logică temporală)

Atunci când Drools funcționează în modul *streaming* el poate fi folosit pentru a sta la baza unui motor de recunoaștere de evenimente complexe. Acesta funcționează evaluând nu fapte, ci evenimente, care pot fi punctuale (apar la un moment de timp și au o durată de viață instantanee) sau bazate pe intervale (sunt definite prin momentele de activare și cel de dezactivare). Această distincție este folositoare și în cazul nostru, în special pentru că ne

permite să implementăm cu ușurință o parte dintre meta-informațiile de care middleware-ul CONSERT își propune să le folosească.
Bazându-se pe logica temporală, Drools implementează și operatori temporali necesari pentru tratarea datelor de acest tip. Spre exemplu, pentru două evenimente *A* (valid între t_start_A și t_end_A) și B (valid între t_start_B și t_end_B) operatorul temporal *before* (înainte de) dintr-o regulă de tipul *A before B* verifică dacă $t\_end_A < t\_start_B$, adică evenimentul *A* s-a terminat înainte ca evenimentul *B* să înceapă.
Mai mult decât atât, motorul Drools dispune de posibilitatea de a grupa și analiza evenimentele înregistrate în sistem. Acesta poate număra câte instanțe din același tip de eveniment au fost înregistrate sau poate folosi o fereastră temporală glisantă care să considere active și să ia în considerare doar evenimentele care sunt în interiorul ferestrei. De exemplu, ar putea exista o regulă care să se activeze atunci când 2 evenimente de tipul *A* au fost înregistrate. În modul numărare, aceasta se va activa oricând acest lucru se întâmplă. Folosind însă ferestre glisante, dacă dimensiunea ferestrei este, spre exemplu, de 10 secunde, dar evenimentele de tip *A* au apărut la 15 secunde unul de celălalt, atunci regula nu se va activa.

2.3 Funcționalitatea motorului CONSERT

Deși framework-ul Drools este robust, este necesar de luat în calcul faptul că specificațiile middleware-ului CONSERT impun o serie de modificări și extinderi ale acestuia pentru a putea fi integrat cu succes în proiect. Mecanismele cu care Drools trebuie extins pot fi identificate analizând în primul rând ciclul de inferență care trebuie implementat în CONSERT.
În primul rând, orice eveniment din sistem care trebuie procesat este construit prin împachetare folosind datele și meta-datele acestuia. Apoi, el este prezentat motorului de inferență, folosind fluxul de evenimente corespunzător evenimentului respectiv (numite *Entry Points* - puncte de intrare).

Înainte de a fi preluat de modulul de inferență și pasat rețelei de noduri a algoritmului *PHREAK* (mai specific, nodurilor care filtrează tipurile de obiecte - *Object Type Nodes*), evenimentul trece printr-o *etapă de validare și verificare a adnotărilor*. În această etapă în funcție de tipul evenimentului, se decide dacă el va fi trimis către un flux în care se realizează extinderea unui eveniment precedent de același tip sau dacă se va introduce ca fiind un eveniment nou. De exemplu, un eveniment de tip A, poate avea un interval maxim de extindere de 5 secunde. Acest interval specifică intervalul în care se poate prelua un nou eveniment de tip A astfel încât ambele să fie considerate parte a aceluiași eveniment. Se poate considera că primul eveniment a determinat startul unui eveniment extins de tip A, iar toate celelalte evenimente de tip A care se succed, fiecare la mai puțin de 5 secunde de precedentul sunt de fapt reconfirmări pentru evenimentul extins A. Orice eveniment de tip A care va fi introdus în sistem la mai mult de 5 secunde de un eveniment A precedent nu va fi folosit pentru extindere, ci va reprezenta un nou potențial start pentru un eveniment extins de tip A. Desigur, există și evenimente pentru care mecanismul de extindere nu este necesar și nu se aplică, având fiecare eveniment inserat considerat drept o instanță unică.

Tot în această etapă sunt verificate și validitatea valorilor evenimentului. De exemplul, dacă tipul unui eveniment descrie o informație al cărei domeniu este reprezentat de mulțimea numerelor naturale, vor fi respinse (și neinserate) acele instanțe de evenimente care conțin numere negative, fracționare, etc. De asemenea, un domeniu poate fi limitat la un set sau interval de valori, iar evenimentele care au valori ce nu corespund domeniului vor fi refuzate. Această filtrare este necesară întrucât există numeroase surse de erori ale căror defecte nu se doresc a fi propagate motorului de inferență (e.g. senzori defecți, măsurători inexacte, comunicație defectuoasă).

Ulterior verificării dacă evenimentul trebuie adăugat unei extinderi (sau nu) și validarea valorilor pentru adnotări, middleware-ul CONSERT trebuie să verifice constrângeri de consecvență specifice evenimentului. Mai exact, aceste constrângeri definesc situații de conflict în baza de cunoștințe a evenimentelor, specificând totodată și metodele de rezolvare

a acestor conflicte. Sursele de erori menționate anterior pot genera evenimente care luate fiecare în parte au valori valide, însă care împreună nu pot coexista. Spre exemplu, să ne imaginăm un tip de eveniment care indică poziția în spațiu a unei persoane. Erori ale senzorilor de poziționare pot duce la generarea a doua evenimente de acest tip (temporal foarte apropiate unul de celălalt), unul care să plaseze persoana în interiorul unei încăperi iar celălalt raportând o poziție în afara încăperii. Este evident că aceeași persoană nu poate fi în două locuri în același timp, prin urmare cele două evenimente sunt marcate ca fiind în conflict. Pentru rezolvarea acestuia CONSERT se bazează pe mecanisme explicite descrise de integratorul middleware-ului. În cazul exemplului de mai sus există mai multe posibilități de aplanare. Se poate alege, de exemplu, instanța de eveniment cu (meta-)valoarea de încredere (*confidence*) mai mare, dacă aceasta există, sau se poate alege instanța de eveniment generată ultima în ordine temporală, ori se poate chiar realiza o interpolare ale celor două poziții folosind valorile de încredere drept ponderi pentru o medie ponderată. Desigur, există și alte metode de rezolvare a conflictelor iar cele descrise anterior nu acoperă exhaustiv posibilitățile.
Constrângerile din exemplul de mai sus pot fi considerate, într-o anume măsură, drept metode de garantare a unicității. Evenimentele care intră în sistem nu trebuie să se contrazică unul pe celălalt (starea conflictuală fiind detectată prin mecanisme explicite). Mai mult decât atât, ciclul de inferență poate genera la rândul său noi cunoștințe care să fie reintroduse în sistem. Întrucât și acestea pot genera conflicte, similar cu exemplul descris mai sus, ele vor trece, la rândul lor, prin etapele de validare ale mecanismelor de verificare a constrângerilor și rezolvarea conflictelor.

După ce evenimentele sunt validate dar înainte să fie inserate, este necesară realizarea etapei de inferență ontologică. În cadrul platformei CONSERT, evenimentele vor putea fi definite și folosind o ontologie. Aceasta poate fi folosită pentru a exprima evenimentele ca entități de diverse tipuri și cu diverse proprietăți, între care există relații de subsumare (sau alte tipuri). Folosind aceste relații, utilizarea unei ontologii permite stabilirea gradului de granularitate al cunoștințelor dorit. De exemplu, un eveniment de tip B poate fi subsumat unui eveniment de tip A. La primirea unui eveniment de tip B, prin inferențe ontologice, se poate alege ca în baza de cunoștințe să fie introdus de fapt un eveniment de tip A, chiar dacă acesta este mai puțin specific.

Odată parcurși pașii de mai sus (validarea valorilor, verificarea constrângerilor și realizarea inferențelor ontologice) evenimentul este în cele din urmă introdus în motorul de reguli de inferență. Acesta va genera acțiunile necesare sau va produce noi cunoștințe bazându-se pe nodurile terminale ale rețelei de inferență. Este important de notat faptul că orice eveniment inferat este reintrodus în sistem, completând astfel ciclul de inferență. El va fi tratat asemenea unui eveniment nou introdus și va fi trecut prin aceleași etape descrise mai sus.
Deși mecanismele prezentate în această secțiune sunt descrise cu un grad mare de generalitate, în secțiunile următoare, odată cu definirea tipurilor de adnotări alese în contextul CONSERT vor decurge natural și soluțiile de implementare pentru mecanismele de validare și verificare.

Operatorii de prelucrare a adnotărilor in CONSERT

În motorul CONSERT există un modul care se ocupă special de adnotarea datelor şi de operatori particulari pentru extensie temporală. Pentru a înţelege mai uşor utilitatea acestui modul, se introduce următorul scenariu - deducția simpla a activității intr-un laborator de cercetare (HLA Scenario).

Se considera cazul unor senzori care oferă date legate de poziţia unor persoane, respectiv de activitatea lor – stau în picioare, stau pe scaun, se află în mişcare.
Senzorii oferă informaţii într-un interval de timp prestabilit (e.g. o data la 5 secunde), acestea devenind evenimente atomice. Pentru fiecare eveniment senzorul este capabil să aserteze gradul de încredere în informația furnizată şi timpul la care este trimisă informaţia respectivă. Scopul scenariului este de a avea nişte intervale de timp bine definite pentru fiecare activitate (persoana X se află mișcă în zona de exerciții a laboratorului între t1 şi t2),

Întrucât informația de bază de care dispune aplicația este dată de evenimente atomice (e.g. senzor de mișcare activat la t1, postură a persoanei identificată la t2), este necesar un mecanism prin care sunt extinse metadatele oferite de senzori, prelungind valabilitatea temporală pentru un eveniment.

Frameworkul DROOLS posedă capacitatea de a gestiona evenimente cu metadate privind aspecte temporale doar dacă acestea nu suferă modificări în timp (i.e. durata temporala rămâne setată la momentul introducerii în sistem).
Modul de operare în motorul CONSERT diferă însă, astfel încât, pentru a oferi o mai mare flexibilitate sistemului şi a putea manevra datele în modul dorit, este nevoie de implementarea unor operatori particulari.
Aceștia acoperă funcționalități precum:

- accesul la informaţii legate de adnotări
- operații asupra adnotărilor: e.g. intersecția a două intervale de timp
- verificarea posibilității de extensie a unei aserțiuni contextuale (e.g. se va prelungi situația de a fi localizat într-o zonă a laboratorului, *doar daca* senzorul de mișcare se declanșează cu acuratețe de peste 50%).
- operații de calcul a valorii pentru adnotarea unei aserțiuni derivate, pe baza valorilor aserțiunilor din premise

Modelul CONSERT permite la nivel teoretic definirea oricărui tip de adnotare și clasificarea lor în *simple* (de natură statică, e.g. sursa informației - senzorul care a produs-o) sau *structurate* (de natură dinamică, ce pot fi combinate și modificate prin operatorii menționați anterior).
În mod comun, însă, în aplicațiile de AmI se întâlnesc următoarele trei tipuri de adnotări structurate:

- *Gradul de încredere*: Pentru a putea extinde un eveniment atomic, trebuie ca gradul său de încredere să nu difere foarte mult de încrederea evenimentului pe care îl extinde (e.g. dacă se știe cu o probabilitate de 80% că persoana X se află în zone de lucru, iar apoi încrederea scade la 40%, cel mai probabil persoana se află în mişcare, eventual a şi părăsit zona, deci cele două evenimente nu sunt corelate). De asemenea, evenimentele trebuie să aibă o anumită încredere limită, atât pentru a fi inserate, cât şi pentru a extinde un alt eveniment.
- *Timestamp-ul ultimei actualizări* pe care o avem de la un senzor. Daca două actualizări sunt foarte depărtate în timp, ele descriu foarte probabil activări ale senzorului care *nu* privesc aceeași situație, neducând astfel la posibilitatea extensiei temporale a aceluiași eveniment.
- *Intervalul de valabilitate al unui eveniment* – este elementul care descrie persistența în timp a unei situații. Timestamp-ul ultimei actualizări și gradul de încredere sunt definite în raport cu un interval de validitate. Ele își schimbă valoarea pe măsură ce intervalul de validitate este prelungit. Operatorii particulari definiți anterior sunt folosiți pentru actualizarea acestor tipuri de adnotări atunci când validitatea temporală este extinsă.

În momentul inserării unei noi aserțiuni se creează un eveniment atomic care conține informațiile pe care le oferă senzorii. În funcție de tipul de adnotări asociate evenimentului, se instanţiază clase corespunzătoare, care sunt adăugate aserțiunii.
Pentru fiecare tip de adnotare sunt definite instanțe ale operatorilor de extensie și combinare, precum si a celor ce verifică dacă, *din perspectiva adnotării analizate,* evenimentul atomic căruia îi este atașată poate extinde o situație existentă.
Pentru ca extensia sau combinarea să poată fi validată, este necesar ca toate condițiile pentru extensie sau combinare din fiecare adnotare individuală să fie satisfăcute. Dacă avem doar o adnotare de un anumit tip pentru un eveniment (de exemplu lipsește gradul de încredere), atunci putem ignora câmpul respectiv (sau, dacă vrem o variantă mai strictă, se poate considera că cele două evenimente nu pot fi suprapuse pentru a forma un eveniment extins).

Pe lângă operațiile de verificare a continuității şi actualizare a adnotărilor în urma unei extensii temporale sunt definiți operatori particulari care permit manipularea intervalelor temporale. Sunt implementate funcții pentru:

- Secvență de intervale
- Suprapunere
- Intersecție
- Incluziune si conținere (un interval conține o dată anume)

De asemenea, sunt implementate funcții ce permit setarea diverselor câmpuri din adnotările individuale, respectiv obţinerea valorilor care se află acolo, având acces uşor la adnotările de fiecare tip. Acest lucru este util dacă se dorește modificarea anumitor câmpuri (de exemplu încrederea pentru o aserțiune) sau dacă este nevoie de diverse detalii legate de un eveniment (cum ar fi timpul în care este valabilă aserțiunea respectivă).

Operatorii particulari de operare asupra adnotărilor sunt așadar folosiți pentru a accesa/seta/modifica valorile fiecărei adnotări în cadrul regulilor de inferență, cât și pentru a opera validarea inserării, extinderea și combinarea evenimentelor atomice din perspectiva meta-proprietăților acestora (i.e. a adnotărilor).

2.4 Întrebuințarea motorului de inferența CONSERT

Dată fiind implementarea motorului CONSERT utilizând framework-ul DROOLS, s-a urmărit analiza performanței sistemului. Au fost urmărite următoarele tipuri de analiză:

- *Viteză de prelucrare a unui eveniment atomic*: considerând că mecanismul de extensie a validității temporale este activat (pe măsură ce se primesc noi informații de la senzori, activitățile sunt prelungite, pentru a nu avea 10 activități atomice în care persoana X se află la bucătărie, de exemplu, ci o singură activitate, care se întinde pe toată durata celor 10 evenimente atomice), se dorește investigarea întârzierii ce apare de la inserarea unui eveniment în sistem până la procesarea acestuia
- *Viteză de execuție a regulii de inferență*: este analizat timpul care apare între inserarea unui aserțiuni necesare pentru a activa o regulă de procesare a acesteia, până la inserarea aserțiunii derivate din aceasta.

O analiză de ansamblu se uită în plus la throughput-ul motorului (i.e. câte evenimente pot fi procesate într-o secundă).

Ca orice sistem care oferă anumite metode de raționament pornind de la anumite informații, şi CONSERT trebuie testat, pentru a şti cât de eficient este în raport cu alte sisteme şi pentru a şti dacă sistemul este unul care scalează în cazul unor scenarii mai mari, cu mult mai multe date.
Este de reținut faptul că testele s-au operat într-un scenariu în care toate evenimentele simulate sunt citite instant. În realitate, la un anumit moment de timp nu se vor înregistra mai mult de câteva informații venite de la senzori pentru o anumită persoană, așa că, teoretic, un sistem în care se pot procesa câteva zeci sau sute de evenimente pe secundă este suficient pentru aplicații precum o casă inteligentă sau un spațiu de lucru inteligent.

Scenariul pentru care au fost efectuate testele de performanță este cel în care există persoane care fac diverse activități. Există date care dau localizarea unei persoane, respectiv activitatea de tip ”low level activity” (LLA) pe care acesta o face – merge, stă în picioare, stă pe scaun. Din aceste activități, se pot deduce altele mai complexe, denumite ”high level activities” (HLA).
Trebuie avute în vedere două tipuri de întârzieri – cele cauzate de extinderea unei activități mai puțin complexe, respectiv cele care sunt cauzate de extinderea unei activități mai complexe, de tip HLA. Activitățile din a doua categorie se obțin direct din aplicarea de reguli de derivare a contextului, exprimate folosind DROOLS. Pentru fiecare HLA există anumite

reguli care sunt aplicate de câte ori este posibil. De fiecare dată când se aplică o nouă regulă, se încearcă extinderea unei instanțe de activitate veche, dacă este posibil, de aici apărând diverse întârzieri în scenariul de test.

De asemenea, este important de știut câte activități se pot procesa, de exemplu, pe durata unei secunde, atât activități de nivel înalt, cât și de nivel scăzut. Aceste statistici sunt importante pentru a putea verifica fiabilitatea sistemului. Pentru a vedea timpul efectiv, s-a împărțit timpul total de rulare, din momentul în care sistemul începe să introducă reguli, până la finalizarea programului, și s-au obținut statistici legate de numărul de evenimente care pot fi procesate într-o secundă de către sistem.

Se prezintă pentru început analiza de throughput. O execuție a sistemului durează, în medie, 2 secunde, pentru scenariul cu activități care a fost prezentat anterior. În cadrul acestui eveniment, s-au procesat 689 de activități de tip LLA (incluzând aici toate extensiile în timp ale activităților atomice oferite de datele de la senzori), respectiv 276 de activități de tip HLA, rezultând un total de 965 de activități.
De aici rezultă aproximativ 1000 de activități procesate în 2 secunde, deci sistemul poate procesa aproximativ 500 de activități pe secundă, sau o activitate la 2 milisecunde.
Sistemul scalează liniar, dacă se dublează numărul de activități se dublează și timpul de procesare, activitățile fiind procesate în timp independent de numărul lor, astfel încât numărul lor va afecta complexitatea totală liniar.

În medie, timpul de procesare pentru o activitate este de 2 milisecunde.
Pe scenariul analizat întârzierile de prelucrare a unui eveniment atomic sunt între 1 ms și 2ms. Excepție face startul scenariului, unde inițializarea structurilor interne motorului CONSERT cauzează întârzieri de procesare de până la 14 ms.

Pentru activitățile de HLA, întârzierile sunt constante, timpul de procesare a regulilor fiind în medie de 2 ms.
Acest lucru se datorează faptului că evenimentele de tip HLA apar în urma inserării celor atomice și ulterior perioadei de inițializare a motorului CONSERT, astfel că timpul pentru procesarea lor rămâne constant, neexistând alți factori care să afecteze complexitatea totală.

Experimentele au fost făcute pe un procesor i7 cu 3.4 GHz și 8GB memorie, rezultatele vor diferi printr-o constantă în funcție de procesorul pe care se rulează, dar ordinul de mărime va rămâne același.

3. Middleware-ul CONSERT - reimplementare si modalitati de deployment

Motorul de inferența descris in secțiunea anterioara face parte din efortul de a dezvolta un întreg sistem informatic (un middleware) dedicat gestiunii informatei contextuale in aplicații pentru scenarii de IoT sau Inteligenta Ambientala (AmI).
Proiectul CONSERT urmărește astfel dezvoltarea unor module informatice care, prin funcționalitatea lor, pot deservi întreg ciclul de lucru al unei aplicații sensibile la context (percepție -> modelare/inferența -> diseminare/consum).

3.1 Functionalitatile middleware-ului CONSERT si descrierea unitatilor logice componente

Middleware-ul CONSERT este gândit in jurul unor unitati logice (CONSERT Middleware Agents) [Sorici et al., 2015a] a căror funcționalitate se focalizează pe un singur aspect al

ciclului de furnizare (eng. provisioning) a informație contextuale. Aceste unitati sunt următoarele:

- CtxSensorAgent - unitate responsabila de preluarea datelor de la senzori (fie reali, fie virtuali - e.g. un mesaj pe o rețea de socializare, un eveniment intr-un calendar online), de transformarea datelor de la senzori in formatul inteles de CONSERT Engine si de trimiterea lor către ContextCoordinatorAgent
- CtxCoordinatorAgent - unitate responsabila de gestiunea motorului de inferența CONSERT si de coordonarea procesului de provisioning al informației contextuale (ce fel de date trebuie culese momentan, cu ce frecventa)
- CtxQueryHandlerAgent - unitate responsabila cu gestiunea interogărilor asupra rezultatelor inferate de motorul CONSERT. Aceste interogări provin de la aplicațiile finale care isi adaptează funcționalitatea după situațiile inferate de motorul CONSERT.
- CtxUserAgent - unitate care asigura interfața intre aplicație si middleware-ul CONSERT (e.g. din care se pot lansa interogările pentru informație contextuala - precum localizarea unei persoane, starea ei de ocupație, activitatea curenta desfășurata, etc.)
- OrgMgrAgent - unitate care gestionează ciclul de viața (creare/activare/dezactivare) al unitatilor definite mai sus, in funcție de gruparea logica din care fac parte (a se vedea in continuare).

Aceste patru unitati pot fi grupate in grupuri logice numite CMU (eng. Context Management Units).
Aceste grupuri constituie unitatea de baza de deployment la nivelul middleware-ului CONSERT. Ele pot cuprinde unul sau mai multe dintre unitățile descrise mai sus, in funcție de cate dintre ele trebuie sa ruleze pe o anumita mașina fizica (e.g. pe una poate rula doar CtxSensorAgent, pe când pe o alta, cu rol de server, pot rula restul).
Fiecare CMU este însa folosit pentru a deservi interesul de aprovizionare a informațiilor provenind dintr-un subset bine definit al unui model de context mai complex (e.g. doar acel set de informații referitoare la activitatea curenta, sau la locul curent in care se afla un utilizator).

Prin urmare, middleware-ul CONSERT ajuta o aplicație sensibila la context sa:

- Gestioneze întregul ciclul de furnizare a informațiilor contextuale, de la colectarea de date primare, la consumarea situațiilor inferate
- Efectueze o impartire la nivel logic a procesului de aprovizionare cu informație contextuala a unei aplicații, utilizând CMU-uri
- Creeze ierarhii de diseminare a informației contextuale pentru a permite rutarea interogărilor precum si grade diferite de granularitate a detaliului informației de context (e.g. utilizatorul este in sala 20 vs. utilizatorul este in corpul de clădire A)

3.2 Reimplementarea unitatilor componente din middleware-ul CONSERT

Middleware-ul CONSERT si, in special, modul de implementare al CMU-urilor a cunoscut o prima iterație prezentata in [Sorici et al, 2015a].
Obiectivele proiectului CONSERT pentru middleware-ul CONSERT stipulează necesitatea unei flexibilitatea mai mare in opțiunile de deployment, astfel incat sa poată fi cuprinse si scenarii venind din mediul IoT, unde deployment-ul peste rețele locale sau peste internet este necesar.
Implementarea abordata in [Sorici et al, 2015a], deși funcționala, era bazata pe o tehnologie din domeniul sistemelor multi-agent (JADE[1]) care limita modul de depolyment al unitatilor dintr-un CMU sub forma de componente web (precum ar fi necesar in scenarii IoT si AmI).

[1] http://jade.tilab.com/

De aceea, in proiectul CONSERT, accentul a căzut pana in momentul de fata pe reimplementarea unitatilor de lucru prezentate in secțiunea 3.1 sub forma unor servere web foarte ușoare ce permit atât comunicație HTTP, cat si prin Websockets[2].
De asemenea, protocolul de comunicație intre componente a fost reimplementat sub forma unui protocol RESTful[3], cu mesaje formulate direct in RDF, conform vocabularului definit de ontologia CONSERT [Sorici et al, 2015b].

Reimplementarea agenților (unitatilor) din CONSERT Middleware s-a făcut pe baza frameworkului Vert.x[4]. Acesta din urma permite construirea de aplicații web activate pe baza de evenimente (event-driven) si non-blocante, permițând comunicație asincrona. In vederea unui deployment peste web, acestea sunt atribute esențiale.
Unitățile din CONSERT Middleware sunt readaptate ca mici servere web ce implementează protocolul de interacțiune CONSERT. Pentru fiecare agent sunt definite componente de tratare de mesaje, folosind cereri pe baza de protocol atât HTTP, cat si Websockets.
Motivul pentru cele din urma este legat de posibilitatea ca agenții unui CMU sa ruleze pe dispozitive care nu au un IP public accesibil. In acest caz, un răspuns asincron de la server către client nu ar putea fi trimis din motiv de lipsa a unei adrese a destinatarului.
Ca urmare, utilizarea Websockets permite menținerea unei comunicații bidirecționale peste web.

Fiecare agent este capabil sa mentina corect starea sa interna si sesiunile de comunicare, indiferent de canalul (HTTP sau Websocket) pe care a sosit o cerere.
Agenții au definite clase ce implementează metode de prelucrare pentru fiecare cale din protocolul CONSERT. Organizarea pe clase poate fi observata in Figura 1.

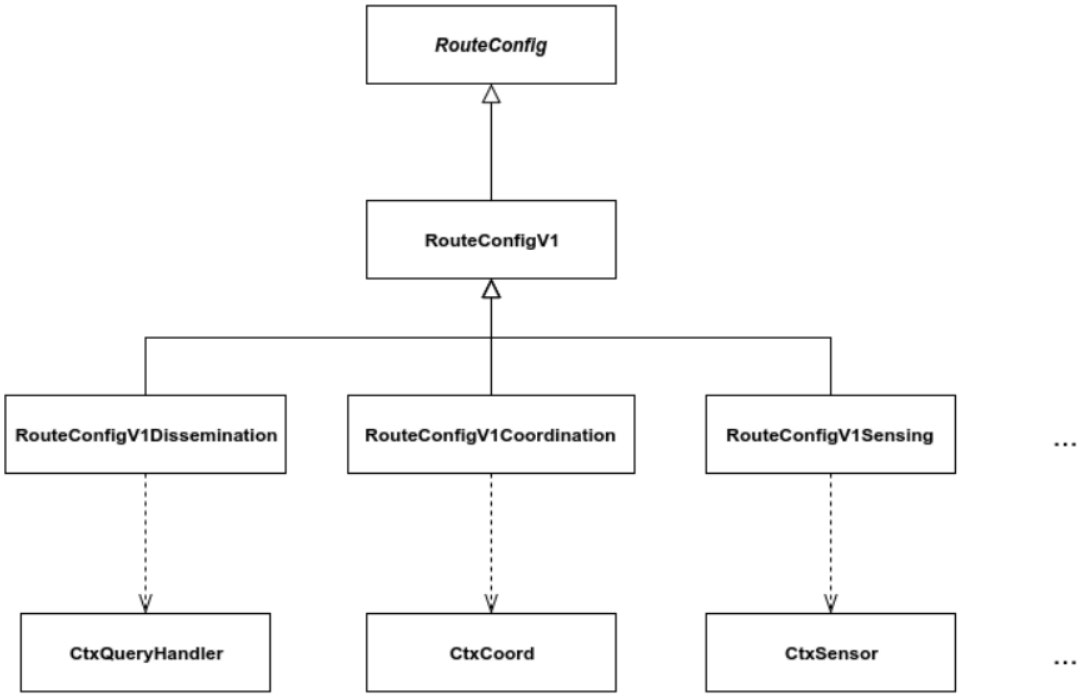

Figura 1. Organizarea rutelor de prelucrare de apeluri din protocolul CONSERT

In secțiunea următoare vom prezenta pe scurt protocolul CONSERT si modul in care agenții din CONSERT Middleware efectuează serializarea conținutului mesajelor.

---

2 https://en.wikipedia.org/wiki/WebSocket
3 https://en.wikipedia.org/wiki/Representational_state_transfer
4 http://vertx.io/

3.3 CONSERT Middleware RESTful protocol implementation

Odată cu reimplementarea agenților din middleware-ul CONSERT sub forma de mici servere web s-a efectuat si reimplementarea modului de comunicare intre aceștia.
Îndeosebi, comunicația a fost redefinita ca un protocol RESTful. Acest lucru ca in loc de primitive folosite in FIPA ACL[5] (precum erau folosite in framework-ul JADE), s-au definit primitive de comunicare centrate in jurul "verbelor" folosite in protocolul HTTP (e.g. GET, POST, PUT, DELETE, PATCH) unde semantica fiecăruia este data de interpretarea data in REST.
In plus, conținutul mesajelor a fost si el actualizat, astfel incat in loc de o encodare in mesaje Java s-a dezvoltat o ontologie suplimentara, numita CONSERT protocol ontology[6]. Ca urmare conținutul mesajelor schimbate intre agenții CONSERT Middleware sunt exprimate in termenii ontologiei si serializati in format RDF.
Astfel, CONSERT Protocol se definește ca o serie de comunicații RESTful cu date efective (payload) RDF.

In cadrul protocolului CONSERT agenții schimba următoarele tipuri principale de mesaje:

- Anunțarea existentei si a capabilităților (ce fel de date poate transmite si sub ce forma - e.g. cat de des) unui CtxSensorAgent
- Actualizarea/Anularea/Interogarea capabilităților unui CtxSensorAgent
- Inserarea unui nou eveniment
- Inserarea/Actualizarea datelor statice (fapte care se schimba foarte rar sau deloc, precum: datele de contact ale unui utilizator, incluziunea spațiala a unor săli intr-o clădire)
- Transmiterea unei interogări asupra situațiilor contextuale inferate
- Abonarea/Dezabonarea de la o interogare periodica a situațiilor contextuale inferate
- Mesaje de descoperire si înregistrare intre agenți (e.g. un CtxSensorAgent descoperă agentul CtxCoordinator asociat, un agent CtxUser descoperă un agent CtxQueryHandler asociat)

5 http://www.fipa.org/repository/aclspecs.html

6 Fisierul RDF care defineste CONSERT Protocol ontology: https://goo.gl/PyFkdh

Mesajele privitoare la inserarea datelor de context sunt receptate de către agentul CtxCoordinator. Acesta folosește biblioteca RDFBeans[7] pentru a converti din format RDF in formatul Java inteles de CONSERT Engine.
Tot agentul CtxCoordinator este responsabil de a transmite mai departe evenimentul primit de la un agent CtxSensor către motorul de inferența CONSERT.
Situațiile inferate sunt apoi reconvertite in RDF pentru a fi stocate intr-un depozitar RDF4J[8] in-memory. De acolo ele pot fi interogate de către agentul CtxQueryHandler prin intermediul cererilor primite de la agentul CtxUser, formulate in limbaj SPARQL[9].

Definirea protocolului CONSERT ca interacțiune RESTful aduce beneficii in ceea ce privește comunicarea asincrona.
Figura 2 prezintă secvența de interacțiuni din cadrul protocolului CONSERT atunci când un agent CtxUser efectuează o abonare la o interogare sau o interogare cu răspuns întârziat (pentru ca procesarea sa poate dura).

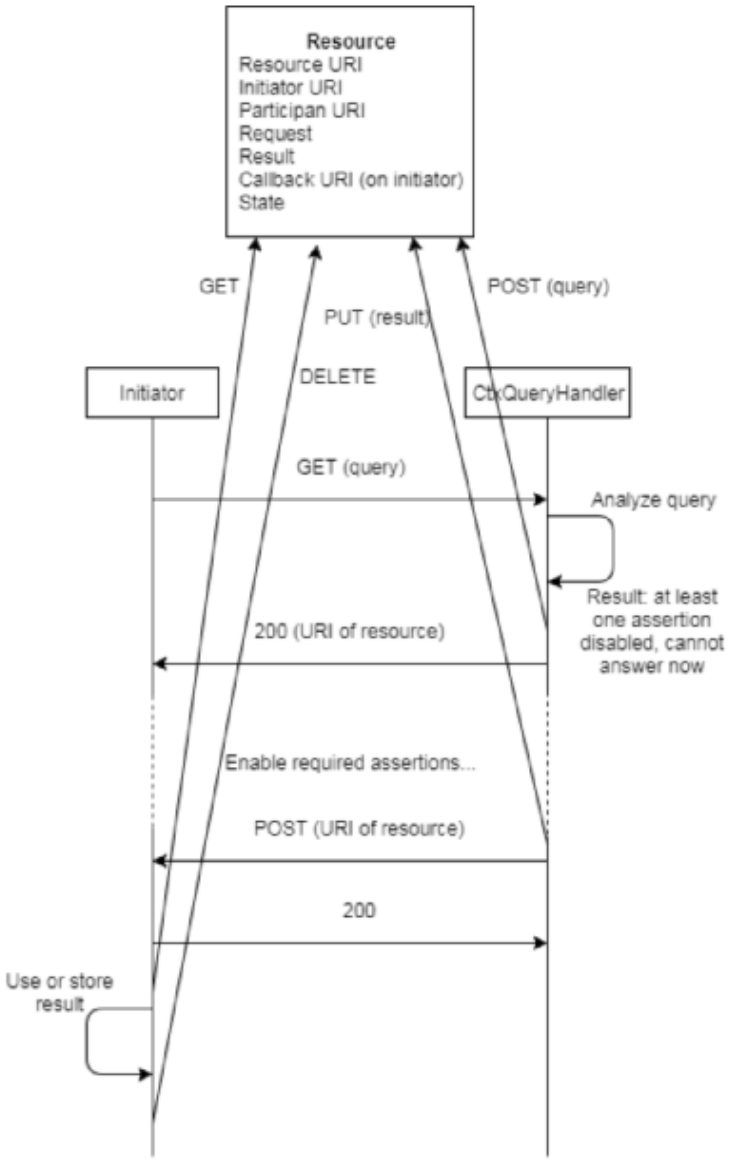

Figura 2. Exemplu de interacțiuni in protocolul CONSERT la interogare asincrona / abonare la interogare

In acest caz, conform definiției REST, o resursa intermediara este creata, care menține starea interacțiunilor pana in momentul in care exista o dezabonare a agentului CtxUser de la primirea răspunsurilor pentru interogarea inițiala.

3.4 Proiectarea opțiunilor de mobilitate în CONSERT Middleware

In cadrul protocolului CONSERT, agenții din cadrul unui CMU schimba si o serie de mesaje de inițializare si căutare a interlocutorilor. Strategia de deployment prezentata in [Sorici et al, 2015a] se păstrează, astfel incat agentul de tip OrgMgr este cel care deține adresele si configurația interna a fiecărui agent din CMU-ul pe care OrgMgr il gestionează.

---

7 https://rdfbeans.github.io/rdfbean.html
8 http://rdf4j.org/
9 https://en.wikipedia.org/wiki/SPARQL

Primul pas după activarea unui agent dintr-un CMU este ca acesta sa-si ceara configurația de la OrgMgr, acesta fiind singurul agent din CMU a cărei adresa ii este cunoscuta la început. De la OrgMgr va primi apoi adresele celorlalți agenți cu care trebuie sa stea in contact (e.g. un CtxSensor va primi adresa unui CtxCoord, iar un CtxUser adresa unui CtxQueryHandler, toți făcând parte din același CMU - cf. Figura 3).

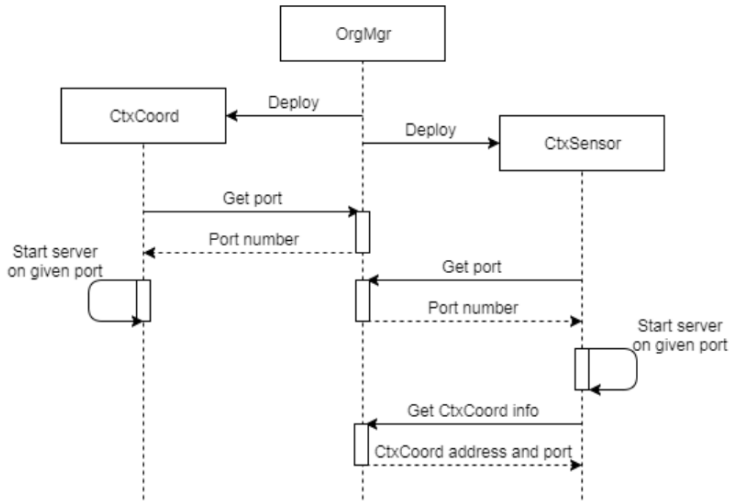

Figura 3. Diagrama de secvența pentru procedura de deploy a agenților CtxSensor si CtxCoordinator

Din punct de vedere al deployment-ului, un aspect important pe care îl oferă reimplementarea sub forma de servere web a agenților CONSERT este flexibilitatea metodei de implementare. Pentru agenții unui CMU precum CtxCoordinator si CtxQueryHandler care ar rula deseori pe o mașina centrala deployment-ul se poate realiza sub forma dockerizata[10], containerele putând fi ușor create, (re)pornite si oprite.
Pentru agenții de tip CtxUser si CtxSensor un deployment simplificat, folosibil si pe dispozitive mobile se va putea face folosind doar nucleul framework-ului Vert.x si o platforma dezvoltata in regim propriu pentru gestiunea ciclului de viața al agentului.
Atât deploymentul dockerizat cat si cel particular pentru dispozitive mobile fac obiectul dezvoltărilor din etapa următoare a proiectului.

In middleware-ul CONSERT un CMU reprezintă o unitate logica de grupare a gestionarii unui subset de aserțiuni contextuale dintr-un model de context al aplicației.
Modelul de context construit prin CONSERT permite definirea unei așa numite *dimensiuni contextuale* (e.g. spațiala, de activitate) care duce la o impartire pe *domenii contextuale* (e.g. după locul in care se afla o persoana, după activitatea care se desfasoara actualmente, după rolul jucat de o persoana in acea activitate).
Mecanismul de mobilitate propus in CONSERT exploatează aceasta impartire logica, propunând metode de detecție pasiva sau activa a schimbării de la un domeniu contextual la altul, prin intermediul unor mesaje de tip anunț care se schimba intre doi agenți de tip OrgMgr.
Implementarea acestui mecanism si evaluarea lui se va face in etapa a 2-a a proiectului.

4. CONSERT IDE

IDE-ul CONSERT este gândit ca unealta de susținere pentru dezvoltatorii de aplicații sensibile la context care utilizează modelul CONSERT in definirea aplicației lor.
IDE-ul utilizează modelele de reprezentare si deployment definite in CONSERT (a se vedea Secțiunea 3 si Sorici et al, 2015a).
Scopul IDE-ului este de a oferi susținere in trei aspecte:

[10] https://www.docker.com/

- Definirea modelului contextual
- Definirea si configurarea opțiunilor de deployment pentru unitățile middleware-ului CONSERT
- Generarea automata de cod pentru utilizarea in motorul de inferența CONSERT

Raportul curent prezintă design-ul si implementarea functionalitatii pentru primul aspect: *definirea modelului contextual.*

4.1 Rolul si functionalitatile CONSERT IDE

Se definesc in cele ce urmează cerințele funcționale identificate inițial pentru IDE-ul CONSERT, cele privitoare la capabilitatea de definire a elementelor ce compun modelul de context al unei aplicații pe baza meta-modelului CONSERT.

Crearea / Deschiderea / Ștergerea unui proiect

- IDE-ul permite crearea, deschiderea si ștergerea unui proiect ce cuprinde definiția modelului contextual al unei aplicații.
- Pentru crearea unui nou proiect se definește un *wizard* (casuta de dialog) ce permite specificarea numelui noului proiect
- Deschiderea unui proiect existent se face selectând printr-un *wizard* fișierul suport care menține serializarea elementelor contextuale ce au fost definite in proiect

Navigarea printr-un set de proiecte si elementele componente ale unui proiect

- Crearea unui element de interfața pentru navigarea prin setul de proiecte definite si prin conținutul fiecărui proiect in parte
- Interfața de navigare afișează proiectele si conținutul acestora sub forma de arbore (TreeView), făcând distincție clara intre diferitele tipuri de elemente contextuale (entitati, aserțiuni, adnotări si descrieri)
- Interfața de navigare permite afișarea unui meniu contextual pentru a putea crea noi entitati si aserțiuni ce vor fi adăugate la modelul de context

Crearea elementelor ce compun un model de context

- Pentru fiecare element de context se definește un *wizard* care ajuta la completarea atributelor specifice fiecărui element contextual
    - ContextEntity: nume si descriere
    - ContexAssertion: nume, tip de achiziție (de la senzor, de profil sau derivat), entitate subiect, entitate obiect
    - EntityDescription: nume, entitate subiect, entitate obiect sau valoare literala
    - ContextAnnotation: nume, categorie (simplu sau structurat), tip (e.g. timestamp, grad de încredere)

Editarea elementelor ce compun un model de context

- Pentru fiecare element de context din cele definite de meta-modelul CONSERT (*ContextEntity, ContextAssertion, EntityDescription* si *ContextAnnotation*) se construiesc doua tipuri de interfața grafica pentru vizualizarea atributelor lor (cele descrise mai sus)
    - Crearea unui editor de tip *formular* (form view) pentru editarea atributelor in mod grafic
    - Crearea unui editor de tip text prin care dezvoltatorii (avansați) pot opera schimbări directe in definiția unui element contextual serializat in format JSON sau RDF

Pe langa funcționalitatea de creare, vizualizare si editare a elementelor dintr-un model contextual, IDE-ul CONSERT trebuie sa poată genera transpuneri ale acestora intr-un format serializabil (e.g. JSON, RDF), astfel incat sa se poată asigura persistenta.

In secțiunea următoare prezentam tehnologia si design-ul ales pentru a îndeplini aceste cerințe funcționale.

4.2 Proiectarea CONSERT IDE ca un Eclipse RCP

Designul CONSERT IDE a fost gândit ca un Eclipse plugin a cărui funcționalitatea de baza sunt bazate pe conceptele de “extension” si “extension point”, prin care dezvoltatorii contribuie la functionalitatile existente din Eclipse IDE, adăugând noi perspective, view-uri, sau noi functionalitati construite de la zero.

Un plugin Eclipse este o componenta software care are rolul de a extinde functionalitatile existente ale Eclipse IDE si permite adăugarea unor noi feature-uri pentru activitati de dezvoltare de aplicații. Acest lucru vine ca argument in favoarea crearii de noi medii de programare prin intermediul plugin-urilor de Eclipse.

Pentru realizarea CONSERT IDE ca un plugin Eclipse, au fost definite urmatoarele puncte de extensie in cadrul fisierului plugin.xml:

- org.eclipse.ui.perspectives
- org.eclipse.ui.perspectiveExtensions
- org.eclipse.core.resources.natures
- org.eclipse.ui.newWizards
- org.eclipse.ui.importWizards
- org.eclipse.ui.editors
- org.eclipse.ui.views

Pentru fiecare din aceste puncte de extensie s-au adaugat, de asemenea, urmatoarele extensii: o noua perspectiva (ConsertPerspective), Package Explorer si Console View ca parte a perspectivei Consert, TreeView – un view dedicat explorarii proiectelor de tip Consert, natura ConsertNature pentru proiectele de tip Consert, wizard-uri pentru definirea de proiecte noi si elemente ale modelului (Entitati si Asertiuni), un wizard pentru importul proiectelor Consert in IDE, editoare pentru Entitati si Asertiuni.

Au fost definite de asemenea categorii specifice pentru: un nou tip de proiect (ConsertProject), wizard-urile care deserves creare de elemente ale modelului (ContextModelDefinition) si subcategorii pentru Entitati (ContextEntity) si Asertiuni (ContextAssertion).

Fiserul MANIFEST.mf contine informatiile de configurare OSGi si este locul unde Eclipse Plugin-ul definește meta-datele sale, cum ar fi identificatorul său unic, API-ul exportat și dependențele acestuia.
IDE-ul Consert se deschide prin rularea Proiectului de tip Eclipse Plugin in workbench-ul Eclipse RCP.

4.3 Functionalitatea curenta si utilizarea CONSERT IDE

In prezent, CONSERT IDE este dezvoltat ca un plugin Eclipse si cuprinde urmatoarele feature-uri:

- **Consert Perspective** – aceasta noua perspectiva are rolul de a grupa toate functionalitatile noi care deservesc CONSERT Middleware si se gaseste in *perspective toolbar,* in coltul din dreapta sus a workbenchului

- **Consert Nature** – este natura fiecarui proiect nou de tip Consert si este utilizata la configurarea proiectului in workspace

- **New Consert Project Wizard** – acest wizard permite crearea de noi proiecte Consert; Utilizatorul trebuie sa introduca numele proiectului nou creat; Acest wizard este disponibil accesand meniul *File -> New -> Project* si alegand categoria *ConsertProject* (cf. Figura 4). La finalizarea acestui Wizard, se va crea un proiect nou, avand natura Consert si o structura predefinita de directoare, care grupeaza elementele modelului, dar si un director care contine fisierul in care se vor scrie toate definitiile elementelor definite de utilizator in cadrul proiectului. De asemenea, se creaza un director care va reprezenta locul in care ontologii existente se vor putea incarca pentru a ajuta utilizatorul sa defineasca mai rapid elemente ale modelului.

- **Import Consert Project Wizard** – acest wizard permite importul unui proiect de tip Conert in CONSERT IDE. Utilizatorul are acces la o functionalitate tip “Browse” prin care poate sa selectez un proiect existent pentru a-l importa.

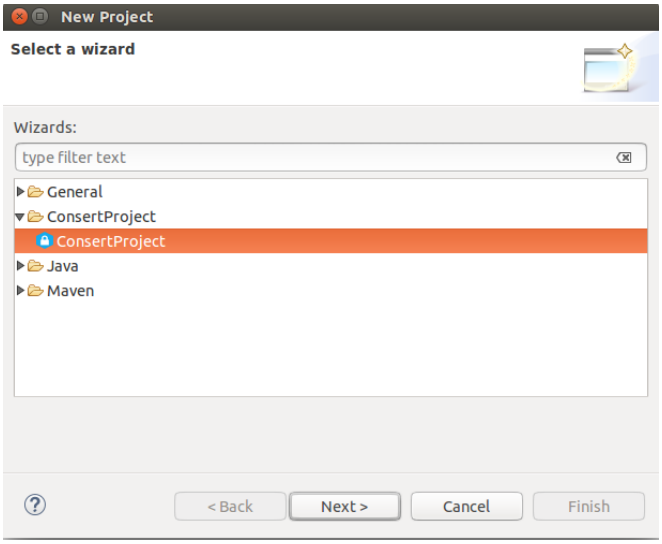

Figura 4. Captura de ecran pentru wizard-ul de proiect nou

- **New Context Model Elements Wizards** – aceste wizard-uri permit utilizatorului crearea intr-un mod usor si intuitiv de noi Entitati si Asertiuni. New Context Entity Wizard se prezinta ca un formular in care utilizatorul introduce numele noii Entitati si un comentariu. New Context Assertion Wizard (cf. Figura 5) permite crearea de asertiuni binare si se prezinta ca un formular in care utilizatorul introduce numele noii Asertiuni, un comentariu, Acquisition Type si cele doua Entitati care iau parte in asertiune. Alegerea Entitatilor participante in assertiune se face prin selectarea de Entitati din cele deja definite in cadrul proiectului, prin intermediul unui drop-down.

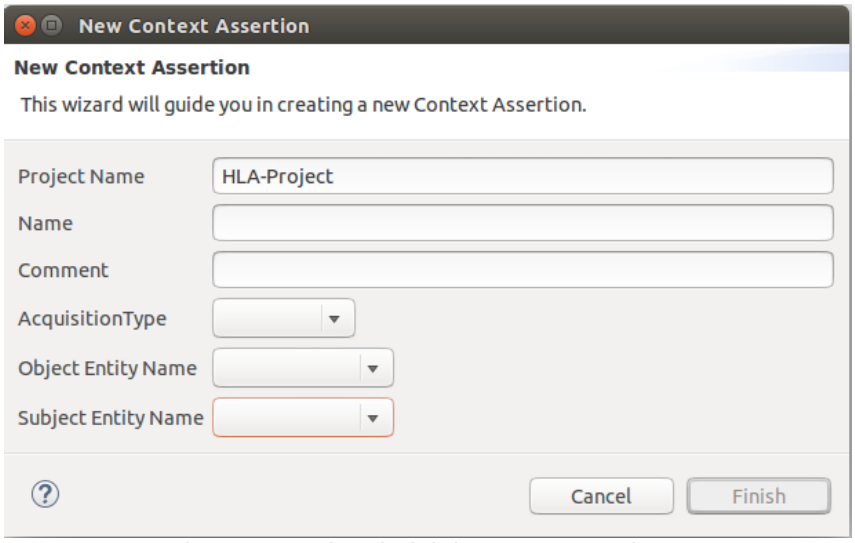

Figura 5. Captura de ecran a wizard-ului de creare a unui nou ContextAssertion

- **Workspace Model si Project Model** – la nivelul IDE-ului, o data cu initializarea perspective CONSERT, are loc si initializarea Workspace Model. Acesta reprezinta un model al tuturor proiectelor de tip CONSERT existente in Workspace, impreuna cu elementele definite de utilizator in fiecare dintre aceste proiecte. Project Model este un model corespunzator unui Proiect Consert si se initializeaza la crearea unui proiect nou, si sufera update-uri la fiecare modificare ale elementelor definite de utilizator sau la definirea de noi elemente in cadrul unui proiect. Rolul Project Model este de a oferi o sincronizare intre actiunile intreprinse de utilizator prin intermediul GUI si fisierul global al proiectului in care sunt tinute definitiile elementelor contextuale.

- **Tree View** – aceasta functionalitate este similara cu cea de package explorer din Java Perspective si permite vizualizarea Entitatilor si Asertiunilor definite de utilizator la nivelul fiecarui proiect in parte, acestea fiind grupate in “directoare” specifice (cf. Figura 6). Acest view disponibil in perspectiva Consert, afiseaza numai acele proiecte deschise, avand natura Consert. Un alt feature al acestei functionalitati este un meniu contextual din care se pot deschide wizard-urile corespunzatoare definirii de noi Entitati si Asertiuni la nivelul fiecarui proiect. De asemenea, in acest meniu, se gaseste si o optiune de stergere a unei Entitati sau Asertiuni. Fiecare din actiunile intreprinse la nivelul acestui view sunt sincronizarte cu Project Model, respectiv Workspace Model.

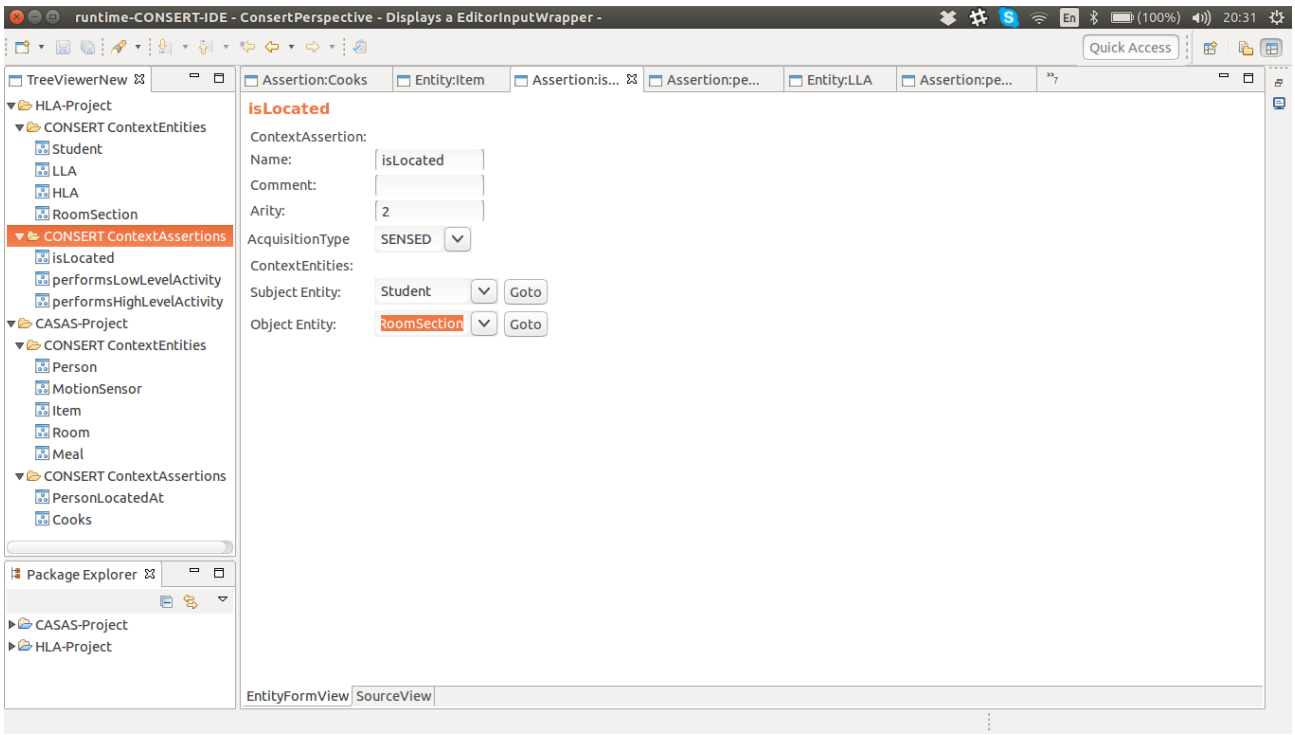

Figura 6. Captura de ecran prezentand interfetele grafice de navigare printre proiecte si continutul acetora (stanga) si de editare a unui ContextAssertion (dreapta).

- **Form View si Source View** – aceste functionalitati permit vizualizarea fiecarei Entitati sau Asertiuni, atat sub forma unui formular editabil cat si in forma "raw", exact asa cum apar definite in fisierul global al proiectului (cf. Figura 6). Pentru acesarea lor este suficient ca utilizatorul sa aleaga din Tree View proiectul si apoi elementul pe care doreste sa il vizualizeze. La dublu-click pe numele Entitatii sau Asertiunii se va deschide o zona in care apar cele doua functionalitati, care se pot accesa prin selectarea tab-ului afferent fiecaruia dintre ele. Modificarile realizate de utilizator in Form View sunt reflectate in Source View si salvate in fisierul global al definitiilor.

5. Continuarea cercetării și implementării

Prima etapa a proiectului s-a concentrat pe reimplementarile necesare pentru imbunatatirea elementelor componente ale middleware-ului CONSERT (motorul CONSERT, unitatile logice ale middleware-ului), cat si pe proiectarea functionalitatilor de baza ale IDE-ului CONSERT.

Etapa urmatoare a proiectului CONSERT prevede evaluarea imbunatatirilor aduse motorului CONSERT, a utilitatii IDE-ului CONSERT si a middleware-ului per ansamblu.

În particular, dezvoltarile ulterioare si evaluarile gandite pentru fiecare aspect al proiectului sunt:

- Motorul de inferenta CONSERT
  - Testarea capabilitatilor de inferenta contextuala pe setul de date CASAS[11] si compararea cu abordari existente de recunoastere a activitatii
  - Testarea capabilitatilor de inferenta contextuala pe baza unei aplicatii suport pentru activitatea studentilor in campusul UPB

11 Setul de date CASAS: http://ailab.wsu.edu/casas/datasets.html

- Evaluarea motorului de inferenta CONSERT in cadrul unor scenarii de asistenta pentru activitatea persoanelor in varsta

- CONSERT IDE-ul
  - Dezvoltarea capabilitatii de exportare sub format RDF a unui model de context creat in CONSERT IDE, facand uz de ontologia CONSERT
  - Dezvoltarea capabilitatii de generare automata de cod (eng. boiler plate code generation), sub forma de clase direct folosibile in motorul de inferenta CONSERT
  - Dezvoltarea capabilitatii de incarcare a unei ontologii existente, astfel incat clasele si entitatiile definite sa poata fi direct marcate drept Entitati Contextuale (ContextEntity) si Asertiuni Contextuale (ContextAssertions)

- Middleware-ul CONSERT
  - Dezvoltarea optiunilor pentru deployment dockerizat a unitatilor (agentilor) din middleware-ul CONSERT
  - Definirea si implementarea unui set initial de *adaptoare* pentru unitatile de tip CtxSensor, astfel incat acestea sa poata prelua date de la senzori folositi in mod curent in laboratorul AI-MAS

În prezent avem 2 lucrări în curs de finalizare care vor fi trimise la:
- ISAMI 2018 International Symposium on Ambient Intelligence, June 2018, Toledo, Spain (deadline 5 februarie 2018)
- 12th International Symposium on Intelligent Distributed Computing, Oct 2018, Bilbao, Spain (deadline 9 aprilie 2018).

Bibliografie

[Drools, 2017a] https://www.drools.org/, accesat dec 2017

[Drools, 2017b] https://docs.jboss.org/drools/release/6.2.0.CR2/drools-docs/html/HybridReasoningChapter.html#PHREAK, accesat dec 2017

[Forgy, 1982] Charles, Forgy (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence. 19: 17–37

[Sorici et al., 2015a] Sorici, A., Picard, G., Boissier, O., & Florea, A. (2015, June). Multi-agent based flexible deployment of context management in ambient intelligence applications. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 225-239). Springer, Cham.

[Sorici et al, 2015b] Sorici, A., Picard, G., Boissier, O., Zimmermann, A., & Florea, A. (2015). CONSERT: Applying semantic web technologies to context modeling in ambient intelligence. Computers & Electrical Engineering, 44, 280-306.