

Sistem de achiziție, procesare si distribuite a datelor

Petrișor-Ștefan Lăcătuș

Septembrie 2015

Automatică și Ingineria Sistemelor
Facultatea de Automatica si Calculatoare

Coordonator: Ș.L. Andreea Udrea

Cuprins

1	Introducere	1
2	Arhitectura soluției	2
2.1	Prezentare generală	2
2.1.1	Baza de date	4
2.1.2	Aplicația Java	5
2.2	Entități	5
2.2.1	Punctul de date	5
2.2.2	Canalul de date	7
2.2.3	Blocul de intrare	7
2.2.4	Blocul de procesare	9
2.2.5	Diagrama funcție bloc(FBD)	10
2.3	Primirea datelor	11
2.4	Executarea unei diagrame	14
2.4.1	Ordinea execuției blocurilor	14
2.4.2	Generarea rezultatului	17
3	Interfața aplicației	18
4	Implementarea Alegria	19
4.1	Aplicația Web	19
4.1.1	Securitate	20
4.1.2	Management	21
4.1.2.1	Managementul blocurilor de intrare	22
4.2	Baza de date	22
5	Studiu de caz: Smart Home	23
6	Concluzii	24

Capitolul 1

Introducere

Capitolul 2

Arhitectura soluției

2.1 Prezentare generală

Alegria a fost concepută ca o platformă de dezvoltare rapidă, bazată pe modele. Prin folosirea unor metode de programare vizuală, cu blocuri reutilizabile în mai multe aplicații diferite, utilizatorul poate să își concentreze resursele asupra soluției finale, abstractizând detaliile implementării. Aplicația a fost construită pe baza unei arhitecturi modulare, cu module cât mai puțin cuplate, care să permită modificări rapide și testarea modulelor individuale. Urmărind această gândire modulară, au fost identificate 4 componente esențiale:

- **Baza de date** Asigură stocarea datelor, dar și a entităților existente în aplicație;
- **Interfața web pentru management** O interfață ușor de utilizat care să permită utilizatorului să manipuleze canale, diagrame, blocuri de procesare, dar și alți utilizatori
- **API-ul pentru date** Un API specializat pentru adăugare de date, achiziționarea datelor, dar și să permită altor dispozitive să fie notificate de fiecare dată când apar date noi.
- **Elemente de procesare** Asigură procesarea datelor, atât în cadrul blocurilor de procesare, dar și în cadrul diagramelor funcționale.

În vederea implementării sistemului s-au identificat următoarele elemente componente esențiale, componente ce reprezintă elementele constructive a sistemului. Acestea, au reprezentare atât în baza de date, ca entități, cât și în aplicația Java, ca clase. Identificarea acestor elemente s-a făcut pe baza analizei cazurilor de utilizare a produsului, în care s-au investigat metodele prin care actorii interacționează cu sistemul.

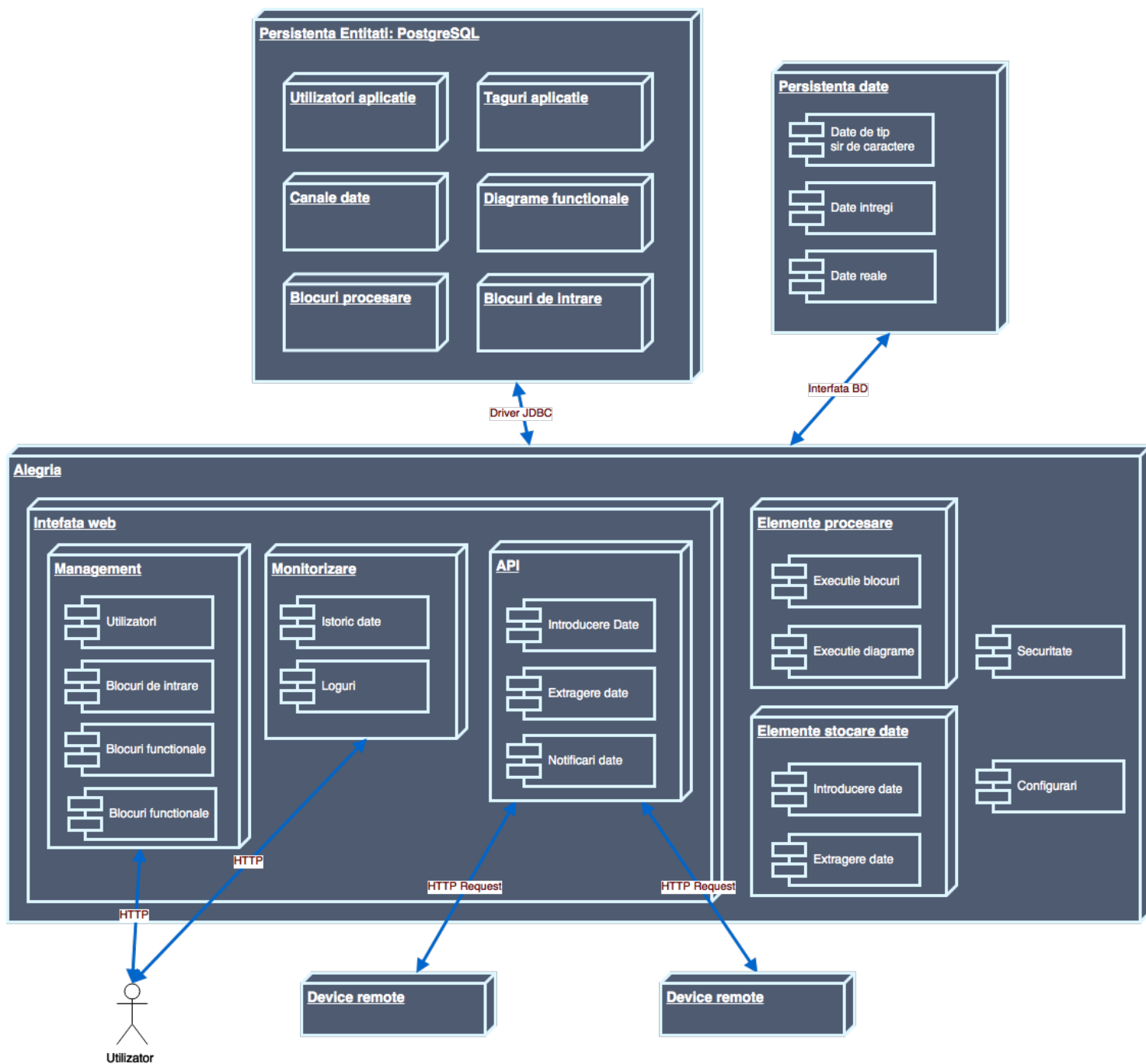


Figura 2.1: Arhitectura generala a aplicatiei

- **Canalul de date:** reprezintă elementul de bază al sistemului, care asigură recepția, persistența și emiterea de date. Datele dintr-un canal trebuie să respecte un format prestabilit la crearea canalului. Pentru transformarea datelor în formatul stabilit, se poate introduce un bloc de pre-procesare care transformă datele din un format brut în formatul standard.
- **Blocul de intrare:** elementul de mapează un element din lumea reală în interiorul platformei. Blocurile de intrare permit gruparea mai multor canale de date într-o structură unică.
- **Blocul de procesare:** elementul dinamic al aplicației, ce aplică transformări asupra datelor. Un bloc de procesare primește ca intrări mai multe canale de date, și are la ieșire un alt canal de date. Utilizatorul poate folosi blocuri standard, existente în sistem, sau poate implementa blocuri noi direct în interfața programului.
- **Diagrama funcție bloc(FBD):** folosește blocuri de intrare, canale de date și blocuri de procesare pentru a descrie o funcție complexă între intrări și o ieșire. Aceste diagrame folosesc la date aflate pe canale de date, care sunt trimise către blocuri de procesare și, la final se obține un singur rezultat care este salvat pe un canal de date.

2.1.1 Baza de date

Fiind vorba despre o aplicație puternic bazată pe date, aceasta are nevoie de un nivel de persistență de înaltă performanță. Urmărind arhitectura propusă din figura 2.1, putem identifica două cazuri de utilizare pentru baza de date:

- **Stocarea modelului entităților:** fiecare entitate descrisă în lista de mai sus trebuie stocată în baza de date într-o structură relațională. Entitățile sunt puternic interconectate, iar o bază de date relațională, de tip SQL este recomandată în acest caz. Din punct de vedere al dimensiunii setului de date, chiar și în aplicațiile de mare complexitate, este vorba despre doar câteva milioane de înregistrări, factorul care face acest număr să crească fiind conectarea a tot mai mult dispozitive, ce duce la din ce în ce mai multe canale de date. Astfel, stocarea entităților nu va aduce probleme de performanță.
- **Stocarea datelor:** în baza de date vor fi stocate atât datele primite pe fiecare canal asociat unui bloc de intrare, cât și datele procesate de diagrame. Aceste date au un puternic caracter istoric, reprezentând o serie de timp, în care se reține, pentru fiecare punct de date, valoarea la un anumit moment. Problema stocării acestor date este una mai complicată, datorită necesității unei puternice scalări a bazei de date.

Această problemă reprezintă un caz de utilizare pentru o baza de date NoSQL, sau chiar o baza de date specializată în stocarea seriilor de timp.¹

2.1.2 Aplicația Java

Legătura dintre baza de date și utilizatorii finali se face prin intermediul unei aplicații Java complexe, care este obiectul acestui proiect. Aplicația conține toată logica platformei, de la operații asupra entităților din baza de date, la adăugarea, și extragerea datelor, cât și pentru procesarea datelor. Separarea modulelor s-a făcut pe baza scopului acestora:

- **Administrare:** pentru administrarea entităților din baza de date. Aceste module permit operații de căutare și afișare, dar și de creare, editare și ștergere a utilizatorilor, a blocurilor de intrare și funcționale precum și a diagramelor. Fiecare dispune de o interfață HTML5 în care moderna.
- **Monitorizare:** permit monitorizarea execuției aplicației, de la vizualizat loguri pentru a diagnostica probleme, la realizarea de grafice a datelor pe anumite canale. Tot aici este disponibilă și funcția de a exporta date în formate uzuale, ca CSV sau fișiere Microsoft Excel.
- **API:** aplicația dispune și de un API pentru a fi folosită programatic de către alte aplicații externe. Acesta poate fi considerat ca fiind format din două componente: serviciile pentru administrarea entităților, și cele pentru adăugarea și extragerea datelor.
- **Elemente de procesare:** împărțite în două subcategorii: cele pentru procesarea blocurilor de intrare și de procesare, și cele pentru procesarea diagramelor.
- **Elemente stocare date:** permit interfațarea cu sursele de date. Acestea asigură servicii de introducere și extragere a datelor, printr-o interfață abstractă, care nu ține cont de modul în care baza de date este implementată.
- **Alte module:** asigură, printre altele securitatea aplicației.

2.2 Entități

2.2.1 Punctul de date

Punctul de date reprezintă elementul constructiv al sistemului, care este obiectul procesării, stocării și distribuției este punctul de date. Sistemul acceptă intern date în formatele:

¹*The Scalable Time Series Database.* URL: <http://opentsdb.net/index.html> (visited on 08/20/2015).

- **Întreg:** numere de la -2^{63} la $2^{63} - 1$, fără virgula, folosește *Long* pentru reprezentare internă;
- **Real:** numere cu virgula, având dubla precizie, reprezentate cu 64-bit conform standardului² IEEE 754 folosește *Double* pentru reprezentare internă;
- **Sir de caractere:** Un sir de fără limite a lungimii, care trebuie formatat conform.³
- **Obiect:** Un obiect Java serializat în text. Intern, asemănător cu tipul de date String.

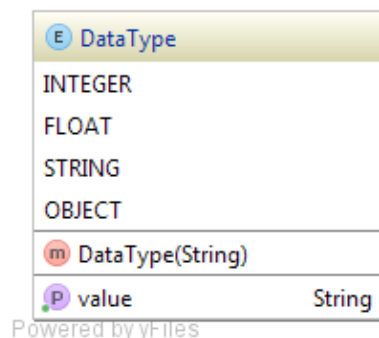


Figura 2.2: Tipurile de date acceptate în sistem

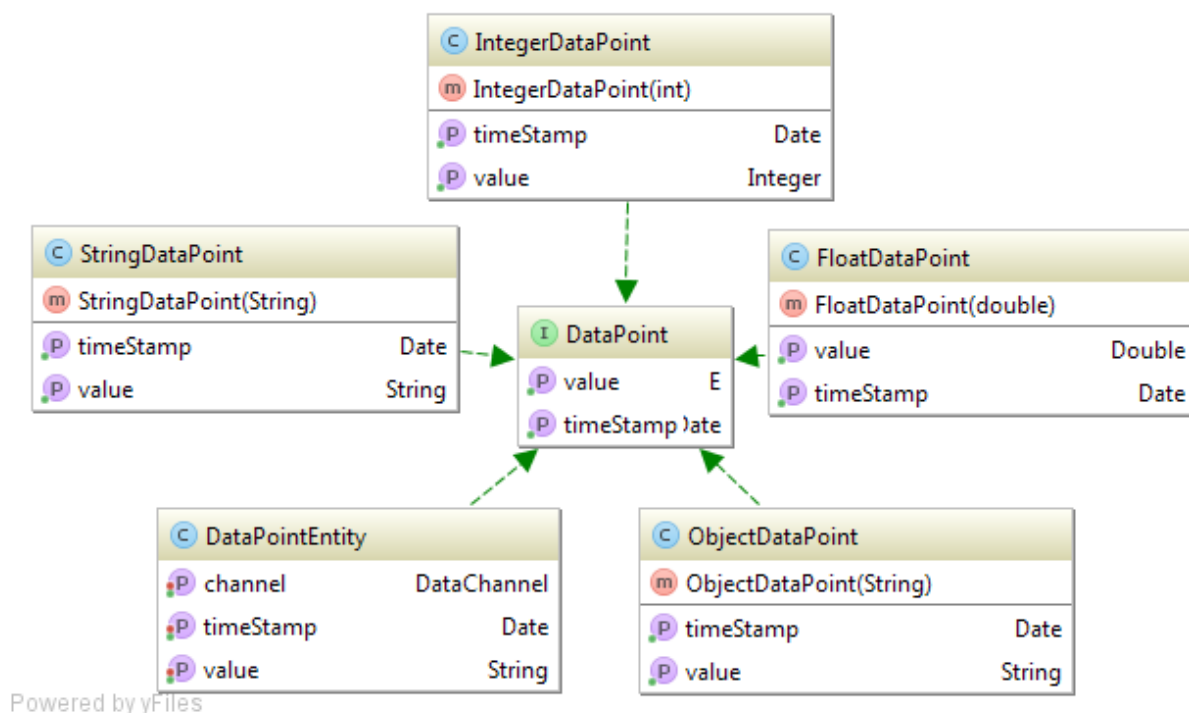


Figura 2.3: Clasele care implementează interfața DataPoint

²IEEE Standard for Floating-Point Arithmetic. Aug. 2008, pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.

³The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627. RFC Editor, July 2006, pp. 1–10. URL: <http://www.ietf.org/rfc/rfc4627.txt>.

2.2.2 Canalul de date

Canalul de date este entitatea care asigura "curgerea" datelor prin sistem. Orice punct de date din sistem aparține unui canal, acest lucru fiind realizat drept constrângere atât la nivelul aplicației, cat si la nivelul bazei de date.

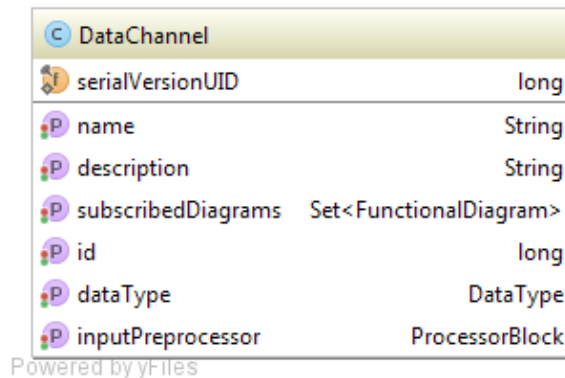


Figura 2.4: Clasa DataChannel

Canalul este si mijlocul prin care utilizatorul interacționează cu punctele de date. Când un dispozitiv adaugă date noi in sistem, acestea sunt atașate unui canal, care poate fi folosit ca una dintre intrările unei diagrame de blocuri funcționale. De asemenea, datele procesate de o diagrama sunt atașate unui canal, permițând apoi accesul pentru extragerea de datelor deja existente, si pentru a primi notificări de fiecare data când pe un canal apar informații noi.

Un canal mai poate avea si un bloc de preprocesare atașat. Acesta este executat de fiecare data când date noi încearcă sa fie introduse in canal, permițând validarea si transformarea datelor brute in date ce respecta tipul de date al canalului.

2.2.3 Blocul de intrare

Blocurile de intrare modelează elemente reale in Alegria, grupând mai multe canale de date si expunându-le pentru a permite introducerea de date din exterior. Acestea descriu modul in care datele sunt legate de un element real.

Spre exemplu, o sursă de alimentare neîntreruptibilă care este inteligenta si conectata din figura 2.6 poate fi privită ca un bloc de intrare, iar fiecare senzor de pe aceasta fiind canal de date. Dispozitivul inteligent devine astfel conectat la platforma si acesta poate sa trimită date către aceasta. Pentru ca datele pot sa aibă perioade de eșantionare diferite, dispozitivul trimite date către unul sau mai multe canale, fără sa fie forțat sa trimită date pentru toate canalele odată.

Un alt mod în care blocurile de intrare pot fi privite este ca obiecte, sau "Things" în cadrul Internetului Tuturor Lucrurilor (IoT). Astfel, putem privi blocurile de intrare ca dispozitive ce monitorizează bătăile inimii sau activitatea celebrară, ca automobile cu rețele complexe de senzori, sau chiar aplicații business ce generează date în timp real. Prin acest mijloc, dispozitive inteligente se pot conecta în platforma, fie trimițând direct date, fie prin intermediul unui agent care se afla pe dispozitiv, agent ce funcționează ca un gateway.

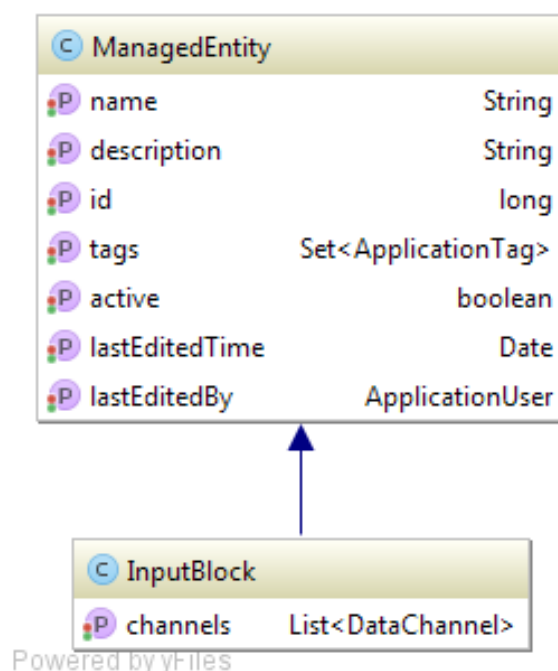


Figura 2.5: Clasa InputBlock

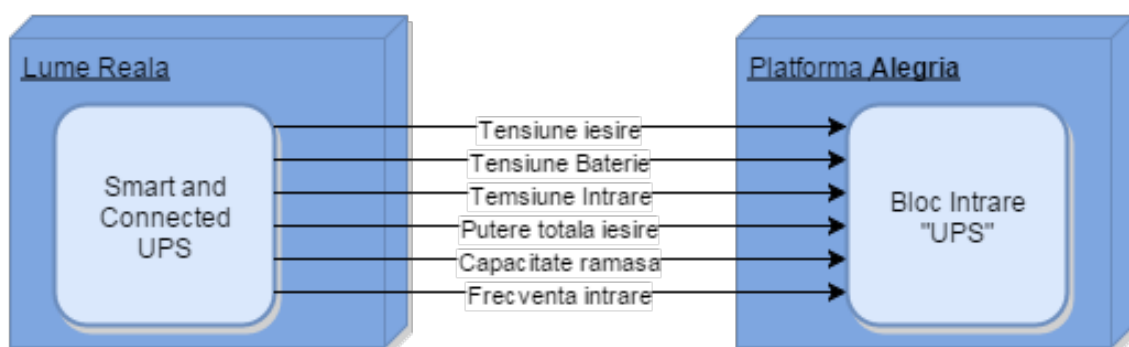


Figura 2.6: Modelarea unui UPS inteligent în Alegria

2.2.4 Blocul de procesare

Blocurile de procesare realizează operațiile transformare a datelor. Din punct de vedere funcțional, acestea au la intrare ultimele date introduse de pe mai multe canale și returnează un singur punct de date, comportându-se ca un element de tip "back-box".⁴ Pentru implementare, utilizatorul folosește limbaje dinamice moderne, ca JavaScript sau Ruby, scriind funcțiile direct în interfața web. Aceste funcții sunt rulate de către server. Conceptul de blocuri de procesare reprezintă o implementare a blocurilor de operații definite în standardul.⁵

Blocuri pre-implementate există în orice instanță a platformei care permit rezolvarea de probleme complexe cu cunoștințe minime de programare. Un alt aspect important al blocurilor de procesare, este că ele sunt complet independente, ele oferind doar mijloace de prelucrare a unor date abstracte. Aceasta abstractizare permite refolosirea lor în mai multe proiecte. Spre exemplu, un bloc care face media tuturor intrărilor nu ține cont de originea datelor.

Blocurile **nu au memorie statică**. Ele pot folosi informații exterioare, cum ar fi timpul curent al zilei, sau alte informații din sistem, însă ele nu au stare internă. Astfel, se poate considera că blocurile reprezintă un element combinațional, și nu unul secvențial.

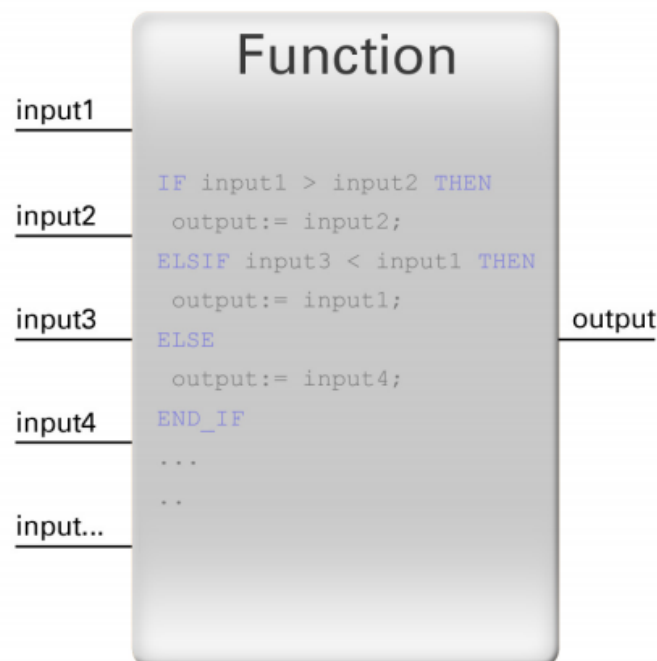


Figura 2.7: Exemplu bloc de procesare

⁴Function Blocks. URL: <http://www.functionblocks.org/index.html> (visited on 08/20/2015).

⁵Programmable controllers - Part 3: Programming languages. IEC. TC 65/SC 65B - Measurement and control devices, July 2013. URL: http://www.dee.ufrj.br/controle_automatico/cursos/IEC61131-3_Programming_Industrial_Automation_Systems.pdf, Appendix C.

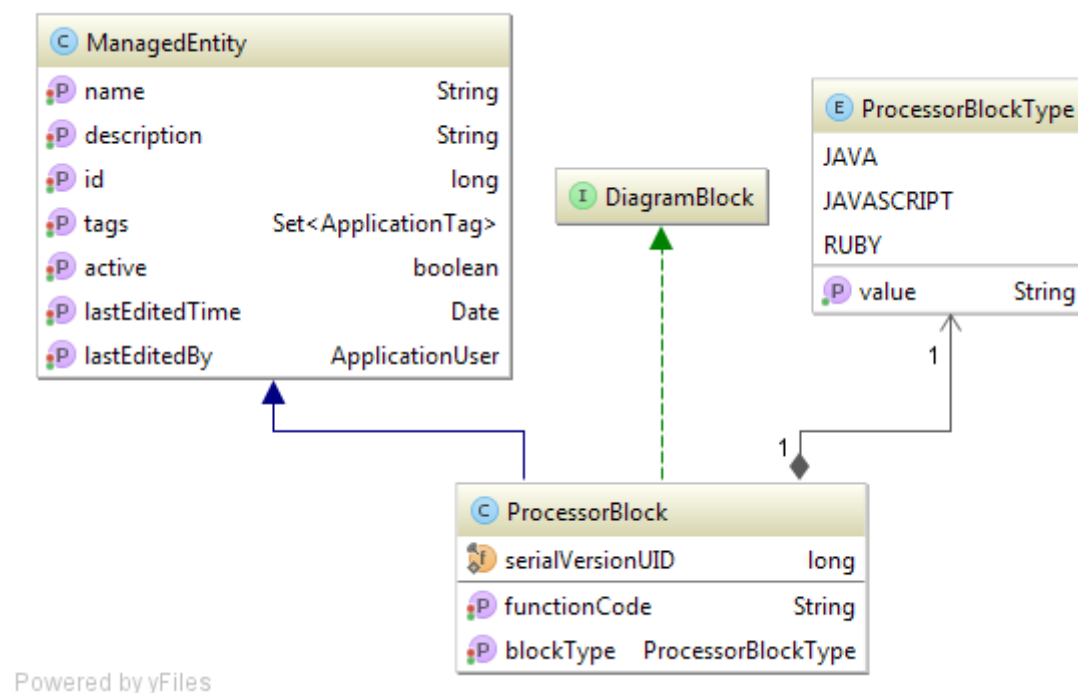


Figura 2.8: Clasa ProcessorBlock

2.2.5 Diagrama funcție bloc(FBD)

Diagramele funcție bloc reprezintă o implementare a unuia din cele patru limbaje de programare pentru automate programabile, specificate de standardul IEC 61131-3:2013.⁶ FBD-ul este un limbaj de control al procesului, în mod normal, toate blocurile de procesare dintr-o diagramă fiind executate. În cel mai simplu caz de utilizare, un FBD realizează următoarele operații:

- Acceptă date de intrare de la unul sau mai multe canale;
- Realizează o operație de transformare asupra acelor date folosind un bloc de procesare;
- Salvează rezultatul pe un canal de ieșire.

Legăturile dintre blocurile de procesare sunt unidirecționale. Un bloc de procesare poate trimite rezultatul la unul sau mai multe blocuri, iar o intrare poate fi conectată la o singură ieșire. Transmiterea de date se face fără întârzieri. Diagramele sunt executate în ordinea definită de ordonarea topologică a nodurilor din graful ce definește diagrama.⁷

⁶Programmable controllers - Part 3: Programming languages, pp. 128-140.

⁷Industrial Use Cases of Distributed Intelligent Automation. IEC 61499. TC 65/SC 65B - Measurement and control devices, Jan. 2011. URL: http://www.vyatkin.org/publ/IES_Mag_1499.pdf, pp. 20-50.

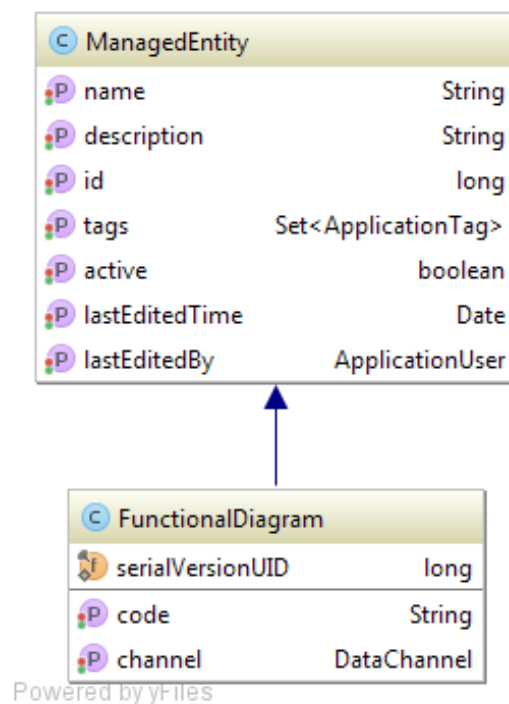


Figura 2.9: Clasa FunctionalDiagram

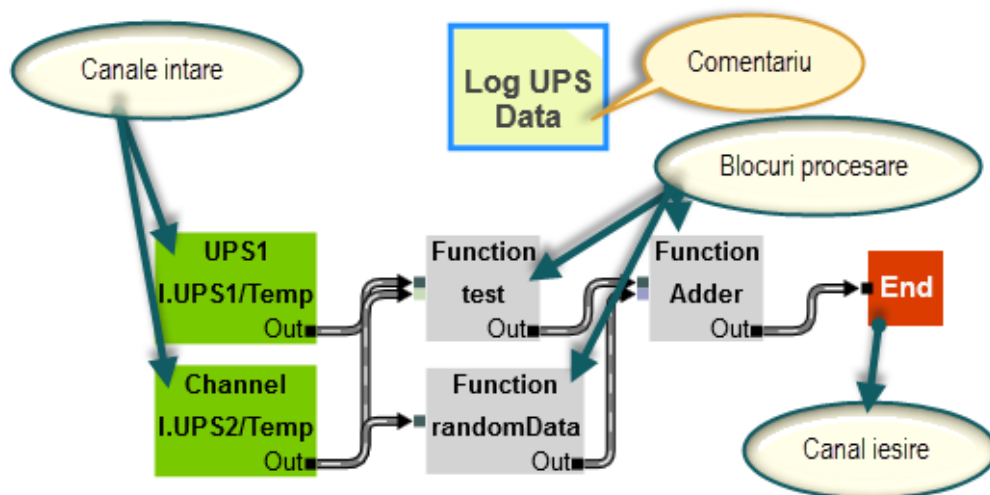


Figura 2.10: Exemplu diagrama funcțională

2.3 Primirea datelor

Primirea datelor se face prin intermediul unei interfețe de transfer a stării (REST). Mai multe formate sunt incluse pentru integrarea mai ușoară cu sisteme deja existente. Astfel,

au fost implementate mai multe procesoare care primesc date atât într-un format special, cât și în formate standard în industrie. Astfel, două modalități de trimitere a datelor există în sistem:

- Trimitere către un singur canal, un singur punct odată: pe baza serviciului */api/put/inputId/channelId/data*. Acest serviciu adaugă un singur punct în baza de date, la momentul curent. Folosit pentru sisteme care trimit date rar, și nu trebuie să se țină cont de data locală de pe device-ul care a trimis punctul de date.
- În formatul standard folosit de openTDSB în care au fost introduse următoarele modificări care păstrează totuși compatibilitatea: metricile reprezintă numele canalului, iar tag-urile sunt opționale. Se acceptă atât formatul în care într-o cerere se află un singur punct, cât și formatul cu o listă de puncte. Canalele dintr-o cerere multidimensională nu trebuie să facă parte din același bloc de intrare. Acest mod de introducere a datelor este sugerat pentru sistemele care folosesc mai multe canale de date și care trimit seturi de date mai mari printr-o singură cerere. Spre exemplu, un dispozitiv poate trimite date de pe mai mulți senzori, și poate stoca local mai multe măsurători pe același senzor pentru a trimite toate datele odată.

Odată primite, noile puncte de date trec prin procesul descris în figura 2.11:

1. Se interoghează baza de date pentru detalii privind canalul ce tocmai a primit date.
2. Dacă un preprocesor există pe canalul specificat, atunci el este încărcat.
3. Se execută preprocesorul cu punctul de date primit
4. Se salvează rezultatul în baza de date.
5. Asincron, se lansează toate diagramele care trebuie să se execute atunci când se primesc date noi pe acest canal.
6. Asincron, se informează ascultătorii că canalul a primit date noi.

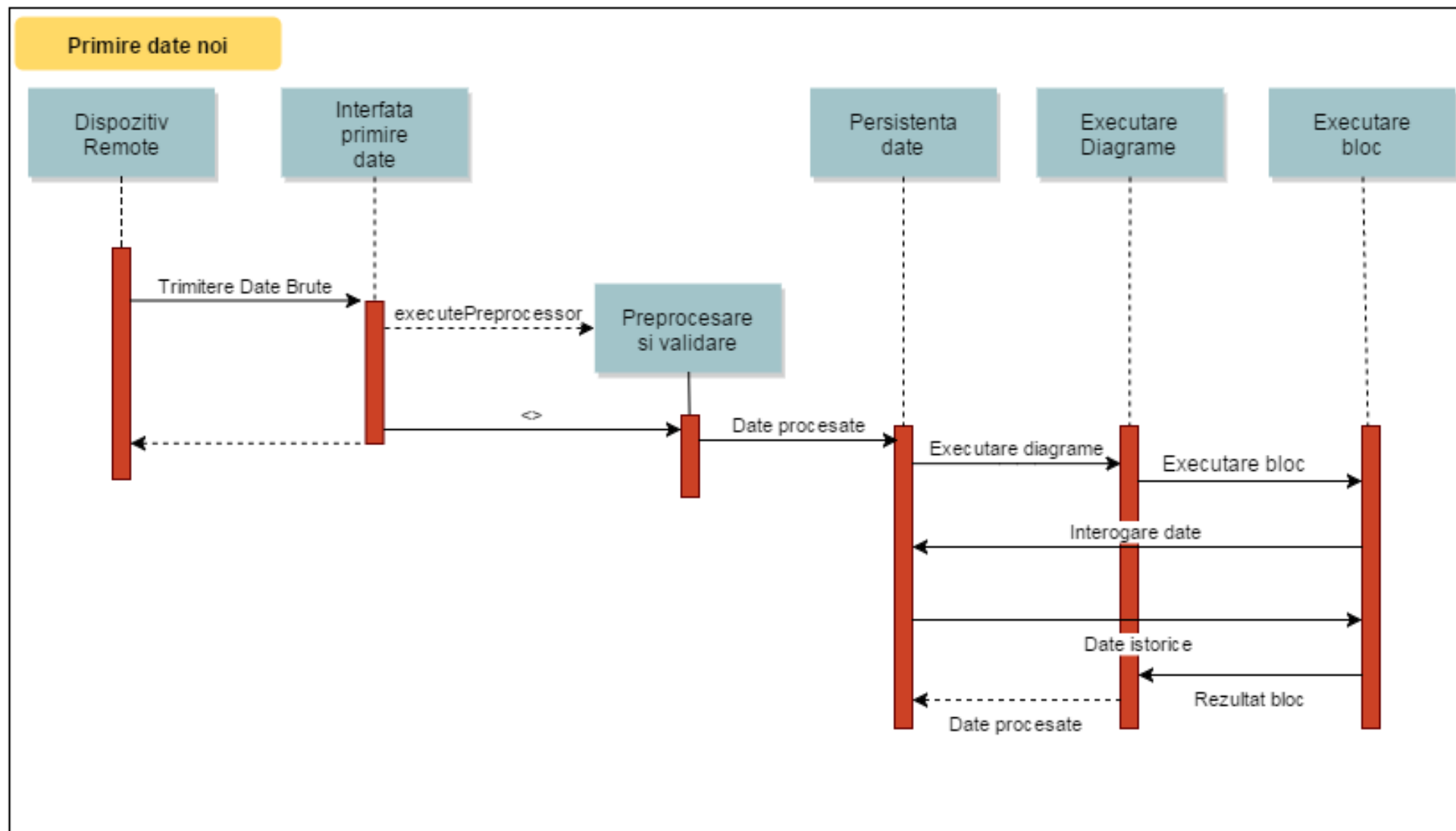


Figura 2.11: Diagrama de secvente pentru introducerea de noi date si execuția diagramelor

2.4 Executarea unei diagrame

Figura 2.11 prezintă și modul în care o diagramă este executată. Diagramele sunt lansate în execuție de fiecare dată când un canal folosit în ea primește informații noi. Dacă două canale primesc date în același timp, atunci diagrama va fi lansată în execuție tot de două ori, pentru ambele puncte de date primite.

2.4.1 Ordinea execuției blocurilor

Problema ordinii execuției unei FBD este intens dezbătută atât în literatură, cât și în aplicațiile industriale. Cum standardul *IEC61131-3*⁸ nu propune o soluție pentru ordinea de execuție, producătorii industriali folosesc metode proprii, de la separarea blocurilor într-un tabel și executarea de la stânga la dreapta, sus în jos,⁹ la definirea manuală a ordinii execuției¹⁰ sau folosind algoritmi care determină automat ordinea de execuție.¹¹

În implementarea din Alegria s-a ales proiectarea unei metode automate pentru depistarea ordinii în care diagrama trebuie executată. Deoarece o diagramă reprezintă un graf aciclic orientat, prima etapă a execuției este transformarea într-un graf reprezentat prin lista de adiacență, unde fiecare bloc de intrare, bloc și de procesare reprezintă un nod. Din această reprezentare se omit blocurile de comentarii. Odată ce transformarea a fost efectuată cu succes se încerca aplicarea unui algoritm de sortare topologică¹² a grafului obținut. Această sortare implică găsirea unei ordini astfel încât pentru orice arc orientat uv de la u la v , u este înaintea lui v . Matematic problema poate fi formulată astfel:

Definiție 1. O *ordine topologică*, notată ord_D , au unui graf orientat aciclic $D = (V, E)$ atribuie fiecărui nod o valoare astfel încât $ord_D(x) < ord_D(y)$ pentru orice arc $x \rightarrow y \in E$.

Mai mulți algoritmi pentru efectuarea unei asemenea sortări există în literatură,¹³ înșă, deoarece grafurile în discuție sunt statice, la care nu se adaugă sau se șterg noduri, s-a ales un algoritm clasic, stabil din punct de vedere numeric descris de Kahn în 1962.¹⁴

Simplificat, algoritmul implementat urmărește următorii pași:

⁸*Programmable controllers - Part 3: Programming languages.*

⁹*TM241 Programming Manual.* URL: <http://www.kongzhi.net/files/download.php?id=8362> (visited on 08/20/2015), p. 11.

¹⁰*Logix5000 Controllers Function Block Diagram Programming Manual.* URL: http://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm009_-en-p.pdf (visited on 08/20/2015), p. 11.

¹¹*GE FANUC Function Block Diagram Lab.* URL: <http://geplc.com/downloads/Labs/GFS-384%20M03%20Function%20Block%20Diagram.pdf> (visited on 08/20/2015), p. 5.

¹²*DAGs and Topological Ordering.* URL: <http://courses.cs.washington.edu/courses/cse326/03wi/lectures/RaoLect20.pdf> (visited on 08/20/2015).

¹³David J. Pearce and Paul H. J. Kelly. *A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs.* Vol. 11. New York, NY, USA: ACM, Feb. 2007. DOI: 10.1145/1187436.1210590. URL: <http://doi.acm.org/10.1145/1187436.1210590>.

¹⁴A. B. Kahn. *Topological Sorting of Large Networks.* Vol. 5. 11. New York, NY, USA: ACM, Nov.

1. Identifica toate nodurile spre care nu vine nici un arc. Valoarea acestor noduri va fi 0. In cazul diagramelor, este vorba de toate canalele de intrare, si de blocurile de procesare care nu au nici o intrare. Dacă aceste noduri nu exista, înseamnă ca graful nu respecta condiția de graf aciclic, deci acesta nu va putea fi executat.
2. Se alege unul din din nodurile găsite mai sus.
3. Se șterge acest nod de valoare unu, împreună cu toate arcele care ies din el.
4. Se repeta pașii 1 si 2 pana când nu mai exista noduri in graf.

Algoritmul descris mai sus rulează in $\mathcal{O}(V + E)$.

Algoritmul 1: Algoritmul lui Khan pentru sortare topologica

Data: Un graf orientat aciclic reprezentat prin lista de adiacenta

Result: Lista nodurilor ordonate topologic

```

1  $L \leftarrow$  Lista goala ce va conține nodurile sortate ;
2  $S \leftarrow$  Lista tuturor nodurilor spre care nu exista nici un arc ;
3 while  $S$  conține elemente do
4   | șterge nodul  $n$  din  $S$  ;
5   | introdu nodul  $n$  in  $L$  ;
6   | foreach nod  $m$  care are un arc  $e$  de la  $n$  la  $m$  do
7     |   | șterge arcul  $e$  din graf;
8     |   | if  $m$  nu mai are arce spre el then
9     |   |   | inserează  $m$  in  $S$ 
10 if mai exista arce in graf then
11 |   return Eroare: Graful are cel puțin un ciclu
12 else
13 |   return  $L$  (graful sortat topologic)
```

Astfel, pentru diagrama din figura 2.10, o posibila ordine de execuție este descrisa in figura 2.12.

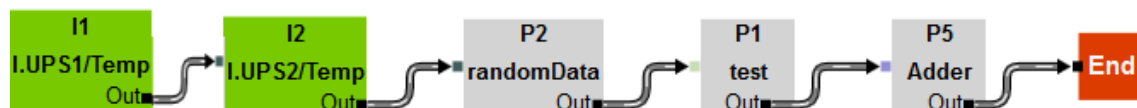


Figura 2.12: Ordinea execuției pentru diagrama din figura 2.10

După cum se observa și în alg. 1 acesta poate detecta grafuri care conțin cicluri și nu pot fi rezolvate. Această verificare permite detectarea cazurilor, precum cel din diagrama 2.13, și permite informarea utilizatorului pentru ca acesta să rezolve problema.

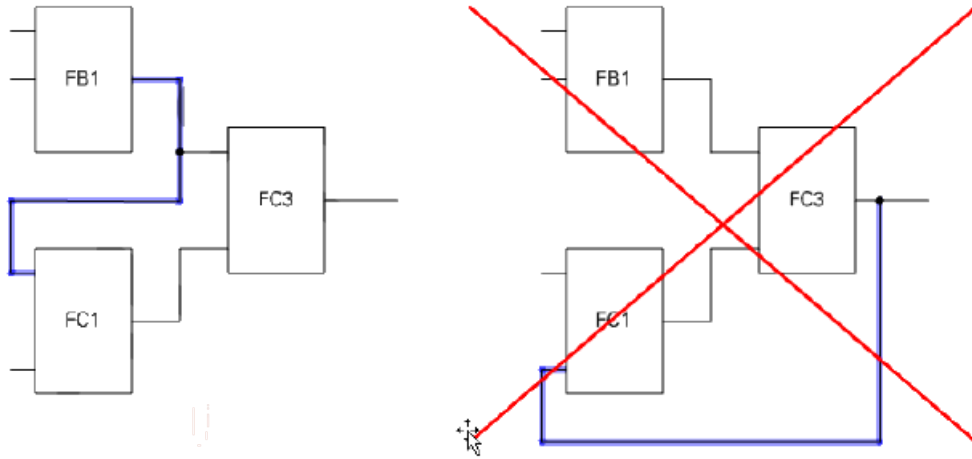


Figura 2.13: Diagrama ce conține cicluri

Deși ciclurile la nivel de diagramă nu sunt permise, cele la nivel de canal sunt. Astfel, ieșirea unei diagrame poate fi setată ca intrare pentru aceeași diagramă, permițând executarea într-o buclă, deoarece odată ce o diagramă se termină de executat și salvează rezultatul pe canal, aceasta se relansează în execuție cu noua informație. Cu astfel de bucle se poate implementa diagrame ce interacționează între ele. Un alt aspect pozitiv al faptului că intrările unor diagrame pot fi ieșirile unor alte diagrame este faptul că procese foarte complexe pot fi descompuse în părțile componente prin simpla înlănțuire a diagramelor.

2.4.2 Generarea rezultatului

Odată ce ordinea de execuție a fost calculată, procesul de calcul al rezultatului este destul de simplu:

Algoritmul 2: Execuția unei diagrame FBD

Data: Lista nodurilor ordonate topologic

Result: Punctul de date ce trebuie adăugat pe canalul de ieșire a diagramei

```
1 rezultate ← Relație cheie-valoare între nod și rezultatul execuției lui;  
2 L ← Lista ce conține nodurile sortate ;  
3 foreach nod m din L do  
4   | intrari ← Lista goală de intrări pentru nodul m ;  
5   | foreach arc de la n către m do  
6     | if In rezultate exista rezultatul pentru blocul n then  
7       |   Adaugă în intrari valoarea de la rezultate(n);  
8     | else  
9       |   return Eroare: Graful nu poate fi executat;  
10  | if m este un bloc de procesare then  
11    |   Execută blocul folosind intrările intrari;  
12    |   Adaugă în rezultate valoarea calculată;  
13  | else  
14    |   else if m este un canal de date then  
15      |   Adaugă în rezultate ultima valoare de pe canal;  
16 return Ultima valoare din rezultate
```

Capitolul 3

Interfața aplicației

Capitolul 4

Implementarea Alegria

4.1 Aplicația Web

Alegria a fost implementată cu ajutorul platformei **Spring Boot**.¹ Platforma a fost aleasă pentru stabilitatea ei excepțională, fiind bazată pe Spring Framework care stă la baza unora din cele mai mari aplicații existente,² dar și pentru ușurința prin care o aplicație poate fi compusă din elemente funcționale, abstractizând peste nivelele de jos a programului, permițând alocarea timpului pe logica aplicației, și nu pe implementarea platformei pe care aplicația să ruleze.

Cum aplicația este bazată pe arhitectura MVC (model-view-controller) aceasta a fost structurată în trei elemente separate:

- **Interfața vizuală:** Realizată în **HTML5**, folosind motorul de templating Thymeleaf pe server și Bootstrap și JQuery în client pentru afișarea paginilor. Această combinație permite realizarea de pagini cu un aspect modern, responsiv, care funcționează atât pe ecranul mare al calculatorului, cât și pe display-ul mic al unui telefon.
- **Modele:** Reprezintă o reprezentare a entităților din baza de date în sistemul Alegria.
- **Controller-e:** Realizează legătura între partea vizuală a aplicației și entitățile din baza de date, asigurând atât metodele care "umple" template-urile cu date, cât și implementarea interfeței API care introduce și extrage date.

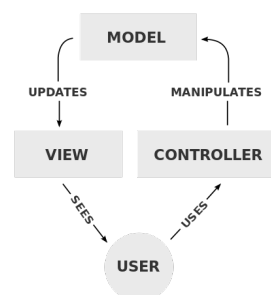


Figura 4.1:
Colaborarea între
componentele MVC

¹Spring Boot. URL: <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/> (visited on 09/02/2015).

²Spring Framework Case Studies. URL: <http://pivotal.io/resources/1/case-studies> (visited on 09/02/2015).

4.1.1 Securitate

Securitatea este un element de baza, atât pentru accesul la date, cat si pentru accesul la entități. Astfel, in implementare s-a folosit framework-ul **Spring Security** care usureaza management-ul securității, fiind puternic integrat si cu restul platformei Spring.

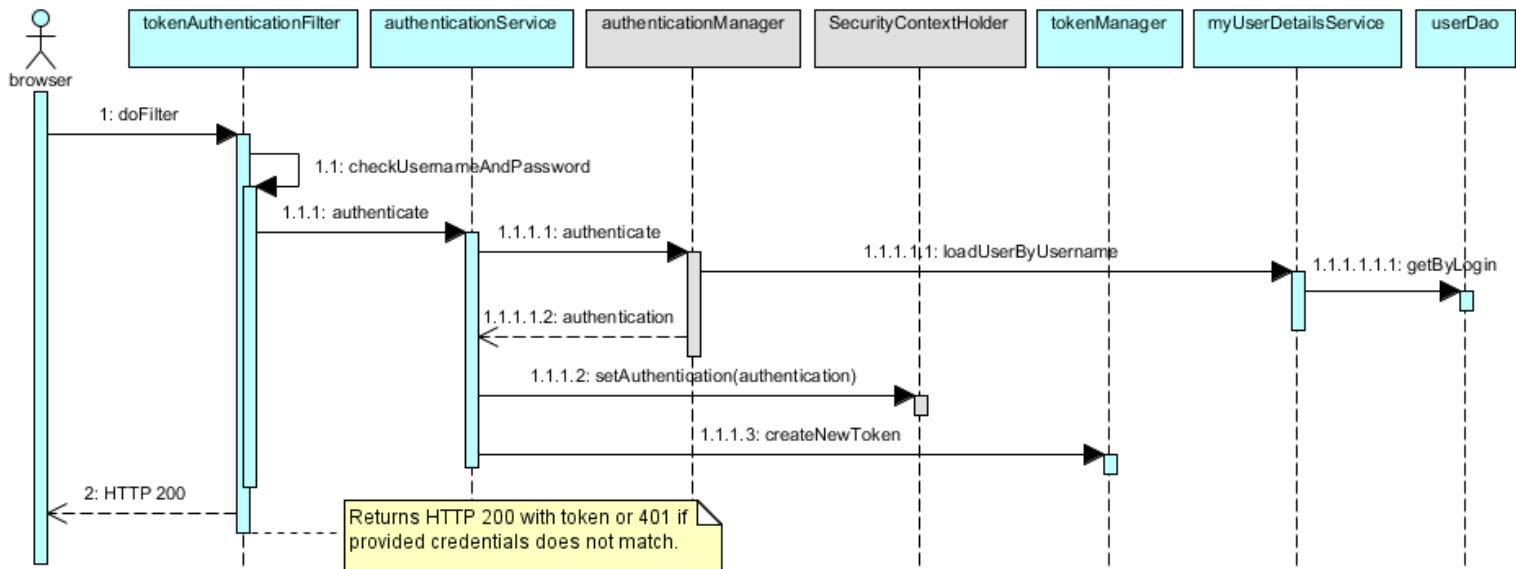


Figura 4.2: Procesul de login in aplicatie

Pentru autentificare a accesa o resursa protejata, utilizatorul va folosi unul din doua mecanisme:

- Autentificare securizata prin username si parola, aceste detalii fiind stocate in tabela *application_user*, unde parola a fost stocata dupa ce a fost trecuta printr-o functie criptografica de hashing. Aceasta metoda de autentificare este folosita pentru autentificarea utilizatorilor in interfață de management si monitorizare. Odată ce procesul a reușit, un token unic va fi generat, iar request-urile următoare vor fi verificate pe baza procesului descris mai jos.
- Autentificare pe baza de token, folosita pentru securizarea API-ului, dar si in cazul in care un user s-a autentificat deja cu username si parola. Fiecare request trebuie sa aibă un token, fie într-un cookie, fie ca parametru in url.

Tot in scopuri de securitate, fiecare entitate care poate fi modificata menține un istoric al tuturor modificărilor, împreuna cu utilizatorul cu care le-a efectuat, iar, pentru o dezvoltare ulterioara, accesul unui utilizator poate fi limitat doar la obiectele care au aceleași tag-uri ca si utilizatorul.

4.1.2 Management

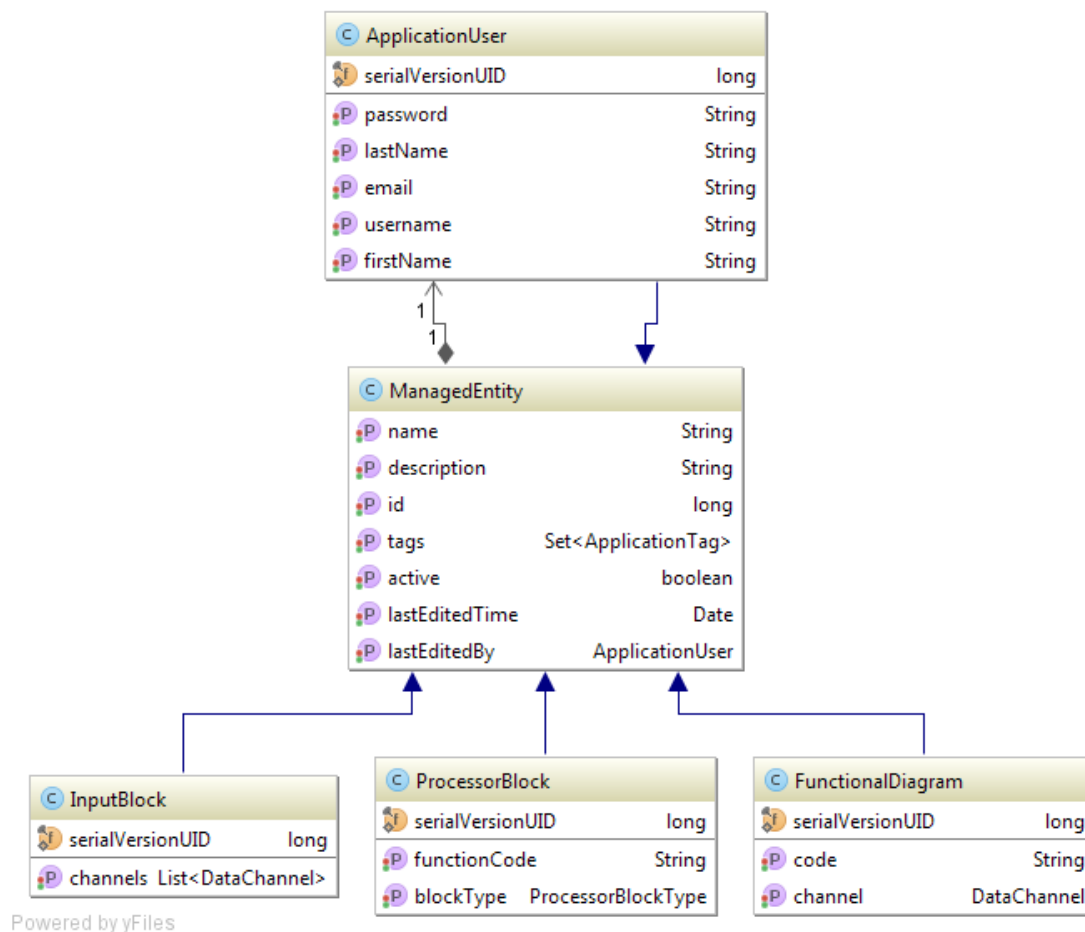


Figura 4.3: Entitățile care sunt administrate de către utilizator și implementează ManagedEntity

Toate entitățile care implementează ManagedEntity permit apoi operații de adăugare, modificare și ștergere. Acest proces de administrare vizuală folosește următoarele resurse, iar un exemplu pentru management-ul :

- Un **repository**, care extinde JpaRepository din framework-ul Spring Data. Acesta asigură operații de căutare, creare, citire, modificare și ștergere a entităților. Un avantaj al folosirii acestui repository, care implementează paradigma Data Access Object (DAO) este că interacțiunea cu baza de date se face într-un mod consistent și sigur, incompatibilitățile de tip fiind detectate la compilare, și nu la rulare. În Alegria, toate repository-urile folosite, împreună cu implementările lor se află în package-ul `ro.pub.acse.sapd.repository`.
- O **vizualizare**, template Thymeleaf, care într-o singură pagină HTML expune către

utilizator toate operațiile suportate de repository. Aceasta pagina este una dinamica, ce folosește dialoguri modale încărcate prin AJAX pentru a edita entități, fără a fi necesar ca utilizatorul sa fie redirecționat către o alta pagina. Entitățile sunt afișate sub forma tabelara, dinamica, care permite sortarea si filtrarea după diverse condiții. Dialogul modal de editare este specific entității care este modificate.

- Un **controller**, clasa cu adnotarea `@Controller`, care leagă repository-ul de vizualizare, dar si specifica către Spring care sunt endpoint-urile (căile pe care acest controller le tratează) prin adnotarea `@RequestMapping`.

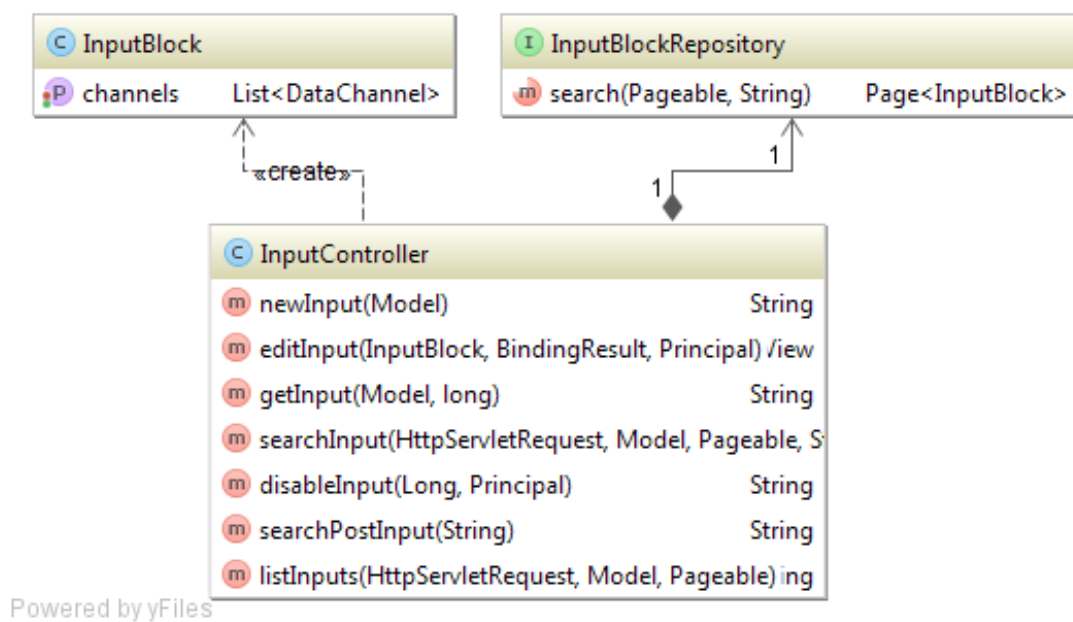


Figura 4.4: Interacțiunea dintre repository, controller si entitate

4.1.2.1 Managementul blocurilor de intrare

Pe lângă elementele descrise mai sus, la management-ul blocurilor de intrare trebuie ca utilizatorul sa poată vizualizeze si modifica lista de canale a unui bloc. In vederea implementării acestei particularități, in dialogul modal pentru adăugare si modificare a fost realizat un formular dinamic cu cate o line pentru fiecare canal de date. Pentru blocurile de intrare care au deja un canale atașate, acest formular este generat de către server, in template-ul `thymeleaf input.html`, iar, dinamicitatea formularului este implementata cu ajutorul unor funcții javascript care manipulează structura documentului.

4.2 Baza de date

Capitolul 5

Studiu de caz: Smart Home

Capitolul 6

Concluzii

Bibliografie

- DAGs and Topological Ordering*. URL: <http://courses.cs.washington.edu/courses/cse326/03wi/lectures/RaoLect20.pdf> (visited on 08/20/2015).
- Function Blocks*. URL: <http://www.functionblocks.org/index.html> (visited on 08/20/2015).
- GE FANUC Function Block Diagram Lab*. URL: <http://geplc.com/downloads/Labs/GFS-384%20M03%20Function%20Block%20Diagram.pdf> (visited on 08/20/2015).
- IEEE Standard for Floating-Point Arithmetic*. Aug. 2008, pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.
- Industrial Use Cases of Distributed Intelligent Automation*. IEC 61499. TC 65/SC 65B - Measurement and control devices, Jan. 2011. URL: http://www.vyatkin.org/publ/IES_Mag_1499.pdf.
- Kahn, A. B. *Topological Sorting of Large Networks*. Vol. 5. 11. New York, NY, USA: ACM, Nov. 1962, pp. 558–562. DOI: 10.1145/368996.369025. URL: <http://doi.acm.org/10.1145/368996.369025>.
- Logix5000 Controllers Function Block Diagram Programming Manual*. URL: http://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm009_-en-p.pdf (visited on 08/20/2015).
- Pearce, David J. and Paul H. J. Kelly. *A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs*. Vol. 11. New York, NY, USA: ACM, Feb. 2007. DOI: 10.1145/1187436.1210590. URL: <http://doi.acm.org/10.1145/1187436.1210590>.
- Programmable controllers - Part 3: Programming languages*. IEC. TC 65/SC 65B - Measurement and control devices, July 2013. URL: http://www.dee.ufrj.br/control_e_automatico/cursos/IEC61131-3_Programming_Industrial_Automation_Systems.pdf.
- Spring Boot*. URL: <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/> (visited on 09/02/2015).
- Spring Framework Case Studies*. URL: <http://pivotal.io/resources/1/case-studies> (visited on 09/02/2015).

The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627. RFC Editor, July 2006, pp. 1–10. URL: <http://www.ietf.org/rfc/rfc4627.txt>.

The Scalable Time Series Database. URL: <http://opentsdb.net/index.html> (visited on 08/20/2015).

TM241 Programming Manual. URL: <http://www.kongzhi.net/files/download.php?id=8362> (visited on 08/20/2015).