Sistem de achiziţie, procesare si distribuite a datelor

Petrişor-Ştefan Lăcătuş
Automatică şi Ingineria Sistemelor
Profesor Coordonator: Andreea Udrea
Facultatea de Automatica si Calculatoare
2015

Cuprins

Capitolul 1

ALkjhlkjhd

jshdlskjhfldksjhflkdjshflkjdshfdlskjhfl kajshflkasjyoias7dlkjshl[1]

[1] Gregory Dix. *The shape of the liturgy*. Bloomsbury Publishing, 2005, p. 234.

Capitolul 2

Da da da da

2.1 kjashdjksa

2.1.1 hoo hoo hoo

ksdjshflkjdshflskjfhlskjhfldksjhfdskjhf ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjd-
shflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhfks
ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksj
ksdjshflkjdshflskjfhlskjhfldksjhfdskjhf ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjd-
shflskjfhlskjhfldksjhfdskjhf ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhf
ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhf

2.1.2 haa haa haa

ksdjshflkjdshflskjfhlskjhfldksjhfdskjhf ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjd-
shflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhfks
ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksj
ksdjshflkjdshflskjfhlskjhfldksjhfdskjhf ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjd-
shflskjfhlskjhfldksjhfdskjhf ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhf
ksdjshflkjdshflskjfhlskjhfldksjhfdskjhfksdjshflkjdshflskjfhlskjhfldksjhfdskjhf

2.2 s28oish0s

sdsdlksjdlksjdfljkshlfkjdshlkjfhds kaslkfjsdlkjfh sjdhkfkjdshfs sdkfjhdsk

sdfkjhdslkjfhdsjkhf

sdfjhdslkjfhdsfdskjhflkjshglkjhlsdkjg

Capitolul 3

Arhitectura soluţiei

3.1 Prezentare generală

Alegria a fost conceputa ca o platforma de dezvoltare rapida, bazata pe modele. Prin folosirea unor metode de programare vizuala, cu blocuri refolosibile in mai multe aplicaţii diferite, utilizatorul poate sa îşi concentreze resursele asupra soluţiei finale, abstractizând detaliile implementării. Aplicaţia a fost construita pe baza unei arhitecturi modulare, cu module cat mai puţin cuplate, care sa permită modificări rapide si testarea modulelor individual. Urmărind aceasta gândire modulare, au fost s-au identificat 4 componente esenţiale:

- **Baza de date** Asigură stocarea datelor, dar si a entităţilor existente in aplicaţie;
- **Interfaţa web pentru management** O interfaţă uşor de utilizat care sa permită utilizatorului să manipuleze canale, diagrame, blocuri de procesare, dar si alţi utilizatori
- **API-ul pentru date** Un API specializat pentru adăugare de date, achiziţionarea datelor, dar si sa permită altor dispozitive sa fie notificate de fiecare data când apar date noi.
- **Elemente de procesare** Asigură procesarea datelor, atât in cadrul blocurilor de procesare, dar si in cadrul diagramelor funcţionale.

In vederea implementării sistemului s-au identificat următoarele elemente componente esenţiale, componente ce reprezinta elementele constructive a sistemului. Acestea, au reprezentare atât in baza de date, ca entităţi, cat si in aplicaţia Java, ca clase. Identificarea acestor elemente s-a făcut pe baza analizei cazurilor de utilizare a produsului, in care s-au investigat metodele prin actorii interacţionează cu sistemul.

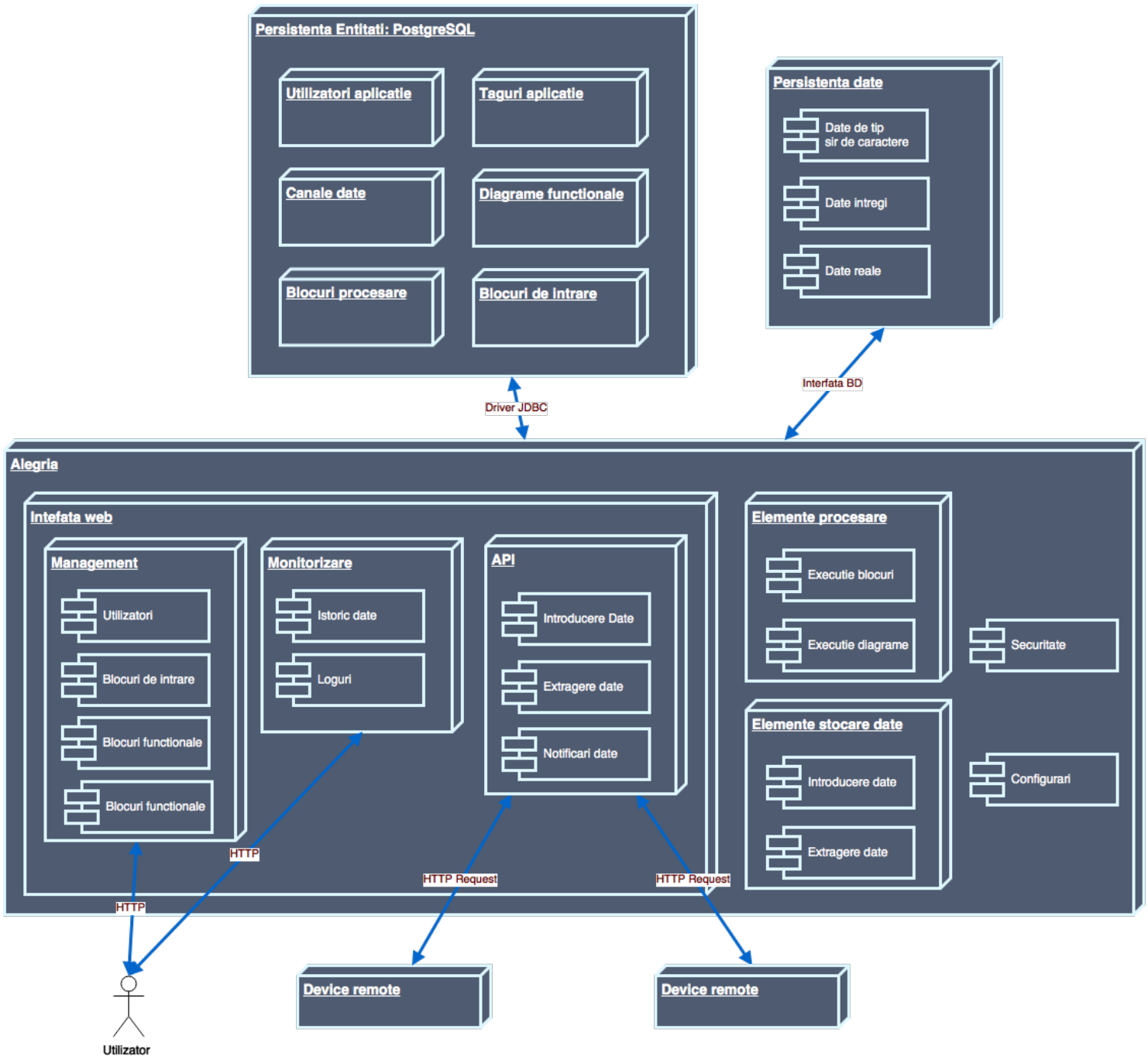

Figura 3.1: Arhitectura generala a aplicaţiei

- **Canalul de date:** reprezinta elementul de baza a sistemului, care asigura recepţia, persistenta si emiterea de date. Datele dintr-un canal trebuie sa respecte un format prestabilit la crearea canalului. Pentru transformarea datelor in formatul stabilit, se poate introduce un bloc de pre-procesarea care transforma datele din un format brut in formatul standard.

- **Blocul de intrare:** elementul de intrare, alcătuit din mai multe canale de date. Blocurile de intrare permit gruparea mai multor canale de date într-o structura unica.

- **Blocul de procesare:** elementul dinamic al aplicaţiei, ce aplica transformări asupra datelor. Un bloc de procesare primeşte ca intrări mai multe canale de date, si are la ieşire un alt canal de date. Utilizatorul poate folosi blocuri standard, existente in sistem, sau poate implementa blocuri noi direct in interfaţa programului.

- **Diagrama funcţie bloc(FBD):** foloseşte blocuri de intrare, canale de date si blocuri de procesare pentru a descrie o funcţie complexa intre intrări si o ieşire. Aceste diagrame folosesc la date aflate pe canale de date, care sunt trimise către blocuri de procesare si, la final se obţine un singur rezultat care este salvat pe un canal de date.

3.1.1 Baza de date

Fiind vorba despre o aplicaţie puternic bazata pe date, aceasta are nevoie de un nivel de persistenta de înaltă performaţă. Urmărind arhitectura propusă din figura 3.1, putem identifica doua cazuri de utilizare pentru baza de date:

- Stocarea modelului entităţilor: fiecare entitate descrisa in lista de mai sus trebuie stocata in baza de date într-o structura relaţională. Entităţile sunt puternic interconectate, iar o baza de date relaţională, de tip SQL este recomandata in acest caz. Din punct de vedere a dimensiunii setului de date, chiar si in aplicaţiile 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 in ce mai multe canale de date. Astfel, stocarea entităţilor nu va aduce probleme de performaţă.

- Stocarea datelor: in baza de date vor fi stocate atât datele primite pe fiecare canal asociat unui bloc de intrare, cat si datele procesate de diagrame. Aceste date au un puternic caracter istoric, reprezentând o serie de timp, in care se retine, pentru fiecare punct de date, valoarea la un anumit moment. Problema stocării acestor date este una mai complicata, datorita necesităţii unei puternice scalari a bazei de date.

Această problemă reprezinta un caz de utilizare pentru o baza de date NoSQL, sau chiar o baza de date specializata in stocarea seriilor de timp.[1]

3.1.2 Aplicaţia Java

Legătura dintre baza de date si utilizatorii finali se face prin intermediul unei aplicaţii Java complexe, care este obiectul acestui proiect. Aplicaţia conţine toată logica platformei, de la operaţii asupra entităţilor din baza de date, la adăugarea, si extragerea datelor, cat si pentru procesarea datelor. Separarea modulelor s-a făcut pe baza scopului acestora:

- **Administrare**: pentru administrarea entităţilor din baza de date. Aceste module permit operaţii de căutare si afişare, dar si de creere, editare si ştergere a utilizatorilor, a blocurilor de intrare si funcţionale precum si a diagramelor. Fiecare dispune de o interfaţă HTML5 in care moderna.
- **Monitorizare**: permit monitorizarea execuţiei aplicaţiei, de la vizualizat loguri pentru a diagnostica probleme, la realizarea de grafice a datelor pe anumite canale. Tot aici este disponibila si funcţia de a exporta date in formate uzuale, ca CSV sau fişiere Microsoft Excel.
- **API**: aplicaţia dispune si de un API pentru a fi folosita programatic de către alte aplicaţii externe. Acesta poate fi considerat ca fiind format din doua componente: serviciile pentru administrarea entităţilor, si cele pentru adăugarea si extragerea datelor.
- **Elemente de procesare**: împărţite in doua subcategorii: cele pentru procesarea blocurilor de intrare si de procesare, si cele pentru procesarea diagramelor.
- **Elemente stocare date**: permit interfaţarea cu sursele de date. Acestea asigura servicii de introducere si extragere a datelor, printr-o interfaţă abstracta, care nu tine cont de modul in care baza de date este implementata.
- **Alte module**: asigura, printre altele securitatea aplicaţiei.

3.2 Entităţi

3.2.1 Punctul de date

Punctul de date reprezinta elementul constructiv al sistemul, care este obiectul procesării, stocării si distribuţiei este punctul de date. Sistemul accepta intern date in 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, foloseşte *Long* pentru reprezentare interna;
- **Real**: numere cu virgula, având dubla precizie, reprezentate cu 64-bit conform standardului[2] IEEE 754 foloseşte *Double* pentru reprezentare interna;
- **Sir de caractere**: Un sir de fără limite a lungimii, care trebuie formatat conform.[3]
- **Obiect**: Un obiect Java serializat in text. Intern, asemănător cu tipul de date String.

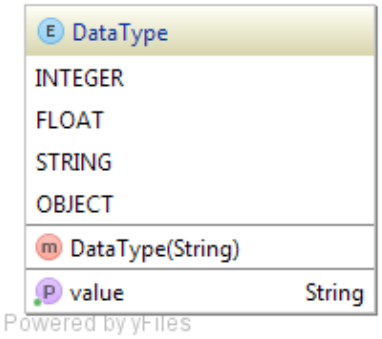

Figura 3.2: Tipurile de date acceptate in sistem

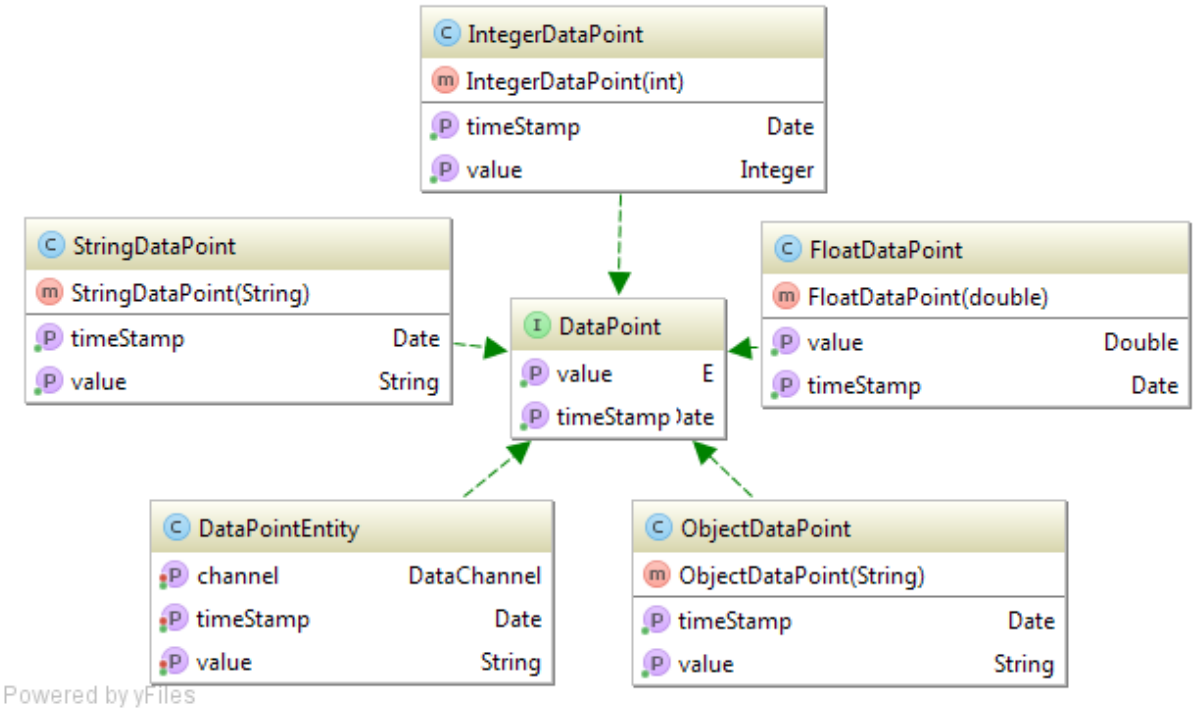

Figura 3.3: Clasele care implementează interfaţa DataPoint

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

[3] *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.

3.2.2 Canalul de date

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

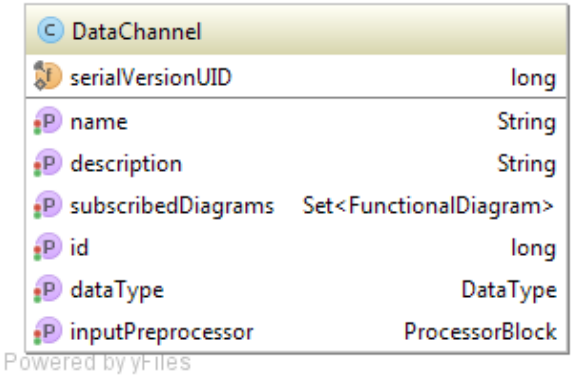

Figura 3.4: Clasa DataChannel

utilizatorul interacţionează cu punctele de date. Pentru a adaugă date noi, un sistem le ataşează unui canal, diagramele folosesc canale atât pentru date de intrare cat si pentru date de ieşire, iar utilizatorii externi pot extrage datele de pe un canal.

3.2.3 Blocul de intrare

3.2.4 Primirea datelor

Primirea datelor se face prin intermediul unei interfeţe de transfer a stării (REST). Mai multe formate sunt incluse pentru integrarea mai uşoară cu sisteme deja existente. Astfel, au fost implementate mai multe procesoare care primesc date atât într-un format special, cat si in formate standard in industrie. Astfel, doua modalităţi de trimitere a datelor exista in sistem:

- Trimitere către un singur canal, un singur punct odată: pe baza serviciului */api/put/inputId/channelId/data*. Acest serviciu adaugă un singur punct in baza de date, la momentul curent. Folosit pentru sisteme care trimit date rar, si nu trebuie sa se tina cont de data locala de pe device-ul care a trimis punctul de date.

- In formatul standard folosit de openTDSB in care au fost introduse următoarele modificări care păstrează totuşi compatibilitatea: metricile reprezinta numele canalului, iar tag-urile sunt opţionale. Se acceptat atât formatul in care într-o cerere se afla un singur punct, cat si formatul cu o lista de puncte. Canalele dintr-o cere

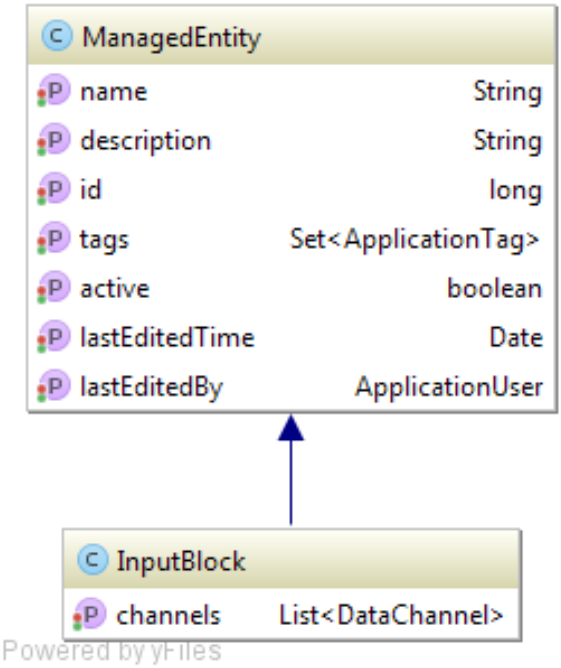

Figura 3.5: Clasa InputBlock

multidimensionala nu trebuie sa facă parte din acelaşi bloc de intrare. Acest mod de introducere a datelor este sugerat pentru sistemele care folosesc mai multe canale de date si care trimit seturi de date mai mari printr-o singura cerere. Spre exemplu, un dispozitiv poate trimite date de pe mai multi senzori, si poate stoca local mai multe măsurători pe acelaşi senzor pentru a trimite toate datele odată.

Odată primite, noile puncte de date trec prin procesul descris in figura 3.6:

1. Se interoghează baza de date pentru detalii privind canaul ce tocmai a primit date

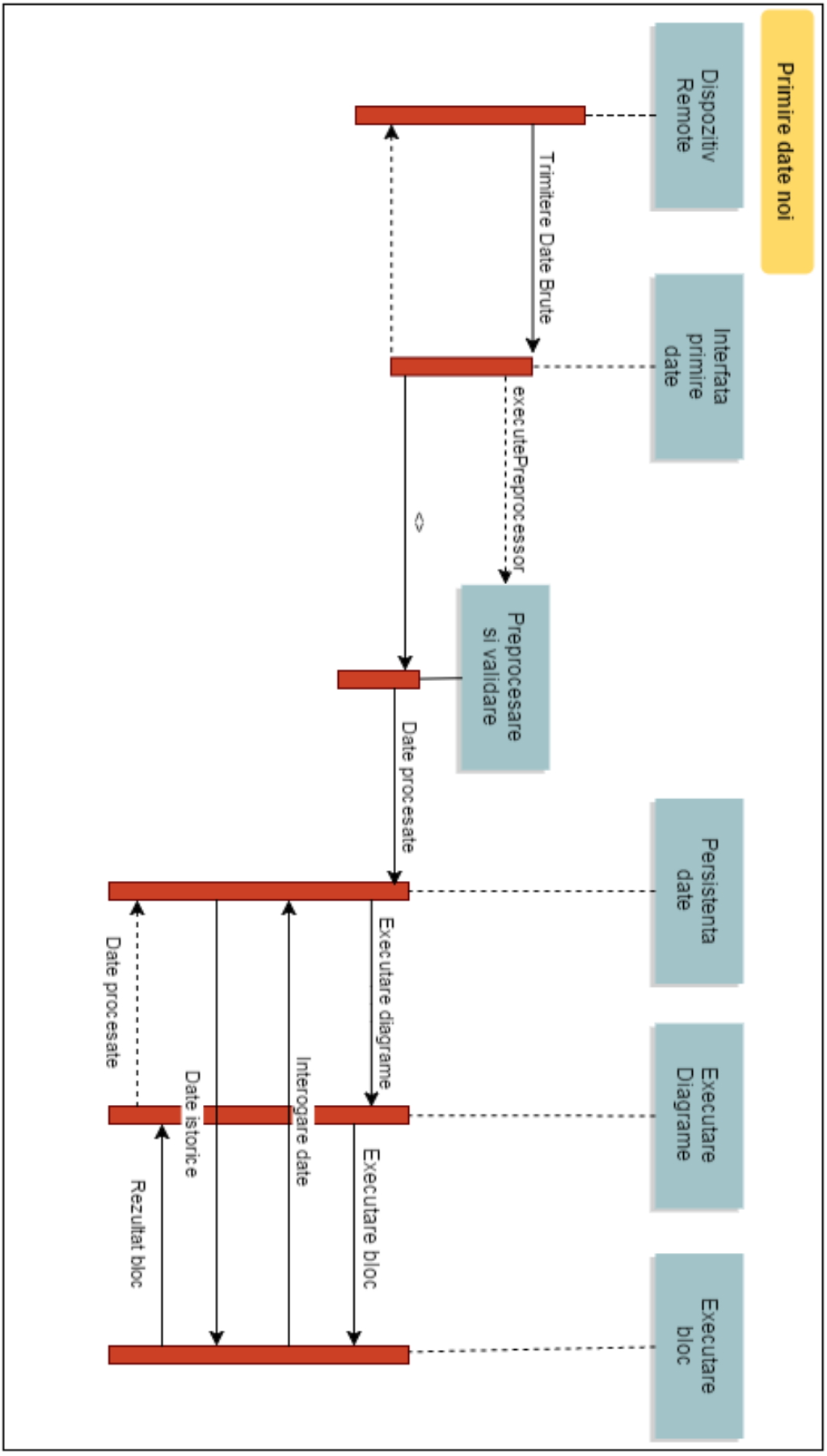

Figura 3.6: Diagrama de secvente pentru intorducerea de noi date

Bibliografie

Dix, Gregory. *The shape of the liturgy*. Bloomsbury Publishing, 2005.

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

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).