Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare

TEZĂ DE DOCTORAT

- rezumat -

Arhitecturi Software Scalabile în Era Cloud

Autor: Vasile M. Tovarnițchi

Comisia de doctorat:

Președinte	Prof. dr. ing. Mihnea Alexandru Moisescu	Universitatea Politehnica din București
Conducător științific	Prof. Emerit dr. ing. Costică Nitu	Universitatea Politehnica din București
Referent	Prof. dr. ing. Nicolae Țăpuș	Universitatea Politehnica din București
Referent	Prof. dr. ec. Ion Smeureanu	Academia de Studii Economice din București
Referent	Prof. dr. ing. George Culea	Universitatea ”Vasile Alecsandri” din Bacău

București, România

2021

Cuprins

Capitolul 1. Introducere

Omenirea influențează, dar și este influențată profund de tehnologiile digitale, care, în mare parte, se bazează pe abordările conexe sistemelor distribuite. Începutul secolului XXI este marcat de o creștere semnificativă a ritmului de inovații și de dezvoltare a tehnologiilor digitale, dar în același timp cresc și cerințele utilizatorilor acelor tehnologii. Acestea, printre altele, se referă la performanțe din ce în ce mai mari, dacă nu chiar cu tendințe de a avea funcționalități ”în timp real”. În acest sens, cercetătorii și inginerii au venit cu propuneri noi de concepte și tehnici specializate pentru manipularea datelor în diverse stadii ale acestora. Necesitatea unor abordări speciale pentru procesarea evenimentelor în contextul unor sisteme distribuite vine ca urmare a nevoii de a prelucra datele imediat ce sunt achiziționate.

Componentele software ale sistemelor informatice nu mai sunt unele rigide, monolitice, care funcționează independent, ci sunt mult mai robuste și flexibile, capabile, în caz de necesitate, de a face schimb de date cu alte componente software, aflate la distanță.

Perfecționarea și tendințele de evoluție a componentelor hardware, maturizarea tehnologiilor Cloud, precum și evoluția algoritmilor și tehnicilor de prelucrare a volumelor imense de date, necesitată elaborarea unor arhitecturi de sisteme informatice care să faciliteze utilizarea optimă și manipularea eficientă a datelor.

Prezenta lucrare vine cu o contribuție în acest sens, fiind propuse câteva tehnici, concepte și abordări de arhitectură software care pot fi aplicate la proiectarea și implementarea unui sistem care implică gestionarea unor fluxuri și volume sporite de date, urmărind următoarele principii de bază: scalabilitate, extensibilitate, reziliență.

Elementele elaborate în cadrul tezei sunt materializate prin propunerea unei arhitecturi software de referință în contextul unui sistem de monitorizare al mediului care a fost confirmată printr-o implementare experimentală.

1.1 Structura Tezei

Prezenta teză este structurată pe 9 capitole.
Capitolul 1 – Introducere
Capitolul 9 – Concluzii, contribuții și direcții viitoare
Bibliografie.

Capitolul 2 cuprinde prezentarea domeniului de arhitectură software, precum și expunerea viziunii autorului privind rolul acesteia în elaborarea unui sistem software din perspectiva asigurării scalabilității și extensibilității sistemului. Sunt descrise și definite, în varianta propusă de autor, elementele din care este compusă o arhitectură a unui sistem software. Se prezintă într-un mod original elementele relevante privind arhitectura software, precum și importanța activităților aferente acestui domeniu asupra unui sistem software pe parcursul întregului ciclu său de viață. Sunt evidențiate detaliile dintr-un sistem software asupra cărora deciziile arhitecturale inițiale au efect determinant privind atributele de calitate ale sistemului, dar în special cele referitoare la scalabilitate și extensibilitate.

Capitolul 3 este dedicat sistemelor distribuite, abordare care nu lipsește în proiectarea unor sisteme informatice complexe moderne. Acesta conține o expunere a modalităților actuale de accesare a resurselor de calcul sub formă de servicii, având la bază concepte și abordări coagulate în jurul tehnologiilor de tip Cloud. Sunt prezentate modalități moderne de operare a resurselor fizice prin utilizare de instrumente software dedicate, dar și tehnici avansate prin care sunt abstractizate infrastructurile de calcul necesare implementării funcționalităților concrete ale componentelor software.

Este prezentată viziunea autorului privind semnificația separării conceptuale dintre componentele software – care, înainte de implementare, presupun elaborarea unei arhitecturi adecvate – și componentele de infrastructură, astfel încât dezvoltarea și operarea componentelor software să fie absolut agnostică față de orice detaliu specific infrastructurii.

Este arătată viziunea autorului asupra tendințelor moderne de proiectare a sistemelor software privind arhitecturile cloud native (eng. *cloud-native architectures*).

Capitolul 4 include detalierea diverselor perspective și modalități utilizate în practică la ora actuală pentru proiectarea și implementarea componentelor software care să asigure o scalabilitate ridicată a unui sistem software. De asemenea, autorul subliniază că un sistem informatic modern trebuie să fie proiectat astfel, încât să poată fi extins și modificat pentru a i se asigura posibilitatea de a evolua în timp. Sunt arătate modalitățile și tehnicile care permit realizarea unor sisteme informatice având la bază arhitecturi software considerate evolutive. Din aceste perspective, este evidențiată necesitatea elaborării unei arhitecturi software, astfel încât anumite aspecte concrete dintr-un sistem informatic care sunt implementate conform unor principii și abordări arhitecturale adecvate determină în mod decisiv principalele atribute de calitate ale acestuia.

Este subliniată importanța abstractizării detaliilor specifice de interacțiune dintre componentele unui sistem software care se reflectă în capacitatea acestuia de a fi scalat și extins. De asemenea, sunt elaborate arhitecturile bazate pe evenimente, care constituie baza implementării unor sisteme informatice moderne cu obiectivele urmărite - care să fie scalabile, extensibile și reziliente.

Capitolul 5 este dedicat detalierii sistemelor software reactive în conexiune cu utilizarea modelului cu actori. Sunt evidențiate abordările specifice prin prisma obiectivelor tezei care permit implementarea unui sistem informatic scalabil, rezilient și extensibil. Este descris *Manifestul Reactiv*, ca o formalizare actuală a principiilor conform cărora poate fi proiectat un sistem software reactiv modern.

Este arătat faptul că modelul cu actori este o modalitate potrivită pentru implementarea unui sistem software extrem de scalabil, rezilient și evolutiv.

Capitolul 6 conține detalierea conceptelor de sisteme inteligente cu evidențierea importanței utilizării unor astfel de abordări într-un sistem de monitorizare. Din nou este pus accentul pe abstractizarea contextului și focusarea pe elementele care reprezintă importanță din perspectiva detaliilor urmărite ale mediului monitorizat. Ca o modalitate evoluată de abstractizare a realității fizice pentru un sistem de monitorizare a mediului, este propusă utilizarea abordărilor privind gemenii digitali (eng. *digital twins*) și a conceptului modern de rețele inteligente digitale (eng. *intelligent digital mesh*).

Având în vedere posibilitățile și punctele forte ale modelului cu actori este propusă modelarea și implementarea gemenilor digitali sub formă de actori.

Este propusă și descrisă o arhitectură software conceptuală pentru un sistem de monitorizare a mediului.

Capitolul 7 conține descrierea detaliată a arhitecturii de referință propusă de autor pentru un sistem software scalabil de monitorizare a mediului. Este continuarea capitolului anterior și include descrierea amănunțită a nivelurilor formulate din perspectiva satisfacerii funcționalităților și cerințelor stipulate la definirea arhitecturii de referință și care implică utilizarea tehnicilor identificate pe parcursul elaborării tezei.

Capitolul 8 este dedicat părții experimentale și prezentării rezultatelor punerii în practică a arhitecturii software propuse. Capitolul conține expunerea unor detalii tehnice de implementare considerate relevante din perspectiva detaliilor elaborate și propuse în cadrul tezei.

Capitolul 2. Arhitectura Software

Sistemele informatice actuale au ajuns la complexități și dimensiuni enorme, iar principalele provocări nu mai sunt determinate doar de algoritmi și structuri de date. Astfel, la proiectarea unui sistem, mai ales care implică utilizarea de componente distribuite specializate, este necesară identificarea și evaluarea unor soluții concrete și abordări specifice domeniului de aplicabilitate care țin de arhitectura viitorului sistem și îi vor determina evoluția, dar și reușita acestuia. În mod particular, sistemele software se află în dependență directă de arhitectura software elaborată pentru acestea în fazele incipiente ale proiectului. Durata de viață, evoluția, eficiența, mentenanța, efortul prin care pot fi făcute modificări în sistem etc., depind de arhitectură.

2.1 Definirea Arhitecturii Software

Elaborarea unei arhitecturi sub diferite forme de reprezentare constituie referința de bază pentru implementarea cerințelor funcționale și siguranța menținerii în timp a construcției în formă operațională. Un sistem informatic complex, are o mulțime de elemente, atribute, proprietăți care trebuie luate în calcul în faza de proiectare, de aceea o arhitectură se poate dovedi a fi in final reușită sau dimpotrivă – un eșec total.

Zările apariției acestui domeniu coincid cu elaborarea lucrărilor lui David L. Parnas [PARNAS1972] care descrie ”modularizarea” ca una dintre metodele de sporire a flexibilității sistemului pe de o parte și eficientizarea timpului de implementare pe de altă parte. Edsger W. Dijkstra a propus modelarea unui sistem prin spargerea acestuia în segmente care să comunice între ele prin mesaje [DIJK1968].

Arhitectura, prin această definiție, conține detalii privind *sistemul, organizarea* și elementele sale *fundamentale* ca formalizare (în sens de abstractizare) a obiectivelor și detaliilor specifice. Sistemul întotdeauna are la bază o arhitectură (*încorporată*), indiferent dacă aceasta a fost definită sau nu a priori.

Prin arhitectura software este accentuată importanța structurii componentelor din perspectiva interacțiunii dintre ele în sens de principii și reguli care vor guverna relația dintre componentele în cauză, dar și evoluția lor și, implicit, al întregului sistem în timp [TOGAF]. Această evoluție presupune atât modificarea componentelor existente ca rezultat al optimizării sau al noilor cerințe, cât și adăugarea de noi componente cu roluri distincte, ceea ce ne duce la proprietăți separate ale sistemelor software – *extensibilitatea* și *evolutivitatea*.

O delimitare corectă a componentelor într-un sistem, precum și o definire clară și cuprinzătoare a interfețelor de intrare și ieșire a acestora, eficientizează etapa de implementare și mentenanță a sistemului, dar și activitățile legate de evoluția lui ulterioară, extensibilitatea și scalabilitatea acestuia.

Procesul de elaborare a arhitecturii software presupune identificarea componentelor relevante și a legăturilor dintre acestea (**Fig. 2.1**), în schimb sunt evitate elementele care nu afectează sistemul ca structură, proprietăți și comportament. Această omisiune a detaliilor ”mai puțin relevante” implică un anumit grad de abstractizare, ceea ce ne permite să simplificăm și să gestionăm mai ușor procesul de elaborare a arhitecturii chiar și pentru sisteme extrem de complexe.

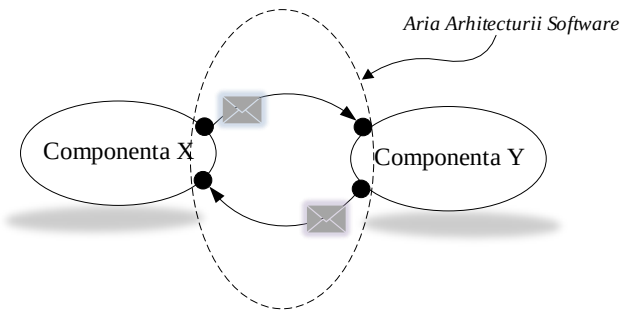

Fig. 2.1 - Aria arhitecturii software

Arhitectura software reprezintă descrierea sub o anumită formă a unui set de decizii strategice, referitoare la elaborarea structurii detaliate a sistemului software în termeni de elemente care abstractizează componentele active ale sistemului, precum și modalitatea și principiile de interacțiune dintre ele pentru a oferi o imagine generală asupra întregului sistem sau a unei părți relevante a acestuia.

2.2 Caracteristicile unei arhitecturi software

Un sistem informatic este format dintr-o o colecție de componente care funcționează în mod colaborativ pentru a atinge un scop bine stabilit, dar care nu poate fi îndeplinit de fiecare componentă în mod independent. Arhitectura se focusează pe elementele care facilitează interacțiunea dintre aceste componente [BASS2003], pe când elementele particulare, interne, care nu influențează exteriorul componentei, nu sunt considerate elemente de arhitectură.

Se consideră că un sistem software îndeplinește cumulativ două tipuri de cerințe:

1) ***funcționale*** – cele care duc la îndeplinirea obiectivelor, adică ceea ce trebuie să facă sistemul, implementând funcțiile cu comportamentul sistemului. Rezultatul implementării cerințelor funcționale este conținut în componentele sistemului.
2) ***non-funcționale*** – cele care nu sunt cuprinse implicit în rezultatul final, dar îl influențează. Aceste cerințe mai sunt denumite și *atribute de calitate* – cum ar fi scalabilitatea, fiabilitatea, disponibilitatea, mentenanța, performanța, extensibilitatea etc.

Cerințele non-funcționale pot fi grupate în două categorii:

- *operaționale* – cele care pot fi urmărite în timpul utilizării sistemului prin diverse metrici – de performanță, securitate etc.
- *evolutive* – scalabilitatea, elasticitatea, flexibilitatea, extensibilitatea etc.

2.3 Arhitectura Software și abstractizarea contextului

2.3.1 Pattern-uri Arhitecturale

Arhitectura software presupune și un aspect ”creativ”, întrucât în mare măsură se bazează pe experiența practică anterioară de proiectare și implementare a componentelor și sistemelor software. Experiența actuală din domeniu arată că după identificarea într-un sistem a elementelor distincte la o granularitate suficient de comună, este foarte probabil ca cineva deja să fi întâlnit și găsit *soluții* la unele *probleme* asemănătoare într-un *context* similar. Astfel, formalizările privind rezolvarea unui tip de probleme, care presupun efectuarea unor pași într-o anumită ordine și pot fi reproduși, au fost denumite *pattern-uri* (din eng. *tipare*).

Pentru că în opinia autorului acestei teze un pattern nu oferă *soluții*, ci indicații pentru a ajunge la o anumită soluție, se poate formula definiția de mai jos.

Un pattern este un set de pași reproductibili, validat și documentat într-o formă consistentă, pentru o problemă comună într-un context asemănător.

Indiferent de problematica acoperită, un pattern trebuie să asigure îndeplinirea cerințelor non-funcționale urmărite ale arhitecturii de sistem, pentru a facilita întreținerea și

evoluția sistemelor complexe. **Pentru un rezultat optim, este foarte importantă congruența dintre obiective, context și soluția aleasă.**

2.3.2 Stiluri Arhitecturale

Spre deosebire de arhitectura sistemului, care este imaginea elementelor esențiale ale sistemului și a legăturilor dintre ele, un stil arhitectural este formalizarea unor aspecte ale anumitor arhitecturi specifice [PERRY1992]. Stilul arhitectural determină implementarea unui anume set de componente și relațiile permise dintre componentele respective, determinând astfel comportamentul sistemului. Conform [KALE2019], ”*un stil arhitectural este bazat pe aranjarea logică a componentelor software*”.

Având în vedere că un stil arhitectural are un scop mai restricționat, arhitectura generală de sistem poate conține mai multe stiluri. Stilurile arhitecturale sunt formalizări care fac abstracție de partea hardware a sistemului, concomitent fiind insesizabile și pentru utilizatorul final. Acestea sunt utilizate de cei implicați direct în implementarea componentelor sistemului software și, în principiu, nu diminuează efortul de implementare, dar cel puțin asigură o coerență conceptuală a detaliilor tehnice ale sistemului software.

2.3.3 Arhitecturi de Referință

Sistemele software moderne sunt construite folosind și componente software dedicate. Având în vedere complexitatea și diversitatea acestora, a apărut nevoia de a gestiona problemele legate de integrare și interoperabilitate. Astfel au apărut abordări formalizate care pot fi utilizate ca referință. Printre acestea se numără *arhitecturile de referință* care reprezintă un model sau o schiță de arhitectură pentru un întreg sistem într-un domeniu particular [ANGEL2012].

Arhitecturile de referință și arhitecturile concrete evoluează și se perfecționează în urma aplicării acestora în practică, existând o relație circulară continuă între ele de-a lungul timpului [BASS2003] (**Fig. 2.2**).

În procesul de evoluție a sistemului efectul modificărilor de la nivelul abordărilor de bază se propagă către abordările finale (**Fig. 2.3**).

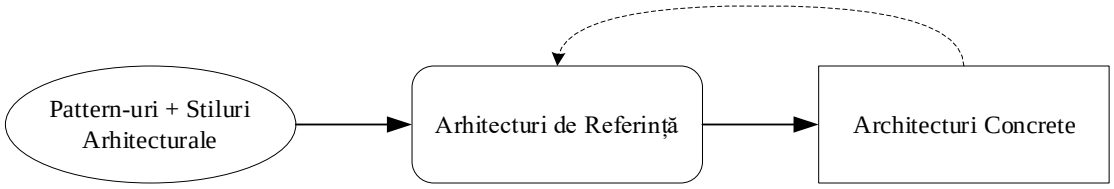

Fig. 2.2 - Relația circulară dintre abordările arhitecturale

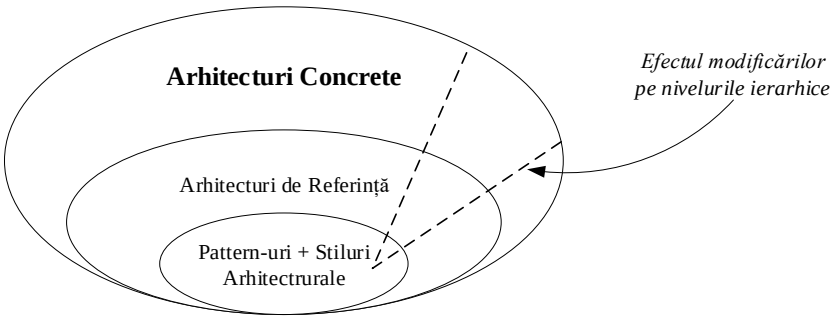

Fig. 2.3 - Abordările arhitecturale și efectul modificărilor

Capitolul 3. Infrastructuri Scalabile. Sisteme Distribuite în Era Tehnologiilor Cloud

Eficiența, performanța, fiabilitatea și funcționarea în general a sistemelor distribuite moderne sunt profund dependente de componentele software care fac pe de o parte posibilă interacțiunea dintre componentele dispersate, dar pe de altă parte vin cu un nivel ridicat de abstractizare a resurselor de calcul (rețea și adrese/locații, protocoale, fire de execuție etc.).

Un sistem distribuit reprezintă o colecție de componente autonome, dar interconectate, care interacționează prin metode specifice și implică partajarea de resurse în mod colaborativ pentru a atinge un anumit scop bine definit.

Resursele unui sistem distribuit reprezintă scopul interacțiunii dintre componente. Acestea pot fi hardware, software sau datele. Componentelor le revine rolul de a permite utilizarea resurselor sistemului într-un mod coerent și transparent, astfel încât utilizatorul final să nu perceapă că respectivele componente sunt divizate și/sau distribuite fizic sau conceptual.

3.1 Aspecte specifice sistemelor distribuite

Menirea sistemelor distribuite este clară și anume acea de a atinge un anumit scop utilizând mai multe componente interconectate între ele, însă acestea ascund și o serie de aspecte [COUL2001], [TAN2007] ”invizibile” pentru utilizatorul final, dar care trebuie abordate cu seriozitate de ingineri din fazele inițiale ale proiectului:

- *Eterogenitatea*. Sistemele distribuite sunt formate din mai multe elemente, care se diferențiază după funcțiile și rolurile pe care le îndeplinesc în cadrul sistemului.
- *Securitatea*. Numărul de puncte și aspecte vulnerabile într-un sistem distribuit este direct proporțional cu numărul de componente distribuite din care este compus sistemul.
- *Fiabilitatea și toleranța la erori*. Lipsa unui singur punct posibil de defecțiune reprezintă un avantaj major cu care vin sistemele distribuite, dar acesta crește complexitatea implementării și mentenanței sistemului, în mod special având în vedere elementul de eterogenitate al componentelor.
- *Extensibilitatea*. O altă caracteristică majoră a sistemelor distribuite reprezintă posibilitatea de a înlocui componentele existente, de a renunța complet la unele sau invers - de a adăuga altele cu funcționalități și roluri noi. Caracteristica respectivă stă la baza conceptului de *Sisteme Deschise*, care presupune că un sistem poate fi extins și modificat în multiple feluri [TOVARNITCHI2012]. Sistemele deschise contribuie la crearea de sisteme mai complexe – sisteme de sisteme [TOVARNITCHI2012], facilitând utilizarea de sisteme eterogene.
- *Scalabilitatea și Performanța*. Performanța este printre principalele caracteristici urmărite și observabile la operarea sistemului, reprezentând un atribut de calitate important pentru toate aplicațiile software [TOVARNITCHI2021].
- *Concurența și partajarea resurselor*. Conceptul de sistem distribuit, implicit, prevede reutilizarea componentelor sau funcționalităților componentelor de către mai mulți ”beneficiari”, dar acest aspect ridică anumite provocări – trebuie asigurată posibilitatea utilizării concomitente sau *concurențiale* ale elementelor vizate.
- *Transparența*. Transparența se referă la ”mascarea” localizării fizice și proprietăților specifice a componentelor față de ”beneficiari” (componente, sisteme externe etc.).

  Alte aspecte ale sistemelor distribuite unde poate fi aplicată transparența, sunt:
  - *Transparența scalabilității* [COUL2001].
  - *Transparența concurenței* [TAN2007].

3.2 Abstractizarea prin virtualizare în mediile digitale

Rezultatul virtualizării este simularea unui dispozitiv sau a unei resurse. Avantajul virtualizării constă faptul că ne ajută să separăm sisteme de aplicații și să partajăm resursele fizice de calcul.

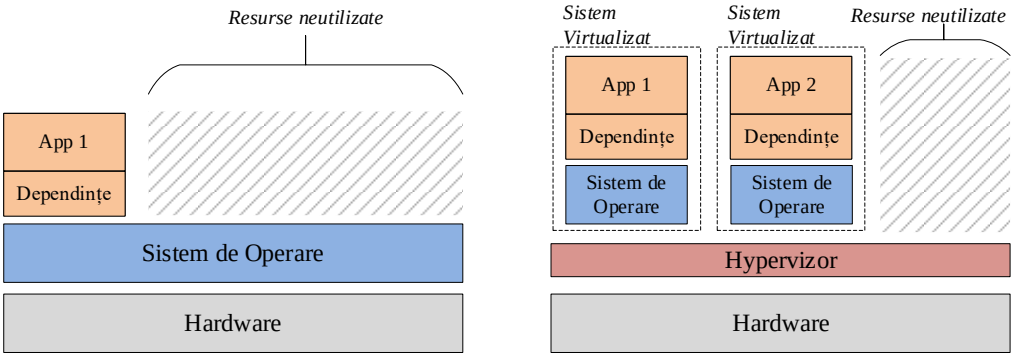

Fig. 3.1 – Virtualizarea sistemelor:
sistem clasic (stânga) versus sistem virtualizat (dreapta)

Datorită elasticității cu care vin serviciile Cloud, prin provizionarea sau de-provizionarea resurselor necesare, sistemelor de aplicații li se ”permite” să se adapteze rapid la solicitările variabile venite din partea utilizatorilor. Posibilitatea acestora de a utiliza un volum variabil de resurse este determinată de o caracteristică esențială a unui sistem software - *scalabilitatea* – care este tratată în detaliu în **Capitolul 4**.

Necesitatea de a avea o infrastructură elastică are la bază două motive principale: *economice* și *tactice*.

Suplimentar resurselor fizice, elementele abstracte dintr-un sistem informatic, precum *datele* și *aplicațiile*, de asemenea pot fi virtualizate.

3.3 Virtualizarea aplicațiilor prin Containerizare. Virtualizarea funcțiilor: soluții Serverless

O formă particulară modernă de virtualizare o reprezintă *containerizarea*, care se concretizează prin utilizarea așa-numitor *containere*. Abordarea implică virtualizarea sistemului de operare, care tehnic presupune partajarea nucleului sistemului de operare [SCHOLL2019].

În contextul actualei teze, ca fundament pentru scalabilitatea componentelor software a unui sistem informatic o considerăm caracteristica privind *elasticitatea* sistemelor distribuite.

Experiența practicilor actuale de dezvoltare de aplicații demonstrează că rezultatele finale sunt mult mai bune atunci când programatorii se concentrează pe scris cod, aplicând sau dezvoltând algoritmi sau tehnici spre soluționarea problemei în cel mai optim și eficient mod, fără să se preocupe de aspectele operaționale a componentelor software dezvoltate.

Una dintre tehnicile actuale de livrare rapidă a unor astfel de servicii scalabile de infrastructură se numește *Serverless Computing*, unde gestiunea resurselor este integral în seama furnizorilor de servicii și este realizată, preponderent, în mod automat. Ideea de bază este separarea dintre aplicația în sine, ca rezultat al efortului de dezvoltare, și partea de operare a infrastructurii necesare. Independența față de platformă, scalarea implicită și neintrusivă, cu accent pe utilizarea optimă a resurselor de calcul, sunt elementele care caracterizează această tehnică.

Tehnica serverless vine cu un nivel suplimentar de abstractizare prin separarea execuției codului (în sens de rezultat final) de locația unde acesta este executat efectiv. Modelul respectiv poate fi asemuit unei arhitecturi reactive (tratată în **Capitolul 5**) bazate pe evenimente (tratată în **Capitolul 4**) [SCHOLL2019], în care execuția funcției este declanșată de un eveniment specific prin apel de API-uri aflate la distanță [SHAHRAD2019].

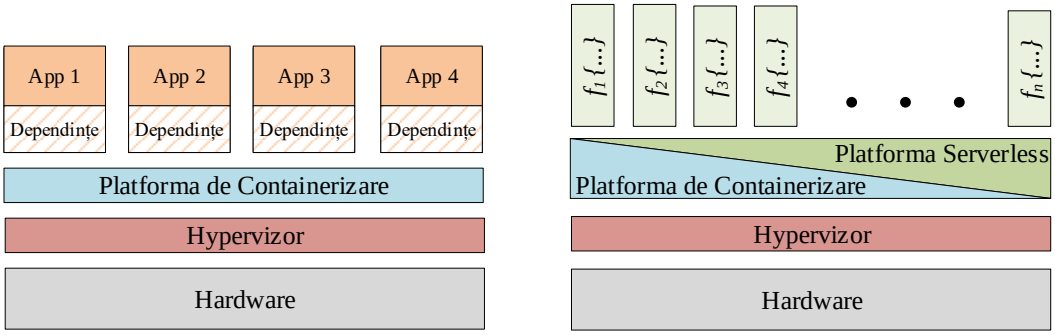

Fig. 3.2 – Virtualizarea aplicațiilor și a funcționalităților: containere (stânga) și model de procesare serverless (dreapta)

Modelele de execuție FaaS sunt specifice fiecărei platforme aparte, dar depind și de abordările selectate pentru implementarea unei funcții concrete. Astfel, poate fi identificată aplicarea de pattern-uri clasice, dar și a unor specifice, elaborate anume în contextul execuției pe modelul serverless [TAIBI2020].

3.4 Arhitecturi Cloud Native

Design-ul de sisteme informatice a fost profund influențat de complexitatea crescândă a cerințelor de business, ceea ce se poate observa în felul în care acestea au fost implementate.

Recent, în contextul dat de diversitatea imensă de tehnologii digitale, s-a formalizat un concept – *aplicații native cloud* (eng. *Cloud-Native Applications - Cloud Native Computing Foundation*[1]), care presupune un set de caracteristici – tehnice și arhitecturale – pe care trebuie să le îndeplinească o soluție informatică pentru a oferi un mod de operare unitar, indiferent de platforma pe care sunt rulate. Astfel de aplicații pot fi rulate în mod identic în cadrul unor soluții Cloud de tip public, privat sau hibrid.

Abordarea cloud-nativă se consideră o tehnică care unifică arhitectura software și arhitectura de sistem.

3.4.1 Green Computing

Metodele clasice de provizionare a resurselor de calcul presupun ”rezervarea” lor a priori și, cel mai adesea, funcționarea lor continuă, cu toate că nu sunt utilizate efectiv pe deplin - conform [BORAH2015], media utilizării anuale este de 10%, ceea ce este extrem de ineficient.

Un aspect tratat cu seriozitate în zilele noastre este abordarea *Green Computing* – utilizarea eficientă a resurselor și, prin urmare, a energiei – reducerea emisiilor de CO_2 și a altor gaze cu efect de seră – ceea ce contribuie la încetinirea încălzirii globale. Un element extrem de important, întrucât se estimează că industria IT&C este responsabilă pentru 2% din totalul emisiilor de CO_2 și îi revine 3% din consumul global de energie [WEBB2008]. Studiile recente arată că spre anul 2025, 4,5% din consumul global de energie va reveni centrelor de date[2]. Conștientizând importanța consecințelor legate de consumul de energie aflat în creștere, marii furnizori de servicii Cloud (Google[3], Microsoft[4], Amazon[5] etc.) se îndreaptă către utilizarea de energie provenită din surse regenerabile, angajându-se să reducă amprenta de CO_2.

Prin urmare, rezultatul aplicării tehnicilor cloud-native are un impact extrem de important atât economic, tehnic, operațional, cât și ecologic.

[1] https://www.cncf.io/ (accesat martie 2021)
[2] https://parkardigital.com/greener-computing-with-serverless-architecture/ (accesat martie 2021)
[3] https://www.google.com/about/datacenters/renewable/ (accesat martie 2021)
[4] https://blogs.microsoft.com/blog/2020/01/16/microsoft-will-be-carbon-negative-by-2030/ (accesat martie 2021)
[5] https://sustainability.aboutamazon.com/environment/sustainable-operations/renewable-energy (accesat martie 2021)

Capitolul 4. Scalabilitatea Sistemelor Software

Îndeplinirea nivelului și calității serviciilor garantat de un sistem informatic depinde în mod direct de atributele de calitate (sau cerințele non-funcționale) stabilite și realizate.

Un atribut important și sesizabil de utilizatorii unui sistem informatic este performanța sistemului. Aceasta poate fi îmbunătățită în mai multe moduri, dar una dintre modalități este scalarea sistemului – a resurselor sau a proceselor.

4.1 Scalabilitatea software. Considerații generale

Scalabilitatea este o ”abilitate” a sistemului care reprezintă un atribut important al sistemelor informatice, des menționat și asumat când vine vorba de caracterizarea și/sau descrierea acestora, mai ales importanța acestuia a crescut în contextul sistemelor moderne – determinate de dimensiuni și cerințe în continuă creștere. Scalabilitatea determină performanța, dar are și o pondere importantă în asigurarea fiabilității, calității serviciilor și extensibilității sistemelor.

Un sistem software îl putem considera ca fiind scalabil, dacă, în urma creșterii solicitării de funcționalități suportate de către componentele constituente, acesta își continuă operarea în parametri normali.

4.1.1 Scalarea pe verticală și orizontală

Scalarea pe verticală presupune creșterea volumului resurselor de calcul implicate (cum ar fi procesoare/nuclee, memorie, spațiu de stocare) per unitate de calcul (fie fizică sau virtuală). Această abordare poate fi denumită și *monolitică* sau având un caracter *static*.

Scalarea pe orizontală presupune alterarea structurii sistemului în sens de adăugare sau înlăturare a unui sau mai multor noduri noi (mașini/unități de calcul fizice sau virtuale), acțiune care va fi transparentă pentru utilizatorul final, fapt pentru care o putem denumi *dinamică*.

Scalarea pe orizontală este efectuată apelând la resurse distribuite, ceea ce scade gradul de dependență față de eventualele erori care pot apărea în timpul execuției anumitor operațiuni în cadrul sistemului.

Scalarea pe verticală	Scalarea pe orizontală
Utilizat preponderent ca suport pentru sisteme software monolitice	Permite utilizarea ”la cerere” a resurselor necesare pentru componente software distribuite
Performanță ridicată, date consistente	Performanță acceptabilă, date inconsistente
Comunicare locală (între procese)	Comunicare la distanță (apeluri prin rețea)
Probabilitate ridicată de comportament eronat în cazul unor defecțiuni (eng. *single point of failure*)	Posibilitate de implementare a unor tehnici care să sporească reziliența sistemului în fața unor defecțiuni
Scalare limitată	Scalare (teoretic) nelimitată

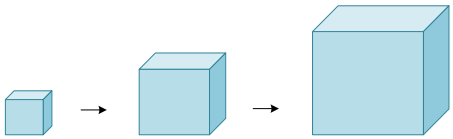

Fig. 4.1 - Scalarea pe verticală

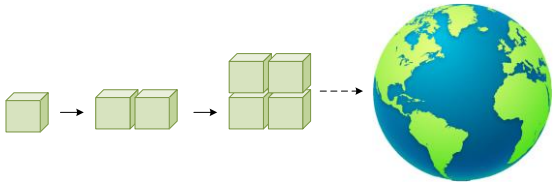

Fig. 4.2 - Scalarea pe orizontală – (teoretic) nu are limite

Scalabilitatea trebuie abordată în fazele de proiectare și elaborare a arhitecturii sistemului, pentru că ulterior, modificările privind gradul de scalabilitate al sistemului, dacă și vor fi posibile – doar cu un necesar de resurse suplimentare (timp, dezvoltatori și alte detalii cu implicații financiare). Este indicat ca în faza de analiză a atributelor de calitate a viitorului sistem, în cazul în care scalabilitatea este identificată ca fiind un criteriu important, să se identifice în sistem acele resurse care ulterior să poată fi redimensionate, iar sistemul să nu presupună explicit utilizarea acelei resurse la o dimensiune fixă [BASS2013].

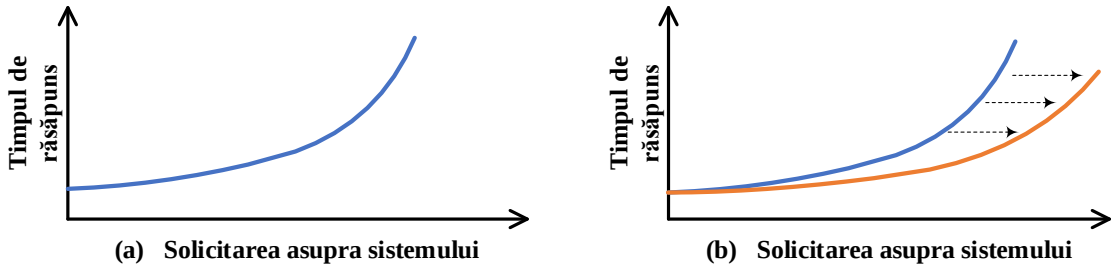

Fig. 4.3 - Cerere - timp răspuns

Odată cu creșterea solicitării asupra sistemului, crește și timpul de răspuns al acestuia - timpul în care sunt procesate cererile (**Fig. 4.3 / a**). Această creștere devine abruptă la un moment dat, punct în care performanța sistemului scade semnificativ, putând să se ajungă la situațiile în care unele cereri sunt procesate cu întârziere, cu erori sau chiar deloc.

Prin contrast, în cazul scalării sistemului, capacitatea acestuia crește în ceea ce privește capabilitatea de a face față unui număr crescut de cereri (**Fig. 4.3 / b**), fapt reprezentat prin deplasarea curbei care relevă relația solicitare/timp-răspuns spre dreapta și menținerea funcționării sistemului în parametri acceptabili.

4.2 Scalabilitatea în format 3D

Una dintre limitările unui sistem software monolitic este scalabilitatea acestuia. **Limitările unui sistem informatic sunt conținute în arhitectura acestuia – felul în care sunt concepute și definite componentele constituente, precum și integrarea dintre ele.**

Abbott și Fisher au popularizat un model de abordare a scalabilității pe trei dimensiuni – x, y, z – fiecare bazându-se pe particularități distincte privind scalabilitatea. Formal, acest model a fost denumit *Cubul Scalabilității* (eng. *Scale Cube*) [ABBOTT2009].

Urmând abordarea propusă în modelul *Cubului Scalabilității*, putem spune că metodele de scalare de asemenea sunt multiple, iar scalabilitatea unui sistem informatic depinde în mod direct de arhitectura acestuia (vezi **Capitolul 2**). Fiecare dintre cele trei direcții de scalabilitate pot fi aplicate concomitent, pornind de la obiectivele urmărite și ținând cont de avantajele și dezavantajele fiecărei metode. Astfel, în cazul scalabilității pe *axa z*, putem avea multiple instanțe identice – scalabil pe *axa x*, iar sistemul în acest caz este scalabil pe direcția *(x, z)*. Putem avea instanțe de microservicii (specializate) – scalare pe *axa y*, concomitent cu instanțe replicate ale acelorași microservicii, însă dedicate pentru seturi separate de date – scalare pe *axa z*, iar sistemul în acest caz îl considerăm scalabil pe direcția *(y, z)*. Punctul cu coordonatele *(0, 0, 0)* este reprezentarea unui sistem care nu este deloc scalabil; punctul opus, în schimb, reprezintă scalabilitatea ”infinită” - putem afirma că sistemul este scalabil pe toate axele.

4.3 Scalabilitatea și integrarea componentelor distribuite

Factorii de bază care diferențiază numeroasele arhitecturi de sisteme informatice, se află în primul rând în particularitățile de integrare dintre componente, iar în cazul unui sistem informatic implementat, eficiența integrării dintre componente determină în mare parte atributele de calitate esențiale ale acestuia, întrucât funcționarea componentelor în cadrul

sistemului depinde de eficacitatea ”colaborării” dintre ele. Importanța aspectului integrării dintre componentele sistemului, dar și integrarea cu componente externe sistemului, este dată de specializarea pe funcționalități a componentelor și, implicit, creșterea eterogenității acestora.

Arhitectural vorbind, integrarea dintre componentele unui sistem are ca scop schimbul cooperativ de date dintre acestea sub diverse forme și se referă la colaborarea dintre ele, reprezentată prin: (i) legăturile dintre componente (în sens de dependență), (ii) modalitatea de interacțiune și (iii) restricțiile (în sens de reguli) aplicate comunicării dintre componentele respective.

Legăturile (i) dintre componente pot fi de două tipuri:

1. *Statice*. Legăturile le numim statice, atunci când relația dintre componente este explicit definită în implementarea sistemului. În astfel de relații, dependența dintre componente nu poate fi modificată decât cu alterarea aplicației.
2. *Dinamice*. Dinamice numim legăturile care pot fi modificate oricând în timpul execuției. Astfel de legături sunt utilizate preponderent în medii distribuite. Legăturile dinamice sporesc flexibilitatea sistemului prin funcționarea independentă și modificarea separată a componentelor.

Interacțiunea (ii) dintre componente se poate realiza în două moduri:

1. *Sincron*. Presupune că ambele componente sunt active în același timp, iar componenta inițiatoare a interacțiunii rămâne ”în așteptare” până primește un răspuns de la cealaltă componentă. Această metodă implică și o dependență temporală dintre componente.
2. *Asincron*. Interacțiunea dintre componente se poate realiza separat în timp. Această modalitate permite executarea paralelă a sarcinilor, decuplarea în timp a componentelor, creșterea performanței și robusteții sistemului.

Restricțiile (iii) privind comunicarea dintre componente pot fi diverse și țin în primul rând de specificul aspectelor funcționale ale sistemului, dar și de legăturile și modalitatea de interacțiune dintre componente. Acestea se reflectă în protocoalele și/sau semantica prin care este înfăptuită comunicarea.

Interoperabilitatea putem considera că are două aspecte, respectiv:

1. *Sintactic.* Detaliile privind structura și formatul mesajelor trebuie să fie agreată de componentele implicate în comunicare.
2. *Semantic.* Este importantă interpretarea corectă a conținutului și înțelegerea semnificației mesajului într-un anumit context.

Interfața unei componente este specificația sintactică a intrărilor și ieșirilor.

În sens tehnic, interfețele reprezintă elemente structurale ale componentelor, având rol de porți de intrare/ieșise cu specificații clare și exacte, care pot fi identificate și localizate și facilitează integrarea componentelor prin schimbul colaborativ de date/mesaje conform unor reguli cunoscute și agreate reciproc.

Componentele sau modulele unui sistem informatic au rolul de a *încapsula* anumite funcționalități. Aceasta presupune că detaliile care țin de logica internă sunt ținute ascunse, deci, din această perspectivă, din exterior, componentele sunt văzute ca niște cutii negre (eng. *black boxes*), iar funcționalitățile sunt expuse către exterior prin apelul *interfețelor*.

Componentele unui sistem distribuit pot fi considerate, ca fiind **distribuite în spațiu**.

În cazul modului de interacțiune asincron, când procesarea mesajelor se poate realiza diferențiat în timp, putem afirma că avem componente **distribuite atât în spațiu, cât și timp**.

4.4 Evenimentele ca declanșatoare a interacțiunii dintre componente și semnalizatoare de schimbări

Ca alternativă față de invocarea explicită [GARLAN1993], cum ar fi metoda *cerere-răspuns*, Garlan and Shaw menționează o altă tehnică de integrare, pe care o numesc invocare implicită, având la bază, pe de o parte, ”anunțarea” unui sau mai multor *evenimente*, iar pe de altă parte ”anunțarea interesului” sub formă de *subscripții* față de respectivul tip de evenimente.

Un eveniment reprezintă o abstractizare a unui fapt împlinit care semnalizează modificarea cuantificabilă a stării unei entități – fie fizică, fie digitală – care are loc într-un anumit context.

Evenimentele pot fi privite ca fiind modalități de reprezentare discretă a semnalelor în ceea ce privește modificările petrecute într-un anumit mediu (din lumea reală, dar și digitală). Astfel, detaliile relevante privind contextul modificărilor sunt modelate ca entități digitale și transmise sub anumite forme de reprezentare ca notificări către părțile interesate. Respectivele entități vor avea o semantică specifică și o modalitate particulară de transmitere. Trebuie să evidențiem, că evenimentul reprezintă importanță într-un sistem de monitorizare nu doar ca un semnal care vine să notifice strict modificarea valorii unei caracteristici urmărite, ci are o semnificație într-un anumit context. Acest context poate fi delimitat în primul rând prin îmbogățirea detaliilor aferente evenimentului cu informații privind timpul și locul apariției acestuia. Respectivele detalii joacă un rol extrem de important pentru corelarea evenimentelor cu o serie de alte evenimente relevante (în context) și la analiza lor.

4.4.1 Arhitecturi bazate pe evenimente

Majoritatea caracteristicilor sistemelor informatice, pentru a satisface la nivelul cerut necesitățile utilizatorilor, sunt determinate de atributele de calitate a sistemului, care depind nemijlocit de arhitectura acestuia.

Construind logica de comunicare în jurul evenimentelor (fiind în esență de tip asincron), fiecare componentă poate să-și ducă la îndeplinire sarcinile în cel mai bun mod posibil, independent și fără sa fie blocate, iar în cazul unor eventuale erori, acestea sunt izolate local.

Sistemele cu o arhitectură bazată pe evenimente, sunt compuse din componente decuplate pentru procesarea evenimentelor, de regulă având scopuri destul de restrânse și bine delimitate și care recepționează și procesează asincron evenimentele [RICHARDS2015].

Cuplarea slabă care este oferită de abordarea construită pe interacțiunea bazată pe evenimente, vine cu o serie avantaje printre care scalabilitatea sporită a sistemului.

Acțiunile consumătorului fiind reacții la anumite evenimente semnalizate de către producător, acestea reprezintă baza unui comportament reactiv (vezi **Capitolul 5**). Sistemele reactive sunt considerate ca au arhitecturi bazate pe evenimente [LAIGNER2020].

Interacțiunea pe bază de evenimente reprezintă una dintre metodele esențiale de schimb de informații în contextul tehnologiilor Cloud, acestea asigurând soluțiilor o agilitate și

scalabilitate crescută. Funcționalitățile oferite de furnizorii de servicii de tip Cloud se bazează, într-o măsură semnificativă, pe soluții bazate pe arhitecturi de acest tip.

Arhitecturile software, în care acțiunile specifice ale componentelor sunt declanșate ca reacție la un eveniment sau serie de evenimente, le denumim arhitecturi software bazate pe evenimente.

Sintagma ”bazat pe evenimente” este strâns legată și de interacțiunea dintre componente și se referă la faptul că activarea componentelor și, respectiv, producerea unor reacții, este consecința apariției unor anume evenimente. Componentele emițătoare de evenimente funcționează independent de consumători, ba chiar nu cunosc destinatarul acestora. **Evenimentele facilitează decuplarea în timp și spațiu a componentelor implicate în comunicare**, iar procesarea acestora nu este condiționată de interacțiune [DEBS2006]. Astfel, în arhitecturile unde interacțiunea este bazată pe evenimente, se pot interconecta componente eterogene, dar orientate pe îndeplinirea unor anumite sarcini specifice și distincte, care funcționează independent, iar ca rezultat să obținem sisteme distribuite complexe, scalabile și extensibile.

4.4.2 Provocările ridicate de adoptarea abordărilor arhitecturale bazate pe evenimente

- ***Consistența datelor*.** Procesarea asincronă a evenimentelor și cuplarea slabă a componentelor, ceea ce aduce un control reciproc scăzut al interacțiunii, poate avea consecințe negative asupra funcționării corecte a sistemului din perspectiva tratării corecte a evenimentelor.
- ***Complexitatea arhitecturală*.** Procesarea fiabilă a tuturor evenimentelor presupune utilizarea mai multor tipuri de componente specializate, fiecare având rol separat de-a lungul ciclului de viață al acestora.
- ***Interoperabilitatea*.** Fiind dată eterogenitatea componentelor folosite în cadrul arhitecturii unui astfel de sistem, se impune adoptarea unei serii de abordări care, pornind de la context, să înlesnească cooperarea dintre componentele implicate în interacțiune în cel mai facil mod.
- ***Non-determinismul*.** Sistemele bazate pe evenimente sunt considerate și sisteme reactive [LAIGNER2020], iar pentru că cele din urmă sunt caracterizate ca fiind în mod inerent non-deterministe [KAISLER2005], putem afirma că și cele bazate pe evenimente vin cu un grad ridicat de non-determinism.
- ***Calibrarea resurselor.*** În cadrul procesului de operare a unui sistem bazat pe evenimente intervin două necunoscute, soluțiile pentru care, de fapt, reprezintă și câteva dintre plusurile importante ale arhitecturii unui sistem. Prima se referă la faptul că volumul sau debitul fluxului de mesaje percepute de sistem nu poate fi determinat a priori, la proiectare. Însă, la elaborarea arhitecturii sistemului, ținând cont de aceste detalii greu de estimat, se impune ca, modulele din aplicație care pot fi afectate în mod direct, să fie proiectate având în vedere tehnici precum ***elasticitatea*** și ***scalabilitatea***. A doua necunoscută este dată de evoluția sistemului în timp – ***arhitecturile evolutive***, determinată de apariția noilor tipuri de evenimente care atrag după sine necesitatea de noi funcționalități din sistem pentru a fi tratate în mod adecvat. Această necesitate este acoperită de o caracteristică pe care o vom denumi ***extensibilitate***, care presupune ***evolutivitatea*** sistemului în termeni de adăugare sistemului de funcționalități noi sau de modificare a unora existente.

4.4.3 Interoperabilitatea bazată pe evenimente dintre componente distribuite. Integritatea și consistența datelor/informațiilor

Un detaliu important care trebuie avut în vedere la proiectarea integrării este recurgerea la utilizarea unei structuri reutilizabile în ceea ce privește semantica informațiilor implicate în interacțiunea dintre componente. Această structură face posibilă integrarea și colaborarea facilă și flexibilă dintre toate componentele din cadrul sistemului. Reprezentând o formă agreată de comun acord de toate părțile implicate în comunicare, poate duce la o colaborare fluentă, absolvindu-ne de necesitatea gestionării unor structuri multiple de date. Utilizarea unei forme standardizate reduce timpul de luare a deciziilor și de întreprindere a acțiunilor, nefiind nevoie de translatarea structurilor de semnalizare a evenimentelor emise de producător. Putem observa un volum redus de date transferate prin canalul de comunicare și o utilizare optimizată a spațiilor de stocare (în sens de volum redus de detalii necesare pentru descrierea structurilor, dar și de informații propriu-zise) în cazul în care acestea necesită persistare.

Într-un sistem de monitorizare a mediului, accentul esențial este focusat pe observarea dinamicii componentelor din care acesta este format ca întreg. Reformulat, monitorizarea presupune observarea modificărilor în timp a anumitor atribute sau proprietăți ale unor componente. Modul de semnalizare a modificărilor este abstractizat sub formă de emitere de evenimente ca modalitate de prezentare a informațiilor relevante privind ceea ce s-a întâmplat. Din această perspectivă, evenimentul este semnalizarea unei modificări care a avut loc în trecut, deci a unui fapt împlinit, care trebuie acceptat și tratat ca atare prin declanșarea unei reacții (acțiuni) sau ignorat.

Un eveniment în sine, strict ca purtător al unei informații, orice semnificație ar avea aceasta, nu reprezintă nici o valoare, de aceea sunt importante detaliile privind contextul în care respectivul a avut loc. Contextul este reprezentat, în primul rând, prin *momentul* și *locul* în care evenimentul a fost produs [CRISTEA2013]. Aceste meta-date sunt esențiale într-un sistem de monitorizare pentru plasarea evenimentului în *timp* și *spațiu*.

Locul producerii (*spațiu*) evenimentului este reprezentat prin identificatorul producătorului de eveniment (care poate fi o referință, o denumire unică etc.).

Momentul de timp (*timp*) în care a avut loc evenimentul este furnizat de către producător sau, în lipsa valorii, prin convenție, este considerat momentul în care este preluat de către consumător.

Un atribut important al evenimentului este **identificatorul unic** al acestuia. Luând în considerare că un eveniment traversează mai multe componente până ajunge să fie ”consumat”, este esențială posibilitatea de identificare exactă și unică a acestuia.

Alt atribut indispensabil pentru distribuirea și procesarea evenimentului este **tipul de eveniment**. Evenimentele de același tip au aceeași structură și semantică și, de regulă, semnalizează același fel de modificări.

S-a propus utilizarea specificației ***CloudEvents***[6] pentru reprezentarea detaliilor evenimentului. O specificație unitară, precum CloudEvents, facilitează interoperabilitatea dintre componente software distribuite prin faptul că oferă o schemă de reprezentare cu o structură comună pentru consumătorii și producătorii de evenimente, făcând abstracție de tehnologia utilizată pentru implementare. În contextul numărului mare de oferte de servicii Cloud și a diversității imense de componente și soluții software dedicate, o abordare comună în ceea ce privește reprezentarea, rutarea și tratarea evenimentelor permite integrarea simplificată a componentelor software distribuite, respectiva fiind agnostică față de platformă.

[6] https://cloudevents.io/ (accesat martie 2021)

Capitolul 5. Arhitecturi Software Reactive

Conceptul de ***sistem reactiv*** a fost propus ca fiind o abordare [HAREL1985] care înglobează componente cu două aspecte distincte - comportamental și de implementare, iar spre deosebire de sistemele clasice (obișnuite), denumite *transformaționale*, care funcționează pe principiul intrare/ieșire, ”*sistemele reactive sunt sisteme solicitate în permanență de mediul exterior și au rolul de a răspunde (ca formă de a reacționa) continuu (fără întreruperi) la intrări.*”.

Această definiție inițială, însă, nu s-a rezumat doar la un sistem digital. Ulterior, Gérard Berry a extins acest concept în contextul sistemelor software [BERRY1989]. Conform acestuia, sunt diferențiate trei tipuri de aplicații:

1. *Transformaționale* – datele de intrare sunt citite și procesate, iar ulterior sunt returnate rezultatele;
2. *Interactive* – acestea interacționează cu utilizatorii sau alte aplicații în ritmul propriu;
3. *Reactive* – cele care mențin o interacțiune continuă cu mediul său, însă în ritmul dictat de mediu și nu de aplicația în sine.

Aplicațiile reactive trebuie să fie ”capabile” (în sens de măsuri tehnice adecvate) să suporte un număr variabil de sarcini ca urmare a volumului imprevizibil de date implicat. Aplicațiile în timp-real pot fi considerate, în general, ca fiind aplicații reactive.

Întrucât marea majoritate a sistemelor software moderne implică utilizarea concomitentă a acestora de către mai mulți utilizatori, avem de-a face cu activități concurente. Tehnic, aceste activități presupun solicitarea concurentă a resurselor sistemului și depind direct de disponibilitatea lor. Ca metodă de abordare arhitecturală, după cum a fost menționat anterior (vezi **Capitolul 4**), se recurge la tehnici care facilitează scalabilitatea sistemului. Concurența se poate manifesta în două moduri:

1. *Local* – se manifestă la nivelul unei singure componente;
2. *Distribuit* – implică interacțiunea generată de procese care rulează în cadrul a două sau mai multe componente independente.

5.1 Manifestul Reactiv

Pentru a satisface (oarecum) cerințele actuale în proiectarea și implementarea de sisteme software moderne, au fost propuse o serie de principii prin respectarea cărora poate fi construit un ”*sistem robust, rezilient și flexibil*” [REACTMAN], toate fiind sistematizate într-o specificație denumită *Manifestul Reactiv* (eng. *The Reactive Manifesto*).

Principiile de bază ale acestui manifest, confirmate, de altfel, fiecare separat ca fiind esențiale într-un sistem, sunt:

1. *Receptivitatea* (eng. *responsiveness*) – trebuie luate o serie de măsuri care să mențină sistemul accesibil, astfel încât funcționalitățile acestuia să fie îndeplinite într-un interval de timp asumat.
2. *Reziliența* (eng. *resiliency*) – în cazul unor probleme, sistemul trebuie să se mențină receptiv, fără ca utilizatorul să fie conștient de acele probleme.
3. *Elasticitatea* (eng. *elasticity*) – în cazul unui flux variabil de solicitări către sistem, acesta poate reacționa prin modificarea proporțională a volumului de resurse necesare îndeplinirii sarcinilor.
4. *Bazat pe Mesaje* (eng. *message-driven*) – aplicând metoda de comunicare asincronă prin mesaje, componentele își asigură o autonomie crescută una față de alta, dar în același timp facilitează aplicarea principiilor privind izolarea și transparența locației. Decuplarea componentelor și comunicarea bazată pe mesaje, cresc nivelul de scalabilitate, flexibilitate și robustețe al sistemului, facilitând un control mai granular asupra elementelor acestuia și a interoperabilității dintre componente.

Un sistem care îndeplinește criteriile menționate se numește ***Sistem Reactiv***.

Printr-o latență controlată un sistem poate fi considerat *receptiv*. Aspectele arhitecturale prin care poate fi asigurată receptivitatea unui sistem sunt *reziliența* și *elasticitatea*. Aceste două caracteristici sunt interdependente, dar ambele sunt determinate de modul de interacțiune. Astfel o interacțiune asincronă, bazată pe mesaje, reprezintă baza unui sistem reactiv.

Sistemele reactive pot fi considerate ca fiind implementarea unui stil arhitectural particular, care presupune divizarea sistemului în mai multe componente, dar care funcționează colaborativ și în mod coerent ca un tot unitar, sunt receptivi la stimulii din mediul înconjurător și elastice în fața volumului variabil al acestora, precum și reziliente în cazul unor eventuale erori.

Arhitecturile reactive se referă la procesul de proiectare a sistemelor reactive.

”Principiile Reactive” reprezintă un set particular de abordări tehnice, în primul rând arhitecturale, care facilitează construirea de sisteme coerente în medii distribuite eterogene. Arhitectura unui sistem software reactiv facilitează implementarea unui sistem complex, compus dintr-un număr mare de componente cu funcționalități distincte, dar care funcționează în mod colaborativ ca un tot întreg și care reacționează în mod coordonat la stimulii din mediul exterior. Un sistem reactiv rămâne funcțional în situațiile de salturi rapide a volumelor de date ingerate, resursele de calcul necesare asigurându-se prin tehnici specifice de scalare.

5.2 Modelul cu Actori

Modelul cu actori (eng. *actor model*) a fost formalizat sub forma unei metodologii care poate fi utilizată ca unealtă teoretică și practică pentru abordarea aspectelor complexe privind concurența în sistemele de calcul [HEWITT1973]. Un *actor* reprezintă unitatea primitivă universală de calcul cu care operează *modelul cu actori*.

Pentru sistemele software care necesită funcționalități ce implică activități concurente sau paralele, sau interacțiuni în medii distribuite, scalabile, modelul de actori oferă suportul teoretic necesar abstractizării acestor aspecte, ceea ce simplifică semnificativ nu doar implementarea acestora, dar și mentenanța și evoluția.

5.2.1 Actori versus Obiecte ca paradigme în programare

În Programarea Orientată Obiect (POO) unitățile fundamentale de calcul sunt modelate sub formă de *obiecte*. În abordarea propusă de Carl Hewitt, atunci când proiectăm un sistem sau, la nivel mai granular, unele module/componente conform principiilor Actor Model, ”*orice reprezintă un actor*”. Obiectele interacționează prin apelul explicit al metodei care expune funcționalitatea necesară, pot expune proprietăți (variabile interne) pentru a fi citite sau și modificate din exterior. Actorii, în contrast cu obiectele, interacționează exclusiv prin intermediul schimbului de mesaje, nu-și expun direct starea internă pentru a fi citită sau modificată din exterior, ci doar prin intermediul mesajelor și ca urmare a deciziilor luate intern în funcție de comportamentul curent al actorului, care, însă, se poate modifica [AGHA1985].

5.2.2 Actorul ca unitate fundamentală de calcul

Ca unitate de calcul, un actor este format din următoarele elemente:

- *Procesare* – întrucât are anumite sarcini de îndeplinit;
- *Stare* – trebuie să fie capabil să memoreze anumite detalii, care sunt utilizate în procesare;
- *Comunicare* – necesară din perspectiva interacțiunii cu alte unități de calcul.

În mod particular, un actor este format din: căsuța poștală, stare, comportament, strategia de supervizare și referințele către actorii-copii.

Starea unui actor într-un moment dat este reprezentată prin valorile variabilelor interne din acel moment, care țin de logica internă și determină **comportamentul actorului** sau, cu alte cuvinte, cum va fi procesat următorul mesaj. În cadrul unui actor nu există nici un fel de concurență, fapt care asigură că starea acestuia nu poate fi alterată decât de actorul respectiv, ceea ce garantează un **comportament consistent**.

5.2.3 Transparența locației și interacțiunea dintre actori

Fizic, componentele se pot afla local sau la distanță, interacțiunea realizându-se strict prin implicarea elementelor specifice de rețea. Logic, în funcție de anumite criterii specifice domeniului, componentele pot fi considerate ca fiind locale, chiar dacă fizic se află la distanță, sau la distanță, chiar dacă fizic se află local (alături). Transparența locației se referă la posibilitatea de a comunica cu o componentă în același mod, indiferent de locația ei.

Interacțiunea dintre actori se realizează exclusiv prin pasarea asincronă de mesaje, iar aceasta se realizează doar dacă este cunoscută adresa destinatarului. În localizarea actorilor sunt utilizate identificatoare unice, general valabile și interpretabile în cadrul sistemului sau cluster-ului. Locația fizică nu este deconspirată și nici nu este relevantă, ceea ce oferă în cadrul sistemelor distribuite o serie de avantaje menționate deja. Deci comunicarea dintre actori se realizează utilizând aceeași semantică, indiferent de locația fiecăruia.

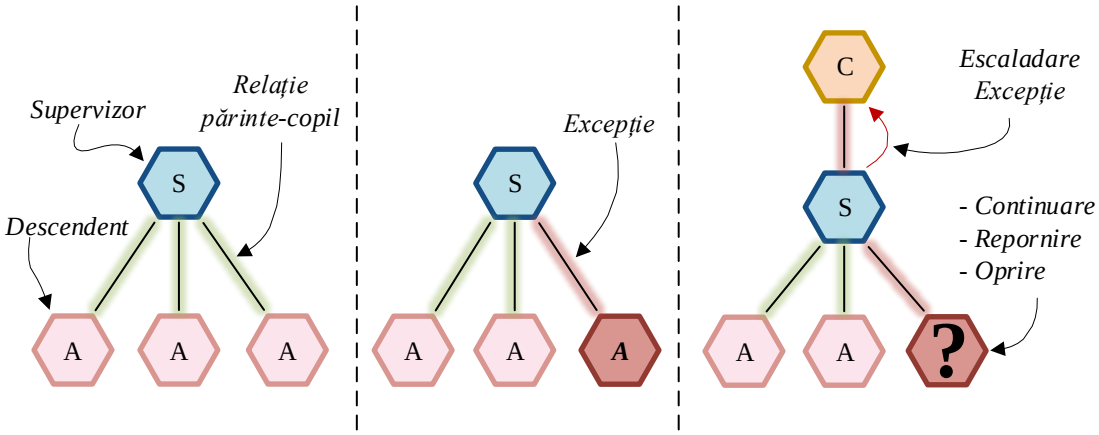

Fig. 5.1 - Ierarhiile de actori. Supravegherea și interacțiunea

5.2.4 Reziliența

Dacă sistemul se confruntă cu anumite situații care generează erori pe care nu poate să le trateze, atunci acesta devine instabil, având efecte negative asupra unei serii întregi de atribute importante, precum performanța, nivelul calității serviciilor etc. Pentru a preveni asemenea situații, se recurge la anumite măsuri care permit asigurarea rezilienței și a nivelului ridicat de toleranță la erori.

Reziliența are o abordare care presupune în primul rând receptivitatea la erori. În esență, un sistem rezilient, în eventualitatea unei probleme la o componentă, încearcă să o ”resusciteze”, iar în cazul în care acest lucru nu este posibil, instanțiază o nouă componentă care să fie în măsură să preia sarcinile acesteia.

Reziliența face parte din proprietățile sistemului care țin de arhitectura acestuia, deci măsurile necesare trebuie prevăzute explicit în etapa de elaborare a arhitecturii.

Capitolul 6. Monitorizarea Inteligentă a Mediului

Starea îngrijorătoare a mediului înconjurător, prin degradarea accelerată a acestuia și efectele care au loc în consecință, influențează în mod direct sănătatea noastră [KUM2013] (atât fizică, cât și mintală), clima, ecosistemele și interacțiunea dintre ele.

Mediul înconjurător se prezintă ca un sistem extrem de complex, cu subsisteme având interacțiuni foarte sofisticate, cu o sumedenie de indicatori și parametri variabili care sunt determinați la rândul lor de o mulțime de factori. Soluțiile de monitorizare trebuie să permită interpretarea, sub diverse forme, și înțelegerea exactă a cauzelor evenimentelor și fenomenelor monitorizate, ceea ce reprezintă elementul-cheie în luarea deciziilor pentru eventualele acțiuni de corecție, prevenire sau înlăturare a anumitor evenimente și fenomene nedorite.

6.1 Geamănul digital în sisteme inteligente

Avansul și maturizarea tehnologiilor digitale din ultimul deceniu, au creat condiții propice pentru proiectarea și implementarea unor sisteme informatice care presupun interacțiunea sub diverse aspecte cu obiecte și ființe din lumea reală. Sistemele respective sunt denumite *Inteligente* (din eng. *Smart*). „*Cea mai esențială caracteristică a mediilor inteligente este abilitatea de a controla dispozitivele la distanță, chiar și automat.*” [COOK2005]

Monitorizarea mediului este caracterizată prin utilizarea de tehnici, metode și tehnologii specifice mediului în cauză, însă toate au la bază identificarea fenomenelor în timp și spațiu. Prin integrarea în astfel de procese a tehnologiilor digitale, rezultă crearea așa-numitor sisteme cibernetic-fizice [DUMITRACHE2017]. Astfel, se pot efectua măsurători exacte aproape în timp real, la un nivel ridicat de automatizare, precum și prin aplicarea tehnicilor de inteligență artificială [CURRY2020].

Aspectele caracteristice care fac un sistem ca fiind inteligent și autonom, sporesc inevitabil complexitatea acestuia, dar, așa cum a fost arătat anterior, pentru a gestiona eficient aceste detalii în toate fazele elaborării sistemului – proiectare, implementare, operaționalizare – soluția o reprezintă modularizarea adecvată a funcționalităților, activitate care trebuie luată în considerare începând cu elaborarea arhitecturii sistemului. Una dintre abordările esențiale în acest sens o reprezintă *Geamănul Digital* (eng. *Digital Twin*), unde elementul-cheie sunt datele și tehnicile cu metodele specifice de gestionare a acestora.

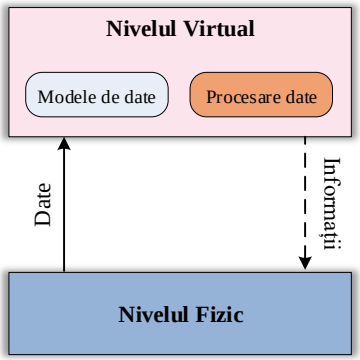

Fig. 6.1 - Modelul unui geamăn digital

Ca abordare general-valabilă, geamănul digital presupune interconectarea lumii virtuale cu cea fizică prin reprezentarea digitală a unui obiect din lumea reală [GRIEVES2014]. Din perspectiva monitorizării mediului, geamănul digital reprezintă imaginea unui dispozitiv fizic, ca element de abstractizare a principalelor caracteristici care constituie obiectul observației sau manipulării. Respectiva reprezentare este utilizată pentru perceperea, interpretarea și acționarea stării dispozitivului modelat.

Th. Borangiu et al. în lucrarea [BORANGIU2019] oferă o descriere concisă și cuprinzătoare privind conceptul de geamăn digital:

”*Imaginea holistică a capabilităților, stării și a caracteristicilor entității reale (echipament, proces, produs), inclusiv reprezentarea sa digitală, contextul execuției, istoricul comportamentului, evoluției în timp și a stării pot fi încapsulate în geamănul digital – definit*

ca un model virtual extins al unei entități fizice, proces, produs, care este persistat chiar dacă echivalentul său fizic nu este permanent online/conectat. ”

Gemenii digitali sunt utilizați pentru abstractizarea realității fizice pe patru dimensiuni care diferă ca scop sau granularitate:

1) obiecte, componente, dispozitive elementare – reprezentarea digitală individuală a unei entități dintr-un sistem;
2) dispozitive sau componente complexe - acestea pot fi privite ca o structură agregată, fiind formată din mai multe elemente de tipul (1), dar din perspectiva funcționării acestora ca un tot întreg;
3) procese – care sunt destinați reprezentării digitale a comportamentului entităților modelate și pot fi priviți ca o reprezentare a dinamicii acestora în timp și spațiu, dar fiind dependente și de context;
4) medii sau sisteme și sisteme de sisteme – cel mai complex, dar și complet mod de reprezentare digitală a unui mediu, prin care este construită imaginea holistică a acestuia.

Geamănul digital este reprezentarea digitală a unei entități/obiect sau a unui grup de entități/obiecte din lumea reală prin reflectarea exactă și actualizată în timp real, în ambele sensuri, a valorilor aferente principalelor atribute și caracteristici urmărite în procesul de monitorizare.

Gemenii digitali sunt interfața dintre lumea reală și lumea digitală, asigurând o legătură continuă, eventual în timp real, dintre cele două lumi, datorită cărora granița dintre ele devine și mai puțin sesizabilă.

În arhitectura sistemului propus, rolurile geamănului digital sunt:

- de a fi ”dublura” dispozitivului fizic în sistemul modelat prin reflectarea exactă a stării acestuia, reprezentată prin ultimele valori cunoscute ale atributelor urmărite;
- de a reprezenta un unic punct de referință în contextul accesării valorilor atributelor atât în ceea ce privește starea curentă (ultimele valori), cât și a valorilor istorice;
- de a reprezenta un singur punct de contact în cazul în care se dorește de a interacționa cu dispozitivul respectiv;
- de a asigura interacțiunea dintre dispozitivul ”de pe teren” cu partea digitală a sistemului;
- de a reacționa la evenimentele specifice emise de dispozitivul fizic;
- de a completa funcționalitățile dispozitivelor reale prin preluarea și executarea unor sarcini în cadrul funcțiilor implementate intern sau prin transferul intermediat [TAPUS2013] către alte componente specializate fie din interiorul sistemului, fie din exterior;
- acționarea, după caz, a dispozitivelor reale ca răspuns la evenimentele provenite de la ele sau retransmiterea de comenzi către acestea ca urmare a deciziilor luate sau a evenimentelor lansate de alte componente implicate în anumite fluxuri specifice de date.

Fiind o entitate digitală, geamănul digital poate fi gestionat extrem de ușor ca și orice alt tip de entitate digitală. Astfel, poate fi creat, recreat și distrus, poate fi persistat, mutat sau copiat în alt sistem aflat la distanță. Spre deosebire de entitățile reale, replicile lor digitale pot fi accesate în mod concurent de alte componente, iar suplimentar li se pot aplica politici de control acces sau alte metode specifice pentru a le gestiona în mod controlat și securizat.

Replica digitală a oricărui dispozitiv poate fi utilizată pentru preluarea și execuția unor sarcini și pentru extinderea capabilităților acestuia în ceea ce privește procesarea datelor și

integrarea cu alte componente. Această abordare reprezintă un avantaj important din punctul de vedere al dispozitivului, întrucât i se poate reduce considerabil consumul de energie [TAPUS2013], [TAPUS2019]. În același timp, sarcinile sunt executate pe servere unde puterea de procesare nu reprezintă o problemă și într-un mod controlat de logica implementată în cadrul geamănului digital. Din perspectiva suportului de servicii și resurse digitale în acest sens, tehnologiile Cloud oferă soluțiile necesare – permit operaționalizarea unor soluții scalabile, reziliente, performante, extensibile, robuste, flexibile.

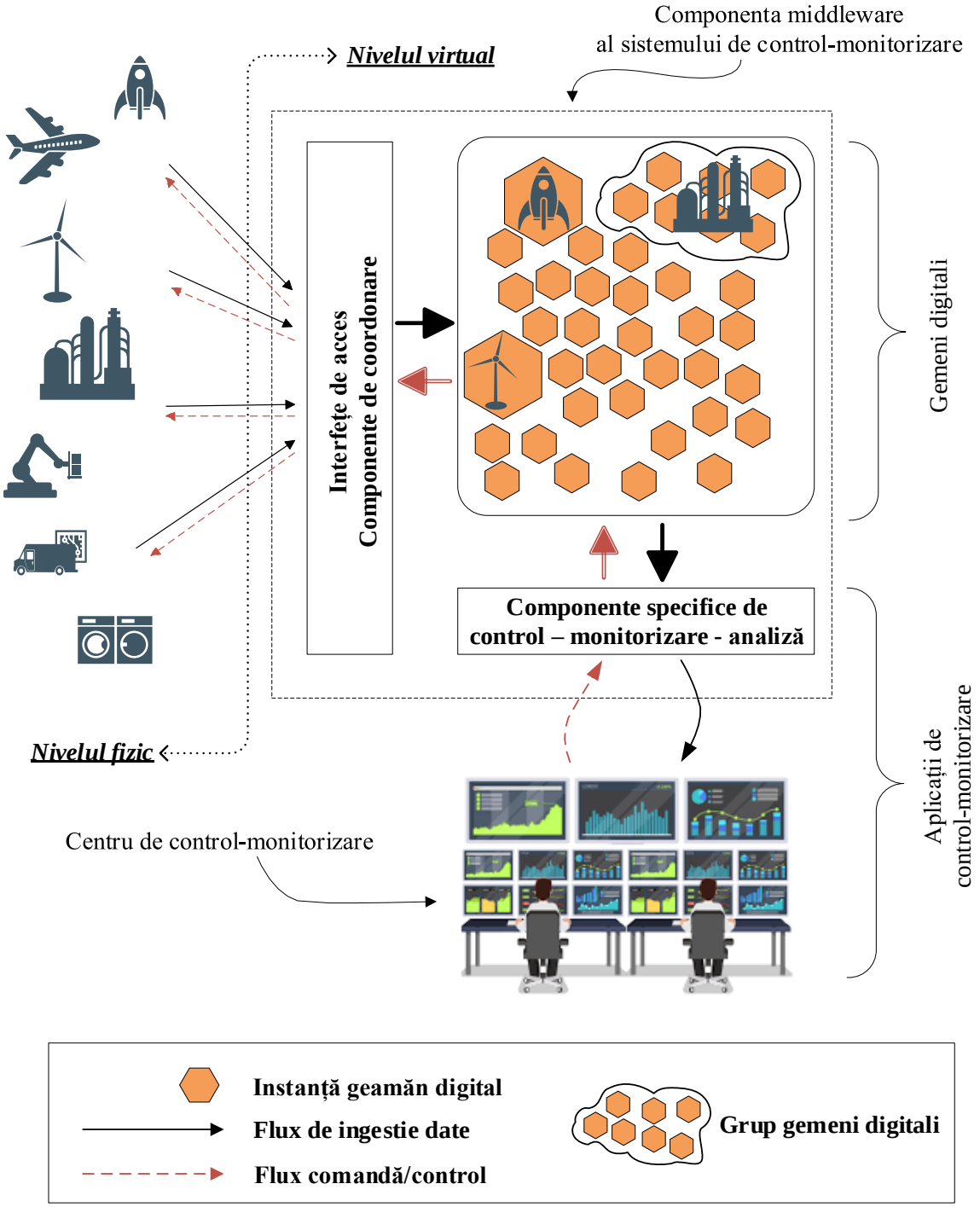

Fig. 6.2 - Arhitectura conceptuală a unui sistem de monitorizare care implică utilizarea gemenilor digitali

6.2 Sistem scalabil pentru monitorizarea inteligentă a mediului

Arhitectura unui sistem reprezintă viziunea arhitectului de sistem asupra mediului, dar în funcție de un anumit context. Pentru a putea fi implementată, însă, aceasta trebuie să cuprindă elementele esențiale din perspectiva problematicii specifice contextului dat, într-o formă coerentă și unitară.

Procesele automate de monitorizare necesită o infrastructură digitală cu accent sporit pe aspectele privind conectivitatea și interoperabilitatea eficientă dintre componentele constituente. Astfel, arhitectura propusă pentru sistemul de monitorizare conține un set de componente specializate care își pasează mesajele între ele, conform unor reguli determinate fie de factori externi (configurări, detalii specifice prezente în structura mesajului etc.), fie de scopul primar al componentei (de exemplu persistența).

În abordarea curentă, un mesaj poate avea următoarele scopuri, care au rolul și de pași finali în cadrul unei operațiuni:

- *Evidență stare* – mesajul este utilizat pentru extragerea informațiilor relevante și păstrarea acestora sub formă de ”valoare curentă” sau ”ultima valoare cunoscută”;
- *Tranzitorii* – mesajul poate fi distribuit unei componente externe sistemului pentru un oarecare scop particular – mesajul părăsește sistemul și este transmis către un altul;
- *Persistență* – mesajul este capturat cu scopul de a fi salvat (de regulă în baza de date) pentru operațiuni viitoare.

Mesajul poate reprezenta o comandă sau să semnalizeze apariția unui eveniment, ceea ce va declanșa execuția unor anumite operațiuni ce țin de funcționalitățile componentei respective. Metoda de bază de interacțiune dintre componente constă în pasarea asincronă de mesaje. Însă, în funcție de specificul funcționalității, mesajul poate trece și prin unii pași care se execută sincron.

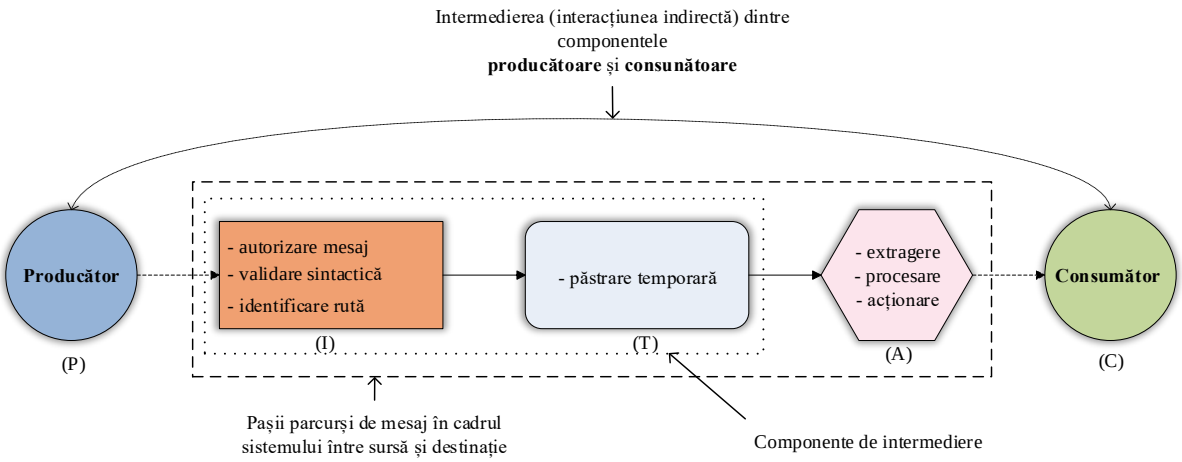

Fig. 6.3 - Modulele unui sistem de monitorizare

Din perspectiva arhitecturii propuse, avem următoarele tipuri generale de componente (în ordinea pașilor parcurși de mesaj de-a lungul fluxului):

- *producătoare de mesaje* (P) – mesajul poate fi generat fie de o componentă externă față de sistem, fie de una internă;
- *de ingestie* (I) – acestea sunt responsabile de acțiuni precum autorizarea componentei (dacă e cazul), validarea sintactică a mesajului, direcționarea către o locație temporară;
- *păstrare temporară* (T) – componente optimizate pentru păstrarea temporară și ordonată a mesajelor. Un scop important, printre altele, al acestor componente este descongestionarea/depresurizarea fluxului de mesaje;

- *componente active* (A) – reprezintă ”creierul” sistemului sau partea ”inteligentă” a acestuia. În cadrul acestor componente are loc procesarea mesajelor sau supravegherea/gestiunea procesării mesajelor în cazul în care acest aspect este delegat unor componente specializate;
- *consumătoare de mesaje* (C) – acestea pot fi reprezentate atât de componente interne, cât și de componente externe. În cazul componentelor externe, mesajele pot avea și rol de instrucțiuni/comenzi cu scop de acționare.

6.2.1 Cerințe față de un sistem de monitorizare a mediului

Mai jos sunt enumerate caracteristicile, expuse de către autorul prezentei teze și în [TOVARNITCHI2016], pe care trebuie să le aibă un sistem distribuit inteligent de monitorizare a mediului. Caracteristicile au fost detaliate și în cadrul lucrărilor [TOVARNITCHI2017], [TOVARNITCHI2019A], [TOVARNITCHI2019B], [TOVARNITCHI2021].

Caracteristicile esențiale:

- să ofere soluții flexibile și eficiente de interconectare a dispozitivelor distribuite de măsurat (senzori) sau de acționare (actuatori) la entități centrale (servere);
- să utilizeze tehnici performante și fiabile de comunicare, dar standardizate, dintre componente eterogene, adaptate la medii și necesități concrete;
- să ofere suport pentru stocarea, prelucrarea și analiza unui volum mare de date, prin utilizarea unor algoritmi eficienți și componente software dedicate;
- să poată funcționa cu un anumit grad de autonomie. Sistemul poate fi setat astfel încât în unele situații să se comporte conform unor reguli, cu succesiuni exacte de pași, în funcție de situații concrete, fără necesitatea unei intervenții din partea operatorului uman;
- să poată funcționa cel puțin în mod reactiv. În funcție de datele care ajung în sistem, dar și alte condiții, acesta să poată apela alte sisteme sau acționa dispozitive la distanță;
- să se poată integra cu alte sisteme. Trebuie să expună interfețe software (API-uri) prin care să se poată integra cu alte sisteme conform unor reguli stricte și bine definite pentru a putea beneficia de funcționalitățile altor sisteme sau de a-și expune funcționalitățile și datele proprii (de văzut paradigma SaaS sau PaaS – în funcție de context). Metodele de integrare sunt un aspect foarte important la ora actuală, în condițiile în care sistemele închise pierd teren în fața celor care permit extensibilitatea (sisteme deschise – eng. *open systems*) [TOVARNITCHI2012];
- să ofere posibilitatea de a integra direct în sistem sau de a apela la serviciile unor sisteme de analiză de date pentru a putea valorifica datele și de a oferi suport pentru deciziile luate de sistem fie automat sau nu;
- să ofere interfețe vizuale prin care un operator uman să poată interveni în sistem sau să urmărească situațiile care apar în sistem în timp real sau ulterior apariției evenimentelor în sistem;
- să facă față necesităților variabile de resurse de calcul - scalabilitatea. Numărul de componente distribuite care produc sau necesită date poate crește sau scădea rapid [TOVARNITCHI2021].

Capitolul 7. Propunere de arhitectură software scalabilă pentru un sistem de monitorizare inteligentă a mediului

În etapa de proiectare software, preocuparea de bază implică parcurgerea obligatorie a pașilor aferenți identificării componentelor software și a delimitării responsabilităților acestora. Acești pași determină în mod decisiv deducerea și formalizarea interacțiunilor dintre componentele identificate, precum și modalitatea prin care legăturile respective urmează să fie orchestrate astfel încât să se atingă o coerență maximă între componente și cu efect direct asupra atingerii scopului final pentru care este proiectat și implementat sistemul. Astfel, în funcție de responsabilitățile fiecărei componente identificate în cadrul sistemului, vor fi selectate pattern-uri și principii adecvate care vor guverna relațiile dintre respectivele componente într-un mod optim și vor constitui baza pentru un sistem flexibil, evolutiv, robust.

În **funcție de proveniența, destinația/scopul și sensul** (uneori) față de sistem – intrare sau ieșire – mesajele pot fi grupate în două tipuri: *evenimente* și *acțiuni*.

Un ***eveniment*** reprezintă un fapt împlinit, care se referă la trecut, asupra căruia nu putem să intervenim și trebuie să-l tratăm ca atare. Evenimentele pot avea legătură cu un fapt atât din exteriorul sistemului, cât și din interiorul lui. Acestea pot avea rol de stimul sau declanșator (trigger).

O ***acțiune*** reprezintă o intenție, o activitate care urmează să aibă loc în viitor. Acestea au rol imperativ și pot fi considerate comenzi – consecințe ale unor decizii interne luate ca urmare a evenimentelor recepționate.

Entitățile de bază (în sens de componente) cu care operează un sistem de monitorizare pot fi încadrate în una din următoarele două categorii:

- *producători*
- *consumători*

7.1 Interacțiunea și fluxurile de date în sisteme software distribuite

Componentele unui sistem pot comunica direct unele cu altele sau prin intermediul unor mediatoare — componente intermediare, cu roluri delimitate și dedicate aspectelor care se referă în primul rând la semantica și sintactica comunicării, dar și de particularitățile ce țin de: dinamica comunicării, restricțiile impuse de arhitectura sistemului, de particularitățile de infrastructură etc. Astfel, o abordare specifică o reprezintă implementarea în funcție de context a interacțiunii dintre componentele sistemului.

Sistemul de față utilizează două tipuri de componente cu rol de mediere:

1) Event Gateway;
2) Cozi de mesaje (eng. *Message Queues*).

7.2 Event Gateway

Expunerea unei serii de funcționalități particulare pe de o parte, iar pe de altă parte încapsularea și ascunderea detaliilor specifice de implementare, reprezintă o abordare comună și necesară în contextul arhitecturilor de sisteme distribuite.

Componenta Event Gateway implementată are următoarele caracteristici:

- funcționează pe baza protocoalelor *HTTP/HTTPS* sau a altora compatibile cu acestea (de exemplu *CoAP*) și a particularităților care permit implementarea operațiilor necesare;
- interacțiunea suportată se bazează pe specificațiile abordării REST [FIELD2000];
- interacțiunea este de tip cerere-răspuns și fără păstrarea stării (eng. *stateless*);
- este bazat pe standarde deschise (REST, JSON etc.), ceea ce-l face accesibil spre a fi utilizat și integrat ușor și fără restricții;
- are rolul de interfață a sistemului cu mediul exterior și reprezintă punctul de intrare a mesajelor în sistem.

7.2.1 Rolurile funcționale

Interacțiunea sistemului proiectat cu mediul exterior este caracterizată printr-o dinamică extrem de crescută. Concret, dinamica este determinată atât de variația numărului de entități terțe care interacționează cu sistemul, dar și de rata variabilă a volumului de informații implicate în comunicarea cu acele entități. Din această perspectivă apare necesitatea unei componente flexibile, consistente, dar în primul rând scalabile.

I. Intermediator flexibil și robust

Event Gateway reprezintă o componentă slab cuplată responsabilă de medierea interacțiunii sistemului cu mediul exterior și are rol de punct de intrare a mesajelor în sistem, punct al sistemului unde este efectuată autorizarea componentelor terțe și aprobarea mesajelor pentru a putea fi acceptate și transmise către componentele interne pentru procesările ulterioare. Componenta este agnostică față de tehnologiile pe baza cărora funcționează componentele externe care generează mesajele de intrare în sistem.

II. Router de nivel 7 (eng. *Layer 7 Router*)

Componenta API Gateway are un rol important în orchestrarea mesajelor de intrare. În funcție de obiectivele de business sau anumite aspecte funcționale sau tehnice, mesajele ingerate urmează să fie redirecționate către componente distribuite interne spre a fi procesate.

III. Reverse Proxy

Expunerea detaliilor interne ale sistemului către exterior trebuie redusă la minim. Componentele însărcinate cu procesarea mesajelor pot fi implementate în absolut orice tehnologie, neexistând o restricție în acest sens.

IV. Translator de mesaje

Pentru simplificarea și generalizarea utilizării componentei, mesajele ingerate au o structură sintactică standard. În schimb, după ingerare, acestea pot suferi anumite ajustări sau transformări mai ample, cu scopul de a fi pregătite sintactic pentru componentele interne ulterioare către care urmează să fie transmise mesajele pe parcursul fluxului.

În literatura de specialitate această abordare este conturată sub forma unui pattern aparte – *nivel anti-corupție* (eng. *Anti-Corruption Layer pattern*), formulat pentru prima dată de Eric Evans în lucrarea sa extrem de cunoscută în domeniul proiectării software ca ”*Domain-Driven Design*” sau *DDD* [EVANS2003].

7.2.2 Avantaje și puncte sensibile

Avantajele principale ale utilizării acestei componente software sunt următoarele:

- *Financiare*. Componentele hardware de tip Gateway sunt semnificativ mai scumpe.

- *Flexibilitate*. Pot fi scalate și instalate în interiorul sistemului gestionând comunicarea dintre componentele interne ale sistemului (trafic est-vest). Sunt slab cuplate (eng. *loosely coupled*) și manifestă o dependență redusă față de componentele externe, ceea ce permite să fie ușor alterate sau înlocuite complet. Aduce un plus de securitate sistemului prin autorizarea componentelor terțe și încapsularea funcționalităților expuse de serviciile interne;
- *Facilitează evoluția arhitecturii*. Fiind robuste și relativ simplu de utilizat, ele pot fi reconfigurate și adaptate la noi situații, abordări, tehnici.

Deși aduc un mare plus în arhitecturile distribuite prin decuplarea în spațiu a componentelor implicate în interacțiune, acestea totuși nu le decuplează în timp, ceea ce presupune că toate componentele implicate trebuie să fie disponibile în timpul interacțiunii.

7.3 Coada de mesaje. Roluri funcționale

Componentele implicate în comunicarea asincronă operează cu mesaje și în mod colaborativ. Pentru a permite ca mesajul să fie procesat la un moment ulterior de timp, fără ca producătorul să fie ocupat așteptând finalizarea procesării, este nevoie ca mesajul să fie păstrat o perioadă de timp de către o componentă intermediară. Abordările tehnice în acest caz sunt multiple, de la unele generale, precum fișiere sau baze de date, la unele mai specializate – cozi de mesaje.

7.3.1 Avantaje și puncte sensibile

Printre avantajele aduse de cozile de mesaje se numără:

- asigură transparența locației;
- generarea/publicarea și consumul asincron al mesajelor;
- notificarea asincronă a consumătorilor;
- în orice moment pot fi eliminate ori adăugate noi componente producătoare sau consumătoare de mesaje, ceea ce aduce o flexibilitate importantă sistemului și îi crește semnificativ abilitatea de a fi scalat [BATES1998] și extins;
- aduce un plus în abstractizarea funcționalităților sistemului;
- facilitează separarea responsabilităților.

Avantajele aduse de utilizarea cozilor de mesaje vin și cu anumite puncte sensibile:

- transparența locației duce la creșterea latenței în fluxul mesajelor;
- parțial, toleranța la erori trebuie să fie tratată de componentele implicate;
- există riscul procesării multiple sau pierderii mesajului;
- comunicarea este unidirecțională;
- ordinea mesajelor nu este garantată de la generare până la procesare.

7.4 Componentele responsabile pentru procesarea mesajelor

Odată avansul sistemelor de calcul bazate pe arhitecturi de procesoare multi-nucleu, pentru proiectarea și implementarea sistemelor software a devenit indispensabilă aplicarea abordărilor care exploatează intens principii axate pe procesare concurentă și procesare paralelă.

7.4.1 Modul reactiv și scalabil pentru procesarea evenimentelor

Modalitățile de bază pentru asigurarea scalabilității, extensibilității și reziliență este modularizarea adecvată a sistemului și cuplarea slabă prin interacțiunea asincronă bazată pe schimbul de mesaje.

Actorii și sistemele de actori în astfel de situații sunt potriviți din perspectiva următoarelor caracteristici:

- interacțiunea implicită este de tip asincron, fapt care permite lansarea operațiunilor și revenirea imediată la procesarea următoarelor mesaje;
- mesajul, în cadrul acestei implementări reprezentă o structură imutabilă de date, prin care este abstractizat un eveniment sau o intenție. Acesta necesită procesare și, eventual, o anumită reacție, dar în funcție de context, care este determinat de starea curentă a actorului, rezultată în urma unor acțiuni și decizii anterioare;
- încapsularea stării și comportament ajustabil, în funcție de context;
- datorită detaliilor interne de funcționare a unui actor, nu va exista concurență în cadrul acestuia, ceea ce ne oferă garanția că starea internă a lui va putea fi păstrată în formă consistentă și modificată în mod controlat și coerent;
- mesajele sunt procesate în ordinea în care au fost recepționate;
- oferă suport pentru implementarea unor tehnici de supervizare a execuției actorilor-copii din ierarhia subordonată, ceea ce sporește reziliența sistemului;
- prin implementarea unor politici adecvate, actorii și ierarhiile subordonate (care includ actori-copii) pot fi ”închiși” eliberând astfel complet resursele utilizate (cele de memorie în primul rând) – scalarea în jos (eng. *scale down*);
- deși tehnic există o diferență în performanța unui apel local sau la distanță prin rețea, din punct de vedere al abordării utilizat în modelul de programare suportat de modelul de actori nu există nici o diferență, întrucât toți actorii, indiferent de locația fizică sunt considerați ca făcând parte din același sistem;
- aduc un nivel sporit de flexibilitate sistemului prin aplicarea principiilor definite ca arhitecturi bazate pe microservicii – încapsularea stării și comportamentului, specializarea pe funcționalități concrete și limitate, scalabilitatea;
- sistemul poate fi extins prin adăugarea de noi ierarhii de actori, ceea ce permite extinderea funcționalităților întregului sistem – scalarea pe orizontală;
- datorită modalității de interacțiune asincronă și a transparenței locației, respectivele componente pot fi ușor scalate atât pe verticală (prin utilizarea multiplelor procesoare/nuclee), cât și pe orizontală (pot fi adăugate în sistem componente care rulează pe mașini fizice/virtuale separate și accesate prin rețea) permițând implementarea unor sisteme extrem de elastice, extensibile/evolutive;
- suportă mai multe modalități de distribuire și rutare a mesajelor adaptate la diverse situații care pot apărea în contextul componentelor distribuite.

Asignarea balansată a firelor de execuție către actorii care au în coadă mesaje în așteptare și care urmează să fie preluate pentru a fi procesate, asigură utilizarea continuă și concomitentă a tuturor nucleelor de procesor alocate sistemului de actori. **Solicitarea constantă a nucleelor și utilizarea neîntreruptă a firelor de execuție, fără a le ține în așteptare sau blocate, reprezintă ”*cel mai eficient sistem de procesare concurentă la care putem să sperăm.*”** [VERNON2015].

Nodurile respective pot fi organizate să funcționeze în mod colaborativ, iar rezultatul este formarea de clustere de noduri.

7.4.2 Procesarea efectivă a mesajelor

Abordarea aleasă pentru reprezentarea digitală a elementelor din lumea reală este legată de utilizarea gemenilor digitali.

Actorii care reprezintă imaginea digitală a dispozitivelor fizice sunt responsabili direct de coordonarea procesării propriu-zise a evenimentelor (mesajelor).

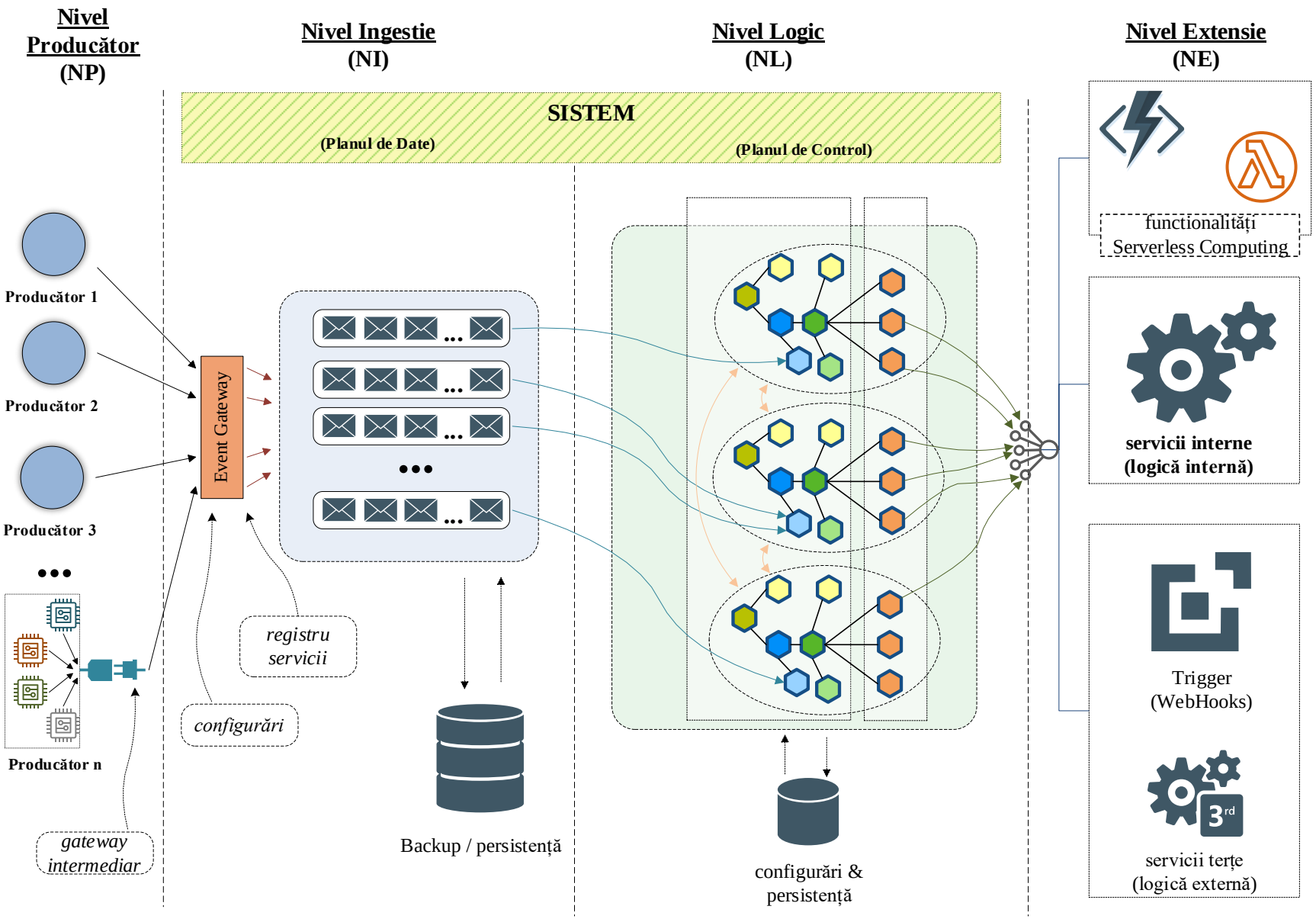

Fig. 7.1 - Arhitectura Sistemului

Capitolul 8. Detalii de implementare ale sistemului software de monitorizare a mediului. Aspecte experimentale

Alegerea tehnologiilor potrivite pentru implementarea componentelor informatice identificate în faza de elaborare a arhitecturii unui sistem software complex sunt decizii strategice care vor determina în mare măsură parametrii de calitate al acestuia.

8.1 Reprezentarea și ingestia evenimentelor. Medierea interacțiunii

Pentru descrierea evenimentelor și a contextului emiterii acestora, așa cum a fost specificat anterior, se propune spre utilizare specificația *CloudEvents*. Detaliile relevante, conform specificației, sunt reprezentate în format *JSON* [7].

8.1.1 Configurările și registrul de servicii

Comportamentul componentei este dat de configurările specifice, care determină și rutarea evenimentelor. Printre configurări se numără aspectele dinamice ale sistemului, care se pot modifica oricând, separat de aplicație, chiar și în timpul rulării acesteia. De exemplu: adrese de rețea, credențiale de acces etc.

8.1.2 Ingestia evenimentelor în sistem. Autorizarea și rutarea

Din această perspectivă, componenta respectivă impune un set de politici și reguli de acces pentru componentele producătoare de mesaje.

Prin utilizarea componentei EventGateway, este abstractizată toată complexitatea specificului componentelor interne, iar componentele din exterior pot beneficia de funcționalitățile lor într-o manieră standardizată, adaptată punctual cerințelor tehnice necesare.

Componenta EventGateway sporește agilitatea arhitecturii sistemului, întrucât aceasta poate fi reconfigurată, adaptată sau înlocuită fără să fie afectate restul componentelor din interiorul sistemului. Suplimentar, datorită implementării simple, se poate adapta ușor să corespundă unor politici și reguli de funcționare noi.

8.1.3 Scalarea funcționalităților de ingestie a evenimentelor

Temelia funcționării fiabile este determinată de scalabilitatea acestei componente. Scalabilitatea este asigurată din două perspective:

- ***conceptuală*** – constă în utilizarea de abordări care asigură premisele unei scalabilități. Rolul cel mai important aici îl ocupă izolarea prin decuplarea funcțională a EventGateway de componentele interne, dar și de cele externe.
- ***tehnică*** – implementarea abordărilor descrise în capitolele anterioare (vezi **Capitolul 4**) care cuprinde soluții software sau hardware.

8.1.4 Medierea interacțiunii

În cadrul implementării curente, specificul interacțiunii este stabilit în dependență de următoarele două criterii:

[7] https://tools.ietf.org/html/rfc8259 (accesat martie 2021)

- ***dinamica interacțiunii***, ceea ce impune ca, în situația în care este nevoie, un topic să poată fi scalat independent de celelalte, deci trebuie să corespundă unor anumite cerințe în ceea ce privește performanța;
- ***coeziunea*** dintre producători și consumători, ceea ce presupune că un topic va asigura suportul pentru comunicarea dintre acele componentele care au o relație strânsă de colaborare și un scop comun bine definit.

8.2 Procesarea evenimentelor. Distribuirea reactivă a sarcinilor

Criteriile care au stat la baza deciziei de a alege acest model au fost:

- model de programare de nivel înalt care permite execuția sarcinilor pe mai multe fire și abstractizarea complexității detaliilor specifice implementării funcționalităților de execuție concurentă pe mai multe fire;
- abstractizează interacțiunea prin rețea și execuția distribuită a sarcinilor;
- permite crearea de module independente, dar care pot fi organizate în clustere pentru execuția distribuită, colaborativă și organizată a sarcinilor și **crearea de sisteme elastice, reziliente, extensibile, evolutive care pot fi scalate atât pe verticală, cât și pe orizontală**, în funcție de necesități;
- oferă posibilitatea salvării de siguranță a stării curente pe medii persistente, fapt care sporește gradul de probabilitate ca mesajele recepționate să fie procesate, dar și posibilitatea de a ”recrea” actorul, poate chiar și în altă locație fizică, și continuarea fluxului inițial de execuție ca și cum n-ar fi avut loc acea întrerupere;
- facilitează implementarea unor componente scalabile și conforme cu principiile *Manifestului Reactiv*, ceea permite implementarea de aplicații care corespund criteriilor de arhitecturi cloud-native.

8.2.1 Structura ierarhică a nodului de procesare. Scalarea modulului

În **Fig. 8.1** este reprezentată structura conceptuală propusă a ierarhiei de actori din care este compus un nod de procesare. Actorii din ierarhia propusă pot fi grupați în patru categorii generale:

I) de Coordonare (tip AC, ACCh, ACG);
II) de Configurare (tip ACfg, ADCfg);
III) de Gestiune (AMCh, AMG);
IV) de Virtualizare Dispozitiv – reprezentați prin acronimul *AD* (Actor Dispozitiv).

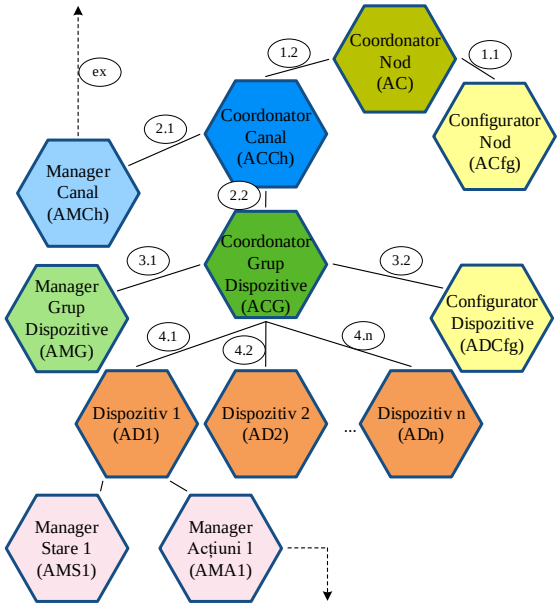

Fig. 8.1 - Arhitectura conceptuală a Nodului de Procesare

Module autonome slab cuplate, care interacționează pe bază de schimb asincron de mesaje, și transparența locației, sunt abordările esențiale de asigurare a scalabilității de care s-a ținut cont la implementarea sistemului pentru monitorizarea mediului detaliat în prezenta teză.

Transparența locației presupune o formă comună de reprezentare care abstractizează interacțiunea dintre componente, indiferent

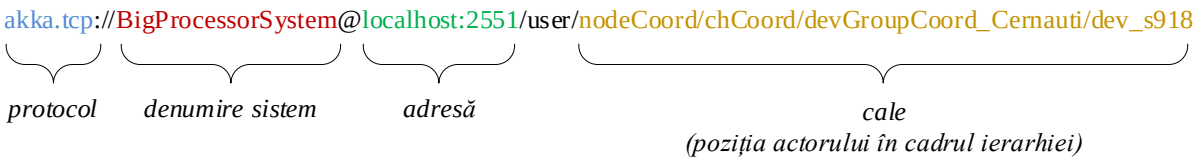

Fig. 8.2 - Exemplu de reprezentare a locației unui actor

de faptul dacă mesajul transmis urmează să fie procesat local sau la distanță. Suplimentar, asigurarea transparenței locației permite să fie adăugate componente noi, în mod dinamic.

Metoda de transmitere asincronă de mesaje decuplează funcțional componentele unui sistem software, ceea ce permite ca producătorul și consumătorul să poată fi **scalați pe verticală**, separat unul de altul.

Includerea mai multor sisteme de actori într-un cluster permite gestionarea organizată a acestora astfel încât să funcționeze ca un tot unitar, ceea ce facilitează **scalarea pe orizontală** și **disponibilitatea înaltă** a sistemului rezultat. Clusterizarea, prin adăugarea și eliminarea de noduri în timpul funcționării sistemului, permite extinderea sau contractarea acestuia în funcție de încărcare.

Clusterizarea o considerăm una dintre metodele de asigurare a unei elasticități dinamice și a evolutivității sistemului.

Nodurile care fac parte dintr-un cluster au ca scop de bază partajarea funcționalităților în scop de colaborare pentru îndeplinirea obiectivelor. Adăugarea de noi noduri permite extinderea funcționalităților sistemului.

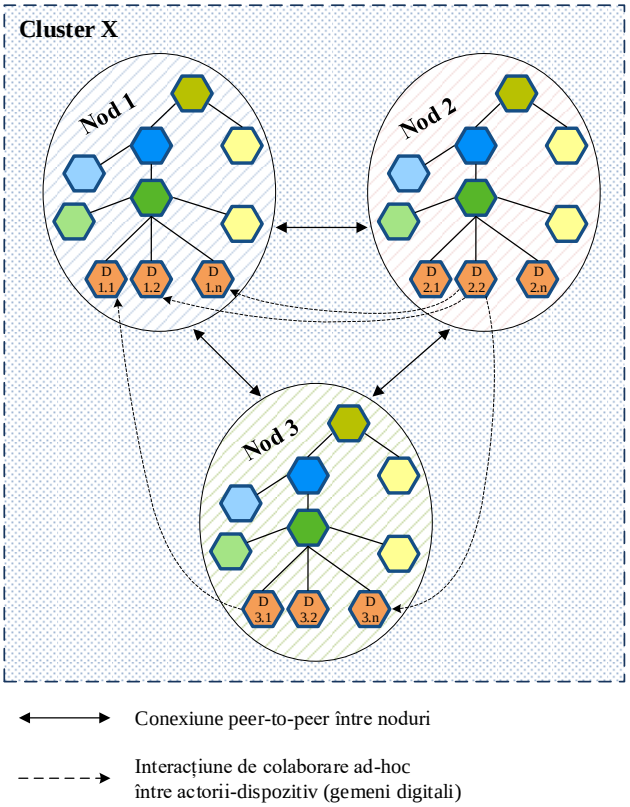

Fig. 8.3 - Cluster de noduri și colaborarea dintre actorii-dispozitiv

8.3 Extinderea funcționalităților și evolutivitatea sistemului

După cum a fost stipulat în lucrările autorului prezentei teze [TOVARNITCHI2017], [TOVARNITCHI2019A], un sistem informatic modern trebuie proiectat astfel încât să poată fi extins și evolua împreună cu cerințele beneficiarilor.

Caracteristicile menționate vin în tandem cu posibilitatea ca sistemul să fie proiectat ca unul deschis [TOVARNITCHI2012] și scalabil [TOVARNITCHI2021], ceea ce oferă avantajul ca acesta să poată fi inclus în cadrul altor sisteme completând funcționalitățile celor din urmă și formând sisteme de sisteme.

8.3.1 Detalii privind externalizarea funcționalităților și implementarea extensiilor

Sistemul descris în cadrul tezei a fost proiectat astfel încât modulele sale să poată fi înlocuite cu efort și costuri minime. Totodată, funcționalitățile deja existente pot fi completate cu unele

noi. Această caracteristică a fost implementată ca modalitate de extindere a funcționalităților dispozitivelor virtuale de la nivelul actorilor responsabili de imaginea digitală a dispozitivelor fizice. Din această perspectivă, actorii respectivi au rolul de mediator în privința extinderii funcționalităților dispozitivelor fizice, caracteristică menționată ca unul dintre avantajele de aplicare a conceptelor privind gemenii digitali [TAPUS2013], [BORANGIU2019].

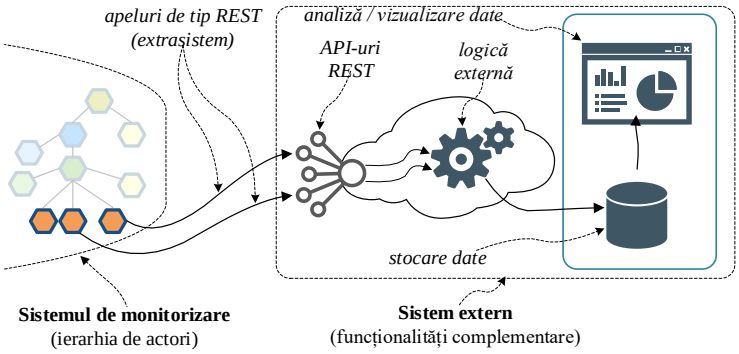

Fig. 8.4 - Externalizarea funcționalităților

Aplicația de monitorizare a mediului prezentată în această teză presupune lucru cu evenimente care sunt identificabile în *timp și spațiu*. Sunt colectate evenimentele succesive produse de surse cunoscute de-a lungul unui interval de timp și urmărită evoluția lor. Etichetarea evenimentelor cu informații privind timpul și locul producerii, vine cu un avantaj important - permite corelarea evenimentelor și determinarea contextului pentru o monitorizare în timp real pe o suprafață largă, dar și pentru o prelucrare ulterioară în scop de analiză, elaborare de prognoze sau modele.

8.4 Sumar detalii implementare

Se observă că elementele de implementare detaliate sunt încrustate adânc în arhitectura sistemului pe toate nivelurile, iar eventualele modificări ale acesteia ar veni cu un efort, deci și cost, semnificativ. Acest fapt confirmă încă o dată importanța elaborării unei arhitecturi de sistem optime, alături de o analiză temeinică și cuprinzătoare a cerințelor inițiale.

Execuția asincronă a sarcinilor determină utilizarea optimizată a resurselor de calcul implicate, permițând finalizarea mai multor sarcini într-un anumit interval de timp. Extinderea controlată a sistemului prin adăugarea de noduri suplimentare organizate în clustere, alături de execuția asincronă a sarcinilor, aduce un avantaj operațional major: permite implicarea volumului de resurse de calcul necesare la un moment dat și utilizarea optimizată a resurselor de calcul avute la dispoziție.

La implementarea sistemului de monitorizare a mediului, care s-a realizat conform cu arhitectura propusă și prezentată în paragrafele anterioare, s-a urmărit:

- implementarea unui sistem distribuit, prin utilizarea de tehnologii și tehnici moderne;
- aplicarea de tehnici detaliate și propuse în capitolele anterioare, care să permită scalarea sistemului;
- utilizarea principiilor conform cărora sistemul să funcționeze în mod reactiv;
- funcționalitățile sistemului să poată fi extinse și/sau modificate;
- experimentarea privind utilizarea actorilor pentru reprezentarea digitală a dispozitivelor fizice prin utilizarea conceptelor privind gemenii digitali;
- elaborarea unei arhitecturi Cloud hibride;
- sistemul să fie cloud-agnostic, confirmat prin utilizarea de tehnologii care nu sunt specifice unui anumit furnizor de servicii Cloud;
- utilizarea de servicii Cloud oferite de furnizori distincți, care să confirme posibilitatea de implementare a unor sisteme distribuite de dimensiuni crescute, cu posibilitatea de extindere sau înlocuire a funcționalităților.

Capitolul 9. Concluzii. Contribuții. Direcții viitoare

9.1 Concluzii

Proiectarea sistemelor informatice complexe este un domeniu fundamental și determină în mod decisiv implementarea și operarea sistemului de-a lungul întregului ciclu de viață

Un sistem software este elaborat pentru a îndeplini un anumit scop și are la bază un set de cerințe funcționale bine stabilite. Pentru abstractizarea complexității cerințelor privind funcționalitățile dorite, se recurge la formalizarea acestora sub diverse forme, rezultând o *arhitectură software*.

Identificarea și gruparea funcționalităților din același domeniu (conectate conceptual) în componente sau module comune și segregarea conceptuală dintre modulele aparținând domeniilor complementare acoperite de funcționalitățile sistemului, înlesnește gestionarea complexității ridicate a sistemelor software moderne determinate de dimensiunile și polivalența cerințelor actuale față de acestea.

Scalabilitatea și extensibilitatea unui sistem software este asigurată din etapa de elaborarea a arhitecturii lui, prin detalierea interacțiunii dintre componente și nu a detaliilor de implementare a acestora. Arhitecturile evolutive oferă posibilitatea modificării și actualizării în timp a sistemului

Factorii de bază care diferențiază numeroasele arhitecturi de sisteme informatice, se află în primul rând în particularitățile de integrare dintre componente. În cazul unui sistem informatic implementat, eficiența integrării dintre componente determină în mare parte atributele de calitate esențiale ale acestuia, întrucât funcționarea componentelor în cadrul sistemului depinde de eficacitatea ”colaborării” dintre ele. În mod particular, sistemele software distribuite sunt formate din componente specifice care pot rula și pe sisteme de calcul cu proprietăți și arhitecturi complet diferite.

Arhitectura software vine să ofere soluții tehnice concrete pentru satisfacerea cerințelor non-funcționale, precum: scalabilitatea, extensibilitatea, reziliența, disponibilitatea, mentenanța. Abstractizarea componentelor sistemului, la care apelează arhitectura software, facilitează înțelegerea simplificată a sistemelor de complexitate ridicată.

Totodată, integrarea de sisteme și instrumente software terțe, care vin cu funcționalități delimitate cu scopul de a completa pe cele necesare în cadrul sistemului proiectat, este una dintre preocupările de bază ale arhitecților de sisteme software.

Un sistem informatic reușit, are la bază o arhitectură optimă. Cu toate acestea, elaborarea arhitecturii software conține și un aspect mai puțin exact, care nu poate fi cuantificat – se bazează într-o măsură oarecare pe experiență și intuiție.

Identificarea și delimitarea componentelor unui sistem software, dispersate fie fizic, fie logic, precum și interoperabilitatea eficientă dintre ele, necesită utilizarea de diverse abordări specifice, adaptate contextului

Lipsa temeiului, din punct de vedere tehnic, conceptual, economic de a proiecta un sistem informatic complex ca o singură componentă monolitică, este motivul de bază pentru elaborarea arhitecturii sistemului ca fiind format din mai multe componente distincte, care sunt delimitate prin gruparea funcționalităților asemănătoare.

Soluțiile de infrastructură moderne, datorită avansului tehnicilor de virtualizare, oferă posibilități (oarecum) ”nemărginite” din perspectiva volumelor de resurse de calcul ce pot fi oferite. Serviciile implementate pe baza acestora, permit implementarea unor sisteme informatice nu doar de dimensiuni (oarecum) ”nelimitate”, dar și cu o serie de proprietăți esențiale de bază precum: scalabile, reziliente, flexibile, extensibile, dinamice.

Un alt aspect important de care trebuie să țină cont arhitecții de sisteme software, este abstractizarea elementelor de infrastructură. Acest detaliu permite implementarea unor soluții agnostice față de infrastructură, ceea ce asigură baza pentru implementarea unor așa-numite *arhitecturi cloud-native*. Respectivele arhitecturi sunt portabile, scalabile și extensibile.

Arhitecturile software moderne nu trebuie concepute ca fiind unele rigide, posibilitățile și funcționalitățile cărora să fie limitate strict la cerințele inițiale. Conform experienței actuale din domeniu, arhitecturile trebuie să fie *evolutive*. Acest lucru presupune elaborarea timpurie a arhitecturii software, astfel încât componentele acesteia să poată fi modificate în timp sau chiar înlocuite. Mai mult de atât, având în vedere tendințele actuale și dinamica dezvoltării tehnologiei, paralel cu diversificarea și sofisticarea cerințelor utilizatorilor finali ale sistemelor informatice, arhitecturile software moderne trebuie să fie proiectate astfel încât sistemele rezultate să fie *extensibile* în sens de adăugare de noi funcționalități sau includerea acestora în sisteme și mai mari (sisteme de sisteme).

Solicitările față de un sistem informatic modern sunt variabile și spontane

Sistemele informatice, ca una dintre cerințele de bază, trebuie să funcționeze astfel încât să ofere un anumit nivel al calității serviciilor. Acest lucru nu este posibil fără două detalii de bază: o dimensionare potrivită a resurselor de calcul și abilitatea de a le utiliza în mod optim. Acestea formează baza practică pentru *tehnicile de scalabilitate*.

Se observă o tendință în care proiectarea și operarea componentelor software devine absolut independentă de partea de infrastructură – prin utilizarea *tehnologiilor de containerizare* și a *tehnologiilor serverless*.

Abilitatea de a utiliza eficient resursele de calcul de dimensiuni variabile puse la dispoziție, este un aspect extrem de important pentru asigurarea scalabilității adecvate a sistemului.

Din punct de vedere al arhitecturii software, acest lucru se obține prin câteva abordări de bază esențiale:

- abstractizarea funcționalităților viitorului sistem;
- delimitarea funcționalităților prin gruparea lor în cadrul unor componente distincte;
- stabilirea modalității de interoperabilitate dintre componente prin modelarea interfețelor cu semantica și sintactica potrivită în funcție de context, precum și modalitatea de interacțiune.

Modularizarea și abstractizarea adecvată a detaliilor privind funcționalitățile reprezintă una dintre soluțiile esențiale de gestiune a complexității unui sistem informatic. Abstracțiile nu trebuie să depindă de detaliile abstractizate, iar detaliile nu trebuie să fie implementate în dependență de abstracții. Interacțiunea dintre componente trebuie să fie efectuată în funcție de acele abstracții. Componentele care interacționează pe baza acelor abstracții sunt considerate ca fiind slab cuplate.

Cuplarea slabă dintre componente, comunicarea asincronă și bazată pe mesaje promit să fie abordările arhitecturale de bază pentru un sistem software scalabil, performant, rezilient, extensibil și evolutiv.

Interacțiunea dintre subiecții unui sistem distribuit modern este continuă și spontană, caracterizată printr-o dinamică variabilă

Mediul de operare al unui sistem informatic distribuit este format, printre altele, de utilizatori și componente terțe care nu se află sub controlul lui direct, iar intervenția acestora, în sens de interacțiune poate fi spontană și variabilă. Sistemele informatice moderne trebuie să mențină legătura continuă cu mediul său, iar aceasta presupune ca ele să fie implementate într-o asemenea manieră, încât să fie capabile să reacționeze în mod adecvat solicitărilor. Astfel de sisteme sunt denumite *sisteme reactive*, iar una dintre abordările conform cărora se pot construi sistemele reactive este *modelul cu actori*.

Cu toate că implementarea sistemelor pe baza modelului cu actori necesită o modalitate diferită de a formaliza problema, acest model permite implementarea de aplicații reactive, iar sistemele rezultate să fie scalabile, extensibile, reziliente, elastice și receptive.

Lumea fizică și cea digitală devin tot mai mult integrate, care funcționează ca un tot unitar, iar linia de demarcare devine din ce în ce mai difuză. Convergența dintre cele două lumi – fizică (reală) și digitală (virtuală) devine inevitabilă, iar cea din urmă, care nu cunoaște, teoretic, limite, vine să extindă realitatea tangibilă

Utilizarea abordărilor moderne de abstractizare a realității tangibile precum *gemenii digitali* (eng. *digital twins*) oferă posibilitatea modelării digitale a obiectelor fizice și a proceselor din lumea reală, ceea ce permite gestionarea și monitorizarea acestora la un nivel de granularitate și exactitate extrem de ridicat și în timp real.

Tehnologiile și tehnicile de la ora actuală permit conectarea în cadrul unui mediu colaborativ a oamenilor, dispozitivelor, proceselor și serviciilor sub formă de *rețele inteligente digitale* (eng. *intelligent digital mesh*), abordări care au potențialul de a spori semnificativ gradul de competitivitate a întreprinderilor moderne și de a le ajuta să vină cu servicii și produse cu o valoare adăugată crescută.

Asemenea tehnici deschid oportunități noi de cercetare și de dezvoltare de soluții automatizate în toate domeniile vieții și activității noastre, în special unde este nevoie de colectarea, analiza și luarea de decizii rapide și corecte pe baza unor date colectate de la multiple entități fizice distincte, dispersate, dar care acționează organizat și se află într-o oarecare formă de dependență fizică sau logică.

Utilizarea îmbinată a acestor abordări, permite implementarea unor sisteme de monitorizare și gestiune a sistemelor fizice într-o manieră extrem de precisă și promptă.

9.2 Contribuții

Contribuțiile aduse în prezenta teză au un caracter preponderent aplicativ, direcția principală fiind urmărirea unei finalități practice și aplicabile a rezultatelor cercetării. Materializarea rezultatelor reprezintă propunerea unei arhitecturi software de referință pentru monitorizarea inteligentă a mediului, confirmată printr-o implementare practică a unui sistem de monitorizare a mediului detaliată în final. Arhitectura propusă a fost elaborată cu respectarea obiectivelor primare stabilite la începutul cercetării – *scalabilitatea*, *extensibilitatea*, *reziliența* componentelor software ale sistemelor informatice moderne. Domeniile de aplicabilitate a rezultatelor prezentei cercetări sunt multiple, unde, cu relativ puține ajustări, arhitectura propusă poate fi extinsă și aplicată cu succes. Domeniile-țintă includ, în primul rând, *mediile inteligente* (eng. *smart environments*).

Un efort considerabil în cadrul cercetării a fost depus pentru identificarea detaliilor, elementelor, abordărilor, tehnicilor și instrumentelor utilizate activ în ziua de azi, care permit implementarea unui sistem software de monitorizare modern astfel încât să asigure, în funcție de context, scalabilitatea maxim posibilă a acestuia. O direcție secundară de care s-a ținut cont în cadrul cercetării, dar nu mai puțin importantă în contextul operării unui sistem final, a fost identificarea soluțiilor practice pentru asigurarea indicatorilor ridicați privind următoarele atribute de calitate ale unui sistem software: performanța, extensibilitatea, reziliența.

În afară de prezentarea modalităților existente la ora actuală în ceea ce privește abordarea scalabilității din diverse perspective, a fost expusă viziunea autorului privind detaliile specifice de elaborare a arhitecturii software astfel încât viitorul sistem să fie atât scalabil, cât și extensibil și rezilient. A fost subliniată importanța abstractizării detaliilor specifice de interacțiune dintre componentele unui sistem software care se reflectă în capacitatea acestuia de a fi scalat și extins. De asemenea, au fost elaborate *arhitecturile bazate*

pe evenimente, care constituie baza implementării unor sisteme informatice moderne cu obiectivele urmărite - care să fie scalabile, extensibile și reziliente.

O contribuție aparte adusă în cadrul prezentei teze este aprofundarea și extinderea conceptului de *sistem pentru monitorizarea inteligentă a mediului*. Sumar, au fost efectuate următoarele propuneri fundamentale în respectiva direcție:

- s-a propus utilizarea abordărilor privind gemenii digitali în modelarea digitală a unui mediu fizic în scopul monitorizării inteligente;
- s-a propus utilizarea într-un mod original a actorilor din modelul cu actori pentru modelarea și implementarea gemenilor digitali ca bază pentru un sistem de monitorizare care să fie scalabil, extensibil, dar și rezilient;
- s-a propus un set de cerințe pe care trebuie să le îndeplinească un sistem de monitorizare inteligentă a mediului conform arhitecturii propuse;
- s-a propus spre utilizare o serie de componente de arhitectură software cu detalierea modulelor grupate pe funcționalități, pentru un sistem de monitorizare a mediului. Accentul a fost pus pe elementele și principiile arhitecturale identificate în cadrul cercetării care să asigure scalabilitatea, reziliența și extensibilitatea sistemului proiectat;
- s-a propus utilizarea unui format standardizat de reprezentare a evenimentelor, ca bază pentru asigurarea interoperabilității dintre componente și sisteme software eterogene.

Contribuția esențială a tezei de față este propunerea unei arhitecturi software de referință pentru monitorizarea inteligentă a mediului, care a fost validată printr-o implementare conceptuală a unui sistem software concret.

Arhitectura software propusă pentru un sistem de monitorizare a mediului este una originală din mai multe perspective. În primul rând, pentru elaborarea acesteia, s-a propus utilizarea unor tehnici și concepte moderne care au fost fundamentate și justificate în cadrul unei cercetări ample. Detaliile propuse au fost aplicate la proiectarea modulelor ale viitorului sistem care să îndeplinească cerințele identificate și să țină cont de specificul fiecărei funcționalități necesare. Felul particular în care s-a propus modularizarea sistemului asigură baza unei *arhitecturi evolutive*.

În al doilea rând, luând în considerare contextul evoluției intensive a tehnologiilor digitale unde totul ”se mută în Cloud”, originalitatea arhitecturii propuse este dată de faptul că a fost elaborată astfel încât să poată fi abordată ca fiind o *arhitectură cloud-nativă* cu aplicare în vastul domeniu de monitorizare a mediului. Acest aspect este unul esențial în asigurarea competitivității unei soluții software moderne, întrucât condiționează decuplarea maxim posibilă a tuturor componentelor software implementate de componentele de infrastructură. Adițional, aplicațiile cloud-native sunt evolutive, scalabile, reziliente și extensibile. Nu este de neglijat și eficiența unor astfel de arhitecturi atât din punct de vedere tehnic, cât și financiar.

Un efort semnificativ a fost depus pentru validarea practică a arhitecturii elaborate, materializată prin implementarea unui sistem software scalabil și extensibil pentru monitorizarea inteligentă a mediului. S-au descris aspectele specifice de implementare care reprezintă punerea în practică a detaliilor arhitecturale propuse și proiectate în faza de elaborare a arhitecturii care stă la baza respectivului sistem. Au fost expuse detaliile tehnice considerate relevante din perspectiva obiectivelor tezei de care s-a ținut cont la implementarea sistemului.

Implementarea a fost abordată din perspectiva a două aspecte distincte, dar complementare. Primul aspect este determinat de specificul implementării modulelor sistemului astfel încât acestea să fie scalabile și să funcționeze decuplat. În dependență de particularitățile identificate ale funcționalităților din cadrul fiecărui modul, s-a recurs la utilizarea de diverse tehnologii specifice, potrivite în funcție de context, decizia fiind argumentată pentru fiecare modul în parte. A fost propus spre utilizare un format standardizat

de reprezentare a evenimentelor, care să asigurare interoperabilitatea dintre componente și sisteme software eterogene atât interne, cât și externe.

Al doilea aspect important privind realizarea sistemului este determinat de detaliile de implementare a modulului de procesare. Baza teoretică o reprezintă modelul cu actori, dar implementarea este una originală din perspectiva mai multor detalii specifice. Modulul respectiv este scalabil pe orizontală și pe verticală și permite expandarea sistemului la nivel de clustere. Suplimentar, componentele acestui modul sunt extensibile și reziliente. Acesta conține implementările actorilor care reprezintă dispozitivele fizice și care gestionează interacțiunea cu mediul exterior (pe modelul gemenilor digitali), dar și transferul de sarcini către componente terțe.

Un alt avantaj important al soluției realizate îl reprezintă extensibilitatea sistemului. Extensibilitatea este gestionată la nivel de nod de procesare și poate fi aplicată în două moduri:

- adăugarea de funcționalități native, prin conectarea în cluster de noduri suplimentare cu funcționalitățile noi necesare;
- externalizarea funcționalităților, prin conectarea de componente externe (față de sistem) utilizând metode de interacțiune generice, agnostice față de tehnologie.

9.3 Direcții viitoare. Perspective

Conceptele tratate și propuse în această teză au potențial sporit de punere în practică în elaborarea de sisteme informatice complexe cu aplicabilitate în majoritatea domeniilor economice, dar și în cele referitoare la viața noastră personală.

Ca exemplu practic, în cadrul tezei, a fost propusă o arhitectură software conceptuală pentru un sistem de monitorizare a mediului. Mediul înconjurător este un ”organism” extrem de complex și care este influențat de o mulțime de actori și factori, care se află, fără pic de îndoială, într-o relație de interdependență. Respectivul domeniu este unul generos în provocări și în diverse detalii care îl determină să fie unul foarte complex, dar și actual.

Direcția principală de dezvoltare pe viitor este îmbunătățirea arhitecturii propuse și crearea unui *Sistem Holistic de Monitorizare a Mediului*, care ar ține cont explicit de diversitatea indicatorilor, factorilor și actorilor din medii diverse, eterogene, dar care, aparent sau nu, se află într-o oarecare relație, fie și indirectă. Din această perspectivă, se consideră oportună extinderea sistemului cu capabilități de inteligență artificială, dat fiind maturitatea și evoluția conceptelor și a întregii baze teoretice referitoare la acest domeniu.

Pentru punerea eficientă în practică a detaliilor privind inteligența artificială, ca o doua direcție, este necesară ajustarea capabilităților de interacțiune a entităților care modelează obiectele din lumea reală – gemenii digitali, care sunt implementați sub formă de actori conform modelului cu actori. În primul rând aceasta ar presupune proiectarea și implementarea lor, astfel încât să funcționeze cu un grad ridicat de autonomie, pe baza deciziilor luate. Deoarece acest lucru ar presupune interacțiuni dinamice dintre entitățile virtuale, este necesară implementarea interacțiunii lor astfel încât să fie capabile să inițieze și să participe în fluxuri complexe și coordonate de activități.

Interfețele om-mașină sunt indispensabile într-un sistem avansat de monitorizare a mediului. Se observă evoluția continuă a acestora, împreună cu domeniile tehnologiilor digitale conexe. Astfel, o altă perspectivă de îmbunătățire necesară a sistemului propus în această teză, impusă și de tendințele actuale, este includerea unor interfețe avansate om-mașină bazate pe realitatea augmentată. În contextul diversității și complexității tipurilor de date cu care poate opera un sistem de monitorizare la ora actuală, respectivul tip de interfețe are un potențial sporit în a oferi un instrument extrem de util în ceea ce privește simplificarea observării și controlului în timp real a situațiilor urmărite.

9.4 Publicațiile autorului

1. **Vasile M. Tovarnițchi**, "Algorithms for data processing in environmental analysis", CSCS18: The 18th International Conference on Control Systems and Computer Science, Bucharest, 2011
2. **Vasile M. Tovarnițchi**, "Open Source Hardware as a Framework", 12th International Multidisciplinary Scientific GeoConference - SGEM2012, Bulgaria, 2012, pp. 193-197, **WOS:000348533800026, ISI Indexed**
3. **Vasile M. Tovarnițchi**, "Arhitecturi software și tehnologii cloud în monitorizarea distribuită a mediului", partea VII (3 capitole) din: Nitu C., Dobrescu A. S., Oprea A., Tovarnițchi M. V. , Popescu S., Iurescu L., "Sisteme Inteligente în Ecologie. Surse regenerabile de energie. Aplicații", Matrix Rom, București, 2016, pp. 437-530, ISBN: 978-606-25-0241-6
4. **Vasile M. Tovarnițchi**, "Cloud-based Architectures for Environment Monitoring", CSCS21: The 21th International Conference on Control Systems and Computer Science, Bucharest, 2017, **WOS:000449004400102, ISI Indexed**
5. Vladimir F. Krapivin, Ferdenant A. Mkrtchyan, Vladimir Yu. Soldatov, **Vasile M. Tovarnițchi**, "An Expert Systems for the Aquatic Systems Investigation", CSCS21: The 21th International Conference on Control Systems and Computer Science, Bucharest, 2017, **WOS:000449004400101, ISI Indexed**
6. **Vasile M. Tovarnițchi**, "Designing Distributed Scalable and Extensible System using Reactive Architectures", CSCS22: The 22th International Conference on Control Systems and Computer Science, Bucharest, 2019, **WOS:000491270300081, ISI Indexed**
7. **Vasile M. Tovarnițchi**, "Intelligent Wireless Sensor Network Applications in Cloud Era", CSCS22: The 22th International Conference on Control Systems and Computer Science, Bucharest, 2019, **WOS:000491270300082, ISI Indexed**
8. **Vasile M. Tovarnițchi**, "Scalable Approaches for Environmental Monitoring Solutions", CSCS23: The 23th International Conference on Control Systems and Computer Science, Bucharest, 2021, (accepted, to be published)**, ISI Indexed**

Bibliografie

[ABBOTT2009] Martin L. Abbott, Michael T. Fisher, The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise 1st Edition, Addison-Wesley Professional (2009)

[AGHA1985] Agha, G.A., ACTORS: A Model of Concurrent Computation in Distributed Systems, Doctoral thesis, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab (1985)

[ANGEL2012] Angelov S., Grefen P., Greefhorst D.; A framework for analysis and design of software reference architectures. Information & Software Technology. 54. 417-431 (2012); 10.1016/j.infsof.2011.11.009

[BASS2003] Bass L., Clements P., Kazman R., Software Architecture In Practice, Addison-Wesley Professional, 2rd Edition (2003)

[BASS2013] Bass L., Clements P., Kazman R., Software Architecture In Practice, Addison-Wesley Professional, 3rd Edition (2013)

[BATES1998] Bates John, Bacon Jean, Moody Ken, Spiteri Mark, Using Events for the Scalable Federation of Heterogeneous Components. Proceedings of 8th ACM SIGOPS European Workshop Sintra, Portugal, (September 1998) doi: 10.1145/319195.319205

[BERRY1989] Berry Gérard., "Real Time Programming: Special Purpose or General Purpose Languages." IFIP Congress (1989)

[BORAH2015] Borah Amlan, Mucharary Deboraj, Singh Sandeep, Borah Janmoni. "Power Saving Strategies in Green Cloud Computing Systems.", International Journal of Grid Distribution Computing. 8. 299-306, (2015) doi: 10.14257/ijgdc.2015.8.1.28.

[BORANGIU2019] Borangiu T., Oltean E., Răileanu S., Anton F., Anton S., Iacob I. (2020) Embedded Digital Twin for ARTI-Type Control of Semi-continuous Production Processes. In: Borangiu T., Trentesaux D., Leitão P., Giret Boggino A., Botti V. (eds) Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future. SOHOMA 2019. Studies in Computational Intelligence, vol 853. Springer, Cham

[COOK2005] Cook Diane, Kumar Das Sajal, "Smart Environments: Technology, Protocols and Applications" (1st Edition), Wiley-Interscience, 2005

[COUL2001] - Coulouris G., Dollimore J., Kindberg T., Distributed Systems: Concepts and Design (3rd edition), Addison-Wesley (2001)

[CRISTEA2013] Cristea V., Dobre C., Pop F., "Context-Aware Environments for the Internet of Things", In: Bessis N., Xhafa F., Varvarigou D., Hill R., Li M. (eds) Internet of Things and Inter-cooperative Computational Technologies for Collective Intelligence. Studies in Computational Intelligence, vol 460. Springer, Berlin, Heidelberg (2013) doi: 10.1007/978-3-642-34952-2_2

[CURRY2020] Curry E., Real-time Linked Dataspaces: A Data Platform for Intelligent Systems Within Internet of Things-Based Smart Environments. In: Real-time Linked Dataspaces. Springer, Cham, (2020), doi: 10.1007/978-3-030-29665-0_1

[DEBS2006] Introduction. In: Distributed Event-Based Systems. Springer, Berlin, Heidelberg (2006), doi: 10.1007/3-540-32653-7_1

[DIJK1968] Dijkstra E. W., The structure of the "THE" multiprogramming system. Communications of the ACM 11, 5 (May 1968), pp. 341–346, doi: 10.1145/800001.811672

[DUMITRACHE2017] Dumitrache I., Caramihai S. I., Sacala I. S. and Moisescu M. A., "A Cyber Physical Systems Approach for Agricultural Enterprise and Sustainable Agriculture," 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, 2017, pp. 477-484, doi: 10.1109/CSCS.2017.74.

[EVANS2003] Evans Eric, "Domain-Driven Design: Tackling Complexity in the Heart of Software", 1st Edition, Addison-Wesley Professional, (2003)

[FIELD2000] Roy T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Doctoral Dissertation, 2000

[GARLAN1993] Garlan D., Shaw M., ”An Introduction to Software Architecture”, Carnegie Mellon University, Pittsburgh, PA, USA (1994)

[GRIEVES2014] Grieves Michael, "Digital twin: manufacturing excellence through virtual factory replication." White paper 1 (2014), pp. 1-7

[HAREL1985] Harel D., Pnueli A. (1985) On the Development of Reactive Systems. In: Apt K.R. (eds) Logics and Models of Concurrent Systems. NATO ASI Series (Series F: Computer and Systems Sciences), vol 13. Springer, Berlin, Heidelberg, pp 477-498

[HEWITT1973] Hewitt Carl, Bishop Peter, Steiger Richard, A universal modular ACTOR formalism for artificial intelligence. In Proceedings of the 3rd international joint conference on Artificial intelligence (IJCAI’73). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1973) pp. 235–245.

[KAISLER2005] Kaisler S. H., "Software Paradigms", 1st Edition, Wiley-Interscience, (2005)

[KALE2019] Kale V., Parallel Computing Architectures and APIs: IoT Big Data Stream Processing (1st Edition), Chapman and Hall/CRC (2019)

[KUM2013] A. Kumar, H. Kim and G. P. Hancke, "Environmental Monitoring Systems: A Review," in IEEE Sensors Journal, vol. 13, no. 4, pp. 1329-1339, (April 2013) doi: 10.1109/JSEN.2012.2233469

[LAIGNER2020] Laigner Rodrigo, Kalinowski Marcos, Diniz Pedro, Barros Leonardo, Cassino Carlos, Lemos Melissa, Arruda Darlan, Lifschitz Sergio, Zhou Yongluan, "From a Monolithic Big Data System to a Microservices Event-Driven Architecture" (2020)

[PARNAS1972] Parnas D. L., On the criteria to be used in decomposing systems into modules. Commun. ACM 15, 12 (December 1972), pp. 1053–1058. doi: 10.1145/361598.361623

[PERRY1992] Perry D. E., Wolf A. L., Foundations for the study of software architecture, ACM SIGSOFT Software Engineering Notes Vol. 17, No. 4 (October 1992), 40–52. doi: 10.1145/141874.141884

[REACTMAN] https://www.reactivemanifesto.org/ (accesat pe 13.02.2020)

[RICHARDS2015] Richards Mark, "Software Architecture Patterns", O'Reilly Media (2015)

[SCHOLL2019] Scholl Boris, Swanson Trent, Jausovec Peter, "Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications", 1st Edition, O'Reilly Media (2019)

[SHAHRAD2019] Shahrad Mohammad, Balkind Jonathan, Wentzlaff David. "Architectural Implications of Function-as-a-Service Computing.", In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52). Association for Computing Machinery, New York, NY, USA, 1063–1075, (2019) doi: 10.1145/3352460.3358296

[TAIBI2020] Taibi Davide, Nabil El Ioini, Claus Pahl, and Jan Raphael Schmid Niederkofler, "Serverless Cloud Computing (Function-as-a-Service) Patterns: A Multivocal Literature Review." In Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20) (2020)

[TAN2007] - Tanenbaum S. Andrew , Van Steen M., "Distributed Systems - Principles and Paradigms", 2nd Edition, Pearson Prentice Hall (2007)

[TAPUS2013] Olteanu A., Tapus N., Iosup A., "Extending the Capabilities of Mobile Devices for Online Social Applications through Cloud Offloading," 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Delft, 2013, pp. 160-163, doi: 10.1109/CCGrid.2013.52

[TAPUS2019] Mohammed M.A., Kamil A.A., Hasan R.A., Tapus N., "An Effective Context Sensitive Offloading System for Mobile Cloud Environments using Support Value-based Classification", Scalable Computing: Practice and Experience, 20, (2019) pp. 687-698

[TOGAF] TOGAF® Standard, Version 9.2, a standard of The Open Group, https://pubs.opengroup.org/architecture/togaf9-doc/arch/ (accesat mai 2020)

[TOVARNITCHI2011] Tovarnițchi Vasile, "Algorithms for data processing in environmental analysis", CSCS18: The 18th International Conference on Control Systems and Computer Science, Bucharest, 2011

[TOVARNITCHI2012] Tovarnițchi Vasile, "Open Source Hardware as a Framework", 12th International Multidisciplinary Scientific GeoConference - SGEM2012, Bulgaria, 2012, pp. 193-197

[TOVARNITCHI2016] Tovarnițchi Vasile, "Arhitecturi software și tehnologii cloud în monitorizarea distribuită a mediului", partea VII (3 capitole) din: Nitu C., Dobrescu A. S., Oprea A., Tovarnițchi M. V. , Popescu S., Iurescu L., "Sisteme Inteligente în Ecologie. Surse regenerabile de energie. Aplicații", Matrix Rom, București, 2016, pp. 437-530, ISBN: 978-606-25-0241-6

[TOVARNITCHI2017] Tovarnițchi Vasile, "Cloud-based Architectures for Environment Monitoring", CSCS21: The 21th International Conference on Control Systems and Computer Science, Bucharest, 2017

[TOVARNITCHI2017A] Vladimir F. Krapivin, Ferdenant A. Mkrtchyan, Vladimir Yu. Soldatov, Vasile M. Tovarnițchi, "An Expert Systems for the Aquatic Systems Investigation", CSCS21: The 21th International Conference on Control Systems and Computer Science, Bucharest, 2017

[TOVARNITCHI2019A] Tovarnițchi Vasile, "Designing Distributed Scalable and Extensible System using Reactive Architectures", CSCS22: The 22th International Conference on Control Systems and Computer Science, Bucharest, 2019

[TOVARNITCHI2019B] Tovarnițchi Vasile, "Intelligent Wireless Sensor Network Applications in Cloud Era", CSCS22: The 22th International Conference on Control Systems and Computer Science, Bucharest, 2019

[TOVARNITCHI2021] Tovarnițchi M. Vasile, "Scalable Approaches for Environmental Monitoring Solutions", CSCS23: The 23th International Conference on Control Systems and Computer Science, Bucharest, 2021, (accepted, to be published), ISI Indexed

[VERNON2015] Vernon Vaughn, "Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka" (1st Edition), Addison-Wesley Professional (2015)

[WEBB2008] Webb Molly, "SMART 2020: enabling the low carbon economy in the information age, a report by The Climate Group on behalf of the Global eSustainability Initiative (GeSI)." Creative Commons (2008)