Universitatea POLITEHNICA din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Universitatea Babeș-Bolyai
Facultatea de Matematică și Informatică

Romagna: o abordare modernă asupra uneltelor de analiză conceptuală a datelor

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Calculatoare și tehnologia informației*
programul de studii *Ingineria informației*

Conducător științific
Christian Săcarea

Absolvent
Mihai Chereji

Anul 2014

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Romagna: o abordare modernă asupra uneltelor de analiză conceptuală a datelor*, prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul Inginerie Electronică şi Telecomunicaţii/ Calculatoare şi Tehnologia Informaţiei, programul de studii *Ingineria informației* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate. Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare. Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti, Iulie 2014.
Absolvent: Mihai Chereji

.....

Cuprins

Lista figurilor 5

Lista tabelelor 6
Lista acronimelor 7

1. Analiza conceptuală formală 10
1.1. Introducere 10
1.2. Concepte matematice de bază 10
1.2.1. Mulțimi ordonate, latice, latice complete 10
1.2.2. Context, concept, ierarhie de concepte 12
1.2.3. Contexte cu valori multiple 16
1.3. Algoritmi relevanți 16
1.3.1. Algoritmi pentru construirea laticei de concepte 17
1.3.2. Algoritmi pentru vizualizarea laticei de concepte 18
1.4. Utilizări practice 18

2. Starea actuală 19
2.1. Navigatoare de concepte 19
2.1.1. Toscana 19
2.1.2. Toscanaj 19
2.1.2.1. Funcționalități 19
2.1.3. GaloisExplorer 21
2.1.4. Galitia 22
2.2. Software conex 22

3. Romagna 24
3.1. Raționament 24
3.1.1. Ușurință de utilizare 24
3.1.2. Folosirea web-ului, păstrarea confidențialității datelor 25
3.2. Tehnologii 25
3.2.1. CoffeeScript 26
3.2.2. Ember.js 26
3.2.3. d3.js 27
3.2.3.1. svg 27
3.2.4. sql.js 27
3.2.4.1. emscripten 27

3.3. Structură 28
3.4. Dezvoltare 28
3.4.1. Proces 28
3.4.2. Probleme întâmpinate 28
3.4.2.1. MySQL versus sql.js 28
3.4.2.2. SVG și probleme în afișarea corectă a textului 28
3.5. Viitor 28
Bibliografie 30

Lista figurilor

1. Exemplu de latice de concepte 8
1.1. Diagramă Hasse a unei mulțimi de numere naturale, ordonate după divizibiltate. 11
1.2. Laticea de concepte corespunzătoare contextului descris în tabelul 1.1. 15
1.3. Algoritmul Vecinii următori, Sursa [CR04] 18
2.1. ToscanaJ, afișând o latice simplă generată din date de acces ale unui site web . . 20
2.2. ToscanaJ, afișând aceeași latice ca în 2.1, de data aceasta imbricată cu altă diagramă 21
2.3. GaloisExplorer, afișând o diagramă în 3d. Sursa: site-ul proiectului [tea14] . . . 22

Lista tabelelor

1.1. Un context al animalelor vertebrate. Sursa: CDA [CR04] 13
1.2. Context cu valori multiple descriind becuri 17
1.3. Același context ca în tabelul 1.2, dar transformat în context cu valori simple . . 17

Lista acronimelor

FCA = Formal Concept Analysis (Analiza Conceptuală Formală)
DOM = Document Object Model
SVG = Scalable Vector Graphics (Grafice Scalabile Vectoriale)
HTML = HyperText Markup Language (limbaj de marcaj hiper-textual)
XML = eXtendable Markup Language (Limbaj extensibil de marcaj)
SQL = Structured Query Language (Limbaj de Interogare Structurată)
LLVM = Low Level Virtual Machine
MVC = Model View Controller

Introducere

Big-data a devenit un domeniu foarte căutat în zilele noastre, devenit un cuvânt repetat de toată lumea, de la programatori la oameni de marketing. Acest lucru se-ntâmplă deoarece lumea se *înnecă* în date, adunate din diferite surse. Site-urile înregistrează fiecare mișcare a utilizatorilor, există pedometre care încarcă pe Internet numărul de pași făcut de utilizator în fiecare oră, datele de analiză medicală întorc tot mai multe date. Interpretarea lor devine din ce în ce mai grea, și tendința este de a se folosi metode de *machine learning*, ceea ce presupune învățarea automată (câteodată semi-automată) a calculatoarelor prin prelucrarea repetată a unei cantități imense de date.

Analiza conceptuală formală oferă o alternativă, în care adunarea datelor este automată dar prelucrarea și interpretarea datelor este realizată de om, bazându-se pe câteva principii matematice bine definite și (relativ) ușor de înțeles.

Acest câmp a început în anii 80 la Universitatea din Darmstadt, ca o evoluție a teoriei clasice a laticelor. Între timp, s-a dezvoltat într-un câmp care adună tot mai mulți cercetători și își dovedește utilitatea practică dincolo de câmpul din care s-a tras și dincolo de matematica abstractă.

Ce e mai spectaculos e că totul se bazează pe câteva idei simple, elegante, generale, frumoase am putea spune, încât duc aproape de *filozofia platonică* (un concept este alăturarea unei mulțimi de obiecte descrise de unele proprietăți și mulțimea acelor proprietăți).

Unele din cele mai utilizate rezultate ale acestei metode de analiză sunt laticele de concept, diagrame care reprezintă ierarhiile conceptelor descrise mai sus.

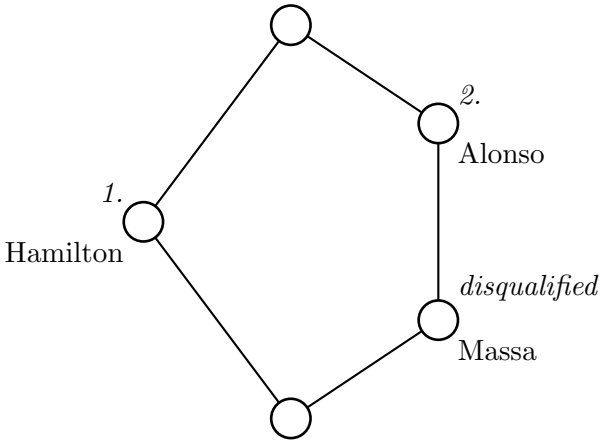

Figura 1: Exemplu de latice de concepte

Se observă că dincolo de utilitatea practică (descrisă în capitolele ce urmează), laticea este *frumoasă*. Fără a încerca să-i descifreze înțelesul, oricine poate observa că este unul din acele

produse ale științei care, ne-intenționat, poate fascina și studenții artelor.

Cu toate acestea, domeniul FCA duce lipsă de unelte frumoase (vom explica, în capitolele ce urmează ce înțelegem prin *software frumos*), chiar dacă există tot mai multe programe care doresc să ajute cercetătorii în acest domeniu. Unele frustrează încă de la încercarea procesului de instalare, altele mistifică cercetătorul prin mesaje de eroare criptice, iar altele, poate, doar se mișcă prea încet, sau sunt prea greu de folosit.

În cele ce urmează dorim să descriem pe scurt domeniul analizei conceptuale formale (în capitolul 1), starea soft-ului existent acum (în capitolul 2), și o nouă aplicație (capitolul 3), care dorește să rezolve unele din problemele întâlnite de oamenii interesați în acest domeniu, și poate, să atragă prin software bine gândit (și, sperăm, *bine realizat*, căci un concept fără nici un obiect nu are cuprins - *veți înțelege după ce veți parcurge primul capitol*) adepți noi.

Capitolul 1
Analiza conceptuală formală

1.1 Introducere

Analiza conceptuală formală este o metodă de sistematizare a datelor în **concepte**, definite la modul larg ca tupluri constituite din mulțimi de obiecte care împărtășesc anumite atribute și mulțimea acelor atribute. Este o reinterpretare a teoriei clasică a laticelor, dezvoltată în principal în anii '30, axată către partea practică. Conceptul a fost introdus în lucrarea seminală a lui Robert Wille din 1982 [Wil82], iar termenul a fost introdus în 1984 de același autor. În ultimele decenii, domeniul a atras multe contribuții și și-a dovedit utilitatea în domenii cum ar fi analiza și vizualizarea datelor, managementul informației.

1.2 Concepte matematice de bază

Pentru a avea o mai bună înțelegere a conceptelor pe care se bazează aplicația, vom explica pe scurt definițiile de bază ale domeniului.

1.2.1 Mulțimi ordonate, latice, latice complete

Vom începe cu câteva concepte elementare de algebră a mulțimilor, deoarece acest domeniu se bazează în întregime pe ele. Următoarele definiții sunt preluate din [GW97], înafara cazului unde este menționat altfel.

Definiție 1. *O mulțime M este ordonată dacă se poate aplica asupra sa o relație binară R care are următoarele proprietăți:*

Reflexivitate $xRx, \forall x \in M$

Antisimetrie $xRy, x \neq y \Rightarrow yRx$ *e fals,* $\forall x, y \in M$

Tranzitivitate $xRy, yRz \Rightarrow xRz, \forall x, y, z \in M$

Relația R se numește o ***relație de ordine****, sau* ***relație de ordine parțială****.*

Cel mai simplu exemplu, intuitiv exemplu este mulțimea numerelor reale $\mathbb{R}$, alături de relația $\leq$. Notăm o mulțime ordonată cu relația $\leq$ cu $(M, \leq)$. Un alt exemplu, relevant domeniului, este mulțimea tuturor submulțimilor sau *mulțimea părților* a unei mulțimi (notată cu $\mathcal{P}(M)$ pentru mulțimea M), și relația de incluziune ($\subseteq$).

Definiție 2. *Un element x al mulțimii M este* ***acoperit*** *de y dacă $x < y$ și nu există nici un z astfel încât $x < z < y$. În mod invers, x* ***acoperit*** *al lui y.*

Putem nota relația de acoperire astfel: $x \prec y, y \succ x$.

Definiție 3. *Două elemente ale unei mulțimi ordonate sunt* ***comparabile*** *(în raport cu $\leq$) dacă $x \leq y$ sau $y \leq x$ (adică relația $\leq$ se aplică asupra lor). Altfel sunt* ***incomparabile****. Un* ***lanț*** *este o submulțime în care oricare două elemente sunt comparabile. Un* ***antilanț*** *este o submulțime în care oricare două elemente sunt incomparabile.*

Definiție 4. ***Diagramele Hasse*** *sunt o reprezentare grafică a unei mulțimi ordonate. Elementele mulțimii sunt desenate ca niște cercuri. Dacă $x, y \in M$ (mulțimea descrisă de diagramă) și $x \prec y$, se desenează o linie între cele două elemente, iar elementul acoperit este desenat sub cercul reprezentând elementul care îl acoperă.*

Astfel, putem citi o diagramă Hasse în felul următor: $x < y$ dacă și numai dacă putem urmări o linie continuă de la cercul care reprezintă x la cercul care reprezintă y.

Exemplu 1. *Vom desena o diagramă Hasse reprezentând mulțimea de numere naturale $\{1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60\}$, ordonate după divizibilitate.*

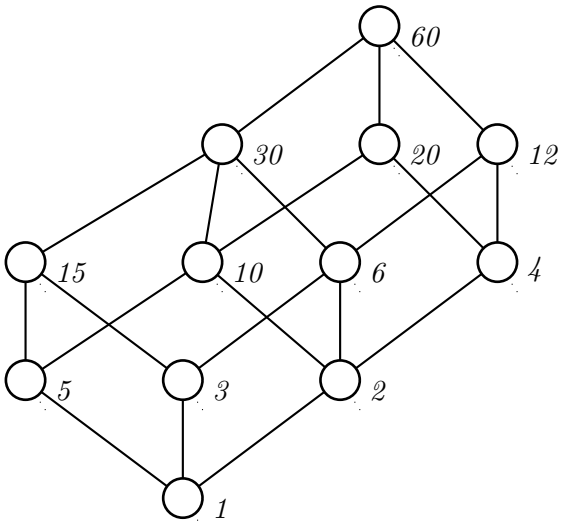

Figura 1.1: Diagramă Hasse a unei mulțimi de numere naturale, ordonate după divizibiltate.

Definiție 5. *Fie $(M, \leq)$ o mulțime ordonată, și N o submulțime a sa. Înțelegem prin* ***minorantul*** *mulțimii N un element i astfel încât $\forall a \in N, i \leq a$. În mod invers,* ***majorantul*** *mulțimii s este definit prin $\forall a \in N, s \geq a$. Putem nota mulțimea tuturor minoranților a grupului N cu I. Elementul cel mai mare din această mulțime este numit* ***infimumul*** *mulțimii N. Invers, cel mai mic element din mulțimea majoranților este numit* ***supremumul*** *mulțimii N.*

Infimumul se poate nota cu $\wedge N$ sau $\inf N$, iar supremumul cu $\vee N$ sau $\sup N$.

Definiție 6. *O mulțime ordonată M este numită o* ***latice*** *dacă $\forall x, y \in M, \exists x \vee y, \exists x \wedge y$. În alte cuvinte, o mulțime ordonată este o latice dacă pentru orice 2 elemente ale mulțimii există supremum și infimum. O latice este* ***completă*** *dacă pentru orice submulțime (finită) a ei există supremum și infimum.*

Orice latice completă are *cel mai mare element*, numit **1**, și cel mai mic element, numit **0**.

Definiție 7. *Citând din [CR04], fie o mulțime H și $\mathcal{K}$ o mulțime de submulțimi de-ale lui G. I este un sistem de închideri peste G dacă și numai dacă:*

$$\bigcap_{i\in I} A_i \in \mathcal{K} pentru\ fiecare\ submulțime\ non\text{-}vidă A_{i\in I} \subseteq \mathcal{K} și \mathcal{K} \in H$$

Astfel, $\mathcal{K}$ trebuie închisă la intersecție și să aibă un cel mai mare element. Dacă $\mathcal{K}$ e un sistem de închideri, atunci tuplul $(K, \subseteq)$ e o latice completă în care

$$\bigwedge\{A_i | i \in I\} = \cap_{i\in I} A_i,$$

$$\bigvee\{A_i | i \in I\} = \bigcap\{B \in \mathcal{K} | \cup_{i\in I} A_i \in B\}$$

Definiție 8. *Conform [CR04], un **operator de închideri** peste H e un morfism $\varphi : \mathcal{P}(H) \leftarrow \mathcal{P}(H)$ astfel încât, pentru toate $A, B \subseteq H$:*

1. $A \subseteq \varphi(A)$
2. $A \subseteq B \Rightarrow \varphi(A) \subseteq \varphi(B)$
3. $\varphi(\varphi(A)) = \varphi(A)$

[CR04] Mulțimea tuturor închiderilor al unui operator de închideri e un sistem de închideri și formează o latice completă, dacă relația de ordonare este relația de incluziune, în care:

$$\bigwedge\{A_i | i \in I\} = \cap_{i\in I} A_i,$$

$$\bigvee\{A_i | i \in I\} = \varphi\left(\bigcup_{i\ inI} A_i\right)$$

Definiție 9. *Conform [CR04] O **conexiune Galois** este compusă dintr-un tuplu de două mulțimi ordonate, M, N și două morfisme γ, ψ astfel ca $\gamma : M \to N, \psi : N \to M$, dacă și numai dacă:*

1. $m_1 \leq m_2 \Rightarrow \gamma m_1 \geq \gamma m_2$
2. $n_1 \leq n_2 \Rightarrow \psi n_1 \geq \psi m_2$
3. $m \leq \psi\gamma m, n \leq \gamma\psi n$

sau, echivalent, $m \leq \psi n \Leftrightarrow n \leq \gamma m$

În cazul unei conexiuni Galois între $\mathcal{P}(M)$ și $\mathcal{P}(N)$, morfismul $\psi\gamma$ e un operator de închidere peste M și morfismul $\gamma\psi$ e un operator de închidere peste N.

1.2.2 Context, concept, ierarhie de concepte

Definiție 10. *Fie un triplet $K = (G, M, I)$ format din 2 mulțimi, G și M, și o relație binară I între acestea. Acesta se numește **context**. Mulțimea G este compusă din obiecte, iar M din atribute.*

Literele provin din limba germană, în care aceste concepte au fost descrise inițial, fiind inițialele cuvintelor Gegenstände și Merkmale, respectiv. Relația I e numită **relația de incidență**, iar gIm poate fi citit ca "obiectul g are atributul m".

Exemplu 2. *Preluăm următorul exemplu din [CR04], un context (foarte redus)al animalelor vertebrate. Atributele sunt reprezentate pe coloane, iar obiectele sunt rânduri. Un $\times$ într-o celulă înseamnă că există relația gIm, pentru atributul și obiectul respectiv.*

		respiră în apă (a)	zboară (b)	are cioc (c)	are mâini (d)	are schelet (e)	are aripi (f)	trăiește în apă (g)	naște pui vii (h)	produce lumină (i)
1	Liliac		×			×	×		×	
2	Vultur		×	×		×	×			
3	Maimuță				×	×			×	
4	Pește papagal	×		×		×		×		
5	Pinguin			×		×	×	×		
6	Rechin	×				×		×		
7	Pește lanternă	×				×		×		×

Tabela 1.1: Un context al animalelor vertebrate. Sursa: CDA [CR04]

Pentru $A \subseteq G$, definim $A' = \{m \in M | gIm, \forall g \in A\}$.

În mod asemănător, pentru $B \subseteq M$, $B' = \{g \in G | gIm, \forall m \in B\}$.

În cuvinte, A' este mulțimea tuturor atributelor (din contextul la care ne raportăm) care descriu toate obiectele din A.

Definiție 11. *Fie $A \subseteq G$, $B \subseteq M$, unde $A' = B$ și $B' = A$. Tuplul (A, B) este un **concept** al contextului (G, M, I).*

În engleză, mulțimea A (a tuturor obiecte descrise de atributele conceptului) este numită **extent** (în română *cuprins*), iar B (atributele care descriu toate obiectele conceptului) **intent**(în română *conținut*).

Fie (G, M, I) un context, și A, A_1, A_2 submulțimi ale lui G, iar B, B_1, B_2 submulțimi de-ale lui M. Conform [GW97], atunci:

1. $A_1 \subseteq A_2 \Rightarrow A_1^{'} \supseteq A_2^{'}$
2. $A \subseteq A'''$
3. $A' = A'''$
4. $A \subseteq B' \iff B \subseteq A' \iff A \times B \subseteq I$

Proprietăți echivalente se observă imediat și pentru B, B_1, B_2.

Având în vedere că $' : \mathcal{P}(G) \to \mathcal{P}(M)$ și $B' : M \to G$, cei doi operatori pot fi combinați pentru a crea A'' și G'', care au ca domeniu mulțimea submulțimilor G și M respectiv.

Se observă pornind de la proprietățile enumerate mai sus că cele două funcții de derivare descriu o conexiune Galois între mulțimile submulțimilor pentru obiecte ($\mathcal{P}(G)$) și ($\mathcal{P}(M)$)

În exemplul de mai sus, $\{2,4\}'' = \{2,4,5\}$, $\{d,h\}'' = \{d,e,h\}$.
Câteva proprietăți de remarcat ale conceptelor, așa cum sunt definite:

- Nu orice submulțime de obiecte definește cuprinsul unui concept. Din cele descrise mai sus, rezultă că e necesar ca $A = A''$ pentru A să fie cuprinsul unui concept.

  Ca exemplu, în tabelul 1.1, $\{6\}$ nu definește un concept, deoarece $\{6\}'' = \{4,6,7\}$, adică toate atributele care descriu rechinul în contextul nostru descriu de-asemenea și peștele lanternă, și peștele papagal.

- Intersecția oricâtor cuprinsuri (sau conținuturi) de concepte are ca rezultat întotdeauna un alt cuprins (respectiv conținut).

- În urma reuniunii lor, pe de altă parte, rareori rezultă un alt cuprins.

- Mulțimea conceptelor unui context este o mulțime ordonată, dacă definim o relație de ordine în felul următor:

  Definiție 12. *Fie (A_1, B_1) și (A_2, B_2) concepte ale contextului $K = (G, M, I)$. Spunem că (A_2, B_2) este un **subconcept** al lui (A_1, B_1) (notat $(A_2, B_2) \leq (A_1, B_1)$ dacă $A_2 \subseteq A_1$. Astfel, (A_1, B_1) este **supraconceptul** lui (A_2, B_2).*

Teoremă 1. ***Teorema de bază a laticelor de concepte*** *- Fie un context (G, M, I), și o mulțime ordonată $\mathcal{C}$ $(G, M, I; \leq)$ se numește laticea de concepte a contextului, care are supremumul și infimumul descrise de:*

$$\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)'' \right)$$

$$\bigvee_{t \in T} (A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t \right)'', \bigcap_{t \in T} B_t \right)$$

Exemplu 3. *Vom prezenta diagrama laticei de concepte derivate din tabelul 1.1. Se observă că supremumul are cuprins non-nul, dar infimumul da (pentru că nu există nici un animal în tabel cu toate prorpietățile). [CR04] Cum se citește diagrama? Gândindu-ne la definiția diagramelor Hasse (4), înseamnă că ne putem da seama dacă există o relație de ordine între două concepte (subconcept vs. supraconcept) dacă putem urmări o linie continuă între ele, cu supraconceptul afișat deasupra. Asta înseamnă că **toate** animalele descrise în context au atributul "au schelet" (deși asta era evident din descierea tabelului, fiind vorba de animale vertebrate). Alte deducții vizibile imediat e că nu toate animalele care au aripi pot zbura, și nu toate animalele care pot zbura sunt păsări.*

Se poate observa că diagrama de mai sus nu are etichete descriptive pentru fiecare concept reprezentat. Reprezentarea acelor informații pentru fiecare concept îngreunează diagrama și oferă multă informație redundantă, având în vedere că multe din concepte sunt subconceptele altor concepte. Totuși, cum hotărâm care etichete sunt afișate?

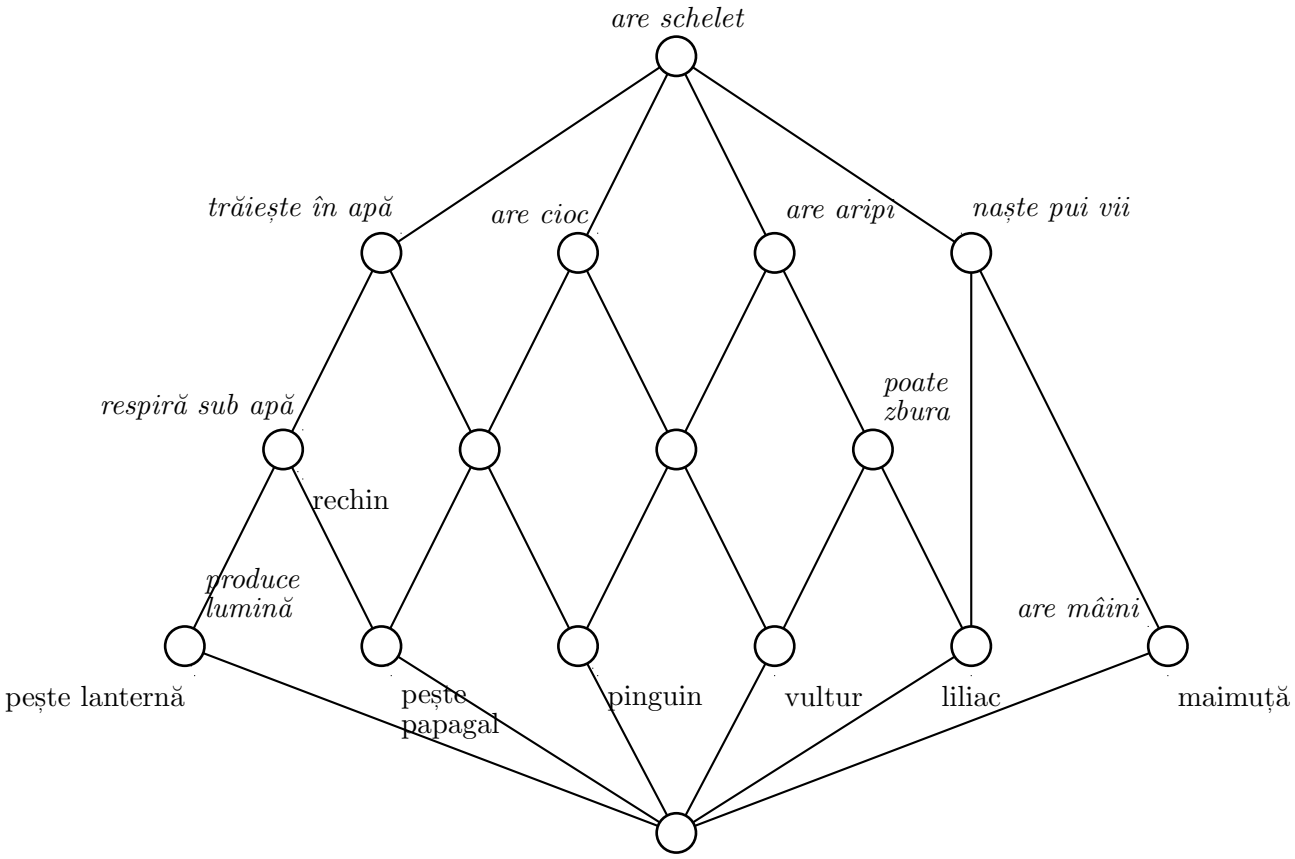

Figura 1.2: Laticea de concepte corespunzătoare contextului descris în tabelul 1.1.

Definiție 13. ***Conceptul obiectului*** $g \in G$ *este conceptul* (g'', g')*, unde* g' *este conținutul obiectului* $(\{m \in M \ gIm\})$ g*. Astfel, conținutul obiectului (notat cu* $\gamma(g)$*) este cel mai mic concept care-l are în cuprins pe* g*. Ca o paralelă,* (m'', m') *este* ***conceptul atributului*** *pentru un atribut* $m \in M$*,* m' *fiind cuprinsul atributului* $(\{g \in G | gIm\})$*. Conceptul atributului, notat cu* $\mu(m)$ *este cel mai mare concept care-l are pe* m *în conținut.*

Conform [GW97], laticile de concepte a mai multor contexte pot să fie isomorfe. Există manipulări ale contextelor pe care le putem face fără a alterastructura laticei de concepte, cum ar fi comasarea obiectelor cu același conținut și a atributelor cu același cuprins.

Definiție 14. *Fie un context* (G, M, I)*. Contextul este* ***clarificat*** *dacă* $\forall g, h \in G, g' = h' \Rightarrow g = h$ *și* $m' = n' \Rightarrow m = n, \forall m, n \in M$*.*

Laticea de concepte nu este influențată nici de prezența atributelor care pot fi scrise ca o combinație de alte atribute. Altfel spus, dacă $m \in M$ e un atribut și $X \subseteq M$ e o mulțime de atribute, iar $m \notin X$, dar $m' = X'$, atunci conceptul atributului μm e infimumul conceptelor atributelor $\mu x, x \in X$.

Altfel spus, putem omite atât atributele cât și obiectele reductibile, fără a influența laticea de concepte.

Revenind la etichetare, în loc să repetăm pentru concept care cuprinde animalul *rechin* această etichetă, putem să afișăm numele său doar pe conceptul obiectului. În loc de a îngreuna citirea diagramei, știind aceste concepte de bază diagrama devine mai lizibilă fără a fi încărcată.

- Reducerea și clarificarea contextelor
- Rezolvarea contextelor cu valori multiple

1.2.3 Contexte cu valori multiple

Contextele prezentate până acum, sunt foarte limitate prin prisma faptului că relația de incidență dintre obiect și atribut permite doar *existența* sau *lipsa* atributului pentru un anumit obiect. Dar lumea și obiectele pe care dorim să le descriem rareori pot fi descrise doar prin prezența sau lipsa unei proprietăți. Dacă descriem flori, unul din atributele de care-am vrea să ținem cont când construim conceptul este culoarea. Acest atribut nu poate fi descris simplu prin dihotomia *are/ nu are*.

Astfel, vedem nevoia de a introduce un nou concept, acela al **contextelor cu valori multiple**.

Definiție 15. *[CR04] "Fie un cvadruplu (G, M, V, I), format din 3 mulțimi G, M, V, G, mulțimea obiectelor, M, mulțimea atributelor cu valori multiple și V, mulțimea valorilor atributelor și dintr-o relație ternară între G, M și V (adică $I \subseteq G \times M \times V$) astfel încât $(g, m, v) \in I$ și $(g, m, w) \in I \Rightarrow v = w$"*

Putem citi $(g, m, v) \in I$ ca și "atributul m are valoarea v pentru obiectul g". Dar contextele cu valori multiple nu se pretează conceptualizării, cum am definit-o în secțiunea anterioară.

Pentru a putea crea concepte, contextele cu valor multiple se vor transforma în contexte simple, prin diferite metode, numite **scale**. Cea mai simplă este înlocuirea oricărui atribut cu valori multiple în tupluri formate din $M \times V$ (de exemplu pentru atributul *culoare* cu valori potențiale albastru, verde, roșu, creem atributele *culoare-albastru, culoare-verde, culoare-roșu*). Aceasta se numește o **scală nominală**.

Scalele nominale se potrivesc în cazul în care domeniul de valori al atributului pe care-l înlocuim este compus din valori care se exclud reciproc (cum e exemplul anterior, al culorilor petalelor unei flori - dacă nu luăm în considerare culorile ca fiind compuse din alte culori primare, valorile se exclud). Dar în cazul în care o valoare este subsumată de alta? Ca un exemplu, să luăm câteva valori posibile pentru atributul "luminescent": *luminos, foarte luminos, orbitor*

Se observă că primele valori posibile sunt subsumate în cele din urmă. Astfel, un bec este *luminos*, dar soarele este și *luminos*, dar și *orbitor*. Acesta este un exemplu de **scală ordinală**.

Scalele ordinale permit comparația într-un singur sens. În practică, de multe ori, vrem să aflăm domeniul în care se află o valoare, nu doar o margine inferioară sau superioară. Astfel, exemplul de mai sus poate fi reformulat cu valori cum ar fi $\leq 100lm$, $\geq 100lm$, $\leq 500lm$, $\geq 500lm$, $\leq 1000lm$, $\geq 1000lm$. Astfel, un bec care luminează cu 750 lumeni va avea atributele $\geq 100lm$, $\geq 500lm$, $\leq 1000lm$. Aceasta este o **scală intra-ordinală**.

Exemplu 4. *Pentru a demonstra scalarea, vom transforma un context cu valori multiple care se referă la becuri aflate în comerț într-un context cu valori simple, folosindu-ne de cele trei tipuri de scale descrise:*

1.3 Algoritmi relevanți

În secțiunea ce urmează vom prezenta câțiva algoritmi folositori pentru a prelucra contextele și laticile de concepte aferente acestora pentru a putea fi afișate.

Algoritmii descriși se împart în două categorii

	culoare	consum curent	luminescență
incandescent standard	caldă	100W	1690lm
bazat pe LED-uri	neutră	27W	1600lm
bec fluorescent	rece	26W	1750lm
bec cu halogen	neutră	43W	870lm

Tabela 1.2: Context cu valori multiple descriind becuri

	culoare			consum curent (în W)			luminescență (în lumeni)		
	caldă	neutră	rece	≥ 30	≥ 70	≥ 100	≥ 1000	≥ 1200	1400
incandescent standard	×			×	×	×	×	×	×
bazat pe LED-uri		×					×		

Tabela 1.3: Același context ca în tabelul 1.2, dar transformat în context cu valori simple

- algoritmi de construcție a laticei de concepte a unui context
- algoritmi de afișare eficientă a laticei de concepte

1.3.1 Algoritmi pentru construirea laticei de concepte

Un algoritm simplu de înțeles pentru construirea laticei de concepte a unui context este cunoscut sub numele de *Next Neighbors* - **Vecinii următori**. Numele explică ideea de bază a algoritmului: generarea iterativă a vecinilor (elementelor *acoperite*), începând dintr-un concept, în relație cu $\prec$.

Conform [CR04], algoritmul este analog unei parcurgeri în lățime a laticei finale. O variantă bazată pe parcurgea în adâncime e de asemenea posibilă.

Începând cu elementul din vârful laticii (G, G'), algoritmul construiește un nivel odată, unde următorul nivel conține conceptele acoperite de toate conceptele prezente în nivelul curent. Mai exact, pentru fiecare concept din nivelul curent o funcție *GasesteElementeleAcoperite* (descrisă mai jos 1.3) e apelată care calculează conceptele acoperite de acel concept; funcția se bazează pe observația că toate conceptele acoperite de un concept se află într-o submulțime redusă de concepte - fiecare se obține prin a adăuga un atribut nou conținutului ($Y_1 = Y \cup \{m\}$) și a calcula $(Y_1^{'}, Y_1^{''})$ apoi se verifică dacă fiecare concept acoperit rezultat nu a mai fost generat anterior, caz în care conceptul este adăugat laticei și în final conceptul e legat de părintele său (e adăugată muchia).

Laticea rezultată este strucutrată ca un tuplu de două mulțimi, (C, E), C fiind mulțimea conceptelor rezultate și E mulțimea laturilor dintre acestea, iar laturile sunt reprezentate ca și perechi ordonate de concepte $(c_1, c_2), c_1, c_2 \in C, c_1 < c_2$.

Avantajul acestui algoritm față de alții care au același scop (de a determina laticea de concepte a unui context) este că generează atât mulțimea conceptelor a unui context dar și diagrama (*conexiunile*) acestuia.

Complexitatea algoritmului este de $O(C \times G \times M^2)$. Deși există algoritmi cu o complexitate mai mică (neglijabil, conform [CR04]), algoritmul prezentat a fost ales deoarece este ușor de urmărit și prezintă conceptele descrise în prima secțiune.

Date de intrare: Context (G, M, I)
Date de ieșire: Latice (C, E)

funcție VECINII URMĂTORI(Context (G, M, I))
  $C := \{(G, G')\}$
  $E := \varnothing$
  $nivelulCurent := \{(G, G')\}$
  cât timp $nivelulCurent \neq \varnothing$ **execută**
    $nivelulUrmator := \varnothing$
    pentru $(X, Y) \in nivelulCurent$ **execută**
      $elementeleAcoperite := GasesteElementeleAcoperite(X, Y)$
      pentru $(X_1, Y_1) \in elementeleAcoperite$ **execută**
        dacă $(X_1, Y_1) \notin C$ **then**
          $C := C \cup \{(X_1, Y_1)\}$
          $nivelulUrmator := nivelulUrmator \cup \{(X_1, Y_1)\}$
        $E := E \cup (X, Y) \rightarrow (X_1, Y_1)$
    $nivelulCurent := nivelulUrmator$
  întoarce $L = (C, E)$
funcție $GasesteElementeleAcoperite((X, Y))$
  $candidati := \varnothing$
  pentru $m \in M \backslash Y$ **execută**
    $X_1 := (Y \cup \{m\})'$
    $Y_1 = X_1^{'}$
    dacă $X_1, Y_1 \notin candidati$ **then**
      $candidati := candidati \cup \{(X_1, Y_1)\}$
  întoarce $candidati$

Figura 1.3: Algoritmul Vecinii următori, Sursa [CR04]

1.3.2 Algoritmi pentru vizualizarea laticei de concepte

De-a lungul acestei lucrări, vorbim foarte mult despre diagramele laticelor de concepte și despre importanța *explorării* acestora, dar până acuma nu am menționat cum sunt generate aceste diagrame. Acesta este un domeniu dificil, întrucât un algoritm de generare trebuie să fie, în cele din urmă optimizat pentru lizibilitatea diagramei pentru oameni. Dispunerea nodurilor în așa fel încât să existe cât mai puține intersecții (acestea scad lizibilitatea diagramei drastic), păstrarea unei diagramei între anumite dimensiuni care să permită înțelegerea spațiului de căutare sunt doar câteva dintre provocările pe care le prezintă acest domeniu. Viteza de generare a diagramei este un alt criteriu important, mai ales când avem de-a face cu contexte complexe.

Dincolo de orice încercări de a reduce dimensiunile diagramelor, în cazul datelor reale, cu sute, sau chiar mii de concepte și atribute, majoritatea diagramelor laticelor de concepte se află în această situație. Acest lucru relevă nevoia pentru navigatoare dinamice a contextelor. Acestea trebuie să poată ascunde sau afișa doar părți dintr-o diagramă, în funcție de cerințele utilizatorului și a manipulărilor pe care acesta le realizează asupra conceptelor, relațiilor dintre ele, etc.

1.4 Utilizări practice

Capitolul 2
Starea actuală

În clipa de față există multe programe folosite pentru diferite aspecte ale analizei conceptuale formale. O listă mai dezvoltată, care adună majoritatea programelor disponibile poate fi găsită la [Pri07].

Mai jos vom discuta doar câteva programe care au influențat dezvoltarea Romagnei, sau sunt relevante din motive istorice, arhitecturale, etc.

2.1 Navigatoare de concepte

2.1.1 Toscana

Toscana[VW95] a fost lansat în 1995, și a fost unul din cele mai folosite unelte de explorare a laticelor de concepte pe parcursul următorilor ani. Aplicația a fost scrisă în C și folosea un format de date proprietar, bazat pe text.

Toscana este astăzi probabil cel mai bine ținută minte ca precursorul programului ToscanaJ, folosit și astăzi, prezentat în detaliu mai jos (2.1.2).

În ciuda eforturilor noastre, unealta nu a fost găsită pentru descărcare.

2.1.2 Toscanaj

ToscanaJ([Dev14a]) este "moștenitorul direct" al lui Toscana, lucru evidențiat și de nume (J-ul vine de la Java).

După cum spune chiar site-ul programului [Dev14b]: "E o unealtă de vizualizare pentru scheme conceptuale foarte avansată, care reușește să afișeze informație interogată dintr-o bază de date în diagrame de latice, sau direct din structuri de date luate din memorie."

2.1.2.1 Funcționalități

Din nou, citând site-ul programului[Dev14b], prezentăm câteva funcționalități ale programului:

"

- Afișarea diagramelor simple și imbricate.
- Culoarea unui nod reprezintă mărimea contingentului obiectelor (poate fi modificat să reprezinte cuprinsul), deasemenea mărimea nodului poate fi folosită pentru același tip de informație.
- Mulțimea de obiecte de interes poate fi filtrată printr-un dublu click asupra nodurilor

- Nodurile din diagramă pot fi selectate pentru a fi scoase în evidență, pentru a ajuta citirea *[n.t. diagramei]*.
- Diagramele pot fi exportate ca SVG, PNG și JPEG. Informații adiționale despre cum diagrama a fost obținută sunt exportate ca fișiere text separate, prin memoria temporară a calculatorului *(clipboard)*, sau direct în fișierul SVG (ca elementul `<desc>`).
- Etichetele nodurilor pot avea conținut diferit, folosindu-se de fragmentele de SQL specifice datelor
- Vizualizări adiționale a bazei de date pot fi deschise din diagramă, de exemplu folosind șabloane HTML în care rezultatele interogărilor sunt afișate.
- Interfața de vizualizare a bazei de date a fost gândită ca o interfață pentru pluginuri pentru a ușura extinderea ToscanaJ pentru scopuri specifice.
- Descrieri HTML pot fi atașate schemei, diagramelor și atributelor
- Vederile bazelor de date pot fi folosite pentru atribute, de exemplu pentru a interoga un URL din baza de date care e mai apoi deschis într-un navigator extern.

”

Având în vedere că scopul programului Romagna este de a oferi o alternativă programului Toscana(J), aceste funcționalități se vor regăsi și în Romagna, alături de altele, descrise în capitolul 3.1.

Mai jos prezentăm două capturi de ecran care prezintă programul ToscanaJ.

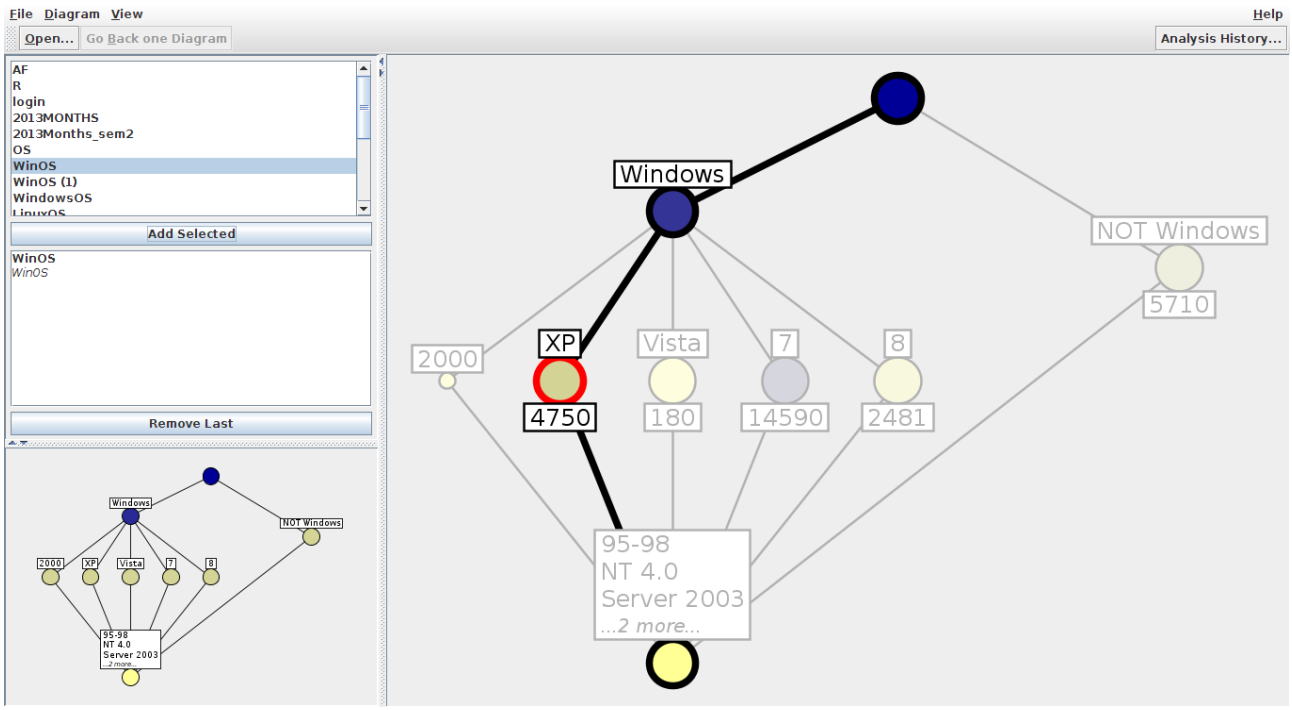

Figura 2.1: ToscanaJ, afișând o latice simplă generată din date de acces ale unui site web

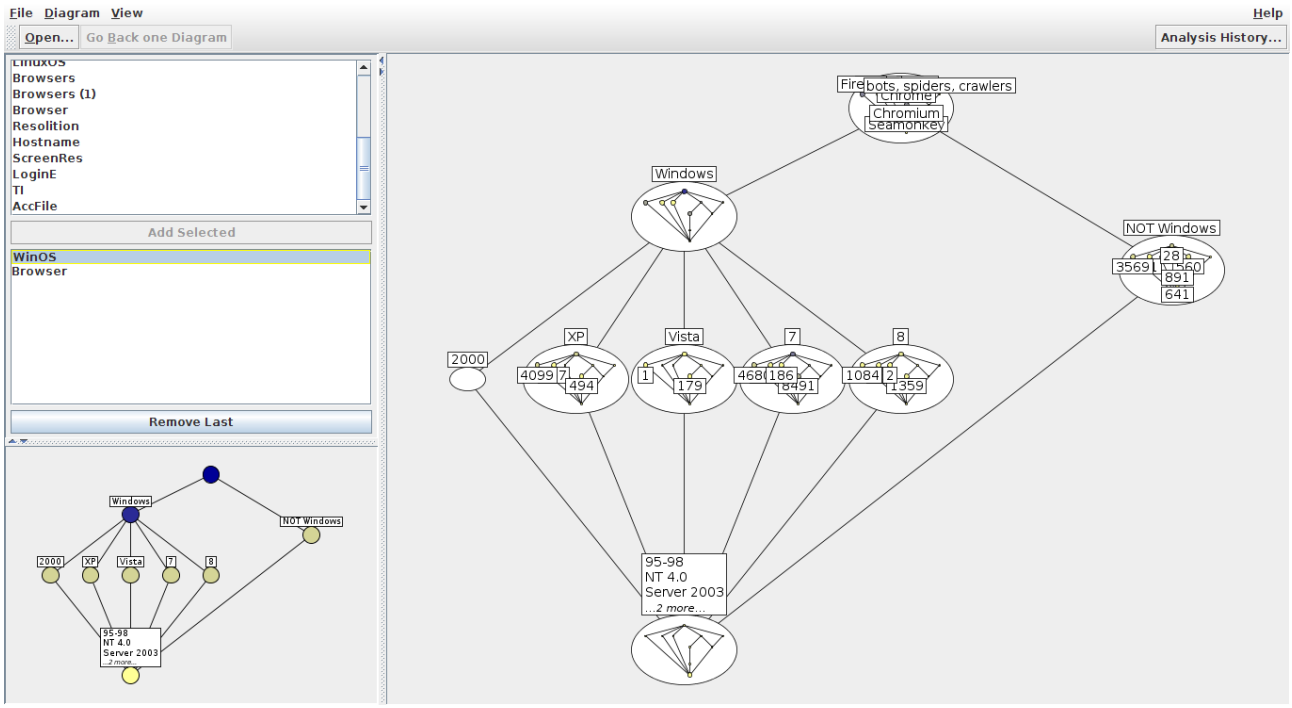

Figura 2.2: ToscanaJ, afișând aceeași latice ca în 2.1, de data aceasta imbricată cu altă diagramă

Interfața ToscanaJ este punctul de pornire pentru Romagna. Vom încerca să păstrăm anumite elemente comune, pentru a ajuta utilizatorii obișnuiți, dar vom face, bine-nțeles schimbări și adaptări, mai ales luând în considerare diferențele convențiilor dintre interfețele aplicațiilor web, și cele desktop.

ToscanaJ este, asemenea Romagna, doar un navigator de concepte. Modelarea conceptelor din date este realizată cu ajutorul altor programe din suita din care face parte și ToscanaJ.

Elba este un editor pentru schemele conceptuale, legat de baze de date.

Siena este un editor pentru scheme conceptuale, foarte asemănător cu **Elba**, diferența fiind că Siena nu are nevoie de o legătură cu o bază de date, putând descrie concepte simple direct în program.

2.1.3 GaloisExplorer

GaloisExplorer [tea09] este un program mai recent, cu ultima actualizare în 2009.

Are o intefață realizată în QT, ceea ce îi permite să funcționeze pe mai multe platforme. O abordare interesantă, dar în cele din urmă doar cu valoare estetică, este prezentarea laticelor în 3d (2.3).

Din păcate, folosește anumite librării 3d (coin3d[**?**]) care nu sunt imediat disponibile și care necesită compilare manuală.

Pentru mulți utilizatori care nu au cunoștințe avansate de calculatore, aceste impedimente sunt motiv suficient pentru a abandona această instalare.

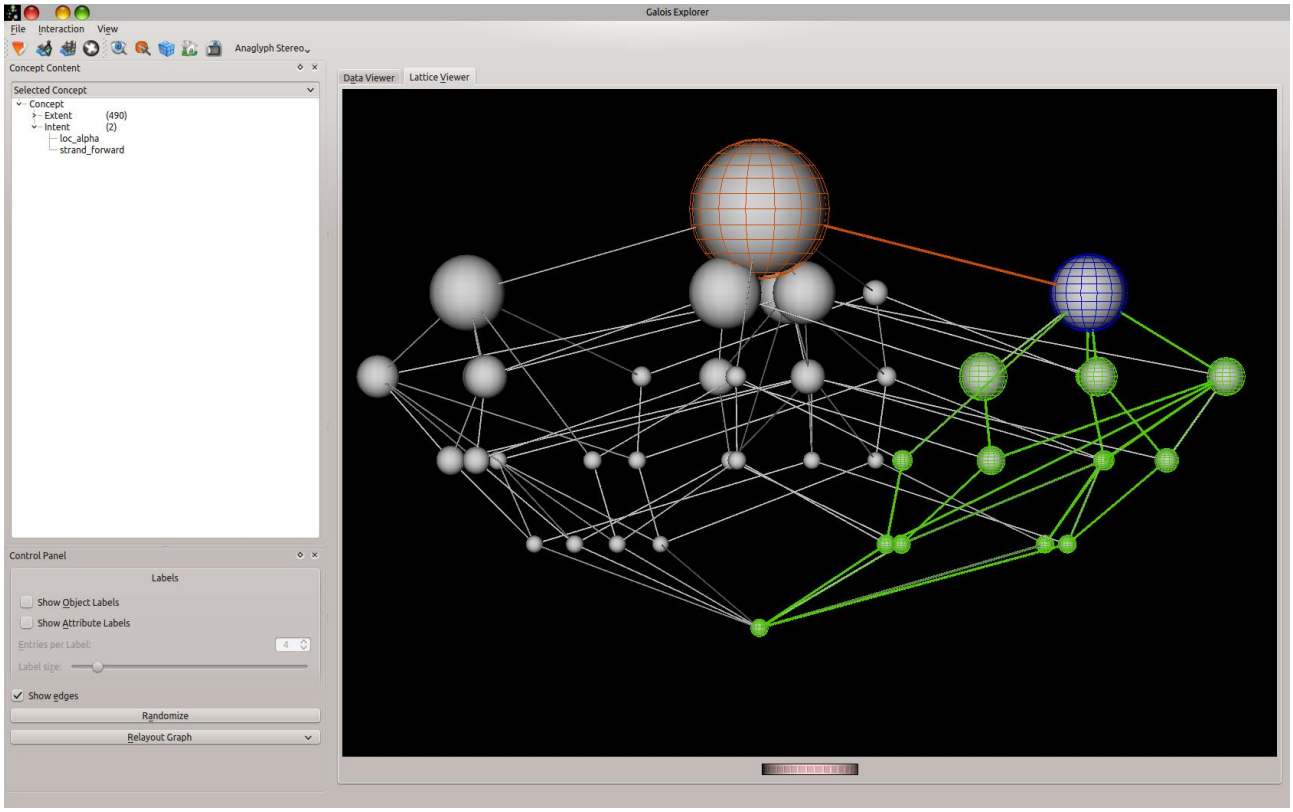

Figura 2.3: GaloisExplorer, afișând o diagramă în 3d. Sursa: site-ul proiectului [tea14]

2.1.4 Galitia

...

2.2 Software conex

În această secțiune vom prezenta câteva alte programe remarcabile în domeniul FCA, sau care au avut un impact indirect asupra dezvoltării Romagna:

FCAStone dezvoltat de Uta Priss, e un program care urmărește obținerea inter-operabilității între diferite programe pentru FCA (numele vine de la Piatra *[n.t. Stone = Piatră]* Rosetta). Astfel, convertește fișierele folosite de mai multe suite în FCA. Pe lângă asta, convertește contexte în latice de concepte, și latice de concepte în formate grafice (atât vectoriale cât și raster).

Conexp dezvoltat de Serhiy A. Yevtushenko, este o unealtă scrisă în Java, folosită în principal în scopuri educaționale (dar nu numai), are numeroase funcționalități de modelare a contextelor, dintre care amintim (preluate de site-ul programului [Yev14]):

- calculează numărul de concepte formale dintr-un context dat
- calculează laticea de concepte
- permite explorarea atributelor
- calculează regulile de asociații

conexp-clj O reimplementare scrisă de Daniel Borchmann în limbajul de programare Clojure a programului de mai sus. Programul pune accentul pe linia de comandă și programarea interactivă în domeniul FCA.

OpenFCA Este o suită de trei aplicații, din care Conflexplore este o aplicație bazată pe Flex, folosită pentru explorarea conceptelor, atât tabelar cât și printr-o formă simplă de latice de concepte.

LaTeXfor FCA [Gan14] Plugin pentru LaTeX, scris de Bernhard Ganter, oferă câteva "environment"-uri noi, folosite și în această lucrare, care permite aranjarea ușoară în pagina a contextelor și a laticilor derivate din acestea.

Capitolul 3
Romagna

Romagna este o aplicație dezvoltată începând cu anul 2014, pornită la Universitatea Babeș-Bolyai ca o alternativă folosindu-se de tehnologiile web pentru un navigator de concepte modern.

Numele (Romagna) este o referință directă suitei de programe care a inspirat această aplicație, Toscana. Romagna este o regiune istorică a Italiei aflată la nordul Toscanei.

Pe lângă funcționalitățile ale aplicației ToscanaJ descrise în sub-subsecțiunea 2.1.2.1, Romagna are câteva funcționalități noi plănuite.

Dezvoltarea acestei aplicații are câteva scopuri *(diferite de funcționalități)* principale, prezentate în secțiunea următoare.

3.1 Raționament

3.1.1 Ușurință de utilizare

Uneltele destinate FCA trebuie gândite ca programe pentru consumatori, utilizatori obișnuiți. Publicul țintă al acestor aplicații nu sunt neapărat programatori sau oameni cu experiență în folosirea avansată a calculatoarelor. Oamenii care beneficiază cel mai mult de asemenea aplicații sunt cercetători în domenii foarte diferite (după cum se vede în secțiunea de aplicații practice 1.4)

Din acest punct de vedere, este părerea autorului că soft-ul existent în acest domeniu nu satisface necesitățile publicului său țintă.

Pentru clarificare, următoarele puncte trebuie îndeplinite pentru a considera o aplicație ușor de folosit:

- Aplicația trebuie să fie **ușor de instalat**. Programatorii de multe ori scapă din vedere acest aspect, dacă nu e vorba de un produs comercial, care are nevoie de cât mai mulți utilizatori. E inadmisibil să se ceară compilarea manuală a unei librării C++, sau instalarea unui server MySQL unui utilizator obișnuit pentru a utiliza un program. Romagna, fiind o aplicație web, sare peste acești pași. Singurii pași pre-mergători folosirii aplicației sunt pregătirea datelor pentru consum de către aplicație și familiarizarea cu interfața aplicației.

- Aplicația trebuie să fie **disponibilă pe mai multe platforme**. Utilizatorii nu vor instala un alt sistem de operare pentru a testa o aplicație. În schimb, majoritatea utilizatorilor au la dispoziție un navigator web modern, cum ar fi Mozilla Firefox, Google Chrome, etc. Aceste medii de aplicații (pentru că navigatoarele web și mediul de execuție al limbajului JavaScript pus la dispoziție de acestea a devenit un mediu de aplicații) se actualizează

automat, permițând utilizarea celor mai noi funcționalități unei majorități a utilizatorilor acestora.

- Aplicația trebuie să ofere **date de ieșire utile** utilizatorului. Ne referim aici bine-nțeles la rezultatele așteptate de utilizator, prezentate într-un mod clar, dar și la mesajele de erori, care trebuie să fie inteligibile, să ofere cursuri de acțiune care să rezolve problema întâmpinată, într-un limbaj non-tehnic, pe cât posibil. Erorile criptice, cum ar fi stack-trace-uri, sau excepții afișate pe ecran descurajează utilizatorul obișnuit.

- Aplicația trebuie să aibă **o interfață plăcută**. E părerea autorului că, deși utilizabile, interfețele generate de aplicații Java în general oferă o experiență neplăcută, nu se integrează în stilul sistemului de operare. Aplicațiile web sunt mult mai ușor de estetizat, mai ales cu ajutorul unor framework-uri CSS, dintre care amintim Twitter Bootsrap sau Zurb Foundation.

- Ca o ultimă alternativă, aplicația trebuie să vină **însoțită de documentație**. Aceasta trebuie formulată într-un limbaj non-tehnic pentru utilizatori.

- Opțional, aplicația ar trebui să aibă **interfața localizată**. Nu toți utilizatorii de calculatoare sunt vorbitori de engleză. Folosirea pictogramelor pentru realizarea interfeței poate ajuta, dar inclusiv în acest caz, diferențele culturale, sau conceptele prea complexe pe care dezvoltatorii încearcă să le transmită, pot descuraja utilizarea aplicației mai mult decât să ajute.

Romagna îndeplinește unele din condițiile de mai sus(ușor de instalat, ușor de găsit) prin natura aplicațiilor web, iar celelalte condiții constituie obiectul efortului autorului.

3.1.2 Folosirea web-ului, păstrarea confidențialității datelor

Este o stare de fapt că în domeniul analizei datelor, integritatea și confidențialitatea acestora este o grijă constantă.

Mulți cercetători lucrează de asemenea cu date care sunt, legal vorbind, secrete. Se înțelege astfel, că majoritatea potențialilor utilizatori **nu** vor avea încredere să trimită datele unui serviciu extern.

Din aceste considerente, o arhitectură server peste web (și oferirea Romagnei ca serviciu) a fost imediat desconsiderată. Dar, cum am menționat în subsecțiunea 3.1.1, ușurința de utilizare este o altă calitate pe care încercăm să o regăsim în Romagna.

S-a ajuns astfel la compromisul (temporar) de a realiza **totul** în browser. Asta înseamnă accesarea datelor SQL în browser, prin tehnologii descrise în secțiunea **??**.

3.2 Tehnologii

După cum am menționat, Romagna se bazează pe tehnologii web. Vom continua prin a descrie sumar principalele librării/framework-uri folosite și un raționament scurt pentru existența lor în proiect.

3.2.1 CoffeeScript

Unul din principalele motive pentru care browser-ul nu a fost mult timp acceptat ca o platformă matură pentru aplicații era absența altor limbaje de programare disponibile înafară de JavaScript.

Această stare de fapt se schimbă încet din mai multe motive:

- JavaScript a evoluat și este considerat un limbaj de programare mai matur decât în perioada în care a fost scris ToscanaJ.
- Existența **emscripten**, descris pe scurt în subsecțiunea 3.2.4.1 și rezultatele obținute de acesta.
- Existența limbajelor care compilează *în* JavaScript.

Unul din aceste limbaje este **CoffeeScript** [**?**]. CoffeeScript nu introduce decât două concepte (clase versus prototipuri și array comprehensions) noi față de JavaScript, dar simplifică dezvoltarea prin câteva îmbunătățiri:

- Evitarea variabilelor globale
- Câteva scurtături de sintaxă (`->` pentru a declara o funcție, `=>` pentru a declara o funcție legată de contextul actual, etc.)
- Clarificarea operatorilor de comparație (`==` în CoffeeScript devine automat `===` în JavaScript, scăpând automat de o întreagă clase de erori).

Alături de sintaxa mai prietenoasă (o opinie personală a autorului), prin elidarea acoladelor în favoarea indentării pentru definirea blocurilor, am ales CoffeeScript ca limbajul principal pentru Romagna.

3.2.2 Ember.js

Ember.js este un framework scris în JavaScript, inspirat de Cocoa, și are ca scop ușurarea dezvoltării aplicațiilor web complexe, prin oferirea unui cadru MVC, cu o structură relativ rigidă.

Inițial aplicația nu era bazată pe niciun framework. Pe măsură ce complexitatea structurii a crescut, am realizat că recreem, involuntar, un sistem MVC, mai mult ca sigur imperfect.

Am hotărât să reorganizăm codul în urma unei deliberări, bazându-ne pe faptul că un framework folosit de mii de oameni, cu teste funcționale și modulare la zi poate oferi o fundație puternică, fără a ne cheltui timpul "re-inventând roata". Din variantele disponibile de framework-uri pentru JavaScript, am ales Ember.js deoarece:

- Structura impusă se potrivește cu necesitățile proiectului.
- Posibilitatea schimbării ușoare a adaptorului a repozitoriului de date inclus în framework înseamnă că vom putea crea o variantă viitoare a Romagna care se includă comunicarea cu un server, funcționalitate care este plănuită pentru versiuni care vor urma.

- Funcționalitatea componentei `Router`, care permite crearea de URL-uri, care pot fi salvate de utilizator, în funcție de navigarea sa prin aplicație, înseamnă că avem la îndemână o formă simplă de serializare și salvare a stării aplicației.

3.2.3 d3.js

D3 (provenit din *Data-Driven Documents* - Documente Bazate pe Date) este o librărie scrisă în JavaScript pentru manipularea DOM-ului relaționând cu date legate de elemente din document. Citând din abstractul lucrării [BOH11]:

> "Cu D3, designerii leagă în mod selectiv date de intrare de elemente arbitrare ale documentului, aplicând transformări dinamice pentru a genera, cât și a modifica conținut."

Devine clar din descrierea de mai sus că această librărie va ușura dezvoltarea aplicației, datorită posibilității de a lega *conceptele* de reprezentarea acestora într-un mod care să permită modificarea acestei reprezentări în funcție de explorarea diagramelor de către utilizator, ceea ce, este, până la urmă scopul aplicației.

3.2.3.1 svg

D3 se bazează pe manipularea DOM-ului pentru a obține vizualizări dinamice de date. Deși HTML-ul poate fi stilizat, formele complexe și imaginile vectoriale sunt descrise în browser prin SVG [Fer01], un standard de grafică vectorială vazată pe XML. Deși ajută la crearea unor aplicații de vizualizare complexe, SVG-ul vine cu problemele sale, care sunt descrise mai pe larg în sub-subsecțiunea 3.4.2.2.

3.2.4 sql.js

După cum am descris în secțiunea 3.1.2, pilonii pe care am decis să construim Romagna (ușurința de utilizare și păstrarea locală a datelor) pun în dificultate folosirea unei arhitecturi client-server, deoarece vrem un proces de instalare și folosire cât mai simplu (nici un proces de instalare nu e mai simplu decât deschiderea unei pagini web), iar utilizatorii nu-și vor trimite datele unei părți terțe, din temeri proprii sau pentru că nu au acest drept, lucrând cu date confidențiale.

Astfel, compromisul este de a analiza SQL în browser, cu ajutorul sql.js [Zak14], care este defapt SQLite re-compilat în JavaScript cu ajutorul emscripten.

SQL.js poate rula într-un Web Worker [Hic12], echivalentul unui thread separat de execuție în JavaScript în browser. Astfel, operațiile de interogare asupra bazei de date nu au un impact direct asupra răspunsului interfeței.

3.2.4.1 emscripten

Din abstractul articolului [Zak11]

> "[...] prezentăm Emscripten, un compilator din limbaj de asamblare LLVM (Low Level Virtual Machine) în Javascript. Acesta deschide două căi de a rula cod scris în alte limbaje decât Javascript pe web: 1. Compilarea codului direct în limbaj de asamblare LLVM și apoi compilarea acestuia în JavaScript folosindu-vă de Emscripten, sau 2. compilarea întregului run-time al unui limbaj interpretat în LLVM și apoin JavaScript."

Impactul pe care acest compilator l-a avut este, cel puțin până acum, unul mai mult teoretic, majoritatea programelor fiind folosite cu rol demonstrativ. Aplicațiile compilate în JavaScript sunt, binențeles, mai lente decât variantele lor compilate direct în limbaj de asamblare. Totuși pentru o anumită categorie de programe, între care se află și Romagna, acest schimb de viteză pentru funcționalitățile oferite de librăria sau aplicația compilată este binevenit. Structurile de date folosite de Romagna nu sunt atât de mari încât pierderea performanței să fie critică.

3.3 Structură

3.4 Dezvoltare

3.4.1 Proces

3.4.2 Probleme întâmpinate

3.4.2.1 MySQL versus sql.js

3.4.2.2 SVG și probleme în afișarea corectă a textului

3.5 Viitor

Concluzii

Bibliografie

[BOH11] M. Bostock, V. Ogievetsky, and J. Heer. D3; data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, Dec 2011.

[CR04] Claudio Carpineto and Giovanni Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, 2004.

[Dev14a] ToscanaJ Developers. Toscanaj homepage. `http://toscanaj.sourceforge.net/`, 2014. [Online; accesat la 15-Iunie-2014].

[Dev14b] ToscanaJ Developers. Toscanaj homepage. `http://toscanaj.sourceforge.net/toscanaJ/index.html`, 2014. [Online; accesat la 15-Iunie-2014].

[Fer01] Jon Ferraiolo. Scalable vector graphics (SVG) 1.0 specification. W3C recommendation, W3C, September 2001. http://www.w3.org/TR/2001/REC-SVG-20010904.

[Gan14] Bernhard Ganter. Latex for fca. `http://www.math.tu-dresden.de/~ganter/fca/`, 2014. [Online; accesat la 15-Iunie-2014].

[GW97] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1997.

[Hic12] Ian Hickson. Web workers. Candidate recommendation, W3C, may 2012. http://www.w3.org/TR/2012/CR-workers-20120501/.

[Pri07] Uta Priss. Formal concept analysis homepage. `http://www.upriss.org.uk/fca/fcasoftware.html`, 2007. [Online; accesat la 15-Iunie-2014].

[tea09] GaloisExplorer team. Galoisexplorer: Project web hosting - open source software. `http://galoisexplorer.sourceforge.net/`, 2009. [Online; accesat la 15-Iunie-2014].

[tea14] GaloisExplorer team. Galoisexplorer — free science & engineering software downloads at sourceforge.net. `http://sourceforge.net/projects/galoisexplorer/`, 2014. [Online; accesat la 15-Iunie-2014].

[VW95] Frank Vogt and Rudolf Wille. Toscana — a graphical tool for analyzing and exploring data. In Roberto Tamassia and IoannisG. Tollis, editors, *Graph Drawing*, volume 894 of *Lecture Notes in Computer Science*, pages 226–233. Springer Berlin Heidelberg, 1995.

[Wil82] Rudolf Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered Sets*, pages 445–470, Dordrecht/Boston, 1982. Reidel.

[Yev14] Serhiy A. Yevtushenko. The concept explorer. `http://conexp.sourceforge.net/users/documentation.html`, 2014. [Online; accesat la 15-Iunie-2014].

[Zak11] Alon Zakai. Emscripten: An llvm-to-javascript compiler. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, SPLASH '11, pages 301–312, New York, NY, USA, 2011. ACM.

[Zak14] Alon Zakai. kripken/sql.js. `https://github.com/kripken/sql.js/`, 2014. [Online; accesat la 15-Iunie-2014].