„Interform” și „Abstract Machine”
Fundamentul meu conceptual pentru proiectare

1. Fundamentele mele conceptuale: interforma și mașina abstractă
În anii ’80 și începutul anilor ’90, ingineria software era în plină tranziție: de la o activitate artizanală, dominată de intuiție și improvizație, spre o disciplină inginerească, în căutarea rigorii, a metodei și a formalizării. Termeni precum design methodology, software process și software architecture începeau să circule, dar nu aveau încă o greutate conceptuală comună. Într-un context tensionat – între haosul industrial al programării în mare și nevoia de control ingineresc – am simțit nevoia unui cadru unificator.

Pentru a da o formă coerentă și operațională concepției mele despre proiectarea software, am propus, în cartea mea „Integrating Tools for Software Development”, două concepte fundamentale care pot ghida proiectanții în organizarea întregului univers al programării: *interforma* (Interform) și *mașina abstractă* (Abstract Machine, AM). Aceste concepte nu sunt simple construcții teoretice, ci expresia unei viziuni inginerești menite să controleze și să structureze complexitatea inerentă proiectării în cadrul procesului software. Prin aceste două instrumente – interforma ca *spațiu cognitiv de reprezentare* și mașina abstractă ca *mecanism de guvernare a deciziei* – am reușit să propun o arhitectură inginerească a gândirii proiectante. Este o contribuție semnificativă, tocmai pentru că răspundea direct la provocările „programării în mare” (*programming in the large*) care intrase tocmai în criză: complexitate, trasabilitate, coerență și colaborare.

A) Conceptul de *interformă* a apărut ca răspuns la două limitări majore:
a) **Limitarea cognitivă** a proiectantului uman de a înțelege sisteme complexe ca un întreg coerent;
b) **Fragmentarea excesivă a documentației și a modelelor de adoptat**, care, în lipsa unei infrastructuri semantice comune, conduceau la pierderea coerenței în procesul de dezvoltare.

Interforma oferea o reprezentare multidimensională a entității software aflate în construcție — nu doar ca structură de cod, ci ca logică, funcție, comportament și arhitectură.
Această abordare își poate dovedi întreaga valoare în proiectele de mare complexitate – ceea ce numeam, în anii ’80–’90, programming in the large. În astfel de contexte, în care sistemele depășesc milioane de linii de cod, cu echipe distribuite geografic și numeroase cerințe deseori contradictorii, complexitatea nu mai putea deriva dintr-o simplă sumă de componente. Ea rezulta din interacțiuni nelineare între acestea, din dependențe volatile și din dinamica instabilă a specificațiilor.

Interforma este o construcție inginerească originală, introdusă de mine ca răspuns tocmai la această complexitate a proiectării software în cadrul sistemelor mari. Ea reprezintă o **reprezentare stratificată, coerentă și evolutivă a unei entități software**, aflate în proces de dezvoltare. Scopul său este să permită înțelegerea, proiectarea, comunicarea în echipă și validarea entității printr-o **pluralitate de fațete**, fără a sacrifica unitatea semantică a entității descrise.
Spre deosebire de modelele fragmentare (diagrame UML, specificații izolate, documentații ad-hoc), interforma funcționează ca **o rețea de perspective (fațete) corelate**, fiecare reflectând un tip de cunoștințe asupra entității modelate necesare pentru a construi, controla și înțelege produsul software
Această pluralitate este intenționat **redundantă și ortogonală**: interforma nu elimină suprapunerile dintre descrieri, ci le integrează și le folosește pentru a garanta robustețea proiectării. Fiecare fațetă este, ideal, exprimată într-un limbaj formal sau semi-formal (ex: automat finit, diagramă de stare, pseudo-cod, logică de predicat), dar poate include și expresii în limbaj natural, acolo unde formalizarea este inadecvată sau prematură.
Interforma este, de asemenea, **recursivă**: orice subcomponentă relevantă a sistemului – fie ea structurală, comportamentală sau funcțională – poate fi, la rândul ei, descrisă printr-o interformă proprie. Astfel, arhitectura unui sistem devine un **arbore de interforme**, cu nivele de granularitate și detaliu adaptabile în funcție de scopul proiectantului.
Poziționare conceptuală:
1. Față de **documentație**: interforma nu este un simplu text explicativ, ci o construcție **activă, modificabilă, executabilă în anumite fațete;**
2. Față de **modele UML**: interforma nu este o colecție de diagrame izolate, ci un nucleu semantic coerent, în care fațetele sunt legate conceptual și logic;
3. Față de **use-case-uri**: interforma poate include sau corespunde unui use-case extins, dar cu o abordare mai formalizată și stratificată;

4. Față de **artefactele MDA (Model Driven Architecture)**: interforma nu separă modelele pe niveluri CIM/PIM/PSM, ci le tratează ca fațete coexistente ale aceleiași entități.

Din această perspectivă, interforma este un limbaj inginerește riguros pentru a vorbi despre un produs software încă neimplementat, dar deja modelat, înțeles, verificabil.

B) Al doilea concept, *mașina abstractă* (Abstract Machine), venea să ofere o *dinamică inginerească* procesului decizional: nu doar stocarea descrierilor, ci simularea, evaluarea și controlul raționamentelor din timpul proiectării. Aceasta continua o tradiție formală inspirată din teoria automatelor și a limbajelor formale, dar cu un scop ingineresc clar: susținerea unei practici de proiectare reproductibile, explicabile și transferabile între membrii unei echipe.

La rândul lor, *mașinile abstracte* pot fi compuse, rafinate și organizate ierarhic sau colaborativ pentru a modela activități, agenți și unelte din procesul proiectării. Împreună, aceste două instrumente oferă un cadru inginerește coerent, capabil să transforme actul de proiectare într-un proces rațional, controlabil și reproductibil.

În aceste condiții, proiectarea disciplinată, condusă de principii clare precum cele oferite de interformă și AM, devine esențială pentru menținerea coerenței și evolutivității produsului software.

2. Originea interformei: un concept inginerește necesar

Privind retrospectiv, consider *interforma* drept una dintre cele mai substanțiale și fertile idei pe care le-am formulat în domeniul ingineriei software. Nu este vorba de o tehnologie sau de un algoritm, ci de un cadru conceptual, născut dintr-o nevoie stringentă: aceea de a înțelege, structura și susține decizia inginerească în contexte de maximă complexitate ale producției de software — în ceea ce ulterior a fost denumit *programming in the large*.

Termenul *interformă* îmi aparține. L-am introdus pentru a denumi o *formă intermediară* de reprezentare a unei entități software aflate în proces de construcție. Inspirația directă a venit dintr-un articol celebru al lui Marvin Minsky, „A Framework for Representing Knowledge" (1974), în care autorul propunea o structură de reprezentare modulară a cunoștințelor prin rețele de cadre (frames). Acestea funcționau ca șabloane cognitive pentru situații tipice, în care anumite elemente (scheletul) erau fixe, iar altele, variabile sau contextuale.

Interformele mele au un scop convergent cu cadrele — organizarea și articularea cunoașterii inginerești — dar se deosebesc fundamental de cadrele lui Minsky prin natura lor operațională, regulile de formare și dinamica procesuală. Dacă cadrele erau statice și orientate pe reprezentarea stereotipurilor cognitive, interformele sunt structuri vii, evolutive, destinate organizării pluralității de viziuni necesare proiectării riguroase.

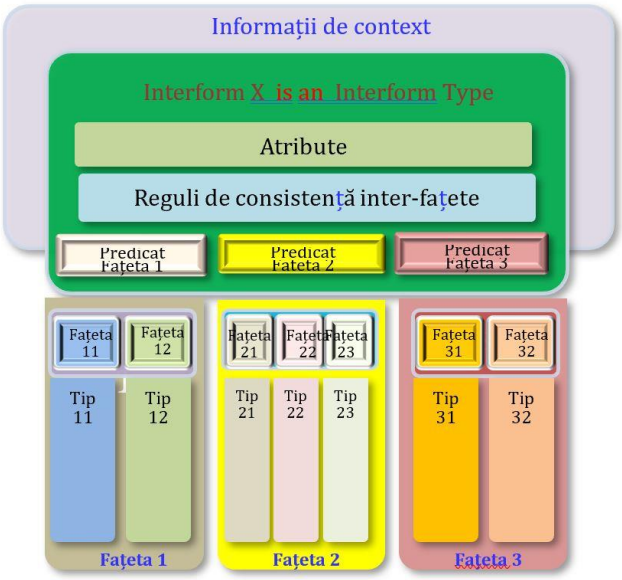


Fig. 1 Reprezentarea grafica a unei interforme

Dacă ar fi să formulez o definiție mai articulata conceptului de interformă: *reprezentare coerentă, completă și structurată a unei entități software, într-un anumit stadiu al proiectării*. Ea adună, într-o formă inteligibilă și reutilizabilă, toate perspectivele relevante ale produsului software – nu doar funcția sau codul, ci și arhitectura, comportamentul, semantica, dinamica, relaționarea contextuală. Nu este o documentație pasivă, nici un model singular, ci un agregat viu de viziuni inginerești.

Fațetele care alcătuiesc o interformă nu sunt simple descrieri din unghiuri diferite de vizualizare, ci *maniere distincte de raționament* asupra aceleiași entități. Ele se pot suprapune parțial, se pot întări reciproc, sau pot necesita clarificări de coerență. Redundanța este nu doar acceptată, ci dorită – o redundanță *controlată*, *semnificativă*, care asigură robustețea proiectului și facilitatea înțelegerii. (Fig. 1)
Interformele, asemenea cadrelor propuse de Minsky, servesc la reprezentarea cunoștințelor prin decompoziție în substructuri mai simple, ele însele, eventual, reprezentabile prin interforme. Însă, spre deosebire de acestea, regulile de formare și modul de utilizare al interformelor răspund unei alte logici: ele sunt concepute ca structuri operaționale, evolutive, profund ancorate în practica proiectării software.
O formulare mai precisă a definiției interformei este următoarea: *interforma este o reprezentare cât mai completă, coerentă și structurată a unei entități software, la un moment dat în cursul procesului său de dezvoltare*. Ea reunește, într-un cadru navigabil și evaluabil, multiple fațete complementare ale entității proiectate.
Interforma nu trebuie confundată nici cu o simplă documentație, nici cu un model singular. Este o *construcție compozită* de viziuni – fiecare reflectând o perspectivă tehnică, funcțională sau semantică – menită să capteze complexitatea reală a obiectului proiectat. Redundanța parțială dintre aceste fațete nu constituie o eroare, ci un principiu: *convergența prin multiplicare de perspective* devine un mijloc de a întări înțelegerea inginerească.

3. Cunoaștere inginerească fragmentară: fațetele interformei
La originea conceptului de interformă stă o observație esențială asupra limitelor cunoașterii inginerești: realitatea unui sistem software complex nu poate fi înțeleasă în totalitatea sa — nici în extensie, nici în profunzime — decât prin descompunere. Această limitare nu este un eșec al minții umane, ci o consecință structurală a modului în care gândim: sistemic. De aceea, în proiectarea software trebuie să găsim mijloace de *a raționaliza ceea ce depășește capacitatea de înțelegere atotcuprinzătoare*.
În tradiția ingineriei, *divide et impera* nu este doar o tehnică de organizare, ci și o strategie cognitivă de stăpânire a complexității. Interforma adaugă însă ceva fundamental acestei decompoziții: *multiplicarea controlată a perspectivelor* asupra aceleiași entități. Nu doar împărțirea în componente funcționale, ci examinarea simultană din unghiuri complementare.
Astfel, interforma nu este doar o colecție de componente, ci un *agregat de fațete*: fiecare fațetă reprezintă o perspectivă distinctă, coerentă și formalizabilă asupra aceleiași entități software. Aceste fațete privesc aspecte diferite ale sistemului de proiectat, cu diferite niveluri de detaliere și/sau de formalizare. Interforma este deci un *fragment de cunoștințe* (knowledge chunk) pe care le avem în legătură cu o entitate software. Agregatul de *fațete* reprezenta perspective multiple, simultane și coerente, asupra aceleiași entități. Fațetele sunt construite pentru a reflecta diferite aspecte „standard”, generice ale unui sistem și corespund unor moduri distincte de raționament ingineresc. Iată pe cele mai importante:

- **Fațetă structurală** – descrie compoziția internă a entității, arhitectura sa, organizarea în module, componente și interfețe;
- **Fațetă funcțională** – vede sistemul ca o cutie neagră: se concentrează pe relația dintre intrări și ieșiri, funcțiile realizate;
- **Fațetă comportamentală** – exprimă modul în care sistemul reacționează în timp: stări, tranziții, protocoale de comunicare;
- **Fațetă dinamică** – introduce contextul: cum răspunde sistemul la evenimente externe, cum evoluează în situații variabile;
- **Fațetă semantică** – conectează entitatea software cu realitatea pe care o modelează sau o servește, interpretarea sa funcțională;
- **Fațetă socială** – reflectă relația dintre software și utilizatorii săi: cine îl folosește, în ce scop, cu ce riscuri sau responsabilități.

Aceste fațete nu sunt complet independente — ele descriu același obiect — dar sunt suficient de ortogonale încât să ne aducă informații complementare. Împreună, formează o viziune poliedrică, coerentă și navigabilă asupra entității proiectate.

Diversitatea fațetelor aceleiași entități provine din punctele de vedere diferite ale diverșilor stakeholder-i asupra produsului software de proiectat. Aceasta diversitate a fost acceptată, mai mult a fost căutată, în faza de colectare si inginerizare a cerințelor, când aceste cerințe au fost confruntate una cu alta pentru a elimina eventualele conflicte. Datorita intereselor (concernelor) distincte ale stakeholder-ilor, în faza de proiectare, aceste cerințe sunt grupate pe categorii funcționale și pot da naștere la fațete diferite pentru aceiași entitate a software-ului.

4. Structura internă a unei interforme: modelul informațional (Fig.2)
Pentru a deveni pe deplin operațională în activitatea de proiectare software, o interformă trebuie să aibă o *structură informațională explicită*, compusă din mai multe componente (numite *slot-uri*). Fiecare slot este o pereche asociată constând dintr-un nume de slot și o listă asociată de liste, fiecare dintre acestea fiind o pereche asociată constând dintr-un nume de fațetă și o listă de valori ale fațetei respective în slotul respectiv. O interformă are mai multe sloturi care definesc identitatea, funcționalitatea, structura , comportamentul, dinamica și coerența unei entități software. Aceste slot-uri nu sunt arbitrare, ci reflectă cerințele inginerești fundamentale ale procesului de proiectare:

a) Atribute de identificare – includ denumirea entității, tipul, taxonomia, versiunea, autorul, proprietarul, data livrării, drepturile de acces și orice altă informație necesară catalogării și controlului configurației produsului software.
b) Context – reprezintă rețeaua de relații ale entității cu alte entități software sau componente ale sistemului. Acest slot ancorează interforma într-un ecosistem conceptual de interforme mai larg.
c) Funcție – definește entitatea ca o mapare între un domeniu de intrare și un interval de valori de ieșire. Este o vedere de cutie neagră, centrată pe scop și comportament general.
d) Interfață de date – descrie semnătura de intrare-ieșire a entității: ce date primește, ce rezultate produce, în ce format și condiții.
e) Fațete – conțin reprezentări diferențiate ale entității în funcție de vederile stakeholder-ilor (utilizatori, dezvoltatori etc.), atât din punct de vedere structural, funcțional, semantic etc., cât și la niveluri diferite de formalizare (textual, sintactic, logic).
f) Motivația proiectării – înregistrează succesiunea deciziilor inginerești care au condus la forma actuală a entității. Este un spațiu al raționamentelor explicite.
g) Reguli de consistență – grupează constrângerile care trebuie respectate de întregul model. Ele pot viza instanțierea slot-urilor cu valori, coerența între fațete, respectarea unor standarde sau validarea unor proprietăți critice.

Slot-urile (a)–(e) sunt folosite pentru definirea entității reprezentate; (f) și (g) guvernează procesul decizional și coerența internă a entității. In acest spațiu al unei interforme, fațetele sunt poziționate în „slot-uri" încărcate de semantică, în sensul că slot-urile sunt tipizate, iar fiecare tip de slot introduce o semantică a slot-ului. Fiecare fațetă are propriile întrebări (predicate) care trebuie verificate la încărcarea unei valori în slot-ul ei. Prin „instanțierea interformei" anumite slot-urile pot căpăta valori corespunzătoare tipurilor de date permise în slot-urile respective. Fațete paralele vorbesc între ele mai ales când provin din aceiași perspectivă sau perspective similare. Atunci ele vorbesc despre aceeași problemă și comparația este posibilă. Fără interformă, riscăm să tratăm aceste fațete ca entități izolate, pierzând tocmai coerența care face ca software-ul să fie robust, extensibil și inteligibil.
Fig. 2 prezintă o interformă generica, gata de a fi instanțiată. Acest model informațional face din interformă un artefact inginerește complet, capabil să susțină verificarea, simularea și, ulterior, implementarea instanțelor de interforme. Odată instanțiată, parțial sau complet, interforma se transformă dintr-o simplă reprezentare descriptivă într-un spațiu activ al gândirii inginerești.

Interformele devin unități de lucru distribuite în echipe mari, care puteau fi gestionate ca elemente (items) de configurație într-o bază de date și puteau susține trasabilitatea decizională și auditul tehnic. În intenția mea, interforma nu era o simplă reprezentare intermediară, ci un obiect conceptual plural, o formă de gândire inginerească, în care toate perspectivele esențiale asupra unei entități software – structurală, comportamentală, funcțională, dinamică și semantică – erau integrate, coerente și navigabile. Într-o epocă în care limbajele de modelare începeau să se diversifice (UML, OMT, SDL), eu propuneam o abordare în care unitatea semantică a entității modelate era păstrată, iar viziunea proiectantului era respectată în toată complexitatea ei.

5. Redundanță controlată și ortogonalitate a perspectivelor.
Interforma nu evită redundanța – dimpotrivă, o folosește constructiv.
Fațetele nu sunt doar vizualizări din puncte de vedere distincte, ele sunt **moduri distincte de a raționa asupra aceleiași realități**, fiecare cu propriile modele, limbaje, reguli și instrumente. Iar provocarea reală este că ele trebuie să coexiste coerent: să nu fie conflictuale pentru a sprijini luarea deciziilor și rafinarea proiectului.
Ortogonalitatea fațetelor apare din necesitatea de a descrie aceeași entitate din unghiuri de observare distincte:

– funcțional (relația input-output),
– lexical și pragmatic (denumiri, simboluri și formate grafice, utilizări personalizate),
– sintactic (structura componentelor interne),
– dinamic (reacția la evenimente externe),
– semantic (semnificația pentru utilizatorii entității),

Aceste descrieri aduc informații distincte și indispensabile pentru înțelegerea și validarea modelului entității: ele sunt tendențial ortogonale**.** Deoarece aparțin aceleiași entități, ele pot fi totuși parțial redundante,. De exemplu, de multe ori sunt utile constrângeri precum:

– structural versus comportamental (cum e alcătuită versus cum se manifestă în timp),
– static versus dinamic (configurație versus evoluție).

Interforma s-a născut din necesitatea de a avea un container conceptual pentru toate aceste fațete, astfel încât să putem lucra simultan cu ele, fără a pierde din vedere unitatea entității proiectate. Interforma nu era nici o diagrama și nici un model fix, ci **un spațiu operațional al proiectării**, unde gândirea inginerească își putea desfășura mișcările fără a fi constrânsă la un singur limbaj sau punct de vedere.
Redundanța între fațete nu este un defect, ci un principiu constructiv: fiecare descriere adaugă claritate, robustețe și verificabilitate. Cheia este ca această redundanță să fie *controlată și convergentă*, nu contradictorie. O fațetă funcțională trebuie să poată fi reconciliată cu cea structurală; o vedere statică, cu comportamentul dinamic.

6. Recursivitatea interformei și fundamentele modularității.
Una dintre trăsăturile definitorii ale interformei este **recursivitatea proiectării**. O interformă nu este o construcție atomică și plată, ci un **sistem de descriere stratificat**, în care fiecare componentă – structurală, comportamentală sau dinamică – dacă are o anumită autonomie structurală și funcțională, poate fi, la rândul ei, o interformă. De aceea, interformele pot fi organizate ierarhic sau compozițional cu relația „*part_of*”, reflectând fidel realitatea în structura modulară a software-ului bine proiectat.
Această abordare recursivă aduce un avantaj crucial: permite controlul granular al proiectului, validarea locală a deciziilor, înțelegerea sistemului prin compunere și, mai ales, reutilizarea structurilor deja descrise. Într-un anumit sens, interforma oferă nu doar o oglindă a produsului software, ci și o hartă a raționamentului care a condus la acel produs.
Această recursivitate derivă din însăși natura **complexității software-ului**: niciun proiect de anvergură nu poate fi înțeles sau construit în întregime deodată. Așa cum înțelepciunea inginerească ne îndeamnă să „împărțim pentru a înțelege”, tot astfel **interforma devine un container de interforme**, un ansamblu de descrieri care reflectă ierarhia conceptuală și funcțională a sistemului.
Acest mecanism se manifestă în două moduri principale:

a) Pe dimensiunea structurală
În plan structural, o interformă care descrie o entitate (sistem, subsistem, modul, clasă de obiecte) poate integra alte interforme aferente subcomponentelor sale. Această relație este analogă cu o compoziție în modelarea software sau cu agregarea de obiecte în limbajele orientate pe obiecte. Totuși, interforma propune o generalizare semantică a acestui mecanism: orice parte a unei entități care are suficientă autonomie semantică (o identitatea) poate deveni sub-interformă, cu propriile fațete și propriul ciclu de dezvoltare.
O interformă care descrie o componentă a sistemului (de exemplu, un subsistem, un modul, o structura complexă de date) **poate include alte interforme** corespunzătoare subcomponentelor acesteia. Dacă am transpune această relație într-un limbaj orientat pe clase, am putea spune că:

- interforma poate **fi descrisă ca o clasă**, cu atribute și relații;
- poate **moșteni** anumite caracteristici de la o interformă-părinte (super-interformă) și poate **agrega subinterforme**, fiecare specializată pe un aspect (de exemplu, comunicare, persistență, logică de business, securitate).

Desigur în această paralelă obiectivele celor doua limbaje sunt diferite: interformele se concentrează pe reprezentarea explicită și intuitivă a cunoștințelor, în timp ce clasele din limbajele orientate pe obiecte se concentrează pe încapsulare și ascunderea informațiilor.
Astfel, **interforma susține modularizarea prin proiectare stratificată**, permițând o diviziune clară a responsabilităților și o izolare a preocupărilor (separation of concerns).
Astfel, interforma oferă un cadru modular natural, în care:

- substructurile funcționale sunt identificate ca unități semantice distincte;
- fiecare dintre ele este descrisă prin propriile sale fațete;
- întreaga construcție menține o coerență globală prin relații între fațete și între interforme.

b) Pe dimensiunea comportamentală
La nivel comportamental, recursivitatea se manifestă prin utilizarea *mașinilor abstracte ierarhice*. Comportamentul global poate fi descompus în sub-comportamente (inițializare, tratare de evenimente, răspunsuri la erori), fiecare descris prin propria sub-mașină, eventual subordonată unui mecanism de control al tranzițiilor. Când descriem comportamentul unui sistem printr-o mașină abstractă, aceasta poate deveni, la rândul ei, **o interformă comportamentală**, alcătuită din:

- sub-mașini abstracte specializate, fiecare descriind un sub-comportament (de exemplu, configurare, inițializare, procesare de evenimente, răspuns la erori);
- mecanisme de control al tranzițiilor între aceste sub-comportamente;
- condiții de activare, sincronizare sau compoziție.

Această **modularizare comportamentală** este echivalentul unei decompoziții funcționale a comportamentului, în care fiecare **unitate logică devine o interformă în sine** – documentată, simulabilă, validabilă.
Prin urmare, **recursivitatea interformei nu este doar o proprietate formală**, ci un **principiu arhitectural** care susține construcția sistematică, raționamentul local, re-utilizarea componentelor și, mai ales, înțelegerea globală prin compunere.
Această abordare permite o modularizare a comportamentului în termeni de agregate dinamice și oferă o cale de rafinare progresivă a logicii de funcționare, până la nivel de cod executabil.
Prin dubla recursivitate – structurală și comportamentală – interforma devine un *cadru arhitectural* în sine, in care diferite soluții arhitecturale se pot susține. Ea nu este doar un model descriptiv, ci o hartă a raționamentelor inginerești care susțin proiectarea, integrarea și evoluția unui sistem complex. Ea permite construirea sistematică, raționamentul local și înțelegerea globală prin compunere.

c) Recursivitate semantică și trasabilitate
Interforma permite nu doar includerea altor interforme, ci și urmărirea istoricului propriei evoluții. Aceeași interformă poate trece prin mai multe versiuni — schiță, prototip, versiune finală — fiecare reprezentând o etapă în rafinarea conceptuală. Relația „*este_o_versiune_a*” face posibilă urmărirea deciziilor inginerești, justificarea lor și eventuala reversibilitate.

Această recursivitate semantică este esențială pentru menținerea coerenței într-un proces evolutiv și adaptiv. Prin integrarea ei într-o bază de date de interforme — analogă unei baze de management al configurației —, întregul proces de proiectare devine trasabil, evaluabil și reconfigurabil.

În concluzie, interforma nu este un model fix, nici un format de documentație. Este un principiu ingineresc al proiectării în condiții de complexitate: a înțelege o entitate în mod plural, stratificat și rațional, pentru a o putea proiecta, verifica și menține.

Această polifonie controlată permite o rafinare sistematică: proiectarea devine astfel o operație de creare, completare, rafinare și configurare de interforme, până când ele devin implementabile, cu alte cuvinte până când ele ajung să fie descrise complet în cod. Interforma unei entități reale își începe viața ca o schiță sau un cod parțial intercalat cu pseudo-cod încă ne-codificat. Pe măsură ce cunoștințele proiectantului despre entitatea de proiectat evoluează, interforma evoluează și ea, căpătând noi fațete sau detaliind fațetele existente. Se creează astfel o secvență de interforme legate prin relația „*este_o_versiune_a*” care modelează aspectul diacronic al proiectării. Astfel, interforma devine o infrastructură cognitivă, colaborativă, care sprijină decizia inginerească, comunicarea între echipe și trasabilitatea deciziilor.

Interforma introduce astfel un spațiu în care coexistă toate fațetele relevante ale unei entități software. Această pluralitate controlată permite proiectantului să navigheze între diversele perspective, să înțeleagă complexitatea sistemului fără a-l simplifica brutal, și să decidă asupra transformărilor într-un mod rațional și justificabil. Interforma e un obiect de lucru inginerește riguros, dar deschis – un stadiu în care elementele ei componente pot fi evaluate, comparate, rafinate, fără a fi încă „înghețate” în cod.

7. Interforma ca unitate inginerească de lucru și colaborare
Dincolo de valoarea sa conceptuală, interforma a fost gândită ca o unitate inginerească operațională – un obiect de lucru clar delimitat, care poate fi integrat în fluxurile de proiectare colaborativă. În proiectele de mari dimensiuni, interformele devin unități naturale de organizare a muncii în echipă. Fiecare interformă corespunde unui obiect de interes ingineresc — un modul, un subsistem, un serviciu software — care poate fi proiectat, validat și întreținut în mod autonom. Ea poate fi atribuită unei echipe distincte și tratată ca un element de configurație gestionat într-o bază de date.

Această abordare permite:

- organizarea sistemului sub forma unui arbore de interforme, în care fiecare ramură corespunde unei subsarcini clar definite;
- lucrul paralel pe componente fără pierderea coerenței semantice;
- trasabilitatea deciziilor tehnice, prin versiuni succesive de interformă și justificarea alegerilor inginerești;
- posibilitatea auditului arhitectural și a mentenanței pe termen lung.

Această abordare susține programarea în echipă pe subarbori semantici. Se pot aloca interforme diferitelor echipe în funcție de specializare, și se pot integra ulterior rezultatele prin mecanisme standardizate de compunere a fațetelor.

Interforma nu este un model intermediar pasiv, ci un artefact viu, iterativ, care poate fi evaluat, rafinat și reutilizat. Ea susține nu doar descrierea unei entități software, ci raționamentul inginerească colectiv care a condus la forma sa actuală. Fără o astfel de structură, riscul este fragmentarea reprezentărilor – tratarea vederilor distincte ca entități izolate – și pierderea unității semantice a sistemului.

În acest sens, interformele devin unități de dezvoltare semantică, depășind rolul de simplă documentație. Ele sunt instrumente active în comunicare, sincronizare și validare a construcției software.

8. Limbajul interformei și trasabilitatea deciziilor
Pentru ca interforma să-și îndeplinească rolul de unitate semnificativă de gândire inginerească și de colaborare practică, ea are nevoie de un limbaj comun, accesibil tuturor actorilor implicați în procesul de dezvoltare software. Nu este vorba doar de un limbaj formal, ci de un cadru convențional de exprimare — suficient de structurat pentru rigoare, dar suficient de flexibil pentru a reflecta nuanțele decizionale și a fi accesibil tuturor membrilor unei echipe implicate în dezvoltarea software-ului.

În acest scop, am propus un pseudocod extins, inspirat din structurile de control al logicii execuției ale limbajelor de programare, dar adaptat pentru a exprima multiplele fațete ale unei entități software. O ramură de tip „CASE” din această descriere nu exprimă un caz de selecție, ci o fațetă (proiecție) alternativă asupra aceluiași obiect. Redundanța controlată dintre aceste fațete permite gândirea multiplă, reconcilierea punctelor de vedere și convergența semantică.

Una dintre marile promisiuni ale interformei este capacitatea sa de a păstra memoria deciziilor inginerești. Fiecare versiune a unei interforme reflectă o etapă a raționamentului proiectantului. Prin marcarea acestor versiuni și legarea lor prin relații de tip „este_o_versiune_a”, putem construi o rețeaua de interforme a evoluției proiectului:

În proiectele de mari dimensiuni, interformele pot fi organizate ierarhic, fiecare fiind încredințată unei echipe specifice. Structura rezultată este un arbore conceptual de interforme, în care fiecare echipă lucrează local, dar integrat semantic în arhitectura globală. Această organizare permite paralelizarea proiectării, evitarea coliziunilor de viziune și reutilizarea inteligentă a componentelor.

Interforma nu este doar un artefact static. Ea este și un nod în rețeaua deciziilor de programare luate de-a lungul dezvoltării unui produs software. Fiecare alegere – structurală, algoritmică, tehnologică – este reflectată (sau ar trebui să fie reflectată) în forma și evoluția interformei.

Interformele pot fi gestionate în sisteme de management al configurației, ca itemi decizionali trasabili. Fiecare versiune a unei interforme înregistrează:

- modificările efectuate;
- justificările tehnice;
- sursele deciziilor;
- relațiile cu alte interforme;
- fiecare decizie (structurală, algoritmică, funcțională) este înregistrată ca transformare a unei interforme;
- motivele deciziilor, constrângerile și implicațiile pot fi documentate contextual.

Traseul de dezvoltare devine astfel auditabil, justificabil și reversibil. Un astfel de mecanism de trasabilitate este esențial pentru proiectele critice sau de lungă durată, unde înțelegerea motivațiilor trecutului este la fel de importantă ca implementarea prezentului. În cadrul unei infrastructuri inginerești moderne, această rețea de interforme și transformări poate fi integrată într-o bază de date inginerească, analogă sistemelor de gestiune a configurațiilor (Configuration Management).

Această trasabilitate susține auditul ingineresc, evoluția controlată a arhitecturii și, mai ales, continuitatea proiectării în fața schimbărilor inevitabile de echipă sau cerințe.

În concluzie, interforma devine un pivot al comunicării inginerești, un liant între gândirea individuală și construcția colectivă, între libertatea decizională și disciplina dezvoltării. Ea este nu doar o unealtă de proiectare, ci și o infrastructură cognitivă pentru colaborare.

Prin urmare, interforma devine un adevărat limbaj ingineresc de echipă — nu doar un instrument de descriere, ci un spațiu în care cunoașterea este exprimată, verificată, partajată și perpetuată.

9. Echilibrul între formalism și expresivitate: un spațiu de gândire inginerească
Interforma a fost concepută ca un cadru inginerește plural, în care diverse moduri de raționament pot coexista și interacționa coerent. Fațetele pe care le cuprinde nu sunt doar vizualizări, ci instrumente cognitive – fiecare cu propriile reguli, limbaje și nivele de formalizare. Ele permit abordarea aceleiași entități din perspective funcționale, structurale, semantice sau dinamice, fără a impune o unificare prematură.

În mod ideal, aceste fațete ar trebui exprimate în limbaje formale – logică, algebră, grafuri, modele din teoria automatelor – care oferă rigoare, simulabilitate și validare automată. Interforma devine astfel un cadru operațional, nu doar conceptual: suportă transformări, analize, verificări.

Totuși, nu toate aspectele sistemului pot fi capturate riguros. De aceea, interforma permite și exprimări în limbaj natural – atâta timp cât sunt clare, bine definite și corelate cu restul structurii. Această deschidere face din interformă un mediu accesibil atât echipelor inginerești, cât și celor interdisciplinare, în care comunicarea, nu doar formalizarea, este esențială.

În fond, interforma este spațiul în care se articulează gândirea inginerească – un teritoriu în care raționamentele pot fi exprimate, comparate și rafinate, indiferent dacă pornesc din intuiție sau din calcul.

10. Interforma ca punte între gândirea umană și exprimarea inginerească
Interforma este, în esență, o unealtă de gândire inginerească care oferă vizibilitate, modularitate și reversibilitate în procesele decizionale din ciclul de viață al dezvoltatorului de software. În accepțiunea pe care am conturat-o în carte și în practica sistemului INTERFORM, ea nu era doar o descriere a produsului software, ci o expresie a procesului cognitiv prin care proiectantul transformă reprezentările sale mentale în construcții verificabile și partajabile. În acest sens, ea constituie inelul de legătură între ceea ce știm și ceea ce putem comunica, între ceea ce intuim și ceea ce putem explica despre o construcție.

La originea unei interforme stă gândirea unui specialist în proiectare, adică un ansamblu de reprezentări mentale - modele cognitive semi-permanente de structurare și funcționare a sistemelor - formate prin învățare, experiență și analogii. Aceste reprezentări nu sunt nici unitare și nici uniforme: ele vin în „straturi de înțelegere" sau „fațete", fiecare reflectând un unghi de abordare – comportamental, structural, funcțional, dinamic. Interforma preia aceste fațete și le agregă într-o formă inteligibilă pentru ceilalți ingineri, dar fără a pierde bogăția viziunii inițiale.

Este, așadar, o tranziție de la raționamentul individual la colaborarea inginerească, un mediu de exprimare suficient de flexibil pentru a păstra intuiția, dar suficient de structurat pentru a permite simularea, evaluarea și, în cele din urmă, implementarea. Pentru exprimarea interformei, am propus un pseudocod extins, inspirat de structurile logice din limbajele de programare, dar capabil să descrie multiplele fațete.

Această tranziție are o dimensiune evolutivă:

1. la început, interforma poate fi schițată informal, într-o propoziție de limbaj natural sau într-o schemă intuitivă;
2. pe măsură ce proiectul avansează și cunoașterea viitorului produs se îmbogățește cu noi detalii și aspecte, fațetele informale vor fi înlocuite cu descrieri formale și noi fațete se vor adaugă, descrise prin diagrame semi-formale (de exemplu, diagrame de clase, de activitate, de stare) sau expresii logico-matematice. Noua interformă devine o versiune mai recenta a vechii interforme;
3. în final, interforma va ajunge să includă componente implementabile: cod executabil, modele testabile, configurații simulabile.

Această evoluție a interformei reflectă dinamica gândirii inginerești – de la înțelegerea vagă, globală, la articularea locală, până la executabilitate tehnologică. Este un proces iterativ de rafinare, în care fiecare adăugare este un pas spre clarificare, iar fiecare transformare este o formă de verificare a deciziilor anterioare.

Astfel, interforma devine nu doar un container de descrieri, ci o infrastructură cognitivă și colaborativă, care susține:

- continuitatea între etapele designului,
- comunicarea între membri ai echipei,
- și fundamentarea deciziilor tehnice.

11. Limitele UML și valoarea anticipativă a interformei

Odată cu primele versiuni ale UML, la mijlocul anilor ’90, am observat o carență conceptuală majoră: deși limbajul permitea modelarea din perspective diferite — structurală (diagrame de clasă), comportamentală (diagrame de stare) și interacțională (diagrame de secvență) — nu exista niciun mecanism formal de a exprima că toate aceste reprezentări aparțin aceleiași entități software. Fiecare diagramă funcționa ca o insulă, fără un liant semantic între ele. Nu există relații de tipul: „*face parte din aceeași interformă*”, „*este o proiecție a aceluiași obiect*” sau „*reprezintă o perspectivă complementară asupra aceluiași nucleu semantic*”.

În acea perioadă,-am adresat un mesaj lui James Rumbaugh, unul dintre creatorii UML, sugerând introducerea unei relații explicite între fețetele parțial suprapuse ale aceleiași entități. I-am dat ca exemplu și ideea de cadru propusă de Minsky, argumentând că o relație formală între vederi ar ajuta proiectantul să navigheze coerent complexitatea conceptuală. Răspunsul său a fost pozitiv în principiu, dar a precizat că astfel de modificări trebuiau aprobate de comitetul OMG — ceea ce, evident, nu s-a întâmplat.

Problema nu era de detaliu, ci structurală: pentru ca fețetele să fie integrate, era nevoie de o semantică comună, iar aceasta nu putea fi dedusă automat. Ar fi fost necesar un mecanism de coerență parțială între reprezentări — ceva ce UML nu oferea și încă nu oferă nici azi. Într-adevăr, această lacună persistă și în UML actual, în ciuda progreselor în limbajele orientate pe views, precum SysML sau ArchiMate, unde problema consistenței între vederi multiple (multi-view consistency) este abia schițată..

Modelul interformei, prin natura sa compozită, răspundea tocmai acestei nevoi: presupunea, de la început, că o entitate software este descrisă prin proiecții coerente, corelate și transformabile, și că aceste proiecții pot fi evaluate împreună, nu izolat. Relațiile dintre fațete sunt reversibile și guvernate de reguli explicite — nu doar comentarii sau constrângeri opționale (precum OCL în UML), ci mecanisme inginerești de consistență. Era, poate, prea devreme pentru va modelul să fie recunoscut, dar rămâne, din perspectiva mea, o abordare încă valabilă – dacă nu chiar necesară – în contextul actual al proiectării sistemelor complexe.

Tabelul 1 face o comparație intre Interform și instrumente conceptuale folosite în proiectarea software introduse ulterior: UML 1.0 (1997) si MDA (Model-Driven Architecture, 2001).

Tabel 1

Aspect	**Interform (Șerbanati)**	**UML**	**MDA (Model-Driven Architecture)**
Tip de reprezentare	Clase multi-fațetate, integrative	Multi-diagramă (clasă, secvență, stare etc.)	Separare între modele CIM–PIM–PSM
Unitate modelată	Entitate conceptuală în construcție	Entități modelate izolat pe diverse dimensiuni	Modele la diferite niveluri de abstractizare
Relația între fațete	Explicită: structurală și ierarhică	Implicită; fațetele (diagrama de clasă, stare etc.) nu sunt legate formal	Separare în niveluri, dar fără integrare fațetată
Scop principal	Sprijin decizional în faza de proiectare	Comunicare vizuală, specificare	Transformare între niveluri de abstractizare
Modelare semantică	Puternică, contextualizată prin AM personalizate	Limitată la comentarii și constrângeri OCL	Depinde de metamodel, adesea superficială
Reversibilitate / navigabilitate	Da, între fațete; proiectantul poate revizita deciziile	Nu direct; diagrama de clasă nu „știe” de cea de stare	Parțial, dacă suportat de platformă
Formalizare internă	Mașină abstractă (AM) a proiectării guvernează tranzițiile între interform-e	OCL este opțional; metodele lipsesc de mecanisme de execuție	Formalism teoretic, dar rar implementat complet

12. Reflecție finală: o idee prematură, o moștenire actuală

Privind înapoi, trebuie să recunosc că *interforma*, deși una dintre cele mai consistente contribuții pe care le-am formulat în ingineria software, nu a avut istorie.. Nu a generat dezbateri, nu a fost citată în articole de referință, nu a influențat metodele moderne. A rămas o idee validată în laborator, dar necontinuată în practică, un concept viu în conștiința autorului său, dar tăcut în istoria disciplinei.
Sistemul INTERFORM a fost implementat, validat tehnic și utilizat într-un context inginerește autentic. Dar nu a fost continuat. A dispărut în lipsa unei echipe, a unei instituții care să-i asigure dezvoltarea, vizibilitatea și adaptarea în timp.
Când am plecat din țară, sistemul avea nevoie de stabilitate și continuitate. Nu le-a avut. Am publicat cartea *Integrating Tools for Software Development* la Prentice Hall, cu speranța că publicarea internațională va duce ideea mai departe. Nu s-a întâmplat. Cartea a fost achiziționată de biblioteci universitare, dar nu a fost promovată, citată, sau discutată.
Am înțeles atunci o lecție care nu ține de software, ci de *ecosistemul recunoașterii*: în cultura occidentală, autorul trebuie să-și facă singur publicitate. Fără recenzii, conferințe, citări strategice, nici cea mai solidă idee nu ajunge să fie luată în seamă. Eu am venit dintr-o cultură academică diferită, unde valoarea trebuia demonstrată în conținut, nu în marketing. Și am plătit prețul pentru această discreție.
Totuși, nu regret. Pentru că ideea rămâne. Interforma este, în mod paradoxal, poate chiar *mai actuală astăzi* decât atunci când am formulat-o. Într-o lume a echipelor distribuite, a cerințelor divergente, a complexității în creștere și a nevoii de trasabilitate decizională, interforma oferă exact ceea ce lipsește: un cadru plural, coerent, evolutiv, în care gândirea inginerească să se poată desfășura nestingherită de formalisme inutile, dar nici lăsată la voia improvizației.
Poate că nu i-am găsit numele cel mai potrivit. Poate că nu am creat în jurul ei o metodologie suficient de extinsă. Interforma ar putea fi **redescoperită astăzi** în contextul:

- modelării colaborative distribuite,
- toolchain-urilor integrate în DevOps,
- evoluției instrumentelor de *model-driven engineering*,
- nevoii de trasabilitate completă în proiecte critice (ex. software în sănătate, sisteme embedded, infrastructuri critice), sau
- chiar în contextul **inteligenței artificiale aplicate în design**, ca structură de sprijin pentru raționamentele unei mașini proiectante.

Conceptual, interforma anticipează idei precum *digital twin*-ul — dar nu în sensul unei replici digitale a unei entități fizice, ci ca o *proiecție fidelă a reprezentării mentale* a proiectantului asupra produsului software aflat în formare: *interforma* ca **agregat digital** este o replică semi-**formală** a reprezentării **mentale a proiectantului asupra unui produs software** aflat în construcție. Iar această **tranziție între reprezentarea mentală și descrierea formală** este, după mine, locul cel mai fragil, dar și cel mai fecund al procesului de proiectare.
Dacă un lucru regret, este că nu am regăsit acest concept reformulat în limbajul generațiilor de după mine.
Poate că nu i-am găsit numele cel mai potrivit. Poate că nu am știut să-i creez un ecosistem metodologic mai larg. Dar știu că, undeva, într-o arhivă a ideilor care n-au fost încă redeschise, interforma așteaptă încă să fie redescoperită.

Interforma azi: între uitare și redescoperire

Din păcate, interforma nu a avut istorie. Sistemul INTERFORM a fost validat inginerește, dar nu a fost continuat. Cartea în care am publicat ideea nu a fost suficient promovată. Lipsa vizibilității – și a unei echipe care să continue ideea – a făcut ca interforma să rămână o contribuție tăcută.

În ciuda faptului că nu a fost receptată în epoca în care am formulat-o, ideea interformei rămâne, în ochii mei, actuală și profund necesară. Interforma este un model încă valid, poate chiar mai necesar astăzi. Lumea software-ului a evoluat spectaculos în ultimele decenii, dar provocările esențiale – complexitatea sistemelor, necesitatea înțelegerii multidimensionale, comunicarea între echipe eterogene – sunt mai vii ca oricând.

Ceea ce propuneam prin interformă – o reprezentare stratificată, multiplă, cu redundanță semantică controlată a entităților software – răspunde exact acestor provocări. Într-un ecosistem tehnologic în care arhitecții, dezvoltatorii, testerii, analiștii business și stakeholderii unui produs software operează cu viziuni diferite asupra aceluiași produs, interforma poate deveni puntea comună, „lingua franca” a colaborării.

Conceptual, interforma se înscrie în linia ideilor prolifice din ingineria sistemelor: recunoașterea imposibilității de a înțelege o entitate complexă într-o singură reprezentare, nevoia de a descrie acea entitate din perspective

diferite, eventual complementare – structurală, comportamentală, funcțională, dinamică sau semantică. O astfel de descriere nu este doar documentație, ci un instrument activ de proiectare, un „artefact viu”, menit să ghideze construcția sistemului. Seria acestor artefacte, de la concepție până la produsul validat, reprezintă vederea diacronica asupra instanței procesului software care a realizat produsul.

Interforma oferă o soluție inginerească la problema fragmentării reprezentărilor – încă nerezolvată în modelele standard folosite în proiectarea software. Faptul că în limbaje moderne utilizate în modelarea conceptuală precum UML, SysML sau chiar în metodologiile agile, nu există o relație între fațetele aceluiași obiect – o relație de suprapunere parțială, nu de compunere sau moștenire – confirmă, retrospectiv, lipsa conceptuală pe care interforma o anticipa. Relațiile UML sunt relații între obiecte distincte. Oferă perspective separate (clasă, stare, secvență de evenimente), dar nu le corelează explicit. Interforma, în schimb, adresează problema consistentei reprezentărilor conceptual distincte ale aceleiași entități (multi-view consistency) – o problemă doar recent abordată conceptual azi de extensii precum SysML sau ArchiMate.

Mașina abstractă

Conceptul de mașină abstractă (Abstract Machine) a apărut in ultimii ani ’60 și desemnează o abstractizare, un model operațional al funcționării unui sistem informatic. Conceptul are un strămoș matematic: automatul cu un număr finit de stări, un obiect formal similar cu o funcție peste șiruri de simboluri care primește intrări și produce ieșiri pe baza unor reguli predefinite. Mașinile abstracte sunt „mașini” deoarece permit executarea de operații dintr-un set de operații permise (instrucțiuni) utilizate pentru a transforma pas cu pas (instrucțiune cu instrucțiune) date de intrare în date de ieșire și sunt „abstracte” deoarece ignoră multe aspecte ale mașinilor (hardware) reale.

In plus, mașina abstractă devine, in concepția mea, un model logic care permite descrierea desfășurării controlate a însuși procesului de proiectare.

În esență, MA este o construcție conceptuală capabilă să execute programe formate din operații „inventate” ad hoc de proiectant, utilizate să simuleze cu ele comportamente dorite de acesta. Fațetele comportamentale ale unei interforme pot fi modelate cu ajutorul mașinilor abstracte. Modelul mașinii abstracte permite formalizarea unui flux decizional, în care fiecare pas – fie el o modificare structurală, o extensie funcțională sau o redefinire semantică – este înregistrat, analizat și, la nevoie, reversibil.

Am introdus conceptul de mașină abstractă nu ca o simplă metaforă pentru un program de execuție, ci ca o mașină de proiectare: un operator logic care guvernează pașii prin care un sistem software este construit, rafinat, evaluat și validat.

Conceptul de mașină abstractă poate fi folosit pentru a modela activități în procesul software al proiectelor: să evalueze coerența interform-elor și să ghideze tranzițiile între stările unui proiect. In acest caz, MA poate fi privită ca o extensie inginerească a noțiunii de design agent, capabil nu doar să genereze variante, ci să le evalueze în funcție de obiectiv, constrângeri și progresul acumulat. În acest fel, proiectarea nu mai este doar o navigare intuitivă prin spațiul soluțiilor, ci o activitate inginerească, în care istoria deciziilor și traseul lor logic sunt păstrate și reutilizabile.

Prin integrarea interformei cu mașina abstractă, am putut modela proiectarea ca pe o rețea de procese colaborative, fiecare proces fiind activ într-un plan diferit (semantic, structural, comportamental), dar toate armonizate de același cadru metodologic. Rezultatul este o arhitectură inginerească a gândirii proiectante – o disciplină a raționamentului aplicat în contextul dezvoltării software-ului complex.

In fine, ierarhiile de mașini abstracte pot fi folosite pentru reprezentarea de straturi computaționale bazate pe relația de rafinare/abstractizare, în care fiecare mașină dintr-un nivel folosește funcționalitatea nivelului imediat inferior și adaugă funcționalități suplimentare proprii pentru a satisface nivelul imediat superior.

Dacă interforma este instrumentul de descriere plurală a unei entități software, mașina abstractă (Abstract Machine, AM) este instrumentul de control al procesului de construcție. Ea reprezintă modul în care gândirea inginerească poate fi modelată operațional, printr-un set de stări, tranziții și mecanisme de decizie care structurează evoluția interformelor în timp.

În literatura clasică, mașina abstractă desemnează un model formal al unui calculator care mapează datele de intrare în rezultate, printr-o succesiune de operații conforme cu o descriere prestabilită. Aceste modele sunt esențiale în teoria limbajelor formale, în semantica operațională și în definirea compilatoarelor.

În abordarea mea, am extins această noțiune pentru a descrie comportamentul unui proiectant în raport cu procesul de dezvoltare:

- AM devine un agent de decizie capabil să selecteze, să rafineze, să valideze interforme;
- fiecare stare a AM corespunde unei etape inginerești (de exemplu, explorare, alternativă, selecție, rafinare, validare);
- tranzițiile dintre stări sunt declanșate de condiții verificabile sau de inputuri externe (de la stakeholderi, de la prototipuri, de la simulări).

În acest sens, mașina abstractă devine o infrastructură logică pentru desfășurarea proiectării într-un spațiu controlat, reproductibil și inteligibil.

b) AM ca motor de proces decizional

Utilizată împreună cu interforma, mașina abstractă permite modelarea:

- fluxului decizional care guvernează tranzițiile între versiuni ale unei interforme;
- raționamentului inginerească ce conduce de la o soluție parțială la o construcție completă;
- stărilor de validare (prin simulare, testare conceptuală, comparare între fațete);
- controlului calității în timpul proiectării.

Mașina abstractă nu este un mecanism închis — ea este extensibilă. Proiectantul definește propriile operații aplicabile asupra interformei (de exemplu, „adaugă o componentă", „rafinează o fațetă", „combină două comportamente") și stabilește regulile de tranziție. În acest mod, AM poate fi văzută ca un agent semi-automat de proiectare, un precursor al instrumentelor moderne de Design Automation și, potențial, al unei inteligențe artificiale inginerești.

c) Componibilitate și ierarhizare

În proiectele de mari dimensiuni, nu o singură mașină abstractă guvernează întregul sistem, ci un ansamblu ierarhic de mașini abstracte:

unele operează la nivel înalt, coordonând tranziții între interforme de subsistem;

altele operează local, guvernând rafinări fine asupra unei fațete sau componente;

între mașini se stabilesc relații de invocare, rafinare, substituție sau paralelism.

Această ierarhie de AM reflectă, în fond, stratificarea arhitecturală a unui proiect complex. În același timp, ea permite distribuirea controlului către diferite roluri inginerești: arhitectul de sistem, inginerul de detaliu, analistul de cerințe etc.

d) O punte către formalizarea procesului inginerească

Rolul mașinii abstracte este acela de a transforma un proces adesea informal — proiectarea — într-un spațiu rațional de explorare și validare. Ea oferă:

- memorie operațională a pașilor parcurși;
- un cadru pentru evaluarea variantelor;
- un mecanism pentru simularea scenariilor de proiectare.

Această orientare spre formalizare nu elimină creativitatea, ci o încadrează într-o disciplină inginerească. Proiectarea devine astfel o activitate cu proprietăți inginerești: predictibilitate, reproductibilitate, transparență decizională.

O arhitectură inginerească pentru complexitatea software

Interforma și Mașina Abstractă nu sunt doar două concepte izolate – ele formează împreună un cadru inginerească coerent pentru modelarea, controlul și comunicarea în procesul de proiectare a produsului software. Relația lor este una de complementaritate profundă: dacă interforma este obiectul gândirii inginerești, mașina abstractă este mecanismul acestei gândiri – ceea ce face posibilă transformarea viziunii în construcție.

Această arhitectură duală poate fi descrisă în termeni simpli:

- Interforma organizează ce știm despre o entitate software, prin fațete multiple, formalizate și coerente;
- Mașina abstractă controlează ce facem cu această cunoaștere, cum o rafinăm, cum o testăm, cum decidem și cum comunicăm.

În esență, interforma este modelul, iar mașina abstractă este procesul care operează pe model.

a) Coevoluția între interformă și AM

Cele două entități evoluează împreună:

- nouă fațetă adăugată unei interforme presupune extinderea comportamentului AM care o gestionează;
- decizie inginerească luată de AM (ex. alegerea unei arhitecturi) se reflectă într-o modificare a unei fațete;
- validarea unei versiuni a interformei este o tranziție într-o stare terminală a unei mașini abstracte de proiectare.

Această coevoluție garantează trasabilitate, consistență și transparență: orice modificare are un motiv, orice decizie are un loc în istoria proiectului.

b) O infrastructură inginerească pentru "programarea în mare"

Modelul dual interform–AM a fost conceput în mod explicit pentru a răspunde provocărilor software-ului complex, acolo unde:

- produsul este rezultatul colaborării între echipe numeroase și distribuite;
- cerințele sunt dinamice, contradictorii, emergente;
- livrarea se face în etape, cu verificări intermediare și adaptări frecvente;
- mentenanța și extensibilitatea sunt esențiale pentru succesul pe termen lung.

În acest context, interforma oferă un mecanism de reprezentare plurală și structurată a produsului, iar mașina abstractă furnizează un mecanism de guvernanță operațională a construcției. Împreună, ele permit o proiectare distribuită dar coerentă, formalizabilă dar adaptabilă, riguroasă dar deschisă dialogului.

c) Precursorul unei epistemologii inginerești

Dincolo de aplicabilitatea lor practică, aceste două concepte propun și o filozofie inginerească: ideea că înțelegerea profundă precede implementarea eficientă. Într-un moment istoric în care software-ul a devenit infrastructură critică pentru societate, o astfel de filozofie este nu doar utilă, ci necesară.

Interforma ne obligă să ne gândim în termeni de concepte clar exprimate, orientate către multi-perspectivă, înainte de a desena diagrame sau de a scrie cod. Mașina abstractă ne invită să raționăm procedural, să articulăm etapele prin care trecem, să păstrăm logica internă a parcursului proiectant.

Această epistemologie nu se bazează pe intuiție, ci pe construirea deliberată de reprezentări și mecanisme care fac raționamentul inginerească vizibil, controlabil și transferabil.

Concluzie – O moștenire neexplorată, un orizont redeschis

Privind în urmă la interforma și mașina abstractă, nu pot să nu resimt o dublă realitate: pe de o parte, aceste concepte au fost formulate cu o maturitate inginerească timpurie, în avans față de nevoile și capacitățile comunității profesionale din vremea lor; pe de altă parte, ele nu au fost receptate, citate sau continuate,

rămânând izolate în paginile unei cărți prea puțin cunoscute și într-un sistem de proiectare – INTERFORM – care nu a fost adoptat la scară largă.

Și totuși, această izolare nu le-a anulat valabilitatea. Dimpotrivă: odată cu trecerea timpului și creșterea complexității software-ului contemporan, nevoia de structuri conceptuale clare, formale, integrative și navigabile este mai acută ca oricând. Marile ecosisteme software din ziua de azi – sisteme de sănătate digitală, infrastructuri de mobilitate autonomă, platforme integrate de date și servicii – sunt atât de vaste și dinamice încât nu pot fi gestionate doar prin instrumente fragmentare și modele vizuale dispersate.

Aici, interforma – în calitate de agregator semantic și formal al tuturor fațetelor unui produs software – își regăsește relevanța. Iar mașina abstractă – ca mecanism de raționament procedural, validare și guvernanță a deciziilor – poate deveni parte din infrastructura logică a unei noi generații de sisteme inteligente de asistență în proiectare.

Mai mult, interforma prefigurează o cale de legătură între modelarea inginerească tradițională și viitoarele forme de design colaborativ susținut de inteligență artificială. Un digital twin de proiectare, așa cum este azi imaginat în industrie, ar putea fi văzut ca o interformă vie: capabilă să evolueze, să memoreze deciziile, să genereze versiuni alternative și să comunice între roluri și straturi organizaționale.

Această posibilă redescoperire nu ar trebui însă să vină fără rigoare. Ea cere un efort de reinterpretare, formalizare și implementare a ideilor într-un context nou, dar păstrând esența lor inginerească: aceea de a structura complexitatea, nu de a o ascunde; de a clarifica procesul decizional, nu de a-l înlocui cu opacitatea algoritmică.

Închei acest capitol nu cu un epilog nostalgic, ci cu o chemare către o generație viitoare de arhitecți software: să redescopere aceste instrumente nu ca relicve, ci ca repere posibile într-o reconstrucție conceptuală a designului inginerească. Poate că ideile de aici au fost prea devreme. Poate că nu au avut ocazia de a intra într-o tradiție. Dar în ele trăiește o ambiție care rămâne vie: aceea de a transforma proiectarea software-ului dintr-un act artizanal, fragmentar și obscur, într-o disciplină vizibilă, formală, umană și reproductibilă.

Reflecție finală

Acest capitol este poate cel mai tehnic din întreaga carte. Nu doar pentru că exprimă idei de autor, ci pentru că încearcă să surprindă ceea ce nu poate fi redus la un formalism: arta invizibilă de a face alegeri bune în condiții incerte. Designul, așa cum îl văd eu, este simultan o disciplină și o artă. O căutare de echilibru între abstracție și detaliu. Între control și deschidere. Între știință și creație.

Poate că unii cititori ai cărții nu au sesizat toate aceste nuanțe. Dar eu le regăsesc astăzi ca pe un cod interior al modului în care am învățat să gândesc software-ul: cu răbdare, cu claritate, cu încredere în formele intermediare ale ideilor încă neimplementate.

Bibliografie

[1] L.D. Șerbanati, Integrating Tools for Software Development, Prentice Hall, 1992.
[2] L.D. Șerbanati, “Towards a Model of Design Methods for Program Products”, 1983.
[3] L.D. Șerbanati, “INTERFORM: A CAD System for Program Development”, Monterey Conference, 1987.
[4] Marvin Minsky, “A Framework for Representing Knowledge”, 1974.
[5] James Rumbaugh et al., UML Specification, OMG.
[6] L.D. Șerbanati, “Către o proiectare dirijată de concern-e”, în A. Bazac et al. (coord.), Logica și provocările sociale, Polipress, București, 2008.