Maşinile X - ca metodă de specificaţie a sistemelor software

Conf.dr. Florentin IPATE
Universitatea Româno-Americană, Bucureşti

Acestă lucrare introduce maşinile X ca metodă de specificaţie şi explică avantajele şi necesitatea acestora în industria de software actuală.
Cuvinte cheie: *maşini X, specificaţii pentru sisteme software, limbaj de specificaţie.*

1. Introducere

Recent, majoritatea cercetărilor din domeniul specificaţiei de sisteme software a fost orientată spre folosirea modelelor de tipuri de date, fie funcţionale sau relaţionale (ca Z, VDM) sau modele axiomatice, ca OBJ. Identificarea de tipuri de date şi definirea algoritmilor ca funcţii sau relaţii pe entităţile matematice a dus la progrese semnificative în designul software. Totuşi, o problemă esenţială cu aceste tipuri de specificaţie abstractă este dificultatea de a exprima dinamica sistemului la un nivel care să corespundă cu intuiţia noastră despre funcţionarea lui. De exemplu, este foarte dificil să reprezentăm structura de meniuri a unei intefeţe folosind o metodă de tipul limbajelor Z sau VDM.
În general, aceste metode pot descrie tot ceea ce face sistemul însă nu şi cum face. Dacă sistemul trebuie să execute mai multe operaţii elementare pentru a se obţine configuraţia finală, este în general foarte dificil de descris evoluţia sistemului care a dus la starea lui finală. Din punct de vedere teoretic, se poate argumenta că în faza de specificaţie tot ce trebuie să ştim este rezultatul final şi că algoritmul folosit este lăsat pentru faza de implementare. Aceasta este în mod evident adevărat dacă funcţionalitatea cerută este foarte simplă (de exemplu, pentru un program care află maximul unui şir neordonat de numere, implementarea este clară şi este foarte puţin probabil ca cineva să ceară algoritmul corespunzător). Totuşi, dacă programul este mai complex - sortarea elementele şirului de exemplu - partea declarativă a specificaţiei poate să nu mai fie considerată suficientă şi algoritmul poate să fie necesar (alegerea algoritmului poate afecta eficienţa programului). Există o tradiţie în comunitatea metodelor formale de a încerca să se evite orice consideraţie asupra felului cum va fi implementat un program, încurajându-se stilul implicit de specificaţie. Abordarea reuşeste să ridice bariere între specificaţia formală - răspunsul la întrebarea "ce" - şi implementare - răspunsul la întrebarea "cum". Calea de trecere peste aceste bariere nu este întotdeauna evidentă şi, în ciuda realizărilor teoriilor moderne de rafinare a specificaţiilor formale, obstacolele practice în transformarea unei specificaţii implicite într-o procedură efectivă sunt imense. Mai mult decât atât, putem folosi Z sau metode similare pentru a specifica ceva care nu poate fi implementat pe un calculator - adică nu există un algoritm care produce rezultatul dat în specificaţie. Aceasta se datorează faptului ca Z nu este bazat pe un model de calcul (funcţii necalculabile pot fi specificate).
O altă problemă care trebuie abordată este scopul limbajelor de specificare formală în descrierea dinamicii sistemului. Interfaţa om-utilizator, de exemplu, este o parte vitală şi foarte complexă a majorităţii sistemelor software actuale; a încerca descrierea ei într-un limbaj ca Z, într-un mod suficient de inteligibil şi folositor este, în general, imposibil. De asemenea majoritatea sistemelor "safety-critical" sunt sisteme reactive, unde dinamica execuţiei şi momentul de apariţie a evenimentelor sunt de o importanţă vitală. Aceste dimensiuni ale problemei sunt foarte greu de reprezentat într-un limbaj precum Z.

Deşi s-ar părea că încercăm să minimalizăm eficienţa metodologiilor abstracte, nu aceas-

ta este intenţia noastra. În mod clar, folosirea lor a dus la un avans considerabil în specificarea sistemelor software, în special cele "safety-critical". Totuşi, credem că este nevoie de o metodologie unificatoare care poate fi folosită atât pentru descrierea aspectelor legate de procesarea datelor cât şi a informaţiilor dinamice ale sistemului. O astfel de metodologie va avea mijloacele de a descrie atât "ce" face sistemul cât şi "cum" face. Când spunem "cum" ne referim la un algoritm care poate rula pe un calculator şi produce rezultatul dorit; cu alte cuvinte un algoritm care poate fi descris de un model executabil pe un calculator.

Ceea ce susţinem aici este ideea că procesul de design şi specificaţie al sistemelor software trebuie să fie orientat în jurul construcţiei de modele executabile pe un calculator. Când construim un sistem software, construim ceva care, în operare, va da naştere unei funcţii calculabile. Pe de alta parte, funcţiile calculabile sunt identificate ca funcţii care rezultă din folosirea maşinilor Turing sau a altor modele algebrice similare. Totuşi, deşi maşina Turing, principalul model de calcul, a fost subiectul unor numeroase studii în diverse arii teoretice, ea nu este folosită în practică pentru construcţia sistemelor software, principalul motiv fiind că acest model este mult prea specializat pentru a permite abstractizarea necesară adaptării lui la specificarea şi designul de software. Modele mai puţin generale, ca de exemplu maşinile cu stări finite, constituie însă baza multor sisteme de specificaţie.

2. Maşinile cu stări finite ca metodă de specificaţie

Maşina cu stări finite este un model matematic al unui sistem cu intrări şi ieşiri discrete. La un moment dat, sistemul este într-una dintr-un număr finit de stări interne ale sistemului. O stare a sistemului conţine informaţia privitoare la intrările deja consumate care este necesară pentru determinarea comportamentului sistemului la aplicarea de noi intrări. Modelul matematic care descrie un astfel de comportament este alcătuit din următoarele elemente principale:

(i) un număr de intrări care pot fi aplicate sistemului

(ii) un număr finit de stări interne ale sistemului; dintre acestea una este stare iniţială iar una sau mai multe sunt stări finale.

(iii) un set de reguli care arată, pe baza stării curente şi intrării aplicate sis-temului, starea lui viitoare. Acestea sunt de multe ori reprezentate sub forma unei diagrame de tranziţie a stărilor.

Folosirea maşinilor cu stări finite este forte frecventă în designul de hardware secvenţial [1] cât şi în specificarea interfeţei om-calculator a sistemelor software [8]. Pentru a reprezenta în mod adecvat interfaţa om-calculator avem nevoie de un formalism care poate modela atât utilizatorul uman cât şi calculatorul. Cum multe modele ale comportamentului uman se bazează pe noţiunea de stare, este naturală modelarea interfeţei în acest mod. Un exemplu de reprezentare a unui procesor de text foarte simplu care are un meniu principal cu două opţiuni (Insert Drawing şi Insert Object), este redat în figura 1.

În general, maşinile cu stări finite sunt potrivite pentru reprezentarea aspectelor dinamice ale sistemului, folosirea elementelor grafice fiind folositoare din punctul de vedere al înţelegerii utilizatorului. De aceea, aspectele legate de partea de control a unui program sunt des reprezentate în acest fel. Totuşi, modelele bazate pe stări şi tranziţiile dintre ele nu au capacitatea de a reprezenta structuri de date cu mare complexitate; este foarte greu, dacă nu imposibil, de reprezen-tat orice structură netrivială de date în acest mod. Pe de altă parte, modele de calcul mai complexe precum maşinile Turing sau cele cu stive (pushdown automată, stack automată etc.) sunt mult prea restrictive în alegerea structurilor de date utilizate pentru a fi de vreun folos în specificarea şi designul aplicaţiilor practice.

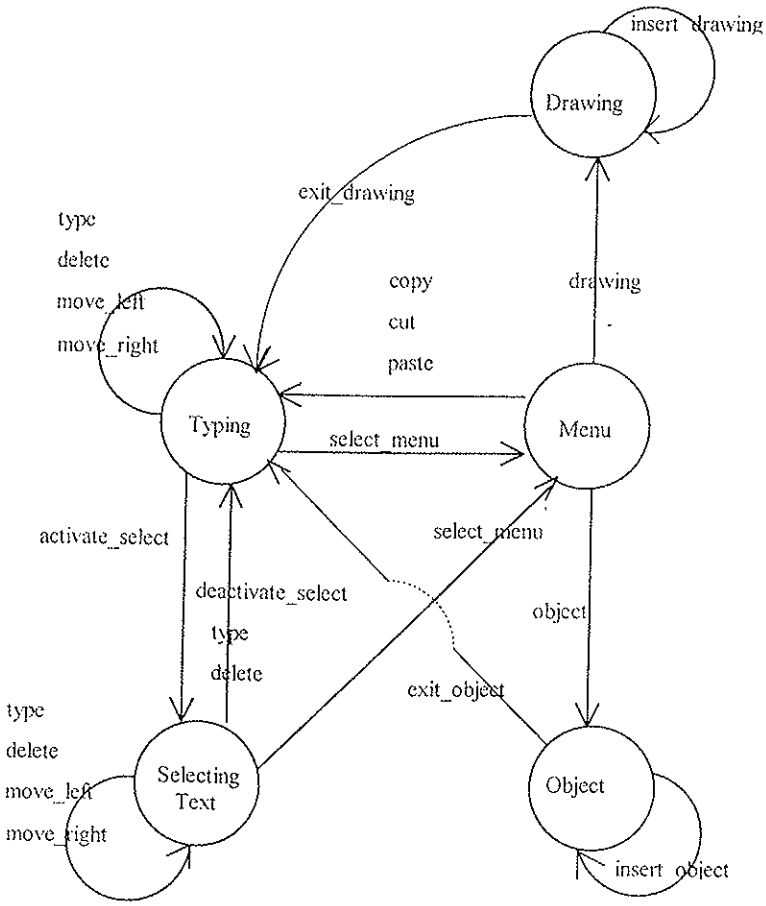

Fig. 1. Meniul principal cu două opțiuni

Există totuși o generalizare a acestor mașini care poate furniza un mediu corespunzător pentru descrierea și analiza sistemelor software. Mașinile X integrează cele două aspecte ale sistemului - partea de control și partea de procesare a datelor - permițând în același timp ca acestea să fie descrise separat. O diagramă de tipul celor folosite de mașinile cu stări finite este folosită pentru modelarea părții de control a sistemului în timp ce procesarea datelor poate fi reprezentată folosind metode ca Z sau VDM sau orice altă notație formală convenabilă.

3. Mașinile X

Mașina X a fost propusă de Holcombe [3] ca un posibil limbaj de specificație și un număr de studii ulterioarea au demonstrat valabilitatea acestei idei. În esență, mașina X este asemănătoare unei mașini cu stări finite, dar cu o deosebire fundamentală. Un set de date X este definit împreună cu un set de funcții (parțiale) Φ care opereaază pe X; Φ este numit tipul mașinii și constituie operațiile elementare pe care mașina este capabilă să le execute. Fiecare arc în diagrama de tranziție a stărilor este apoi etichetat cu o funcție din Φ, astfel că secvența tranzițiilor mașinii determină modul de procesare a datelor și, implicit, funcția calculată de mașină. Setul de date X poate conține informații despre memoria internă a sistemului cât și diferite tipuri de intrări sau ieșiri, astfel încât este posibilă modelarea unor sisteme foarte generale în acest mod. O definiție mai riguroasă a masinilor X este dată în ceea ce urmează.

Definiție:
O mașină X este o mulțime ordonată M = (X, Y, Z, α, β, Q, Φ, F, I, T), unde
1. X este setul fundamental de date pe care mașina opercază.
2. Y și Z sunt seturile de intrări și respectiv ieșiri ale mașinii.

3. α şi β sunt funcţiile de intrare şi respectiv ieşire, folosite să convertească intrările şi ieşirile în şi respectiv din setul fundamental de date, α: $Y \to X$, β: $X \to Z$
4. Q este un set finit de stări.
5. Φ este tipul lui M, un set de funcţii parţiale pe X, Φ: $2^{(X \to X)}$
7. F este funcţia (parţială) a tranziţiilor de stare, F: $Q \times \Phi \to Q$.
De obicei F este descrisă ca o diagramă de tranziţie a stărilor.
8. $I \subseteq Q$ şi $T \subseteq Q$ sunt seturile de stări iniţiale, respectiv finale.
Maşina este deterministă dacă are o singură stare iniţială ($I = \{q_0\}$) şi dacă oricare două funcţii care sunt folosite ca etichete de arcuri distincte care pornesc din acceaşi stare au domeniile disjuncte. Mai riguros, $\forall q \in Q$, $\varphi, \varphi' \in \Phi$, dacă $(q, \varphi), (q, \varphi') \in$ dom F atunci dom $\varphi \cap$ dom $\varphi' = \varnothing$. În acest caz, secvenţa de tranziţii ale maşinii pentru orice element $x \in X$ este unică.
Modelul prezentat anterior este foarte general şi este uşor de arătat că o maşină Turing, de exemplu, poate fi reprezentată uşor în acest fel (vezi [7]). Pe de altă parte, metoda permite separarea părţii de control a sistemului de partea de procesare a datelor şi aceasta este un mare avantaj din punctul de vedere al claritătii şi înţelegerii utlizatorului.
Să vedem, de exemplu cum ar putea fi specificat în acest mod procesorul de text prezentat în figura 1. În acest caz, seturile de intrări şi ieşiri vor fi secvenţe de intrări respectiv ieşiri intermediare. Deci $Y = \Sigma^*$ şi $Z = \Gamma^*$, unde Σ şi Γ reprezintă seturile de intrări, respectiv iesiri, intermediare; vom numi Σ şi Γ alfabete de intrare, respectiv ieşire, ale maşinii. Atunci, setul fundamental de date va fi de forma $X = \Gamma^* \times M \times \Sigma^*$, unde M reprezintă memoria internă a sistemului. În cazul nostru, alfabetul de intrare va fi compus din taste caracter, obiecte şi desene ce pot fi inserate cât şi orice altă acţiune care va cauza schimbarea stării interne a sistemului (de exemplu tasta delete, orice opţiune a meniului etc.), deci Σ = ELEMENTS $\cup$ ACTIONS, unde ELEMENTS = CHARACTERS $\cup$ DRAWINGS $\cup$ OBJECTS.
Memoria internă poate fi reprezentată ca produsul cartezian
M = DOCUMENTS x CLIPBOARDS, unde DOCUMENTS şi CLIPBOARDS sunt reprezentări convenabile pentru documentul de procesat, respectiv clipboard. De asemenea, se poate considera că ieşirea sistemului este documentul însuşi plus eventual un set de mesaje date de sistem, deci Γ = DOCUMENTS x MESSAGES.
Bineînţeles, o specificaţie detaliată trebuie să cuprindă definiţiile seturilor folosite anterior, dar aceasta nu ridică probleme deosebite (de exemplu CLIPBOARDS = ELEMENT*, DOCUMENTS = $(\text{ELEMENTS}^*)^3$ iar celelalte seturi se descriu ca enumerări de elemente) şi nu este scopul lucrării de faţă.
Etichetele arcurilor din figura 1 reprezintă acum funcţii (parţiale) Φ: $\Gamma^* \times M \times \Sigma^* \to \Gamma^* \times M \times \Sigma^*$, ele formează tipul maşinii Φ. Pentru o valoare de memorie şi o intrare, orice astfel de φ va schimba valoarea memoriei şi va produce o ieşire, intrarea citită fiind eliminată. Mai riguros, orice $\varphi \in \Phi$ va fi de forma următoare:
(i) φ (g, m,ε) este nedefinit $\forall g \in \Gamma^*$, $m \in M$ şi
(ii) $\forall m \in M$, $\sigma \in \Sigma$ fie:
(a) φ(g, m, σs) este nedefinit $\forall g \in \Gamma^*$, $s \in \Sigma^*$ sau
(b) $\exists m' \in M$, $\gamma \in \Gamma$ (care depind de m and σ) astfel încât φ(g, m, σs) = ($g\gamma$, m', s) $\forall g \in \Gamma^*$, $s \in \Sigma^*$.
Nu vom intra în detaliu în ceea ce priveşte definirea fiecarei funcţii în parte, pentru amănunte vezi [5] care prezintă specificaţia unui procesor de text ceva mai detaliat.
Funcţiile de intrare şi respective ieşire vor fi de forma $\alpha(s) = (\varepsilon, m_0, s)$ $\forall s \in \Sigma^*$, unde m_0 reprezintă valoarea iniţială a memoriei interne iar ε este secvenţa nulă, respectiv

$$\beta(g,m,s) = \begin{cases} g, \text{daca } s = \varepsilon \\ \text{nedefinit, altfel} \end{cases} \quad \forall m \in M,\ g \in \Gamma^*$$

Deci funcția de intrare va inițializa memoria internă și secvența intrărilor (cea a ieșirilor fiind evident nulă), iar funcția de ieșire va extrage secvența de ieșiri rezultată în cazul în care mașina a terminat de citit secvența de intrări.

Un mare avantaj al specificațiilor folosind mașinile X este ușurința cu care acestea se pot rafina treptat putîndu-se construi astfel specificații complexe într-un mod foarte intuitiv. În termeni foarte generali, acest proces este realizat prin înlocuirea arcurilor din mașina inițială cu mașini care descriu, la un nivel mai detaliat, aceeași funcționalitate ca și funcțiile ce etichetează arcurile înlocuite. De exemplu, putem să descriem în detaliu procesul de inserare a unui desen în document. Pentru simplitate, să presupunem că singurile desene care pot fi inserate sunt linii și dreptunghiuri și că operațiunea de desenare a acestora constă din următorii pași: alegera figurii (linie sau dreptunghi) și alegerea a două puncte (reprezentând capetele segmentului sau colțurile opuse ale dreptunghiului). Ca rezultat, mașina a cărei diagramă este reprezentată în figura 2 va înlocui arcul etichetat *insert_drawing* în diagrama mașinii inițiale.

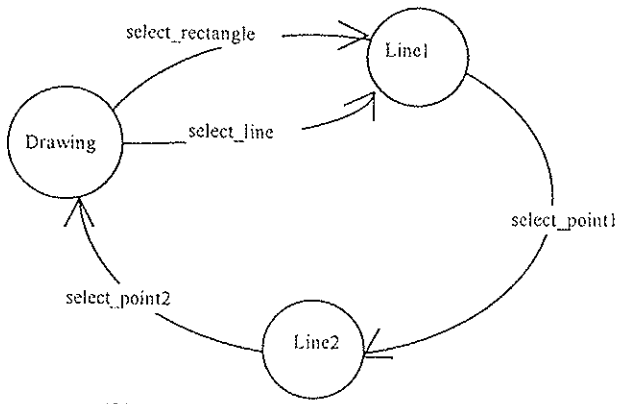

Fig. 2. Arc rafinat al mașinii inițiale

Bineînțeles, alfabetele de intrare și ieșire cât și memoria sistemului trebuiesc expandate în mod corespunzător. În mod evident, acest proces de rafinare trebuie însoțit de o serie de teoreme care să garanteze că transformarea păstrează, într-un anumit sens, funcționalitatea mașinii inițiale. Baza teoretică pentru câteva tipuri de astfel de transformări există deja (vezi [5]), alte tipuri de transformări constituie obiectul unor cercetări curente.

4. Concluzii

Avantajele mașinilor X ca metodă de specificație includ:

- Este intuitivă și ușor de folosit.
- Folosește cunoștințe existente și nu introduce vreun concept revoluționar: modelul este o combinație de diagrame de tip mașini cu stări finite și o descriere formală a unor tipuri de date și funcții care pot fi reprezentate ușor în Z sau un limbaj funcțional ca ML, de exemplu, sau chiar folosind notații matematice tradiționale.
- Permite reprezentarea aspectelor dinamice ale sistemului într-un mod intuitiv.
- Permite specificarea unui sistem la mai multe nivele de detaliu: o funcție de procesare într-o specificație de nivel superior poate fi reprezentată ca o mașină într-o specificație mai detaliată.
- Poate reprezenta orice tip de operație de calcul, oricât de complexă: mașina Turing este un caz particular al mașinilor X.

În ultimii ani, numeroase studii de caz au arătat eficiența mașinilor X ca metodă de specificare a sistemelor software (vezi [2] sau [6]). De asemenea, a fost dezvoltată o teorie a testării sistemelor software bazată pe acest model de spe-cificație (vezi [6], [4]) care, spre deosebire de metodele cunoscute până acum, garantează corectitudinea sistemului cu condiția ca anumite condiții de

design să fie satisfăcute. Aceasta constituie un avans deosebit în domeniul testării sistemelor software, care a adus modelul mașinilor X în centrul atenției industriei, concernul Daimler-Benz, colaborând cu Universitatea din Sheffield, Mare Britanie, pentru crearea instrumentelor necesare aplicării metodei.

Bibliografie

[1] Clements, A.: *The Principles of Computer Hardware*. Oxford Science, 1991.

[2] Fairtlough, M., Holcombe, M., Ipate, F., Jordan, C., Laycock, G., Duan, Z..: Using an X-machine to Model a Video Cassette Recorder. *Current issues in electronic modelling*, 3, 141-161, 1995.

[3] Holcombe, M.: X-machines as a Basis for Dynamic System specification. *Software Engineering Journal*, **3**(2), 69-76, 1988.

[4] Holcombe M., Ipate F., Grondoudis A.: Complete Functional Testing of Safety-Critical Systems. *Proceedings of Second IFAC Workshop on Safety and Reliability in Emerging Control Technologies*, Daytona Beach, Florida, USA, 1-3 November 1995.

[5] Ipate F.: *Theory of X-machines and Applications in Specification and Testing*. PhD thesis, University of Sheffield, UK, 1995.

[6] Ipate F, Holcombe M.: An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics*, 1997.

[7] Ipate F., Holcombe M.: Another look at computability. *Informatica*, 20(3), 359-372, 1996.

[8] Wasserman, A. I., Pircher, P. A., Shewmake, D. T.: Building Reliable Interactive Information Systems. *IEEE Transactions on Software Engineering*, SE 12, 1986, 147-156.