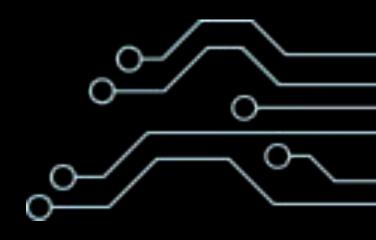


Modelare și Analiză cu Event-B

Folosind platforma Rodin



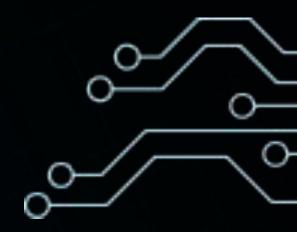
Conținut

01 Introducere în Modelarea Formală

02 Ce este Event-B?

03 Ce este platforma Rodin?

04 Exemplul echipei - ATM



Ce este Modelarea Formală?

Tehnică de dezvoltare ce folosește matematica pentru a descrie și verifica sisteme software și hardware.

Asigură corectitudinea sistemelor, evitând erori critice și identificând probleme din timp.





01

Analiză statică

Verificarea proprietăților de siguranță prin demonstrarea matematică a invarianților

02

Analiză de consistență

Verificarea că rafinarea păstrează comportamentul corect al modelului abstract

03

Verificare formală

Generarea și demonstrarea obligațiilor de dovadă pentru a garanta corectitudinea matematică a modelului

Ce este Event-B

Event-B asigură o bază solidă pentru dezvoltarea de sisteme critice, prin verificarea formală a sistemelor înainte de implementarea propriu-zisă, reducând riscul de erori și optimizând procesul de dezvoltare.

Dezvoltarea Software Tradițională

- Metodă: "Build & Fix"
- Proces: Testare după implementare
- Rezultat: Buguri descoperite târziu
- Impact: Posibile întârzieri şi costuri crescute

Event-B

- Metodă: Verificare matematică ÎNAINTE de implementare
- Proces: Garanții formale
- Rezultat: Sisteme mai sigure
- Impact: Eficiență crescută și fiabilitate

Utilizare reală: Sisteme feroviare, bancare, medicale, aviatice



Concepte Fundamentale în

Event-B

Modelul bazat pe stări și evenimente

Într-un sistem
bancar, variabilele ar
putea include
soldul_contului, iar
evenimentele ar
putea fi depunere sau
retragere.

Proprietăți verificabile matematic

Invariantul "soldul nu poate fi negativ" asigură că toate tranzacțiile respectă regulile financiare fundamentale.

Rafinare incrementală

Se poate începe cu un model general al tranzacţiilor bancare, apoi se adaugă detalii suplimentare, cum ar fi gestionarea diferitelor tipuri de conturi și taxe.

Ce este platforma Rodin



Un instrument pentru proiectarea sigură a sistemelor complexe

Rodin este o platformă software open-source dezvoltată pentru multiple scopuri:

- Modelarea formală precisă pentru sisteme care nu își permit greșeli
- Verificarea automată a modelelor, utilizând metoda formală Event-B
- Dezvoltare graduală trecerea de la idee generală la modele detaliate

Rodin ajută la proiectarea de sisteme unde siguranța este esențială, cum ar fi în transport, energie sau sănătate.

Rodin: rol, avantaje și limitări

Rodin oferă:

- Generarea automată pentru Proof Obligations
- Demonstratoare automate integrate
- Verificare înainte de implementare

Avantaje

- Detectare rapidă a erorilor
- Siguranţă sporită
- Dezvoltare incrementală
- Automatizarea procesului

Limitări

- Necesită cunoștinte de metodologii formale și Event-B
- Limitat la Event-B
- Suport limitat pentru alte tool-uri

Proof Obligations în Event-B

Proof Obligations sunt formule logice generate automat de platforma Rodin care verifică corectitudinea formală a modelului.

Ele asigură că:

- Invarianții sunt păstrați în toate evenimentele
- Rafinamentele sunt corecte
- Iniţializarea respectă proprietăţile definite
- Gărzile sunt suficient de puternice pentru a garanta acțiunile

Tipuri de Proof Obligations



Demonstrează că expresiile sunt bine definite



Verifică menținerea unui invariant de către un eveniment

GRD

Verifică consistența gărzilor

SIM

Verifică validarea unui comportament într-o simulare

Verificarea Proprietăților



Verificarea proprietăților în ATMSystem

Tipuri de obligații de dovadă și statusul lor:

- Proof obligations automate (verificate automat de tool-ul Event-B)
- Proof obligations manuale (cer verificare umană)

Exemple din ATMSystem:

- @inv2/WD (Proof obligation automată, validă)
- INITIALISATION/@inv3/INV (Proof obligation automată, validă)
- CheckBalance/@grd1/GRD 1 (Proof obligation manuală, necesită verificare)
- Deposit/@act1/WD V (Proof obligation automată, validă)
- CheckBalance/@act1/SIM V (Proof obligation manuală, necesită verificare)

Exemplul nostru - Contextul ATM

Seturi principale

- USERS: utilizatori care pot accesa ATM-ul
- CARDS: carduri asociate utilizatorilor
- PINS: Coduri PIN pentru fiecare card

Constante

- maxWithdrawal: suma maximă care poate fi retrasă
- dailyLimit: limita
 zilnică de retragere
- ownsCard: asociază utilizatorii cu cardurile lor
- validPIN: PIN valid pentru fiecare card
- initialBalance: soldul inițial al conturilor

Axiome

Setează valorile pentru maximum, limita zilnică și relațiile între utilizatori, carduri și PIN-uri

Structura Contextului

```
ATMContext
USERS
maxWithdrawal
dailyLimit
ownsCard
validPIN
initialBalance
user2
cardl
card3
pinl
pin2
pin3
        maxWithdrawal ∈ N1 not theorem
        maxWithdrawal = 2000 not theorem >
        dailyLimit ∈ N1 not theorem
        dailyLimit = 5000 not theorem
        ownsCard ∈ USERS ↔ CARDS not theorem
        ownsCard~ ∈ CARDS → USERS not theorem >
        validPIN ∈ CARDS → PINS not theorem
        initialBalance ∈ N1 not theorem
        initialBalance = 1000 not theorem >
@axm10: partition(USERS, {user1}, {user2}) not theorem
@axmll: partition(CARDS, {cardl}, {card2}, {card3}) not theorem >
@axm12: partition(PINS, {pin1}, {pin2}, {pin3}) not theorem >
@axm13: ownsCard = {user1 + card1, user1 + card2, user2 + card3} not theorem >
@axm14: validPIN = {card1 +> pin1, card2 +> pin2, card3 +> pin3} not theorem >
@axml5: userl ∈ USERS not theorem
@axml6: user2 ∈ USERS not theorem
@axml7: cardl ∈ CARDS not theorem
@axml8: card2 ∈ CARDS not theorem
@axm19: card3 ∈ CARDS not theorem
@axm20: pin1 ∈ PINS not theorem
@axm21: pin2 ∈ PINS not theorem
@axm22: pin3 ∈ PINS not theorem
```

Maşina inițială - Definirea Variabilelor și Evenimentelor

Variabile

- loggedIn: Utilizatori care sunt autentificaţi
- accountBalance: Soldul fiecărui card
- cardBlocked: Carduri blocate
- dailyWithdrawn: Retragerile zilnice ale cardurilor

Evenimente

- Login: Permite autentificarea utilizatorilor
- CheckBalance: Verifică soldul contului
- Withdraw: Permite retragerea de bani
- Logout: Deconectează utilizatorul
- BlockCard: Blochează un card în caz de comportament necorespunzător (ex: prea multe încercări)

Variabile și Evenimente

VARIABLES loggedIn accountBalance cardBlocked dailyWithdrawn

Rafinarea Modelului

Gestionarea Încercărilor Eșuate de Autentificare

```
Login: not extended ordinary >
REFINES
Login
ANY
aCard
user
enteredPin
WHERE
aGgrd1: user © USERS not theorem
aGgrd2: aCard © CARDS not theorem
aGgrd3: user > aCard © ownsCard not theorem
aGgrd4: aCard © cardBlocked not theorem
aGgrd5: user © loggedIn not theorem
aGgrd6: enteredPin © PINS not theorem
aGgrd7: enteredPin = validPIN(aCard) not theorem
aGgrd8: failedAttempts(aCard) < 3 not theorem
aGgrd8: failedAttempts(aCard) = 0
aGact1: loggedIn = loggedIn u {user} >
END
```

```
UnblockCard: not extended ordinary

ANY

aCard

WHERE

aCard ∈ cardBlocked not theorem

THEN

aCard: cardBlocked = cardBlocked \ {aCard}

aCard}

BOD

BOD
```

Modificări:

- Introducerea variabilei failedAttempts pentru a urmări încercările de autentificare
- Cardurile sunt **blocate** după 3 încercări eșuate

Evenimente suplimentare:

- FailedLogin: Blochează cardul la 3 tentative eșuate de login
- UnblockCard adăugat pentru deblocarea unui card

Invariant:

- Dacă un card are 3 încercări eșuate, acesta devine blocat
- Orice utilizator asociat unui card blocat nu poate să se logheze

Refinment - pentru Login: Modificarea evenimentului pentru a ține cont de numărul de încercări.

Concluzii și Avantaje după Rafinare

Avantaje

- Gestionarea încercărilor eșuate de autentificare adaugă un strat de securitate suplimentar.
- Modelul este flexibil şi permite extinderea uşoară pentru noi funcţionalităţi.
- Folosirea axiomei şi a invarianţilor garantează corectitudinea
 comportamentului sistemului

Limitări

- Complexitatea modelului creşte odată cu introducerea mai multor evenimente
- Managementul cardurilor bancare poate necesita monitorizare suplimentară în scenarii reale

Exemplu Funcție Rafinată

```
not extended ordinary >
Withdraw:
REFINES
     Withdraw
    user
    aCard
    amount
    @grdl: user ∈ loggedIn not theorem >
            aCard ∈ CARDS not theorem
           user → aCard ∈ ownsCard not theorem >
    @grd4: aCard ∉ cardBlocked not theorem >
            amount ∈ N1 not theorem >
    @grd6: amount ≤ accountBalance(aCard) not theorem >
    @grd7: amount ≤ maxWithdrawal not theorem >
            dailyWithdrawn(aCard) + amount ≤ dailyLimit not theorem
    @actl: accountBalance(aCard) = accountBalance(aCard) - amount >
    @act2: dailyWithdrawn(aCard) = dailyWithdrawn(aCard) + amount >
END
```

Evenimentul Withdraw din Maşina Rafinată

Avantajele Modelării cu Event-B

Fiabilitate Crescută:

Asigură verificarea riguroasă a corectitudinii sistemului înainte de implementare.

Prevenirea Erorilor:

Identifică problemele potențiale în etapele inițiale ale dezvoltării, reducând costurile de remediere.

Rafinare Incrementală:

Permite dezvoltarea treptată a sistemului de la modele simple la modele complexe.

Limitările și Provocările Modelării cu Event-B

1 Complexitatea Învățării:

Necesită o curbă de învățare abruptă datorită utilizării conceptelor matematice avansate.

Costul Iniţial:

Investiții inițiale mai mari în timp și resurse pentru setarea și modelarea formală.

3 Scalabilitate:

Poate fi provocator să modelezi sisteme extrem de mari sau foarte detaliate din cauza complexității crescute.

Multumim!