



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.

Tipuri de analiză realizate

01

Analiză statică

Verificarea proprietăților de siguranță prin demonstrarea matematică a invariantilor

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 și 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 tranzacțiilor 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
- Siguranță sporită
- Dezvoltare incrementală
- Automatizarea procesului

Limitări

- Necesită cunoștințe 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
- Inițializarea respectă proprietățile definite
- Gărzile sunt suficient de puternice pentru a garanta acțiunile

Tipuri de Proof Obligations

WD

Demonstrează că expresiile sunt bine definite

INV

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 - ⚠ (Proof obligation manuală, necesită verificare)
- Deposit/@act1/WD - ✓ (Proof obligation automată, validă)
- CheckBalance/@act1/SIM - ✓ (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

```
CONTEXT
  ATMContext
SETS
  USERS
  CARDS
  PINS
CONSTANTS
  maxWithdrawal
  dailyLimit
  ownsCard
  validPIN
  initialBalance
  user1
  user2
  card1
  card2
  card3
  pin1
  pin2
  pin3
AXIOMS
  @axm1: maxWithdrawal ∈ N1 not theorem
  @axm2: maxWithdrawal = 2000 not theorem
  @axm3: dailyLimit ∈ N1 not theorem
  @axm4: dailyLimit = 5000 not theorem
  @axm5: ownsCard ∈ USERS ↔ CARDS not theorem
  @axm6: ownsCard~ ∈ CARDS ↔ USERS not theorem
  @axm7: validPIN ∈ CARDS → PINS not theorem
  @axm8: initialBalance ∈ N1 not theorem
  @axm9: initialBalance = 1000 not theorem
  @axm10: partition(USERS, {user1}, {user2}) not theorem
  @axm11: partition(CARDS, {card1}, {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
  @axm15: user1 ∈ USERS not theorem
  @axm16: user2 ∈ USERS not theorem
  @axm17: card1 ∈ CARDS not theorem
  @axm18: card2 ∈ CARDS not theorem
  @axm19: card3 ∈ CARDS not theorem
  @axm20: pin1 ∈ PINS not theorem
  @axm21: pin2 ∈ PINS not theorem
  @axm22: pin3 ∈ PINS not theorem
END
```

ATMContext.buc



Mașina inițială - Definirea Variabilelor și Evenimentelor

Variabile

- *loggedIn*: Utilizatori care sunt autentificați
- *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

```
EVENTS
  INITIALISATION: not extended ordinary
  THEN
    @act1: loggedIn = {}
    @act2: accountBalance = CARDS × {initialBalance}
    @act3: cardBlocked = {}
    @act4: dailyWithdrawn = CARDS × {0}
  END

  Login: not extended ordinary
  ANY
    user
    aCard
    enteredPin
  WHERE
    @grd1: user ∈ USERS not theorem
    @grd2: aCard ∈ CARDS not theorem
    @grd3: user ⇨ aCard ∈ ownsCard not theorem
    @grd4: ¬(aCard ∈ cardBlocked) not theorem
    @grd5: user ∉ loggedIn not theorem
    @grd6: enteredPin ∈ PINS not theorem
    @grd7: enteredPin = validPIN(aCard) not theorem
  THEN
    @act1: loggedIn = loggedIn u {user}
  END
```

```
VARIABLES
  loggedIn
  accountBalance
  cardBlocked
  dailyWithdrawn
```


Rafinarea Modelului

Gestionarea Încercărilor Eșuate de Autentificare

```
« Login: not extended ordinary »
REFINES
« Login
ANY
« aCard »
« user »
« enteredPin »
WHERE
« @grd1: user ∈ USERS not theorem »
« @grd2: aCard ∈ CARDS not theorem »
« @grd3: user → aCard ∈ ownsCard not theorem »
« @grd4: aCard ∉ cardBlocked not theorem »
« @grd5: user ∉ loggedIn not theorem »
« @grd6: enteredPin ∈ PINS not theorem »
« @grd7: enteredPin = validPIN(aCard) not theorem »
« @grd8: failedAttempts(aCard) < 3 not theorem »
THEN
« @act2: failedAttempts(aCard) := 0 »
« @act1: loggedIn := loggedIn ∪ {user} »
END
```

```
« UnblockCard: not extended ordinary »
ANY
« aCard »
WHERE
« @grd1: aCard ∈ cardBlocked not theorem »
THEN
« @act1: cardBlocked = cardBlocked \ {aCard} »
« @act2: failedAttempts(aCard) := 0 »
END
```

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

Refinement - 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 și permite extinderea ușoară pentru noi funcționalități.
- Folosirea axiomei și a invariantilor garantează corectitudinea comportamentului sistemului

Limitări

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

Exemplu Funcție Rafinată

```
Withdraw: not extended ordinary ›
REFINES
  ◦ Withdraw
ANY
  ◦ user ›
  ◦ aCard ›
  ◦ amount ›
WHERE
  ◦ @grd1: user ∈ loggedIn not theorem ›
  ◦ @grd2: aCard ∈ CARDS not theorem ›
  ◦ @grd3: user → aCard ∈ ownsCard not theorem ›
  ◦ @grd4: aCard ∉ cardBlocked not theorem ›
  ◦ @grd5: amount ∈ N1 not theorem ›
  ◦ @grd6: amount ≤ accountBalance(aCard) not theorem ›
  ◦ @grd7: amount ≤ maxWithdrawal not theorem ›
  ◦ @grd8: dailyWithdrawn(aCard) + amount ≤ dailyLimit not theorem ›
THEN
  ◦ @act1: accountBalance(aCard) = accountBalance(aCard) - amount ›
  ◦ @act2: dailyWithdrawn(aCard) = dailyWithdrawn(aCard) + amount ›
END
```

Evenimentul Withdraw din Mașina Rafinată

Avantajele Modelării cu Event-B

1. **Fiabilitate Crescută:**
Asigură verificarea riguroasă a corectitudinii sistemului înainte de implementare.
2. **Prevenirea Erorilor:**
Identifică problemele potențiale în etapele inițiale ale dezvoltării, reducând costurile de remediere.
3. **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.
2. Costul Inițial:
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!