Especificação Formal

Métodos formais

- A especificação formal é parte de um coleção mais geral de técnicas que são conhecidas como 'métodos formais'.
- São todas baseadas na representação matemática e na análise de software.
- Os métodos formais incluem
  - Especificação formal;
  - Análise e prova de especificação;
  - Desenvolvimento transformacional;
  - Verificação de programa.

Aceitação de métodos formais

- Os métodos formais não se tornaram técnicas principais de desenvolvimento de software conforme foram previstos
  - Outras técnicas de engenharia de softawre tiveram sucesso no aumento da qualidade de sistema.
  - Mudanças de mercado fizeram do tempo de entrega o fator chave, ao invés do software com uma baixa contagem de erros.
  - O escopo de métodos formais é limitado
  - Os métodos formais ainda são difíceis de serem extendidos para sistemas de grande porte.

O uso de métodos formais

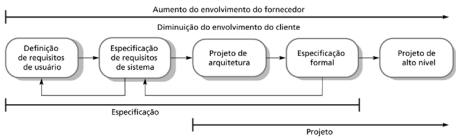
- Um dos principais benefícios dos métodos formais é a redução do número de defeitos nos sistemas.
- Principal área de aplicação é a engenharia de sistemas críticos.
  - O uso de métodos formais é mais apropriado em termos de custo, porque os altos custos de falha de sistema devem ser evitados.

Especificação formal no processo de software

- Especificação e projeto são fortemente interligados.
- O projeto de arquitetura é essencial para estruturar uma especificação e o processo de especificação.
- Especificações formais são expressos em uma notação matemática com vocabulário, sintaxe e semântica precisamente definidos.

Especificação e projeto

Figura 10.1
Especificação e projeto.



Especificação formal no processo de software

Figura 10.2
Especificação formal no processo de software.

Uso da especificação formal

- Fases iniciais do desenvolvimento de software.
- Isso reduz os erros de requisitos à medida que força uma análise detalhada dos requisitos.
- Não completeza e a inconsistên podem ser descobertas e resolvidas.
  - Economiza-se à medida que a quantidade de retrabalho diminui devido à redução dos problemas de requisitos.

Técnicas de especificação

- Especificação algébrica
  - O sistema é especificado em termos de operações e de seus relacionamentos.
- Especificação baseada em modelos
  - O sistema é especificado nos termos de um modelo de estados que é desenvolvido usando construções matemáticas, tais como conjuntos e seqüências. As operações são definidas pelas modificações no estado do sistema.

Linguagens formais de especificação

Tabela 10.1 Linguagens formais de especificação

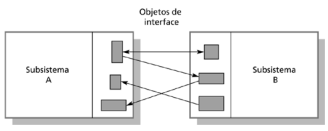
	Seqüencial	Concorrente
Algébrica	Larch (Guttag, et al., 1993); OBJ (Futatsugi, et al., 1985)	Lotos (Bolognesi e Brinksma, 1987)
Baseada em modelos	Z (Spivey, 1992); VDM (Jones, 1980); B (Wordsworth, 1996)	CSP (Hoare, 1985); redes de Petri (Peterson, 1981)

Especificação de interface

- Sistemas de grande porte são decompostos em subsistemas com interfaces bem definidas entre eles.
- A especificação das interfaces dos subsistemas permite o desenvolvimento independente dos diferentes subsistemas.
- As interfaces podem ser definidas como tipos de dados abstratos ou classes de objetos.
- A abordagem algébrica para a especificação formal é, particularmente, bem adequada para a especificação da interface, já que ela se concentra nas operações definidas em um objeto.

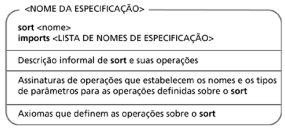
Interfaces de subsistemas

Figura 10.4
Objetos de interface de subsistemas.



A estrutura de uma especificação algébrica

Figura 10.5
Estrutura de uma especificação algébrica.



Componentes de especificação

- Introdução
  - Define o sort (o nome do tipo) e declara outras especificações que são usadas.
- Descrição
  - Descreve Informalmente as operações sobre o tipo.
- Assinatura
  - Define a sintaxe das operações na interface e seus parâmetros.
- Axiomas
  - Define a semâmtica de operação pela definição de axiomas que caracterizam o comportamento.

Operações de especificação

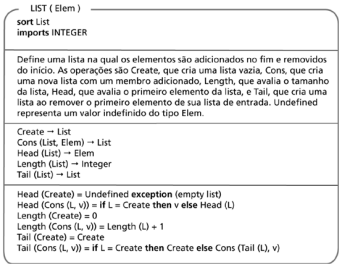
- Operações de construtor. São operações que criam entidades do tipo que está sendo especificado.
- Operações de inspeção. Operações que avaliam entidades do tipo que está sedo especificado.
- Servem para especificar o comportamento, definir as operações do inspetor para cada operação de construtor.

Operações sobre uma lista de tipo abstrato de dados (ADT)

- Operações do construtor que avaliam a sort List
  - Create, Cons e Tail.
- Operações de inspeção que tomam a sort List como parâmetro e retornam algum outro sort
  - Head e Length.
- Tail pode ser definido usando os construtores mais simples Create e Cons. Não há necessidade de definir Head e Length com Tail.

Especificação de lista

Figura 10.6
Simples especificação de lista.



Recursão nas especificações

- As operações são freqüentemente especificadas recursivamente.
- Tail (Cons (L, v)) = **if** L = Create **then** Create **else** Cons (Tail (L), v).
  - Cons ([5, 7], 9) = [5, 7, 9]
  - Tail ([5, 7, 9]) = Tail (Cons ([5, 7], 9)) =

Cons (Tail ([5, 7]), 9) = Cons (Tail (Cons ([5], 7)), 9) =
Cons (Cons (Tail ([5]), 7), 9) =
Cons (Cons (Tail (Cons ([], 5)), 7), 9) =
Cons (Cons ([Create], 7), 9) = Cons ([7], 9) = [7, 9]

Especificação da interface nos sistemas críticos

- Considere um sistema de controle de tráfego aéreo onde uma aeronave voa através dos setores de espaço aéreo controlados.
- Cada setor pode incluir uma série de aeronaves, mas, por razões de segurança, devem estar separados.
- Nesse exemplo, uma simples separação vertical de 300 metros é proposta.
- O sistema deve avisar o controlador se uma aeronave é instruída a mover-se de tal maneira que a regra de separação seja desrespeitada.

Um objeto setor

- As operações críticas sobre um objeto que representa um setor controlado são
  - Enter. Adiciona uma aeronave no espaço aéreo controlado;
  - Leave. Remove uma aeronave do espaço aéreo controlado;
  - Move. Move uma aeronave de uma altitude para outra;
  - Lookup. Dado um identificador da aeronave, essa atitude retorna a altitude real da aeronave;

Operações primitivas

- Algumas vezes, é necessário introduzir operações adicionais para simplificar a especificação.
- As outras operações podem, então, ser definidas usando essas operações primitivas.
- Operações primitivas
  - Create. Causa a criação de uma instância em um setor;
  - Put. Adiciona uma aeronave sem verificações de segurança;
  - In-space. Determina se uma dada aeronave está no setor;
  - Occupied. Dada uma altitude, determina se há uma aeronave dentro de 300 metros dessa altitude.

Especificação de setor (1)

Figura 10.7
Especificação de um setor controlado.

SECTOR

sort Sector
imports INTEGER, BOOLEAN

Enter	— adiciona uma aeronave no setor se as condições de segurança forem satisfeitas
Leave	— remove uma aeronave do setor
Move	— move uma aeronave de uma altitude para outra, se for seguro
Lookup	— encontra a altitude de uma aeronave no setor
Create	— cria um setor vazio
Put	— adiciona uma aeronave a um setor sem verificações de restrições
In-space	— verifica se uma aeronave já está em um setor
Occupied	— verifica se uma altitude especificada está disponível

Enter (Sector, Call-sign, Height) → Sector
Leave (Sector, Call-sign) → Sector
Move (Sector, Call-sign, Height) → Sector
Lookup (Sector, Call-sign) → Height

Create → Sector
Put (Sector, Call-sign, Height) → Sector
In-space (Sector, Call-sign) → Boolean
Occupied (Sector, Height) → Boolean

Especificação de setor (2)

Figura 10.7
Especificação de um setor controlado.

```
Enter (S, CS, H) =
    if      In-space (S, CS) then S exception (Aircraft already in sector)
    elsif   Occupied (S, H) then S exception (Height conflict)
    else    Put (S, CS, H)

Leave (Create, CS) = Create exception (Aircraft not in sector)
Leave (Put (S, CS1, H1), CS) =
    if CS = CS1 then S else Put (Leave (S, CS), CS1 , H1)

Move (S, CS, H) =
    if      S = Create then Create exception (No aircraft in sector)
    elsif   not In-space (S, CS) then S exception (Aircraft not in sector)
    elsif   Occupied (S, H) then S exception (Height conflict)
    else    Put (Leave (S, CS), CS, H)

-- NO-HEIGTH é uma constante que indica se uma altitude válida não pode ser retornada

Lookup (Create, CS) = NO-HEIGHT exception (Aircraft not in sector)
Lookup (Put (S, CS1, H1), CS) =
    if CS = CS1 then H1 else Lookup (S, CS)

Occupied (Create, H) = false
Occupied (Put (S, CS1 , H1), H) =
    if      (H1 > H and H1 - H ≤ 300) or (H > H1 and H - H1 ≤ 300) then true
    else    Occupied (S, H)

In-space (Create, CS) = false
In-space (Put (S, CS1, H1), CS) =
    if CS = CS1 then true else In-space (S, CS)
```


Comentários de especificação

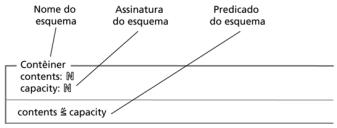
- Usar construtores básicos Create e Put para especificar outras operações.
- Definir Occupied e In-space usando Create e Put e usá-los para fazer verificações em outras definições de operação.
- Todas as operações que resultem em mudanças para o setor devem verificar se os critérios de segurançase se mantém.

Especificação de comportamento

- A especificação algébrica pode ser incômoda quando as operações do objeto não são independentes de seu estado.
- A especificação baseada em modelos expõe o estado do sistema e define as operações em termos de mudanças para esse estado.
- A notação Z é uma técnica madura para especificação baseada em modelos. Ela combina a descrição formal e informal e usa destaques gráficos na apresentação das especificações.

A estrutura de um esquema em Z

Figura 10.8
Estrutura de um esquema Z.



Modelagem da bomba de insulina

- O esquema em Z para a bomba de insulina declara uma série de variáveis de estado, incluindo:
  - Variáveis de entrada tais como switch? (a chave do dispositivo), InsulinReservoir? (a quantidade atual de insulina no reservatório) e Reading? (a leitura obtida do sensor);
  - Variáveis de saída tais como alarm! (um alarme do sistema), display!, display2! (os *displays* da bomba) e dose! (a dose de insulina a ser fornecida).

Invariante de esquema

- O estado descrito em Z tem uma parte invariante que define as condições que são sempre verdadeiras.
- Para o esquema de bomba de insulina é sempre verdadeiro que
  - A dose deve ser menor ou igual à capacidade do reservatório de insulina;
  - Nenhuma dose única pode ser maior do que 4 unidades de insulina e o total de dose fornecido em um período de tempo não deve exceder 25 unidades de insulina. Essa é uma restrição de segurança;
  - display2! mostra a quantidade de insulina fornecida.

Esquema de bomba de insulina

Figura 10.9
Esquema State para a bomba de insulina.

```
INSULIN_PUMP_STATE
  //Definição de dispositivo de entrada
  switch?: (off, manual, auto)
  ManualDeliveryButton?: ℕ
  Reading?: ℕ
  HardwareTest?: (OK, batterylow, pumpfail, sensorfail, deliveryfail)
  InsulinReservoir?: (present, notpresent)
  Needle?: (present, notpresent)
  clock?: TIME

  //Definição de dispositivo de saída
  alarm! = (on, off)
  display1!: string
  display2!: string
  clock!: TIME
  dose!: ℕ

  //Variáveis de estado usadas para o cálculo da dose
  status: (running, warning, error)
  r0, r1, r2: ℕ
  capacity, insulin_available : ℕ
  max_daily_dose, max_single_dose, minimum_dose: ℕ
  safemin, safemax: ℤ
  CompDose, cumulative_dose: ℕ
```


Invariantes de estado

Figura 10.9
Esquema State para a bomba de insulina.

```
r2 = Reading?
dose! ≤ insulin_available
insulin_available ≤ capacity

// A dose cumulativa de insulina fornecida zera uma vez a cada 24 horas
clock? = 000000 ⇒ cumulative_dose = 0

// Se a dose cumulativa exceder o limite, então a operação será suspensa
 cumulative_dose ≥ max_daily_dose ∧ status = error ⇒
     display1! = "Daily dose exceeded"

// Parâmetros de configuração da bomba
capacity = 100 ∧ safemin = 6 ∧ safemax = 14
max_daily_dose = 25 ∧ max_single_dos        ∧          _dose = 1

display2! = nat_to_string (dose!)
clock! = clock?
```

O cálculo da dose

- A bomba de insulina calcula a quantidade de insulina requerida pela comparação da leitura atual com as duas leituras anteriores.
- Se ela sugerem que a glicose do sangue está aumentando, então a insulina é fornecida.
- As informações sobre a dose total fornecida são mantidas para permitir que a invariante de verificação de segurança seja aplicada.
- Note que esta invariante sempre se aplica – não há necessidade de repetí-la no cálculo da dose.

Esquema RUN (1)

Figura 10.10 Esquema RUN.

RUN

INSULIN_PUMP_STATE

switch? = auto _
status = running ∨ status = warning
insulin_available ≥ max_single_dose
cumulative_dose < max_daily_dose
// A dose de insulina é calculada dependendo do nível de açúcar no sangue
(SUGAR_LOW! ∨ SUGAR_OK ∨ SUGAR_HIGH!)
// 1. Se a dose de insulina calculada for zero, não fornecer insulina
CompDose = 0 ⇒ dose! = 0
∨
// 2. A dose máxima diária seria excedida se a dose calculada fosse fornecida, de modo que a dose de insulina é estabelecida pela diferença entre a dose máxima diária permitida e a dose cumulativa fornecida até o momento presente
CompDose + cumulative_dose > max_daily_dose ⇒ alarm! = on ∧ status' = warning ∧ dose! = max_daily_dose–cumulative_dose
∨

Esquema RUN (2)

Figura 10.10 Esquema RUN.

// 3. Situação normal. Se a dose única máxima não for excedida, então fornecer a dose calculada. Se a dose única calculada for muito alta, restringir a dose fornecida para a dose única máxima
CompDose + cumulative_dose < max_daily_dose ⇒
(CompDose ≤ max_single_dose ⇒ dose! = CompDose
∨
CompDose > max_single_dose ⇒ dose! = max_single_dose)
insulin_available' = insulin_available–dose!
cumulative_dose' = cumulative_dose + dose!

insulin_available ≤ max_single_dose * 4 ⇒ status' = warning ∧
display1! = "Insulin low"

r1' = r2
r0' = r1

Esquema SUGAR_OK

Figura 10.11 O esquema SUGAR_OK.

SUGAR_OK

r2 ≥ safemin ∨ r2 ∧ safemax
// nível de açúcar estável ou decrescente
r2 ≤ r1 ⇒ CompDose = 0
∨
// nível de açúcar crescente, mas com taxa de aumento decrescente
r2 > r1 ∧ (r2-r1) < (r1-r0) ⇒ CompDose = 0
∨
// nível de açúcar crescente e taxa de aumento crescente, calcular dose
// uma dose mínima deve ser fornecida se próximo de zero
r2 > r1 ∧ (r2-r1) ≥ (r1-r0) ∧ (round ((r2-r1)/4) = 0) ⇒
CompDose = minimum_dose
∨
r2 > r1 ∧ (r2-r1) ≥ (r1-r0) ∧ (round ((r2-r1)/4) > 0) ⇒
CompDose = round ((r2-r1)/4)