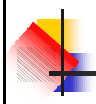
Modelação rigorosa e especificação formal de software em VDM++ e UML: apresentação de um caso de estudo

João Pascoal Faria
(jpf@fe.up.pt)
Faculdade de Engenharia da Universidade do Porto

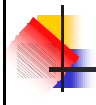
Jornadas de Engenharia Informática 2003
Instituto Politécnico da Guarda
28 de Maio de 2003

Índice



Objectivos e motivação (1)

- Consensual e prática cada vez mais corrente:
  - modelação visual com diagramas UML durante as fases de análise e especificação de requisitos, desenho de alto nível (arquitectura) e desenho detalhado
  - geração de algum código (Java, C#, SQL, XSD, etc.) a partir de UML e vice-versa
- Passos seguintes:
  - modelos como especificações completas e rigorosas de alto nível
    - especificar restrições **formalmente** (com invariantes)
    - especificar **formalmente** a semântica das operações (com pré e pós condições) e, possivelmente, casos de utilização
    - importante para aumentar o rigor (complementando especificações informais)
    - uma especificação pretende ser declarativa (diz os objectivos) e não necessariamente executável (não pretende dizer o caminho)
    - OCL e VDM++ são opções possíveis



Objectivos e motivação (2)

- Passos seguintes (cont.):
  - modelos executáveis de alto nível
    - necessário especificar acções e algoritmos (resposta a eventos, corpo de operações, ...), em linguagem de muito alto nível
    - importante para validar os requisitos o mais cedo possível
    - Executable UML e VDM++ são opções possíveis
  - geração automática de código completo (e não só esqueletos de classes) a partir de modelos de alto nível
    - Executable UML e VDM++ são opções possíveis
- VDM++ mais maduro e parcialmente integrado com UML, mas combinação de OCL e UML executável poderá ganhar
- Objectivo desta apresentação: mostrar que os passos seguintes estão próximos

Índice

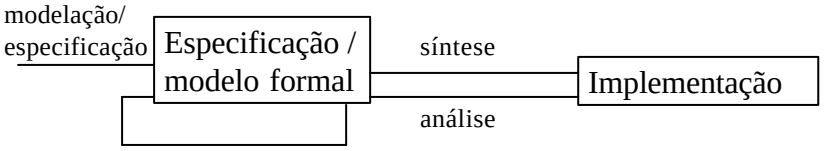
- Objectivos e motivação
- Breve introdução a UML (ver UML overview.pdf)
- Breve introdução aos métodos formais
- Visão geral da linguagem de especificação formal VDM++
- Visão geral da ferramenta IFAD VDMTools®
- Apresentação do caso de estudo (agenda corporativa)
- Conclusões

Índice

- Objectivos e motivação
- Breve introdução a UML
- Breve introdução aos métodos formais
- Visão geral da linguagem de especificação formal VDM++
- Visão geral da ferramenta IFAD VDMTools®
- Apresentação do caso de estudo (agenda corporativa)
- Conclusões

O que são métodos formais ?

- Método Formal =
  Linguagem de Especificação Formal
  + Raciocínio Formal (verificação formal, ...)
- As técnicas são suportadas por
  - Matemática precisa
  - Ferramentas de análise poderosas
- Constituem um mecanismo rigoroso e efectivo para modelação, síntese e análise de sistemas



Classificação de métodos formais

- Baseados em Modelos
  - Definem o comportamento do sistema através da construção de um modelo usando estruturas matemáticas (conjuntos, sequências, funções finitas, etc.) - VDM, Z
  - Particularmente apropriados para desenvolvimento de sistemas de informação
- Baseados em Propriedades
  - Modelam tipos de dados implicitamente definindo o comportamento do sistema através de um conjunto de propriedades
  - Algébricos – definem operações através de uma colecção de relações de equivalência (axiomas equacionais) - Obj, Larch, ...
  - Axiomáticos – definem operações através de predicados da lógica de primeira ordem - Larch, ...
- Baseados no Comportamento
  - Definem sistemas em termos de sequências possíveis de estados em vez de tipos de dados e são normalmente usados para especificar sistemas concorrentes e distribuídos.
  - Exemplos de formalismos: redes de Petri, Calculus of Communicating Systems (CCS), Communicating Sequencial Processes (CSP); lógica temporal; sistemas de transição; álgebras de processos

Exemplo em VDM++ (baseada em modelos)

É necessário escolher uma representação do estado de uma stack!

Representação de dados (estado) baseada em construções matemáticas (**conjuntos**, **sequências**, **funções finitas**, etc.)!

Se fosse caso disso, podiam-se especificar **invariantes**, i.e., condições a que têm de obedecer os estados válidos.

Corpo algorítmico

Pré-condição é uma condição (nos argumentos de chamada e estado do objecto) a que tem de obedecer qualquer chamada válida.

Pós-condição é uma condição que relaciona o resultado da operação e o estado final do objecto com os argumentos de chamada e o estado inicial do objecto.

Pré e pós-condição especificam **semântica** da operação!

```
class Stack
 instance variables
  elems : seq of int := [];
operations
  Stack () ==> ()
    Stack () == elems := []
     post elems  =  [];
  Push : int ==> ()
    Push(i) == elems := [i] ^ elems
     post stack  = [i]  ^  ~elems;
  Pop : () ==> ()
    Pop() == elems := tl elems
     pre elems <> []
    post elems =  tl ~elems;
  Top : () ==> int
    Top() ==  return (hd elems)
     pre elems <> []
     post RESULT = hd elems
          and elems = ~elems;
end Stack
```


Exemplo em OBJ (algébrica)

Spec: Stack;
Extend Nat **by**
Sorts: Stack;
Operations:
  newstack: → Stack
  push: Stack × Nat → Stack
  pop: Stack → Stack
  top: Stack → Nat
Variables:
  s: Stack; n: Nat
Axioms:
  pop(newstack) = newstack;
  top(newstack) = zero;
  pop(push(s,n)) = s;
  top(push(s,n)) = n;

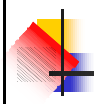
Mais **abstracta**: especifica semântica de operações por axiomas, sem necessidade de escolher uma representação de dados interna (cf. noção de ADT)!

Uma stack é sempre representada por uma expressão de construção, e.g.,

push(3, push(2, push(1, newstack)))

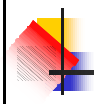
Axiomas permitem simplificar/avaliar expressões

top(pop(push(2, push(1, newstack)))) =
= top(push(1, newstack))
= 1



Índice

- Objectivos e motivação
- Breve introdução a UML
- Breve introdução aos métodos formais
- Visão geral da linguagem de especificação formal VDM++ (ver VDM-SL & VDM++ overview.pdf)
- Visão geral da ferramenta IFAD VDMTools®
- Apresentação do caso de estudo (agenda corporativa)
- Conclusões



Índice

- Objectivos e motivação
- Breve introdução a UML
- Breve introdução aos métodos formais
- Visão geral da linguagem de especificação formal VDM++
- Visão geral da ferramenta IFAD VDMTools® (ver VDMTools overview.pdf)
- Apresentação do caso de estudo (agenda corporativa)
- Conclusões

Índice

- Objectivos e motivação
- Breve introdução a UML
- Breve introdução aos métodos formais
- Visão geral da linguagem de especificação formal VDM++
- Visão geral da ferramenta IFAD VDMTools®
- Apresentação do caso de estudo (agenda corporativa)
- Conclusões

Requisitos iniciais (1)

- O sistema deve permitir gerir a marcação de compromissos envolvendo múltiplos recursos numa organização.
- Os recursos são as pessoas, espaços e equipamentos.
- Cada recurso tem uma agenda associada.
- Cada agenda tem um dono, que é uma pessoa da organização. No caso da agenda de uma pessoa, o dono da agenda é a própria pessoa.
- Uma agenda (*datebook*) tem elementos de dois tipos: compromissos (*appointments*) e disponibilidades (*slots*).
- Um compromisso é de um certo tipo (reunião, aula, etc.), envolve um ou mais recursos e ocupa um intervalo de tempo ou um conjunto de intervalos de tempo, com resolução ao minuto.
- Deve ficar registado quem marcou um compromisso (autor do compromisso).
- Um recurso não pode ter dois compromissos ao mesmo tempo.

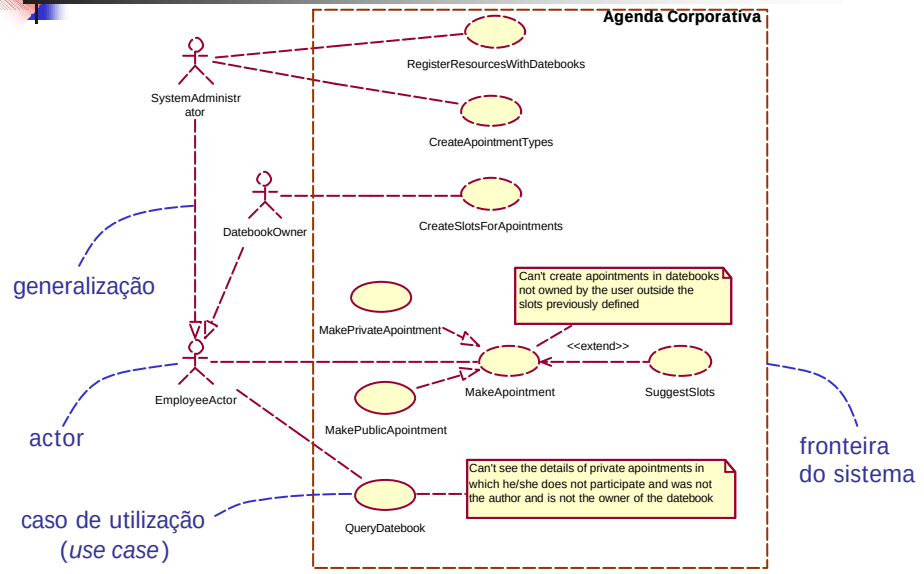
Requisitos iniciais (2)

- Uma disponibilidade refere-se a um recurso, um tipo de compromisso e um intervalo de tempo ou um conjunto de intervalos de tempo.
- Uma pessoa só pode marcar um compromisso numa agenda de que não é dona, dentro das disponibilidades definidas pelo dono da agenda. Só o dono da agenda pode marcar compromissos sem verificação de disponibilidades.
- As agendas podem ser consultadas por todas as pessoas da organização.
- Os compromissos podem ser públicos ou privados. Um compromisso privado só pode ser consultado pelo autor do compromisso, pelas pessoas envolvidas no compromisso e pelos donos das agendas dos recursos envolvidos.

Requisitos iniciais (3)

- O sistema deve ajudar o utilizador a marcar compromissos da seguinte forma: o utilizador indica o tipo de compromisso a marcar (exemplo: reunião), os recursos envolvidos, a duração do compromisso e restrições temporais para a marcação do compromisso; o sistema deve fornecer uma lista de hipóteses de marcação; cada hipótese mostra um intervalo de tempo dentro do qual é possível marcar o compromisso em todos os recursos (porque têm declarada disponibilidade para o tipo de compromisso especificado e não têm marcados compromissos de qualquer tipo).
- Compete ao administrador do sistema registar os recursos da organização (criando automaticamente as respectivas agendas) e definir os tipos de compromissos possíveis.

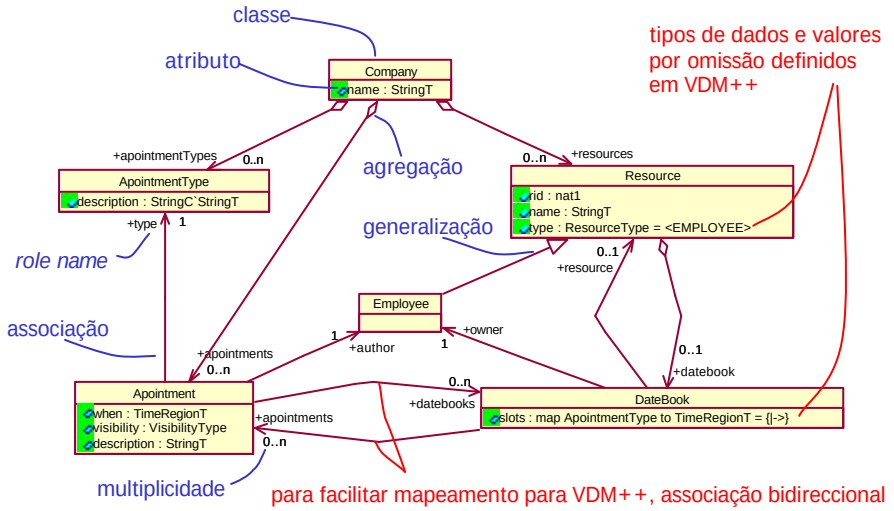
Modelo de casos de utilização em UML



Descrição de actores e casos de utilização

- Actores
  - EmployeeActor - empregado da organização
  - DatebookOwner - empregado da organização que interage com o sistema como dono de uma agenda
  - (...)
- Casos de utilização
  - MakeApointment - registar um compromisso público ou privado de um certo tipo envolvendo um ou mais recursos e ocupando um intervalo de tempo ou conjunto de intervalos de tempo
    - Não se pode registar um compromisso na agenda de um recurso fora dos "slots" previamente definidos, a não ser que o empregado que está a marcar o compromisso seja o dono da agenda.
  - (...)

Modelo de classes de domínio em UML (1)

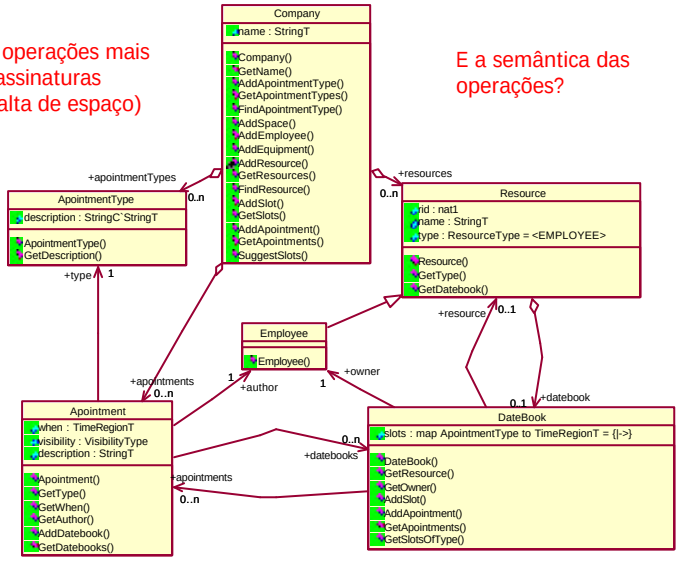


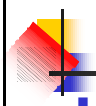
para facilitar mapeamento para VDM++, associação bidireccional é representada por duas associações unidireccionais

Modelo de classes de domínio em UML (2)

Agora com as operações mais importantes (assinaturas omitidas por falta de espaço)

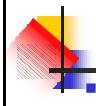
E a semântica das operações?





Definição de tipos de dados em VDM++ (1)

- Em VDM++, tipos e classes são conceitos diferentes
  - Instâncias de um tipo são valores (tipos são tipos-valor)
  - Instâncias de uma classe são objectos (classes são tipos-referência)
  - Tipos são usados para representar tipos de valores de atributos (tipos de dados)
  - Classes são usadas para representar o estado do sistema
- Tipos de dados a definir neste caso:
  - **`StringT`** – cadeia de um ou mais caracteres
  - **`Date`** – data representada pelo número de dias decorridos desde uma data da referência (1/1/1900), útil para processamento interno
  - **`UserDate`** – data representada por uma combinação de dia (1 a 31), mês (1 a 12) e ano (>=1900)
  - **`Time`** – instante de tempo, representado pelo número de minutos decorridos desde as 0 horas da data de referência
  - **`UserTime`** – instante de tempo, representado por uma combinação de minutos (0 a 59), horas (0 a 23), dia, mês e ano
  - **`TimeInterval`** – intervalo entre dois instantes de tempo, fechado à esquerda e aberto à direita, representado por 2 valores do tipo `Time`



Definição de tipos de dados em VDM++ (2)

- Tipos de dados a definir neste caso (cont.):
  - **`Duration`** – uma duração em minutos
  - **`UserTimeInterval`** – intervalo entre dois instantes de tempo, fechado à esquerda e aberto à direita, representado por um par de valores do tipo `UserTime`
  - **`TimeRegion`** – região de tempo (conjunto de intervalos de tempo) representada por um conjunto de intervalos de tempo unitários (com 1 min de duração) representados pelos seus inícios (valores do tipo `Time`)
    - representação abstracta pouco eficiente, mas que permite efectuar operações com regiões de tempo usando directamente os operadores de conjuntos (reunião, intersecção e diferença), simplificando a especificação
  - **`ResourceType`** – tipo enumerado que admite os valores `<EMPLOYEE>`, `<EQUIPMENT>` ou `<SPACE>`
  - **`VisibilityType`** – tipo enumerado que admite os valores `<PUBLIC>` ou `<PRIVATE>`
- Em VDM++, os tipos têm de ser definidos dentro de classes (as classes funcionam aqui como meros espaços de nomes)
- Normalmente, estes tipos já estariam definidos numa biblioteca

Exemplo de definição de tipos em VDM++ (1)

```
class DateC -- classe que alberga definições de tipos
 types
  -- a date represented in a user friendly structure
  public UserDate :: year : nat1
                     month: nat1
                     day  : nat1
        inv d == d.month >= 1 and d.month <= 12
                 and d.year >= Year0
                 and d.day >= 1
                 and d.day <= DaysOfMonth(d.year,d.month);

  -- date representated as number of elapsed days since UserDate0
  -- good for internal procesing
  public Date = nat;

 functions
  public static UserDateToDate(d: UserDate) res : Date ==
    d.day-1+DaysBeforeMonth(d.year,d.month)+DaysBeforeYear(d.year);

  public static DateToUserDate(d: Date) res : UserDate ==
    IncrementDate(UserDate0, d);
(...)
```

definição de tipo de *record*

invariante

Exemplo de definição de tipos em VDM++ (2)

```
(...)
 values
  private Year0 : nat = 1900;
  private UserDate0 : UserDate = mk_UserDate(Year0,1,1);

 functions
  private static IsLeapYear(year: nat) res : bool ==
    year mod 4 = 0 and year mod 100 <> 0 or year mod 400 = 0;

  private static DaysOfMonth(year, month: nat) res : nat ==
    (cases month :
      1, 3, 5, 7, 8, 10, 12 -> 31,
      4, 6, 9, 11 -> 30,
      2 -> if IsLeapYear(year) then 29 else 28
     end)
  pre month >= 1 and month <= 12;

(...omitidas outras funções privadas auxiliares ...)
end DateC
```

constantes

notação para construção de *record*

pré-condição

Definição de classes em VDM++

- Em VDM++, as classes são usadas para modelar o estado do sistema e as operações de consulta e alteração de estado
- É possível gerar o esqueleto das classes em VDM++ a partir de um diagrama de classes UML (usando as ferramentas Rational Rose e VDMTools), e vice-versa
- VDM++ permite completar a especificação das classes com invariantes de estado, pré e pós-condições das operações e corpo algorítmico das operações

Exemplo de definição de classe em VDM++ (1)

```
class Company
 types
  -- short names for types used
  public TimeDurationT = TimeC`TimeDuration;
  public TimeRegionT = TimeRegionC`TimeRegion;
  public StringT = StringC`StringT;
  public ResourceTypeT = Resource`ResourceType;
  public VisibilityTypeT = Apointment`VisibilityType;
  public UserTimeIntervalT = TimeIntervalC`UserTimeInterval;

 instance variables
  public name : StringT;
  public apointmentTypes : set of ApointmentType := {};
   -- Must be initialized because of invariant
  public resources : set of Resource := {};
   -- Must be initialized because of invariant
   -- Because there is a many to many relationship between datebooks
   -- and apointments, they "belong" to the company and are referenced
   -- by datebooks (and vice versa)
  public apointments : set of Apointment := {};
(...)
```

só para facilitar testes

Exemplo de definição de classe em VDM++ (2)

```
(...)
  -- KEY CONSTRAINTS

  -- uniqueness of Resource.rid
  inv forall r1, r2 in set resources & r1<>r2 => r1.rid<>r2.rid;
      -- LIMITATION: not checked if r.rid is changed

(...)
  -- REFERENTIAL INTEGRITY CONSTRAINTS
  -- referential integrity of Apointment.type
  inv forall a in set apointments &
        a.type in set apointmentTypes;
      -- LIMITATION: not checked if a.type is changed

   -- GENERIC CONSTRAINTS
(...)
```


Exemplo de definição de classe em VDM++ (3)

```
(...)
 operations
 -- adds an apointment type
  public AddApointmentType(descr: StringT) res : ApointmentType ==
  (
    dcl t: ApointmentType := new ApointmentType(descr);
    apointmentTypes := apointmentTypes union {t};
    return t
  )
  ext wr apointmentTypes
  pre descr not in set {at.description | at in set apointmentTypes}
  post apointmentTypes = apointmentTypes~ union {res}
       and res.description = descr;
(...)
```

variável de instância que a operação pode alterar

"~" designa valor antigo da variável

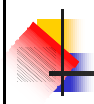
pré-condição implicada por invariante
(pode-se efectuar verificação formal)

Exemplo de definição de classe em VDM++ (4)

```
(...)
 public SuggestSlots(type: ApointmentType, rs: set of Resource,
                     duration: TimeDurationT, within: TimeRegionT)
            res: set of UserTimeIntervalT ==
 return
  { TimeIntervalC`TimeIntervalToUserTimeInterval(ti)
    | ti in set TimeRegionC`TimeRegionToTimeIntervals
      (
        dinter
        { let r_slots = r.GetDatebook().GetSlotsOfType(type),
              r_apoints = dunion {a.when | a in set
                                  r.GetDatebook().GetApointments()}
          in (within inter r_slots \ r_apoints)
          | r in set rs
        }
      )
      & TimeIntervalC`Duration(ti) >= duration
  };
end Company
```

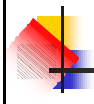

Teste da especificação

- Especificação pode ser testada interactivamente (com interpretador de VDM++) ou com base em casos de teste pré-definidos
- Cada caso de teste *tc1* é especificado por:
  - ficheiro *tc1.arg* – com o comando a executar pelo interpretador
  - ficheiro *tc1.arg.exp* – com o resultado esperado da execução do comando
- É produzida informação dos testes que sucederam e dos testes que falharam
- É produzida informação de cobertura dos testes – o *pretty printer* assinala as partes da especificação que foram de facto executadas



Documentação

- Cada classe VDM++ é especificada num ficheiro em RTF, usando estilos pré-definidos para assinalar as partes de VDM++ (cf. programação literária)
- *Pretty printer* formata as partes de VDM++ e gera versão mais bonita
- Pode-se juntar tudo num documento final com *paste link*



Geração de código

- Geração de código completamente funcional em Java e C++
- Facilmente legível no caso de Java
- Código geralmente pouco eficiente tanto em Java como em C++
- Principal dificuldade tem a ver com utilizar bibliotecas pré-existentes da linguagem alvo desde a fase de especificação

Índice

- Objectivos e motivação
- Breve introdução a UML
- Breve introdução aos métodos formais
- Visão geral da linguagem de especificação formal VDM++
- Visão geral da ferramenta IFAD VDMTools®
- Apresentação do caso de estudo (agenda corporativa)
- Conclusões

Conclusões (1)

- Benefícios da modelação visual em UML
  - Diagramas ricos em informação (uma figura vale mais do que 1000 palavras) utilizando notação normalizada
  - Desde modelos totalmente informais até modelos formais
- Benefícios da especificação formal (VDM++)
  - Permite obter rapidamente uma especificação completa (sintaxe + semântica) do sistema a desenvolver a um nível de abstracção elevado
  - O rigor e análise detalhada implicados pela utilização de métodos formais de especificação, ajudam a remover ambiguidades, omissões e erros desde muito cedo
  - Execução e teste das especificações (de preferência associada com prototipagem rápida da interface com o utilizador), permite validar requisitos muito cedo
  - Suporte para verificação formal de propriedades (por demonstração)
  - Suporte para geração automática de código

Conclusões (2)

- Benefícios da integração de VDM++ e UML
- Abordagens alternativas
  - OCL - Object Constraint Language (ver www.uml.org) – especificação declarativa (não executável) de invariantes, pré-condições e pós-condições
  - UML executável (ver www.projtech.com) - especificação procedimental de comportamento em forma executável através de máquinas de estados executáveis, obedecendo a *action semantics* de UML
  - A vantagem de VDM++ sobre OCL + UML executável é que tem estas duas vertentes fortemente integradas e tem uma maturidade superior
  - A vantagem de OCL + UML executável é que estão no mundo UML
- Aplicação dos métodos formais
  - Sobretudo desenvolvimento de sistemas críticos
    - *safety critical, mission critical, business critical*

Mais informação

- www.ifad.dk/ - sobre VDM++ e VDMTools
- www.projtech.com – sobre UML executável
- www.uml.org – sobre UML, OCL, etc.
- DataBookAll.pdf - documentação completa do caso de estudo