Rigorous Modelling of a Date Book Application with VDM++ and UML

Version 3.0

João Carlos Pascoal Faria

Porto, 28 May 2003

Table of Contents

1 Introduction 3
2 Requirements: Use Case Model 4
2.1 Use case diagram 4
2.2 Actors 5
2.3 Use Cases 5
3 Data Types 7
3.1 Type StringC`StringT 7
3.2 Types DateC`Date and DateC`UserDate 7
3.3 Types TimeC`Time, TimeC`UserTime and TimeC`TimeDuration 9
3.4 Types TimeIntervalC`TimeInterval and TimeIntervalC`UserTimeInterval 11
3.5 Type TimeRegionC`TimeRegion 12
4 Business Classes 15
4.1 Class diagram 15
4.2 Class Company 16
4.3 Class Resource 22
4.4 Class Employee 23
4.5 Class DateBook 24
4.6 Class ApointmentType 26
4.7 Class Appointment 26
5 Testing the specifications 29
5.1 Test classes 29
5.2 Test cases 30
5.3 Test procedures 31
5.4 Test results 32
6 Code generation 33
6.1 C++ code generation 33
6.2 Java code generation 36
6.3 Performance comparison 37
6.4 Readability comparison 37
7 References 38
Appendix A - Major VDM++ Language Characteristics 39
Appendix B - Features of VDMTools® 40

1 Introduction

This document describes a case study of applying VDM++ and UML.
The document includes source files (using the link features of Microsoft Word).

2 Requirements: Use Case Model

The functional requirements of the date book application are modeled in UML by means of use cases and actors.

2.1 Use case diagram

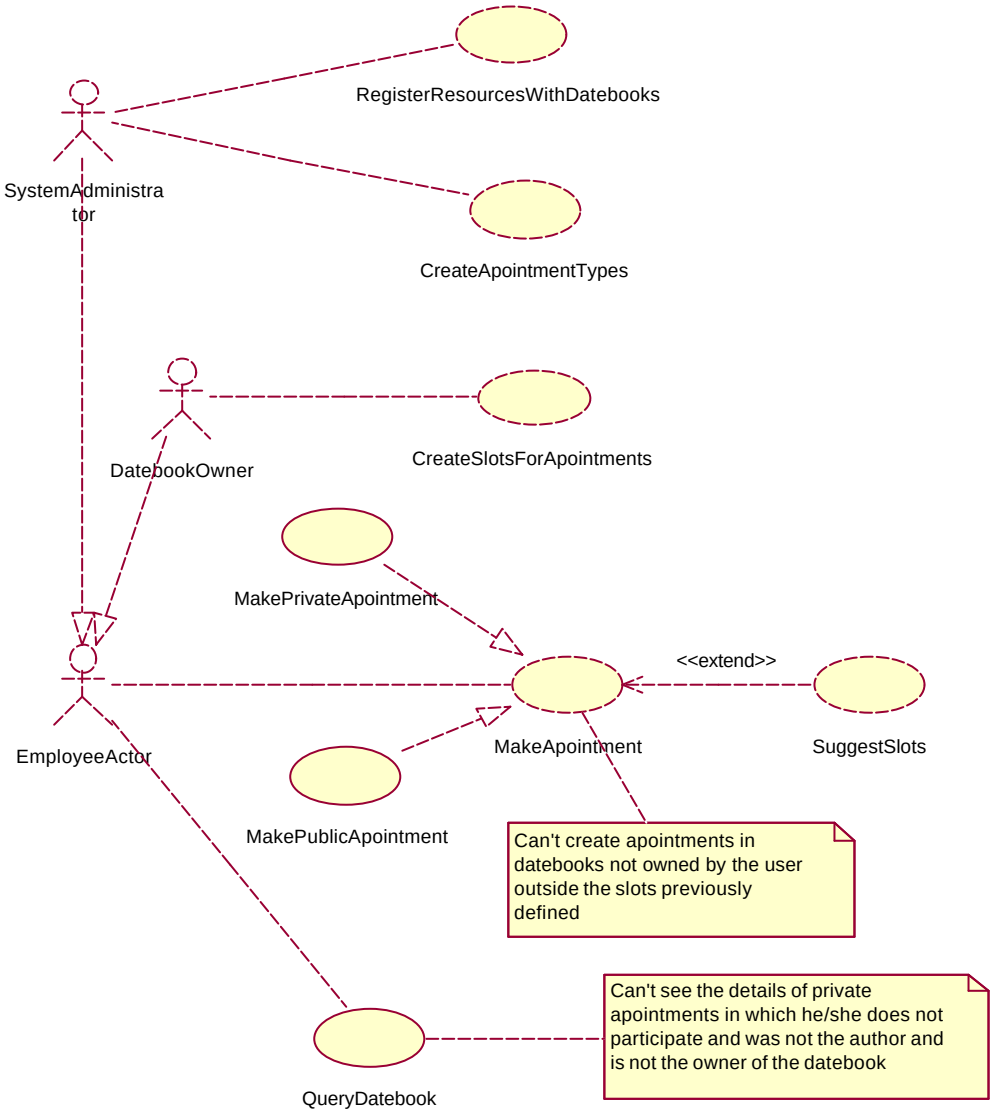

Fig. 1 - Use case diagram of the date book application.

2.2 Actors

2.2.1 EmployeeActor

Um empregado da organização.

2.2.2 SystemAdministrator

Um empregado da organização que interage com o sistema como responsável pela administração do sistema.

2.2.3 DatebookOwner

Um empregado da organização que interage com o sistema como dono de uma agenda.

2.3 Use Cases

2.3.1 RegisterResourcesWithDatebooks

Registar recursos da organização (empregados, equipamentos, salas, etc.), associando automaticamente uma agenda a cada recurso, e especificar o dono de cada agenda.
O dono de uma agenda é sempre um empregado da organização.
O dono da agenda de um empregado é o próprio empregado.

2.3.2 CreateApointmentTypes

Criar tipos de compromissos tais como reunião, aula, etc.

2.3.3 CreateSlotsForApointments

Criar numa agenda de que é dono "slots" (intervalos de tempo, possivelmente repetidos periodicamente) para certos tipos de compromissos. Qualquer empregado da organização poderá marcar compromissos nesses "slots".

2.3.4 MakeApointment

Registar um compromisso público ou privado de um certo tipo envolvendo um ou mais recursos.
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 dono da agenda.

2.3.5 SuggestSlots

Com base nos "slots" previamente definidos nas várias agendas, o sistema sugere intervalos de tempo em que é possível marcar um compromisso, dada a lista de recursos participantes, o tipo de compromisso e restrições temporais.

2.3.6 MakePrivateApointment

Registar um compromisso privado de um certo tipo envolvendo um ou mais recursos.
Os detalhes de um compromisso privado só são visíveis para os donos das agendas desses recursos e para o empregado que marcou o compromisso.

2.3.7 MakePublicApointment

Registar um compromisso público de um certo tipo envolvendo um ou mais recursos.
Os detalhes de um compromisso público são visíveis por todos os empregados da organização.

2.3.8 QueryDatebook

Consultar livremente a agenda de um recurso.
Um empregado não pode ver os detalhes (recursos participantes e tipo de compromisso) de um compromisso privado se não é dono da agenda de um recurso participante (o que implica que o empregado também não participa no compromisso) nem foi ele que marcou o compromisso.

3 Data Types

In the following sections several data types are specified in VDM++.

Remember that, in VDM++, data types and associated values and functions must be defined as members of classes. Classes and types are different concepts. While instances of a class are objects with mutable state (time varyying values of instance variables), instances of a type are pure values (immutable).

3.1 Type StringC`StringT

```
class StringC

types
  public StringT = seq1 of char

-- String entrava em conflito com Java
end StringC
```

3.1.1 Test Coverage info

name	*#calls*	*coverage*
total		***undef***

3.2 Types DateC`Date and DateC`UserDate

3.2.1 Public members

```
class DateC

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);
```

3.2.2 Private members

```
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
  DaysOfYear(year: nat) res : nat ==
    if IsLeapYear(year) then 366 else 365;

  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;

  private static
  DayOfYear(year, month, day: nat) res : nat ==
    DaysBeforeMonth(year,month) + day;

  private static
  DaysBeforeMonth(year, month: nat) res : nat ==
    if month <= 1
    then 0
    else DaysOfMonth(year, month-1) + DaysBeforeMonth(year, month-1)
  pre year >= Year0 and month >= 1 and month <= 12;

  private static
  DaysBeforeYear(year: nat) res : nat ==
    if year <= Year0
```

```
    then 0
    else DaysOfYear(year-1) + DaysBeforeYear(year-1)
  pre year >= Year0;

  private static
  IncrementDate(d: UserDate, ndays: nat) res : UserDate ==
    let resto1 = DaysOfYear(d.year) - DayOfYear(d.year,d.month,d.day)
    in if ndays > resto1
       then IncrementDate(mk_UserDate(d.year+1,1,1), ndays-resto1-1)
       else let resto2 = DaysOfMonth(d.year,d.month) - d.day
            in if ndays > resto2
               then IncrementDate(mk_UserDate(
                       if d.month=12 then d.year+1 else d.year,
                       if d.month=12 then 1 else d.month+1, 1),
                                  ndays-resto2+1)
               else mk_UserDate(d.year,d.month,d.day+ndays);

end DateC
```

3.2.3 Test Coverage info

name	*#calls*	*coverage*
DateC`DayOfYear	408	100%
DateC`DaysOfYear	1216	100%
DateC`IsLeapYear	1216	100%
DateC`DaysOfMonth	1248	40%
DateC`IncrementDate	408	64%
DateC`DateToUserDate	4	100%
DateC`DaysBeforeYear	816	100%
DateC`UserDateToDate	8	100%
DateC`DaysBeforeMonth	416	55%
total		***71%***

3.3 Types TimeC`Time, TimeC`UserTime and TimeC`TimeDuration

```
class TimeC
```

3.3.1 Public members

```
types
  -- short names
  public DateT = DateC`Date;
  public UserDateT = DateC`UserDate;
```

```
  -- a point in time in some scale (minutes) since start of date origin
  public
  Time = nat;

  -- a point in time in a user friendly representation
  public
  UserTime :: year : nat1
              month: nat1
              day  : nat1
              hour : nat
              min  : nat
  inv t == t.month >= 1 and t.month <= 12
           and t.day >= 1 and t.day <= 31
           and t.hour >= 0 and t.day <= 23
           and t.min >= 0 and t.min <= 59;
  -- to be completed with better day and year checks

  -- the length between two points in time
  public
  TimeDuration = nat;


functions

  public static
  UserTimeToTime(t: UserTime) res : Time ==
     t.min + 60*t.hour + 60*24*
         DateC`UserDateToDate(mk_DateC`UserDate(t.year,t.month,t.day));

  public static
  TimeToUserTime(t: Time) res : UserTime ==
-- 1440 = 24 * 60; BUG: se colocar a expressão dá erro em C++!
    let d = DateC`DateToUserDate(t div (1440))
    in mk_UserTime(d.year,d.month,d.day,(t div 60) mod 24,t mod 60);

  public static
  mkTimeDuration(days: nat, hours: nat, mins: nat)
      res : TimeDuration ==
         mins + (hours + days * 24 ) * 60;


end TimeC
```

3.3.2 Test Coverage info

name	*#calls*	*coverage*
TimeC`TimeToUserTime	4	100%
TimeC`UserTimeToTime	8	100%
TimeC`mkTimeDuration	2	100%
total		***100%***

3.4 Types TimeIntervalC`TimeInterval and TimeIntervalC`UserTimeInterval

```
class TimeIntervalC

types
  -- a non empty interval between two points in time
  -- (left closed and right opened, i.e., [start, stop[,
  --  for mathematical correctness)
  public
  TimeInterval :: Start: TimeC`Time
                  Stop : TimeC`Time
  inv ti == ti.Stop > ti.Start;

  public
  UserTimeInterval :: Start: TimeC`UserTime
                      Stop : TimeC`UserTime
  inv t == less([t.Start.year, t.Start.month, t.Start.day,
                 t.Start.hour, t.Start.min],
                [t.Stop.year, t.Stop.month, t.Stop.day,
                 t.Stop.hour, t.Stop.min]);

functions
  public static
  Duration(ti: TimeInterval) res : TimeC`TimeDuration ==
    ti.Stop - ti.Start;

  public static
  TimeIntervalToUserTimeInterval(ti: TimeInterval)
    res : UserTimeInterval ==
      mk_UserTimeInterval( TimeC`TimeToUserTime(ti.Start),
                           TimeC`TimeToUserTime(ti.Stop) );
```

3.4.1 Private members

```
functions
  private static
  less(l1 : seq1 of nat, l2 : seq1 of nat) res : bool ==
    if hd l1 < hd l2
    then true
    else if hd l1 > hd l2
         then false
         else if tl l1 = []
              then true
              else less(tl l1, tl l2)
  pre len l1 = len l2;

end TimeIntervalC
```

3.4.2 Test Coverage info

name	*#calls*	*coverage*
TimeIntervalC`less	32	93%
TimeIntervalC`Duration	2	100%
TimeIntervalC`TimeIntervalToUserTimeInterval	2	100%
total		***95%***

3.5 Type TimeRegionC`TimeRegion

3.5.1 Public members

```
class TimeRegionC

types
  -- a set of atomic time intervals [t, t+1[ represented by t alone
  -- (allows the usage of built-in set operators!)
  public
  TimeRegion = set of TimeC`Time;

functions

  public static
  TimeIntervalToTimeRegion(ti: TimeIntervalC`TimeInterval)
           res : TimeRegion ==
    {ti.Start, ..., ti.Stop-1};

  -- converts a time region from internal representation to a set of
  -- maximal disjoint (and non adjacent) time intervals
  public static
  TimeRegionToTimeIntervals(atr: TimeRegion)
        res : set of TimeIntervalC`TimeInterval ==
    if atr = {}
    then {}
    else let ti = FirstInterval(atr)
         in {ti} union TimeRegionToTimeIntervals(
                         atr \ TimeIntervalToTimeRegion(ti));

  -- produces a periodic time region based on the repetition of
  -- a "base" time region a certain number of times with a certain
  -- interval or step
  public static
  PeriodicTimeRegion(base:TimeRegion,
                     step: TimeC`TimeDuration,
                     times: nat1) res : TimeRegion ==
    if times = 1
    then base
    else base union
         PeriodicTimeRegion(ShiftTimeRegion(base,step), step, times-1);
```

```
functions
  -- shortcut
  public static
  UserTimesToTimeRegion(Start: TimeC`UserTime, Stop: TimeC`UserTime)
    res : TimeRegion ==
      {TimeC`UserTimeToTime(Start), ...,
       TimeC`UserTimeToTime(Stop)-1};
```

3.5.2 Private members

```
  private static
  FirstInterval(atr: TimeRegion) res : TimeIntervalC`TimeInterval ==
    let t = ApplyMin(atr)
    in mk_TimeIntervalC`TimeInterval(t,FindEndOfInterval(atr,t)+1)
  pre atr <> {};

  private static
  FindEndOfInterval(atr: TimeRegion, t: TimeC`Time) res : TimeC`Time ==
    if t+1 in set atr
    then FindEndOfInterval(atr, t+1)
    else t;

  private static
  ShiftTimeRegion(base:TimeRegion,
                  step: TimeC`TimeDuration) res : TimeRegion ==
    {t + step | t in set base};

  private static
  min(a, b: nat) res : nat ==
    if a < b then a else b;

  private static
  ApplyMin(s: set of nat) res : nat ==
    let e in set s
    in if card s = 1
       then e
       else min(e,ApplyMin(s\{e}))
  pre s <> {};


end TimeRegionC
```

3.5.3 Test Coverage info

name	*#calls*	*coverage*
TimeRegionC`min	158	83%
TimeRegionC`ApplyMin	160	100%
TimeRegionC`FirstInterval	2	100%
TimeRegionC`ShiftTimeRegion	19	100%
TimeRegionC`FindEndOfInterval	110	100%

TimeRegionC`PeriodicTimeRegion	20	100%
TimeRegionC`UserTimesToTimeRegion	4	100%
TimeRegionC`TimeIntervalToTimeRegion	2	100%
TimeRegionC`TimeRegionToTimeIntervals	3	100%
total		***99%***

4 Business Classes

In this section the business classes of the application are specified in VDM++. Business classes are used to model the state of the application.
As an overview, it is also presented an UML class diagram generated from within VDM++ using the Rose Link feature.

4.1 Class diagram

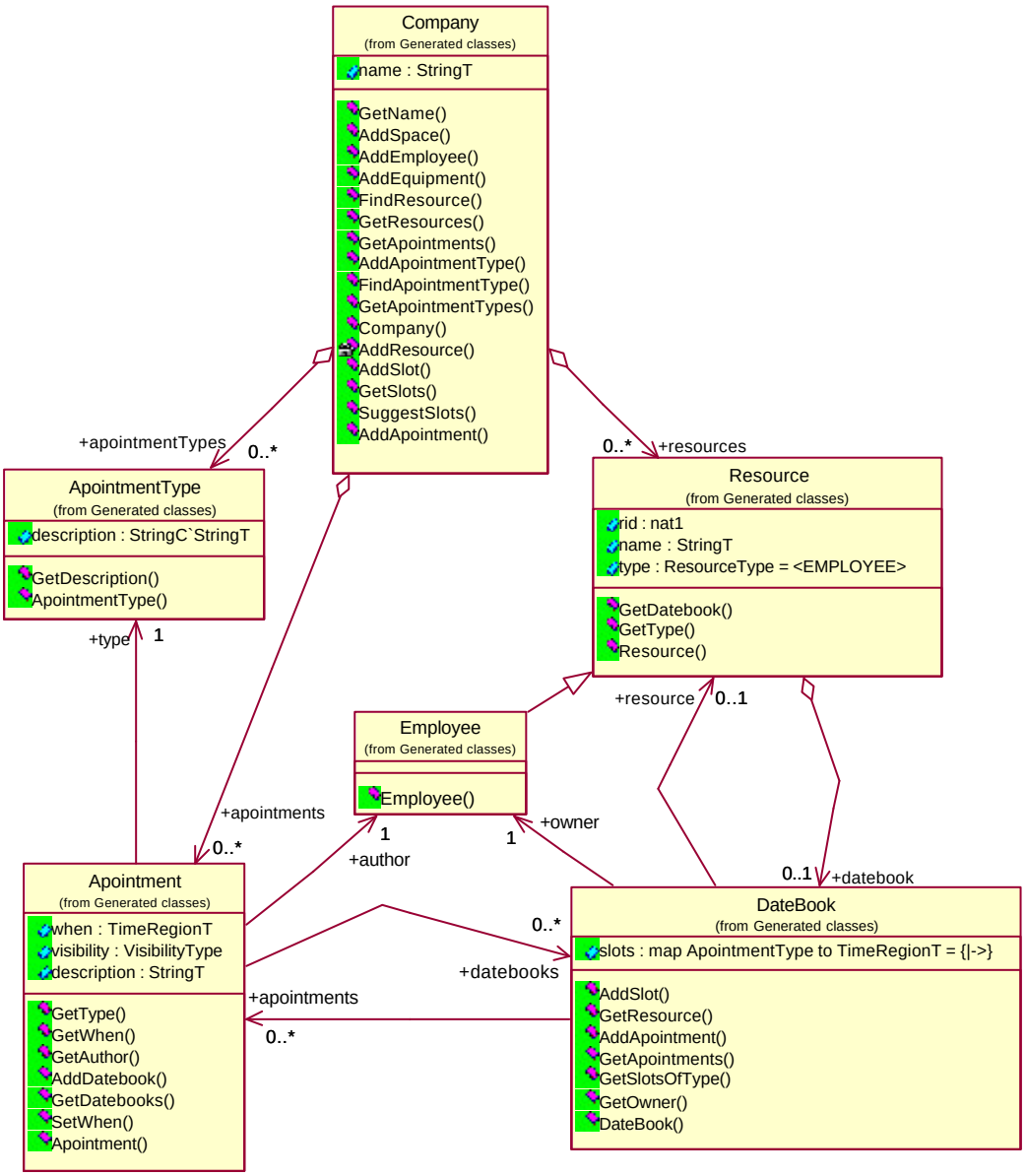

Fig 2 - Class diagram showing business classes in date book application

4.2 Class Company

```
class Company

types
  -- short names
  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 := {};

  -- KEY CONSTRAINTS

  -- uniqueness of ApointmentType.description
  inv forall at1, at2 in set apointmentTypes &
        at1 <> at2 => at1.GetDescription() <> at2.GetDescription();
  -- BUG: doesn't work with at.description
  -- LIMITATION: isn't checked if at.description is changed

  -- uniqueness of Resource.rid
  inv forall r1, r2 in set resources & r1 <> r2 => r1.rid <> r2.rid;
  -- BUG: didn't work with .id, renamed to .rid
  -- LIMITATION: isn't 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

  -- referential integrity of Apointment.author
  inv forall a in set apointments &
        a.author in set resources
        and a.author.GetType() = <EMPLOYEE>;
  -- LIMITATION: not checked if a.author is changed
  -- BUG: doesn't work with a.author.type

  -- referential integrity of Apointment.datebooks
  inv forall a in set apointments &
        forall b in set a.datebooks &
           b.resource <> nil and b.resource in set resources;
  -- LIMITATION: not checked if a.datebooks or b.resource are changed

  -- referential integrity of DateBook.owner
  inv forall r in set resources &
        r.datebook <> nil =>
           let o = r.datebook.GetOwner()
           in o in set resources
              and o.GetType() = <EMPLOYEE>;
  -- BUG: doesn't work if written "o.type"

  -- referential integrity of dom DateBook.slots
  inv forall r in set resources &
        r.datebook <> nil =>
           dom r.datebook.slots subset apointmentTypes;
  -- LIMITATION: not checked if r.datebook.slots changed

  -- OTHER CONSTRAINTS

  -- apointments of the same resource cannot overlap
  -- => checked in Apointment and DateBook

  -- if the author of the apointment is not the owner of the datebook,
  -- the apointment must be made inside a slot for the same type of
  -- apointment in that datebook
  -- => checked in Apointment and DateBook
```

4.2.1 Write operations

```
operations

  -- initializes a new company
  public
  Company(nm: StringT) res : Company ==
  (
    name := nm;
    apointmentTypes := {};
    resources := {};
    return self
  )
  ext wr name : StringT
      wr apointmentTypes : set of ApointmentType
```

```
    wr resources : set of Resource
post name = nm
     and apointmentTypes = {} and resources = {};

-- adds an employee
public
AddEmployee(id: nat1, name: StringT) res : Employee ==
(
    dcl e : Employee := new Employee(id,name);
    resources := resources union {e};
    return e
)
ext wr resources : set of Resource
pre (not exists r in set resources & r.rid = id)
post resources = resources~ union {res}
     and res.type = <EMPLOYEE>
     and res.name = name
     and res.rid = id
     and res.datebook <> nil
     and res.datebook.owner = res
     and res.datebook.resource = res
     and res.datebook.apointments = {}
     and res.datebook.slots = {|->};

-- adds a space (room, etc.)
public
AddSpace(id: nat1, name: StringT, dbowner: Employee) res: Resource ==
    return AddResource(id, name, <SPACE>, dbowner);

-- add an equipment
public
AddEquipment(id: nat1, name: StringT, dbowner: Employee) res: Resource
==
  return AddResource(id, name, <EQUIPMENT>, dbowner);

-- adds a resource
private
AddResource(id     : nat1,
            name   : StringT,
            type   : Resource`ResourceType,
            dbowner: Employee)
               res : Resource ==
(
  dcl r: Resource := new Resource(id,name,type,dbowner);
  resources := resources union {r};
  return r
)
ext wr resources : set of Resource
pre type <> <EMPLOYEE>
    and (not exists r in set resources & r.rid = id)
    and (dbowner in set resources)
post resources = resources~ union {res}
     and res.type = type
     and res.name = name
     and res.rid = id
     and res.datebook <> nil
     and res.datebook.owner = dbowner
```

```
      and res.datebook.resource = res
      and res.datebook.apointments = {}
      and res.datebook.slots = {|->};

  -- 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 : set of ApointmentType
  pre descr not in set {at.description | at in set apointmentTypes}
  post apointmentTypes = apointmentTypes~ union {res}
       and res.description = descr;

  public AddSlot(r: Resource, type: ApointmentType, when:
TimeRegionC`TimeRegion) ==
  (
    r.datebook.AddSlot(type,when)
  )
--  ext wr resources : set of Resource
  pre (type in set apointmentTypes)
      and (r in set resources) and (r.datebook <> nil) ;
--  post if type in set dom r.datebook.slots
--       then r.datebook.slots = r~.datebook.slots ++
--             {type |-> (r~.datebook.slots(type) union when)}
--     else r.datebook.slots = r~.datebook.slots munion {type |-> when};
-- LIMITATION: can't access the old state of referenced objects

  public AddApointment(type       : ApointmentType,
                       when       : TimeRegionC`TimeRegion,
                       rs         : set of Resource,
                       author     : Employee,
                       visibility : Apointment`VisibilityType,
                       description: StringT)
                          res : Apointment ==
   (
     dcl ap : Apointment;
     dcl r : Resource;

     -- create and fill
     ap := new Apointment(type, when, author, visibility,
                                    description);

     apointments := apointments union { ap };

     -- link together to resources
     for all r in set rs do
         r.datebook.AddApointment(ap);

     return ap
   )
  pre (type in set apointmentTypes)
      and (rs subset resources) and (rs <> {})
      and (author in set resources)
```

```
      and (forall r in set rs &
             author <> r.datebook.owner =>
               when subset r.datebook.GetSlotsOfType(type));
```

4.2.2 Read operations

```
  public
  GetName() res : StringT ==
    return name;

  public
  GetResources() res : set of Resource ==
    return resources;

  public
  GetApointmentTypes() res : set of ApointmentType ==
    return apointmentTypes;

  public
  GetApointments(user: Employee, r: Resource)
              res : set of Apointment ==
    if user = r.datebook.owner then
      return r.datebook.apointments
    else
      return {a | a in set r.datebook.apointments
                & a.visibility = <PUBLIC>
                  or (user = a.author)
--  implied       or (user.datebook in set a.datebooks)
                  or (exists b in set a.datebooks & user = b.owner)
             }
  pre (r in set resources) and (user in set resources);

  public
  GetSlots(r: Resource,
           type: ApointmentType,
           within: TimeRegionC`TimeRegion)
        res : TimeRegionC`TimeRegion ==
     return within inter r.datebook.GetSlotsOfType(type)
  pre (r in set resources) and (r.datebook <> nil);

  public FindResource(id : nat1) res : [Resource] ==
    if id in set {r.rid | r in set resources}
    then return (iota r in set resources & r.rid = id)
    else return nil;

  public FindApointmentType(descr : StringT)
         res : [ApointmentType] ==
    if descr in set {at.description | at in set apointmentTypes}
    then return (iota at in set apointmentTypes & at.description = descr)
    else return nil;
```

4.2.3 "Inteligent" operations

Returns a set of maximal and disjoint time intervals that can accommodate an appointment of a given type, with a given duration, involving a given set of resources, within a given time region. A time interval can accommodate the appointment if:

a) for each resource given, the time interval is contained in a slot for the same type of appointment in the resource's date book;
b) for each resource given, there is no appointment of any type in the resource's date book that overlaps with the time interval;
c) the length of the time interval is greater or equal than the duration wanted.

```
  public
  SuggestSlots(type    : ApointmentType,
               rs      : set of Resource,
               duration: TimeC`TimeDuration,
               within  : TimeRegionC`TimeRegion)
                  res  : set of TimeIntervalC`UserTimeInterval ==
    return
      {TimeIntervalC`TimeIntervalToUserTimeInterval(ti)
       | ti in set TimeRegionC`TimeRegionToTimeIntervals
         (
           dinter
           {let r_slots = r.GetDatebook().GetSlotsOfType(type),
                  -- BUG: dava erro run time C++ com r.datebook.slots
                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
```

4.2.4 Test Coverage info

name	*#calls*	*coverage*
Company`AddSlot	2	100%
Company`Company	1	100%
Company`GetName	0	0%
Company`AddSpace	1	100%
Company`GetSlots	0	0%
Company`AddEmployee	2	100%
Company`AddResource	2	100%
Company`AddEquipment	1	100%
Company`FindResource	0	0%
Company`GetResources	0	0%
Company`SuggestSlots	1	100%
Company`AddApointment	1	100%
Company`GetApointments	0	0%

Company`AddApointmentType	2	100%
Company`FindApointmentType	0	0%
Company`GetApointmentTypes	0	0%
total		***75%***

4.3 Class Resource

```
class Resource

types
  public ResourceType = <EMPLOYEE> | <EQUIPMENT> | <SPACE>;

  public TimeRegionT = TimeRegionC`TimeRegion;
  public StringT = StringC`StringT;

instance variables

  public rid : nat1;
  -- BUG: if named "id" instead of "rid" the invariant in Company
  -- is not correctly evaluated

  public name : StringT;

  public type : ResourceType := <EMPLOYEE>;

  public datebook : [DateBook] := nil;
  -- initialization with nil required because of invariant

  -- consistency of bidirectional assotiation Resource <--> DateBook
  inv datebook = nil or datebook.resource = self;

operations
  -- initializes a new resource
  public
  Resource(id1: nat1, name1: StringT, type1: ResourceType,
           dbowner: Employee) res : Resource ==
  (
     rid := id1;
     name := name1;
     type := type1;
     -- must be done by this order because of invariant on datebook
     datebook := new DateBook(self, dbowner);
    return self
  );

  public
  GetDatebook() res: [DateBook] ==
    (return datebook);
```

```
  public
  GetType() res: ResourceType ==
    (return type);

end Resource
```

4.3.1 Test Coverage info

name	*#calls*	*coverage*
Resource`GetType	19	100%
Resource`Resource	2	100%
Resource`GetDatebook	4	100%
total		***100%***

4.4 Class Employee

```
class Employee is subclass of Resource

instance variables
  inv type = <EMPLOYEE>;

operations
  -- initializes a new resource
  public
  Employee(id1: nat1, name1: StringT) res : Resource ==
  (
     rid := id1;
     name := name1;
     type := <EMPLOYEE>;
     -- must be done by this order because of invariant on datebook
     datebook := new DateBook(self, self);
    return self
  );


end Employee
```

4.4.1 Test Coverage info

name	*#calls*	*coverage*
Employee`Employee	2	100%
total		***100%***

4.5 Class DateBook

```
class DateBook

types
  public TimeRegionT = TimeRegionC`TimeRegion;

instance variables
  public resource   : [Resource] := nil;
  public owner      : Employee;
  public slots      : map ApointmentType to TimeRegionT := {|->};
  public apointments: set of Apointment := {};

  -- the owner of an employee's datebook must be himself
  inv resource <> nil and resource.type = <EMPLOYEE> => owner = resource;

  -- if the author of the apointment is not the owner of the datebook,
  -- the apointment must be made inside a slot for the same type of
  -- apointment in that datebook
  inv forall a in set apointments &
        a.author <> owner => a.when subset GetSlotsOfType(a.type);
  -- LIMITATION: the same invariant is repeated in class Apointment,
  -- in case a.when is changed

  -- consistency of bidirectional assotiation DateBook <--> Resource,
  -- (only a part of the consistency, the other part in Resource)
  -- inv resource.datebook = self;
  -- DOESN'T WORK , only in on of the sides (in this case, in Resource
  -- because of the access path Company->Resource->DateBook)

  -- consistency of bidirectional assotiation DateBook <--> Apointment
  inv forall a in set apointments &
        self in set a.datebooks;
  -- this is only a part of the real invariant (changes to
  -- self.apointments), the other part in class Apointment

  -- apointments in the same datebook cannot overlap
  -- inv forall a1, a2 in set apointments &
  --        a2 <> a2 => (a1.when inter a2.when = {});
  -- LIMITATION: moved to class Apointment, because here would not
  -- be checked in case a.when was changed

operations

  -- initializes a new datebook
  public
  DateBook(r: Resource, o: Employee) res : DateBook ==
  (
    -- atomic required because of invariant relating resource and owner
    atomic(
      resource := r;
      owner := o);
    return self
  )
  pre r.type = <EMPLOYEE> => o = r;
```

```
  public
  AddApointment(a: Apointment) ==
  (
    a.AddDatebook(self);
    apointments := apointments union {a};
  )
  ext wr apointments : set of Apointment
  pre a not in set apointments
  post apointments = apointments~ union {a};

  public
  AddSlot(type: ApointmentType, when: TimeRegionT) ==
  (if type in set dom slots
   then slots(type) := slots(type) union when
   else slots(type) := when
  )
  ext wr slots : map ApointmentType to TimeRegionT
  post if type in set dom slots~
       then slots = slots~ ++ {type |-> (slots~(type) union when)}
       else slots = slots~ munion {type |-> when};

  public
  GetResource() res : Resource ==
   return resource;

  public
  GetSlotsOfType(type: ApointmentType) res : TimeRegionT ==
   return
     if type in set dom slots
     then slots(type)
     else {};

  public
  GetApointments() res : set of Apointment ==
   return apointments;

  public
  GetOwner() res : Resource ==
    return owner;

end DateBook
```

4.5.1 Test Coverage info

name	*#calls*	*coverage*
DateBook`AddSlot	2	52%
DateBook`DateBook	4	100%
DateBook`GetOwner	18	100%
DateBook`GetResource	0	0%
DateBook`AddApointment	1	100%
DateBook`GetApointments	2	100%
DateBook`GetSlotsOfType	5	90%
total		***75%***

4.6 Class ApointmentType

```
class ApointmentType

instance variables
  public description : StringC`StringT;

operations
  public ApointmentType(descr : StringC`StringT) res : ApointmentType ==
  (
    description := descr;
    return self
  );

  public GetDescription() res : StringC`StringT ==
    return description;

end ApointmentType
```

4.6.1 Test Coverage info

name	*#calls*	*coverage*
ApointmentType`ApointmentType	2	100%
ApointmentType`GetDescription	28	100%
total		***100%***

4.7 Class Appointment

```
class Apointment

types
  public VisibilityType = <PUBLIC> | <PRIVATE>;

  -- short names
  public TimeRegionT = TimeRegionC`TimeRegion;
  public StringT = StringC`StringT;

instance variables
  public type : ApointmentType;
```

```
  public when : TimeRegionT;
  public author : Employee;
  public visibility : VisibilityType;
  public description: StringT;

  public datebooks: set of DateBook := {};
  -- this is just a reference


  -- if the author of the apointment is not the owner of the datebook,
  -- the apointment must be made inside a slot for the same type of
  -- apointment in that datebook
  inv forall b in set datebooks &
        author <> b.owner => when subset b.GetSlotsOfType(type);
  -- LIMITATION: the same invariant is repeated in class DateBook,
  -- in case b.owner or b.slots are changed

  -- apointments in the same datebook cannot overlap
  inv forall b in set datebooks &
        forall a2 in set b.apointments &
          self <> a2 => (when inter a2.when = {});
  -- LIMITATION: it's here and not in DateBook because it would not
  -- be checked whan an apointment is changed;
  -- in case b.apointments is changed, some a.datebooks will also be
  -- changed, so there is no need for an invariant in class DateBook

  -- consistency of bidirectional assotiation DateBook <--> Apointment
  -- inv forall b in set datebooks &
  --      self in set b.apointments;
  -- this is only a part of the real invariant (changes to
  -- self.datebooks), the other part is in class DateBook
  -- BUG: doesn't work

operations
  public
  Apointment(type1       : ApointmentType,
             when1       : TimeRegionT,
             author1     : Employee,
             visibility1 : VisibilityType,
             description1: StringT)
                         res : Apointment ==
   (
     datebooks := {};
     type := type1;
     when := when1;
     author := author1;
     visibility := visibility1;
     description := description1;
     return self
   );

  public AddDatebook(b: DateBook) ==
    datebooks := datebooks union {b};

  public GetWhen() res : TimeRegionT ==
    return when;
```

```
  public SetWhen(w : TimeRegionT) ==
    when := w;

  public GetType() res : ApointmentType ==
    return type;

  public GetAuthor() res : Employee ==
    return author;

  public GetDatebooks() res : set of DateBook==
    return datebooks;

end Apointment
```

4.7.1 Test Coverage info

name	*#calls*	*coverage*
Apointment`GetType	0	0%
Apointment`GetWhen	0	0%
Apointment`SetWhen	0	0%
Apointment`GetAuthor	0	0%
Apointment`Apointment	1	100%
Apointment`AddDatebook	1	100%
Apointment`GetDatebooks	0	0%
total		***66%***

5 Testing the specifications

5.1 Test classes

Para não ter de estar sempre a escrever comandos de teste complexos no interpretador, foi criada propositadamente uma classe para efeito de testes. Cada método desta classe corresponde a um caso de teste. Segue-se a especificação da classe de teste (já depois de passada pelo "Pretty Print").

5.1.1 Class TestCompany

```
class TestCompany

instance variables

  public c : Company := new Company("HGSA");

operations
  -- BUG: Gives result (600 / 24 * 60) in C++
  public
  Test2() res : int  ==
  ( return 600 div (24*60));

  -- Tests the SuggestSlots operation
  public
  Test1() res : set of TimeIntervalC`UserTimeInterval ==
  (
    dcl t1: ApointmentType := c.AddApointmentType("consulta");
    dcl t2: ApointmentType := c.AddApointmentType("urgencia");
    dcl r1: Employee := c.AddEmployee(1,"Dr. Joao");
    dcl r2: Employee := c.AddEmployee(2,"Dra. Maria");
    dcl r3: Resource := c.AddSpace(3,"Sala 1",r2);
    dcl r4: Resource := c.AddEquipment(5,"Microscopio",r1);
    dcl a1: Apointment;

    c.AddSlot(r1, t1,
              TimeRegionC`UserTimesToTimeRegion(
                  mk_TimeC`UserTime(2001,1,1,10,0),
                  mk_TimeC`UserTime(2001,1,1,12,0)));

    c.AddSlot(r3, t1,
            TimeRegionC`PeriodicTimeRegion(
              TimeRegionC`UserTimesToTimeRegion(
                mk_TimeC`UserTime(2001,1,1,10,0),
                mk_TimeC`UserTime(2001,1,1,12,0)),
              TimeC`mkTimeDuration(1,0,0),
              20));

    a1 := c.AddApointment(
      t1,
```

```
      TimeRegionC`UserTimesToTimeRegion
      ( mk_TimeC`UserTime(2001,1,1,11,0),
        mk_TimeC`UserTime(2001,1,1,11,10)
      ),
      {r3}, r1, <PUBLIC>, "Consulta Sr. Esteves");

    return
      c.SuggestSlots(
        t1,
        {r1, r3},
        TimeC`mkTimeDuration(0,0,30),
        TimeRegionC`UserTimesToTimeRegion(
            mk_TimeC`UserTime(2001,1,1,9,0),
            mk_TimeC`UserTime(2001,1,10,12,0)))
  );


end TestCompany
```

name	*#calls*	*coverage*
TestCompany`Test1	1	100%
TestCompany`Test2	0	0%
total		***94%***

5.2 Test cases

Cada caso de teste é especificado por intermédio de dois ficheiros:

- um ficheiro "*testcase*.arg" (em que *testcase* é para substituir pelo nome do caso de teste) com o comando (*script*) a executar pelo interpretador;
- um ficheiro "*testcase*.arg.exp" com o resultado esperado da execução do comando especificado em "*testcase*.arg".

Para esta aplicação apenas foi definido um caso de teste, designado "testcase1", que simplesmente cria um objecto da classe "TestCompany" e invoca o método "Test1" (onde ao fim ao caso está especificado o caso de teste).

5.2.1 Test case "testcase1"

5.2.1.1 File "testecase1.arg"

```
new TestCompany().Test1()
```

5.2.1.2 File "testecase1.arg.exp"

```
{ mk_TimeIntervalC`UserTimeInterval(
     mk_TimeC`UserTime( 2001,1,1,10,0 ),
     mk_TimeC`UserTime( 2001,1,1,11,0 ) ),
  mk_TimeIntervalC`UserTimeInterval(
     mk_TimeC`UserTime( 2001,1,1,11,10 ),
     mk_TimeC`UserTime( 2001,1,1,12,0 ) ) }
```

5.3 Test procedures

O procedimento de teste (procedimento que corre os casos de teste) está implementado em dois ficheiros:

- um ficheiro "vdmloop.bat" a invocar a partir da linha de que invoca o comando "vdmtest.bat" para cada caso de teste encontrado no directório corrente (recorde-se que cada caso de teste é especificado por um ficheiro com extensão ".arg");
- um ficheiro "vdmtest.bat" que é chamado a partir de "vdmloop.bat" para um caso de teste de cada vez, e que invoca o interpretador para executar o comando especificado no ficheiro ".arg" do caso de teste, guarda o resultado num ficheiro com extensão ".arg.res" e compara o resultado com o resultado esperado especificado no ficheiro com extensão ".arg.exp".

Após a execução de "vdmloop.bat" é também gerado um ficheiro "vdm.tc" (test coverage file) no directório corrente. Esse ficheiro deve ser depois movido manualmente para o directório que contém o ficheiro do projecto (datebook.prj), para se poder depois invocar a facilidade "Pretty Print" da VDM++ Toolbox (com opção "Enable test coverage coloring"), que gera um ficheiro "*classfile*.rtf.rtf" (com informação da cobertura do teste) para cada ficheiro "*classfile*.rtf" (ficheiro com a especificação original). O presente documento incorpora já os ficheiros ".rtf.rtf", produzidos pela facilidade "Pretty Print", em vez dos ficheiros ".rtf" originais.

5.3.1 Ficheiro "vdmloop.bat"

```
@echo off
rem Runs a collection of VDM++ test examples for the date book
application
rem Assumes specification is in Word RTF files

set S1=..\Company.rtf
set S2=..\Apointment.rtf
set S3=..\ApointmentType.rtf
set S5=..\DateBook.rtf
set S6=..\Resource.rtf
set S7=..\Employee.rtf
set S8=..\String.rtf
set S9=..\Date.rtf
set S10=..\Time.rtf
set S11=..\TimeInterval.rtf
```

```
set S12=..\TimeRegion.rtf
set S13=..\TestCompany.rtf


"C:\Program Files\The IFAD VDM++ Toolbox v6.7.19\bin\vppde" -p -R vdm.tc
%S1% %S2% %S3% %S5% %S6% %S7% %S8% %S9% %S10% %S11% %S12% %S13%
for /R %%f in (*.arg) do call vdmtest "%%f"
```

5.3.2 Ficheiro "vdmtest.bat"

```
@echo off
rem Tests the date book specification for one test case (argument)
rem -- Output the argument to stdout (for redirect) and "con" (for user
feedback)
echo VDM Test: '%1' > con
echo VDM Test: '%1'

rem short names for specification files in Word RTF Format
set S1=..\Company.rtf
set S2=..\Apointment.rtf
set S3=..\ApointmentType.rtf
set S5=..\DateBook.rtf
set S6=..\Resource.rtf
set S7=..\Employee.rtf
set S8=..\String.rtf
set S9=..\Date.rtf
set S10=..\Time.rtf
set S11=..\TimeInterval.rtf
set S12=..\TimeRegion.rtf
set S13=..\TestCompany.rtf


rem -- Calls the interpreter for this test case
"C:\Program Files\The IFAD VDM++ Toolbox v6.7.19\bin\vppde" -i -D -I -P -
Q -R vdm.tc -O %1.res %1 %S1% %S2% %S3% %S5% %S6% %S7% %S8% %S9% %S10%
%S11% %S12% %S13%

rem -- Check for difference between result of execution and expected
result.
if EXIST %1.exp fc /w %1.res %1.exp

:end
```

5.4 Test results

O teste realizado passou com sucesso. Observando as especificações das classes incluídas neste documento, já resultantes do "Pretty Print", conclui-se que o teste realizado cobre grande parte da especificação.

6 Code generation

6.1 C++ code generation

C++ source files (.cpp and .h) were generated with the facility "Generate C++" from within the toolbox. The source files generated were moved to a separate directory "datebookcpp" and compiled (as Win32 Console Application) with Microsotf Visual C++ 6.0 with the following project setting "Activate run-time type information".
Additional source files were created manually: "*classname*_userdef.h" and "main.cpp".
It was necessary to add the directive "#include "*classname*.h" to some "*classname*_userdef.h" files. Some changes were to be made to the specification in order to get C++ code that compiles and runs as expected.

O VDM++ permite combinar numa classe métodos implementados manualmente com métodos cujo código é gerado automaticamente, mas essa facilidade não foi explorada.

Seguem-se listagens do ficheiro "main.cpp", criado manualmente, e de dois ficheiros ".h" correspondentes a uma classe de exemplo (a classe "DateC"): o ficheiro "DateC_userdef.h" criado manualmente e o ficheiro "DateC.h" gerado automaticamente.

6.1.1 File "main.cpp"

```
// JPF

// Ja' e' incluido iostream com using namespace std #include <iostream.h>

#include "DateC.h"
#include "TestCompany.h"


main()
{
      char c;
      cout << "Vai criar objecto ...\n";
      vdm_TestCompany *tc = new vdm_TestCompany();


      cout << "Vai executar Test2 ...\n";
      cout << (tc->vdm_Test2()).ascii() << endl;
      cout << "Carregue numa tecla ...";
      cin >> c;

      cout << "Vai executar Test1 ...\n";
      cout << (tc->vdm_Test1()).ascii() << endl;
      cout << "Carregue numa tecla ...";
      cin >> c;

      return 0;
}
```

6.1.2 File "DateC_userdef.h"

```
//JPF

#define TAG_DateC 1000
```

6.1.3 File "DateC.h"

```
//
// THIS FILE IS AUTOMATICALLY GENERATED!!
//
// Generated at dom 24-Jun-2001 by the VDM++ C++ Code Generator
// (v6.7.19 - Wed 24-Oct-2001)
//
// Supported compilers:
//    gcc version 2.95.2 on Sun Solaris 2.6, Linux, HP-UX10
//    VC++ version 6.0 on Windows98, Windows NT
//

#ifndef _DateC_h
#define _DateC_h

#include <math.h>

#include "metaiv.h"

#include "cg.h"

#include "cg_aux.h"

#include "CGBase.h"

#include "DateC_anonym.h"



class TYPE_DateC_UserDate : public Record {
public:

  TYPE_DateC_UserDate () : Record(TAG_TYPE_DateC_UserDate, 3) {}

  TYPE_DateC_UserDate &Init (Int p2, Int p3, Int p4);

  TYPE_DateC_UserDate (const Generic &c) : Record(c) {}


  const wchar_t * GetTypeName () const {
    return L"TYPE_DateC_UserDate";
  }

  Int get_year () const;
  void set_year (const Int &p);
  Int get_month () const;
```

```
  void set_month (const Int &p);
  Int get_day () const;
  void set_day (const Int &p);
}
 ;

class type_ref_DateC : public virtual ObjectRef {
public:

  type_ref_DateC () : ObjectRef() {}


  type_ref_DateC (const Generic &c) : ObjectRef(c) {}


  type_ref_DateC (vdmBase * p) : ObjectRef(p) {}


  const wchar_t * GetTypeName () const {
    return L"type_ref_DateC";
  }

}
 ;


class vdm_DateC : public virtual CGBase {

  friend class init_DateC ;
public:

  vdm_DateC * Get_vdm_DateC () {
    return this;
  }


  ObjectRef Self () {
    return ObjectRef(Get_vdm_DateC());
  }


  int vdm_GetId () {
    return VDM_DateC;
  }

  enum  {
    vdm_UserDate = TAG_TYPE_DateC_UserDate,
    length_UserDate = 3,
    pos_UserDate_year = 1,
    pos_UserDate_month = 2,
    pos_UserDate_day = 3
  } ;
   vdm_DateC ();

  virtual ~vdm_DateC () {}

protected:
```

```
  virtual Bool vdm_inv_UserDate (const TYPE_DateC_UserDate &);
private:
  static Int vdm_Year0;
  static Record vdm_UserDate0;
  static Int vdm_DayOfYear (const Int &, const Int &, const Int &);
  static Int vdm_DaysOfYear (const Int &);
  static Bool vdm_IsLeapYear (const Int &);
  static Int vdm_DaysOfMonth (const Int &, const Int &);
  static Bool vdm_pre_DaysOfMonth (const Int &, const Int &);
  static TYPE_DateC_UserDate vdm_IncrementDate (const TYPE_DateC_UserDate
&, const Int &);
  static Int vdm_DaysBeforeYear (const Int &);
  static Bool vdm_pre_DaysBeforeYear (const Int &);
  static Int vdm_DaysBeforeMonth (const Int &, const Int &);
  static Bool vdm_pre_DaysBeforeMonth (const Int &, const Int &);
public:
  static TYPE_DateC_UserDate vdm_DateToUserDate (const TYPE_DateC_Date
&);
  static Int vdm_UserDateToDate (const TYPE_DateC_UserDate &);
}
;

#endif
```

6.2 Java code generation

Java source files (.java) were generated with the facility "Generate Java" from within the toolbox. It was necessary to disabled concurrency in VDM++ - Options - Java code generator. The type String`String was renamed to StringC`StringT to avoid name conflicts with java.lang.String.
A main source file was created manually: "MainDatebook.java".
The source files generated were moved to a separate directory "datebookjava" and compiled with Java 2 sdk version 1.3.0, with the command:

javac *.java

Program was run with the command:

java MainDatebook.

The environment variables CLASSPATH and PATH were changed.

O VDM++ permite combinar numa classe métodos implementados manualmente com métodos cujo código é gerado automaticamente, mas essa facilidade não foi explorada.

Segue-se listagem do ficheiro "MainDatebook.java", criado manualmente.

6.2.1 File "MainDatebook.java"

```
import dk.ifad.toolbox.VDM.*;
```

```
import java.util.*;

public class MainDatebook {

    public static void main(String[] args)
    {
        try
        {
            System.out.println("new TestCompany");
            TestCompany t = new TestCompany();

            System.out.println("Test2:" + UTIL.toString(t.Test2()) );
            System.out.println("Test1:" + UTIL.toString(t.Test1()) );
        }

        catch(CGException e){
            System.out.println(e.getMessage());
        }
    }
}
```

6.3 Performance comparison

Note: obtained with a different version of teste cases!

Test Case	**Language**	**Time**
TestCompany.Test1()	VDM++ interpreter	42 seconds
	Java	50 seconds
	C++	27 seconds

Java code generated by VDM++ tools is about half as efficient as C++ code.

6.4 Readability comparison

Java code generated by VDM++ tools is much easier to read than the C++ code. Java code generator seems to be better implemented than the C++ code generator.

7 References

http://www.ifad.dk

IFAD VDM Tools manuals.

Modelling Systems: Practical Tools and Techniques in Software Development.
John Fitzgerald, Peter G. Larsen. Cambridge University Press. 1998.

Appendix A - Major VDM++ Language Characteristics

(Source: http://www.ifad.dk/Products/VDMTools/vdmlangchar.htm)

The formal notation VDM++ is an object-oriented, model-based specification language, and is largely a superset of the ISO standardized notation VDM-SL.

In VDM++ a complete formal specification consists of a collection of class specifications. A class specification has the following components:

- *Class header*: This contains the class name declaration and inheritance information (single or multiple).
- *Types*: Definitions of any types used in the class.
- *Values*: Definitions of constant values.
- *Instance variables*: The state of an object consists of variables which can be of simple types, VDM-SL types such as sets, sequences and maps, and object references (the clientship relation). Instance variables can have invariant and initial expressions.
- *Operations*: Class methods that may be defined implicitly, explicitly (through imperative statements), or as a mixture of both. The implicit style uses pre and post condition expressions in the VDM-SL syntax.
- *Functions*: Functions are similar to operations except that the body of a function is an expression rather than a statement. Also, functions are not allowed to refer to instance variables.
- *Synchronization*: Operation invocation is defined with the Rendez-Vous semantics. It is possible to specify the circumstances in which an operation may be executed using a permission predicate for the operation. This predicate is over the instance variables of the object.
- *Thread*: In VDM++ active objects are considered to model active world entities. An object can be made active by the specification of a thread. A thread is a sequence of statements which are executed to completion, at which point the thread dies.

In addition, value, type, instance variable, operation and function definitions may be endowed with an access modifier (public, private or protected) dictating how that particular class member may be used by other classes.

Appendix B - Features of VDMTools®

(http://www.ifad.dk/Products/VDMTools/features.htm)

The IFAD VDM-SL Toolbox is a set of tools that supports the development of formal specifications using the ISO VDM-SL standard, extended with a structuring mechanism based on the concept of modules.

The IFAD VDM^{++} Toolbox is a set of tools that supports the object-oriented VDM^{++} extension of VDM-SL.

What separates **VDMTools®** from most other CASE tools for formal methods is the way the functional aspects of a specification is analysed. Although it is possible to produce mathematical proofs of properties of a specification, proof is complex and resource demanding. Even for medium size specifications, the automated proof support tools are often still not good enough.

An overview of the structure of **VDMTools®** can be seen here

Below the different features of **VDMTools®** are explained:

Specification Manager

The Specification Manager is developed in order to support the management of projects. In this context, a project is a collection of modules or a flat (unstructured) specification both distributed through a number of files.

The Specification Manager maintains a project by keeping track of the status of modules/classes and remembering this status from session to session.

View the graphical user interface of the Toolbox

These status indications present an easy overview of the specification and they facilitate the continuation of work from one session to another. Furthermore, the Specification Manager can automatically update the specification w.r.t. syntax checking, type checking, code generation etc.

Syntax Checker

The Syntax Checker checks if the syntax of your specification is correct according to the grammar for VDM-SL/VDM^{++}. In case any syntax errors are discovered they will be recorded via an error tool.

View the Error Tool

For each error identified the syntax checker will explain what was expected and what has been done to repair the syntax error.

Type Checker

When a file has been syntax checked it is possible to type check the modules/classes in that file. This is a static check for correctness of the use of the VDM concepts in the specification, i.e. whether the operands are of the right type for the operators which they have been used for and whether the variables used are in scope. The Type Checker can either be used in a mode corresponding to that of compilers for programming languages or alternatively in a mode which can identify all places where a run-time error could occur. When an error has been detected it will be shown in the error tool.

View the Error Tool with type errors

Interpreter and Debugger

The Interpreter/Debugger component supports all executable constructs in VDM-SL/VDM^{++}. This ranges from simple value constructors like set comprehension and sequence enumeration to more advanced constructs like exception handling, lambda expressions, loose expressions and pattern matching.

One of the benefits of executing specifications is that testing techniques can be used to assist validation of the specifications. In the development process small examples for parts of a specification can be executed to enhance the designer's knowledge of, and confidence in the specification. Furthermore, an executable specification can form a running prototype.

The Interpreter also provides debug facilities at specification level. This provides an easy overview of break points, function trace, and definitions available in the specification. Furthermore, the most common actions such as initialise, continue, step, single step etc. are easy to access through command buttons. A graphical user interface for this feature is available.

View the Interpreter Tool

The Dynamic Link Facility

The IFAD VDM-SL Toolbox has an add-on feature which makes it possible to integrate external C++ code into the specification. It also allows the user to execute external code as part of an interpretation of a specification. This feature is known as the Dynamic Link Facility.

This feature is particularly useful when:

- developing a new component to an already existing system, or
- there is a wish to use facilities that are directly supported by the operating system and/or the desired programming language - but which are not present in VDM-SL (e.g. I/O, trigonometric functions, graphical user interface).

This way it is possible to illustrate a specification for customers/managers who do not know the VDM notation at all.

The C++ Code Generator

Have your first implementation ready a few minutes after you have validated your formal specification!! **VDMTools®** supports automatic generation of C++ code from a VDM-SL/VDM^{++} specification. This will help to solve the consistency problem between specification and implementation.

VDMTools® generates fully executable code for most of the VDM-SL/VDM^{++} constructs leaving facilities for including user defined code for non-executable parts of the specification.

The C++ code can be generated with an option allowing run-time errors at the implementation level to be mapped back to the original specification. This gives an easy way to estimate if an implementation error is caused by an error at the specification level.

The Java Code Generator

Have your first implementation ready a few minutes after you have validated your formal VDM++ specification!! VDMTools® supports automatic generation of Java code from a VDM++ specification. This will help to solve the consistency problem between specification and implementation.

VDMTools® generates fully executable code for most of the VDM++ constructs leaving facilities for including user defined code for non-executable parts of the specification.

The Jave Code Generator contains more options than the existing C++ Code Generators. This includes the possibility to only generate code for the classes and the types from VDM++.

There is also an option in the Java Code Generator that enables it to produce different threads if one has used the concurrent part of VDM++. This feature will be valuable to users who wish to describe the concurrent aspects of their system directly inside VDM++.

Java does not support multiple inheritance directly. Thus, the Java Code Generator also contains an option for using interfaces from Java to handle VDM++ specifications with multiple inheritance; VDMTools® automatically computes those VDM++ classes which are suitable to be generated as Java interfaces, and the user simply selects from this list.

A facility has also been included to allow integration of automatically generated code with code that has been hand edited. When generating a Java file for which hand edited code exists, VDMTools® merges the generated file with the hand edited one to create one combined file. This means that both hand optimizations, and calls to other Java code can be preserved during the code generation process.

The Pretty Printer

The formal specification is usually accompanied with supporting text and figures. The Pretty Printer enables the production of such a mix by generating LaTeX sources. The pretty printer can automatically produce cross-reference indexing of all definitions.

The Test Coverage and Statistics Tool

When the Interpreter is used with a special option it is able to collect run-time information about test cases in a test environment. An overview of of this kind of test coverage information can be seen directly in **VDMTools®**.
View run-time information

In addition detailed information about the expressions which have not been covered can be displayed by the Pretty Printer by colouring the uncovered parts.
Dependency Browser

The VDM^{++} Toolbox has a Dependency Tool (view it) and an Inheritance Tool (view it) which together can be used for getting a better overview of the structure of a specification and the dependencies between the different classes.

VDM Toolbox API

A Corba-based Application Programmers Interface (API) is included in the October 2000 release of VDMTools® on both the Linux and the Windows platforms. This enables users to access all functionalities of VDMTools® externally from another process. Such external access to VDMTools® may be very valuable for users who:

- wish to combine their VDM model created using VDMTools® with a graphical user interface
- wish to extend the functionality of their own software development tool.

The former usage provides a prototyping capability in the early life-cycle phases which may be used to demonstrate how the requirements have been captured. This is illustrated in a number of examples on IFAD's web pages including the cash dispenser example. The latter usage will be more relevant to other tool developers or researchers.

The only requirement in order to be able to use this facility is to be able to create a process which can communicate with the VDMTools® process via a Corba compliant layer. See the new manual for the VDMTools® API for details

Couplings to Third-party Tools

The VDM^{++} Toolbox has a link to Rational Rose; a graphical CASE tool supporting UML. This solution is known as The Rose-VDM^{++} Link. Rational Rose is from Rational.

On-line Help

On-line help is available in a built-in hypertext browser: