[simula . research laboratory]

Formal Methods for Software Engineers

Professor Ray Welland

Department of Computing Science

University of Glasgow

E-mail: ray@dcs.gla.ac.uk

[simula . research laboratory]

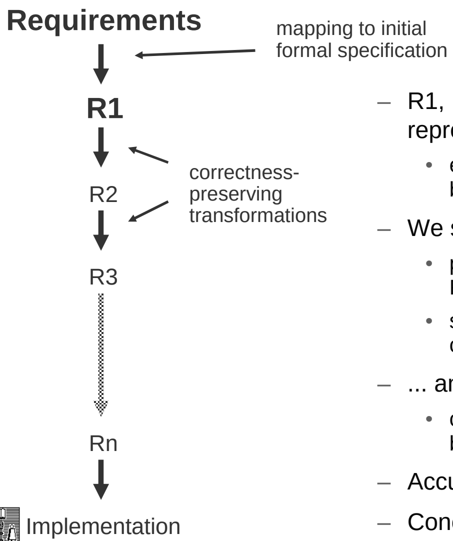
Overview

- Motivation
  - Why have formal specifications?
  - Where is their use appropriate?
  - What are the problems with using formal methods?
- Aims
  - Provide the background to formal methods (the 'big picture')
  - Cover examples of the use of one formal specification technique
- Contents
  - General introduction to formal specification (see Sommerville chapter 10)
  - Introduction to OCL (Object Constraint Language) associated with UML

Motivation - Certifiable Correctness

- Consider *safety-critical* systems
  - patient monitoring in hospitals
  - air-traffic control
  - railway signalling
  - process control of industrial/nuclear plants
  - on-board systems in a car
- Testing does not give us enough confidence
  - we need a *formal proof* that software is *correct*
- Proving an existing programs correct too difficult
- Instead, *construct* correct program by a series of steps known to be safe

Constructing a correct program



- R1, R2, ... Rn are formally specified representations
  - each step (after the first) is small enough to be automated or carried out simply
- We still need verification...
  - proving that R2 is a correct transformation of R1
  - should be trivial if transformations preserve correctness
- ... and validation
  - convince the user that the right system is being built
- Accurate requirements are essential!
- Concentrate on R1 - the initial specification

How to write a formal specification....?

- Not in natural language!
  - impossible to supply sufficient precision
  - although there have been attempts at "structured English", but....
- Diagrams tricky...
  - cannot formally manipulate them easily
  - but they might be used as an adjunct to formal specification - animation
- Must use a notation that is mathematically based
  - formal semantics
  - can be manipulated, in a mathematical sense

Why not use Formal Specs for all program development?

- The effort involved (mostly by hand) and skills required
- Lack of tool support, although some are becoming available
- Lack of necessary background and poor training of existing staff, together with the use of unfamiliar notations
- Lack of knowledge among project managers
- Validation problems
  - hard to communicate ideas to users - might build perfect, but invalid, system
  - again, tools required - animation, alternative representations
- Problems of scale
  - formal specification techniques not suited for very large projects - lack of modularity, information hiding in some traditional f.s. techniques

Background Reading

- The Mystery of Formal Methods Disuse (A story of zealotry and chicanery) – Robert Glass (Practical Programmer column) – Comm. ACM 47(8), August 2004, p15-17
  - a typical Robert Glass column taking a sceptical view of formal methods!
  - refers to the paper below

- Getting the best from formal methods, John B Wordsworth – Information and Software Technology, 41 (1999), 1027-1032
  - reviews progress made in the use of formal methods in the last 15 years
  - suggests reasons for lack of widespread use of formal methods
  - proposes ways to 'infiltrate' formal methods into software development

Background Reading (2)

Two older references but both very readable and still relevant:

- An Invitation to Formal Methods - IEEE Computer, April 1996)
  - consists of a collection of short papers giving widely differing views about formal methods, ranging from formal methods enthusiasts to sceptical practitioners;
  - good overview of the 'state of the art' and easy to read.

- Seven More Myths of Formal Methods, J Bowen & M Hinchey - IEEE Software, July 1995
  - this article is written by two formal methods enthusiasts and strongly advocates the use of formal methods;
  - very biased but again easy to read.

Formal Specification of Large Systems

- Algebraic specification
  - system described using interfaces between sub-systems
    - operations of an interface and the relationships between them
  - entities and operations defined along with axioms defining the semantics of the operations - hence the behaviour of the entity
    - 'formality' is in the axioms
  - More in Sommerville, 10.2
- Model-based specification
  - model constructed using well-understood mathematical entities - sets, sequences
  - specification is expressed as a system state model over these entities
  - Two major model-based approaches
    - VDM (Vienna Development Method), IBM Vienna Research Labs
    - Z, Programming Research Group, Oxford
  - More in Sommerville, 10.3

Specifying Constraints

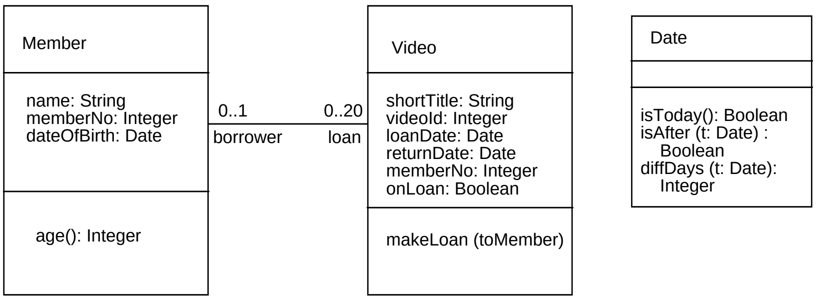
- A less comprehensive approach to formal specification is to combine formal notations with existing diagrammatic notations to improve precision
- Constraints allow us to define parts of our system model more precisely than using only diagrams
  - define the basic model using diagrams (e.g. class diagram)
  - add detail using constraints attached to the diagram elements
  - ensure that all requirements are captured and can be traced
  - there are trade-offs between adding detail to diagrams and using constraints - when does a diagram become too complex to be useful?

OCL - Object Constraint Language

- Note – OCL 2.0 (with UML 2.0)
- A constraint is a restriction on one or more values of (part of) an object-oriented model or system
- Constraints may be visually represented (restriction constraints) or expressed textually
- OCL provides a well defined language for expressing constraints textually
- UML diagrams provide the visual representation of the object model, restriction constraints and the context for OCL constraints

[simula . research laboratory]

A Simple UML Example



© Simula Research Laboratory - R. Welland 2006

Why use textual constraints?

- Better documentation
  - additional information is linked to system model(s)
  - can be versioned together with model(s)
- Reduce diagram complexity
- Improve precision
  - mathematical theory underpinning the language
  - textual constraints can be parsed and checked
- Communication
  - an agreed common language for expressing requirements
  - analyst to designer; designer to developer
- Link to detailed requirements capture
  - tracing requirements through development

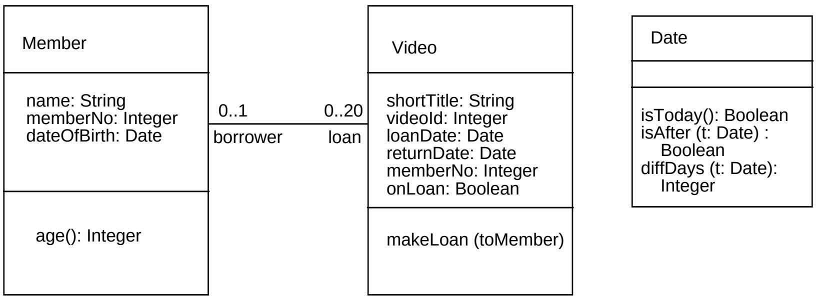
OCL - Requirements

- Precise and unambiguous language, easily read and written by *practitioners*
  - based on sound mathematical principles
  - written in a more ‘natural’ style (avoids special symbols)
- Declarative
  - No side-effects of expressions
  - Not operational (no corrective actions)
- Typed, so that it can be checked (but not executed)
- NOT a programming language!

Types of Constraints

- Invariant - a constraint that must always be met by all instances of a class, type or interface. An expression that must evaluate to true at all times.
- Pre-condition - a constraint that must be true at the moment an operation (method) is to be executed
- Post-condition - a constraint that must be true at the moment an operation has just ended
- And many others not covered in these lectures …

UML Class Diagram - Example



Simple Invariants

context **Member** *or* context **Member**

inv: **memberNo > 999** inv: **self.memberNo > 999**

context **Member** *or* context **Member**

inv: **age () > 17** inv **minAge**: **age () > 17**

context **Video** context **Video**

inv: **shortTitle.size() <= 20** inv: **returnDate.isAfter (loanDate)**

context **Video**

inv: **loanDate.diffDays (returnDate) = 14**

Pre and Post Conditions

context **Video :: makeLoan (toMember)**

pre: **not onLoan**

post: **result = (loanDate .isToday ())**

context **Video :: makeLoan (toMember)**

pre: **onLoan = false**

post: **result = (loanDate.isToday ()**

and

loanDate.diffDays (returnDate) = 14)

Navigating Associations

- Navigating an association from the context class to another class creates a SET of objects.
- Operations on sets are denoted by ->
- There are many operations available, for example:
  - *set* -> isEmpty -- Boolean, true if set contains no elements
  - *set* -> notEmpty -- Boolean, true if set contains one or more elements
  - *set* -> size() -- Integer, number of objects in set
  - *set* -> forAll (expression) -- Boolean, true if expression is true for all elements of the set
  - *set* -> exists (expression) -- Boolean, true if expression is true for at least one element of the set

[simula . research laboratory]

Associations and Sets (Examples)

- - loan is a set of Video instances; all of which

- - must have the same memberNo as the borrower (Member)

context **Member**

inv: **loan -> forall (memberNo = self.memberNo)**

- - borrower is also a set, of 0 or 1 values!

context **Video**

inv: **borrower -> notEmpty implies**

borrower -> forall (memberNo = self.memberNo)

The forall Operation

- - the constraint on the previous slide can be written more explicitly as:

context **Member**

inv: **loan -> forall (v : Video | v.memberNo = self.MemberNo)**

- - allinstances returns a set of all instances of a class

context **Member**

inv: **Member.allinstances -> forall (m1, m2 | m1 < > m2**

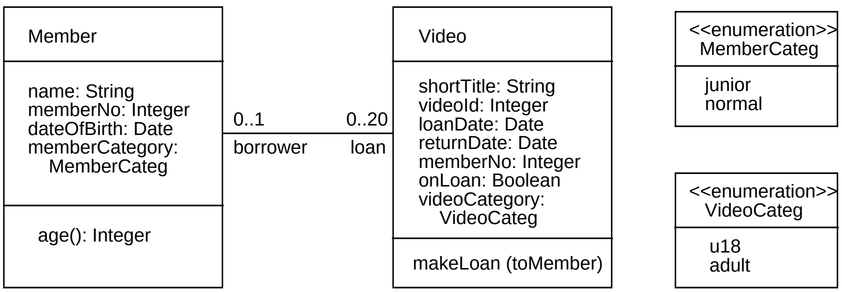
implies m1.memberNo < > m2.memberNo)

context **Member**

inv: **Member.allinstances -> forall (m | m < > self**

implies m.memberNo < > self.memberNo)

Video Example (extended)



More Invariants (on enumerated types)

context **Member**

inv: **memberCategory = MemberCateg::junior** implies **Age () < 18**

and

memberCategory = MemberCateg::normal implies **Age () > 17**

context **Member**

if **memberCategory = MemberCateg::junior**

then **Age () < 18**

else **Age () > 17**

endif

More Invariants (2)

- - Restricting the number of loans for junior members:

context **Member**

inv: **memberCategory = MemberCateg::junior implies**

loan -> size() <= 10

- - Restricting video categories for junior members:

context **Member**

inv: **memberCategory = MemberCateg::junior implies**

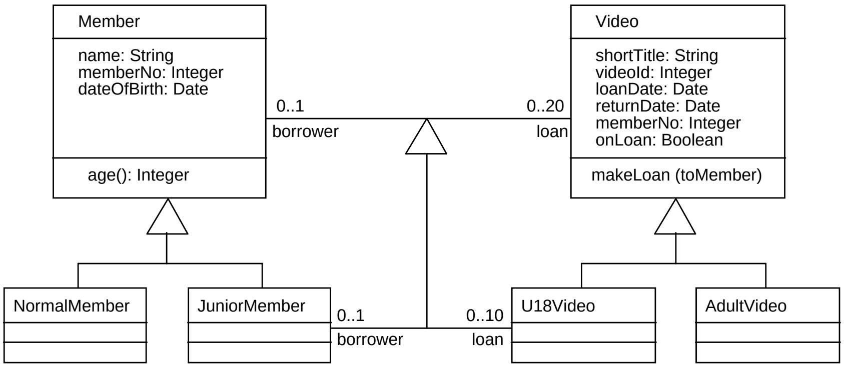
loan -> forall (videoCategory = VideoCateg::u18)

context **Video**

inv: **videoCategory = VideoCateg::adult implies**

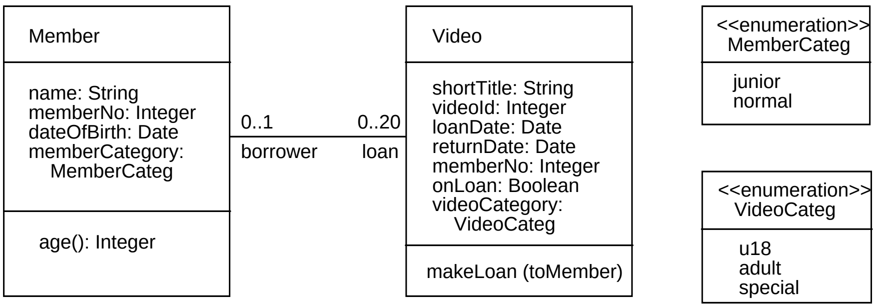
borrower -> forall (memberCategory = MemberCateg::normal)

Diagram or Textual Constraints?



This still does not work! A U18Video may only be borrowed by a juniorMember in this model.

Video Example (extended again!)



More Invariants

- - No normal member may have more than 3 special videos

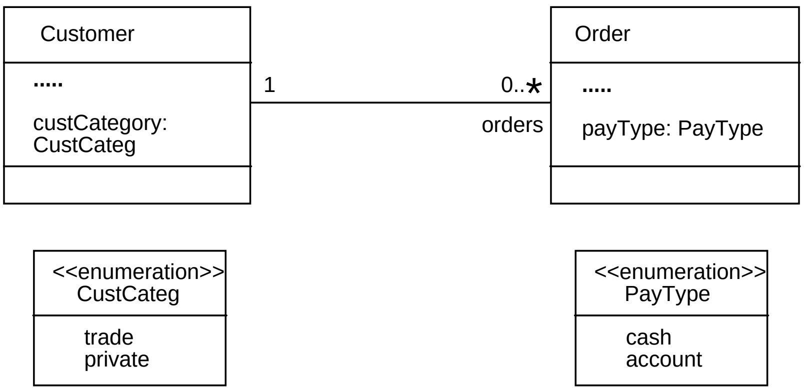
context **Member**

inv: **memberCategory = MemberCateg::normal implies**

loan -> select (videoCategory = VideoCateg::special) -> size() <=3

- - cannot express this diagrammatically

Another Simplified Example



Constraints on Customer/Orders

context **Customer**

inv: **custCategory = CustCateg::trade implies**

orders -> forall (payType = PayType::account)

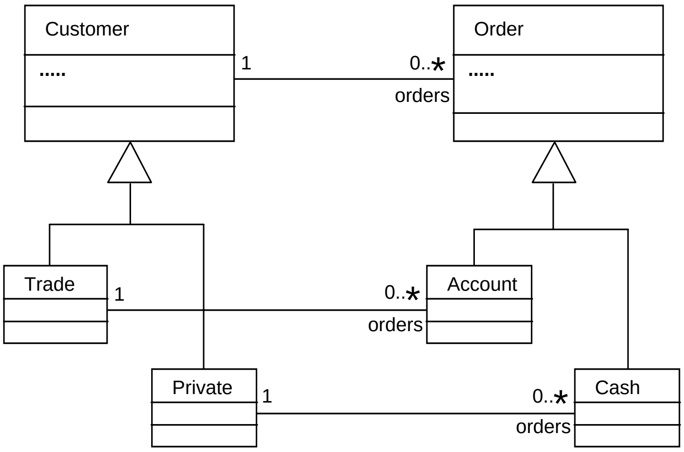
context **Customer**

inv: **custCategory = CustCateg::private implies**

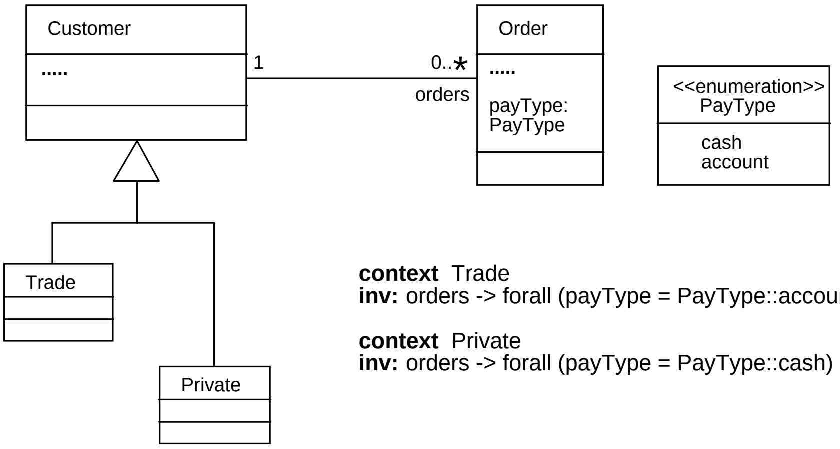
orders -> forall (payType = PayType::cash)

- - could write constraints on Order

Diagrammatic Constraints



Mixing the constraints!



context Trade
inv: orders -> forall (payType = PayType::account)

context Private
inv: orders -> forall (payType = PayType::cash)

[simula . research laboratory]

Summary

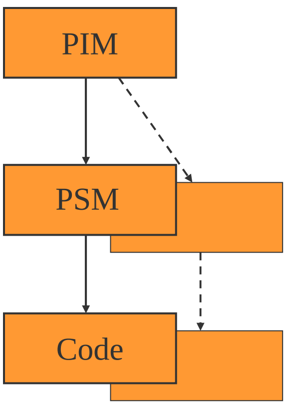
- Within the context of a class, we can write invariants on:
  - the attributes of that class
  - the members of classes associated with that class
- Can write pre and post conditions on an operation (method) of a class
- OCL can be used in conjunction with other UML diagrams (not covered in these lectures)
- OCL is declarative not operational
- All OCL expression used in constraints are:
  - Boolean type (i.e. must evaluate to true or false)
  - free of side effects (i.e. no update operations)

Reference(s)

- The Object Constraint Language Second Edition – Getting Your Models Ready for MDA. Jos Warmer and Anneke Kleppe. Addison-Wesley 2003.

- Web sites to check out:
  - The website of the authors of the above book
    http://www.klasse.nl/ocl
    that provides useful background information,
    including an OCL syntax checking tool called Octopus
  - OMG standard for UML including OCL:
    http://www.omg.org
    {only if you really like standards!!}

Model Driven Architecture (MDA)



- PIM = Platform Independent Model; UML + OCL
- PSM = Platform Specific Model; could be Database model or EJB, for example
- Code is generated from PSM automatically
- PIM can be transformed to PSMs automatically
- PIM to PSM tools are limited