Validated Designs for Object-Oriented Systems

Part of
Advanced Course on Reliable Software Technology
Course leader: Marina Waldén
April-May 2005

Literature

- Examples at [http://www.vdmbook.com](http://www.vdmbook.com)
About the Book

- Combine techniques and tools improved by careful, rigorous and machine supported analysis
- Design of object-oriented systems validated with automated tool support
- Industrial examples; authors have industrial co-operation and experience

- Modelling object-oriented systems in VDM++
- Modelling in practice
- From models to code

Motivation

The challenge of software development:
- Develop software to satisfactory standards within time and budgetary constraints
- Requirements change rapidly
- Traceability between requirements and software structure is needed
The Role of Modelling

- **Model** - simplified or idealised description of a system, situation or process as a basis for theoretical or empirical understanding of calculations etc. (by Oxford English Dictionary)

- For understanding the requirements
- Graphical representation of data and functionality
- Program-like representations of algorithms
- Formal mathematical representation of functional and temporal behaviour using logics

The Role of Modelling

**Abstraction**
- omission of detail
- what the model is intended to address

**Rigour**
- extent to which the model can be analysed objectively
- high level of rigour by formal methods
  - mathematical semantics,
  - unambiguity
Modelling within Software Development

Software development - implementing a system that satisfies the requirement

- Requirements (natural language, graphics, mathematics)
- Implementation (implementation medium, target language, computing system)

SW development strives to decrease the gap

Design
- constructing outline of the implementation from requirements
- Implementation details are introduced later

Validation
- examining a model to detect conformity with user needs

Verification
- examining a developed product to determine conformity with its specification
Modelling Systems

- Systems are here viewed as collections of interactive objects that contain data and provide services
- Formal modelling language of VDM++ (Vienna Development Method)
  - Type definitions and Data type invariant
  - Instance variables
  - Function definitions
  - Operation definitions
- Formalising the requirements
- Rapid prototyping techniques
- Consistency checking (proofs)
- Translation to Java

Book Content

Case studies:
- Chemical plant (ch. 2-3)
  - Guided tour of VDM and tool support
- Electronic patient records (ch. 4-5)
  - Basics of modelling language
- Robot Controller (ch. 6)
  - How to handle collections
  - Model motion through a space
- Congestion warning system (ch. 7-8)
  - Ordered collections, relationships
  - Progressively more sophisticated models
  - Monitoring system for traffic
Book Content

- **Enigma** (ch. 9)
  - Modelling technology in practice
  - Cryptography, Enigma cipher
- **Control speed limitation and monitoring** (ch. 10)
  - Linking UML and VDM++ modelling consistence
  - Railway system
- **TradeOne** (ch. 11)
  - Development costs and product quality
  - Trading in stock options (Japan)
- **POP3 server** (ch. 12-14)
  - Advanced aspects of modelling
  - Protocol enables fetching email from a server

Course Schedule

- **20.04.2005**
  - Ch. 4 -
  - Ch. 5 - Discussion
  - Ch. 6 -

- **27.04.2005**
  - Ch. 7 - Marcus Alanen
  - Ch. 8 - Leonidas Tsiopoulos
  - Ch. 9 - Dubravka Illic

- **04.05.2005**
  - Ch. 10 - Mats Neovius
  - Ch. 11 - Qaisar Malik

- **11.05.2005**
  - Ch. 12 - Johannes Eriksson
  - Ch. 13 - Fredrik Degerlund
  - Ch. 14 - Pontus Boström
Building a model in VDM++

Guide to developing a formal model in VDM++:
- Purpose of construction of the model
- Read the requirements
- Analyse functional behaviour
- List of classes and operations
- UML class diagrams to VDM++
- Signatures for operations (Consistency in VDM++)
- Invariant properties
- Pre- and postconditions for operations
- Validate by testing
- Implementation via automatic code generation

A Chemical Plant Example

Alarm management system for a chemical plant
- Inspired by a subcomponent of a large alarm system developed by IFAD A/S
- System manages the calling out of experts to deal with operational faults discovered in a chemical plant
- Sensors are able to raise alarms in response to conditions in the plant
- When an alarm is raised, an expert must be called to the scene
- Experts have different qualifications for coping with different kind of alarms
- Model the rules concerning the duty schedule and the calling out of experts
Dictionary of Requirements

Potential Classes and Types (Nouns)
- Alarm: required qualification and description
- Plant: the entire system
- Qualification (electrical, mechanical, biological, chemical)
- Expert: list of qualifications
- Period (depending on the shift system)
- System and system database? This is probably a kind of schedule

Potential Operations (Actions)
- Expert to page: when an alarm appears (involves alarm operator and system)
- Expert is on duty: check when on duty (expert and system)
- Number of experts on duty: presumably given periods (operator and system)

Making the Model More Comprehensive

- Optimal level of abstraction
  - analysable vs. executable
  - abstract; not too complex to analyse in depth
  - clear and precise; details for implementation

- Important properties or constraints as invariants
- Implicit definition of the operation on most abstract level in form of pre- and postcondition to postpone details
- May identify additional invariants if needed when defining operations
Validating the Model

- "How good is the model?"
- The model should have the properties expected of a system satisfying the client's expectations

- Stronger conditions and more comprehensive testing
  - Higher confidence in the model
  - Earlier discovery of defects
  - Less rework
  - Lower defect correction cost

Approaches of Validating

- Integrity properties (proof obligations)
  - Automatically generated and mathematically proved with VDM Tools
- Systematic testing
  - Test models interactively and in batch mode
- Rapid prototyping (cf. animation in B)
  - Application Programmers Interface
  - Test the model directly via GUI
  - Use CORBA check abstract specifications with e.g. legacy code
Automatic Code Generation

- Strong correspondence between abstract model and code
- Reuse of data
- Reduction in time
- Efficiency of code?

VDM++ Tool Support

- **Rational Rose®**
  - Info on class diagrams
  - Moving on from UML to VDM++

- **VDM Tools®**
  - Syntax checking
  - Type checking
  - Mapping back to Rose to update diagrams
  - Executing the model
  - Integrity checking
    - Increasing confidence in the consistency of the model
  - Code generation
  - CORBA-based API