Practice-oriented courses in formal methods using VDM

Peter Gorm Larsen, John S. Fitzgerald, Steve Riddle

DOI 10.1007/s00165-008-0068-5
BCS © 2008
Formal Aspects of Computing (2009) 21: 245–257

Formal Aspects of Computing

Practice-oriented courses in formal methods using VDM^{++}

Peter Gorm Larsen[1], John S. Fitzgerald[2] and Steve Riddle[2]

[1]Engineering College of Aarhus, Dalgas Avenue 2, 8000 Aarhus C, Denmark. E-mail: pgl@iha.dk

[2]School of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK

Abstract. We describe the design and delivery of two courses that aim to develop skills of use to students in their subsequent professional practice, whether or not they apply formal methods directly. Both courses emphasise skills in model construction and analysis by testing rather than formal verification. The accessibility of the formalism is enhanced by the use of established notations (VDM-SL and VDM^{++}). Motivation is improved by using credible examples drawn from industrial projects, and by using an industrial-strength tool set. We present examples from the courses and discuss student evaluation and examination performance. We stress the need for exercises and tests to support the development of abstraction skills.

1. Introduction

Why do we teach formal methods? The majority of graduates from university computer science courses may not use formal methods directly in their subsequent professional practice. However, if they are to make the most of advances in software technologies, and if they are to have the skill to evaluate new technologies and tools as they emerge, they require an appreciation of the principles of abstraction and rigorous reasoning that underpin them. It follows that the vast majority of students in computer science and software engineering should be exposed to formal methods in some form. However, there are some impediments to this. First, there is a perception that formal methods are a heavyweight technology, requiring advanced mathematical skills and demanding an investment of learning effort far in excess of the potential returns. Second, there is a view in the profession as well as among students that these techniques have not been applied seriously, that they are a branch of theory and that they lack credibility.

Our experience of applying formal methods in industry (e.g. [LFB96, FL07]) led to the view that "lightweight" application of formal methods could be cost-effective. Lightweight approaches have been advocated for some time [Jon96, JW96]. They emphasise carefully targeted formal modelling as an adjunct to other development processes, rather than a wholesale replacement for them. They also emphasise the tractability of formal specification languages over their expressiveness and aim for models that cover significant parts of systems rather than aiming for comprehensiveness. They also open the possibility that partial analysis might be preferred to fully formal verification. Lightweight methods and tools in this sense provide at least some of the benefits of formalism without requiring the full application of highly specialised technology.

Correspondence and offprint requests to: P. G. Larsen, E-mail: pgl@iha.dk

This paper describes experiences developing and delivering courses that endeavour to equip students with generic skills of abstraction and rigorous analysis by means of lightweight formal methods using VDM[1] and its support tools. We concentrate on two introductory 5 ECTS credit[2] courses offered at undergraduate or Masters level in different institutions: the Engineering College of Aarhus (IHA), Denmark, and Newcastle University, UK. Both courses are designed to be accessible to the majority of Computer Science or IT students and both emphasise the credibility of formal methods by using industrial examples and extensive practical work. This emphasis on practice carries through to the assessment techniques used in the courses.

We describe the institutional context and the capabilities of students at whom our courses are targeted in Sect. 2. Responses to the challenges of accessibility and credibility embodied in our courses are described in Sects. 3 and 4. We describe the structure and delivery of the courses in Sect. 5 and evaluate their current status in Sect. 6, considering the improvements and modifications we would like to make in future. Although the paper concentrates on introductory courses, we briefly describe the more advanced courses that follow on from these. We conclude with a discussion of the role of lightweight formal methods in the computing curriculum.

2. Context

In order to explain the approach taken to skills development in our courses, it is necessary to understand a little of the institutional contexts as well as the background of the students taking our courses and the experience of the faculty.

2.1. Institutional context

IHA is a small college specialising in engineering disciplines, including civil, construction, mechanical and electronic engineering, as well as computer technology and embedded systems. The college has particularly strong links with Danish industry, and consequently the educational focus is on applied research aimed at improving industrial competitiveness. The Bachelor of Science (BSc) studies in the IT area have a focus on embedded systems. In collaboration with Aarhus University a Masters programme is provided and it is here that the VDM courses are given.

Newcastle University is one of the UK's smaller research-intensive universities, and one of the first to offer undergraduate degrees in Computing, producing its first graduates in 1969. The majority of students in the School of Computing Science are now undergraduates studying for the BSc honours degree in Computing Science. The School's strong research ethos means that the curriculum is influenced both by the research interests of the faculty as well as by the needs of industry and the student market. Historically, the school has had a strong practical rather than theoretical focus in its programmes, stressing software design, programming and fault tolerance rather than formal methods of defect avoidance.

In different ways, both institutional settings call for formal methods courses that are primarily practical. In IHA, the imperative is to equip students with generic skills for industry application. At Newcastle, the need is for courses that are consistent with a degree programme and research ethos that is stronger on systems development than on theory.

2.2. Student backgrounds

The IHA course is taken as an option by MSc students just after BSc level. Students may go on to take a further specialised course in modelling and analysis of embedded systems. A typical class size is around 15.

In general the background for the IHA students is very practical. They have studied and used C++ and traditional object-oriented paradigms in their BSc studies. However, they have not been exposed to functional programming languages. Their skills in discrete mathematics are quite limited, although some of them have taken another optional 5 ECTS credit course introducing propositional and predicate calculus, relations and functions, recursive structures and induction.

[1] The Vienna Development Method—see http://www.vdmportal.org.

[2] ECTS is the European Credit Transfer and Accumulation System. A typical full-time student will study 60 ECTS credits in one academic year.

The Newcastle course is taken by all students in the fourth semester of a 6-semester BSc honours programme in Computing Science. The class size is around 60.

When they encounter their first formal methods course, Newcastle students have at least what Boute describes as "Basic Computing Engineering Mathematics" [Bou03]. We rely on basic skills in formal propositional and predicate calculus, relations and functions, recursive structures and induction, in addition to some continuous mathematics. Students have studied and used Java together with the background on the object-oriented paradigm that Java requires, as well as basic program design and algorithm analysis. They have also been exposed to functional programming (in our case, using Haskell); in common with Pepper [Pep04], we found this was very beneficial for students' practical work in formal modelling.

It is fair to say in both institutions that, when they encounter our courses, students have had much more experience at programming than they have had at conscious abstraction or rigorous reasoning. Indeed, the concept of abstraction is still quite novel, and hence understanding abstraction is a major "knowledge outcome" for our courses. Other outcomes include: understanding the value of rigorous system description; knowledge of relevant analysis and validation methods such as animation, testing, counterexamples and, to a limited extent, formal reasoning.

2.3. Prior experience with VDM

VDM [Jon90, FL98] is a well established formal method which has been taught in specialist courses for over twenty years. In its current form, it consists of a formal modelling language VDM-SL, standardised in 1996 [And96], a proof theory [BFL^{+}94] and a refinement theory [Jon90]. A comparative study of industrial developments with and without formal techniques [LFB96] encouraged us to believe that a lightweight model-oriented formalism could be embedded successfully in an industrial development process provided:

- the formalism is treated as a precise modelling language, a tool for use at levels of abstraction determined by the user, not as part of a greater (refinement-based) methodology;
- the formalism is presented in an accessible way; tools should link to existing tool support and not require users to be trained on platforms specifically to support the formalism;
- tool support is powerful but lightweight; always favouring automation over completeness.

Our experience using this approach, developing relevant industrial training courses and the first courses prepared for Newcastle in the mid-1990s, led to the approach advocated in a text in 1998 [FL98]. Subsequent experience with industrial applications led to the extension of VDM to support object-oriented design and concurrency, producing VDM^{++} [FLM^{+}05]. Further developments aim to provide a capability for modelling real-time distributed systems [VLH06, VL07, FLT^{+}07].

To summarise, the challenge at IHA and Newcastle is to develop courses that are suitable for students with limited mathematical experience but good programming background, within practice-oriented degree programmes. In particular, we must address the issues of accessibility and credibility of formal methods. In the following sections, we describe how the courses offered so far attempt to address these challenges, using examples drawn from the course materials.

3. Enhancing the accessibility of formal methods

3.1. Choice of formalism

VDM-SL and VDM^{++} are both *model-oriented*. Data is expressed in terms of base types such as numeric types, structureless tokens and enumerations. Structured types may be built from these basic ones using type constructors which include collections such as sets, sequences and mappings. Distinguished instance variables model persistent data if required. Invariants can be expressed over types and over instance variables. Functionality is defined in terms of referentially transparent functions over the defined data types, or operations over the instance variables. Abstraction is provided in data by the unconstrained character of the basic types and type constructors: for example, there is no maximum integer, and sets have unconstrained, though finite, cardinality. Functions and operations may be specified implicitly in terms of preconditions and postconditions, or explicitly in terms of expressions and statements.

Several factors in the choice of the formalism are relevant to its use in teaching. First, the structure of models is familiar to students already versed in object-oriented design. Second, the use of referentially transparent functions is familiar from functional programming. It is difficult to engage students with a language that is not perceived as mainstream [PO04]. Using a formalism with some familiar features allowed us to focus attention on the less familiar aspects: the abstraction afforded by types and type constructors, specification by means of pre and post-conditions and key issues such as the use of partial operators. Following this, we are able to introduce some elements of rigorous analysis, in particular the generation of proof obligations. These are perhaps best illustrated by means of an example taken from the course material.

Example: the trusted gateway

This example is derived from a model developed as part of the ConForm project with British Aerospace (Systems and Equipment Ltd) [LFB96]). We will briefly describe the original model, before discussing how it was simplified in order to provide teaching material. We note initially, however, that we are here presenting models based on a real industry application that itself yielded valuable insights into the effort profile and other characteristics of the development process when formal techniques are employed. The use of such examples lends credibility to the underlying technology.

A trusted gateway implements a security policy on messages that may have some confidential content. It acts as a filter, preventing messages that contain secret material from entering computing systems that are not trusted to process such messages securely. Messages arrive at the gateway's input port, to be assessed to determine their security classification. In a simple two-output gateway, messages deemed to be high-security are passed to a designated high-security output port, and low-security messages to a low-security output port. Message classification is based on whether the message contains special *marker strings* from sets known as *categories*. In the British Aerospace study a state-based model was originally developed in which messages were classified individually. The model incorporated the category sets and the message under analysis. Subsequently, a useful further abstraction was found in which the messages were arranged into an input sequence.

In our approach, a key principle of abstraction [FL98] is the removal of detail that is not relevant to a model's purpose. When considering how to use this example in teaching, the question of the model's purpose changes subtly. While remaining a credible model of a trusted gateway, it must additionally provide a non-trivial, representative example of the use of relevant abstractions. At the point in the course where this model is used, the students are taught about sequences, their operators and how to write functions to manipulate them. The model should therefore illustrate the main issues with the use of sequences, which include:

- appropriate choice of sequence datatype;
- sequence definition through enumeration and subrange;
- the relation between sequences and sets, through `elems` and `inds` operators;
- sequence operators head, tail, index and dangers inherent in partial operators;
- how and when to use datatype invariants;
- explicit functions and definition of preconditions.

An extract from the original model is shown in Fig. 1. This uses sequence operators such as indexing and concatenation, head (`hd`) and tail (`tl`), some of them partial. However, even this representation can be simplified without reducing its utility for teaching. Since the set abstraction has already been dealt with through separate examples, the use of a set of strings to represent a category is not strictly necessary. Instead, we can simplify by eliminating the category sets, classifying a message as high security if it contains a specified string "SECRET" and low-security if it contains the word "UNCLASSIFIED" (without the word "SECRET"). This allows us to simplify the `Classify` function to merely look for occurrences of the single trigger words, which still achieves the aim of illustrating sequence indexing. A classroom exercise can then be added, extending the model to deal with the category set.

In presenting the example the basic data types are described first, with discussion to motivate the choice of representations such as quote types for the classifications and the invariants on `Message` and `String`. The functions to recursively traverse the message stream and classify the input are then introduced in a piecewise fashion. The example is revisited in a later part of the course when validation is considered, by examining the domain-checking proof obligation in terms of the `Gateway` function. A useful feature of the gateway is the balance that must be struck between security and liveness. In discussing validation using this example, we point

```
types
String = seq of char
inv s == s <> [];

Classification = <HI> | <LO>;

Category = set of String;

Message = String
inv m == len m <= 100;

Ports :: high: seq of Message
         low: seq of Message;

functions
Gateway: seq of Message * Category * Category -> Ports
Gateway(ms, hicat, lowcat) ==
  if ms = []
  then mk_Ports([],[])
  else let rest_p = Gateway(tl ms, hicat, locat)
       in
         ProcessMessage(hd ms, hicat, locat, rest_p);

Classify: Message * Category * Category -> Classification
Classify(m, hicat, locat) ==
   if exists hi in set hicat & Occurs(hi,m)
   then <HI>
   else if exists lo in set locat & Occurs(lo,m)
        then <LO>
        else <HI>;

ProcessMessage: Message * Category * Category * Ports -> Ports
ProcessMessage(m,hicat,locat,ps) ==
   if Classify(m,hicat,locat) = <HI>
   then mk_Ports([m]^ps.high,ps.low)
   else mk_Ports(ps.high,[m]^ps.low)
```

Fig. 1. Trusted Gateway Model with category sets

out that a validation conjecture might be the security property that only those messages that are definitely low-security are output to the low security port. A simple implementation of this is, of course, to treat all messages as high security, but this conflicts with the functional requirements.

3.2. Role of tool support

VDM's recent successes in industrial application have been closely tied to its tool support, and the courses reflect this. Alongside the development of the denotational semantics given in the ISO VDM Standard, a toolset that implemented an operational semantics was developed [LL91, ELL94, FL08]. The product that ultimately resulted from this work, VDMTools, aimed for cost-effective industrial utility rather than expressive completeness. The VDMTools product is now maintained on a commercial basis by CSK Systems.

VDMTools support syntax and type checking for the full VDM^{++} language, and contains an interpreter for test-based analysis of specifications written within an executable subset. Testing is further supported by coverage analysis tools. The interpreter has a dynamic link library feature allowing external (non-VDM) code to be incorporated. Automatic code generation to C++ and Java are also supported. VDMTools do not (yet) contain direct proof support, although automatic proof obligation generation is available and automated discharging of proof obligations (via HOL) has been demonstrated [AS99, Ver07]. A significant feature of VDMTools is the inclusion of a bidirectional link to the Rose tool for UML modelling. In particular, UML class diagrams can be linked directly to object-oriented VDM^{++} models so that changes to the model can be reflected directly in changes to the diagram and vice versa.

The main tool features used in our introductory courses are syntax and type checking, the interpreter and test coverage tools. For example, for the trusted gateway, students use VDMTools to interpret the model with actual test values. The coverage of the tests selected can then be analysed automatically. In addition, students can automatically generate proof obligations and determine whether for each case they have, for example, properly guarded the use of partial operators with pre-conditions.

The use of the interpreter naturally favours executable specifications. Not all models should be written to be directly executable [IH89], and even underspecified functions may be executed if putative result values are provided. However, for our students, we agree with Utting and Reeves [UR01] that the direct execution of a model encourages its exploration. Speed is often a problem with executable formal models, but we are fortunate

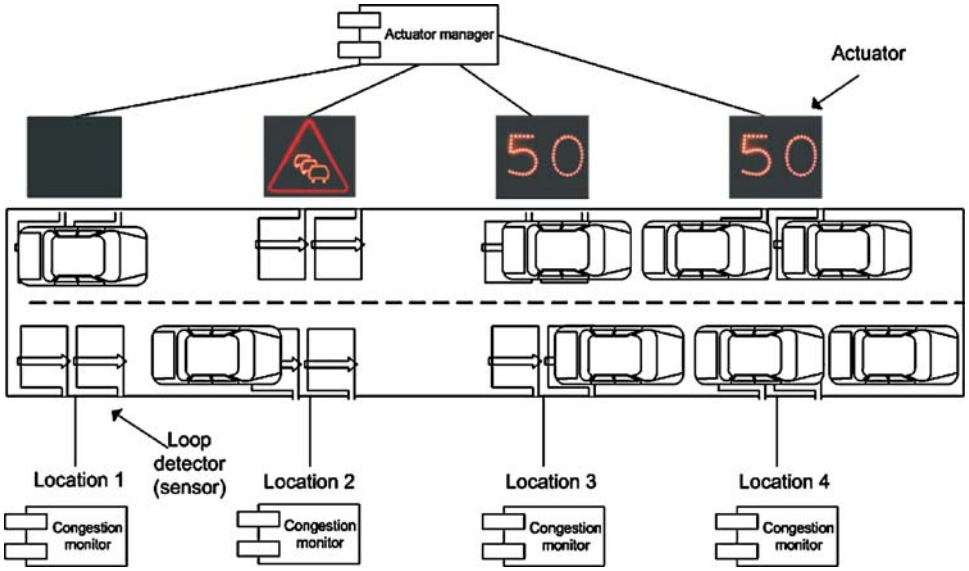

Fig. 2. Overview of the CWS system, from [FLM+05]

that the VDMTools interpreter has been greatly enhanced in recent years to support its deployment on large industry-scale test sets.

4. Enhancing the credibility of formal methods

A goal in our teaching is to engage students with the subject by showing relevance to the practice of systems and software development. We therefore try to motivate the topic with realistic examples. This serves to give students confidence in the industrial applicability of formal techniques. Simple examples such as stacks and dictionaries have an important role in introducing particular technical issues, but they do not provide the students an appreciation of the ability to apply the technique in real system development. Every major abstraction concept in our courses is immediately illustrated on an example derived from a real industry case study. Although we often have to simplify the case study so that it is not cluttered with irrelevant material, we can at least still provide information about the original industrial application.

Example: a congestion warning system

To illustrate our approach to the use of industrial-scale examples, we consider a traffic congestion warning system (CWS). This model originates from ongoing work carried out in The Netherlands by a specialist information systems consultancy in collaboration with the Dutch government transport ministry. The full example is described in VDM++ in [FLM+05]; here we concentrate on its use within the course at IHA.

A CWS (Fig. 2) warns drivers of areas of high traffic density, suggesting that they reduce speed to minimise the risk of collision. Data from a variety of sensors, including video, loop detectors, and even human observers, is used to determine whether congestion has occurred. Signals are sent to actuators such as roadside signs to convey information to drivers. Such signals obviously need to be consistent and coherent.

In lectures at IHA, a CWS system model is built in a top-down fashion during lectures about sequences and mappings. The overall structure of the CWS is developed in UML (an example class diagram is presented in Fig. 3). In our courses, UML class diagrams are treated as graphical overviews of textual VDM++ classes. For example, a fragment of the `CWS` class from Fig. 3 is shown in Fig. 4. Note how the ordered associations in the UML class diagram are represented as instance variables in the VDM++ class and how additional invariants can be constructed, ranging over multiple associations. In practice, the VDMTools-Rose link can be used to maintain consistency between these two views as either is updated [FLM+05].

As with the trusted gateway example, the CWS case has been adapted for teaching purposes. In class we recreate some elements of the industrial environment, including changing the requirements and updating the

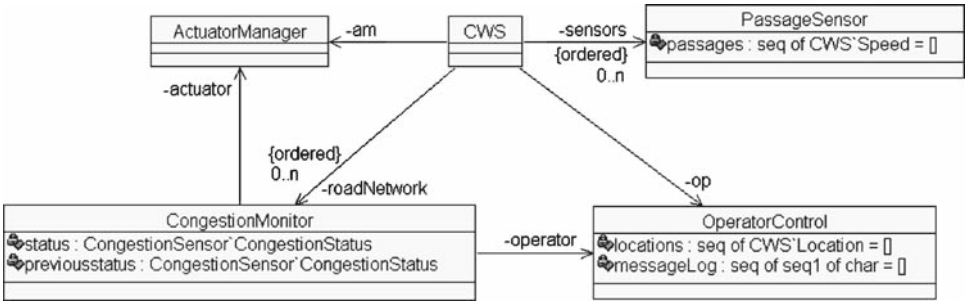

Fig. 3. UML class diagram for CWS, from [FLM^{+}05]

```
class CWS
...
instance variables
roadNetwork: seq of CongestionMonitor := [];
sensors    : seq of PassageSensor := [];
inv len roadNetwork = len sensors;

end CWS
```

Fig. 4. Congestion warning system: extract

VDM^{++} model accordingly. For example, we introduce a requirement to deal with multiple lanes on a highway, requiring the student to move from modelling using sequences to the use of mappings instead.

5. Course aims, structure and delivery

5.1. Course aims and syllabuses

The introductory courses at IHA and Newcastle aim to develop students' knowledge and skill in building and analysing suitably abstract models of computing systems. In addition, we aim to develop students' understanding of *why* formal models should be built and the issues involved in commercial applications of formal techniques. Basic modelling principles are taught using a combination of UML and VDM^{++}.

The IHA and Newcastle courses have similar structures. Following initial motivation, the elements of the modelling language are rapidly introduced in overview. The key abstractions are then introduced, starting with basic data and functional modelling, then moving on to the main collection abstractions (sets, mappings and sequences). Along the way, additional functional abstraction techniques including implicit (pre/post) specification and data type invariants are introduced. Each abstraction is illustrated on an industry-inspired example like the trusted gateway and CWS models described above. These presentations use material from Chapters 1–8 of [FLM^{+}05]. Although the focus is on modelling, rigorous approaches to analysis are introduced via the proof obligations.

5.2. Delivery methods

Both courses are presented over a 6–7 week period within one semester. At Newcastle, this amounts to about 24 h of lectures and 12 h of practical laboratory classes (with teaching support) in addition to private study. The IHA course[3] involves 24 h of lectures and 18 h of presentations by each group of students reporting on their practical work.

The difference in class sizes has to be taken into account in considering the delivery methods at each institution. In both IHA and Newcastle, we aim for a high level of interaction between lecturer and students. Typically, practical questions are presented on the slides immediately after a subject has been introduced. In addition, at

[3] http://kurser.iha.dk/eit/tivdm1/

IHA, with a smaller class size, the students know that in effect they will take turns in answering these questions. As a consequence they need to stay alert and active to be able to answer the next question they get. This has the consequence that the students learn the new concepts over a longer period than if they just prepare for the final examination. We believe that learning the topics in this fashion increases the retention rate.

At both IHA and Newcastle, students carry out practical coursework using VDMTools, to which they are introduced at a very early stage. Here they need to learn abstraction "by doing" it themselves on their projects. Although the projects carried out by the students are not large, they are all inspired by real systems and thus have more of a flavour of realism. This coursework element is particularly strong in the IHA course, where it forms the main part of the assessment. At IHA, students are offered a list of suggested projects. The current selection includes:

- SAFER (Simplified Aid For Extra Vehicular Activity Rescue) from [NAS97];
- the production cell from [LL95];
- a cash dispenser from [Gro05];
- CyberRail from [RTR06].

Students may also select projects from other MSc courses in distributed real-time and IT systems or suggest their own topic. The IHA course is mainly targeted at students studying the development of embedded software-based systems, so the topics are all reactive systems. Students in the most recent cohort selected all of the projects listed above, and three groups suggested their own projects based on a car radio navigation system, a self-navigating vehicle and a personal medical unit. Some of the projects already have substantive VDM models whereas for others only ideas are present and/or models are provided in different formalisms. Those students starting from existing models concentrate on analysing and modifying those models. Students selecting projects without existing models enjoy more freedom but face the challenge of choosing an appropriate level of abstraction.

Coursework at Newcastle accounts for 20% of the total course assessment. The students initially have to complete a short series of tutorial exercises working with basic data types and set, sequence and mapping abstractions, and interacting with VDMTools. In contrast to the choice of projects at IHA, at Newcastle the main exercise requires all students to understand and extend a given model. In recent years the model has taken the form of an automated supermarket checkout system, flight cargo management and an organisation's security access control protocol. In each case, students are given a basic model in which some abstraction decisions have already been made, and are asked to extend it by writing functions and extending data types. Typically the final part of the coursework will require the student to decide how to model a more complex extension: for example, in the access control model, students were asked to model the Least Privileges Problem ("Every program and every user of the system should operate using the least set of privileges necessary to complete the job"[SS75]). This gives students more opportunity to make their own abstraction decisions. With the large class size, having all students work on the same scenario permits an even level of support to all students, though it does reduce the scope for more innovative ideas from stronger students. The open-ended final question is intended to address this point.

5.3. Assessment

We are not concerned with assessing the students' ability to repeat language syntax from memory, without understanding the semantics. Rather, we wish to assess their deeper understanding of abstraction, rigorous system modelling and reasoning. Our assessment approaches for the two courses share this principle, but realise it in different ways.

The assessment of the IHA course is carried out by means of an oral examination. Students are required to prepare a presentation of no more than 10 min, to be delivered without multimedia support. They are also told to expect to be interrupted with questions. Each student is examined for 15 min and 5 additional minutes were used to decide upon the grade with the external examiner. The nature and difficulty of the questions asked is tailored to each student on the basis of the report submitted prior to the examination, and during the examination itself. Broadly speaking, the better the written and oral performance, the more demanding the questions. The examination attempts to cover as much of the curriculum as possible both in depth and in breath. The final grade is an aggregate of the assessments of the written report and the oral examination.

The Newcastle class is about four times the size of the IHA class, so a written examination is used, coupled with assessment of laboratory coursework. Currently, 80% of the final mark for the course is derived from the

Imagine that you are a member of a team commissioned to develop memory management functions for an operating system. In order to gain a better understanding of the management policy to be used, you have been asked to produce a formal model. An initial draft of the model in VDM-SL is shown here. Memory management is used to allocate partitions (contiguous regions of computer memory) to jobs scheduled to run in the system. Each job to be executed has an identifier. This is modelled by the `JobId` type defined as follows:

```
JobId = token;
```

The partitions allocated to jobs have a start location in memory. Locations are modelled as natural numbers. Each job has a certain size which represents the amount of memory required to execute the job. The system maintains two mappings recording job size and memory location. There is also an upper limit on the memory locations handled by the system. These concepts are modelled by the following type definitions:

```
Location = nat;
Size = nat;

JobLoc = map JobId to Location;
JobSize = map JobId to Size;

System :: jobs : JobLoc
          alloc : JobSize
          upperLimit : Size
inv sys == dom sys.alloc subset dom sys.jobs;
```

Jobs are allocated to a particular memory location by the function `allocate`, defined as follows:

```
allocate: System * JobId * Location -> System
allocate(mk_System(jobs, alloc, lim),j, l) ==
   mk_System(jobs, alloc munion {j |-> l}, lim)
pre j not in set dom alloc;
```

There are also further constraints on the system, such as:

1. Jobs allocated must fit within the upper memory limit
2. Jobs must not overlap

Further functionality to be added to the model will include:

- De-allocating a job from memory
- Providing a memory address where a given job will fit
- Managing the scheduling of jobs

Note that neither of these lists is intended to be exhaustive.

Fig. 5. Examination scenario

written examination, 20% from the coursework (to pass, students must obtain passing marks separately in both components). At first glance, this is a very conventional arrangement. However, the examination is quite unconventional. First, it is "open-book", meaning that students may consult printed or written materials. Second, it is based on a scenario that students receive some weeks ahead of the examination. An example scenario is in Fig. 5.

Students have every opportunity to read the scenario, anticipate questions and prepare answers which they can validate through VDMTools. Since the examination is open-book, these prepared answers can be consulted during the examination itself. Students are presented with previously unseen questions in which they are asked to comment on, explain, extend and revise the model described in the scenario. This scheme allows us to ask some gentle questions that the students might reasonably have anticipated, but also deeper questions requiring extensions to the model, for example, that we could not ask in a conventional closed-book examination. In our example scenario, basic questions could include asking students to provide comments explaining the data type invariant on `System`, or formalising the constraints. After asking students to explain the purpose of the precondition on the `allocate` function, we can ask if the function respects the invariant (checking understanding of proof obligations) and to modify it appropriately. Many of these questions could be anticipated by a well prepared student. The more challenging question would ask the student to extend the model with specifications of priority-based queueing for jobs, for example.

6. Experiences, evaluation and results

We consider the students' evaluation of the courses as well as their performance in assessments. The IHA course has been offered twice, while the Newcastle course has been running for three years in its current form,

and for about ten years in previous incarnations. Both institutions gather student feedback via anonymous questionnaires asking students to assess the difficulty of the course and its relevance to the degree programme. Evaluations at both institutions have been encouraging. Student evaluation is high on relevance to the degree programme (all students rating this good or very good at IHA; 80% of students rating this satisfactory or high at Newcastle). There is a broader spread of opinion on difficulty, about a third of IHA students and around half of Newcastle students finding their course challenging. Bearing in mind that the Newcastle course is compulsory for all BSc Computing Science students, this evaluation is broadly typical of Computing Science courses at present. In their free comments, students have been particularly positive about clarity and precision in the teaching (some finding the presentation too slow), and about the practical tool-supported approach taken.

In Sect. 3.1 we remarked that it can be difficult to engage students in studying a language that is not regarded as mainstream. In student evaluations, we have had few comments on the perceived relevance or irrelevance of the material to professional practice. We intend to track this in future by including a specific question in the evaluation.

The practical projects carried out by students provided experience producing and analysing abstract VDM^{++} models independently. Projects building on a pre-existing model were less challenging because an appropriate level of abstraction had already been found, making them suitable for weaker students. On the other hand, projects in which students had to create the model *ab initio* provided an opportunity to explore and practise a wider range of modelling elements and techniques covered in the courses. Such projects, crucially, provided opportunities for students to make, and learn from, abstraction errors.

At IHA the students in each of the two cohorts taught so far have performed well at the examination, well above average compared with other courses. In Newcastle the distribution of examination performance has been typical for compulsory courses.

Bearing in mind the student evaluations and performance in assessments, there are several changes that we will consider for future offerings of both courses. At IHA, no specific training was given in the use of VDMTools and Rational Rose. It turned out that a little more practical introduction to VDMTools, as provided at Newcastle, would have been an advantage. Our plans for the next academic year include conducting a few small practical exercises with VDMTools together with the students during the first week of the course to accommodate this.

During the IHA course, strict progress checks were imposed on the group projects, with one presentation on progress made by each group per week. Communication between the different groups was encouraged by asking the groups to give each other feedback on their presentations. Unfortunately some of the reports are in Danish so it will be of only limited use to international teachers. We will consider making the use of English mandatory in future.

Higher-level courses

IHA students have a further optional course on embedded and real-time systems, building on their introduction to formal modelling. This was offered for the first time in October 2006 and included:

- Model Structuring and Combining Views (Chaps. 9 and 10 from [FLM+05])
- Concurrency in VDM^{++} (Chap. 12 from [FLM+05])
- Real-time modelling in VDM^{++} over 2 weeks
- Distributed systems in VDM^{++}
- Model Quality (Chap. 13 from [FLM+05])

The real-time and distribution subjects were taught using new primitives being added to VDM^{++} in current research [Gro06, VLH06]. In fact, the students have become the first users of tool support for this extended language. The new course was well received, with some student support for making it a required successor to the modelling course.

At Newcastle between one third and one half of students taking the first modelling course choose to take a further optional 10 ECTS credit course in their final year. This focusses on the achievement of *correctness*, via a spectrum of formal and informal, static and dynamic approaches. This course builds on students' abstraction and modelling skills to introduce more advanced skills in formal verification, refinement, annotation-based design by contract, and static analysis. This latter course is not focussed on a single formalism, but uses Hoare Logic, VDM for refinement and Java Modelling Language for static analysis. The underlying goal is to equip the next generation of software tools developers with an understanding of the principles that underpin static and

dynamic analysis of models, designs and code. A particular aim is to develop students' critical skills in appraising the costs and benefits of new tools as they emerge.

7. Concluding remarks

We have presented a practical approach to the lightweight use of an established formal method as a means of developing abstraction and rigorous analysis skills among Computing Science students. The principles underpinning the approach relate to the use of accessible formal notation, supported by strong tools and introduced via examples derived from industrial practice. The approach has been realised in two introductory courses in separate institutions, within different institutional frameworks in small and large classes, in optional and compulsory courses. We believe that a pragmatic approach to introducing lightweight use of formal techniques with plenty of hands-on experience is a way to stimulate interest in this technology and to establish its credibility.

Our courses, although delivered in a higher education setting, have their origins in industry training. Such training is improved if the methods taught are rapidly applicable [LCD04]. In the academic context, the focus must be more on deeper long-term skills that are independent of the particular formalism, but we believe that this has to be balanced with the need to encourage student engagement by using an established, tool-supported formalism.

Understanding student motivation is important. As Reed and Sinclair [RS04] have suggested, many students are driven by the need to develop skills useful to them in a subsequent career, rather than a fascination for the beauty of computing science. Many, in our experience, are also driven by the laudable desire to create good software and see it run. Students may view formal methods as an obstacle to this; our lightweight use of formalism is an attempt to deal with this tension.

Although our examples are inspired by industrial applications, we recognise a need to demonstrate the practical effect of formal modelling on the development of software. One approach is to make the formal link by teaching refinement (a topic in the Newcastle advanced course discussed above). Another is to introduce students to the literature evaluating formal techniques in practice. Our trusted gateway example is derived from one of very few studies comparing a software development done with and without formal modelling [FBGL94, BFL96]. Other industry application reports, including information on effort profile and defect rates, are reported in the student texts and the literature, e.g., [FLM^{+}05, FL07, FL07b].

We agree with Boute [Bou03] that the concept of lightweight formal methods may be mis-used as a means of lowering the threshold of mathematical skill needed to develop good computing systems. We make a point of requiring specific mathematical skills. Indeed, we would argue that the formalism used is not compromised in our courses.[4] By "lightweight" we mean sharply focussing the formalism on particular system concerns (abstraction), in the manner suggested by Jackson and Wing [JW96].

To revisit our opening question: why do we teach formal methods? Our response was that the study of formal methods develops skills, notably those of abstraction and rigorous analysis, that will be of use beyond the immediate course in which they are taught. Sobel's study [SC02], albeit the subject of a debate on experimental design [BT03, SC03], is a first attempt to assess whether a training in formal techniques may improve students' general analytic and problem solving skills.

Are our students better at abstraction in general as a result of taking our courses? We believe that few teachers could answer this question convincingly. Kramer and Hazzan have recently argued that abstraction skills are core to computing and that we should try to develop abstraction skills explicitly through students' development [Kra07]. It has been pointed out that we lack tests to gauge abstraction skills, as most relevant tests focus on logical reasoning. Hazzan [HK07] presents patterns of exercises, each intended to encourage students to consider abstraction explicitly. We intend to adapt these to our courses, using them as a basis for encouraging students to think consciously about abstraction, and also to give us an informal appraisal of the development of abstraction skills. For example, one pattern of exercise presents students with several alternative models of the same underlying system, asking them to identify which are more or less abstract than the others. Such an exercise encourages students to consider what constitutes an abstraction. In the context of our VDM courses, in which we stress the importance of the model's purpose in governing the selection of abstractions, we would also ask students to comment on the suitability of different models for assessing different system properties.

[4] Bernhard Steffen has suggested that we have not introduced a lightweight formal method. The specific weight of VDM remains the same—we have merely provided tools for lifting it!

In an alternative exercise, we might ask students themselves to construct models at different levels of abstraction. In future work, we aim to devise an evaluation that will help us determine the extent to which formal methods skills are truly "transferable".

Acknowledgments

We are grateful to our colleagues Erik Ernst, Jozef Hooman, Troels Fedder Jensen, Hugo Macedo, Marcel Verhoef, Stefan Wagner, Jeremy Bryans and the anonymous referees for their valuable comments. This work has been partly supported by the European Union's Framework 6 Network of Excellence on Resilience in IST (ReSIST).

References

[And96] Andrews DJ (ed) Information technology—Programming languages, their environments and system software interfaces—Vienna development method—specification language—Part 1: Base language. International Organization for Standardization, December 1996. International Standard ISO/IEC 13817–1

[AS99] Agerholm S, Sunesen K (1999) Reasoning about VDM-SL proof obligations in HOL. Technical report, IFAD

[BFL+94] Bicarregui JC, Fitzgerald JS, Lindsay PA, Moore R, Ritchie B (1994) Proof in VDM: a practitioner's guide. FACIT. Springer, Heidelberg

[BFL96] Brookes TM, Fitzgerald JS, Larsen PG (1996) Formal and informal specifications of a secure system component: final results in a comparative study. In: Gaudel M-C, Woodcock J (eds) FME'96: industrial benefit and advances in formal methods. Springer, Heidelberg, pp 214–227

[Bou03] Boute RT (2003) Can lightweight formal methods carry the weight? In: Duce DA et al (eds) Teaching formal methods: practice and experience 2003. Oxford Brookes University. Available at http://cms.brookes.ac.uk/tfm2003/

[BT03] Berry DM, Tichy WF (2003) Comments on "Formal methods application: an empirical tale of software development". IEEE Trans Softw Eng 29(6):567–571

[ELL94] Elmstrøm R, Larsen PG, Lassen PB (1994) The IFAD VDM-SL Toolbox: a practical approach to formal specifications. ACM Sigplan Notices 29(9):77–80

[FBGL94] Fitzgerald J, Brookes TM, Green MA, Larsen PG (1994) Formal and informal specifications of a secure system component: first results in a comparative study. In: Denvir BT, Naftalin M, Bertran M (eds) Formal methods Europe'94: industrial benefit of formal methods. Lecture notes in computer science, vol 873. Springer, Heidelberg, pp 35–44

[FL98] Fitzgerald J, Larsen PG (1998) Modelling systems—practical tools and techniques in software development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK. ISBN 0–521–62348–0

[FL07b] Fitzgerald JS, Larsen PG (2007) Balancing insight and effort: the industrial uptake of Formal methods. In: Jones CB, Liu Z, Woodcock J (eds) Formal methods and hybrid real-time systems, essays in Honour of Dines Bjørner and chaochen zhou on the occasion of their 70th birthdays. Lecture notes in computer science, vol 4700, Springer, Heidelberg, pp 237–254. ISBN 978-3-540-75220-2

[FL07] Fitzgerald JS, Larsen PG (2008) Triumphs and challenges for the industrial application of model-oriented formal methods. In: Margaria T, Philippou A, Steffen B (eds) Proc. 2nd intl. symp. on leveraging applications of formal methods, verification and validation. Also Technical Report CS-TR-999, School of Computing Science, Newcastle University

[FLM+05] Fitzgerald J, Larsen PG, Mukherjee P, Plat N, Verhoef M (2005) Validated Designs for Object-oriented Systems. Springer, New York

[FLT+07] Fitzgerald JS, Larsen PG, Tjell S, Verhoef M (2007) Validation support for real-time embedded systems in VDM++. In: Cukic B, Dong J (eds) Proceedings of HASE 2007: 10th IEEE high assurance system engineering symposium, pp 331–340. IEEE

[FL08] Fitzgerald J, Larsen PG, Sahara S (2008) VDMTools: advances in support for Formal modeling in VDM. Sigplan Not (submitted)

[Gro05] The VDM Tool Group (2005) A "Cash-point" service example. Technical report, CSK, June 2005. http://www.vdmportal.org/twiki/pub/Main/VDMPPexamples/cashdispenser_a4.pdf

[Gro06] The VDM Tool Group (2006) Development guidelines for real time systems using VDMTools. Technical report, CSK

[HK07] Hazzan O, Kramer J (2007) Abstraction in computer science and software engineering: a pedagogical perspective. Front J 4(1):6–14

[IH89] Jones CB, Hayes IJ (1989) Specifications are not (necessarily) executable. Softw Eng J 330–338

[Jon90] Jones CB (1990) Systematic software development using VDM 2nd edn. Prentice-Hall International, Englewood Cliffs

[Jon96] Jones CB (1996) A rigorous approach to formal methods. IEEE Comput 29(4):20–21

[JW96] Jackson D, Wing J (1996) Lightweight Formal Methods. IEEE Comput 29(4):22–23

[Kra07] Kramer J (2007) Is abstraction the key to computing? Commun ACM 50(4):37–42

[LCD04] Loomes M, Christianson B, Davey N (2004) Formal systems, not methods. In: Dean CN, Boute RT (eds) Teaching formal methods. Lecture notes in computer science, vol 3294. Springer, Heidelberg, pp 47–64

[LFB96] Larsen PG, Fitzgerald JS, Brookes T (1996) Applying formal specification in industry. IEEE Softw 13(3):48–56

[LL91] Larsen PG, Lassen PB (1991) An executable subset of Meta-IV with loose specification. In: VDM'91: formal software development methods. VDM Europe, Springer, Heidelberg

[LL95] Lewerentz C, Lindner T (eds) (1995) Formal development of reactive systems: case study production cell. LNCS, vol 891. Springer, New York

[NAS97] NASA (1997) Formal methods, specification and verification guidebook for verification of software and computer systems. A Practitioner's Companion. Technical Report NASA-GB-001-97, vol 2. Washington, DC 20546, USA, May 1997. Available from http://eis.jpl.nasa.gov/quality/Formal_Methods/

[Pep04] Pepper P (2004) Distributed teaching of formal methods. In: Dean CN, Boute RT (eds) Teaching formal methods. Lecture notes in computer science, vol 3294. Springer, Heidelberg, pp 140–152

[PO04] Paige RF, Ostroff JS (2004) Specification-driven design with Eiffel and agents for teaching lightweight formal methods. In: Dean CN, Boute RT (eds) Teaching formal methods. Lecture notes in computer science, vol 3294. Springer, Heidelberg, pp 107–123

[RS04] Reed JN, Sinclair JE (2004) Motivating study of formal methods in the classroom. In: Dean CN, Boute RT (eds) Teaching formal methods. Lecture notes in computer science, vol 3294. Springer, Heidelberg, pp 32–46

[RTR06] RTRI (2006) The Concept of CyberRail. http://cyberrail.rtri.or.jp/english/

[SC02] Kelley Sobel AE, Clarkson MR (2002) Formal methods application: an empirical tale of software development. IEEE Trans Softw Eng 28(3):308–320

[SC03] Kelley Sobel AE, Clarkson MR (2003) Response to "Comments on 'Formal methods application: an empirical tale of software development". IEEE Trans Softw Eng 29(6):572–575

[SS75] Saltzer JH, Schroeder MD (1975) The protection of information in computer systems. Proc IEEE 63(9):1278–1308

[UR01] Utting M, Reeves S (2001) Teaching formal methods lite via testing. J Softw Testing Verif Reliab 11(3):181–195

[VL07] Verhoef M, Larsen PG (2007) Interpreting distributed system architectures using VDM^{++}—a case study. In: Sauser B, Muller G (eds). Proceedings of 5th annual conference on systems engineering research. Available at http://www.stevens.edu/engineering/cser/

[Ver07] Vermolen S (2007) Automatically discharging VDM proof obligations using HOL, Radboud University Nijmegen, computer science department

[VLH06] Verhoef M, Larsen PG, Hooman J (2006) Modeling and validating distributed embedded real-time systems with VDM^{++}. In: Misra J, Nipkow T, Sekerinski E (eds) FM 2006: formal methods. Lecture notes in computer science, vol 4085. Springer, Heidelberg, pp 147–162

Received 26 March 2007

Accepted in revised form 6 November 2007 by D. A. Duce, J. Oliveira, P. Boca and R. Boute

Published online 2 February 2008