A Lightweight Approach to Formal Methods*

Sten Agerholm Peter Gorm Larsen

IFAD, Forskerparken 10, DK-5230 Odense M, Denmark
E-mail: {sten,peter}@ifad.dk Web: http://www.ifad.dk

Abstract. The main current trend in applied formal methods can be characterized by the term "lightweight". Historically, formal methods have been viewed as pure alternatives to traditional development methodologies, demanding a revolutionary change in industry to adopt them. With a pragmatic, lightweight approach, the use of formal methods is complementing and improving existing development practices in a company in an evolutionary way, demonstrating more clearly the cost-effectiveness of formal methods. This paper presents our view on lightweight formal methods as a strategy for successful formal methods technology transfer to industry.

1 Introduction

Initially formal methods were promoted aggressively by idealistic academics. Formal methods were seen as *the* solution to the "software crisis" and implied the uncompromising use of mathematics and rigor for the whole development life cycle where entire software systems were developed from scratch. Hence, the development process should start with an abstract specification of the system and proceed by formal refinements and proofs of correctness towards a final implementation. From an industrial point of view this is not realistic because 1) systems are rarely developed from scratch, 2) only parts of the systems would benefit from a formal model, and 3) the general skill level in industry is not adequate to cope with the techniques for fully formal development. Many companies would be reluctant to throw away current practice and investments required by such a revolutionary approach.

Rather than focusing entirely on the ambitious certainty of correctness, the formal methods communities are starting to loosen up and find lightweight approaches to applying their technologies. For example, this new trend was documented recently by Jones [15], Jackson and Wing [14], and by Easterbrook et al. [6]. The authors use the term "light" or "lightweight" in the sense of "less-than-completely formal" or "partial" where the methods can be used to perform "partial analysis on partial specifications, without a commitment to developing and baselining complete, consistent formal specifications" [6].

With this approach, formal methods are used more as a defect detection technique in the early stages of the software development life cycle. Here, the

* To appear in *Proceedings of the International Workshop on Current Trends in Applied Formal Methods*, Boppard Germany, LNCS, Springer-Verlag, October 1998.

abstraction inherently associated with formal modeling is a powerful means of reducing complexity and improving a development team's understanding of the requirements. The motivation for the early application of formal methods is twofold. Firstly, it is commonly agreed that the expensive mistakes are made in a project's early stages [15, 11]. Secondly, current requirements engineering practice is often based on ad hoc methods and natural language specifications with no or little tool support [11, 6]. In contrast, formal notations provide a structured way of expressing requirements, which makes inspections more effective and reliable, and allow different sorts of checking to be performed by automation tools.

However, the lack of proper tools has often been claimed to be a main obstacle to the industrial take-up of formal methods. Many existing tools are merely prototypes with academic aspects such as Emacs and LaTeX. For proper take-up, tools must have industrial quality, be based on industrial platforms such as Windows, and support document generators such as Word. We have already taken this step with the IFAD VDM Tools [12, 7, 18].

The IFAD VDM Tools are particularly geared towards a lightweight approach. The key here is the focus on executable specifications which makes it easier for traditional engineers to produce and analyze formal models in a familiar testing framework. The tools also enable users to use a combination of conventional graphical techniques such as UML together with formal models described in VDM++, which is an object-oriented extension of the ISO Standard formal specification language VDM-SL [13, 10]. Further, emphasis is put on the capabilities for interaction between formally specified parts and parts of systems which are developed traditionally. Thus the IFAD VDM technology is easy to adapt by traditional software developers in an evolutionary way. The main obstacle is to teach the engineers how to choose which parts to model and how to make appropriate abstractions of these parts. In our courses we focus exactly on these aspects. It is our experience that because of the emphasis on hands-on usage of tool support with testing facilities to try out the semantics of new constructs, even engineers with no formal methods background are capable of writing appropriate VDM models after a one week course.

In this paper we provide an overview of the pragmatic, lightweight approach that we advocate to successfully introduce formal methods in an industrial setting. We focus on the requirements capturing part of the life cycle. We demonstrate how abstract models can be used as complements to graphical models. The main focus is on how such precise models can be validated and which kind of design errors can be captured with this approach. Finally, we report on some of the success stories we have had in transferring this approach to industrial end-users and give some concluding remarks on how we see the future for this trend.

2 Modeling Systems

This section first places formal modeling in a context of the software life cycle. It continues to present pointers to previous work on formal modeling using

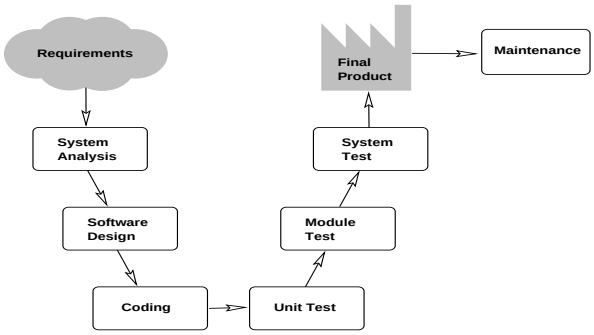

Fig. 1. Phases of the software life cycle

the VDM formal specification languages and on the use of these languages in conjunction with graphical modeling notations. This leads up to Sect. 3, where the same examples are used to illustrate the lightweight validation techniques facilitated by the IFAD VDM Tools. More introductory and better examples of abstract formal modeling can be found in [10].

2.1 Software Life Cycle

Many organizations use a development process which is a variant of the V life cycle model presented in Fig. 1. Starting from requirements, the development process proceeds through a number of phases until a final product is delivered. Then, the maintenance phase begins.

The traditional approach to the V life cycle model has a number of problems. A central one is that errors are discovered too late, when they are expensive to fix. This can result in overly late delivery and badly structured software due to last minute error patching. As stated in [11], the early stages of the software life cycle are especially prone to errors that can have a lasting influence on reliability, costs and safety of a system.

Essentially, the current methods used in the early stages of development are too ad hoc, as they are based mainly on natural language specifications and informal or semi-formal diagrammatic notations. These make inspections woolly and not so useful in locating errors, in contrast to, for example, code inspections [15]. Moreover, such descriptions do not allow automation tools, so the reviewing process becomes a manual process that is subject to human limitations and costly to repeat when requirements change.

Formal modeling is viewed as a way of improving the system analysis phase. It can help to achieve a better understanding of requirements and to reduce risk in the development process by finding defects early. The following aspects of formal modeling are considered important:

Abstraction: The abstraction supported by formal specification languages can help to cope with the complexity of a system to be developed. Abstract mod-

els are built by focusing on a specific purpose of the analysis and including only the relevant aspects of the system in the model being built. Russell said: "Abstraction, difficult as it is, is the source of all practical power."

Precise modeling: Formal notations can help to present the requirements in a structured manner, which makes reviewing and inspection easier and therefore useful in locating errors.

Tools: Formal notations allow tools to be developed for automating the analysis and inspection. Examples range from basic syntax and type checkers to interpreters for executing specifications and automated theorem provers.

Section 3 discusses the use of tools to validate formal models.

2.2 Modeling in VDM-SL

A formal specification language such as VDM-SL supports the development of abstract descriptions of data and functional behavior of a system. Its data type constructors for sets, sequences and mappings, and their associated constructions such as comprehensions and first-order quantifications, let the engineer concentrate on "what the system must do, not how it should do it" [16]. Moreover, the engineer can document desired properties explicitly in the model through the use of invariants and pre- and post-conditions.

Metro Door Management Example In [3], a VDM-SL model of a metro door management system was presented. The main safety requirement of such a system is that the doors must not be open at any time while the train is moving, which is essential for the safety of the passengers.

By focusing the purpose of the analysis on ensuring that a model satisfies this requirement, we can reduce the complexity of a metro system drastically. The central components of the system are the train and the doors, modeled as a record type definition in VDM-SL:

```
Metro :: train: Train
         doors: Doors;
```

The only thing we need to know about the train and the doors is their state, modeled as the following enumeration type definitions:

```
Train = <stopped> | <moving>;

Doors = <open> | <closed> | <halfopen>;
```

(This model is a cut-down version of the one presented in [3].)

The safety requirement can be formalized and documented in the model directly, by introducing an invariant on the metro record type above:

```
Metro :: train: Train
         doors: Doors
inv metro ==
  not(metro.train = <moving> and metro.doors <> <closed>);
```

As we shall see in Sect. 3, such a property is checked automatically in executing the specification.

The model provides a very state diagram oriented view on the metro system. In fact, the model was inspired by some OMT state diagrams for the train and the doors, and the main functions of the VDM-SL model formalize the transitions defined in these diagrams (see Sect. 2.3). For example, the doors transition function is defined along these lines:

```
DoorsTransition: Doors * Button * Train -> Doors
DoorsTransition(doors,butt,train) ==
  cases doors:
    <open>     -> if butt = <do_close>
                  then <halfopen>
                  else <open>,
    <closed>   -> if butt = <do_open> and train = <stopped>
                  then <open>
                  else <closed>,
    <halfopen> -> ...
  end;
```

The VDM-SL model contributed a more rigorous view than OMT, e.g. through the use of the invariant and an executable model. Surprisingly, we were able to break the invariant in validation (see Sect. 3). VDM-SL would support more abstract, less state oriented models of the metro system.

SAFER Example In [2], we presented a VDM-SL model of the SAFER system, which is a small, lightweight propulsive backpack module designed to provide self-rescue capabilities to a NASA space crewmember. It is used in emergency situations during an extravehicular activity if a crewmember becomes separated from a space station or a space shuttle orbiter docked to a space station.

The SAFER system was described in the NASA Formal Methods Guidebook [19], which included both the informally stated requirements and a PVS model of the thruster selection functionality. In response to translation and rotation commands issued by a crewmember via a hand controller module, and in response to rotation commands issued by an automatic attitude hold capability (AAH) which a crewmember can activate to detumble himself, the SAFER software must select a certain number of gaseous nitrogen (GN_2) thrusters to fire, out of possible 24. We essentially translated the PVS model to VDM-SL in order to show how validation by testing could be viewed as a complement to verification by proof used in the guidebook (see Sect. 3). The purpose of the model is to check that the thruster selection functionality works properly, and thus, for example, the actual calculation of the AAH output is outside the scope of the model (following the Guidebook).

As the SAFER example is too big to be presented at length in this paper, we shall only discuss one important aspect of our model related to two main safety requirements of the SAFER system listed in [19].

Property (41): The avionics software shall provide accelerations with a maximum of four simultaneous thruster firing commands.

Property (E1): Thruster firing consistency: No two selected thrusters should oppose each other, i.e., have canceling thrust with respect to the center of mass.

A central operation in this respect is the control cycle, which reads the crewmember commands and the AAH output command as inputs, and calculates the thruster settings. By defining a post-condition on the control cycle we are able to document these properties in our model:

```
ControlCycle: HCM`HandGrip * AUX`RotCommand * ... ==>
              set of TS`ThrusterName
ControlCycle(raw_grip,aah_cmd,...) ==
  let grip_cmd  = HCM`GripCommand(raw_grip,...),
      thrusters = TS`SelectedThrusters(grip_cmd,aah_cmd,...)
  in
     (AAH`Transition(grip_cmd,clock,...);
      clock := clock + 1;
      return thrusters)
post card RESULT <= 4 and ThrusterConsistency(RESULT)
```

The specification is modular, where back quote is used to prefix with a module's name. First the raw hand grip command is converted to a combined translation and rotation command and then the thruster settings are calculated. The control cycle maintains a clock and a state in the automatic attitude hold module, before the set of thruster settings is returned. The post-condition uses an auxiliary function for thruster consistency, defined as follows:

```
ThrusterConsistency: set of TS`ThrusterName -> bool
ThrusterConsistency(thrusters) ==
  not ({<B1>,<F1>} subset thrusters) and
  not ({<B2>,<F2>} subset thrusters) and
  not ({<B3>,<F3>} subset thrusters) and
  not ({<B4>,<F4>} subset thrusters) and ...
```

Section 3 shows how validation is used to achieve a high confidence that the model satisfies the post-condition.

2.3 Combining Graphical and Formal Modeling

Despite all benefits, formal models can be complicated to organize and structure during development. Often diagrammatic views, e.g. in object-oriented modeling, can help to master the complexity of software systems. Graphical notations can give an abstract, structural overview of a model, which can significantly aid in a developer's understanding of the model. Hence, we believe that graphical notations such as UML are particularly well-suited for making the first sketches of an object-oriented model of a software system, presenting and visualizing the

main components of a system and the relations between them in a high level way. Moreover, graphical notations are becoming widely used in industry and so can be an entry point for formal methods application in industrial practice.

Metro Example In the metro example presented above, OMT was used to analyze the requirements prior to formally specifying a model of the system. As noted in [17], this approach can help to enhance the initial formal specifications and reduce the effort required to produce them, but we also felt that this made the focus of the model more low-level as it became very state oriented. The example also showed an instance of the misunderstanding of a graphical model, because OMT makes an implicit assumption that events cannot happen simultaneously. The translation to VDM-SL did not respect this semantics, which made it possible to violate the main safety requirement (see Sect. 3).

SAFER Example In a recent paper [5], the SAFER example has been reworked in an alternative, object-oriented development rather than the modular approach in [19, 2]. This makes use of graphical modeling using UML in conjunction with formal modeling using VDM++, an object-oriented extension of VDM-SL. The overall object-oriented aspects of SAFER are first modeled in a UML object model, which is translated to VDM++ for further analysis by adding concise type information and functional behavior to the model. The transition forward and backward between the two notations is supported by the Rose-VDM++ Link [4]. This provides round trip engineering capabilities to the user by integrating Rational Rose and the IFAD VDM Tools. In this way, the complementary benefits of graphical and formal modeling can be exploited effectively, as visualization becomes a direct and key element of the modeling process, where the graphical notation presents a structural overview and the formal notation enables a rigorous analysis by automation tools.

3 Validating Models

Though experience shows that valuable insight can be gained already by writing a formal model [16, 6], formal modeling alone does not provide a clear enough argument for industry's adoption of formal methods. It has to be accompanied by professional, quality tools, not to mention training and consultancy, which are targeted at the current skills of traditional engineers. By incorporating formal methods into the software practice in small deltas [9], the technology transfer process is likely to have a more significant effect.

Therefore we propose the use of testing techniques to validate formal models. In industry, these techniques are already commonly used and known to be effective tools in finding defects. Hence, there is a much clearer cost-benefit argument of this approach than with traditional formal methods, since it is commonly agreed that it is desirable and cost-saving to find errors early in the life cycle.

Moreover, the test cases applied to the formal model can be reused on the final implementation, providing a strong argument for conformance of the code

with respect to the requirements. In this way, the formal model acts as an oracle against which the implementation is tested in an automated way, complementing traditional code inspections and perhaps reducing the effort and cost required to conduct these.

3.1 Executing Specifications

Executable specifications are a prerequisite for applying validation using testing techniques. Another essential thing is a tool for performing the execution, as clearly hand-execution is not feasible.

A large subset of VDM (i.e. VDM-SL and VDM++) is executable and supported by an integrated interpreter and debugger of the IFAD VDM Tools. The interpreter can be used interactively by feeding it with expressions or script files, but it can also be used in batch mode for evaluating large test suites.

Test Expressions The interpreter can read and execute any VDM expression in the executable subset. These include trivial expressions like `5+3` and function calls, and expressions such as the map comprehension expression

```
{mk_(a,b,c,d,m) |->
   SAFER`ControlCycle(mk_HCM`HandGrip(a,b,c,d),...,m,...) |
 a,b,c,d in set {<zero>,<pos>,<neg>}, m in set {<tran>,<rot>}}
```

which in a compact way provides a kind of test case generation expression [2]. The above expression executes the SAFER control cycle 162 times and returns a mapping from inputs to the corresponding outputs.

Test Statistics and Test Coverage During execution the interpreter collects a number of test statistics and it can be asked to show test coverage information in Word (or LaTeX) documents, using multiple fonts or coloring to distinguish the executed and non-executed parts. Such information can be an aid in designing test cases, but can also aid in locating errors in a specification. This was illustrated in [5], where the test statistics said a certain function was executed only 6 times with a coverage of 81%, while we expected 81 executions with 100% coverage. The test coverage coloring led us on the right track for detecting the error, and the correction was easy.

In the SAFER example [2], we used the test coverage coloring to increase our confidence that certain non-applicable cases stated in large tables of the requirements document were indeed never executed. Using a map comprehension as above, the control cycle was executed 8748 times, yielding a coverage of only 72% of one of the tables, though executed in every cycle. The tables were represented as functions in VDM-SL using cases expressions of the form:

```
LRUDThruster(a,b,c) ==
cases mk_(a,b,c):
  mk_(<neg>,<neg>,<zero>) -> mk_({},{}),
```

```
    mk_(<neg>,<zero>,<zero>) -> mk_({<L1R>,<L3R>},{<L1F>,<L3F>}),
    mk_(<pos>,<zero>,<pos>) -> mk_({<R2R>},{<R2F>,<R4F>}),
    ...
end;
```

Here, the test coverage facility shows the non-executed parts in boxes, and these correspond exactly to the non-applicable cases.

3.2 Consistency Checking during Execution

While specifications are executed, the interpreter checks consistency of assumptions and requirements documented in invariants and pre- and post-conditions. This can be an aid in finding bugs or gaining confidence in models when a large enough set of test cases is executed successfully.

Metro Example: Invariant Checking In the metro model, we used an invariant to state the main safety requirement of the system, saying that the doors should not be open while the train is moving. In several executions, the invariant was checked and found to be true. However, by defining and executing a certain scenario it was possible to break the invariant, reported as a runtime error by the interpreter. This happens in the state where the doors are closed and the train is stopped, when we ask the doors to open and the train to accelerate at the same time [3]. In this case, the doors will start to open and the train will start to move, which violates the invariant on the metro type.

The same situation cannot arise in the OMT state diagrams from which the VDM-SL model was strongly inspired. OMT operates with an idealized model where events can happen infinitely close, but not simultaneously. This implicit assumption was forgotten while translating the model to VDM-SL. This would yield an equivalent problem if the final code was produced from OMT directly.

SAFER Example: Post-condition Checking In the SAFER model, we used a post-condition on the control cycle to state the properties (41) and (E1), saying that at most four thrusters should be selected at the same time and that no two selected thrusters should oppose each other. This means that in all executions of the control cycle these properties are checked automatically. As mentioned above, we formulated a compact map comprehension expression yielding 8748 executions of the control cycle. These gave 189 different thruster settings as output, which all satisfied the post-condition. The large number of test executions gave us confidence in both the stability and conformance of our model with respect to the above mentioned requirements.

In the NASA Guidebook [19], verification by interactive theorem proving with PVS [21] was used to illustrate the analysis of property (41). The proof required considerably more skill to formulate and conduct than the automated analysis with execution described above. However, the level of confidence obtained in the model as a result of verification by proof is higher than with validation by testing.

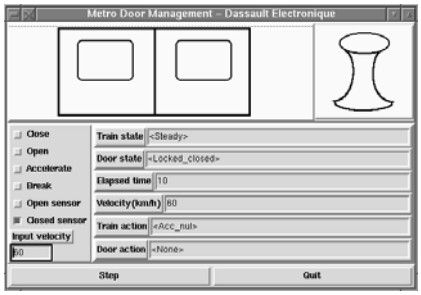

Fig. 2. Graphical front-end for metro door management model

Therefore, we propose to apply validation by testing first to achieve a certain level of confidence in a model, and then proceed by verification if demanded or beneficial. Moreover, we believe that engineers can use validation by testing cost-effectively early in their formal methods careers, but it is unlikely that the same holds for verification.

3.3 Integrating External Components

In an industrial setting, systems are rarely developed from scratch and will often consist of parts where formal modeling is not useful, e.g. a graphical user interface or database handling. In order to integrate formal modeling with the development process, it is therefore essential to allow the formal model to include external components, for example, to facilitate testing of the model in conjunction with already developed parts of a system. The IFAD VDM Tools provide such functionality through a Dynamic Link facility which allows a user to execute code together with a VDM model.

In addition to supporting the technology transfer process, the integration of external components into a model can support the assessment of a formal model by a customer or staff member not trained in VDM. By developing a graphical front-end and linking this with a model, a customer or colleague can "inspect" a model in an interactive way without reading the VDM, and thus get a better understanding of the specifier's view of the requirements. This use of formal modeling and the Dynamic Link facility enables rapid prototyping.

Metro Example Front-end In the metro case study, a graphical front-end was developed quickly. Though rather simple, it had great value for demonstration purposes, but also for the engineers themselves in order to ease the testing process. Once appropriate test scenarios were defined they were easy to enter through the graphical front-end.

Figure 2 presents one view of the metro door management front-end. The top part shows the metro doors and a bell which is used to warn passengers before doors start to close (this aspect was excluded from our description in Sect. 2.2).

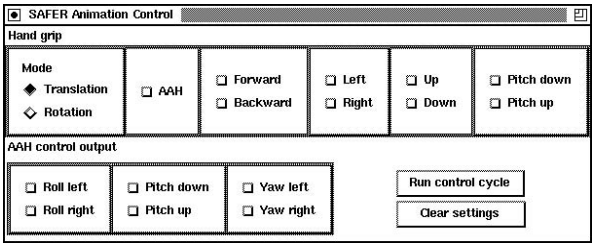

Fig. 3. Graphical model of the SAFER hand controller

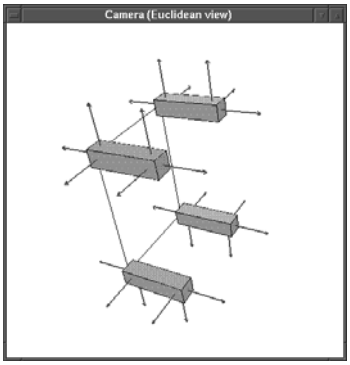

Fig. 4. Graphical model of the SAFER backpack

The left part shows the input to the VDM model, which are buttons operated by the driver, door sensors and the speed of the train. The right part shows output from the model and various status information. Hence, the front-end provides a visualization of the VDM model which a customer or colleague can immediately understand and interact with.

SAFER Example Front-end In the SAFER example, we developed a graphical model of the hand controller unit used by the crewmember to issue commands to the thruster control software (see Fig. 3). The front-end also includes a panel for inputting the rotation command of the automatic attitude hold (AAH), as its calculation was abstracted away from the VDM model.

Moreover, we developed a graphical model of the backpack module in order to visualize the thruster settings calculated by the VDM model. For illustration, Figure 4 shows all possible 24 thruster settings.

These figures could naturally be enhanced significantly, but the purpose here is just to illustrate the feasibility of the approach. For example, in testing the SAFER model we can simply enter commands via the hand controller front-end

rather than by writing complex input arguments for the SAFER control cycle. These are generated automatically by the code underneath the front-end.

The graphical front-ends were a major help in validating a number of properties mentioned in the requirements document for SAFER [19]. During this analysis process we found some interesting "unclarities" in the requirements by using the front-ends in a structured way to enter a number of test scenarios and immediately check the visualized results (see [2]).

4 Industrial Application Success Stories

This section provides some pointers to a few projects where the lightweight approach to formal methods has worked successfully as a strategy for formal methods technology transfer to industry.

4.1 Trusted Gateway by BASE

At British Aerospace (Systems and Equipment) a controlled experiment was carried out measuring the effects of introducing formal modeling of a small security-critical component known as the trusted gateway [16, 8]. The trusted gateway acts as a filter between a high-security system and less secure systems, ensuring that messages of high secrecy are not passed to systems deemed too insecure to handle them.

In this project a number of comparisons was made between two separate developments of the trusted gateway, one development using conventional practice, the other using VDM-SL in addition to this when considered helpful. The conventional techniques included the graphical notation SA in the Ward and Mellor dialect, without special support for the combination with VDM-SL.

The main conclusion from this project was that the use of formal modeling and VDM Tools in the early phases was beneficial and not more expensive than a conventional development. Moreover, the conventional development failed to handle an exceptional case which was clarified by the formal development in the system analysis phase. This error was not found until after the acceptance testing, and the patch for this made the software developed conventionally fourteen times slower than the software developed using VDM.

Based on the experience, the VDM technology was recommended in future projects at BASE. We know of at least three more projects that have been carried out using the VDM technology, but they have not needed our assistance in any of these projects. This is a clear indication that the technology transfer process has worked successfully.

4.2 Metro Door Management by Dassault Electronique

Dassault Electronique (D.E.) is a major French provider of safety-critical systems for terrestrial transportation, space, and military and commercial avionics. In these markets, the use of formal methods is often mandatory or encouraged,

for example, the French RATP (Régie Autonome des Transports Parisiens) mandates the use of the B method [1] for the development of safety-critical software for the Paris Metro.

While some French authorities encourage the use of the B method, D.E. find that B does not meet their needs early in the software development process, when requirements are often changing and may need to be clarified and agreed upon by customers. They evaluated the IFAD VDM Tools in a case study focusing on the door management system of a metro. The results were presented in [3], and have been used for illustration throughout this paper.

The biggest surprise to Dassault Electronique was that they were able to master the VDM technology so quickly. After only three days of training in a one-week training course, they were able to both write and do the initial analysis of their first model of the door management system, based on the informal and semi-formal (OMT) requirements document written in advance. They believe that the support for animation, testing and rapid prototyping provided by VDM Tools is very valuable in the early stages of development. For example, this helps to clarify and better understand requirements and to present the developer's view of the requirements to the customer. By using the VDM technology prior to a development in B, they estimate to achieve a total cost reduction of 15–30% compared to the costs of using B for the entire development.

4.3 Applications in the Non-critical Domain

Traditionally, formal methods have been promoted mainly for safety-, security-, or mission-critical systems, as a tool to ensure the "correctness" of such systems. However, in particular, lightweight formal methods can be used in a much wider spectrum of applications. It is therefore very encouraging that we are starting to see applications of the VDM technology outside the critical sector. Two of these are described briefly below.

Banknote Processing by GAO The Gesellschaft für Automation und Organisation mbH (GAO) in Germany develops banknote processing systems. Such machines are very complex with embedded computer systems which must be extremely configurable with respect to both types of sensors and bank-notes to be analyzed. The volume of the details and the number of potential error cases involved in its design proved to be too extensive to be clearly expressed in an informal notation. By formulating the abstract model in VDM, and executing realistic test scenarios with the IFAD VDM Tools interpreter, the software engineers were able to gain confidence that their design was watertight. Only one week of training was necessary to enable them to use formal modeling beneficially in an industrial context.

Sales Configuration by Baan Front Office Systems At Baan Front Office Systems they develop a general tool – SalesPLUS – for the configuration of services and products. For the next generation of the salesPLUS configurators they

are developing a new declarative programming language jointly with an American company. The language is currently undergoing a language analysis phase using VDM in the lightweight fashion described in this paper. A large group of people from both Denmark and USA have been trained in this technology and they feel that this approach has had a number of advantages:

- Common basis for precise discussions.
- Improved cooperation over long distances.
- Testing of VDM models and early construction of test environments.
- Faster route to a prototype.

This project is of vital importance for Baan Front Office Systems and the use of VDM is placed right in the heart of the development process.

5 Conclusions

In this paper we have presented a lightweight approach to formal methods with examples of its use and pointers to a few of the industrial applications. Through formal modeling in an executable notation, the user gets access to validation facilities based on the testing techniques that he is already comfortable with. Hence, the introduction of the IFAD VDM technology in a company is a small, evolutionary change of existing practice, but a big and cost-effective improvement. Training efforts are usually limited to a one-week course. In our opinion the technology transfer is only successful when traditional engineers use the technology rather than formal methods experts as suggested in [6].

The application of our pragmatic, lightweight approach is focused on the early stages of the development process. At this point, the use of abstraction is important to cope with the complexity of systems to better understand the requirements. Moreover, the use of techniques and automation tools that are effective for locating defects is essential at these stages as the expensive mistakes are known to be made here. By integrating external components such as graphical front-ends with the execution of formal models, it becomes possible for a customer or non-trained staff member to assess and interact with a VDM model in a direct way.

Once the requirements are properly analyzed and understood, experience shows that it is relatively easy to implement the system using conventional techniques. Moreover, automatic code generation as supported by the IFAD VDM Tools can also be extremely valuable. However, the trade-off usually is that models need to be more low-level to allow code generation than desirable in requirements analysis.

In addition to executable models, the use of graphical modeling in conjunction with formal modeling is an important contribution to the lightweight approach of the IFAD VDM technology. Through round trip engineering capabilities between UML and the object-oriented formal specification language VDM++, the user gets direct and immediate access to the complementary benefits of graphical and formal modeling. The graphical, diagrammatic view is used

for the structural visualization of high level aspects of a model, while the formal, textual view is used for the formalization of processing details of a model, enabling the analysis of behavioral aspects by automation tools. As graphical modeling is already widely used in industry, such practical combinations can be viewed as simple, cost-effective improvements of existing practice, and may open up many new possibilities of industrial application of formal methods.

The lightweight approach presented in this paper is viewed as a first step improvement of the software development process in industry. At a later stage we also believe that more rigorous approaches including verification may be viable for critical applications, but many enhancements of the existing technology are needed before this will be the case. We are entering a major new Esprit funded project on verification called PROSPER [20], where the industrialization of verification technology will be the long term goal.

We believe that the lightweight trend in applied formal methods continues to provide a really strong strategy for formal methods technology transfer to industry. We are currently experiencing at IFAD that software organizations in the non-critical domains are starting to use the VDM technology. In the Autumn 1998, we already know that Matra, Dassault Aviation, CISI and Bruhn NewTech will start to use VDM in the lightweight fashion in new projects measuring the effect of this. So far the results from, for example, Baan Front Office Systems and GAO look very promising. We expect the main expansion of our users to come from the non-critical domains where our pragmatic approach will be adopted because of cost and quality arguments.

Acknowledgments

We would like to thank Bernhard Aichernig for comments on a draft of the paper.

References

1. J.-R. Abrial. *The B Book – Assigning Programs to Meanings.* Cambridge University Press, 1996.
2. S. Agerholm and P.G. Larsen. Modeling and Validating SAFER in VDM-SL. In *Fourth NASA Langley Formal Methods Workshop.* NASA, September 1997. NASA Conference Publication 3356. Available at http://atb-www.larc.nasa.gov/Lfm97/.
3. S. Agerholm, P.-J. Lecoeur, and E. Reichert. Formal Specification and Validation at Work: A Case Study using VDM-SL. In *Proceedings of Second Workshop on Formal Methods in Software Practice.* ACM, Marts 1998.
4. S. Agerholm and O.S. Pedersen. Enhanced UML Analysis with Formal Modeling and Validation. Draft, Marts 1998.
5. S. Agerholm and W. Schafer. Analyzing SAFER using UML and VDM++. Draft, Marts 1998.
6. S. Easterbrook, R.R. Lutz, R. Covington, J.C. Kelly, Y. Ampo, and D. Hamilton. Experiences Using Lightweight Formal Methods for Requirements Modeling. *IEEE Transactions on Software Engineering*, 24(1):1–11, January 1998.

7. R. Elmstrøm, P.G. Larsen, and P.B. Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994.
8. J.S. Fitzgerald. ESSI Project ConForm: Home Page. WWW at URL http://www.csr.ncl.ac.uk/projects/ConForm.html, 1994.
9. J.S. Fitzgerald and P.G. Larsen. Formal Specification Techniques in the Commercial Development Process. In M. Wirsing, editor, *Position Papers from the Workshop on Formal Methods Application in Software Engineering Practice, International Conference on Software Engineering (ICSE-17), Seattle*, April 1995. ftp://ftp.ifad.dk/pub/papers/icse.ps.gz.
10. J.S. Fitzgerald and P.G. Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development.* Cambridge University Press, 1998.
11. D. Hamilton, R. Covington, and J.C. Kelly. Experience in Applying Formal Methods to the Analysis of Software and System Requirements. In *Workshop on Industrial-Strength Formal Specification Techniques*, pages 30–43. IEEE Computer Society Press, April 1995.
12. IFAD World Wide Web. `http://www.ifad.dk`.
13. International Standard, ISO/IEC 13817-1. Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language, December 1996.
14. D. Jackson and J. Wing. Lightweight Formal Methods. *IEEE Computer*, 29(4):22–23, April 1996.
15. C.B. Jones. A Rigorous Approach to Formal Methods. *IEEE Computer*, 29(4):20–21, April 1996.
16. P.G. Larsen, J.S. Fitzgerald, and T. Brookes. Applying Formal Specification in Industry. *IEEE Software*, 13(3):48–56, May 1996.
17. R.R. Lutz. Reuse of a Formal Model for Requirements Validation. In *Fourth NASA Langley Formal Methods Workshop*. NASA, September 1997. NASA Conference Publication 3356. Available at http://atb-www.larc.nasa.gov/Lfm97/.
18. P. Mukherjee. Computer-aided Validation of Formal Specifications. *Software Engineering Journal*, pages 133–140, July 1995.
19. NASA. Formal Methods, Specification and Verification Guidebook for Verification of Software and Computer Systems. Vol 2: A Practitioner's Companion. Technical Report NASA-GB-001-97, Washington, DC 20546, USA, May 1997. Available from http://eis.jpl.nasa.gov/quality/Formal_Methods/.
20. PROSPER World Wide Web. `http://www.dcs.gla.ac.uk/prosper`.
21. PVS World Wide Web. `http://www.csl.sri.com/pvs.html`.