Formal Specification and Development of a Mission Critical Data Handling Subsystem, an Industrial Usage Report

Manuel van den Berg, Marcel Verhoef, Mark Wigmans
{Manuel.vanden.Berg, Marcel.Verhoef, Mark.Wigmans}@chess.nl

Chess Information Technology, P.O. Box 5021, 2000 CA Haarlem, The Netherlands

Abstract. The authors are very often confronted with the general existing belief that formal methods are still a risky way to produce results in an industrial environment. Therefore, introduction of this type of technology into commercial system development projects remains a challenging task. We report on the experiences that we have gained from the introduction and successful application of VDM-SL (and in particular the IFAD VDM-SL toolkit) to develop a mission critical data handling subsystem for one of our clients. The results presented in this paper will show that there is no need to be overly reluctant to apply formal methods in an industrial context.

We will present the role that VDM-SL technology has played in the different phases of the project, from project start-up to project hand-over. Both technical (such as database integration, validation, verification and code generation) as well as non-technical aspects (such as client acceptance, training needs and impact on the IEEE 12207[1] development process) will be discussed.

Context

The data handling subsystem, which is the system component that has been developed using VDM-SL technology, is the input- and output service of a large data mining application. In this paper, we will focus on the import service only. The data mining application supports the key business process of the client and consists of a very large database and several user-interface components. The data handling subsystem is considered to be mission-critical, since the data mining application is worthless without a reliable I/O facility.

The data mining application is a highly interactive system that accommodates several users to work simultaneously. The user interfaces of the data mining application and database were developed using the Oracle product suite and supporting development tools. In contrast, the data handling subsystem is batch oriented, messages are processed in high volume from an input queue to the Oracle database and vice versa, without any user interaction.

On the input side of the data handling subsystem, the arriving messages can be very complex, due to the richness of the message syntax and semantics. This leaves the opportunity for highly ambiguous messages to be send to the system, analogous to natural language interpretation. The data model on the output side of the data handling subsystem is very elaborate and well–defined, leaving no doubt about how the data should be interpreted. Bridging the gap between the complex input messages and the very well structured output format raises the need for language translation tools. Standard tools like lex and yacc alone are not sufficient for this purpose.

Furthermore, the format of the input messages evolves over time (and in practice, so will the data model of the data mining application), so the requirements with respect to the maintainability of the data handling subsystem were very high. Due to the clear need for a very expressive method

[1] formerly known as MIL-STD-498, which in turn was the successor to the well-known DOD 2167a standard

for specification of the subsystem, we selected VDM-SL (and the IFAD VDM–SL toolkit) as our means to implement the data handling subsystem. VDM–SL is very well suited for language definition and translation specifications, since it has its roots in compiler design.

Applying VDM-SL technology

Our client was not familiar with formal methods at the start of the project and had clearly no ambition to spend a lot of time learning a new modelling paradigm. In stead of forcing our client to accept formal methods, we decided to use VDM–SL [3] as our project internal way of working. Design patterns [1] and UML diagrams [7] were used extensively, to show our client how the application is put together conceptually.

The starting point of the development work was the Interface Requirements Specification that had been produced in the precursor project. This was a natural language document that described which language constructs from the input messages should be mapped onto elements of the logical data model of the data mining application. This maps onto the standard architecture of a three phase compiler, consisting of:

– A scanner/parser front-end. The front-end was hand-written in C++ using lex and yacc [2] and the IFAD VDM–SL C++ abstract data type library [5]. The scanner/parser analyses the syntactic rules and checks the basic semantic properties that must be obeyed. It produces a source abstract syntax as its output.
– A translator. The translator is the core of the application and is fully specified in VDM–SL. The translator takes a source abstract syntax instances as its input and generates a destination abstract syntax instances as its output. During this translation phase, all the complex semantic properties are analysed, comparable to the type checking phase of a normal compiler.
– A database back-end. The back-end compares to the code generation phase of a compiler. The destination abstract syntax is transformed into SQL database modification statementsi to update the database.

During the project, we wrote a 450 page Software Design Document (SDD, containing a 300 page VDM–SL specification) to formally specify how an input message is transformed into the SQL statements to query and update the database. It was very easy to explain the concept of the system to our client using the coloured pipe and filter architecture [8].

Database integration

The IFAD VDM-SL toolkit provides the user with an interface to link to existing code written in C or C++. We used this dynamic linking facility [6] to build an interface to the Oracle database. The Oracle dynamic SQL run-time libraries for C++ enable us to give a text string containing a SQL statement to the database. The result is returned as a linked list of structures containing the result in a late bound fashion (column names mapped onto column values). It was relatively easy to write a C++ wrapper around these Oracle functions. This wrapper class basically translates Oracle database types to VDM–SL types and vice-versa.

The interface to the Oracle database enabled us to send SQL statements to the database and retrieve the results expressed as VDM-SL values already at the specification level (from within the VDMTools interpreter). This was very important, since other teams were still fine tuning the database design, simultaneously with our development effort. Using this interface, we could already try out our specification in a very early phase of the development process. We created a test-suite containing several thousand (real-life) test cases that could be checked automatically (regression testing). Secondly, we were able to ask our client very detailed questions about the expected behaviour of the application, cases that were often not covered by (or even conflicting with) the natural language IRS. Raising those questions during the design phase and interactively showing the client the impact of proposed changes, raised the level of confidence of the client in the quality of our work already during the development phase. As a side effect, we also improved the (natural language) IRS dramatically since we encountered many ambiguities in the specification. We therefore als provided a firm basis for the system validation process, which also takes the IRS as its input.

Code generation

For the development of the production code, we relied fully on the C++ code generator [4] provided by the IFAD VDM-SL toolkit. The run-time performance of the generated code (some 90 kloc) was more than sufficient for our purposes. During unit testing, the generated application was validated against the same test-suite that was used to develop the specification. The generated code has turned out to be very reliable, we have never needed to look at the generated code once.

Verification and validation

The validation and verification of the data handling application was assigned to a special person, who had not been involved in the development of the application itself. He was given access to the IRS document, but not to the SDD. He made a VDM-SL specification to generate (positive and negative) and automatically verify test cases from a formalised version of the IRS. Due to the highly automized way of testing, the software testing plan and the software test report could be generated automatically.

The test-suite that was used for unit testing was massive in size, but only covered 80 percent of the input language definition. During this phase (preliminairy software qualification test) every language construct was checked using the specially developed tools. Some minor errors were found during the validation process. They could be identified in test cases that rarely occur in real-life data (the other 20included in the test-suite used for white-box testing the specification. All of the errors could be fixed in matter of minutes. This again raised the clients' confidence in the application that was developed.

Conclusions

The system as a whole was developed under a fixed-price, fixed date contract. About 4800 man-hours (which corresponds to about 10 % of the total available project resources) where spent on the development of the data handling component. From the start of the project, the data handling subsystem has been in the critical path of the project plan, mainly due to the sequential nature of the activities. Nevertheless, it was the first subproject to deliver results, on time, within the allocated budget and was accepted by the client without a single change. The application of VDM–SL was a major success and our client is continue to work on the data handling subsystem using VDM–SL and the IFAD tools.

References

1. F. Buschmann et al. *Pattern–oriented Software Architectures: A System of Patterns.* John Wiley & Sons, 1996.
2. J. R. Levine et al. *Lex & Yacc.* Unix programming tools. O'Reilly & Associates, Inc, 1995.
3. J. Fitzgerald and P.G. Larsen. *Modelling Systems, Practical Tools and Techniques in Software Development.* Cambridge University Press, 1998.

4. The Institute for Applied Computer Science. *VDMTools: The IFAD VDM–SL to C++ code generator*, 1998.
5. The Institute for Applied Computer Science. *VDMTools: The VDM–SL C++ libaries*, 1998.
6. The Institute for Applied Computer Science. *VDMTools: The VDM–SL Dynamic Linking facility*, 1998.
7. G. Booch J. Rumbaugh, I. Jacobson. *The Unified Modeling Language Reference Manual.* Addison–Wesley, 1998.
8. D. Garlan M. Shaw. *Software Architecture – Perspectives on an emerging discipline.* Prentice–Hall, Inc, 1996.

Acknowledgements

The authors wish to thank Bert Bos, Max Geerling and Peter Gorm Larsen for reviewing early drafts of this paper.