Formal Modelling and Simulation in the Development of a Security-critical Message Processing System*

P. G. Larsen†, J. S. Fitzgerald‡, T. M. Brookes and M. A. Green§

Abstract

This paper describes an experiment evaluating the application of formal techniques to the modelling and development of a security-critical system to high (ITSEC) levels of assurance. The experiment has been done in a commercial environment by engineers working within an existing development process. Two independent teams of engineers in British Aerospace have been concurrently developing a message processing system (a *trusted gateway*). Both teams use CASE technology (Yourdon with Ward & Mellor's extensions supported by requirements traceability tools), but one team additionally uses the formal specification language VDM-SL. Tool support is available for both the CASE techniques and VDM-SL. In this paper we consider the merits of various forms of specification of the trusted gateway, emphasising the value of validating the specification by using it as a direct simulation of the system.

1 Introduction

British Aerospace (Systems and Equipment) (BASE) produces a number of message processing systems for a range of applications. The use of formal methods in the development of these systems is not mandatory, but to attain the higher evaluation levels [ITSEC91], certain aspects of the design should be modelled formally. For example, there is a requirement at the E5 level of assurance for the security policy enforced by the product to be described formally. At E6, formal development with proofs is required.

As part of its process improvement programme, BASE is conducting an experiment to develop a *trusted gateway (network guard)* to a high level of evaluation using CASE technology. Conformance between the implementation and security policy is demonstrated by showing how the simple security policy described in the customer requirement is embodied in each stage of the implementation process and by demonstrating that tests for the security policy can be passed by the implementation at each stage of development. It should be stressed that this is an experimental development: there is no intention of

*Presented at the "Formal Methods, Modelling and Simulation for System Engineering" workshop, February 1995, Saint-Quentin, France

†The Institute for Applied Computer Science (IFAD), Forskerparken 10, 5250 Odense M, Denmark

‡Centre for Software Reliability, University of Newcastle upon Tyne, UK

§British Aerospace (Systems and Equipment) Ltd, UK

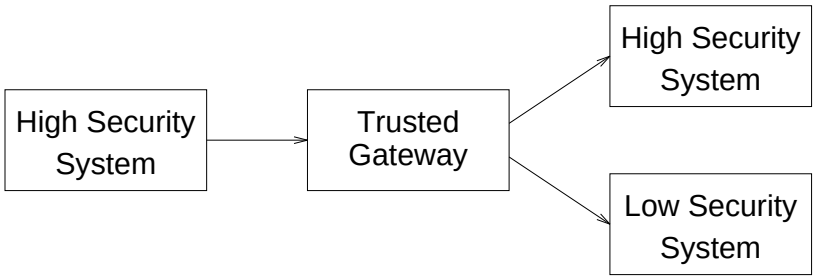

Figure 1: System Function of the Trusted Gateway

attempting to gain an evaluation certificate with the programme as described. A full development process with all required documentation would be needed for this.

The gateway's limited functionality makes it ideal for testing formal methods against current design practices. Following an initial case study at Newcastle University, BASE was awarded a contract under the European Systems and Software Initiative to undertake an experimental development of a trusted gateway using formal and conventional techniques in parallel. Comparing the two development processes will yield insights into the costs and benefits of applying formal techniques in the design of such security-critical components. The collaborators in the study are BASE, the University of Newcastle upon Tyne and the Institute of Applied Computer Science (IFAD), Denmark.

The experiment consists of the parallel development of a trusted gateway by two separate design teams. Independence is enforced by ensuring that the teams work in physically separate areas and are instructed not to discuss the project with each other. One team uses the Ward and Mellor [Ward&85] methodology, supported by the Teamwork Computer Aided System/Software Engineering tool set[1] and the RTM requirement traceability tool[2]. The other team follows a similar design methodology, but uses formal specification to describe at least the function enforcing the security policy implemented by the gateway. A central authority acts as the customer, keeping records of the queries, problems and progress made on the project.

2 Trusted Gateways

A trusted gateway is a device located in the communications path between systems of differing security levels. It sorts incoming messages into different classifications and ensures that they are sent to a destination of the correct security level (Figure 1). For the purposes of our experiment at BASE, the trusted gateway being developed is a simple device consisting of a single input and two outputs. Messages are read into the gateway, are analysed for the presence of character strings which indicate the message classification and then written to the appropriate (high or low security) output port.

[1]Teamwork is a Registered Trademark of Cadre Technology Inc.

[2]RTM is a Registered Trademark of GEC Marconi, Addlestone, Surrey

3 Techniques and Tools

An important restriction on the techniques which could be used for this experiment was that the development still had to satisfy existing BASE standards already certified under ISO 9000. We believe that this is a common restriction which means that formal methods must be introduced as minor increments of the existing software process for a company. Therefore, these two teams both had to follow a similar methodology.

We split the development process into a number of stages:

1. **System Design** Given the customer requirement, systems engineers split and focus the requirements to provide a number of clauses of roughly equal complexity. A system model, based on these requirements, is constructed in Teamwork (with the formal methods team also using VDM-SL). As the model is constructed, ambiguities and incompleteness in the original customer requirement may be exposed. These are resolved by recorded queries and answers.

2. **Software Design** The system design is given further detail and elaborated with code, structure charts and pseudo-code (and, again, VDM-SL in one team).

3. **Implementation** The software design is implemented in the language of choice.

4. **Testing** The test plans produced with the system and software designs are satisfied by a process of testing. We will compare and swap the test sets generated by the "competing" teams in the trusted gateway development.

Each development phase is ended with a formal design inspection and review, and a separate private comparative review of the teams' relative progress. At the time of writing, the system and software design phases are complete.

The RTM requirement traceability tool was used to log all decisions and documentation about the development. This logging process allows us to trace the evolution of the final system specification from the initial customer requirements, despite the divergence in design due to the teams' differing interpretations of the original requirements and different queries raised. The number of requests and their complexity are used as a measure of how well the requirements capture process forces the system designer to clarify the requirement with the customer. At the time of writing, the system specification and the software design have been completed in both development paths.

The Teamwork tool provides a diagrammatic notation for the functional decomposition of the system by means of data flow diagrams. The engineer using VDM-SL used this as a basis for deciding what parts of the system to specify formally. The specifications of types used in data flows and common data stores were given in VDM-SL, as were some of the process specifications. The engineer following the traditional development process used natural language in these situations.

The IFAD VDM-SL Toolbox [Elmstrøm&94] is a collection of tools for formal specifications development using the latest version of the VDM-SL standard [ISOVDM93]. In addition to the full language it also supports a module-based structuring mechanism for large specifications. The Toolbox features extensive semantics checking, documentation

support, test coverage analysis and debugging support. In particular the ability to execute (and even debug) specifications turned out to be valuable for the engineers. It is this ability to execute and test specifications which allows the formal specification to act as a kind of simulation for the trusted gateway. It also affects the style of specification, a factor we consider in more detail below.

Having to write the specification in a formal notation did not pose a major problem for the engineer. The syntax of the language was checked quickly by the tool and the correctness of the specification with respect to informal requirements was validated by executing the specification on simple test cases (cf. the notion of *validation conjecture* in [Bicarregui&94]) and using the facilities of the Toolbox for test suite construction and coverage analysis. A tendency was noted towards the use of familiar constructions instead of the most appropriate or concise form. This problem would reduce in time as the language and its use become more familiar to the engineer. External consultants aid this process by giving the engineer feedback on the style of specification being employed.

The extent to which formal specification was used was largely up to the engineers following the formal methods path. The only "given" was that the security policy implemented by the gateway should be described formally to meet the E5 assurance requirement. The mixture of levels of abstraction in the informal user requirements also makes it unlikely that the engineer would model all the details, since that would compromise the level of abstraction of the formal model, for comparatively small gains in coverage.

4 Modelling

There have been a considerable amount of discussion about the pros and cons for executable specifications [Hayes&89, Fuchs92, Andersen&92]. The main disadvantage about using executable specifications is that the specification necessarily must be more explicit, and thus less abstract. However, in this section we will make a little more analysis of the limitations of writing executable specifications.

In VDM-SL one can use an explicit specification style, by explicitly stating the result of a function (or an operation) by providing an expression (or a statement) using the input parameters. Alternatively, one can use an implicit specification style, by stating what should hold before the function (or operation) is invoked (a pre-condition) and stating what should hold after the function (or operation) has terminated (a post-condition). In general it is possible to state specifications more abstractly by using the implicit style because it is possible to define the post-condition in a way which does not directly show how the output can be produced from the input. However, it is possible to use a similar abstraction technique with the explicit specification style by means of a so-called let-be-such-that expression. This means that one can rewrite an implicit function definition such as:

$Fn\ (i : I)\ r : R$
post $Q(i, r)$

can be rewritten into an equivalent explicit function definition:

$Fn : I \to R$

$Fn\,(i) \triangleq$
  let $r : R$ be st $Q(i, r)$ in r

Implicit specifications may be felt to be more abstract than the explicit, even when the post-condition simply defines the result value as an expression on the input parameters (i.e. "$r = \ldots$"), a style which we have seen in a majority of specifications[3]. We believe that the reason for this is that in order to state a post condition more implicitly, much deeper insight in the problem is required. Very implicit definitions which simply state the necessary properties are not normally the first idea which springs to mind. In addition, it is easy to forget some of the necessary properties such that the post condition actually becomes too loose. We illustrate various levels of abstraction by means of a small example.

At an abstract level, the trusted gateway receives a stream of messages at its input and produces streams of messages at its outputs. Each message has some content (a character string) and some classification, which for our purposes will be either HI or LO. In VDM-SL, we would define the data type of messages as follows:

$Msg :: contents : \mathsf{char}^*$
  $classification : \text{HI} \mid \text{LO}$

The state of the trusted gateway is then modelled as a "black box", where just the inputs and outputs are visible as streams of messages:

state $Gateway$ of
  $input : Msg^*$
  $outHi : Msg^*$
  $outLo : Msg^*$
end

An implicit specification of the gateway's essential functionality should state that:

1. only high-classification messages should appear at the high-classification output $outHi$;
2. only low-classification messages should appear at the low-classification output $outLo$;
3. the messages appearing at each output arrived at the input in the same order;
4. all messages written to the output appear at only one output;
5. all messages arriving at the input appear at an output.

These requirements give us a very implicit specification of the gateway's desired functionality. For the abstract model suggested above, they are expressed formally as the conjuncts in the post-condition of the operation below:

[3]In principle these specifications can easily be executed as well, but this is not yet possible in the Toolbox.

$$
\begin{array}{l}
\mathit{Classify}\ () \\
\mathsf{ext\ rd}\ \ \mathit{input} : \mathit{Msg}^* \\
\quad \mathsf{wr}\ \mathit{outHi} : \mathit{Msg}^* \\
\quad \mathsf{wr}\ \mathit{outLo} : \mathit{Msg}^* \\
\mathsf{post}\ \forall\, i \in \mathsf{inds}\ \mathit{outHi} \cdot \mathit{outHi}\,(i).\mathit{classification} = \mathrm{HI} \wedge \\
\qquad \forall\, i \in \mathsf{inds}\ \mathit{outLo} \cdot \mathit{outLo}\,(i).\mathit{classification} = \mathrm{LO} \wedge \\
\qquad \exists\, \mathit{sHi} : \mathbb{N}_1\text{-}\mathsf{set} \cdot \mathit{outHi} = \overleftarrow{\mathit{outHi}} \frown [\mathit{input}\,(n) \mid n \in \mathit{sHi}] \wedge \\
\qquad\qquad \exists\, \mathit{sLo} : \mathbb{N}_1\text{-}\mathsf{set} \cdot \mathit{outLo} = \overleftarrow{\mathit{outLo}} \frown [\mathit{input}\,(n) \mid n \in \mathit{sHi}] \wedge \\
\qquad\qquad\qquad \mathit{sHi} \cap \mathit{sLo} = \{\} \wedge \\
\qquad\qquad\qquad \mathit{sHi} \cup \mathit{sLo} = \mathsf{inds}\ \mathit{input}
\end{array}
$$

This implicit style of specification is common in abstract axiomatisations of the behaviour of a range of communications systems. The concise description of the gateway's behaviour makes it straightforward to conduct a proof of the security property that nothing with a high classification appears at the low classification output port. The theorem is stated as follows:

$$
\boxed{\text{TG-secure-1}}\ \frac{\mathit{post}\text{-}\mathit{Classify}(\mathit{mk\_Gateway}(\text{-}, \text{-}, \mathit{outLo}))}{\neg\, \exists\, n \in \mathsf{inds}\ \mathit{outLo} \cdot \mathit{outLo}(n).\mathit{classification} = \mathrm{HI}}
$$

Its proof follows easily from the second conjunct of *post-Classify*.

In contrast with the ease of proof construction, it is not possible to test this specification, because of the non-executability of the existential quantifications in the third and fourth conjuncts of *post-Classify*.

This specification describes the essence of the gateway's behaviour: directing incoming messages to the appropriate output in a secure way. However, it abstracts away from a number of aspects of the gateway's behaviour as described in the original customer requirement produced by BASE. We note three aspects:

- The customer requirement views the input as a stream of characters which must be split into messages delineated by start and stop sequences. Here we have viewed the input a sequence of "ready-formed" messages.

- The customer requirement describes a process of receiving and dealing with messages one-by-one. We have instead described the gateway's behaviour over an arbitrary sequence of inputs.

- The customer requirement describes how a message's level of classification is to be determined (by searching for strings from two predefined categories). Here we have modelled messages as having a predefined classification component.

The very abstract specification just discussed was developed by one of the authors (JSF) some time after the project had begun. The next specification we consider is much more similar to that developed by the same author in the feasibility study and, more importantly, that developed by the systems engineer working in BASE.

Here the system state is modelled at a similar level of abstraction to a number of the clauses in the customer requirement. The input and outputs are now sequences of

characters rather than sequences of messages. The beginning and end of a valid message is determined by start- and end-of-message functions (*somf*, *eomf*) allowing for "junk" to arrive at the input between valid messages. Categories of strings indicating the two levels of classification are stored in *catHi* and *catLo*. A message read at the input is stored in internal memory called the *block* and is there analysed for the presence of strings from the two categories. If a high-category string is present, the message has a high classification and is output accordingly. If a low-category string is present, but no high-category string is present, the message is treated as having a low-classification. If strings from both or neither category are present, the message is treated as having a high classification.

The system state space is specified as follows:

state $Gateway$ of
  $input$: char*
  $somf$: char$^+$
  $eomf$: char$^+$
  $block$: char*
  $catHi$: (char*)-set
  $catLo$: (char*)-set
  $outHi$: char*
  $outLo$: char*

  inv mk-$Gateway$ $(input, somf, eomf, block, catHi, catLo, outHi, outLo) \triangleq$
    len $somf$ + len $eomf \leq 10000 \wedge$
    len $block \leq 10000 \wedge$
    card $catHi \leq 20 \wedge$
    card $catLo \leq 10 \wedge$
    $catHi \cap catLo = \{\} \wedge \ldots$
end

The invariant formally records restrictions on the components of the state, as derived from the customer requirement. It is the authors' experience from this, and from a number of other projects, that the production of the invariant is especially valuable. Formalising the customer requirements on data types highlights ambiguities and incompleteness in the original expression of requirements. Indeed, our findings in the trusted gateway project [Fitzgerald&94] suggest that the use of VDM-SL encourages concentration on correct description of data types and exceptional behaviour. The improved clarity of understanding which results from working out the formal clauses of the invariant is often a factor in persuading engineers of the value of formal modelling at this early stage in the development process.

Moving on to the description of functionality at this level of abstraction, the BASE systems engineer chose an explicit definition style. The essential behaviour of the gateway is described by an operation which analyses the contents of the *block* and outputs the contents to the appropriate port:

$$analyse : () \stackrel{o}{\rightarrow} ()$$

$$analyse\ () \triangleq$$
$$(\textsf{if}\ (\exists\, sLo \in (catLo) \cdot occurs\ (sLo, block))$$
$$\textsf{then}\ (\textsf{if}\ freeHi\ (catLo, catHi, block)$$
$$\textsf{then}\ (outHi := outHi \frown block)$$
$$\textsf{else}\ (outLo := outLo \frown block))$$
$$\textsf{else}\ (outHi := outHi \frown block))$$

The check for the occurrence of a substring is in the auxiliary function *occurs*, specified separately. The function *freeHi* looks for an occurrence of a high-category string which does not occur as a substring of a low-category string in *block*. This unusual check is included as a result of a query raised by the systems engineer while writing and testing the specification. The possibility that this situation can arise has not so far been detected in the conventional development path, nor was it considered by any of the "formal methods experts" on the project!

The *Analyse* operation specified above only deals with part of the functioning of the trusted gateway. Formal specifications were also written for the extraction of valid messages from the input, purging of the block, etc.

This specification is executable, in contrast with the more abstract version given earlier. This facilitates testing which makes a valuable contribution to the system test plan, although the formal proof of the security property mentioned above is rather more difficult.

4.1 For and against executability

The engineers on the trusted gateway project chose to write system specifications at about the same level of abstraction as that used in the key parts of the customer requirement document. It takes a much longer period of experience with formal specification to be able to produce more abstract specifications, such as that shown first above. Working at the lower level may mean that the specification is larger and more complex, but it does have the advantage of dealing directly with the objects mentioned by the customer (e.g. input as a stream of characters, rather than, at a lower level, bit streams, or at a higher level, sequences of messages). This can improve the ability to communicate with the customer while incompleteness and ambiguities in the original requirements are being resolved. Many of the clauses in the invariant of the more concrete specification raised useful queries, highlighting problems with the customer requirement which would not have been addressed so soon had the more abstract state been used. One might argue that the more abstract specification is "over-abstract" for this simple system. An implicit version of the *Analyse* operation might have been written on the more concrete state space. The orthogonality of implicit specification and data abstraction is an interesting subject for further discussion.

The engineers have also chosen to write executable specifications. This is not surprising given that the main means of validation available to them was specification testing rather than general proof. The advantage of testing a specification is that the engineer can get immediate feedback on the behaviour which has been formally specified, and can test

pathological cases with ease. A fast turn-around time also contributes to the thoroughness with which it is possible to run validation tests with the customer. We have already noted that many specifications which are given implicitly in model-oriented specification languages such as Z or VDM are highly constructive, especially at the levels of abstraction used in the trusted gateway specifications at BASE.

The decision to write executable specifications rather than performing proof naturally limits confidence in the specification to the test cases employed. Of more concern, however, is the bias which an executable system specification can exert on subsequent software design and implementation. We have, for example, observed that the software engineer writing more detailed executable VDM-SL specifications based on those produced by the systems engineer will tend to use the same algorithm structures, and not be strongly influenced by the implementation language and its features. The consequence of this is that implementors are expected to have to make some design decisions. It is a matter for current research and debate in the implementation phase of the trusted gateway project whether this will prove difficult for the programmer.

5 Conclusion

This modest use of formal specification has so far been felt to be valuable in BASE. The costs of applying VDM-SL are comparable to those of the existing development process, but there do appear to be benefits in permitting system modelling and simulation at a higher level of abstraction than has been the case hitherto. Notably, the early resolution of problems with the customer requirement and the concrete contribution to the test plan which follows from the systems (and software) engineers being able to test the executable specifications.

We generally feel that the way of validating specifications used in this project is the most cost-effective way of gaining confidence in a specification with the state-of-the-art tools. We feel that existing proof support tools are not mature enough to be widely applicable within commercial constraints on cost, support and training time. We must remember that much deeper understanding of the semantics of a specification language is required of an engineer conducting a formal verification. So, although some of the authors are undertaking considerable research in formal verification we see specification execution and testing as the place at which to begin the transfer of formal methods technology to industry. We can then, step by step, improve the skills of the industrial engineers, while taking account of the commercial risk involved at each step. We consider it imperative that tool support for formal verification will be improved in the same period. Here we believe that it is essential to gain more knowledge about what is required for such systems to be industrially applicable, before these improved systems are developed. Here we refer to simple, but insufficiently discussed, issues such as limiting the effects of changes in a specification on the relevant proofs.

In conclusion, although we recognise the value of proof, we are of the opinion that the value of executable specification as an aid to early simulation of a system should not be overlooked if we are to apply formal methods effectively in a wide commercial context.

A References

[Andersen&92] Michael Andersen, René Elmstrøm, Poul Bøgh Lassen and Peter Gorm Larsen. Making Specifications Executable – Using IPTES Meta-IV. *Microprocessing and Microprogramming*, 35(1-5):521–528, September 1992. 8 pages.

[Bicarregui&94] Juan Bicarregui, John Fitzgerald, Peter Lindsay, Richard Moore and Brian Ritchie. *Proof in VDM: A Practitioner's Guide. FACIT*, Springer-Verlag, 1994. 245 pages. ISBN 3-540-19813-X.

[Elmstrøm&94] René Elmstrøm, Peter Gorm Larsen and Poul Bøgh Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994. 4 pages.

[Fitzgerald&94] J. S. Fitzgerald and T. M. Brookes and M. A. Green and P. G. Larsen. Formal and Informal Specifications of a Secure System Component: first results in a comparative study. In M. Naftalin and B. T. Denvir and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, pages 35–44, Springer-Verlag, 1994.

[Fuchs92] Norbert E. Fuchs. Specifications are (Preferably) Executable. *Software Engineering Journal*, 323–334, September 1992. 12 pages. .

[Hayes&89] I.J. Hayes, C.B. Jones. Specifications are not (necessarily) executable. *Software Engineering Journal*, 330–338, November 1989.

[ISOVDM93] *Information Technology Programming Languages – VDM-SL.* Technical Report, First Committee Draft Standard: CD 13817-1, November 1993. 410 pages. ISO/IEC JTC1/SC22/WG19 N-20.

[ITSEC91] *Information Technology Security Evaluation Criteria.* Office for Official Publications of the European Community, June 1991.

[Ward&85] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems.* Volume 1-3, Yourdon Press, New York, 1985-1986.