FORMAL METHODS STRATEGY STUDY REPORT

Defines a strategy and programme to introduce formal methods into EURO-INTERLOCKING Project 2

Version 1.1

Created: 11.05.00

Saved: 15/05/00 09:41

Total Number of Pages: 26

Filing Name: R019 FMStrategy Study.DOC

Document Data Sheet

Filing name R019 FMStrategy Study.DOC	Document Type	Last saved 15/05/00 09:41	This version 1.1	Last saved by N. König

Title of Document Formal Methods Strategy Study Report	Languages		
	Original English	Translations	
Subject	Pages 26	Figures	Tables
Author(s) L.-H. Eriksson (Industrilogik), G. Finnie (Praxis Critical Systems), I. Herttua, N. König	Price		
	Document	Right of Use	

Performing Body	Sponsoring Body

Approved by Performing Body	Approved by Sponsoring Body	Availability of Document
Name Project Manager	Name	Unrestricted

Application Used	Template Name Study final.dot	Last Printed 18.05.00 11:05	Date of Publication May 2000

Abstract

Defines a strategy and programme to introduce formal methods into EURO-INTERLOCKING Project 2

Document Control Sheet

Name	Institution	Level	Task	Date

Table of Contents

References to Cited Texts

1. Eriksson, L-H and Fahlén, M.: *An Interlocking Specification Language*, In: *ASPECT99, Papers of the International Conference*, The Institution of Railway Signalling Engineers, London 1999.

2. Spivey, J.M.: *The Z Notation*, Prentice-Hall 1992

3. Abrial, J-R., *The B-Book*, Cambridge University Press 1996.

4. Owre, S., Rushby, J. and Shankar, N., *PVS: A Prototype Verification System*, In: *Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 607, Springer-Verlag 1992.

5. *The RAISE Development Method*, Prentice-Hall 1995.

6. Jones, C.: *Systematic Software Development Using VDM*, Prentice-Hall 1990.

7. Ghezzi, G., Mandrioli, D. and Morzenti, A, *TRIO: a logic language for executable specifications of real-time systems*, The Journal of Systems and Software, vol. 12, no. 2, May 1990.

8. Behm, P. and Meynadier, J.-M.: *Météor; A Successful Application of B in a Large Project*, In: J.M. Wing, J. Woodcock and J. Davies (eds.): *FM'99 – Formal Methods, Vol. I*, Springer Lecture Notes in Computer Science 1708, Springer-Verlag 1999.

9. Eriksson, L-H: *ISL – Interlocking Specification Language*, Industrilogik report L4i-99/361, 1999. (In Swedish.)

1. Introduction

1.1 Background

The immediate goal of the EURO-INTERLOCKING project is to develop a set of requirements that will capture the full range of requirements for interlocking across Europe. This stage is called Project 2 – Railway Requirements.

The EURO-INTERLOCKING Core Team in Zurich (about 8 people from several different rail authorities across Europe) will collect requirements from the supporting organisations, and other relevant sources (eg standards bodies). They will analyse the information obtained to produce clear consistent generic interlocking requirements. The draft requirements will then be reviewed by the supporting organisations, prior to their approval.

A goal of Project 2 is to utilise *Formal Methods* to describe the requirements in an unambiguous manner, to facilitate the maintenance of the requirements and to assist in the verification and validation of systems to meet the requirements. Praxis Critical Systems and Industrilogik L4i AB have been contracted by the EURO-INTERLOCKING project to carry out a short study to define a strategy and programme for the introduction of Formal Methods in Project 2.

1.2 Purpose of this document

This document contains the results of the Formal Methods Strategy Study. It therefore:

- describes the issues that need to be addressed and the decisions that need to be made in order to make effective use of Formal Methods for the EURO-INTERLOCKING Requirements;
- recommends solutions to the key issues, based on the technical judgement and experience of the study team;
- identifies both a long-term strategy and an outline action plan for the introduction of formal notation into Project 2.

1.3 Scope of this document

The main issue addressed by the study is the choice of formal notation (informally referred to as IRL, Interlocking Requirements Language) to be promoted by the EURO-INTERLOCKING project. This includes consideration of how that notation should be used and its support by tools. The formal notation needs to be complemented in use by a less mathematical, though still coherent, description of the EURO-INTERLOCKING requirements. The document therefore also identifies a structured approach to expressing EURO-INTERLOCKING requirements in natural (English) language along with other rigorous techniques (eg diagrams) and discusses the relationship between these and the mathematically formal requirements description.

To satisfy budget and time constraints, the report does not set out to provide a detailed justification of all its recommendations. These recommendations are based on the technical judgement and experience of the study team rather than an exhaustive investigation of all possible options.

1.4 Structure

Section 2 describes the key issues for IRL that must be considered and the selection criteria for choosing appropriate components (principally language and tools) of a formal methods strategy for the EURO-INTERLOCKING Project.

Section 3 describes the candidate choices for language and tools, outlines their key features and evaluates their suitability for the project.

Section 4 discusses how a less mathematical description of the requirements will be necessary to complement the formal notation, and sets out a recommended approach to achieve this in the most effective way, including the structured use of natural (English) language.

Section 5 provides both a long-term strategy and outline action plan for the introduction of formal notation into Project 2.

Section 6 presents a summary of the main conclusions and recommendations of the study.

2. Interlocking Requirement Language (IRL)

The overall aim of the EURO-INTERLOCKING project is to reduce total life cycle cost of interlocking systems and using formal methods is one way to achieve it. Interlocking Requirement Language is foreseen as a method to express signalling requirements. A railway specific language gives the possibility to use a mathematically precise technique to describe an interlocking within its domain, and improve the quality and productivity of the development process.

The main aim of IRL is to give an unambiguous definition of requirements based on a formal notation and enable the verification and validation processes. Reducing development and verification cost by using automated tools and methods are some of the possible contributions. The use of formal language like IRL specially designed to engineering domain will lead to a higher quality requirements definition and reduce the costs of requirements maintenance.

Among the issues involved in the formal methods strategy for EURO-INTERLOCKING, the choice of formal language and tools is considered to be one of the most important. An established and generally known language (like B or Z) should be used to achieve the widest possible dissemination and use of the formal specification. IRL is to be a further development of a know language in order to adapt it to the needs of the interlocking development process while using existing language notation and tools. The chosen notation and tools must also satisfy several diverse and partly contradictory requirements. The question of language selection criteria will be addressed first and then it will turn to the question of tools.

The discussion in this section is mainly motivated by Industrilogik's concept of an Interlocking Specification Language (ISL) [Ref.1].

2.1 Language Selection Criteria

The practical use of formal methods requires a good tool support. The choice of formal language must take into account the availability of tools. The need for a good toolset to support the writer and user of formal specifications can hardly be overstated. Tools must be available for such tasks as type checking, animation and theorem proving (see section 2.2). The EURO-INTERLOCKING project cannot be expected to do any development of tools apart from necessary adaptations of tools to the task at hand (e.g. development of theorem proving tactics for the library concepts). Choosing a notation supported by a good selection of tools increases the likelihood of finding the right tool for the task and reduces the risks of being left without support if some tool should be withdrawn from the market.

An obvious criterion is that the language be sufficient in the sense that essentially all requirements can be expressed with a minimum of reformulation or 'encoding'. However, in any formal language, there is a conflict between the expressiveness of the language and ease of implementing tools to process the language. A high degree of expressiveness means that the language offers powerful and flexible means of expression – much can be said in few words – so that the writer of requirements finds the language a natural means in which to express himself. Since the use of computer tools is of major importance for the practical use of formal methods, it is important that the language is carefully designed to obtain maximum benefit from state of the art of tool technology.

This is particularly important for tools performing animation or theorem proving tasks. Unfortunately, there is a conflict between the expressiveness of a language and the extent to which the language can be automatically processed by tools. A high degree of expressiveness has the drawback that the task of tools becomes harder. In particular, to utilise modern highly efficient automated propositional theorem proving and/or model checking tools, it must be possible to translate the requirements into propositional logic. This is possible in principle in many applications once the requirement has been ‘instantiated’ by giving particular finite values to free variables and sets (e.g. the ‘given sets’ in Z).

To resolve this conflict, it has been noted that there are two fundamentally different kinds of facts that need to be expressed in a formal requirement. The first one being the *actual* requirements that one wishes to formalise (e.g. requirements such as those given in section 4.2). The second one being the *domain theory* or ‘conceptual model’, comprising descriptions of the basic concepts (such as ‘route’, geographical relations etc.) used to express the actual requirements.

The parts of a formal requirement generally requiring the greatest expressibility are the ones dealing with the domain theory. The parts dealing with the actual requirements require less expressiveness. If the language is sufficiently expressive to permit a natural description of the domain theory, the possibility of automating formal reasoning is reduced. Thus, there is a conflict between the expressiveness needed for the domain theory and the lack thereof needed for efficient implementation.

The previous experience with domain theories for railway signalling applications shows that the domain theory itself would not be difficult for a tool to handle efficiently. The difficulty lies in the generality of the language constructs needed to express the theory. As the core of the domain theory can be expected to change little or not at all once it has found its proper form, a possible solution could be to use a language with limited expressiveness, but with a fixed, predefined, domain theory. A tool could then implement the domain theory directly without having to handle a description of it in a very general formal notation.

In fact, making a domain theory requires a considerable expertise in the use of formal methods, while making use of that theory in formalising actual requirements can generally be done by domain experts (i.e. signalling engineers) trained in formal methods use (possibly with the support of formal methods experts). This makes it more likely that the domain theory will remain essentially fixed in a given application.

From a technical point of view, having a simple language with a built-in domain theory would seem to be the best approach, the major drawback being that specialised tools would be needed. However, practical considerations below will force us to modify this conclusion.

The overall aim of the EURO-INTERLOCKING project is to reduce total life cycle cost of interlocking systems and the choice of formal language must be made in light of this. For a formal requirement effort to contribute towards this goal, the formal requirement must be usable by prospective suppliers in their product development processes. This means that although tailored solutions like the specialised language with built-in domain theory outlined above could be preferable for the internal work of the EURO-INTERLOCKING project, an established and generally known language should be used to achieve the widest possible dissemination and use of the formal specification.

The intended users of the EURO-INTERLOCKING requirements are more likely to accept a formal notation if it is based on an existing notation which is widely used, or at least widely accepted. It will also make it easier to find staff with knowledge of the notation or to find training material. Likewise, the availability and quality of tools is likely to be greater.

A solution to this dilemma is to use an established highly expressive language, but to use the full power of the language only for describing the domain theory. This domain theory could be seen as an unchanging library which is referenced by the proper specification. For writing the actual requirements, a suitable less expressive subset of the language would be chosen. Coding rules could also be used to restrict the expressiveness. Since the domain theory is presumed to be essentially fixed, tools could still treat references to is as special cases, while being able to efficiently process the subset language used for the actual requirements.

This approach has the additional advantage that specialised tools are not strictly needed. The requirement could still be used by a general tool, although performance would suffer.

When evaluating a language for EURO-INTERLOCKING requirements, in addition to the general language features discussed above, two particular language features should be considered.

The first one is the extent to which the language supports modularisation of specifications. It is generally a good idea to break up a large specification into smaller parts. Also, as it has been outlined above, the domain theory will likely have to be a separate library module. Since the EURO-INTERLOCKING project aims at presenting several national requirements in a common format, it can be expected that the presentation can be simplified if common or similar parts are shared and the requirements structured accordingly. To support these needs, the chosen notation must provide adequate support for modularising requirements.

The second feature is the description of change over time (temporal properties). Experience shows that most requirements involved in railway signalling only refer to a single moment of time. (This is in contrast to many other application areas where the sequencing of events is of major importance.) Still, in some cases (e.g. route locking/release) requirements will need to reference time or event sequences. The more powerful the language facilities for expressing relations over time, the more natural the formalisation of such requirements will be.

In summary, the language selection criteria are :

1. **Tool availability** – availability must be evaluated in relation to the Tool selection criteria below.

2. **Good expressiveness** – the task of writing the domain theory will likely be carried out by formal methods experts, and the core of the theory will seldom or never be modified after its initial development, it is only moderately important that the notation fulfils this criterion.

3. **Established/accepted** – notations not fulfilling this criterion should not be considered unless they are expected to bring major technical advantages over other notations.

4. **Possibility of defining subset/coding rules** to allow efficient automatic processing by tools.

5. **Support for modular requirements** – basic support is expected to be needed

6. **Temporal expressiveness** – most requirements can be expected to refer to the same time instance, so only very basic expressiveness in this area will actually be needed.

2.2 Tool Selection Criteria

As mentioned in the previous section, tool support is a necessity for the practical use of formal methods.

There are four major kinds of tool-supported tasks that are of use to the writer of a formal requirement (e.g. a EURO-INTERLOCKING team member or consultant).

- Type checking. This involves checking that the specification is well-formed and does not include undefined or poorly defined constructs. Type checking may involve theorem proving as a subtask.

- Animation. The animator tool will regard the specification as a very high level program and ‘execute’ it. Animation is a very powerful technique to validate a specification and to learn how the specification works.

- Theorem proving. This involves making a logical analysis to check whether or not some statement is a logical consequence of a set of formulae (e.g. the specification). Theorem proving is a complex task that should be automated to the fullest extent possible. Generally, some amount of manual guidance of the theorem proving process will be needed.

- Clerical tasks like version control, formatting, document preparation etc.

All these functions should be available, either in an integrated toolset or in a collection of individual tools. If a suite of individual tools is chosen, it is important that the tools used fit together well (or can be made to do so with a reasonable effort) – in particular, that the language versions accepted by the individual tools are compatible.

It can be expected that a future need will arise to use new and/or tailored tools for particular functions, e.g. in a formal verification environment. In particular it is important that an integrated toolset is ‘open’ and provides some way of interfacing with other tools. Although this is more likely to be of use to suppliers of signalling equipment using the EURO-INTERLOCKING requirements than to the EURO-INTERLOCKING project itself, tools should be selected with an eye to facilitate future use of the requirement by other parties.

To allow efficient animation and automatic theorem proving, the operation of the tools should be adaptable to the specification at hand. In particular, it should be possible to make the tool handle the concepts of the domain theory directly as if they were built- it. This could be done by programming new tactics, proof or execution rules.

Some criteria are no different from those applied to any software. The quality of the user interface should be taken into account. The tool should preferably be available on a number of different platforms. The quality of documentation and level of support that can be expected from the tool vendor (or other organisation) should be taken into account, as should the future prospects of the tool – in particular, what user base the tool has and how established it is in the formal methods community. This is especially important for formal methods tools which typically have a small, highly specialised user base and run a higher than average risk of being discontinued by their vendors.

Note that ‘academic’ tools should not be prejudiced against. Many tools developed by universities and research organisations have high quality, good support and a (comparatively) large user base.

In summary, the tool selection criteria are:

1. **Complete functionality** – all required functions should be available (possibly in conjunction with other tools).
2. **Integrated environment** – if the environment is based on individual tools, they must fit together well.
3. **Ease of interfacing with other tools** for future applications.
4. **Adaptable/programmable** – important to allow a high degree of automation.
5. **Good user interface** – a modern graphical interface is desirable.
6. **Good support/documentation/future development** – this does not necessarily mean that the tool has to be ‘commercial’.

3. Evaluation of Languages and Tools

3.1 Language Evaluation

There are a large number of notations that could possibly be used for expressing the EURO-INTERLOCKING requirements. An evaluation that can be said to be in any way complete is unfortunately outside the scope of this study. Instead, two notations (Z [Ref. 2] and B [Ref. 3]) have been selected for evaluation. The selected languages are clearly established and have also been used for substantial specification work in the railway signalling domain. Also the PVS notation has been chosen to be evaluated. PVS has been developed by SRI International [Ref. 4]. Although not as established as Z or B, it has been used for a large number of studies and it features an excellent theorem-proving tool.

There are many other notations (e.g. RAISE [Ref. 5], VDM–SL [Ref. 6], TRIO [Ref. 7]) which are arguably candidates for EURO-INTERLOCKING specification work, but which had to be left out due to time constraints

Below the evaluation of the three selected notations has been done against each of the criteria given above.

3.1.1 Z

1. Tool availability is good. However, it is unclear to what extent the available tools satisfy the tool selection criteria. (See 3.2.2)

2. Expressiveness is good. Z is based on set theory and provides a rich set of operations.

3. Z is one of the oldest and most widely used and taught specification notations.

4. It is unclear to what extent it is possible to define a subset that can be efficiently processed by an automatic tool. While powerful, operations on sets can require substantial computation in an animator. Also, state transitions in Z are expressed as relations between the old and the new state. These relations may not have a form that permits an animator to directly compute a new state given an old one. Transforming of set theory into logic for the purpose of theorem proving is naturally done using characteristic predicates for each occurring set. Since many Z operators create new intermediate sets, a fair amount of analysis of the Z expressions may be required by the translator to keep down the size of the translation.

5. The Z schema calculus is a very powerful mechanism for modularising specifications.

6. The temporal expressiveness of Z is poor. Only relations between a state and the next can be expressed.

3.1.2 B

1. Tool availability is very good. Two complete toolsets are available which are both used in industrial applications.

2. Expressiveness is good. Like Z, B is based on set theory and provides a rich set of operations.

3. B is quite well-known. Although not as established as Z, B figures in some remarkable success stories of industrial applications of formal methods, eg by MATRA [Ref. 8].

4. It is unclear to what extent it is possible to define a subset that can be efficiently processed by an automatic tool, but the situation is better than for Z. The same comments given for Z apply here, with one important exception: A new state is given as a function of the old one, so it is straightforward for an animation tool to compute the new state. Non-deterministic state transitions are obtained using operations for explicit non-determinism.

5. B includes facilities for modular specifications, although not as powerful as those of Z.

6. The temporal expressiveness of B is poor. Only relations between a state and the next can be expressed.

3.1.3 PVS

1. Tool availability is poor. Only a single tool is available, providing type checking and theorem proving – no animation. (Although the tool is an excellent one.)

2. Expressiveness is good. PVS is based on higher-order logic which is a formalism at least as powerful as set theory. It also provides a powerful typing mechanism.

3. PVS is not well-known outside the research community. It has been used rather extensively as a vehicle for experimental work in formal verification.

4. There is a good possibility to define a subset for efficient processing by tools. It has been argued that a logic-based language is generally better suited for implementations since sets are difficult objects to compute with efficiently.

5. PVS includes facilities for modular specifications, although not as powerful as those of Z.

6. The temporal expressiveness of PVS is potentially good. The basic language does not include any temporal features, however it is straightforward to extend the language to include arbitrary temporal features. The PVS library includes an extension for model checking in CTL (computational tree logic) and the   -calculus.

3.2 Conclusions

All three languages evaluated are suitable as specification languages for the EURO-INTERLOCKING requirements. PVS appears to be a better choice than B or Z from a purely technical point of view (criteria 2, 4, 5 and 6), although for the purposes of EURO-INTERLOCKING, the differences are minor. B and, particularly, Z are much more well known than PVS.

The result of the tool survey (section 3.3) indicates that B is by far the best choice, followed by Z and then PVS. Considering the importance of tool support, the relatively minor differences in technical merit and that PVS is considerably less known than B or Z, the conclusion is that **B is the language that best fulfils the criteria as a whole**.

3.3 Tool Survey

The timescale for this study did not permit a practical evaluation of any tools. It was only possible to survey the tools known to be available, based on subjective experience and on information available via the Internet.

3.3.1 Z tools

A very wide range of tools is available to support Z. The Z page on the Oxford University Programming Research Group's web site at oldwww.comlab.ox.ac.uk/archive/z.html lists over thirty tools, among which the best known are:

- Mike Spivey's FuZZ Z type-checker, available commercially together with an associated *fuzz.sty* LateX style file which has better fonts for the more esoteric Z symbols. The type-checker runs on Sun workstation, PC and VAX/VMS equipment.
- CADiZ from York Software Engineering Ltd provides support for Z using troff and LateX on Unix systems and Microsoft Word on PCs
- ProofPower, available from ICL, is an industrial strength Z theorem prover based on Higher Order Logic
- Z/EVES, providing a Z front-end to the EVES proof tool based on ZF set theory from ORA, Canada, is available via on-line distribution.

Fonts and style files to support the mathematical symbols are available from a number of sources.

The number of tools available for Z does indicate a wide interest in the language and those listed above are known to be sufficiently robust for industrial use. However, it also shows the fragmented nature of the Z tools market. There is no single tool for Z that would provide all the facilities needed for the EURO-INTERLOCKING project (type checking, theorem proving and animation) and no integrated Z toolset is available.

3.3.2 B tools

The B Method was developed from the start with industrial-scale tool support in mind. Consequently, powerful and integrated tool support exists, providing assistance for all stages of the B development process.

Currently there are two different B toolkits on the market:

- The B-Toolkit is a set of integrated tools which fully supports the B-Method for formal software development, built on top of the B-Tool, a language interpreter and a run-time environment for supporting B. These tools are available from B-Core (UK) Limited, UK. The B-Toolkit provides syntax and type checking, proof obligation generation, a combination of automatic and interactive theorem proving, and animation. It also provides support for document production and has a graphical user interface. For full development using the B-Method, the B-Toolkit provides code generation in ANSI C.

  The EURO-INTERLOCKING project is using the DOORS requirements analysis tool from QSS. It has been understand that QSS, in conjunction with B-Core, have developed an interface between DOORS and the B-Toolkit.

- Atelier B is an integrated toolkit for the B method developed by Steria Mediterranee (France), with the collaboration of Jean-Raymond Abrial and GEC Alsthom Transport. Atelier B is sponsored by RATP, SNCF and INRETS and developed with their technical co-operation. The range of facilities in Atelier B is broadly similar to those of the B-Toolkit and includes syntax analysis, type checking, proof obligation generation, theorem proving and animation. Code generation into C and Ada is available. Access to Atelier B software tools is by a graphical MOTIF interface or by a batch language, and in each mode it manages multi-user projects. The first version of Atelier B was upgraded in 1995 with an animator, a new generation of prover, a powerful project documentation generator and various interfaces with standard tools including Word and Interleaf.

A small number of other independent tools are available, such as editors, parsers and supporting fonts, but are unlikely to be of interest given the integrated and comprehensive nature of the two main contenders.

The B-Toolkit and Atelier B appear to be broadly equivalent in terms of technical specification and both seem capable of satisfying the requirements of the EURO-INTERLOCKING project. A detailed practical evaluation would be required in order to make a choice between them; ultimately, the decisive factors are likely to be usability and commercial considerations.

3.3.3 PVS tools

PVS is supported by a theorem proving tool available from SRI International.

The PVS theorem prover provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a sequent calculus framework. The primitive inferences include propositional and quantifier rules, induction, rewriting, and decision procedures for linear arithmetic. User-defined procedures can combine these primitive inferences to yield higher-level proof strategies. Proofs yield scripts that can be edited, attached to additional formulas, and rerun. This allows many similar theorems to be proved efficiently, permits proofs to be adjusted economically to follow changes in requirements or design, and encourages the development of readable proofs. PVS includes a BDD-based decision procedure for the relational mu-calculus and thereby provides an experimental integration between theorem proving and CTL model checking.

PVS uses Gnu or X Emacs to provide an integrated interface to its specification language and prover. Commands can be selected either by pull-down menus or by extended Emacs commands. Extensive help, status-reporting and browsing tools are available, as well as the ability to generate typeset specifications (in user-defined notation) using LaTeX. Proof trees and theory hierarchies can be displayed graphically using Tcl/Tk.

Although clearly PVS has an excellent theorem prover, it is weak in the other areas of tool support required by the EURO-INTERLOCKING project. In particular, it lacks an animation tool.

3.4 Summary

As far as appropriate tool support is concerned, it is clear that the most suitable choice is B, followed secondly by Z, and thirdly by PVS.

- B is the only notation that clearly meets the EURO-INTERLOCKING requirements for tool support and it does so within an integrated environment.

- A well-chosen set of independent Z tools could possibly meet the requirements, but the tools would not be integrated together without a lot of additional work.

- PVS is good for theorem proving but lacks the full range of tools required.

4. Describing Requirements Informally

The formal notation on its own will not meet all the needs of the EURO-INTERLOCKING Project. There are a number of reasons for this:

- non-technical readers will require a natural language commentary as a ‘gentle’ introduction to the more formal description;
- even for readers fully familiar with the notation, it is common practice to complement a formal specification with a less formal description which provides real-world context and an overview of the formal material;
- initial capture of requirements will proceed while the formal notation is still being developed, so there is a need to express requirements rigorously in another way until the notation becomes available for use;
- use of the formal notation is likely to be limited in the first instance to expressing safety constraints rather than positive operational requirements;
- some requirements (eg performance) will always remain outside the scope of the formal notation.

Therefore, the use of the formal notation must be complemented by less mathematical, though still rigorous and coherent, descriptions of the EURO-INTERLOCKING requirements. In this section, a structured approach has been identified to express EURO-INTERLOCKING requirements in natural (English) language along with other informal techniques (eg diagrams) and discuss the relationship between these and the formal requirements description.

4.1 Structured English Requirements

As explained above, there is a need to describe the EURO-INTERLOCKING requirements in a natural language form as well as in the formal notation. However, it is well recognised that requirements written in natural language are often expressed in an ambiguous and unclear way; indeed, this was the principal motivation for the development of formal methods.

The challenge therefore is to devise a means of rigorously describing requirements in a natural way without a strictly mathematical formal notation. The experience in software engineering is that there is no single solution to this problem. Instead, it is believed that the required objective can best be met by the combination of a number of approaches which taken together will lead to the production of clear and unambiguous requirements statements.

The elements of this strategy are as follows:

- High-level structure

  Clarity can be greatly improved by adopting a standard high-level structure for requirements definitions. By clearly distinguishing domain knowledge from system requirements, and by separating rules, conditions and operations from each other, one provides a consistent framework for the different sorts of requirements statements. By clearly delimiting the context of each statement, it greatly reduce the scope for ambiguous interpretation.

- Good term definitions

  To provide a firm foundation for expressing requirements, it is essential to have a clear definition of the terms used. It is very important to use terms consistently and to avoid overlapping terms, such as having more than one name for the same thing. Object modelling and diagramming techniques can assist in the definition of terms, as well as in the exploration of concepts leading to those definitions.

- Structuring of operations

  Just as the high-level structure provides a clear context for the different classes of requirements, a consistent structure for defining operations focuses attention on the essential aspects of each operation. For example, by clearly identifying inputs, outputs, preconditions and postconditions, one helps to ensure that the definition of each operation is complete and unambiguous.

- Logic construction guidelines

  Trying to define a rigid subset of English to express requirements is unlikely to be cost-effective or even successful. A more productive approach is to draw up guidelines, based on experience, to avoid the most common pitfalls. A set of guidelines for expressing logic constructs clearly in English would cover, for example, distinguishing clearly between ‘if’ and ‘if and only if’, the use of inclusive and exclusive ‘or’, use of quantifiers and advice on the degree of nesting acceptable.

- Sentence construction guidelines

  Similarly, guidelines for the construction of clear English sentences are required. These would include issues such as using present tense instead of future and minimising the number of subordinate clauses.

  Good advice on how to write clearly in English can be found in material from the Plain English Campaign (www.plainenglishcampaign.com).

4.2 Examples

As a brief example of the approach outlined above, consider the following statements from a hypothetical requirements specification.

> A **set of points** becomes **locked** *if* it is **part of** a **locked route**.
>
> A **route** becomes **locked** *if and only if* a **lock command** is received and conditions for locking are satisfied.
>
> A **route** is **released** *if and only if* the conditions for release are satisfied.

These statements use basic domain terms such as ‘set of points’, ‘route’, ‘lock command’ and the concept of ‘locked’ and ‘released’ points and routes. These terms would be fully defined in a separate section, together with definitions of any significant relationships between terms (for example, a set of points may be ‘part of’ a route). The terms and relationships would also be shown schematically in the form of an object model diagram to provide an overview and a starting point for the reader.

Logic constructs are restricted to a well-defined set of basic connectives such as ‘if and only if’. The actual conditions for locking and release are defined in further statements to avoid over-complex sentence structure. For example,

> The conditions for **locking** a **route** are that the **route** is **set** *and* **obstacle free** *and* **conflict free** *and not* being **manually released**.

Clarity is further enhanced by adopting a consistent typographical convention to show the reader which words are to be interpreted as basic terms or as logical connectives. For example, here it has been chosen to write terms in bold font and logical connectives in italics.

This style of writing requirements in natural language assists in developing and understanding the mathematically formal expression of the requirements, since the transformation between the natural language statements and their expression in the formal notation is relatively small. The basic objects appear in the formal notation as variables while relationships between objects and changes in the state of objects are expressed as predicates. For example, the first statement might have an equivalent formal form of

ALL pt (SOME rt (locked(rt) AND part_of(pt, rt) -> points_locked(pt))

It is interesting that even here the formal expression is more precise than the English statement. Note how the indefinite article ‘a’ set of points has produced a universal quantifier (ALL pt) whereas ‘a’ route has produced an existential quantifier (SOME rt).

5. Long-Term Strategy and Action Plan for Project 2

The study has identified a basis for the development of an Interlocking Requirement Language (IRL). IRL is a formal notation for expressing interlocking requirements that is to be ultimately be used by the EURO-INTERLOCKING project. However, it will take some time to develop the notation fully and to put the necessary tool support in place. Consequently, it is unlikely to be ready for serious use within the timeframe of Project 2 (end 2000).

Planning therefore needs to be considered from two distinct perspectives:

1. Actions required in Project 2 to enable current work to be done in a way that is consistent with the eventual use of the IRL notation;

2. Longer term strategy for development of the IRL and its introduction into use on the EURO-INTERLOCKING project.

5.1 Actions in Project 2

Work on development of the formal notation will proceed in parallel with the work being done during 2000 to capture EURO-INTERLOCKING requirements. How can it be ensured that the requirements are expressed in a way that will smooth the transition to the eventual use of the IRL notation?

The key to satisfying this objective lies in having a sound foundation for the concepts that will be expressed in the formal notation. This will be achieved by following the strategy described in Section 4, and in particular from the use of object modelling and clear definition of terms.

The required actions are therefore:

1. Develop object models for the basic concepts underlying the EURO-INTERLOCKING requirements, and provide clear definitions of all terms.

2. Express the requirements in natural language using the approach and style described in Section 4, using terms taken from the object model.

3. Provide close liaison between the Core Team and the IRL developers to ensure that there is a close match between the basic building blocks provided by the IRL and the core components of the object model. This is clearly a two-way process.

4. As the IRL takes shape, start to express the fundamental safety constraints in the notation.

5.2 Long-Term Strategy

5.2.1 Banverket's ISL study

During 1999, Industrilogik carried out a pre study of ISL (Interlocking Specification Language) [Ref. 9] on behalf of the Swedish National Rail Administration (Banverket). Part of the pre study concerned a rough definition of an ISL development project. This study envisioned an independent development project, but conclusions are in the main part valid also when IRL development is seen in the context of the EURO-INTERLOCKING project. Here will be summarised the development project as outlined in the pre study report.

The purpose of an IRL development project would be:

- To define the IRL language;
- To define the methodology of using IRL during requirements definition, interlocking construction and analysis;
- To carry out a pre study of tools development, including an outline functional specification for tool support;
- To develop a prototype/demonstrator for the tool functions.

The project would roughly be divided into six main activities:

1. Outlining the methodology of IRL use, including tool support;
2. Defining the IRL language;
3. Defining basic concepts and developing a concept library (domain theory);
4. A pre study of tool support and outlining the functional requirements of tools;
5. Developing a detailed methodology of IRL use;
6. Developing a tool prototype/demonstrator.

The work should be carried out in an international context with a reference group formed by representatives of interested administrations and possibly suppliers. It is important that a coherent working team is chosen and that the work is not split between too many parties. It is presumed that the development project be staffed with formal methods experts who also have relevant knowledge of and experience from the railway signalling domain.

Possible funding sources could be: national R&D programs such as that of Banverket and other administrations with an interest in formal methods, international projects of the UIC and EU, and vendors interested in the use of formal methods.

(The *language definition* corresponds to selecting the formal language and defining a subset and coding rules according to section 2.1. The *tool prototype* *development* corresponds partly to the selection of appropriate tools and performing appropriate adaptations and tailoring according to section 2.2, but also involves developing prototypes (or demonstrators) of tools relating to interlocking development.)

5.2.2 IRL and EURO-INTERLOCKING

This project outline fits well into the long-term goals of EURO-INTERLOCKING. The EURO-INTERLOCKING project provides just the kind of international context that is necessary to ensure general acceptance of the concept.

The close cooperation, or even integration, of IRL and EURO-INTERLOCKING work would be beneficial for both. The EURO-INTERLOCKING requirements could serve as a trial application of IRL. Successful application of IRL to the wide span of different national requirements, with a common conceptual core comprising the EURO-INTERLOCKING requirements, would demonstrate the general usefulness of the language. Since the EURO-INTERLOCKING requirements would provide the principal test cases while IRL was developed, the EURO-INTERLOCKING project would be ensured of a notation well suited to its needs.

Banverket’s ISL concept does go further than the EURO-INTERLOCKING project itself since it also considers the use of ISL in the development and verification of interlocking systems – something that is primarily the interest of suppliers. However, a long-term strategy for IRL should also consider this aspect since the use of formal methods in development is an important factor in life cycle cost reduction. The successful application of IRL to formal specification in EURO-INTERLOCKING would raise interest also in the application of IRL to interlocking development and verification.

EURO-INTERLOCKING should support or even itself undertake to organise an IRL development project. The work should initially concern itself with the specification aspects of IRL to provide the necessary support to the EURO-INTERLOCKING work. Once this phase is completed, the work on interlocking development aspects could begin. The aim would be to develop methodologies and prototype tools which would demonstrate to suppliers and administrations the advantage of using formal methods for development and analysis of interlocking systems. The cooperation of suppliers could be sought for this concluding phase of the project.

6. Conclusions and recommendations

The use of a formal notation for expressing interlocking requirements will lead to a higher quality requirements definition and reduce the costs of requirements maintenance and validation.

The principal conclusion of the study is that the B notation [Ref. 3] provides the best basis for the development of a formal notation – an Interlocking Requirement Language (IRL) – for expressing the EURO-INTERLOCKING requirements.

The approach has also been identified, which ensures that the requirements can be expressed rigorously in natural language. This approach, based on object modelling, is designed to be consistent with the eventual use of the IRL notation, smoothing the transition between the two styles.

In Section 5 of this report, it has been presented an action plan to meet both the objectives of Project 2 and the longer-term requirements of the EURO-INTERLOCKING project. It is therefore recommended that:

1. Work in Project 2 follows the action plan described in Section 5.1, that is:

    - Develop object models for the basic concepts underlying the EURO-INTERLOCKING requirements, and provide clear definitions of all terms.

    - Express the requirements in natural language using the approach and style described in Section 4, using terms taken from the object model.

    - Provide close liaison between the Core Team and the IRL developers to ensure that there is a close match between the basic building blocks provided by the IRL and the core components of the object model. This is clearly a two-way process.

    - As the IRL takes shape, start to express the fundamental safety constraints in the notation.

2. EURO-INTERLOCKING supports or even itself undertakes to organise an IRL development project. Funding could be sought from one or both of the UIC and the EU, and additionally from national R&D programs such as that of Banverket, as the IRL notation has potential benefits to the railway industry beyond the immediate scope of the EURO-INTERLOCKING project.

Amendment Sheet

No.	Version	Section Amended	By Whom	Amendment	Date
1	0.1		GF	First draft report outline	14.03.00
2	0.2		GF	Revised report outline following comment	15.03.00
3	0.3		GF	Revised report outline following meeting in Stockholm of 23rd March. Some initial material provided for Section 4.	27.03.00
4	0.4		GF	Full draft of Praxis Critical Systems' contribution to the report, pending input from Industrilogik	06.04.00
5	1.0	2, 3.1, 5.2	GF	Definitive version of the report, incorporating input from Industrilogik	13.04.00
6	1.1	1.1, 2, 3.1.3, 3.2, 5	NK	Minor changes for clarity of report. Major changes to section 2 to include objectives and justification for IRL	15.05.00