Chapter 6

Case study: Flexport

6.1 Introduction

In this chapter we present an industrial case study to test out some of the methodology described in previous chapters. This consists of the formal analysis of part of the Universal Flexport protocol for medical device communications[Spa89] which we introduced in section 2.7.1. The purpose of the treatment is twofold: to examine the ability of the formal approach to model and analyse the protocol; and to validate the protocol as effectively as this approach allows. The formal context is the refinement of a specification that is to incorporate successively more detail which would lead eventually towards the modelling of the whole of Flexport. The main languages we have used are LOTOS and temporal logic (viz the modal mu-calculus), as recommended in Chapter 2.

Since the protocol is complex, we start with a simple specification of a part, and then continue refining this gradually in a rigorous manner. We envisage several 'paths' of refinement as we investigate the formal development using a number of approaches and techniques. The heart of the formal design is in the form of specifications of the link connection phase. We support their analysis and development by distinct approaches to verification and validation – the main 'machinery' – which we contrast and compare within the context of ongoing refinement. The tasks are summarised in the next section.

The validation typically consists of proving safety related properties, particularly liveness. This key requirement has obvious implications for real-time operation, which we can usefully examine in an abstracted setting without explicit mention of time. The validation is achieved through simulation and property testing, some of which are preserved through CPTs (see section 3.6.1).

These kinds of task have been reported to varying extent in other work, usually with the attention on issues such as expressivity of particular formalisms and computational

concerns. There are few examples of an engineering approach to such case studies, with the odd exception such as a comparative case study of formalisms for automatic protection switching [ACJ+96] which assesses various formalisms according to software engineering criteria. However, the specification is small (26 states, 138 transitions), and although some ongoing development is mentioned, there are apparently no further reports. In our case study, the intensional specificatin starts off small (44 states, 60 transitions), but becomes much larger during refinement – thousands of states and transitions.

Since communications protocols have been much researched, we also are able to test our techniques with some knowledge of what to expect. Yet, although safety requirements such as avoidance of deadlock and livelock have been checked for numerous systems in the past, there is seldom any indication of their being derived from standard safety analysis techniques. Further, somewhat strangely for an analytical system, there is relatively little guidance on how to identify causes of any problems that may arise, let alone the statement and subsequent demonstration of requirements for their resolution. Some useful work on procedures has been done, however: see e.g. the use of reachability analysis in [LS87] to determine hazardous states of Petri Nets.

It should be noted that the case study had already begun before the various procedures had been developed, so some of the CM required a kind of reverse enginering: for the ideas to be more properly tested would require a fresh case study.

6.2 Instantiating in the Lifecycle Framework

In this section we set the development in the context of the lifecycle model developed in section 3.4.1.

We choose Σ to be an Intensive Care Unit (ICU), consisting of a number of interconnected medical devices, and a subsystem S to be the communications protocol for these devices – in this particular case, Flexport. The choice of such a protocol is fitting since it is a safety-critical element that controls part of the communications system in the ICU, and we may expect safety analysis for the communications system to be derived from safety analysis of the ICU since the former may well depend upon the operating environment of the devices. Medical protocols have to address, in particular, the need for 'plug and play' in which a device may be easily connected to a system and should be soon up and running safely and reliably alongside other devices.

6.2.1 Main Terms

We now proceed systematically to illustrate the methodology above, starting by making explicit what we mean by each of the respective terms.

- *requirements definition* Using the terminology defined there, we first note that user requirements (USER-REQS) were not available, so we devised a very general statement:

  > the safe and dependable transmission of medical-related data between a third party system and a SpaceLabs system.

  This view brings out the key elements required for a safety case. For SYS-REQS, we have the Flexport protocol which specifies how communication should be carried out between a third party system and a SpaceLabs system. This details how this is to be effected through the use of three layers, termed *Hardware Interface*, *Low Level Link Interface*, and *Upper Level Link Interface*.

- The *design* consists primarily of specifications in LOTOS that are being refined towards implementation..

  Initially we use the document to conceive an architecture for the specifications and then draw up a plan for implementation. The system will be constructed with three aspects: the provision of a main structure to provide the necessary functionality; the handling of all faults associated with or impinging on the various components of the structure, revealed at various stages; and methods of control for the specified faults, derived from the safety analysis.

  The process of construction will follow the procedure **FTBuild**. We hope to show that this procedure allows closer attention to safety requirements – their derivation and validation; and that the integrated use of fault trees is a means both to identify fault causes and then to help in proposing further requirements plus changes in the model.

- We concentrate on two tasks for the verification of the protocol specification. The first is to show the consistency of two distinct views, namely the peer-to-peer working between devices and the how this is achieved through internal services within the devices. These are termed *extensional* and *intensional*, respectively and detailed later. The second is to show the consistency between the behaviour of different versions of a given design item. Preservation of behaviour is in terms of both symmetric and

asymmetric relations as discussed in section 2.6.1; in some cases, these ensure the preservation of properties; if not, then model-checking may be repeated. Overall preservation is in terms of property conformance defined in section 4.4.

- We use the methodology of chapter 4 to perform two activities for the *validation*:

  1. derive requirements which the model must possess for it to be a valid implementation of (SYS-REQS), the protocol definition. These fulfil the *functional requirements* to implement 'correctly' and fully the requirements of the Flexport protocol.
  2. derive safety requirements for the validation of Flexport based on the requirements definition (USER-REQS) and the safety analysis. For our illustration, we choose just one top level fault, develop a partial tree and formalise some requirements in temporal logic, specifically the modal mu-calculus. To check the properties, we use various tools to initially show the properties directly and subsequently either re-iterate the check or use CPTs (see section 3.6.1). These are part of the *non-functional requirements*, which in practice also include non safety-related requirements such as performance requirements, which are not addressed here.

  The user requirements for the case study and the means to retain the focus on these as the project progresses are discussed in the next subsection.

6.2.2 Requirements Analysis

A protocol such as Flexport is originally a response to the requirements of hospital staff, so they determine the user requirements. An assumption seems to exist that protocol engineers know already what kind of requirements are needed by any such communication system, but this should not be taken for granted. Thus, a questionnaire was devised (see Appendix H) to try to gain some further appreciation and understanding about the kinds of hazards that may be encountered in an ICU, especially with respect to the emerging MIB standard. Unfortunately, time and other constraints have meant that we have been unable to carry out the survey. However, this remains an important consideration, not least as a way of ascertaining the awareness of and reception to such a standard amongst those who are most likely to be exposed to it.

The absence of such data is not so critical for this project since this is only a prototype development to test the framework, which requires some rudimentary knowledge

of the operating environment to test the ideas. It sufficed to iterate the following analyses throughout the development to keep requirements to the fore.

Hazard Analysis, Risk Assessment

In the case of Flexport we must treat the hazards that exist in communication between devices. To facilitate some of the brainstorming, we used a selection of guidewords for communication protocols (not conforming to any particular standard), loosely based on SHARD[FJMP94]. These are given in the Appendix B.

For this small scale, we do not formally determine the associated risks, but do give some measures to be implemented.

Safety Integrity

The specification is built stepwise, incorporating both functionality and methods of control for risks that arise from hazards. We anticipate that the protocol has been designed to account for these risks, but such analysis is not indicated. Thus, for our formal development we have to make informal judgements in deciding both what we regard as methods of control for a given risk and what parts of the Flexport document we consider to specify them. We indicate how we expect the protocol to provide these methods of control to effect risk reduction, preferably risk elimination for all the risks we may identify.

In order to show that safety integrity of the system is assured we have to use a formal analogue, since we are developing a non-executable specification. Thus we formulate the safety requirements in some formal language (another subjective step), not necessarily LOTOS, and then seek to demonstrate that this formulation inconsistent with the specification(s). This then becomes a task of verification. Subsequently, our formalisation can validate the definition of Flexport, possibly pointing to ambiguities, incompleteness etc. For instance, if we identify a fault or hazard which, we believe, the protocol, does not handle, then we shall specify it and make some recommendation for the protocol.

6.3 Overview of the Flexport protocol

We now turn to SYS_REQS. The definition of Flexport is provided in a document that consists of a mixture of text description, charts, tables and diagrams. It defines the connection of a Spacelabs system (*station 1*) to a third party device (*station 2*) and consists of 3 layers:

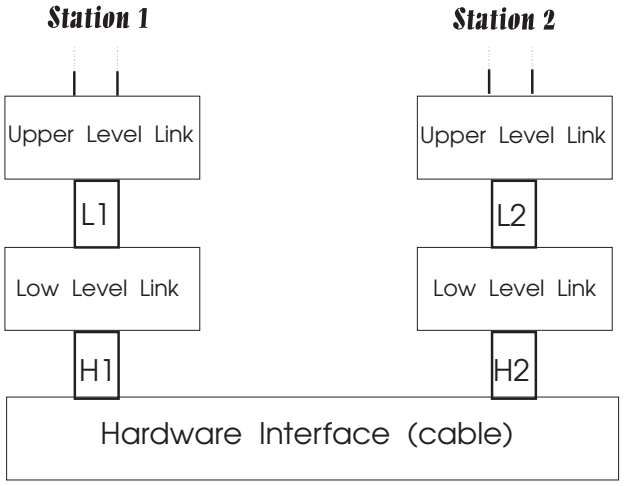

Figure 6.1: Flexport's layered architecture

1. *Hardware interface*, providing a data transmission service between stations 1 and 2, with no handshaking.

2. *Low Level link interface*: a connection oriented protocol with low level link control to implement secure data transmission.

3. *Upper level link interface*: used to actually communicate numeric and waveform data.

We conceive the architecture as in Figure 1, where H1, H2, L1 and L2 are regarded as *Service Access Points* (SAPs). Layer 1 may be viewed as corresponding to Level 1 of the OSI reference model, with Layers 2 and 3 corresponding to Level 2.

The hardware layer specifies an asynchronous protocol, with ASCII encoding and a full duplex transmission mode, thus somewhat different from the MIB which is bit-oriented and half duplex. The transmission of data between the stations is in terms of *packets* – given as data (called a *frame*) wrapped in some other bytes giving information about the frame (page I-5,[Spa89]). The packets have to be passed down to the low level link and then converted into characters for subsequent passing on to the other station via the hardware interface. Thus the low level is called upon to provide services supporting the upper level's transmission of these packets. The low level link control may be regarded as one such service for the upper level link.

For our applications, we restrict our attention to the Link Connection phase for which a connection mode service is specified as an enumerated list of points, given in Table 6.1. This phase is evidently a necessary precursor for any session to take place, and is certainly non-trivial. Thus it is useful to analyse it, especially since rigorous treatment has

Link Connect
1. When trying to connect, Station 1 sends ENQ once a second.
2. When station 2 receives an ENQ, it sends DLE,X_ON.
3. When Station 1 receives DLE, it sends X_ON
4. If the character received after DLE was not X_ON then goto 1.
If the character received after ENQ is not X_ON then goto 1.
5. Station 1 sends its Link Config Packet
6. Station 2 acknowledges receipt of the Link Config Packet
7. Station 2 sends its Link Config Packet
8. Station 1 acknowledges receipt of the Link Config Packet. The two stations now use the lowest common settings in the link config packets.
9. Station 2 starts sending its data automatically to station 1 at an appropriate data rate (typically 1 reading per second for continuous data).

Table 6.1: Flexport Definition: Link Connection

previously been sparse. The importance of analysing this phase for medical protocols has already been established by initial work on the MIB [CN92, NC96]. It is hoped that that this work on a simpler protocol may provide useful information for the emerging IEEE1073 standard.

Below the list of points there is a finite sequence, also numbered, which gives a temporal ordering of events to provide Link Establishment. This is reproduced in Table 6.2. Such a sequence is interpreted as a trace. The events of the trace are a mixture of ASCII codes – which we term *protocol data units* (PDUs) and packets (of which we use just the one – Link Configuration). This particular trace is an example of a *message sequence chart* (MSC), describing in this instance the most desirable sequence for the achievement of link establishment. In general, MSCs are simple and intuitive means of specifying and representing in a graphical notation the exchange of data between two or more components. They are often used to define requirements for selective aspects of protocols, especially for telecommunications. MSCs may also be given a formal semantics, thus providing a sound basis for validation [LL94]. MSCs are subject to international standardisation by the ITU [Int93, Int96].

6.3.1 Intensional and Extensional Views

Communication protocols possess two aspects: peer-to-peer protocols and inter-level services. Peer-to-peer communication is dependent upon the consistent performance

```
Link Establishment:     Message Sequence Chart

    Station 1                 Station 2

1. ENQ              ---->
2.                  <----      DLE
3. X_ON             ---->
4.                  <----      X_ON
5. (Link config)    ---->
                               ACK
6.                  <----      Sequence #
7.                  <----      (Link config)
    ACK
8. Sequence #       ---->

         data transmission starts here
```

Table 6.2: Flexport Definition: Message Sequence Chart

of services of lower levels. Thus to aid clarity, we make in effect an initial design decision to separate concerns by using these two distinct views to generate separate specifications which are called *intensional* and *extensional*. According to the terms of the main framework, these will require *verification* of mutual consistency and *validation* of reliability. This motivates the provision of two specifications for the connection phase of the Low Level Link, corresponding to two views from different levels.

1. An *extensional* specification which provides a high level description of the desired behaviour, treating lower levels as a black box;

2. An *intensional* specification which possesses a more detailed description of the desired behaviour, especially services carried out in lower levels.

The use of intensional and extensional specifications is quite widespread, and there has been work done in LOTOS for the ISO reference model [CN92]. Here we derived the intensional specification from the main protocol, so where there is reference to 'the [main] specification', it is this pecification we have in mind. The extensional specification, being much simpler, was derived from the MSC. As the MSC is just a trace, the extensional specification needs no architectural design.

6.4 Configuration Management Plan

In this section we present some aspects of Configuration Management to indicate how they support the formal development. Our reference source is [Whi91]. This does not constitute a complete project: for instance, the development is the work of an individual with periodic informal reviews as befitting a research studentship. However, it is hoped that it does serve to give a flavour of how the framework can operate on a larger scale.

6.4.1 Classification of the items in the system

Listed below are the item types and subtypes (enumerated) and their instances (bulleted) which are used in the project. A full project would require the delivered items to be as complete as possible.

1. Documentation

   (a) Client Definitions and other Official Documents

   - Universal Flexport Protocol

- ISO 8807 LOTOS standard
- Fault Tree Handbook
- References to other guidelines, e.g. background to CM (Whitgift), safety techniques (esp. DEFSTAND00-58), formal theory, data communications

(b) Technical Reports – Overview and Results of Development

i. KUCSES Technical reports presenting overview of work in progress

(c) Safety Case

- Hazard Analyses
- FTA (and other safety analysis)
- CM Plan (*sic*)
- RM Logs
- Tools Summary

2. Software

(a) Specifications in LOTOS text

i. ISO libraries for basic Data types

ii. intensional

iii. extensional

iv. tests

A. trace tests

B. property tests

C. conformance tests

(b) Transformations/other representations of specs

i. Transition graphs

.CR 'Common Representation' format used in LOTOSPHERE

.fc2 transition system in fc2 format (a common format for various suites of tools)

.bcg a compact representation using BDDs.

.net an interpreted Petri Net format (generated by Cæsar)

.gph finite state automaton (graph)

.m0 graph in AUTO format

.aut graph in ALDEBARAN format (generated by Cæsar)

ii. C code

A. aid to compilation to LTS

B. executable code

iii. Pictorial representations

- G-LOTOS graphs
- State Transition graphs generated by `bcg_draw`

We used a variety of tools on a SUN 670, running UNIX. All these and their versions are given in Appendix C.

6.4.2 Baselines

Most of the effort for Flexport is devoted to supporting the LOTOS specifications and how they are composed in order to model the protocol. Hence the baselines, which we specify below, are in terms of this evolving model. They start off with a highly simplified structure, and later branch out to two alternative refinement paths that are conceived as milestones to aim for. Further sub-baselines are introduced to indicate changes at varying levels of abstraction. Baselines may be regarded as phases at which new user requirements are introduced, reflecting the cyclic nature of the design. It is important that these requirements are introduced in an appropriate order so that big structural changes are not required later on.

1. Basic structure of architecture; Link Connection using 1 place buffer, perfect channel, no special features implemented.

2. As above except implement $n-$place buffer

3. Add Baud rate hunting

4. Fault Inclusion

   - lossy channel
   - corrupting channel
   - message re-ordering channel

5. *Branch to two paths*

(a) Path 1

- Fault Tolerance
  i. Add Flow Control
  ii. Add Error Recovery
- Add Data Transmission phase
- Reify data – detail Packet types definition

(b) Path 2

- Add Data Transmission phase
- Detail Packet Types definition
- Fault Tolerance
  i. Add Flow Control
  ii. Add Error Recovery

This is illustrated in diagrammatic form in Figure 6.2 as an instantiation of figure 3.2:

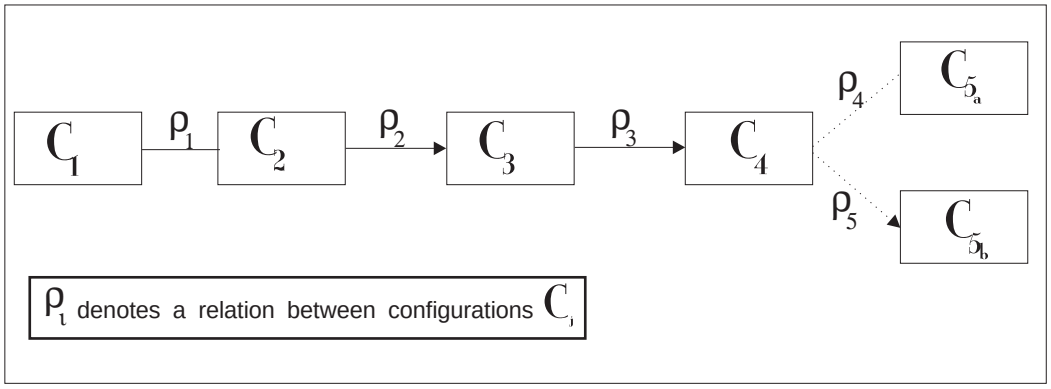

Figure 6.2: Refinement Graph of Main Baselines

The figure shows the refinement at a high level of abstraction, with C_i denoting the baselines and the ρ_i denoting refinements between baselines. At a deeper level of granularity, we have, e.g., ρ_3 is itself a sequence of successive subrefinements – the introduction into the channel of lossiness; data corruption and message re-ordering. Similarly, ρ_4 and ρ_5 consist of subrefinements for the steps in paths 1 and 2 respectively.

6.4.3 Item Identification

This is the task of providing each item with a unique and meaningful name. We choose names to indicate the baseline as well as describe its function. A naming scheme

could be based on the item hierarchy and have as many parts as the number of levels. Our project is small by CM standards, so we choose names to have 2 bits plus version number:

$$< name > . < extension >, < version >$$

names include:

- `flexint`: flexport intensional specification
- `flexext`: flexport extensional specification
- `flexdata`: data types definition for the specifications
- `station1`, `station2`: stations 1 and 2
- `buffer`: hardware channel
- `unitestmsc`: Unified Tester for MSC-based specification
- `<spec_name>_t` : template for `<spec_name>` (see section 6.5.3)

extensions include:

.lotsrc source file for LOTOS behaviour

.lotdat LOTOS DATA Types source file

.lot LOTOS specifications for LITE toolset[PvEE92] (usually derived, so built from relevant sources)

.lotos LOTOS specifications for CADP toolset

.cw CCS specification for Concurrency Workbench

Version numbers are maintained for source components and also for some derived elements, being simply the tuple of versions of the respective sources. The numbering is a numerical ordering governed by `rcs` [Tic85]. Source versions are indicated as close to the front as possible – e.g., in initial comments for a specification, and on the front page of documentation. For derived elements, the composition may be gleaned by searching through for the histories that are part of the included sources.

6.4.4 CM and Version Control

Even for a relatively small case study such as this, it is useful to keep careful watch over the various components of the development. Hence we use aspects of version control under `rcs` together with `Make`[Fel79], to cover the specification sources and derived elements. We consider in detail below one main software item, the intensional specification, `flexint.lot`, which is a derived item that has four source components: `flexdata.lotdat`, `buffer.lotsrc`, `station1.lotsrc` and `station2.lotsrc`. Each of these components is placed under version control, so the intensional specification is also a configuration.

Version control needs the structures to be determined beforehand. Guided by the principles of abstraction and decomposition, the baselines expressed at configuration level may be decomposed in terms of the sequences of versions planned for each item and such items may be similarly decomposed etc. Once a complete component structure of the whole system has been established (so extending the list of item instances above), then one can proceed to take the top-level baselines, identify which components are within the scope of the various changes, e.g. 'add baud rate hunting' affects `flexint.lot` which in turn affects its component station 1, but not 2. Similarly, alternative paths will be reflected in one or more components.

In this way, one can generate the order of construction for each item, yielding proposed stages of development for the intensional specification as follows.

1. `flexint.lot` *derived item*

   Versions are in terms of `flexdata.lotdat`, `buffer.lotsrc`, `station1.lotsrc` and `station2.lotsrc`

2. `flexdata.lotdat`

   (a) Boolean and Natural numbers from `mod-is.lot` (supplied with LITE3.0) plus definitions for PDU's

   (b) addition of string type (for n-place buffer)

3. `buffer.lotsrc`

   (a) 1-place buffer, perfect channel

   (b) make n-place buffer

   (c) add lossiness

   (d) add data corruption

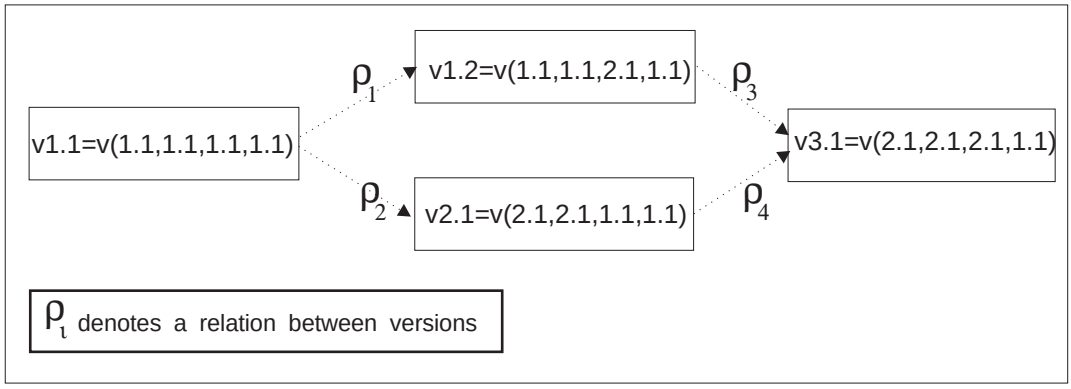

Figure 6.3: Refinement Graph of Intensional Specification

4. `station1.lotsrc`

   (a) simple PDU dance, no features implemented

   (b) add baud rate hunting

   (c) Add Flow Control

   (d) Add Error Recovery

   (e) Add packets

   (f) Add Data Transmission phase

   (g) Reify data – detail Packet types definition

5. `station2.lotsrc`

   (a) simple PDU dance, no features implemented

   (b) Add Data Transmission phase

   (c) Reify data – detail Packet types definition

The version of the composite item `flexint.lot` can then be expressed as a tuple flexint.lot,vx.y = $v(v_1, v_2, v_3, v_4)$, where v_1 denotes the version number of data types; v_2 the version number of the buffer; v_3 the version number of station1; v_4 the version number of station2.

This can then give the refinement path indicated in Figure 6.3. In Appendix E there are given two tables which cover in more detail the versions (planned and actual) for the sources and some derived items.

At this level of granularity we can specify the relations we seek for the items in the refinement trajectory. Since these items are formal, these relations express precise

mathematical properties (as mentioned in section **??**). Thus, for ρ_2, we need to show that the new buffer is a *generalisation*. Through the compositional structure this needs analysis of the buffer item. Version $v1.2$ is technically a variant, but we consider it as an alternative path on equal footing with others. The alternative path starts by implementing baud rate hunting. Both should converge at version v_3.

Let $Buff(n)$ denote the model of a general Buffer of capacity n. Then the requirement that the the new buffer is a generalisation of the old is expressed as:

$$Buff(1)_{v2.1} \quad equiv \quad Buff_{v1.1},$$

where $equiv$ denotes strong equivalence.

Similarly, replacing the simple linear trace in `S1_sendENQ` by a Baudrate hunting routine should be another kind of generalisation, which we can model by trace inclusion:

$$Tr(Station1_{v2.1}) \subseteq Tr(Station1_{v1.1})$$

where for processes P and Q, $Tr(P) \subseteq Tr(Q)$ if and only if the set of traces of P is contained in the set of traces of Q.

In this work there is no formal procedure for approval of items which are placed under version control. However, we specify that at the minimum, each LOTOS component – from the definition of Data Types to a full specification of Flexport – is to be checked for correctness of syntax and semantics. Composite items are to be 'tested' more rigorously through additional tasks, e.g. simulation, property testing and verification according to the respective requirements. Item statuses are awarded accordingly. In a complete project, one could stipulate that an item may not be approved before all the risk management requirements have been satisfied.

6.5 Overview of system construction

In this section we present a summary of the steps involved in producing the various specifications.

6.5.1 Architectural Design of the Intensional Specification

The LOTOS specifications have architecture corresponding to Figure 6.1 and are built in modular fashion, having three components, corresponding to the two stations plus the physical layer. The communication process is modelled in terms of synchronisation at

gates – $h1$, $h2$, $l1$ and $l2$ – connecting adjacent layers; *actions* are instantaneous occurrences, consisting precisely of these gates with the offering of values, plus a few other special events. The level of interaction between components is governed by the parallel operator ||, which specifies the set of gates for which actions have to be mutually agreed; independent behaviour is modelled by interleaving using the ||| operator. The specification thus has structure given by:

```
 (Station1[h1,l1]  |||  Station2[h2,l2] )
|[h1,h2]|
  Duplex_chan[h1, h2]
```

where `Duplex_chan[h1,h2]` denotes a call to a process that models the behavious of the buffer.

For each station, we distinguish between two main phases of the Link Connect and Data Transmission, specifying that the successful completion of the former enables the latter to go ahead (denoted by the enable operator '>>'). Thus, e.g., Station 1 has process definition:

```
process Station1[h1,l1] : noexit :=
   S1_Connect[h1,l1] >> S1_DataTransmit[h1,l1]
endproc (* Station1 *)
```

Similarly, the link connection is itself can be viewed as composed of 3 subphases and specified as:

```
S1_SendENQ[h1,l1] >> S1_TestX_On[h1,l1] >> S1_SendLCPack[h1,l1]
```

for Station 1, and

```
S2_AwaitENQ[h2,l2] >> S2_TestX_On[h2,l2] >> S2_SendLCPack[h2,l2]
```

for Station 2.

Once the connection is established, we model repeated data transmission as simply as possible, just as a repeated action at the gates $l1$ and $l2$.

6.5.2 Behaviour

To model behaviour, we introduce primitives, chosen to reflect the view that data is identified as *sent* and *received* to/from a given level, as indicated in the MSC. All 'sends' indicate a transmission downwards, whilst all 'receives' indicate transmission upwards.

LOTOS has no built in data types – these have to be constructed, though there is available as part of the ISO standard a library of basic types (such as Booleans and Natural Numbers) and these are supplied with the distribution of most tools. We made some use of these. For the specification we defined PDUs and packets to correspond to the ASCII codes and packets used in the Link Connection of the protocol. These were implemented in LOTOS by using parameterised actions of the form $g!send(v)$ or $g!receive(v)$ where g is a gate, and the $[send/]receive(v)$ is some parameterised value. Hence, e.g., `h1!send(ENQ)` denotes the sending by Station 1 of the PDU 'ENQ' at the SAP 'h1' down to the physical layer.

As an example, the `S1_TestX_ON[h1,l1]` fragment below (from version 1.1 of `station1.lotsrc`) tests for the receipt of the Transmission On indication `X_ON` from Station 2:

```
process S1_TestX_ON[h1,l1] : exit :=

  h1!send(X_ON);
   (choice x:PDU []
     h1!receive(x); ([x eq X_ON] -> exit
                    []
                     [x ne X_ON] -> S1_connect[h1,l1])
   )

endproc (* S1_TestX_ON *)
```

During the connection phase for these specifications, the upper level link's only involvement is in the sending and receiving of the Link Configuration Packets. Hence, the extensional specification's behaviour is a short trace.

Regarding the hardware interface for the intensional specification, the full duplex mode is modelled using interleaving of two simplex channels.

6.5.3 The use of a template for the intensional specification

Component source items are glued together to form a valid LOTOS specification through the use of a *template* which is pre-processed using a macro processor – we employed `m4`, which is commonly available as part of distributions of UNIX. Among the templates was one to generate the intensional specification, which 'included' the data types definition, buffer definition, plus definitions for each of the stations, given as follows:

```
(*******************************************)
(*                                         *)
(*  LOTOS intensional specification for    *)
(*  Flexport Protocol Lower Level Link Layer  *)
(*                                         *)
(*******************************************)

(* $History$ *)
(* $Log: flexint_t.lot,v $
# Revision 2.1  1997/04/23  11:56:31  cs_s447
# This version supports Baud rate hunting
# with two extra gates 'baud' and 'tick'
#
# Revision 1.2  1997/04/23  11:53:40  cs_s447
# library call for data types omitted
#
# Revision 1.1  1997/04/23  11:51:34  cs_s447
# Initial revision
#   *)

specification Flexport [h1,h2,l1,l2,baud,tick] : noexit

include(flexdata.lotdat)

behaviour

 (
  (  Station1[h1,l1,baud,tick]  |||  Station2[h2,l2] )
   |[h1,h2]|
  (  Duplex_chan[h1,h2] )
 )

where

include(buffer.lotsrc)
include(station1.lotsrc)
include(station2.lotsrc)

endspec
```

6.5.4 Refinement and Verification

The refinement is carried out within the procedure **FTBuild** and is driven mainly by two aspects: the target baselines and the outcome of ongoing safety analysis. Approaches to refinement include analysing and implementing transformations such as process refinement and action refinement. These bring with them obligations: certain specific tasks of verification and validation have to be iterated for new versions. We summarise here the approaches to verification for Flexport – the validation is discussed in the next sections.

In this project we use side by side two approaches to the formal verification. These approaches may be called 'internal' and 'external'. To verify consistency by an internal approach, we seek to show the symmetric relation, observation equivalence, and also the asymmetric relation, the reduction preorder through the generation and comparison of the expanded trees for the relevant specifications. Apart from performing model-checking directly on each specification in turn, the 'external' approach of testing may be used as described in the previous chapter. For this, we perform simulation of test compositions, including most notably those involving unified testers.

We aimed to use a selection of tools with the emphasis on minimal intervention, which we regard as an important factor that would receive more favourable consideration from industry, as argued in section 2.6.3. A summary of the tools and their versions is given in Appendix C. The specifications were built mainly using the LITE toolset, with simulation performed in SMILE[EW93]. For validating properties in the modal mu-calculus, we used Cæsar[FGM+92] to translate the LOTOS specification into a corresponding LTS, saved as an automaton and then translated to FC2, a format designed for file exchange between different tools; after manual editing of action names, this was imported into the Concurrency Workbench[CPS89] where formulae in the modal mu-calculus were validated.

In the next two sections we present two iterations of the procedure **FTBuild** including the derivation of safety requirements, their formulation as properties and their validation; the tasks of verification are also covered.

6.6 Applying FTBuild : First Iteration

In the first specification we wished only to check the integrity of the 'PDU dance' which is designed to establish link connection. In order to make things as simple as possible,

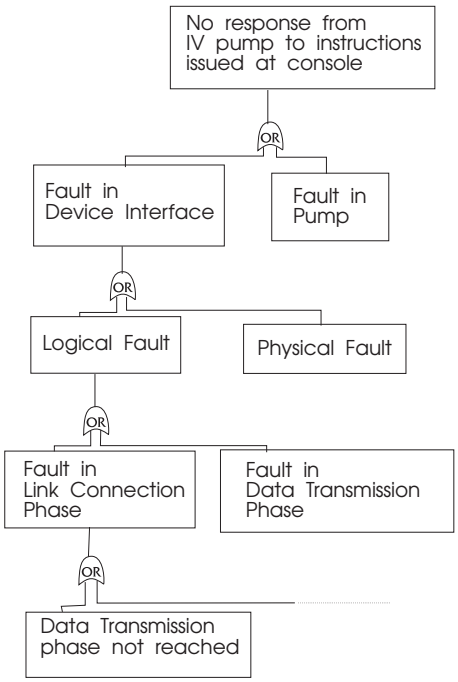

Figure 6.4: Fault Tree for system (ICU)

our initial (intensional) specification, S_1, say, omitted all the special features. Further, we assumed a one place buffer for each of the channels – once a low level link sends a character down the physical layer, then it cannot send another until the character has been received by the low level link of the other station.

6.6.1 Fault Tree Construction

We constructed an initial informal tree to show how the communication system is a component of the overall system. This is given in Figure 6.4.

For the protocol, we choose to start our formalisation by considering the node 'Data Transmission phase not reached'. Hence, a sample iteration of the procedure **FTBuild** proceeds as follows.

1. Select the event E: 'Data Transmission Phase not reached'
2. Event causes are:
   - E_1: 'Deadlock [in link connection stage]'
   - E_2: 'Livelock [in link connection stage]'
3. Formalise E as the atomic proposition 'DTNR'

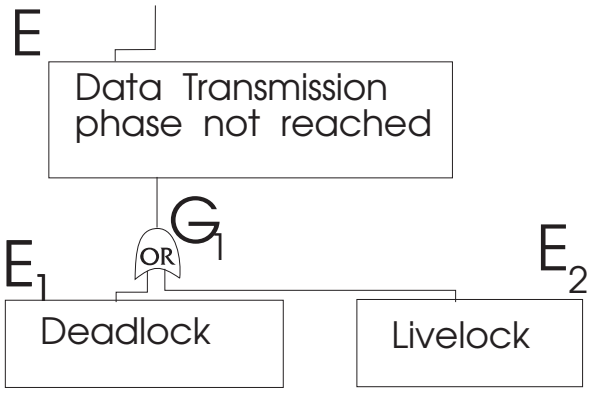

Figure 6.5: The extension to the ICU Fault Tree resulting from the first iteration

4. E_1 and E_2 are alternative causes which we regard as part of a generalisation, thus we have a generalisation-OR gate.

5. The gate is formalised as GOR_1, which is given as: $DTNR \iff E_1 \vee E_2$.

   The initial formalised fault tree has, like the initial specification, a very simple representation. It is given in Figure 6.5, and has as leaf nodes 'Livelock' and 'Deadlock'.

6.6.2 Requirements Derivation

We perform step 6 of **FTBuild** :

1. We choose to omit the formal analysis of the tree

2. The overall consideration is to provide assurance that any execution of the LOTOS specification carries out desired behaviour in a safe, reliable and efficient manner. In response to the events, E_1 and E_2, we elicit the following requirements:

   (a) The specification should be free from deadlock – i.e. data may always be transmitted;

   (b) The link connection should be free from livelock – e.g. there should be no infinite loops during this phase.

   (c) The link is (eventually) established after a finite number of actions.

   The first two requirements are safety properties, whilst the third, a strengthening of the second requirement, is a strong fairness property (so 2c implies 2b).

   We may formulate requirement 2c implicitly by simply stating that some event(s) in the Data Transmission phase should eventually happen or, some other property

should eventually hold. More specifically we could show the (weak) condition that for both stations 1 and 2, some event in the Data Transmission phase, but not in the Link Connection phase, eventually occurs. A stronger requirement is to show explicitly *how* the link is actually established – namely, that all execution sequences (of the intensional specification) contain an initial finite subsequence which satisfies the protocol. Accordingly, we may look to show that all execution sequences possess a temporal ordering of actions conforming to the MSC or to some satisfactory trace based on the MSC. As this shows precisely that link connection is established reliably, our only further concern as regards validation is to ensure that the specification allows easy computation.

The safety requirements and their formalisation are derived as follows:

- **Gate Requirements** We wish to preserve the gate condition, hence $\sigma(GOR_1) := DTNR \iff E_1 \vee E_2$
- **Event Requirements**
  - E_1:

    Informal: 'Specification should be free from deadlock'. This is a safety condition

    Formalisation:

    $$\nu Z. < - > \mathtt{tt} \wedge [-]Z. \tag{6.1}$$

  - E_2:

    Informal: We treat the third of the requirements listed above: 'Link connection should be free from livelock' which we rephrase as 'link connection should established in a finite number of steps'. We propose two alternative formulations which indicate an open-ended problem-solving approach:
    (I)

    > For both stations it is the case that eventually they are both able to perform actions that are only in the data transmission phase.

    If we denote by s_0 the initial state of S_1, then we require:

    Formulation of I:

    $$s_0 \models \mu X.(< S_T > \mathtt{tt} \wedge [-S_T]\mathtt{ff}) \vee (< - > \mathtt{tt} \wedge [-]X), \tag{6.2}$$

where

$$S_T = \{h1!send(DAT), h1!receive(DAT), h2!send(DAT), h2!receive(DAT)\}$$

OR

(II)

S_1 should satisfy some testing relation $\leq_{Rtest}$ (such as the testing or reduction preorder) with an MSC-based specification that makes explicit a set of desirable traces.

Formulation of II:

$$S_1 \leq_{Rtest} MSCBase \tag{6.3}$$

6.6.3 Incorporation of Requirements

For step 6c), we check that the requirements hold for the model by selecting the second conformance relation (definition 4.4.2).

We instantiated the values as follows:

- $\mathcal{F}' = \{E\}$; hence $\mathcal{F} = E$
- $t_{MAX}(E)$ is the tree given in Figure 6.5. (We could have this as the larger tree).
- $\delta(t_{MAX}(E)) = ft(E)$
- $\zeta(\Gamma) = gates(\Gamma)$
- σ and ϵ are defined above; $\varepsilon(g) = \emptyset$

We take $\mathcal{M}$ to be the LTS corresponding to S_1 together with its valuation $\mathcal{V}$.

The conformance relation becomes:

M**conf** $S_1(\mathcal{F}', \delta, \zeta, \epsilon)$ if

$$s_0 \models_{\mathcal{M}} \bigwedge_{\Gamma \in \delta(ft(E))} \bigwedge_{g \in gates(\Gamma)} \left([[\sigma(g)]] \bigwedge_{e \in g} \psi(e) \right) \tag{6.4}$$

which reduces simply to

$$(s_0 \models_{\mathcal{M}} ((\mu X.(< S_T > \mathtt{tt} \wedge [-S_T]\mathtt{ff}) \vee (< - > \mathtt{tt} \wedge [-]X))) \wedge (\nu Z. < - > \mathtt{tt} \wedge [-]Z))$$

$$\vee\ S_1 \leq_{red} MSCBase \tag{6.5}$$

where $S_T = \{h1!send(DAT), h1!receive(DAT), h2!send(DAT), h2!receive(DAT)\}$.

6.6.4 Derivation of the Unified Tester

We describe the derivation of the unified tester (given in Appendix D.2) to test the intensional specification for robust conformance. In this instance the intensional specification is IUT.

We choose the reference specification S, to which IUT must conform is the MSC given in Table 6.2. Then we extend the MSC with all other viable connection sequences resulting from an interpretation of Table 6.1. This leads to the more elaborate specification, given in Appendix B as `TestMSC`. We now require that IUT conforms robustly to `Test MSC`, which is to act as the source S for the canonical tester.

6.6.4.1 Construction of the Tester

As `TestMSC` has finite behaviour and a tree structure, it is an example of the special case detailed in section **??**. Thus we may construct it as a LOTOS process in the manner described there.

First of all, to enable exhaustive testing the data types, we define a super data type 'DataSet' which includes a complete list of equations that enable comparisons between every parameterised action, whether send's or receive's of packets and PDU's. The process body itself consists of a tree of calls to other processes - `TestEvent` and `TestEventinChoice`. The main structure of the tree is the same as that for `TestMSC`.

Each iteration of `TestEvent` receives as parameters the set of events possible at a particular node and proceeds to offer all these events. We now explain the process in more detail:

1. For example, corresponding to the first event in `TestMSC`, which is `h1!send(ENQ)`, the first call is:

```
TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(ENQ))
```

where `TestEvent` reconstructs the acceptable event `h1!send(ENQ)` from the first gate parameter and the data item enclosed in parenthesis. Thus the variable 'g' is bound to `h1` and 'z' is bound to `send(ENQ)`.

In the body of `Test Event`, the first part of the choice expression offers `h1!send(ENQ) ; exit`, plus all the other events in the label set that are possible at the other gates. If IUT synchronises on any of these, then the `fail_gate` action is performed before the process terminates.

The second part of the choice expression does two further tests for more diagnostics: first events of the form h1!zz are offered, where zz is not equal to send(ENQ). Any synchronisation here will give rise to a failed termination via `fail_data`. Finally, the last three terms offer events where neither the gate nor data match `h1!send(ENQ)`, whence a failed termination via `fail_both`.

2. If IUT synchronises on `h1!send(ENQ)`, then this process is completed via `exit` and the next action to be interpreted is the next one in the body of the main process. The procedure is now repeated for the next action in `TestMSC`, i.e. for `h2!receive(ENQ)` and then for subsequent events. When there is a choice in `TestMSC`, one is given in the unified tester together with the appropriate internal action prefixes followed by calls to `TestEventinChoice`, which is like `TestEvent` except that this has two set parameters.

   For example in `TestMSC` there is a choice between the event `h1!receive(ACK)` and `l2!send(LC_Packet)` which yields in the tester:

   ```
   TestEventinChoice[[h1,h2,l1,l1,fail_data,fail_gate,fail_both](receive(AC
   K),Insert(send(LCPacket), Insert_1(receive(ACK),{} )))
   ```

   This tests for matches with the action at 'h1', but note the absence of the gate 'l2'. This ensures that we do not get false failure indications – the tests for 'l2' with respect to its acceptable data are given in the call to `TestEventinChoice` further down the page.

3. Continue in this manner until reaching the end of `TestMSC` and finish each branch of the unified tester with `success;stop` to denote successful termination.

6.6.5 Results

Property conformance (equation 6.5) was shown in various ways through simulation, testing and property checking. The specifications were initially prepared using the

LITE toolset, being simulated in SMILE, which allows the step-by-step unfolding of the behaviour defined by a LOTOS expression. We chose reduction as the testing relation for the requirement expressed in equation 6.3 and constructed the corresponding unified tester – given in Appendix D.2. In this case, the specifications were small enough such that the tools were able to cope and provide conclusive answers. The results for the list of requirements are given in more detail below:

1. *absence of deadlock* Simulation of the intensional specification under SMILE reveals a small tree with few branches, whose simplicity is due to the modelling of the duplex channel by using two one place buffers. There were no `stop` indications, so no deadlock.

2. (& 3.) *reliable link establishment after a finite number of actions* We used the MSC as the basis for deriving a LOTOS specification `UnitestMSc` that was to be used as a tester. The tester took account of behaviour not expressed in the MSC, yet would be acceptable for a system operating according to Flexport. This was a matter of just incorporating alternative interleavings of events in the link connection phase of the MSC.

   We were able to prove that the behaviour of the intensional specification was a reduction of the MSC-based specification using robust conformance testing. Every execution of the test composition below terminates successfully, showing that the intensional specification is a reduction of the MSC-based specification,

```
UnifiedTesterMSCBase[h1,h2,l1,l2,successMSC,data_failure,gate_failure,
twofold_failure]
  |[h1,h2,l1,l2]|
    FlexInt[h1,h2,l1,l2]
```

   Furthermore, following the procedure for verification, we were able to compile the two specifications into LTS representations using CÆSAR and CÆSAR.ADT. We then used ALDEBARAN to compare the two LTS representations for observation equivalence. The output from ALDEBARAN indicated that the intensional and MSC-based specifications were indeed observation equivalent, a very strong result (which did not hold in the subsequent refinement).

For the verification of internal consistency, we showed that the intensional specification, with actions at gates $l1$ and $l2$ hidden, was observationally equivalent to the extensional specification. This was done using CÆSAR and CÆSAR.ADT, for which the specifications required some modification, largely the addition of extra comments in the definition of data types which are interpreted for compilation into a C program representation. We also had to use an alternative definition of the data type for Natural numbers, which was available in a library as part of CADP as there is an error in the ISO library (which is actually picked up by SMILE as well). Some other restrictions are reported in section 6.8.4.

Having carried out these minor modifications, we were able to compile the intensional and extensional specifications into LTS representations using CÆSAR and CÆSAR.ADT. We then invoked ALDEBARAN to compare the two LTS representations for observation equivalence, and it duly indicated that the intensional and extensional specifications were indeed consistent.

We also applied robust conformance testing, deriving a tester from the extensional specification and composing it in parallel with the intensional specification, whose actions at the physical layer were hidden:

```
  Tester_Extensional[h1,h2,l1,l2,success,data_failure,gate_failure,
twofold_failure]
|[l1,l2]|
  (hide h1,h2 in Intensional[h1,h2,l1,l2])
```

Every execution of this test terminates successfully, showing that the intensional specification with gates h_1 and h_2 hidden is a reduction of the extensional specification, i.e. the two specifications are consistent.

6.7 Applying FTBuild : Second Iteration

The second iteration was engaged as a consequence of completing the 1st iteration by repeating step 6c for an initial refinement (S_2) of S_1. This particular refinement was to include baud rate hunting, an example of process refinement. The model that was built is given in Appendix D. We describe below how performing a number of procedures for safety analysis contributed to its construction.

6.7.1 Safety Analysis of Specification No. 2

For S_2, the second version of the intensional specification, we implemented 'Baud rate hunting', the recursive process by which station 1 periodically sends an 'ENQ' until it receives a recognizeable 'DLE' in response. We assumed initially that Station 2 was automatically transmitting at the correct rate. Thus we were only testing additionally the protocol's ability to handle delays in response.

It was somewhat of a surprise then that this modest refinement gave rise not only to state explosion (which is exacerbated by the protocol's duplex mode of operation), but also other problems – simulation of an initial refinement revealing various cases of deadlock; and further exploration revealed livelock. This prompted reexamination of the specification to consider:

1. Is the model a valid interpretation of the system definition of the protocol? If not, what are the inconsistencies in interpretation?

2. If the model is valid and there are still problems, then is the system adequate?

Initially, we assumed the design to be adequate, so we set about trying to determine the faults in the specification. In the examination of the simulation tree of the first versions that included the baud rate hunting process, we identified a potential weakness in the two specifications of station components in their ability to allow faithfully the duplex mode of transmission. This mode allows for a large number of permutations in link connection, for instance, station 2 can receive station 1's X_ON between sending out its DLE and X_ON (as indicated in the MSC).

It was revealed that the model omitted some behaviour which we might expect (specifically, the constant ability to receive PDUs). Thus, the specifications were modifed by squeezing into both stations 1 and 2 the ability to receive data in each phase, and we interpreted the statement "when station X receives Data Y" by an iterative waiting process. This enabled the removal of some instances of deadlock identified earlier, but left many others.

At this stage, we proceeded to use FTA to try to prompt the discovery of further information about what had gone wrong. We analysed the specification so that the nodes 'livelock' and 'deadlock' could be expanded and the faults isolated. We found that the analysis became specific to the way the model is constructed, making use of the structure and style of the specification. In this instance, we sought to locate where problems occurred

within this structure and then to specify as safety requirements that these problems do not occur. Appropriate modifications would subsequently be made.

The procedure was enacted as follows:

1. Specification Structure:

   (a) Specification structured as 3 components – 2 stations and channel.

   (b) Component structure for each station (as detailed in section 6.5.1): the link connection can be characterised as having three distinct phases (we abbreviate Station 1 by St1 and Station 2 by St2).

   ```
   (Station 1) Baud Rate Hunting >> Test X_ON >> Send Link Configuration
   Data

   (Station 2) Receive ENQ >> Test X_ON >> Send Link Configuration Data
   ```

2. Fault Tree elaboration and analysis

   Further analysis of the simulation tree revealed that the main problem that was occurring was the potential for one of the stations to enter a more advanced phase of link connection than the other through a kind of mistiming which we call *phase mismatch*. Hence we identify the following faults which can give rise to both deadlock and livelock:

   (a) Station 1 never reaches phase 3 OR Station 2 never reaches phase 3

   (b) There is a trace which leads to a state in which Station 1 (Station 2) is in phases 1 or 2 whilst Station 2 (Station 1) is in phase 3

   These give rise to the fault tree in Figure 6.6.

Fault (b) revealed some limitations in the guidelines for fault tree construction as expounded in [VGHR81], particularly the sections V.1 – V.7. 'Station 1 is in phase 1 or 2' whilst 'Station 2 is in phase 3' is a kind of conjunction and at first glance the 'AND' gate may seem appropriate, bearing in mind that these events are dependent. However, neither of these events are faults in themselves, so there is no point in investigating them in isolation. The fault lies entirely in the nature of the interaction of the stations as constrained by the communications channel.

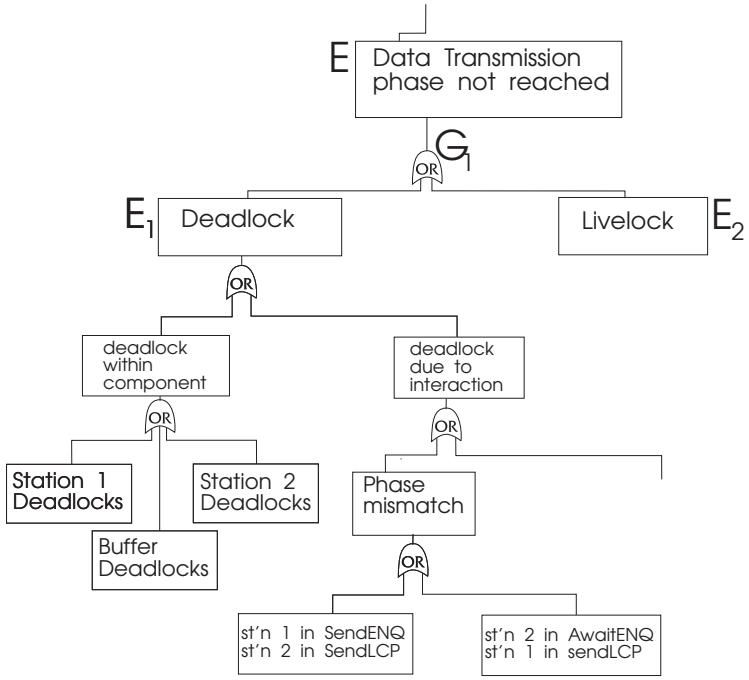

Figure 6.6: Extension to the ICU Fault Tree resulting from the second iteration

The Fault Tree Handbook appears to omit discussion of such interaction between events at the same level – the inter-relationships treated are consequences of individual effects, not of several events. This is amplified in the restrictive "immediate cause concept", which forces complex behaviour to be squashed into single events.

As the handbook seems much more geared to successively working back to individual failure mechanisms, such analysis is not well supported and we have left these as leaf nodes. For further analysis, we explored the fault's cause by using the simulation tool, SMILE.

6.7.2 Safety Requirements

A larger number of requirements can be generated as a result of the new events generated in the fault tree. For the new leaf nodes it appears desirable to have a state-based requirement, which in the case of LOTOS would be a means to specify requirements in terms of the processes that have been identified. Unfortunately it is not generally possible to formulate these directly in a temporal logic such as the modal mu-calculus. Instead, we have to use formulations in terms of event sequences and their temporal ordering, making use of features such as the exclusive occurrence of certain events in the respective phases.

The requirements were:

- the negation of conditions 2a, 2b above.
- 'Data transmission should occur within 3 ticks'. (A special 'tick' action introduced).

  (We can also phrase this as: 'we can have at most three occurrences of station 1 sending an 'ENQ'.)

6.7.3 Modifications, Further Analysis and Results

deadlock/livelock for specs that are obs equ etc...

Simulating the Baud rate hunting procedure in the light of these requirements revealed that when station 1 changed its baud rate, it was possible to reach a state in which a 'send' could not be followed by a 'receive'. The following modifications were then tried to fulfil the requirements:

1. Baud rate hunting procedure amended to allow for receives by Station 1 at any time after the initial h1!send(ENQ).
2. The procedure TestX_ON for Station 2 amended to allow a second chance for the receipt of Station 1's X_ON before returning to the start.

Despite the amendments, the problem of phase mismatch was not solved. The resulting intensional specification (given in Appendix D), which implements the first modification only, admits the following sequence that leads to deadlock:

```
baud!9600;
h1!send(ENQ);
h2!receive(ENQ);
h2!send(DLE);
tick; baud!9600;

(* a second elapses, so station 1 sends another 'ENQ' *)

h1!send(ENQ);
h2!receive(ENQ);

(* therefore, character received after ENQ is ENQ, not X_ON *)
(* so station 2 goes back to await ENQ, but according to    *)
```

```
(* the protocol it must first send an X_ON ...              *)

h1!receive(DLE);
h2!send(X_ON);
h1!send(X_ON);
h1!receive(DLE);
h2!receive(X_ON)

(* so for station 1, the character received after 'DLE' is  *)
(* X_ON, therefore proceed with the rest of link connection *)
```

Phase mismatch has occurred when Station 1 has sent a superfluous 'ENQ' which is picked up by station 2 when it expects an X_ON (which may well be coming immediately after).

However, this still left instances of livelock, occurring especially when Station 1 enters its third and final phase in link connection, whilst Station 2 lags behind in an earlier phase. In certain circumstances, we would expect livelock: for instance, if we had allowed an infinite buffer; but here we used a duplex buffer which had capacity of just 1 in each direction. Increasing the capacity of the buffer did not help, generally only serving to allow the channel to be filled with redundant PDU's before reaching the same kind of deadlock.

At this stage, we considered that we had now implemented the protocol as best we could in LOTOS. We then had to consider the suitability of LOTOS. Here we felt that LOTOS was not the cause of the problems: sometimes a criticism is made that the interleaving semantics of LOTOS can give rise to counterintuitive event sequences, but these do not appear here owing to the synchronisation of both stations with the duplex channel. Also, although the current LOTOS standard has no explicit mechanism for handling time, we feel our simple denotation of the passing of time – through the use of a special 'tick' action – was adequate. This action was only needed to represent a simple mechanism that prompts the Baud rate hunting process to enter the next stage. There are timed extensions of LOTOS (see e.g. [BL92]) which may be valuable if we are to examine more timing aspects. However, we feel that in this case, the standard (untimed) LOTOS sufficed.

This leaves us having to consider fault inclusion and its associated risk. Looking at the trace leading to deadlock indicates that this eventuality could be unlikely for it requires that it takes about a second or longer for Station 1 to receive a response to its 'ENQ' given that Station 2 receives it fine and sends its response immediately. However, a more realistic

assessment of the probabilities and the potential severity can only be determined by those with suitable experience.

In summary, as the language seemed to capture the behaviour required, we have concluded that the protocol could be better defined. We followed the protocol in stipulating that when station 2 receives an ENQ, it really does send DLE followed by X_ON even though it may not have received an X_ON from station 1. We regard this as contributing to the problem in mutual co-ordination and believe that it is a fault in the protocol.

Having reached the conclusion that the protocol contains a fault, then we suggest a method of control for this part of the protocol. To respond to the problem of phase mismatch, this method of control imposes an added constraint on when the protocol can return back to the beginning to start all over again.

It is implemented in Station 2 as an extension to the process `S2_TestX_ON`. The extension traps repeat occurrences of Station 1's sending of the 'ENQ' that may be caused through lags in communication and puts Station 2 into a new process, `S2_AwaitX_ON`, where it still waits for the 'X_ON' if it receives further ENQ's; otherwise it duly returns to the beginning to start all over again.

This is modelled as:

```
process S2_TestX_ON [h2,l2]: exit :=

  choice y2:PDU []
      h2!receive(y2); h2!send(X_ON);
           (
            [y2 eq X_ON] -> S2_SendLCPack[h2,l2]
           []
            [y2 eq ENQ] -> S2_AwaitX_ON[h2,l2]
           []
            [(y2 ne X_ON) and (y2 ne ENQ)] -> S2_Connect[h2,l2]
           )
      where

   process S2_AwaitX_ON[h2,l2](yy:PDU) : exit :=

     choice y3: PDU []
       h2!receive(y3);
           (
```

```
            [y3 eq X_ON] -> S2_SendLCPack[h2,l2]
           []
            [y3 eq ENQ] -> S2_AwaitX_ON[h2,l2]
           []
            [(y3 ne X_ON) and (y3 ne ENQ)] -> S2_Connect[h2,l2]
            )
   endproc (* S2_AwaitX_ON *)

endproc (* S2_TestX_ON *)
```

6.8 Experiences in development

In this section we present more general comments about experiences in the development.

6.8.1 Building of the fault trees

We found that the task of building the fault trees for Flexport encouraged us to address faults in a more systematic and focused manner. One they were recorded in the fault tree, then the safety analysis was well anchored. After a while, the nature of interplay became evident between the building of the fault tree and the analysis of the model (particularly its simulation), with one activity leading onto the other.

Although we did not make explicit the semantics for the fault trees used in the FTA methodology, we found effective the use of temporal logic. Note that we derived only one gate requirement, which was actually superfluous to the combination of the event requirements. Also, we did not consider the analysis of the formalised tree itself. Other examples are required to explore these areas.

6.8.2 Refinement and Validation

Validating the specifications during the refinement of the two stations proved okay at first but then became problematic. For the properties, the specifications were simple enough to allow instead a verification approach where all the desired properties are encapsulated in a specially defined LOTOS specification (`MSCBase`). However, subsequent refinements suffered from state explosion due largely to the full duplex transmission mode:

running SMILE for 24 hours (on a SUN 670MP with few other processes running) was only able to expand the simulation tree to a depth of 30 (amounting to some 10,000 transitions). Hence, even the use of temporal logic would prove less fruitful in view of this lack of completeness.

The initial versions of station 1 and 2 do not cater for either station receiving unexpected data - i.e. faults are not included. These should be considered for further refinements. The incorporation of this extra behaviour means that extra traces must be possible, so the reduction preorder will fail since trace inclusion will not hold. In this instance, the extension relation may be more suitable.

6.8.3 On the use of Configuration Management

CM proved generally supportive, encouraging a disciplined approach to the project. It was particularly useful in analysing where faults were located since the use of versions and component structure allowed fault identification to be finely tuned. However, we were only able to complete half of the intended baselines. The versions that were actually built were fairly close to those planned, though in the event some baselines required sub baselines due to limitations with the tools which could only have been anticipated with some prior knowledge or experience of them.

6.8.4 On the Use of Tools

1. From the outset it was the aim to make use of various toolsets for the specification and analysis of the protocol, especially LITE, CADP and CWB. It appears that none of these were explicitly designed to be integrated with each other. The initial toolset used was LITE to check syntax and semantics (via TOPO) and then perform simulation (SMILE) of the initial intensional specification. However, on introducing the extensional specification and/or refinements, one of CADP or CWB would have to be invoked to check consistency.

2. For using CADP, the first modifications required were for DATA types: extra comments of certain types to facilitate compilation under Cæsar.adt. Since the program was written in 'C', a technical drawback was revealed: the failure to support overloading of data types. Thus the *send/receive* equations were not supported. Fortunately, the design of the specifications means that the equations need only be defined on the parameterised bit of these, i.e. the PDU's and Packets. The equations for sends and receives were simply dropped.

Having made the above changes, a constraint of Cæsar was evident. The first structural change of the intensional specification concerned the replacement of sequences of >> (*enables* operator) since Cæsar does not allow "recursive instantiation of processes to the left of >>). The error messages enabled the cause of the problem to be isolated to the station 1 and station 2 components. This was resolved immediately using a CPT that preserved observation equivalence.

3. CM under `rcs` forced closer integration of the toolsets: in order to show that versions were derived from other versions, data types had to be isolated as a component since LITE and CADP have different interpretations for loading libraries, e.g. given

```
library
  foo
endlib
```

LITE3.0 looks for a *data type* `foo` in one of the library files with '.lot' extension, or some other library file specified by the user, whereas CADP looks for a *file* `foo.lib` which defines the data types, perhaps with nested calls to other libraries.

Both toolsets had available in a single file the data types defined for ISO8807. However, they won't work with Cæsar and Cæsar.adt since the definitions make use of non-formal sort and operation actualization. Hence, CADP comes with its own modified versions, extensions and additions - the collection has a particularly wide selection.

The LITE version works generally OK with the ISO8807, though the TOPO compiler recognises some problems, e.g. the NaturalNumbers definition gives:

```
Warning: operation 0:11 not declared correctly
```

Thus, for compatibility, one library was created with all datatypes defined and 'included' using the UNIX macro preprocessor `m4`. This was in effect a replacement.

4. `m4` has a number of reserved words which are interpreted as macros, including 'include', 'len'(!) If the specifications contain any of these, then this will cause problems.

5. (tests)

Note that the tests depend upon overloading of data types, so we cannot use Cæsar (and so need not cater especially for its C compiler). However, we did make use of the library that comes with CADP, especially for the definition of Type 'Natural' (in

`flexdata.lotdat` v1.2 and above), though the label `Nat` was replaced by `Natl` since the LITE toolset reads in data type definitions from a specified library (MOD-IS in our case) which leads to clashes.

6.9 Observations and Conclusions

We have shown that the framework with its various procedures is useful for the development of an industrial safety-critical case study, a medical communications protocol. It has shown that common software engineering 'production' methods are readily applicable to mathematical objects, and highlights naturally many of the issues relating to the application of formal methods. One of the most important aspects has been the demonstration that safety-related properties really can and do have a system basis. However, this work is in its infancy – the methodology needs to be tested out on a wide range of examples; and the theory for relating requirements and models needs enhancing in the context of refinement.

It took a lot of effort to set up the methodology, especially the CM, but this was helpful in delineating the various tasks. For instance, the use of version control facilitated comparisons between various items – specifications of both model behaviour and of properties. Further, this approach becomes more attractive in light of the fact that the framework was not devised until the specifications had been started in a less structured manner, and that for most of the duration of the project, the methodology was being refined and theory enhanced.

As expected, LOTOS proved generally suitable for formally specifying the protocol, and we were able to specify quite easily the intensional and extensional specifications, including some simple timing aspects in the baud rate hunting. The success of the procedure is largely dependent upon the proof techniques for verification and validation.

The first approach we employed is based on the theory of testing discussed in Chapter 5. This has the merit of having already established ISO guidelines for communication systems. We were able to take advantage of the ability of LOTOS to implement tests as a single composition of (LOTOS) processes. As such, this opened up the use of all LOTOS analysis techniques, including the use of simulation tools.

As our second approach, we translated the LOTOS specifications into Labelled Transition Systems and could then select between a number of relations for verification. We chose observation equivalence, which is stronger than any testing relation. The procedure is just a matter of issuing a few commands at a command line prompt which, compared with generating a test and then simulating a test composition, is easier to perform and reduces the

risk of human error. However, the trade-off is that the non-interactive nature of checking a bisimulation by a tool means that feedback is limited, sometimes to just a 'Yes'/'No' answer. To establish further information is tricky since it requires the comparative analysis of two separately evolving specifications.

For the early versions, we were able to carry out these tasks without much effort: for instance, the initial versions of these specifications were shown to be observationally equivalent. However, computational resources were insufficient when we refined the intensional specification to account for baud rate hunting. Also, for a given model which has a buffer of any size, it was observed that as the buffer capacity increases, many deadlocks would occur further down the simulation tree. In view of the burgeoning complexity, such deadlock became more difficult to spot, indicating the need for better techniques to handle complexity.

It was found that the task of following the **FTBuild** procedure facilitates and prompts analysis, particularly a more complete consideration of hazards. It is initially quite laborious, but worthwhile; we feel it will become more straightforward with practice and that it should then be relatively easy to provide automated support, if only in the form of a menu driven routine that prompts a complete following of **FTBuild**, thereby releasing the burden of recalling what is the next step.

Regarding the safety analysis, even for our simple examples, it is evident that determining where a model has failed a requirement is not always straightforward. In order to be able to develop a fault tree in a helpful manner, it becomes important to have a well-structured specification in order to isolate causes of problems. We did not use risk management owing to lack of time. However, in Appendix G it is shown how risk management could record some of the information that is generated.

Regarding the protocol itself, the document appeared to contain some very elementary mistakes, most of which should have been spotted in an informal examination – for instance, there is a reference in page 1-3 to Station1 sending 'DLE' again. We assume that this is a misprint and should read 'ENQ'. This and other suspected errors are summarised in Appendix F.

Another general aspect was that the document was minimally prescriptive, so open to great amount of interpretation. It became manifest that many specifications could have been built that are arguably valid with respect to the document, but which have greatly differing behaviour. Also of concern was some ambiguity: particularly confusing is the definition of **Link Connect** (page 1-2), where there is the placing in one table of the

independent behaviours of the two stations, compounded by the loose use of actions.

A more serious problem appears to exist in the design, leading to deadlock – as detailed in section 6.7.1, there is the potential for one of the stations to enter a more advanced phase of link connection than the other through a kind of mismatch in timing. This fault was not resolved, so any safety case using our specifications would have this as a weakness whose likelihood and severity would have to be estimated.

Throughout, we have made considerable use of a variety of tools, especially some of those in the LITE and CADP collections. It was found that when employed together they were invaluable in the analyses, though some manipulation of specifications was required to enable this.

Chapter 7

Conclusions

7.1 Summary of Contribution

We have presented in this thesis a new integrated approach to the development of formal models for safety-critical systems. The theme of integration has operated at several levels: at the discipline level, we have set formal methods within the context of systems engineering, supported by software management; at the methodology level, we have brought together in the procedure **FTBuild** the tasks of safety analysis and model refinement; and at the theoretical level, we have enhanced formal testing by describing how the Experimental System of Hennessy and de Nicola fits inside the wider Observation framework that was developed by the Lotosphere Consortium, and by producing a new canonical tester for the reduction preorder. Furthermore, this approach has been illustrated in the formal treatment of a safety-critical industrial case study of a medical communications protocol.

The work has consisted of:

1. A state-of-the-art review that has a broad perspective

   - An overview of engineering methods for safety-critical systems that focuses on user and industrial requirements. This has contained definitions of the main concepts in safety engineering, especially hazards, risks and safety integrity, followed by a discussion of safety in a software context, with the attention on complexity. There has emerged consequently the need for a suitable generic model that has safety as a central provision. Hence we have reviewed a commonly accepted safety lifecycle model and given some motivating examples in the medical field.
   - A description has been given of what are formal methods and how they provide the rigour necessary for high integrity systems. The process of formal refinement

has been expressed in a (new) generic model for software development. Formal analogues of some safety-related concepts in engineering have been identified in the formal notions of safety and liveness. These notions have been illustrated in a dual language approach of process algebra (LOTOS) and temporal logic, which are the main formalisms used in the thesis. An examination of notions of consistency has been presented together with a review of methods of proof, covering theorem proving and model checking. Tool support for these activities has also been reviewed.

- An indication of the level of industrial uptake has been given in a discussion of work done in the medical field, with particular attention to medical device communications.

2. A new framework for developing systems through stepwise refinement underpinned by a formal perspective

   - A deeper appraisal has been presented in the light of the system view to establish the extent of the uptake of formal methods culminating in a survey of broader approaches that allow formal methods to be anchored in the production of safety-related software. This has led to an examination of approaches that bind formal approaches with traditional safety analyses, concluding with an affirmation of Methods Integration.
   - A framework has been proposed based on consideration of the safety lifecycle model. The framework includes a definition of concepts and safety-related principles for the system and an illustration of how hazards may be captured by formal denotation and subsequently reasoned about. The formalisation includes a notional definition of system safety that is in terms of completeness and consistency.
   - A management perspective has been adopted for the formal refinement that is inspired by the work of Bustard *et al* [BW94], but treated much more with formal methods in mind. In this thesis, the design items have been formal models and a generic graph model has been presented for Configuration Management, where formal relations are required between the items, satisfying notions of behavioural consistency as in work conducted by, e.g., Bowman *et al* [SBD95].

3. A new methodology that integrates the processes of deriving requirements from fault trees with the development of a formal model

- A review of the use of fault tree analysis for software has been conducted which highlights the role of formal methods in sharpening the quality of analysis. An example has been given that shows how the close scrutiny of semantics can expose the potential weaknesses of informal approaches, supplementing similar other observations by Górski [G94] and Bruns and Anderson [BA93].
- At the heart of the methodology a novel procedure **FTBuild** has been developed that integrates the usually disparate activities of system safety analysis, the refinement of formal models and their validation. The procedure achieves this by specifying a method based on the step-by-step construction of fault trees.
- A common formal semantics grounded in labelled transition systems has been defined for the fault tree analysis and system models with a procedure for deriving safety requirements based upon events and gates in fault trees. This has included a general means of formally evaluating events and gates, at any position in the tree.
- Work of Bruns and Anderson[BA93] that relates fault trees to models has been generalised to new conditions for conformance and consistency for models with respect not just to gate conditions, but to more general requirements. This has been supported by a discussion of issues in the formal analysis of trees; and of criteria for relations for property conformance between trees and models.

4. An enhancement of the theory of conformance testing, with special focus on robustness

   - An introduction to testing and the formal perspective has been presented, introducing the main concepts plus some examples to aid understanding of the theory that follows.
   - The Experimental System of Hennessy and De Nicola [Hen88] has been integrated thoroughly within the Observation Framework of Lotosphere [ABe+90]. The work has centred on the reduction preorder (the conjunction of the **conf** relation and trace preorder), complete with a proof that reduction is testable.
   - A new unified canonical tester has been derived for the reduction preorder for Basic LOTOS, based upon a newly defined acceptance function. In addition there has been given a method for implementing the tester as a specially defined LOTOS process for a subset of Full LOTOS (finite processes with predicates and guards resolved).

- A number of assorted Lemmas and examples have been included to illustrate the definitions, plus guidelines to facilitate the use of the reduction relation in practice. The perspective is further rounded off by a discussion suggesting other notions of conformance.

5. An industrial case study to illustrate the methodology and theory

   A safety-critical system of a medical communications protocol has been analysed using the framework and methods developed in the thesis.

   - Specifications in LOTOS have been prepared and refined of part of the Link Connection phase for the Flexport protocol[Spa89]. The refinement has been conducted within the proposed framework, using a modular approach for the specifications based upon the architecture of the system.
   - Two iterations of **FTBuild** were conducted, forming the basis for various kinds of analysis, supported by a suite of toolsets. For the initial model (with no features), internal consistency was shown between two views of the system – intensional and extensional – through the demonstration of observation equivalence. For the demonstration of safety-related properties, a canonical tester was generated from a specification based upon the MSC in the manner prescribed earlier in the thesis. It was shown that the initial intensional specification conformed robustly to this, so proper link connection was assured in a finite number of steps.
   - For the second iteration which included Baud Rate hunting, simulation of the model and the development of a fault tree have been employed in tandem for the safety analysis. These highlighted weaknesses in the document presentation and may have revealed a fault in the definition. Specifically, it was found that the design of the baud rate hunting process may lead to a mismatch between the two stations: after initial contact is made, one may miss an acknowledgement from the other and return to a state where it is still waiting for the first PDU, whilst the other has proceeded to a later stage in link connection.

7.2 Results and Assessment of Contribution

This thesis advocates the use of formal methods for safety-critical systems. Yet, the most valuable distinctiveness of the material here, especially the case study, lies in the emphasis on methodological continuity from requirements through to system modelling.

This has enabled the rigour of formal methods to breathe through the process in a supportive manner, thereby enhancing their role.

For the development, we have combined fault tree analysis, software management and formal methods in such a way that they encourage greater discipline and provide coherent analysis. These points are discussed in more detail below.

7.2.1 The safety-oriented framework

When we decided to tackle systematically the problem of validating a communications protocol, we initially dwelt on traditional concerns of formal methods, but it became clear that demonstrating the safety of this or any other high integrity system really needs an integrated perspective which allows one to translate safety-related engineering concepts to a formal setting and keep them in focus throughout the software development.

So we have adopted such a perspective by producing a safety-oriented framework which drives both the employment and further investigation of the mathematical theory. For instance, the consideration of system hazards in section 3.5.2.1 led naturally to a formulation of completeness and safety; this approach also was instrumental in the work on property conformance in Chapter 4. Thus formal methods are well grounded in the needs of real systems, rather than floating in isolation, as often appears. Basing the development around a safety lifecycle model has meant that user requirements are kept to the fore, so properties that are shown are relevant.

Formal methods have not gained much favour in most companies, which are not usually prepared to suffer much disruption in their procedures to accommodate what they generally admit is a technology that has high potential. The emergence of more national and international standards will provide some useful impetus, but it would be much better if companies volunteer without coercion. An important factor in this regard is education and training, where it is worth noting that large companies with big research departments can be expected to be aware of and employ state of the art techniques and be familiar with all the relevant standards; problems are far more likely with small companies making relatively small (or targeted) contributions. They rely more on external contacts such as academic departments.

Thus we have used Methods Integration, integrating FTA with formal refinement, itself a dual language framework, since we believe it to be a very important means of encouraging the use of formal methods by allowing current practices to continue whilst absorbing formal approaches. Certainly, this strategy is being investigated by a number of centres, so our approach of matches current thinking. Our contribution is one of the

very few that have managed to establish proper roles for formal approaches with regards to safety analysis.

Another important but much neglected area is software management to support formal development. We have addressed this by bringing to bear to the refinement of specifications aspects of CM. These have a simple formal conception as has been illustrated through a directed graph model showing how formal relations exist between design items within CM. This has supplied a useful indication of how formal support and formal construction may eventually be married.

Overall our work has shown, amongst other things, that safety-related properties really can and do have a system basis. Also, as an indication of its viability, many of the usual theoretical issues in formal methods are raised quite naturally.

7.2.2 The Procedures

We feel that **FTBuild** is a significant methodological innovation that unites the safety analysis technique of fault tree analysis and the development of formal models. Requirements are generated directly from the FTA and comparision of requirements with the model may be performed immediately. Although we provide formal semantics based on LTS, the procedure is independent of such semantics and as such is able to encompass all the activities of CSDM [BCG91], one of the few models that provide a formal basis for safety analysis.

Fault tree analysis imposes extra discipline and is difficult. It requires expertise in fault finding and really needs the practitioner to have detailed knowledge of the kind of system being built – a skill that is developed with experience, lying outside formal methods. On the other hand, we contend that with appropriate tool support (intimated below) safety analysts may successfully employ formal methods.

It should be straightforward to incorporate the procedure **FTBuild** within a company's already existing safety analysis procedures. This is due to the procedure allowing formalisation to be invoked at any stage during the development of both the fault tree and model, which may be regarded as concurrent activities subject to any amount of mutual constraint. The process of formalisation is flexible on a number of accounts: for any event or gate any number of requirements may be generated, the nature of the requirements not necessarily in terms of the events in the tree. Further, in the field of relating requrements to models, the generalisation of the relations of consistency expressed in [BA93] should prove a worthwhile extension that makes for more realistic use – evidenced by our case study (there

is no case study given in that paper).

Although the procedure is independent of particular semantics, it is worth noting that the choice of LTS semantics has allowed for direct validation of requirements with respect to models, in contrast to some Integrated methodologies that require transformations before consistency may be checked.

7.2.3 The Main Theoretical Contribution

This thesis has properly motivated a number of issues regarding theory, especially the need to handle complexity. A vital requirement that has been highlighted here – both in the notion of system safety and in terms of computational concerns – is the demonstration that formal methods can satisfy completeness as well as be able to demonstrate correctness. Completeness is essential for safety, hence our main theoretical contribution, the work on conformance testing, has emphasised robustness.

Robustness is the key requirement for safety-critical systems. Most work on conformance testing has tended to omit this consideration due to computational requirements. However, for safety-critical systems the omission of certain behaviour can be perilous, so we have concentrated on this issue. The LOTOSphere Consortium produced a vast output in the early 90's regarding testing, but relatively little has been published beyond the reports such as [ABe$^+$90]. This may be an indication of its poor uptake, which could be due to a number of reasons, one significant one being that the theory is subtle and hence less accessible. This is probably due to the requirement that in this framework specifications cannot be analysed directly, but only through observation.

Problems can persist if the theoretical foundations are often assumed and not always clear to developers. This was the case for the reduction preorder that has been established as a testing relation with respect to the Experimental System of Hennessy and De Nicola, but in the literature we have only been able to find a brief paragraph alluding to this link [BAL$^+$89]. Our thorough treatment has filled in the gaps and should clarify understanding of this fundamental relation.

The next issue that has been treated is the implementation of the tester for reduction. Again, this had only been alluded to due to the major obstacle of state explosion. Nevertheless, as we have argued previously, there are finite state systems where this is not the case. Thus the new canonical tester we have provided for reduction is beneficial and its design leads to useful diagnostics in the case of failure. Finally, the applicability of the tester has been demonstrated for a special class of processes through the implementation of the tester in a subset of Full LOTOS, allowing comprehensive analysis through simulation.

We have also provided examples and some guidelines to clarify the use of the reduction preorder since it is not immediately clear what conforming specifications look like. A few of the examples show that the relation has some undesirable aspects, which has led us to consider that some alternatives may be useful.

7.2.4 Findings from the Case Study

A novel feature of this case study was the emphasis on the safety-oriented nature of the design, built up from consideration of user requirements, and employing methods such as parts of software management which included RCS version control for all the specification components. The two iterations of **FTBuild** procedure showed that safety analysis became much more focused, enabling us to make observations within contexts that are perhaps more industrially realistic. These observations are reported in chapter 6;we make just a couple of points here – about tools and about fault tree analysis.

Having decided to use more than one toolset, the use of version control forced us to determine the extent of their compatibility. We found that in order to make use of verification and validation facilities for Full LOTOS, the CADP toolset was indispensible since it was the only one that possessed a well developed component Cæsar (early versions going back to the late 80s), that is able to accept as input a very useful subset of Full LOTOS and generate a transition graph ready for input into other tools. Moreover this toolset is still being developed and extended. It is well supported: on encountering bugs in one of the tools (`bcg`), the response was swift and several hours were spent in working at the problem.

However, this particular problem – essentially the inability of the file converter `bcg_io` to parse certain control characters discovered in text files – was not really resolved, and seemed to hinge upon some fickle problem in the setup in the user account on the UNIX system. Even though a simple alternative procedure of running the program in a temporary directory worked OK, this showed some fragility in such tools.

Overall, the tools that have been developed are very useful and we were able to perform analyses which would not have been possible by pen and paper. However, they still have some way to go before they offer both a reliable and complete enough set of facilities – for instance, it would be useful if more relations could be catered for. The greatest omission appears to be tool support for safety case development that includes formal items.

As a prerequisite, there needs to be better integration of what is currently available: at present there are some differences in the input accepted, though minor, which make it

obvious that little effort has been made in this direction. These may be largely dependent upon greater co-operation between the tool producers: no single tool can do everything, but if it various functions can be partitioned, then more completeness is possible and when the toolmakers have established their respective bounds, then reliability should improve.

We have shown that the procedure **FTBuild** is effective by illustrating it for aspects of the Flexport protocol, though the elaboration of fault trees is not easy. Some of the difficulty lies in the fact that they can be developed in a variety of ways, depending upon the viewpoint chosen in the search for causes: for instance, one can choose between temporal and structural event causes. In this case, the generalisation at a gate can be based on temporal inconsistencies, or faults in the physical or logical structure, which are particularly pertinent to communication protocols. We found that some of these differences are well highlighted by considering guidewords, adding weight to the contention that a HAZOP-style approach is advisable in FTA. Hence, to reflect the various ways of developing a tree, it seems preferable to talk about a family of fault trees for a given fault, derive requirements for each and validate the model with respect to all of these.

7.3 Scope for Future Work

The formal development of Flexport within the framework is only in its initial stages and has concentrated on bringing together common methods and tools. The framework is thus an early prototype that has achieved some validity in the case study. The case study should be continued so that some of the many issues raised, particularly in Chapter 3, may be addressed more fully. For instance, the issues surrounding change and formal methods should become clearer if the case study is increased in scale, which would probably require a team of developers rather than an individual. Then greater experiences could be gained into the parts played by the processes described in sections 3.6 and 3.6.1.

Similarly, the work described in **FTBuild** is also a prototype in its infancy – the procedure needs to be tested out on a wide range of examples; and the theory for relating requirements and models needs enhancing. It would certainly be useful to extend the Flexport specification accordingly: the continuation of the refinement, perhaps prompted by the development of other trees, would enable a much better assessment of how well the methodology works in practice.

In increasing the complexity of the system, proving the same underlying properties will likely require more effort, with the inclusion of more sophisticated mathematical techniques such as the invocation of more results on processes and also the greater use of

temporal logic. For instance, we may no longer expect to know *a priori* how a link may be established in terms of sequences of events – hence, we would seek to show equation (6.2) rather than (6.3). The application of the testing methodology would likely change to validating partial behaviour of the specification – perhaps showing that after a given trace, a given specification conforms subsequently for n steps. In any case, a combination of approaches to verification and validation seems sensible.

We list below some of the many issues that may be examined by incorporating the remaining features of the Low Level Link Interface that were specified in the target baselines (i.e. timing elements, flow control and error checking). It would be valuable to address these with or without reference to the case study.

1. *Notions of Correctness* Bisimulation-based observation equivalence has been shown suitable as the cornerstone for internal consistency. Bisimulation is general enough to cover any level of abstraction. However, it is not generally suitable as a refinement relation where behaviour is either added or removed. We have looked at the case of removing (optional) behaviour, for which the reduction relation is more suitable. It would be useful to look at the *extension* relation, which is the counterpart to reduction in that it allows extra behaviour. How could robustness be treated then? It would also be useful to develop some laws for processes which satisfy the various notions of conformance, analagous to those that have been developed for equivalence and congruence. This would enable the simplification of problems.

2. *Action refinement* Taking the general design methodology of step-wise refinement requires a theory for allowing transformation from one level of abstraction to another. We can achieve this for process algebras by grouping the processes into modules and treating these as actions at a higher level. Or, conversely, we may transform a single event into some process, either by the use of syntactic substitution or the definition of a special refinement operator. This is the notion of action refinement which has been explored in depth [AH94], but remains hardly tested in practice. For LOTOS it has been argued that the given interleaving semantics for LOTOS makes for undue complexity [CS93].

   Action refinement may be implemented in the intensional specification of Flexport: the single action which denotes a conversion from a packet to a sequence of PDUs can be replaced by some process which provides more detail on the conversion. The procedures can can then be assessed in more general contexts.

3. *Safety properties* In our initial specification, we were able to demonstrate through

an argument based on a testing preorder that some very strong properties held. On refining the specification, this has no longer been the case and such validation may be better performed through partial testing together with checks of properties expressed in temporal logic. It is a fact that safety properties are preserved in specifications that are bisimulation equivalent. But what about weaker relations? This question ought to be examined also.

4. *Expressiveness of LOTOS as a Process Algebra* The LOTOS specification of Flexport used a number of structuring operators which allow for modularity and other aids to style. The refinement may be extended to investigate whether or not extra components such as flow control can be slotted in without having to reconfigure the specification. If LOTOS is found inadequate, it should be proposed how it may be improved.

5. *Expressiveness of Process Algebras* How prescriptive should an initial specification be? Is an initial specification in a process algebra too restrictive? Would specifications in some modal process logic [LT88] be better and if so, what notions of refinement should be used? Can such modal specifications be eventually transformed into standard process algebras? Some work has been done with respect to CCS that indicates that the use of modal logic may be necessary for modelling some kinds of uncertain faults where process algebras can 'overspecify' [Bru95].

6. *Expressiveness of Logics* How expressive are each of the temporal logics? The modal-mu calculus has been shown powerful enough to express a range of safety properties here and elsewhere [BA91, NC96]. During the refinement of the Flexport specification, more elaborate properties will be needed and perhaps in some other logic to compare.

   The introduction of an unreliable channel in the Flexport specification will require the modelling of uncertainty of actions. The main mechanism for modelling uncertainty in LOTOS is the use of non-determinism, particularly via the internal **i** action. It may be established how well LOTOS handles such unreliability and worth considering alternatives, possibly leading to the construction of some 'modal LOTOS' (see above).

7. *Time* Communications protocols typically have timers and allow for unreliable circumstances in transmission. For Flexport, we have already modelled a simple timing aspect in the 'Baud Rate Hunting' process, for which we believe the model was adequate. The inclusion of more timing aspects would enable a better determination of whether or not (unextended) process calculi are lacking in their ability to model these, and if so why. It is widely reported that issues such as timeliness need extensions,

and a number of extensions have been put forward. Indeed, an international working group has included two approaches to time in an extension of LOTOS [ISO95]. The refinement of Flexport would supply more evidence for the validity or otherwise of such contentions.

8. *Relating requirements to models* The conformance relations defined in Chapter 4 may be refined as and when further experience is gained in applying **FTBuild** to case studies. On a different note, it would be useful to compare alternative formal theories that related fault trees to system models. Work undertaken in CSDM involving Petri Nets could be fruitfully integrated with work into system modelling using Statecharts [Nowicki].

9. *Enhancement of the testing theory* It should be fairly straightforward to apply the tester to more specialised contexts. One may introduce conformance *modulo* a fixed number of transitions. The consideration of alternative relations to reduction especially through reasoning about small examples (like our vending machine) should lead to more insight into what is required by developers from refinement relations in process calculi.

7.3.1 Towards a fully automated tool for formalising safety analysis

The enhancement of the methods and theory raised as issues above should eventually have automated support in the form of a toolset that integrates safety analysis, configuration management and formal methods. We provide below a taste of what to aim for.

Ideally, the safety expert should be able to formulate the requirements for the system builder without having to know much about temporal logic. So one requirement for the toolset would be to develop a formal language which is easy to interpret. In this respect, one may develop a kind of 'front end' to logical formula, in terms of a 'Natural-looking Language' for Safety Analysis, which we may call SAFELINGO.

Consider FTA, for example. If one looks at the events written informally in a fault tree, it becomes evident that the phrases often make use of just simple constructs with large amounts of repetition, particularly of some verbs and conjunctions ('when', 'until' etc). Most such constructs have been formalised in some language of logic. So it is conceivable that SAFELINGO may be built up by being an essentially some logic TL, say, but with syntactic sugaring that has words recognisable in natural language and a sentence structure with rules for syntax and semantics. This should be such that there is a well-defined

mapping for each valid sentence in SAFELINGO to a corresponding sentence in TL.

The safety analyst is then able to express events in SAFELINGO which can then be formally analysed. Ideally, he or she would acquire these from some dictionary, perhaps standardised, of informal safety-related requirements and their formal counterparts in a selection of formalisms. If the analyst finds SAFELINGO not expressive enough for some properties, then he or she can try to formulate directly the desired event/property in TL.

There exist already a number of graphical tools which support the construction of fault trees – the Safety Arguments Manager (SAM) is an example. Any such tool can now be augmented by the incorporation of SAFELINGO and TL. If also we have a model M such that TL and M have the same underlying semantics, then we may have an integrated system for conducting safety analysis and analysis of models.

The construction of fault trees and subsequent safety analysis for the model may be then be conducted within a procedure as follows.

1. An event E is created by defining an event box type for the output plus box types for each of the inputs.

2. E may now be written as a sentence in SAFELINGO. On-line help may consist of a selection of template sentences to choose from.

3. Some analysis of the tree *per se* may follow event splitting: e.g., Chap VII of FTHandbook [VGRH81]:

   > "tank rupture due to internal over-pressure caused by continuous pump operation for $t > 60$ seconds"

   Splitting is triggered by words 'due' and 'caused by', thereby giving rise to a branch of three events. (analyst will be given option of how much to split).

   ...

4. The next stage is to generate requirements for M. (internal step): A translation from SAFELINGO to TL is performed

5. A gate condition may be selected from a list of choices.

6. A model M is selected according to specification and version.

7. . ... On selecting an option 'Validate Model M for Fault Tree', one is requested to select validation criteria such as conformance and consistency relations (defaults

available) and then hit 'CR'. For some relations, this may involve selecting (with a mouse) a selection of those events and gates defined so far. Events and gates are then translated to TL to be checked for the model according to the specified criteria using some model checker (perhaps an external program). Feedback is given to indicate whether or not the model is valid, and if not some diagnosis as to why not.

... etc.

7.3.2 Other avenues

The majority of work for process calculi has emphasised the behavioural analysis. There has not been a great deal of consideration of data and, in particuler, relatively little formal examination of refinement which explicitly pays attention to Abstract Data Types. Yet data has been at the centre of the recent software engineering paradigms, such as the various flavours of object-oriented approaches. Some consideration has been given to LOTOS since it has a very useful subcomponent of ADTs, based on ACT ONE. There has been some work in a formal object-oriented design framework in LOTOS [Gib93] and issues of translation from ASN.1 to ACT ONE [Tho93b], but very little work on real-life examples.

Motivation for further research in the area comes from many applications, including the MIB, where an object-oriented language for virtual medical objects, has been specified in ASN.1, fitting in the top layer [SW90]. Thus, it would be useful to analyse LOTOS's ability to handle the modelling of data types and, especially object-oriented concepts, basing the analysis on attempting LOTOS specifications of the upper layer of the MIB. This would complement work already carried out in [CN92, NC96].

Another area is Software Metrics, which are a useful means for determining assessment criteria and encouraging/promoting software quality. Although we have not discussed them explicitly, we have actually defined at each stage in the lifecycle relations (valuations) that lend themselves to metrics, so it would be interesting to see whether a system could be developed to make this link worthwhile.

Here we have treated safety-related requirements. Other requirements such as mission and performance requirements could similarly be formulated and tested for some notion of 'conformance', thereby leading to a potentially complete and integrated requirements driven methodology for validating formal models.