ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Antony Bryant and Lesley Semmens (Eds)

Methods Integration

Proceedings of the Methods Integration Workshop, Leeds, 25-26 March 1996

Paper:

The Application of Structured Analysis/Formal Design Method to a Case Study from the Nuclear Industry

Alexandria Walker, Martyn Spink and Paul Vlissidis

Published in collaboration with the
British Computer Society

Application of Structured Analysis/Formal Design Method to a Case Study from the Nuclear Industry

Alexandria Walker

Martyn Spink

PEVE_π Unit
Department of Computer Science
The University of Manchester
Oxford Road, Manchester, M13 9PL.
email: {martyn, alexw}@cs.man.ac.uk

Paul Vlissidis
NCC, Oxford Road,
Manchester
email : Paul.Vlissidis@ncc.co.uk
formerly with BNFL Engineering Ltd.
Salford Quays

Abstract

The Yourdon Structured Method and the Vienna Development Method have been used together, in various ways, by several R & D groups in the industry. BNFL is tracking, and building on, the successes of these groups as part of a programme of work with the University of Manchester to produce an integrated software development method suitable for wide-spread use within BNFL. This paper looks at the method proposed in this research and the results of application to a realistic case study. The conclusions from this work are recorded with recommendations for future direction.

1 Introduction

The project, carried out as part of work by the PEVE_{IT} Unit was started in 1993 as a joint research investigation between the University of Manchester and British Nuclear Fuels Limited (BNFL). The brief was to produce a software development method suitable for safety-critical related projects within BNFL. This was achieved by, first producing a list of criteria for the method, then carrying out a literature survey of research already reported in the area and discussing the varying approaches and the suitability to the agreed criteria. The survey included work by Plat et. al [10-11], some papers presented at the Methods Integration Conference 1991, [7,12] and other work cited in the references for this paper. Through a series of workshops, using a system that had been specified the previous year using VDM [3], an outline method evolved. Following presentation of a report of this work [4], a further project for the 1994/95 year applied the proposed method to a selected case study. The case study was intended to judge the practical applicability of the method by considering its application to development of a realistic scale, and if it could be 'sold' as usable to the engineers working on BNFL projects. The work on this case study [14] forms the basis of the method developed in this paper and a discussion of the application of the original method is the topic of this paper.

The paper examines the criteria applied to and the structure of the BNFL Structured and Formal Development Method (BSFDM). This is followed by an outline of the chosen case study with examples of some of the products of applying BSFDM to that case study. Finally we explore the results of that application and raise issues and pointers to further work for developing the method.

2 The Criteria

BNFL already had a quality standard document [Vlis93] that outlined a software development method for use by systems personnel and this was to be used as a basis, around which the integrated method would be built. The method follows the Yourdon Structured Analysis and Design Methodology with Ward/Mellor real-time extensions and uses the techniques and documentation illustrated on the lefthand column of Figure 1. The fact that all system development projects, by both internal and external developers, are expected to conform to this quality standard influenced the resulting BSFDM criteria.

The criteria applied to the development of the new method [4] were :-

1. A single technique to be used at each development phase.
To ease the difficulty of introducing and enthusing engineers to adopt a new design method, which in being more rigorous would also mean more time in design (not a favourite occupation of engineers), it was important that the users did not feel they were repeating a task using two different approaches.
2. No changes to be made to (the semantics of) either Yourdon or VDM.
This is to ensure that existing and future tool support was not compromised. Another consideration was the high possibility of confusion amongst users who had already been trained in standard 'Yourdon' or 'VDM' techniques and to ensure ease of access for future training.
3. Minimal formalisation of Yourdon techniques.
Project groups can sometimes form an agreement to interpret the standard notation in a way that applies specifically to the project. This was considered an important 'freedom' within the bounds of the accepted standard and that maintenance of this would assist acceptance.
4. Non-critical systems must have essentially, a Yourdon-only route.
The whole reason for adopting more formal techniques as part of 'best engineering practice', was to enhance the development of safety-critical systems. From the BNFL point of view it would be much harder to justify the time/cost factor on non-safety-critical projects.

The method developed in the following sections conforms rigidly to these four criteria.

Figure 1 : Standard Yourdon Method compared to New Integrated Method

3 The BNFL Structured and Formal Development Method(BSFDM)

BSFDM is documented in [4] and Figure 1 taken from that report shows the stepwise stages of the method (in the righthand column) and the current recommended steps [15] in the lefthand column. The elements in the righthand column introduced with an asterisk are the newly introduced VDM based steps. A brief review of the techniques and explanation of decisions taken are covered the the following text.

3.1 Context Diagram

A Structured Method style Context Diagram is produced to establish the system boundaries, a (partial) event list is also developed to show data events, control events and temporal events. These products provide the analyst/designers with an initial view and focus for the system.

3.2 State Invariant Model

The State Invariant Model is used to capture static information about the system data. In a structured method this role would be performed by the construction of an Entity Relationship Diagram (or equivalent data model). Comparison between the two models was made by the group to discover if any information derived by using either technique was missing or different from constructing the alternative option. The final decision to use the State Invariant Model at this early stage was based on several factors :-

1. VDM-SL type definitions include more information about internal representations of data than an Entity-Relationship model, the latter being primarily a graphical representation that needs to be supported by a data dictionary, i.e. VDM-SL is a more expressive language.
2. The ability to record a state invariant means that safety properties can be elicited and recorded on the model at this early stage.
3. The VDM-SL data model is in a form suitable for use in the later design stages. In particular the recording of safety constraints in the invariant provides a formal basis for checking that later design stages respect those properties.

As a side effect this decision also provided an unconsidered benefit to the 'single technique' criteria, removing the need to translate from Entity Relationship Model and Data Dictionary to a State Invariant Model at a later stage in the development process.

3.3 The Data Flow Model

Using the information gathered in the first two steps, the Context Diagram is decomposed into a hierarchy of data flow diagrams, to be called a Data Flow Model (DFM). This DFM gives a diagrammatic representation of the flow of information through the system processes.

3.4 Selection of Approach

The project referred to the work of Plat et. al. [10] when deciding how to translate the DFM into a VDM specification. The following approach from Plat [10] most closely modelled the semantics of dataflows as BNFL Engineers perceived them.

Sequential ACcess (SAC) - here data flows are modelled as sequential channels that communicate between data transforms and the developer can make assumptions about the timing of the arrival of data input to transforms. In this approach the dataflow is modelled as a channel used for either input or output by the data transformer, in

other words operations can be written to place data as a sequence onto the channel and to remove data sequentially from the channel.

The SAC approach (taken from [10]) is :

1. Map each datastore onto a VDM state component - the State Invariant Model that was developed as the second step of the method should be checked and any store that has appeared during decomposition of the Data Flow Model should now be added to the State Invariant Model.
2. For each dataflow create a state component. The dataflow is modelled as a sequence of elements that is used as a queue (first in first out).
3. Create VDM operations that can access the dataflows to add and remove elements from the dataflow.
4. Create VDM operations representing the functionality of each Data Transformer and for each external process in the DFM.

The detail of these steps will be described in the relevant sections illustrating the application of the method to the case study.

3.5 Process Descriptions

This step applies to the BNFL non-formal route and designers observe the conventions from the quality document [15]. In the case of the formal route only the State Transition Diagrams used to represent control transforms (eg. transforms where all input and output flows are control flows, indicating that the transform acts in a coordination role for other data processing transforms.) need be produced. This allows the (re-)appearance of events and triggers.

4. The Waste Storage Case Study

The application of the proposed method to a case study followed the steps outlined precisely. Since the method under development was intended for practical use, it was felt that a significant case study based on an existing system would help determine the extent to which the constraints on the method had been met. The following extract from the case study is used initially :-

- a. To demonstrate how the method worked and the documentation produced.
- b. As reference material to emphasise the conclusions drawn from this application.

4.1 SCICS System Overview

The system concerns storage of encapsulated Intermediate Level Waste at sites used by BNFL. Waste is delivered from processing plants in drums on a four-pack stillage via an underground tunnel connecting the store and several plants. Each stillage has a unique identifying number etched on the side. For the case study purpose only stillages from two plants are considered, that of the **MAGNOX Encapsulation Plant (MEP)** and the **Waste Encapsulation Plant (WEP)**.

The store itself comprises a matrix of columns (14x14) into which stillages are lowered by a charge machine. The stillages are stacked up to 16 or 18 high depending on the stillage and the shock absorber type in the base of each column. A stillage's location within the store is determined according to a stacking regime (a set of rules eg. Short and Tall Stillages must not be mixed in the same column, and a current storage strategy, eg. try to find a suitable storage column on the West side of the store first.). Stillages can be imported via a receipt shaft leading to the transfer tunnel or through an export cave facility. In the case study only the processes using the receipt shaft were developed. The charge machine is controlled by a dedicated PLC system as is the bogie used to transfer stillage from the plants to the receipt shaft through the underground tunnel. Figure 2 gives a diagrammatic view of the store.

The operator interface is via three outstations located in the Charge Machine Control Room, the MEP Control Room and the Export Cave Plant area.

The Store Control and Inventory Computer System (SCICS) is responsible for supervising the control of the placement and/or removal of stillages in the store and for securely maintaining the map of the store. It runs continuously. In controlling placement, the system needs to be able to refer to a set of constraints that set out the safety criteria for storage and base the selection of the storage point on adherence to these rules.

The store map is a database that holds information about each column and the current content.

The map record information held is : location, temperature, shock-absorber-type, height, maximum-height, plug-status and stillages. The attributes location and stillages are composite types, the others are basic types. Maintenance of the map is classed as process-critical, being part of a 'Waste Tracking' procedure that should know where any item of waste is at any point in time.

Figure 2 : Diagrammatic Representation of the Store

4.2 The Context Diagram

Figure 3 shows the environment for the whole system and was constructed by an analyst based on User Requirements Specification and interviews with BNFL personnel. The textual description of the system used as reference for this Case Study was a Confidential Internal BNFL Document. All clarifications or assumptions required throughout the study were provided by BNFL personnel familiar with the system.

Figure 3: SCICS Context Diagram

4.3 State Invariant Model

The development of this State Invariant Model was an iterative process, with components and definitions of types being added as the analyst's understanding of the system developed. Initially information was taken from the textual description of the system. The extract shown in Figure 6 was then supplemented throughout the complete system model development and subsequent stages will be included as appropriate, the complete State Invariant Model can be found in Appendix 1. [N.B. The VDM-SL is expressed in "the interchange syntax (which uses the ASCII character set)" the ISO Draft International Standard]

4.4 The Data Flow Model (DFM)

This is a hierarchical model starting from the Top-Level Overview and breaking down each of the data transforms to its lowest level in Data Flow Diagram terms. The Data Flow Diagram in Figure 4 is the Top-Level view of the complete system and Figure 5 is a decomposition of the Placement Advice data transformer. Examples for the text in this paper are drawn from the dataflows and data transformers illustrated in Figure 5.

Figure 4: Level One Diagram - SCICS

Figure 5: Level Two Placement Advice - SCICS

```

module SCICS
definitions
types
    EPS1_Map          = seq of Column

    Column ::
        location      : Location
        temperature    : R
        shock-absorber-type : AbsorberType
        height         : N
        maximum-height : N
        plug-status     : PlugStatus
        stillages       : seq of Stillage

    inv mk-Column(loc,temp,sa-type,height,max-height,plug,stills) such that
        (sa-type=BARRED -> max-height=0) and
        (sa-type=TALL -> max-height=16) and
        (sa-type=SHORT -> max-height=18) and
        height <= max-height and
        len stills = height and
        (len stills<>0 -> stills(1).stillage-type.storage-type = sa-type)

    state EPS1Store of
        eps1_map          : EPS1_Map
    end

end SCICS

```

Figure 6 : Initial State Invariant Model

4.5 Selection of Approach

Based on the system understanding gained from the development of the DFM and observable in the low-level Data Flow Diagrams, select the best approach for further development, as discussed in Section 3.4, from :-

1. BSFDM
2. BPMDDM : this is the alternative approach where the developer can opt out of the 'VDM' aspects of the method and select a straightforward 'Yourdon' route for continuing development. As this paper is concentrating on integration and the Yourdon method is distinguished by its' maturity, this route will receive no further attention.

The SAC approach, as explained in Section 3.4, was selected as the most appropriate for use in the case study.

In the first instance every datastore exposed in the DFM must be mapped to a VDM component, if not already included. The second step creates a state component for each dataflow, modelled as a sequence of elements. These additions to the State are shown in Figure 7.

state EPS1Store **of**

eps1_map	: EPS1_Map
stillagedataandlocation	: StillageDataandLocation
storagestrategy	: StorageStrategy
estimated_stillage_type	: seq of EstimatedStillageType
placement_advice	: seq of PlacementAdvice
accepted_location	: seq of AcceptedLocation
rejected_location	: seq of RejectedLocation
start_rule_check	: seq of StartRuleCheck
restart_search	: seq of RestartSearch
checked_stillage_data	: seq of CheckedStillageData
location	: seq of Location

end

Figure 7 : The State Invariant Model updated to show DataFlows

In step three a set of operations are developed to represent the placing of data onto the dataflow channels and extracting data from a channel. Then operations are developed for each data transformer at the lowest level in the DFM hierarchy.

4.6 The "Put" and "Get" Operations

The template provided by Plat et. al. [11], shown in Figure 8, for the Put operation was adopted . The template in Figure 9 for the Get operation was developed for the project, to avoid using a VDM-SL syntactical extension that would have broken criteria 2. (See Section 2). The operations were developed from the Data Flow Diagram shown in Figure 5. The templates were developed by viewing the dataflow as an arbitrarily large (finite) channel (queue), the input being placed on the tail of the dataflow arrow and the output taken from the dataflow arrow head. The advantage of modelling the dataflow channel as two operations that put data on and take data off the channel, is that the data transformer operation is restricted in its access to the dataflow, being confined to making data available or requesting data input. The channel operations deal with placing the data at the end of the sequence or making the first element of the sequence available.

```

Put_df(i : DF)
-- This operation places data available onto the data flow channel
ext wr df : seq of DF
-- allow the reading and writing of the sequence DF to enable
-- the adding of data to the end of the current dataflow sequence
post df = df~ ^ [i]

```

Figure 8: Template for Put Operation [11]

```

Get_df () r: DF
-- This operation makes the data at the front of
-- the sequence available to the data transformer.
ext wr df : seq of DF
-- allow the reading and writing of the sequence DF
pre    df <> [ ]
-- if the precondition is not met then follow error route
post   df~ = [r] ^ df
-- take the data from the front of the dataflow sequence DF
errs   empty_DF : df = [ ] -> r = nil and df ~= df

```

Figure 9: Template for Get Operation

A point to be noted from the templates is that both Put and Get operations have write access to the dataflow sequence. This means that the invariant on the state has to be checked after each operation.

4.7 Example Put and Get Operations for Case Study

An example of a Put and Get operation for the case study is shown in Figure 10, the complete set of operations can be found in [14].

```

Put_EstimatedStillageType(e:EstimatedStillageType)
ext wr estimated_stillage_type: seq of EstimatedStillageType

post   estimated_stillage_type = estimated_stillage_type~ ^ [e] ;

Get_EstimatedStillageType () est : EstimatedStillageType
ext wr estimated_stillage_type: seq of EstimatedStillageType
pre    estimated_stillage_type <> [ ]
post   estimated_stillage_type ~ = [est] ^ estimated_stillage_type
errs   empty_DF :    estimated_stillage_type = [ ] ->
                      est= nil and
                      estimated_stillage_type~ = estimated_stillage_type;

```

Figure 10 : Example Put and Get Operations

4.8 Operations for Data Transformers

Operations for each data transformer are constructed. In the SAC approach the operations need no parameters either input or output, this is due to the modelling of the Put and Get operations. All input and output can be accessed directly as state elements.

Quantified expressions are used to create the input and output for the data transformer and the effect of the data transformer on the inputs is also described. Once again use is made of the 'errs' clause to overcome the problem of data being unavailable on a flow. The template developed during the project and used for these operations is shown in Figure 11.

```

DT ()
— All external reads and writes are put here for inputs outputs and datastores
ext wr      i1          : Input1
wr         i2          : Input2
wr         o1          : Output1
rd         datastore   : DataStore
— Check that any input required for the operation is present
— Check that any datastore data is present
pre i1 <> [] and i2 <> [] and datastore <> nil
post
— the predicate expressions  $P_{DT}$  describe the effect
— of the data transformer on the input values.
(post_Get_i1(i1~, i1, i1') and post_Get_i2(i2~, i2, i2') and
post_Put_o1(o1~, o1, o1') and  $P_{DT}(i1, i2, o1)$ )
errs
— If use of the operation is invalid ensure that nothing changes
invalid_operation :      i1=[] or i2=[] or datastore <> nil ->
                        i1~=i1 and i2~=i2 and o1~=o1

```

Figure 11 : Template for Data Transformer Operations

N.B. The operations differ in syntax to the template, this is due to the necessity of conforming with the tools used for checking the syntax not any fundamental change in the format.

An example Data Transformer operation is shown in Figure 12, the complete set of operations is shown in Appendix 1.

```

CheckStillageData ( )
— This operation checks that the information entered by the SCICS
— operator is consistent with the system expectations by using the
— function valid, then this data is used as output.

ext wr estimated_stillage_type : seq of EstimatedStillageType
      wr checked_stillage_data  : seq of CheckedStillageData

pre    estimated_stillage_type <> [ ]

post   let est : EstimatedStillageType in
      (post_Get_EstimatedStillageType
        (est, estimated_stillage_type ~, estimated_stillage_type)
        and valid(est)
      and
      post_Put_CheckedStillageData(mk_CheckedStillageData(est),
        checked_stillage_data ~,
        checked_stillage_data))

errs   Invalid_estimated_stillage_data :
      estimated_stillage_type = [ ] ->
        estimated_stillage_type~ = estimated_stillage_type and
        checked_stillage_data~ = checked_stillage_data

```

Figure 12: Example Data Transform Operation

5: Conclusions from the Case Study

The application to a case study of the BNFL method was carried out to confirm the applicability or otherwise of the proposed method. The method produced a mixed 'Yourdon/VDM' specification which would be usable to code the required system. In applying the method to a system of this size and complexity, several issues were raised. These are discussed in the remainder of this paper.

The system selected for the case study was chosen for two reasons. Firstly as being of a standard size in relation to the development projects carried out by BNFL personnel and secondly due to the safety critical aspects essential to the system. It was apparent at a very early stage that the system had to be reduced in size due to the time constraints on the project therefore the resulting formal specification was judged to be approximately half the size that the complete system would have required. It is important to note this point as some of the problems of usability of the method could be related to this issue.

5.1 State Invariant Model

Application of the method to the Case Study verified the belief that the use of a State Invariant Model would provide information not available through an Entity Relationship Diagram especially when related to safety-critical systems. The State Invariant Model gave the ability to define constraints on types and values through the use of the invariant. It was not clear, however, if the safety properties of the system should also be dealt with as constraints and this

caused some confusion to the developers. The method does not currently formally or informally force tracing how or where system requirements are to be dealt with, especially the safety property requirements.

The State Invariant Model is initially built as the second step in the development process. Another area of concern was that whilst the DFM was developed the State Invariant Model had to be revisited many times for major additions. All the subsequent dataflows and any constraints affecting these are added. This sometimes meant rewriting other constraints that are affected as a by-product. This could lead to inconsistencies and inaccuracy in maintaining the integrity of the State Invariant Model, especially as the result of all these iterations forces the state to become large and difficult to read.

5.2 : Specification Size

With a Put_df and Get_df operation for every dataflow in the DFM the number of operations could become unwieldy, for example in the Case Study there were approximately seventy five dataflows which translates into one hundred and fifty operations specifying putting data onto and getting data from data channels. However, it does act as a regular review stage on the current validity of the VDM plus providing a cross reference between the two.

This explosion of size in the specification seems to add weight to the common argument that Formal Specifications are unreadable to the 'non-expert'. The construction of these operations can also be a tedious chore for the developer, which carries danger of inaccuracies, especially if the development is done 'manually'.

For the purposes of the Case Study, where the DFM was produced without the benefit of sophisticated tool support, a 'C' program was developed that took a list of dataflow names and produced the Put and Get operations automatically in VDM-SL, using the template formats. Although this alleviated some of the tedium it did not solve the problem of specification size. The VDM was verified initially using SpecBox [16] and later in the project the IFAD VDM_SL Toolbox [17] was also applied.

Each Put and Get operation using the template is of identical construction, therefore generic forms can be defined using modules - we are working on ways to improve the formal specification presentation, including the use of tools.

One of the points raised in 3.6 was that a write is necessary in the Get operation and it is important to discharge the 'satisfiability obligation' in relation to the dataflow component, this may not be trivial. Invariant preservation is of paramount importance to the integrity of the specification. It is imperative that this issue is addressed and that it can be shown that, however the situation is resolved, the whole State Invariant is preserved at all times. Two immediate areas to be explored in relation to this problem could be :-

- 1: Finding a way of modelling the dataflow access that takes them out of the State.
- 2: Discharge a lemma for each flow so that an empty dataflow doesn't break the Invariant.

Another problem of a sequence is that although it is stated that the data transformer can only access the first element, the reality is of course that any element in the sequence could be accessed by using an index. Introducing the concept of a queue or stack to the VDM-SL armoury could provide a solution, and could be achieved by introducing a module representing dataflows.

5.3 The Data Transformer Operations

When carrying out the VDM specification of the Data Transformer operations several issues became apparent -

1. When decomposing the DFM following the standard techniques only two levels of decomposition from a top level transformation are expected, which often leaves the dataflow to transformation relationship as one-to-many or many-to-one. The data transformers were often difficult to model, due to the number of inputs or outputs to a transformation, and retain an abstract rather than concrete specification in VDM. This is a problem documented in other research [1,8] .
2. The DFM takes no account of the control flow of the system, neither does it provide information about timing. In a Structured Method the DFM would be supported by other documentation, such as state transition

diagrams and process specifications. In developing the VDM specifications of the data transformations, without this type of additional information and relating to the problem outlined in 1. it was extremely difficult to remain in the area of 'WHAT needs to be achieved' and very easy to stray into the 'HOW it is going to be achieved'.

3. it was also observed that a breaking up of the VDM specification would benefit the method, making use of the module facility to achieve two perceived requirements, the first to make the specification more readable, the second to enable choices in the level of specification needed for any one part of the system development, for example a very abstract VDM specification of the non safety-critical elements and a more detailed concrete specification of the safety-critical parts.

5.4 Tool Support

It was realised very early in the project that without tool support the development could become difficult. The tools that are needed include :-

1. A Design Tool for the Structured Diagrams that checks consistency through the hierarchy of diagrams and provides a reporting facility that can be tailored to produce information needed in later stages; for example list of data flow names.
2. A script to generate Put and Get operations that takes a list of dataflow names and produces templates for all the Put and Get operations.
3. A similar script to take the Data Transform names and produce templates.
4. An analyser for the VDM specification. The project used SpecBox_{TM} [16] a later version was run through the IFAD VDM_SL Toolbox [17].
5. To complete the set a proof support system, something like MURAL [18], would be a valuable aid to fulfilment of proof obligations and validation conjectures.

5.5 The Structured Method

The structured techniques incorporated into the method consisted of just the Data Flow Model and State Transition Diagrams. It was felt that some vital elements of information could have been provided by including other types of informal documentation .

The project was also constrained to the Structured Method to be used, in the continuation of this work no such constraint will exist, therefore it is felt that a discussion of what notation would best fit the needs should be carried out; for example would the Statecharts provide better and more relevant documentation of the safety critical system ?

Structured methods are also supposed to allow an eclectic use of techniques, therefore it may be more appropriate to examine techniques from different methods and bring them together for this purpose.

5.6 Reviews of Formal Specifications

During the course of the project the formal specification under went a number of different reviews. Because the use of VDM appears at different stages in BSFDM the review phases are different from those used during a fully formal development. However, a number of important lessons were learnt on this project.

The formal specification went through four different reviews:

1. At all stages the specification was reviewed for internal consistency using both manual effort and tool support. This review included ensuring that the various parts of the invariant were consistent and that operations were correctly specified.

2. The initial state and invariant model was reviewed to verify it against the requirements specification and the Context Diagram. This provided not only a review of the specification but a review of the requirements and the Context Diagram. At the end of this review all three 'specifications' were consistent.
3. The expanded state and invariant model (with the data flows) was intensively reviewed against the dataflow diagrams -- even though the VDM was automatically generated.
4. The interfaces to the transformation operations were reviewed against the final data flow diagrams.

From this project and from other projects we have been involved with we have identified three entirely different forms of review:-

- a. A review of the specification against the requirements. This is essentially a verification of the specification. This should involve the people involved in the development of the specification and the customer.
- b. A review to ensure that the formal specification and the associated structured diagrams are consistent. This should involve the development team only and should not be concerned with the requirements but with verifying the consistency. It is believed that with appropriate tool support the time spent on this review can be minimized.
- c. A review of the formal specification that should be carried out by formal methods experts and the development team. The purpose of this review is to ensure that the specification is 'correct', internally consistent and states what the development team believe it states. Again much of the work of this review can be achieved with appropriate tool support.

It is clear that although using a formal development method as part of the development process provides a number of benefits without appropriate tool support the review process is extremely time consuming.

5.7 Summary

The final conclusion from the project was that the method as applied certainly produced a specification that could be used to provide a sound basis for implementation. The weaknesses and problems encountered are not insurmountable and the exercise has, we feel, given us a sound basis to move forward.

5.7.1 The Criteria Revisited

1. Single Technique to be used at each development phase.
This was achieved as illustrated in Figure 1 and we would maintain this idea in any further development of the method. We are very positive that building on the case study experience this is possible.
2. No changes to be made to (the semantics of) either Yourdon or VDM.
Although there was a temptation to follow very closely the ideas found in Plat's work [10] this would have involved adding a syntactical θ extension. This was avoided by making use of the err clause.
We would take a more eclectic view of the techniques used in the structured method steps and extend exploration to other methods and the techniques offered. The 'no changes to semantics' should, however, apply for all the reasons given at the start of this paper.
3. Minimal Formalisation of Yourdon techniques.
4. Non-critical systems must have essentially a Yourdon-only route.

Both of these criteria were adhered to and caused no difficulty. In future development these will move into the background, however, through the use of VDM-SL modules we propose that non-critical elements of a system can be specified in an abstract way, whilst the more safety critical can be specified in more detail.

5.7.2 Future Work

We are going to continue our investigation of this method and in the immediate future take the following directions -

1. Look at the different structured methods especially the use of Statecharts, requirements tracing and process specification.
2. Examining the use of VDM modules to :-
 - i. Solve the size explosion re. dataflows.
 - ii. To provide generic Put and Get.
 - iii. To allow choice in the level of abstraction for different elements of a system.
3. Make use of existing tools and investigate the development of new tools to support the proposed method.

6: References

1. Bruno Albiso :Transformation of Data Flow Analysis Models to Object Oriented Design. ACM , November 1988, pp 335-353 (OOPSLA'88 Proceedings)
2. S. Aujla, T. Bryant and L. Semmens: Rigorous Review Technique - using formal notations within conventional development methods. BT/Leeds Metropolitan University Collaborative Project, 1993. (Working Paper)
3. Ian Cottam, Bill Neary, Paul Vlassidis. A Case Study Evaluation of the Vienna Development Method Specification Language : VDM-SL. BNFL Internal Report, April 1993
4. Ian Cottam, Alexandria Walker, Bill Neary, Paul Vlassidis. A Proposal for Integrating the Yourdon and VDM Software Development Methods. BNFL Internal Report, April 1994
5. John Dawes. The VDM-SL Reference Guide. Pitman Publishing, 1991, ISBN 0-273-03151-1
6. S. Goldsmith. Evaluation of YSM and VDM for time-critical systems. March 1990, Computing and Control Engineering Journal,
7. V. Hamilton. Experience of combining Yourdon and VDM. Methods Integration Conference, Leeds Polytechnic 1991
8. Peter Gorm Larsen, Nico Plat and Hans Toetenel. A Formal Semantics of Data Flow Diagrams. FACJ 1994 No6.
9. F. Meziane and S. Vadera. From English to Formal Specifications. University of Salford, Technical Report MCS-93-06, 1993
10. Nico Plat, Jan van Katwijk, Kees Pronk. A Case for Structured Analysis/Formal Design. Springer-Verlag, 1991. LNCS 551. pp. 81-105, VDM'91 Formal Software Development Methods,
11. Nico Plat. Experiments with Formal Methods in Software Engineering. PHd Thesis, Technische Universiteit Delft, September 1993
12. Lesley Semmens and P. Allen. Formalising Yourdon. Methods Integration Conference, Leeds Polytechnic 1991
13. Lesley Semmens, Robert. France and T. Docker. Integrated Structured Analysis and Formal Specification Techniques. The Computer Journal, vol 36, no. 6, 1992
14. Martyn Spink, Alexandria Walker, Paul Vlassidis. A Specification of the EPS1 Store using an Integrated Structured and Formal Method. BNFL Internal Report, February 1995
15. Paul Vlassidis. Design Method :- Yourdon Methodology: Reference No. CE&I/04/15. BNFL Internal Publication
16. R.Blomfield, P. Froome, B. Monahan. Specbox: A toolkit for BSI-VDM. Safety Net, 1989. pp. 4-7, Safety Net, No. 5.
17. R. Elstrom, P.G.Larsen, P.B.Lassen. The IFAD VDM_SL Toolbox: A Practical Approach to Formal Specifications. ACM, September 1994. pp. 77-80, ACM Sigplan Notices, Vol 29, No. 9
18. C.B.Jones, K.D.Jones, P.A.Lindsay, R.Moore. mural: A Formal Development Support System. Springer Verlag, 1991. ISBN 3-540-19651-X

Appendix 1

state EPS1_Store of

eps1_map	: EPS1_Map
stillagedataandlocation	: [StillageDataAndLocation]
storagestrategy	: StorageStrategy
passwords	: PasswordStore
infringement	: bool
trans_reports	: set of TransReport
infringe_reports	: set of InfrinReport
acceptedlocation	: seq of AcceptedLocation
chargemachineio	: seq of ChargeMachineIO
chargemachineoperationrequest	: seq of ChargeMachineOperationRequest
chargemachineoperations	: seq of ChargeMachineOperations
checkedinventorychanges	: seq of CheckedInventoryChanges
checkedstillagedata	: seq of CheckedStillageData
columnstatusmodification	: seq of ColumnStatusModification
confirmedplacementdata	: seq of ConfirmedPlacementData
databasemodification	: seq of DatabaseModification
enteredstillagedata	: seq of EnteredStillageData
estimatedstillagetype	: seq of EstimatedStillageType
estimatedstillagedata	: seq of EstimatedStillageData
invalidoperationrequest	: seq of InvalidOperationRequest
inventorychangedata	: seq of InventoryChangeData
inventorychangesmep	: seq of InventoryChangesMEP
inventorychangeswep	: seq of InventoryChangesWEP
locationdata	: seq of LocationData
moveoperationcomplete	: seq of MoveOperationComplete
mepexportedstillage	: seq of MEPExportedStillage
newplacementdata	: seq of NewPlacementData
operatorconfirmation	: seq of OperatorConfirmation
operatorinformation	: seq of OperatorInformation
operatorrequest	: seq of OperatorRequest
overidelocationadvice	: seq of OverrideLocationAdvice
password	: seq of Password
placementadvice	: seq of PlacementAdvice
placementdata	: seq of PlacementData
placementoperationcomplete	: seq of PlacementOperationComplete
qaoverride	: seq of QAOverride
rejectplacementdata	: seq of RejectPlacementData
rejectedlocation	: seq of RejectedLocation

requestlocation	: seq of RequestLocation
restartsearch	: seq of RestartSearch
startengineerlevel	: seq of StartEngineerLevel
startimportoperation	: seq of StartImportOperation
startmanagerlevel	: seq of StartManagerLevel
startmoveoperation	: seq of StartMoveOperation
startplugreplacementoperation	: seq of StartPlugReplacementOperation
startrelocationoperation	: seq of StartRelocationOperation
startrulecheck	: seq of StartRuleCheck
startscicslevel	: seq of StartSCICSLevel
startstorage	: seq of StartStorage
startsupervisorlevel	: seq of StartSupervisorLevel
starttransfer	: seq of StartTransfer
stillagedata	: seq of StillageData
stillagenuminvalid	: seq of StillageNumInvalid
stopengineerlevel	: seq of StopEngineerLevel
stopimportoperation	: seq of StopImportOperation
stopmanagerlevel	: seq of StopManagerLevel
stopplugreplacementoperation	: seq of StopPlugReplacementOperation
stoprelocationoperation	: seq of StopRelocationOperation
stopscicslevel	: seq of StopSCICSLevel
stopsupervisorlevel	: seq of StopSupervisorLevel
storagecomplete	: seq of StorageComplete
storedinventorychanges	: seq of StoredInventoryChanges
transfercomplete	: seq of TransferComplete
transfertunneloperationrequest	: seq of TransferTunnelOperationRequest
tunnelio	: seq of TunnelIO
tunneloperation	: seq of TunnelOperation
wepexportedstillage	: seq of WEPEXportedStillage

inv eps_store ==

— if infrin boolean is true there must be infringement reports

eps_store.infringement <=> (eps_store.infringe_reports = {})

and

— if infrin boolean is true, the function must return true

eps_store.infringement <=> infringed_rules(eps_store.eps1_map)

```

init store ==
  store = mk_EPS1_Store([
    ,nil
    ,{ |-> }
    ,{<OPERATOR> |-> "operator", <MANAGER> |-> "manager",
      <SUPERVISOR> |-> "supervisor", <ENGINEER> |-> "engineer"}
    ,false
    ,{}
    ,{}
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    ,[]
    , ... , [ ])
end

```