Integrated Structured Analysis and Formal Specification Techniques

L. T. SEMMENS[(1)]*, R. B. FRANCE[(2)] AND T. W. G. DOCKER[(3)]

[1] *Leeds Polytechnic, Faculty of Information and Engineering Systems, Leeds Polytechnic, The Grange, Beckett Park, Leeds LS6 3QS*

[2] *University of Maryland*

[3] *Cranfield IT Institute*

The last decade has seen a large increase in the use of 'structured' methods of software development. They grew out of their predecessors, which consisted of a toolkit of techniques with little method of how and when to apply them. Methods such as Yourdon Structured Analysis[1] and SSADM[2] provide means of managing the complexity of large systems. They provide techniques and associated procedures for the development of such systems. In parallel with this, and almost entirely independently, formal specification languages and methods have been developed. Both approaches have their strengths and weaknesses. Recently it has been recognised that benefits can be gained from integrating the two.

A number of researchers have reported progress towards the successful integration of formal and structured methods. This paper reports on a selection of this work. The aim in each case has been to develop specifications which are both structured and formal, and so combine the proven advantages of both approaches. We believe that such integrated methods remove some of the 'culture shock' associated with the introduction of mathematically formal languages, and make their use more acceptable to managers and engineers in software development organisations. There is some evidence for the feasibility and cost-effectiveness of this approach when used in an industrial setting; developers at Rolls Royce Associates[3] have reported the successful combined use of VDM and Yourdon.

Work has also started on the integration of formal notations and graphical techniques for Object-Oriented Analysis and Design, but this is all at a very early stage and little has been reported formally.

Received June 1992

1. STRENGTHS AND WEAKNESSES OF STRUCTURED AND FORMAL METHODS

Structured analysis methods use techniques such as Entity Relationship Modelling (ERM), Data Flow Diagrams (DFD) and State Transistion Diagrams (STD) to represent the static and dynamic properties of systems. These are usually underpinned by a Data Dictionary, which will contain information such as the attributes associated with a particular entity in the ERM and the attributes associated with data flows on the DFD. These methods provide not only techniques but a structured approach to the development process. They are good for analysing and structuring systems and are relatively easily understood by the customer. They also have the advantage of being well tried and understood and are used by the more conscientious developers of systems. However, some of the techniques lack formality. In particular, because of the variety of notations (diagrams and text) used it is not possible to reason about specifications.

Formal notations have the advantage that they can be reasoned about. They are concise and unambiguous. One major drawback is that they lack structure and thus make it difficult to manage the development of large systems. Most so-called formal 'methods' offer little more than a mathematically based notation and a modelling technique. The use of such notations and techniques must be situated within a software development method.

The advantages of the two approaches would seem to be complimentary. Structured methods assist in the management of size and complexity and provide a means to structure a specification. Formal notations provide a means by which developers can make precise, unambiguous statements about systems, and use those statements as a basis for reasoning about the system.

* To whom correspondence should be addressed.

2. APPROACHES TO INTEGRATION

There are several possible approaches to integrating structured methods and formal notations. The first is to use the two side by side and not attempt to show any formal link. Approaches of this kind include that used at Rolls Royce & Associates[3] and proposed by the SAZ project,[4] the first of which will be outlined below. The second is to formalise the link between the structured notations and the target formal notation. This is the approach proposed by Goldsmith[5] and taken by Tse,[6] France & Docker,[7] Semmens & Allen,[8] Larsen *et al.*,[10] Redmond-Pyle & Josephs[11] and Fencott *et al.*[12] all of which are outlined below. Rose has done work on JSD and Z which is to be reported on in his forthcoming thesis.[13]

Much of the work has focused on Yourdon Structured Analysis (and related methods such as deMarco[29] and SA),[20] perhaps because this is more often used on non-DP systems than other structured methods such as SSADM and its relatives. However, Redmond-Pyle & Josephs have done work on LBMS SE, which is related to SSADM, and there has been some (unpublished) work at BT[37] which has been applied mainly to DP systems.

There is also the question of which type of formal notation to use. The approaches we discuss fall into three categories: model-based specification, algebraic specifi-

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

cation and process algebra. We will discuss each separately.

3. MODEL-BASED SPECIFICATION TECHNIQUES

Research relating to the integration of structured analysis and model-based specification techniques began in the late eighties. The first proposals were made by Goldsmith,[5] who proposed the integration of Yourdon[1] and VDM,[14] and Bryant,[15] who proposed the integration of SSADM and Z.[16] Practical pilot studies were being undertaken at Rolls Royce & Associates using Yourdon and VDM. The earliest detailed work was reported at the 1990 Z User Group Workshop,[17] where Semmens & Allen[8] and Randell[18, 19] presented their work. These both dealt with the translation of structured notations into Z.

Integration takes two forms. There is the relatively loose approach taken by Rolls Royce, where Yourdon and VDM are used side by side, and the more formal approach of performing a direct translation of the structured notations into the target language, thus providing a formal semantics for the diagrams.

In the following sections the work at Rolls Royce, that of Larsen *et al.*, of Semmens & Allen and Redmond-Pyle & Josephs will be outlined.

3.1 Yourdon/VDM (Rolls Royce)

The approach taken at Rolls Royce & Associates[3] involves using a Yourdon-based analysis to capture the structure of the requirements. The structure of the specification is that of Yourdon, but the detailed process specifications and the data structures are described in VDM, with explanatory English text. Care is taken that implicit specification is used; this ensures that the specification states what is to be done rather than how.

This approach has utilised the power and usability of the analysis method to ensure that complete and consistent requirements are captured. The structured method makes it easier to capture the essential details of what is wanted rather than how to do it. The precision and rigour of the formal language contributes to this at the detailed level by imposing discipline on the analysis. The designers need to be able to read the formal specification, but the usability of the structured specification means that this is much easier to do.

The approach was found to be cost-effective even in non-critical projects. The slight increase in the estimates for producing the specification was recovered in reducing the contingency in the development costs from then on. In critical projects program proving was made easier by the close mapping of the structure of the specification and the code. Modularity also made the proving of parts of a system feasible.

They have found that structured methods make formal specifications much more approachable. They recognise that in requirements capture it is not practical to work from the top down; that structured methods allow analysts to iterate between top-down, bottom-up and middle-out, gradually improving the analysis until it is complete and can be presented in a top-down fashion; and that used in conjunction with structured methods, formal methods provide much-needed discipline and clarity, and can be introduced without major restaffing or retraining, and without increasing costs. Whether or not the final software is to be proved, they believe that the formal specification justifies itself in the quality and maintainability of the system.

3.2 SA/SD and VDM

Researchers at TU Delft propose a method SAVDM.[10] This method provides the graphical notations from SA/SD,[20] the methodological guidelines from the SA phase of SA/SD and the formal aspects of VDM. The combined method, illustrated in Fig. 1, comprises the following steps.

(1) Analyse the problem and develop a context diagram, representing the system boundaries.

(2) Decompose the context diagram by splitting the high-level data transformer into several lower-level data transformers. Each of these data transformers is subsequently decomposed until an acceptably complete hierarchy of DFDs has been produced.

(3) Provide type information for all data stores and data flows. This type information may be supplied either textually, by means of VDM domain definitions or, if they are more complex, graphically by means of entity-relationship diagrams. It is now possible to (automatically) derive a first VDM specification. (The *level 0 VDM specification.*) This document is considered to be a secondary document, of little use to the designer, but which can be used to perform a consistency check on the system specified so far. It essentially gives a semantics to the hierarchy of DFDs.

(4) Complete the analysis phase by specifying all primitive data transformers. These specifications are called mini-specifications and must be described either as function or operations definitions in VDM. It is now possible to (automatically) generate a VDM specification where the mini-specifications are taken into account. (The *level 1 VDM specification.*)

(5) For each DFD containing two or more data transformers, control information must be provided defining the order in which the data transformers should be combined. Again it is possible to (automatically) generate a VDM specification using the control information provided in this step. (The *level 2 VDM specification.*) It is also now possible to generate (automatically) *level 1 structure charts* for the designed system.

(6) Refine the mini-specifications into explicit function and operation definitions. It is now possible to generate (automatically) a textual representation of the design in the form of a VDM specification. (The *level 3 VDM specification.*) At the same time it is possible to generate a level 2 structure chart description.

(7) Optimise the formal specification. In this (optional) step the level 3 VDM specification is changed so that it better reflects the non-functional requirements of the problem to be solved. (The *level 4 VDM specification.*) Structure charts which also reflect these changes can be constructed as well. (The *level 3 structure charts.*)

3.3 Yourdon and Z

Semmens and Allen[8, 9] have done extensive work on integrating Yourdon and Z. The approach they have taken is to define a formal syntax, in Z, for the diagrams

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

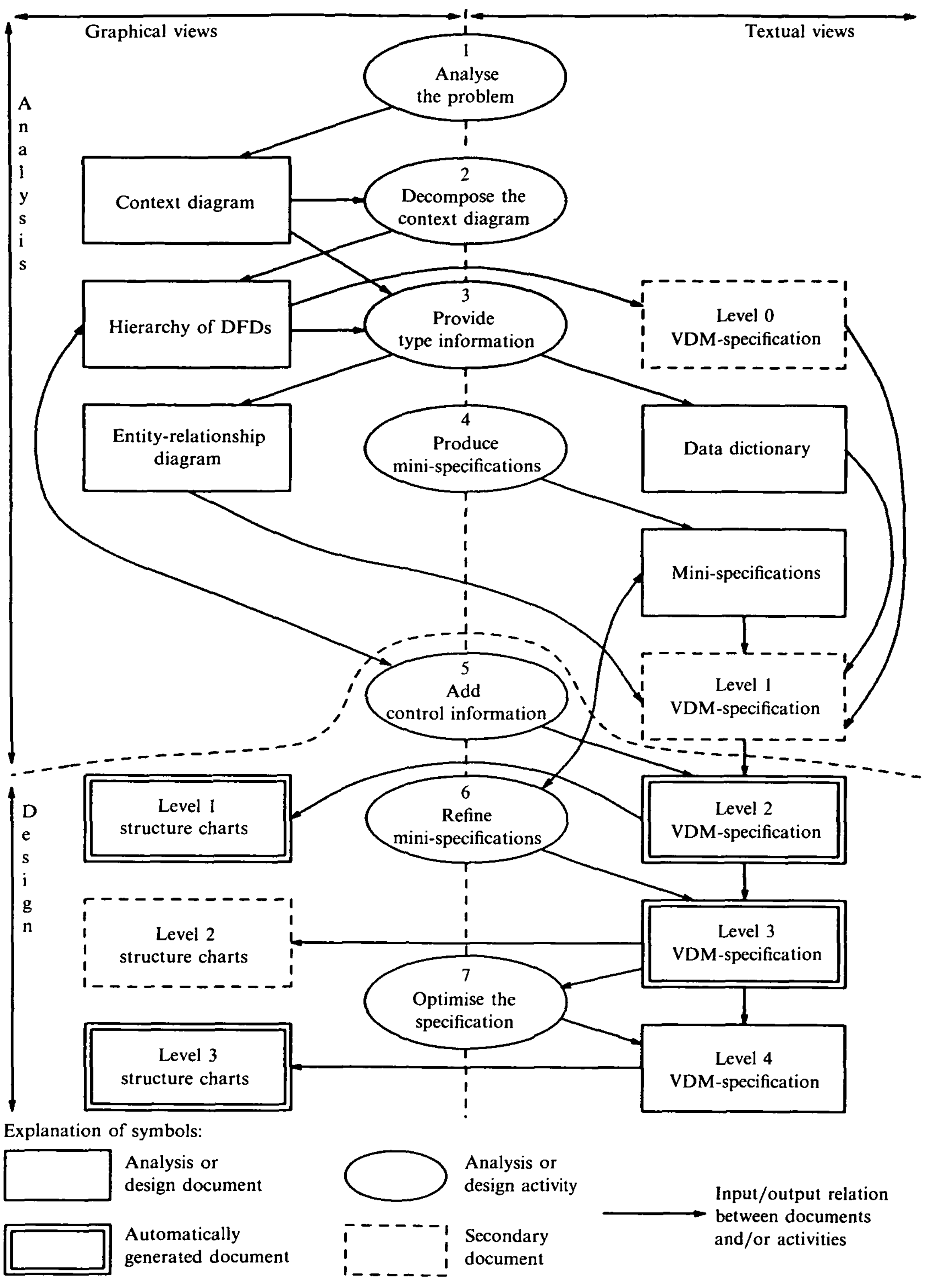

Graphical views
Textual views
Analysis
Design
1 Analyse the problem
Context diagram
2 Decompose the context diagram
Hierarchy of DFDs
3 Provide type information
Level 0 VDM-specification
Entity-relationship diagram
4 Produce mini-specifications
Data dictionary
Mini-specifications
5 Add control information
Level 1 VDM-specification
Level 1 structure charts
6 Refine mini-specifications
Level 2 VDM-specification
Level 2 structure charts
Level 3 VDM-specification
7 Optimise the specification
Level 3 structure charts
Level 4 VDM-specification
Explanation of symbols:
Analysis or design document
Analysis or design activity
Input/output relation between documents and/or activities
Automatically generated document
Secondary document

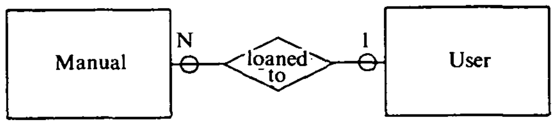
Figure 1. An overview of SVDM.

and data dictionary. This formal syntax together with a set of well-formedness rules provides the specification for a Yourdon tool which comprises the graphical notations (ERD, DFD and STD) and an underlying data dictionary. They have defined functions (in Z) mapping the Z representations of the diagrams to type definitions, state and operation schemas. The formal specification of the system state is generated automatically from the diagrams and the underlying data dictionary, as are the signatures of the operation schemas. A prototype tool[21] is being built which incorporates the Yourdon techniques and the translation rules which generate the (partial) Z specification. The developer then adds pre- and post-conditions to the operation schemas, which are then syntax- and type-checked. The tool will eventually incorporate a proof assistant enabling a completely formal development to proceed.

An example of the relationship between the dia-

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

grammatic representation and the Z representation is shown below.



Manual == @MANNO+TITLE+SUBJECT
User == @USERNO+NAME+LOCATION

Figure 2. Entity relationship diagram and data dictionary.

Z Basic types are defined for each distinct attribute type:

$[MANNO, TITLE, SUBJECT]$
$[USERNO, NAME]$
$[LOCATION]$

Schemas define the entitytypes:

Manual
$manno : MANNO$
$title : TITLE$
$subject : SUBJECT$

User
$userno : USERNO$
$name : NAME$
$location : LOCATION$

Data store schemas represent the instances of each entity, and include an injection from the entity type to its key attribute (unique identifier):

Manual_ds
$manual\_set : \mathbb{P}\ Manual$
$manual\_id :$
$\quad Manual \rightarrowtail MANNO$

$\mathrm{dom}\ manual\_id = manual\_set$
$\forall m : manual\_set \bullet$
$\quad manual\_id(m) = m.manno$

User_ds
$user\_set : \mathbb{P}\ User$
$user\_id :$
$\quad User \rightarrowtail USERNO$

$\mathrm{dom}\ user\_id = user\_set$
$\forall u : user\_set \bullet$
$\quad user\_id(u) = u.userno$

The relationships are declared

Loaned_to_ds
$loaned\_to : Manual \rightarrowtail User$

The complete state is then:

ManualLoans
$Manual\_ds$
$User\_ds$
$Loaned\_to\_ds$

$\mathrm{dom}\ loaned\_to \subseteq manual\_set$
$\mathrm{ran}\ loaned\_to \subseteq user\_set$

If there are any additional constraints on the state they can be added to the entity, data store or state schemas.

Operations schema signatures are generated from the DFD, which provides an operation name, the names of the updated data stores and the names of data flows. The type of the data flow is first specified using a schema.

ManInput
$title : TITLE$
$subject : SUBJECT$

AddManual
$\Delta ManualLoans$
$\Xi user\_ds$
$\Xi Loaned\_to\_ds$
$man\_input? : ManInput$

Pre- and post-conditions are then added by the developer. The main elements of the method incorporating the transformation from Yourdon to Z are illustrated in Fig. 3.

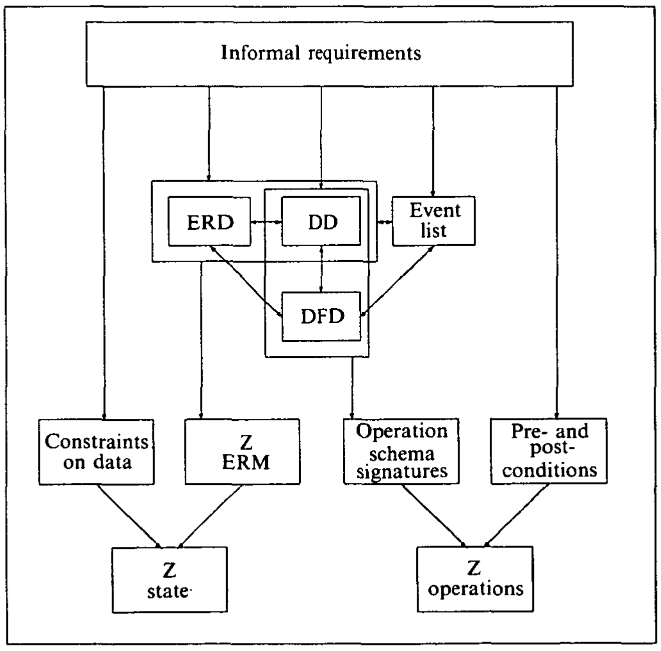


Figure 3. Combining Yourdon and Z for system specification.

3.4 SE/Z

Work on the integration of LBMS SE and Z[11] takes as its starting point the assumption that the use of structured methods as a front end to formal development is a useful approach. They have also examined the scope for enrichment of a structured method by integrating it with a formal notation. The two main aspects of enrichment considered are – enrichment of the method's representational ability (i.e. the notations and their meaning) and enrichment of its transformational ability (i.e. the development process).

The basic mapping of the entity relationship model used in SE is similar to that used by Semmens and Allen, and described above. However, the translation is done at a slightly earlier stage in the development process, before any decisions about which attribute(s) will form the

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

primary key. An abstract identifier is used for each entity. So using the example of a user entity above,

```
 [No,Name,Location]
┌─UserRecord──────────────────────
│ userNo : No
│ userName : Name
│ location : Location
└─────────────────────────────────
```

a table schema is defined:

```
 [UserId]
┌─UserTable───────────────────────
│ knownUser : 𝔽 UserId
│ tableUser : UserId ⇸ UserRecord
├─────────────────────────────────
│ knownUser = dom tableUser
└─────────────────────────────────
```

Database operations are then defined using operations schemas.

What has been done to take the specification constructs used in structured methods (e.g. entity, one-to-many relationship) and use them as building blocks of a Z specification. This is seen as directly analogous to the way that the standard building blocks in the mathematical toolkit (functions, sequences, etc.) are used in Z. To integrate these additional specification constructs they have defined a library of Z generics.[22] These generic schemas define the common semantics of each type of construct.

3.4.1 Enrichment of representation

Natural extensions have been made to the existing specification constructs. Entity invariants, which in most structured specifications are no more than natural language comments, can be captured formally. The subtyping mechanism used in many structured methods is not in any sense rigorous. The use of a formal notation allows their semantics to be defined precisely, which in turn has led to the development of a richer graphical notation which can be used with the confidence that it is semantically sound.

3.4.2 Enrichment of the development process

Four possible ways of enriching the development process are being researched:

- derivation of preconditions
- derivation of consequences
- data refinement
- consequentional operation refinement

4. ALGEBRAIC SPECIFICATION TECHNIQUES

4.1 Introduction

Research related to the integration of the structured analysis (SA) method and algebraic specification techniques started on two fronts in the early to mid-eighties. The work of Tse[6] was aimed primarily at providing a common formal framework for structured methods, including the SA method, to facilitate the translation of artifacts among the methods. On the other hand, the work of Docker and France[7, 24, 26, 25] focused on providing an integrated prototyping and formal specification framework for SA tools to create a flexible and formal specification environment based on SA.

Tse's emphasis on the ability to translate specification products across methods meant that his work focused more on the syntactic structure of specification artifacts, while France and Docker's emphasis on formally stating and investigating behavioural properties modelled by SA artifacts meant that their work focused more on the semantic aspects of SA artifacts. In fact, the algebraic characterisation of the syntax of data flow diagrams (DFDs) presented in France[24] is close to Tse's algebraic characterisation of DFD structure (there are differences in the notations used), but the characterisation itself plays a very insignificant part in the specification framework of France and Docker.

In the next section we discuss the work of France and Docker. This is followed by a brief overview of Tse's approach.

4.2 Formally specifying the semantics of DFDs with algebraic specifications

In this section we describe two frameworks for associating DFDs with formal specifications characterising their behaviour, developed as part of the research carried out by France and Docker. One framework supports the formal specification of sequential systems[25] while the other also supports the formal specification of non-sequential systems.[26] Both frameworks facilitate formal investigation of application properties, and provide bases for formal decomposition and refinement, and formal verification. The framework supporting the specification of sequential systems is more suited to formally specifying requirements, since one is not too concerned about the details of interactions among processing components at that stage. The other framework is suited to the specification of designs, especially designs of complex systems. In what follows a DFD associated with a formal specification is called a semantically Extended DFD (ExtDFD). We conclude with a brief account of a tool SAME (Structured Analysis Modelling Environment), which provides a front end to the formal system.

4.2.1 Specifying the semantics of sequential systems

In the framework supporting the formal specification of sequential systems, a DFD is interpreted as a system of atomic operations accessing instances of shared data structures, where shared data structures are depicted by data stores, and atomic operations by data transforms. The specification characterising the intended semantics of a DFD in this framework is called an Atomic-level Specification (ASpec). An ExtDFD in this framework is thus a DFD associated with an ASpec characterising its intended semantics. We shall use the DFD in Fig. 4 to illustrate the specification technique outlined below.

The ASpec of a DFD consists of two parts: a data domain definition part and an operation specification part. The data domain definition part is made up of three parts (see Fig. 5).

Data definition. This part contains type definitions for all data elements depicted in the DFD. The data definition part is analogous to the data dictionary of SA. Type definitions are stated in a data description (DD) language which provides functions for constructing data types

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

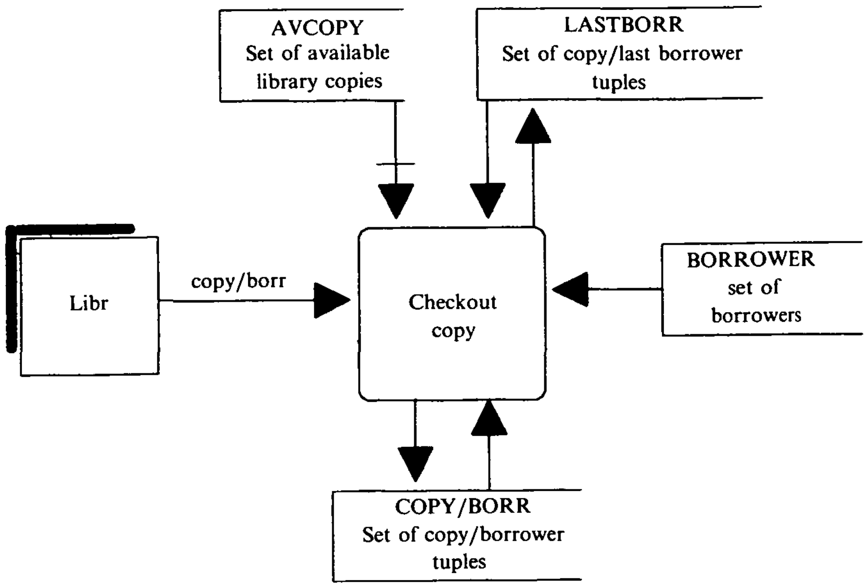

AVCOPY
Set of available library copies
LASTBORR
Set of copy/last borrower tuples
Libr
copy/borr
Checkout copy
BORROWER
set of borrowers
COPY/BORR
Set of copy/borrower tuples

Figure 4. Example DFD for borrowing a library book.

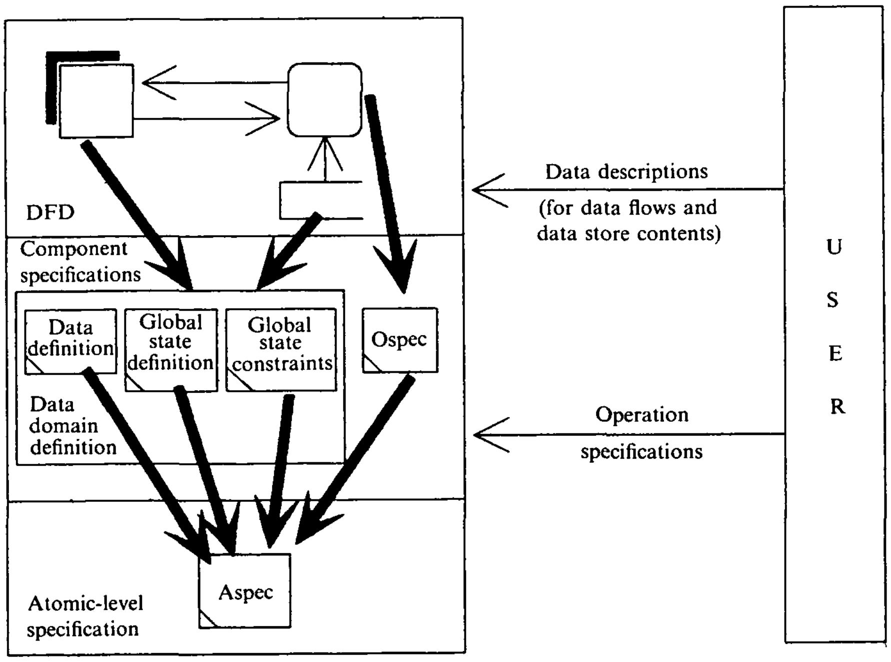


Figure 5. Components in specifying the semantics of sequential systems.

from elementary and pre-defined, parameterised data types. Each type definition is actually an (hierarchical) algebraic specification which defines the type in terms of operations that create and manipulate instances of the type defined. The input/output effects of data transforms are expressed solely in terms of the operations defined in the type definitions. An (partial) example of an algebraic specification corresponding to a type definition is given in Fig. 6.

Global state definition. This part consists of declarations of the data structures associated with the data stores in the DFD. An instance of the data structure associated with a data store is called a state of the data store. France uses sets to model data stores, thus a state of a data store is simply a set whose elements represent

Data specification is set
  Using *NatNum*
  With parameter data (sort *data*)
  sort *set*
  Constructors
  $\emptyset : \rightarrow set$
  insert: *data set* $\rightarrow$ *set*
  Operations
  count: *set* $\rightarrow$ *natnum*
  — Counts number of elements in a set.
  ...
  Operation axioms
  ...

Figure 6. Algebraic specification associated with the pre-defined type set.

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

A data store state specification. The data store $COPY/BORR$ stores 2-tuples of sort $copy/borr = \langle copy, borrower\rangle$, where *copy* is the type of a library book copy and *borrower* is the type of a library borrower.

$COPY/BORR{:}set(copy/bor)$

Access Operations

Signature

$get{:}\ set(copy/borr)\ borrower \rightarrow set(copy)$

—Returns the copies currently checked out by a borrower.

...

Axioms

For all $c{:}copy/borr$; $S{:}set(copy/borr)$; $b{:}borrower$

1. $get(\emptyset, b) = \emptyset$
2. $c \cdot borrower = b \Rightarrow get(insert(c, S), b) = insert(c \cdot copy, get(S, b))$
3. $c \cdot borrower \neq b \Rightarrow get(insert(c, S), b) = get(S, b)$

...

Figure 7. State specification for a data store.

the contents of the data store. This part also contains definitions of the states associated with external entities whose behaviour affects how the application responds to its stimuli. State definitions are representations of algebraic specifications which define states in terms of operations that create and manipulate their instances. State modifications by data transforms must be expressed solely in terms of the operations in these specifications. An example of a state specification for a data store is given in Fig. 7.

Global state constraints. This part stipulates the relationships that must be maintained among the states of data stores and specified external entities. The constraints are stated in first-order predicate logic.

A global state of a DFD is a set consisting exactly of a state for each data store depicted in the DFD and for each external entity associated with states in the global state definition part. A global state that satisfies the global state constraints is said to be valid, otherwise it is said to be invalid.

The operation specification part of an ASpec consists of specifications called OSpecs, each uniquely associated with an operation depicted by a data transform. An OSpec characterises the desired pre- and post-conditions of an operation, and is analogous to a process specification in SA.[29] The pre- and post-conditions stipulated by an OSpec determine the effect a data transform has on the global state of a DFD. An example of an OSpec is given in Fig. 7.

An OSpec is said to be consistent with respect to the global constraints of an ASpec if and only if preconditions do not preclude all valid global states, and an execution starting in a valid global state ends in a valid global state. This consistency condition is formally stated in Ref. 25. Details of how the above framework can be used to verify refinements of ASpecs can also be found there.

4.2.2 Specifying the semantics of non-sequential systems

In the framework supporting the specification of non-sequential systems, DFDs with control extensions, similar to those used in Refs 27 and 28, are associated with algebraic specifications characterising their behaviour.

The OSpec for a 'checkout copy' operation in a library application is given below.

OSpec CheckOutCopy

input

$c{:}copy/borr$; $AVCOPY_{in}{:}set(copy)$; $BORROWER_{in}{:}set(borrower)$; $LASTBORR_{in}{:}set(copy/borr)$; $COPY/BORR_{in}{:}set(copy/borr)$

output

$AVCOPY_{out}{:}set(copy)$; $LASTBORR_{out}{:}set(copy/borr)$; $COPY/BORR_{out}{:}set(copy/borr)$

Operation I/O definition

$c \cdot copy \in AVCOPY_{in}$, $c \cdot borrower \in BORROWER_{in}$, $count(get(COPY/BORR_{in}, c \cdot borrower)) < max \Rightarrow$

$COPY/BORR_{out} = insert(c, COPY/BORR_{in})$,

$AVCOPY_{out} = delete(AVCOPY_{in}, c \cdot copy)$,

$LASTBORR_{out} = updatelb(LASTBORR_{in}, c)$

PreCondition. The copy to be checked out ($c \cdot copy$) must be available for checkout, the borrower ($c \cdot borrower$) must be a registered borrower, and the number of copies currently checked out by the borrower must be strictly less than *max*.

PostCondition. The input c is stored in $COPY/BORR$, the copy checked out ($c \cdot copy$) is made unavailable, and the 'last borrower' relation for the copy is updated to reflect the new last borrower of the checked-out copy.

Figure 8. An Ospec for the data transform CheckOutCopy.

In this framework a control-extended DFD is interpreted as a system of communicating processes (which can also be viewed as a single process), where a process is an entity defined by a set of states and events, and a class of behaviours. The behaviour of a process is defined by a state transition system, which is characterised by an algebraic specification, using a technique developed by Astesiano *et al.*[30] The form of an algebraic specification characterizing a process's state transition system (called an algebraic state transition system or ASTS) is given below:

Transition Specification is *SpecName*

State Specification is *StateSpec*

Label Specification is *LabelSpec*

Transition relation is $\_ \overset{-}{\rightarrow} \_$: *state label state*

Transition axioms *Axioms*

where *StateSpec* is the name of an algebraic specification defining the states of the process, *LabelSpec* is the name of the algebraic specification defining the labels of the transition and *Axioms* are first-order axioms of the form $C \Rightarrow s_i \overset{l}{\rightarrow} s_j$, where s_i, s_j are states defined in *StateSpec*, and l is a label defined in *LabelSpec*.

The semantics of a control-extended DFD (C-DFD) are characterised by an ASTS created in a bottom-up manner from ASTSs characterizing the semantics of individual DFD components in the following way (see Figure 9, where TS1 to TS4 are ASTSs).

(1) Derive ASTSs characterising the behaviour of each C-DFD component from specifier-supplied descriptions. The resulting sets of ASTSs together with the C-DFD is called the Basic Interpreted C-DFD.

(2) Derive an ASTS characterising the synchronous interactions that can take place among C-DFD components from the Basic Interpreted C-DFD. This ASTS is called the Synchronous Interaction Specification (SIS).

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

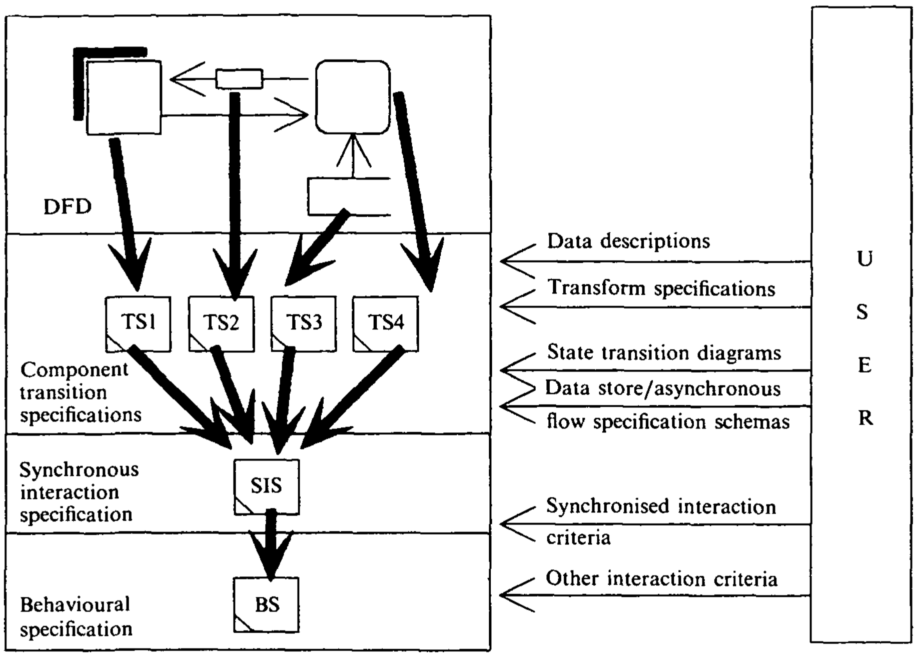

DFD
TS1
TS2
TS3
TS4
Component transition specifications
Synchronous interaction specification
SIS
Behavioural specification
BS
Data descriptions
Transform specifications
State transition diagrams
Data store/asynchronous flow specification schemas
Synchronised interaction criteria
Other interaction criteria
U S E R

Figure 9. Components in specifying the semantics of non-sequential systems.

The SIS together with the C-DFD is called the Basic ExtDFD.

(3) Derive an ASTS characterising the permissible time-dependent relationships among the synchronous interactions specified in the SIS from the Basic ExtDFD. The resulting ASTS is called the Behavioural Specification (BS).

In phase 1 of the specification generation method, ASTSs characterising semantic models of C-DFD components are created from specifier-supplied descriptions and specification schemas. The specifier-supplied descriptions are analogous to the traditional structured analysis (SA) data dictionary definitions and data transform specifications.[20, 29] For data flow and data store components, specification schemas are instantiated with algebraic specifications derived from the specifier's descriptions of transmitted or stored data to produce algebraic specifications characterising their structure and data access behaviour.

In phase 2 it is determined which process actions defined in phase 1 are to be synchronised. For example, a read from a data store action of a data transform must be synchronised with a read action of the data store. A specification defining the effects of synchronous transitions, that is, transitions caused by synchronising actions, is derived from the data and control relationships depicted in the control-extended DFD, using rules which state what type of interactions depicted in a control-extended DFD result in synchronous transitions. The ASTS resulting from this phase consists of transition axioms that define the effect of synchronised events. For example, given that the transitions $p_1 \xrightarrow{l_1} p'_1, \ldots p_j \xrightarrow{l_j} p'_j$ can take place in an application state $\langle p_1, \ldots, p_j, p_{j+1}, \ldots, p_n \rangle$, the synchronized effect of the actions labelled $l_1, \ldots, l_j$, is defined by the axiom:

$$p_1 \xrightarrow{l_1} p'_1, \ldots, p_j \xrightarrow{l_j} p'_j, l = SYNCH(l_1, \ldots, l_j) \Rightarrow$$
$$\langle p_1, \ldots, p_j, p_{j+1}, \ldots, p_n \rangle \xrightarrow{l} \langle p'_0, \ldots, p'_j, p_{j+1}, \ldots, p_n \rangle$$

In phase 3 constraints on when the synchronous interactions can take place are specified. The ASTS derived in this step (the BS) is an extension of the ASTS derived in phase 2, where the extensions concern definitions of the effects of actions represented by parallel action labels. A more detailed account of the activities in each phase can be found in Ref. 26.

4.2.3 SAME (Structured Analysis Modelling Environment)

The research of Docker and France extends beyond a formal interpretation of structured analysis. SAME provides a front end to the formal system which provides:

- the graphical specification of DFDs;
- the definition of data;
- an executable dictionary (repository) which stores the DFDs and data definitions, and allows the execution of application models (as prototypes);
- the definition and execution of incomplete models;
- a 'soft-fail' environment, in which errors are trapped and analysed and in which the user is generally able to correct the error and continue from the point of failure.

The major feature of SAME have been described in some detail elsewhere,[31] and will not be discussed further here. The models defined in SAME can be translated into the formal specifications outlined in this paper. Effort is currently being spent on this translation, and on the reporting back of the essential details of the results of the translation through the SAME interface.

4.3 Tse's unifying framework for structured analysis

To facilitate translation of products among structured systems development models, Tse developed a unifying formal framework for the models. The framework is based on an abstract model which captures the common structures of the structured methods. This unifying

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

model is the initial-term algebra of an algebraic specification characterising the common structures. Structured development models can then be translated to the unifying model, which in turn can be translated to other structured systems models (via the unique homomorphism property of initial algebras), thus effecting translation among the models.

In his work Tse focuses on three structured development tools, DeMarco type date flow diagrams,[29] Yourdon Structure Charts (SCs)[32] and Jackson's Structure Text (JST).[33]

5. PROCESS ALGEBRAS

Some work has been done on the integration of structured methods and process algebras. Some research at Oxford[36] has looked at JSD[34] and CSP, while recent work at Teesside Polytechnic has examined the link between Yourdon and CCS. It is the second of these which we will describe.

5.1 Yourdon/CCS

An integrated method using Yourdon and CCS for the specification of real-time systems has been developed at Teesside Polytechnic.[12]

The method proceeds thus.

(1) Derive context diagram. In the case of real-time systems the externals will tend to be devices which are controlled by the system rather than people who interact with the system. Externals may also, of course, include other systems.

(2) Derive system properties in English. In order to (automatically) check the formal model against the requirements it is necessary to formalise the requirements themselves. To achieve this, desirable properties of the system are identified from the user and user requirements.

(3) Specify system properties formally, the required properties are then translated into Hennessey–Milner logic, which is supported by the concurrency workbench. This step also continues to define the vocabulary for the intercommunication components which was begun in step 2.

(4) Derive DFD explosions. Break the transforms down into as many levels as necessary. It is helpful if the data and control aspects of the system can be separated without compromising the model. The behaviour of the data transforms can be defined either during this step or the next one. It is possible, by allocating a control transform to each device (a device controller), to develop Ward/Mellor data and control flow diagrams which are equivalent to CCS flow graphs (see Ref. 12 for a detailed discussion of this approach).

(5) Derive State Transition diagrams. There should in fact be two STDs for each control transform, one for the active state initiated by the enable action and the other for the inactive state initiated by the disable. In practice the inactive state is defined by implication.

(6) Convert STDs to CCS. This is done automatically, but some constructs produce complex alternative behaviours. It may be necessary to manually refine some agents; ideally this would be done via the STDs to maintain the continuity and revisability of the entire model.

The behaviour of a Ward/Mellor control transform is defined as an STD. This diagram documents, for each state the transform can attain, the event flows it will respond to (transitions), the event flows it will create (actions) and the states it will attain as a result. Similarly, in CCS an agent's behaviour is defined as the possible sequences of input and output actions it may take part in and the agent(s) it will become as a result.

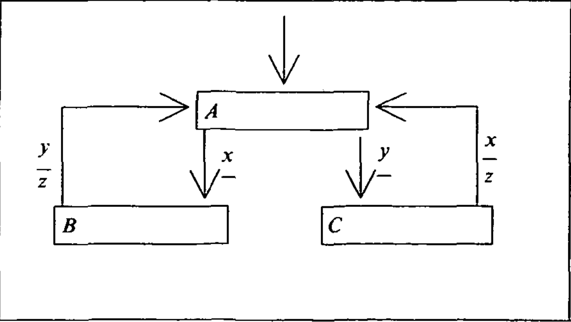
An example of the translation of an STD into CCS is given in Fig. 10. There is one CCS agent for each state in the STD and one for the STD itself. The latter agent is required to handle the *enable* signal (the unlabelled arrow) which every control transform must have.

(7) Check functioning of model. Ensure that the model can behave in the desired manner. This entails using the Concurrency Workbench to find deadlocks and to animate the model in various ways.

(8) Change model. Any changes required to get the model to function as required must be performed on the structured model. At best this means changing the flow in an STD (step 5) and at worst soliciting new requirements from the customer (step 1).

(9) Check properties against model. When it has been established that the model can behave as required, use the Concurrency Workbench to validate the model against the properties expressed formally in HML. Changes may again result from this step.

The work at Teesside is continuing, and they are exploring the possible use of LOTOS[35] in place of CCS. They are also working on the interface between their own structured methods tool 'Ascent' and the Concurrency Workbench so as to underpin the method.



$$\text{Example\_STD} \overset{\text{def}}{=} \text{enable}.\text{A}$$

$$A \overset{def}{=} x.B+y.C$$

$$B \overset{def}{=} y.\overline{Z}.A$$

$$C \overset{def}{=} x.\overline{z}.A$$

Figure 10. Example translation of an STD into CCS.

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

6. OTHER RELATED RESEARCH

Work at BT[37] has led to the development of a rigorous review technique where the products of structured analysis are transformed into a formal system specification. This transformation process has been used to review a number of relatively complex specifications. The process reveals many errors in the structured specification which have not been found during the normal review process. It is hoped to publish the results of this work in the near future.

The problem of comprehending large formal specifications is being looked at by some researchers. Graphical notations are being developed which correspond to the detailed structure of the formal specification. These notations are being developed for VDM using Software Through Pictures[38] and for Z by Randell.[18]

7. CONCLUSION

This paper has outlined some of the current approaches to integrating structured methods of software development with formal notations. Most of this research is at a relatively early stage, although there is evidence that the approach is workable. The integration of structured methods and formal notations aims to overcome some of the problems encountered when formally specifying large systems. Users of structured methods have found that without tool support the methods become unmanageable. It would therefore seem evident that if the integrated methods discussed here are to be of practical use on large systems, tools are essential. Many of the researchers are aware of this, and we can soon expect to see reports and demonstrations of their efforts.

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024

REFERENCES

1. E. Yourdon, *Modern Structured Analysis.* Prentice-Hall, Englewood Cliffs, New Jersey (1989).
2. *SSADM Version 4 Reference Manual.* NCC/Blackwell, Oxford (1990).
3. V. Hamilton, Experience of combining Yourdon and VDM. *Proceedings of the Methods Integration Workshop, Leeds, September 1991.* Springer-Verlag, Heidelberg (1992).
4. F. Polack, M. Whiston and P. Hitchcock, Structured analysis – A draft method for writing Z specifications. In *Proceedings of the Sixth Z User Workshop, York, 16–17 December 1991*, edited J. Nicholls. Springer-Verlag, Heidelberg (1992).
5. S. Goldsmith, Using the Yourdon Structured Method (YSM) and the Vienna Development Method (VDM) together during the system lifecycle. *BCS CASE TOOLS seminar, IEE* (November 1989).
6. T. H. Tse, *A Unifying Framework for Structured Analysis and Design Models.* Cambridge University Press (1991).
7. R. B. France and T. W. G. Docker, A formal basis for structured analysis. *Software Engineering* **88** (1988).
8. L. T. Semmens and P. M. Allen, Using Yourdon and Z: An approach to formal specification. In *Proceedings of the Fifth Z User Workshop, Oxford, 17–18 December 1990*, edited J. Nicholls. Springer-Verlag, Heidelberg (1991).
9. L. T. Semmens and P. Allen, Formalising Yourdon. In *Proceedings of the Methods Integration Workshop, Leeds September 1991*, edited P. Allen, A. Bryant and L. Semmens. Springer-Verlag, Heidelberg (1992).
10. P. G. Larsen, J van Katwijk, N. Plat, K. Pronk & H. Toetenel, SVDM: an integrated combination of SA and VDM. In *Proceedings of the Methods Integration Workshop, Leeds, September 1991*, edited P. Allen, A. Bryant and L. Semmens. Springer-Verlag, Heidelberg (1992).
11. D. Redmond-Pyle and M. B. Josephs, Enriching a structured method with Z. In *Proceedings of the Methods Integration Workshop, Leeds, September 1991*, edited P. Allen, A. Bryant and L. Semmens. Springer-Verlag, Heidelberg (1992).
12. P. C. Fencott, M. A. Lockyer and P. Taylor, Experiences in integrating structured and formal notations for real-time systems. In *Proceedings of the Methods Integration Workshop, Leeds, September 1991*, edited P. Allen, A. Bryant and L. Semmens. Springer-Verlag, Heidelberg (1992).
13. J. Rose, *PhD thesis.* University of Bristol (1992).
14. C. Jones, *Systematic Software Development using VDM.* Prentice-Hall, Englewood Cliffs, New Jersey (1986).
15. A. Bryant, Structured methodologies and formal notations: developing a framework for synthesis and investigation. In *Z user workshop, Oxford 1989*, edited J. E. Nicholls. Springer-Verlag, Heidelberg (1991).
16. J. M. Spivey, *The Z Notation: A Reference Manual.* Prentice-Hall, Englewood Cliffs, New Jersey (1989).
17. J. Nicholls (ed.), *Proceedings of the Fifth Z User Workshop, Oxford 17–18 December 1990.* Springer-Verlag, Heidelberg (1991).
18. G. P. Randell, *Translating Data Flow Diagrams into Z (and vice versa).* RSRE Report 90010 (1990).
19. G. P. Randell, *Improving the Translation from Data Flow Diagrams into Z by Incorporating the Data Dictionary.* RSRE Report 92004 (1992).
20. Chris Gane and Trish Sarson, *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, Englewood Cliffs, New Jersey (1977).
21. A. S. Evans, A Prototype Tool for Yourdon and Z. *MSc Dissertation*, Sheffield Polytechnic (1992).
22. M. B. Josephs and D. A. Redmond-Pyle. *A Library of Z Schemas for use in Entity Relationship Modelling.* Programming Research Group, Oxford University (1991).
23. R. B. France and T. W. G. Docker, Flexibility and rigour in structured analysis. In *Information Processing 89.* Elsevier Science, Amsterdam (1989).
24. R. B. France, *A Formal Framework for Data Flow Diagrams with Control Extensions.* Massey University, New Zealand (1990).
25. R. B. France, *Formally Specifying Sequential Systems with Semantically Extended Data Flow Diagrams.* (Details from R. B. France, Inst. for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA) (1992).
26. R. B. France, *Semantically Extended Data Flow Diagrams: A Formal Specification Tool.* UMIACS, University of Maryland at College Park (1991).
27. D. Hatley and I. Pirbhai, *Strategies for Real-Time System Specification.* Dover Press (1987).
28. P. T. Ward and S. J. Mellor, *Structured Development for Real-Time Systems.* Yourdon Press (1981).
29. T. DeMarco, *Structured Analysis and System Specification.* Prentice-Hall, Englewood Cliffs, New Jersey (1978).
30. E. Astesiano, A. Giovini and G. Reggio, Data in a concurrent environment. In *International Conference on Concurrency.* Springer-Verlag, Heidelberg (1988).
31. T. W. G. Docker, SAME – a structured analysis tool and its implementation in Prolog. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium.* MIT Press, Cambridge, Mass. (1988).
32. E. Yourdon and L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and*

Systems Design. Prentice-Hall, Englewood Cliffs, New Jersey (1979).
33. M. A. Jackson, *Principles of Program Design*. Academic Press, London (1975).
34. M. A. Jackson, *System Development* Prentice-Hall, Englewood Cliffs, New Jersey (1981).
35. K. J. Turner, LOTOS – a practical formal description technique for OSI. *Proceedings, International Conference on Open Systems, London, March 1987.*
36. M. B. Josephs, *JSD and CSP*. Programming Research Group, Oxford (1989).
37. L. Renshaw and P. Davies, *TELSTAR Rigorous Review Technique*. BTRL Martlesham, Internal Report (1992).
38. J. Dick and J. Loubersac, Integrating structured and formal methods: a visual approach to VDM. In *Proceedings of ESEC '91 Milan, Oct. 1991.*

Book Review

ALAN C. GILLIES
The Integration of Expert Systems into Mainstream Software
Chapman & Hall, 1991. £19.95. ISBN 0-412-39930-X.

The various branches of computer science have developed distinct subcultures which are, in part, defined by the programming languages that they favour and their styles of design and development. The profession of systems programmer arises from the need to bridge the gap between the subculture of operating systems developers and that of application programmers. If database systems had had their roots in artificial intelligence, there would be a similar niche for programmers who could work across the Lisp–Cobol fault line. But it is the real gap between the subcultures of expert systems and software engineering that provides the subject matter of Alan Gillies' book.

It recommends three strategies for bridging this gap. One is a modified version of the technique described in T. DeMarco's *Structured Analysis and Systems Specification*. This is referred to as 'structured integrated expert system methodology' and aims to involve the disciplines of software engineering during the knowledge-acquisition phase of developing a system. The second strategy, called 'data centred design', is proposed for systems in which a database and a knowledge-based system mutually interact in use but which may be developed, at least, partly, independently of each other. The final strategy, 'prototyping and porting', is succinctly described by its name.

Although these strategies are supported by case studies, their initial exposition is quite brief. They seem plausible enough, but no justification for them is offered and no alternative reviewed. Space in the book which could have been used for this purpose is devoted to introductory treatments of expert systems, software engineering and human–computer interaction. This seems to me to be a mistake; and I am not sure that the author, who is a lecturer in the Information Technology Institute of the University of Salford, has succeeded in his aim of producing a suitable 'teaching text' for BSc and MSc students. Such students need a rather more comprehensive understanding of these topics than can be provided in single chapters before they study the problems of software integration.

But if the book is weak because it includes too much introductory material, its strength lies in its generous use of extended examples. Three chapters are devoted to case studies: one is concerned with a system to automate photoelastic stress analysis, a second case study describes a military decision support system and the final case study is an 'expert database system', which handles PAYE problems. These are clearly written in unpretentious language and contain references to source material.

Until some happy future time when the various branches of computer science are unified by an all-embracing theory, there will be a demand for books like Alan Gillies' which attempts to help those forced to deal simultaneously with more than one computing subculture.

R. E. COOLEY
Canterbury

Announcement

21–23 MAY 1993

PDK '93, International Workshop on Processing Declarative Knowledge – Representation and Implementation Methods – Yorktown Heights, New York, U.S.A.

The PDK workshops aim to provide a forum for discussing the practical and theoretical aspects of processing declarative knowledge. A main topic of interest is the trade-off between the need for efficient processing and the desire for meaningful and modular specifications. Work that addresses linguistic, semantic and processing issues in an integrated fashion is specially welcome.

The first PDK workshop was held in Kaiserslautern, Germany, and gathered researchers and practitioners from the areas of AI, Logic Programming and Databases. For our next workshop in 1993 we are inviting contributions on theoretical ideas, practical techniques, and the development of systems or software tools in the area of knowledge-based systems.

Topics of interest include but are not limited to:

- extensions and variations of declarative programming paradigms (e.g. logic, rule-based, constraint and object-oriented programming);
- reasoning methods for dealing with disjunctions, constraints, probabilities, etc.;
- architectures for default, causal, abductive and temporal reasoning;
- studies of processing algorithms for complexity–expressivity and completeness–soundness trade-offs.

Abstracts submission

Please submit four (4) copies of an abstract of up to 10 pages, written in English, by 5 January 1993 (not via email or fax).

The abstracts should accurately represent the theme of the proposed presentation during the workshop: synthesis of important previous results, new significant results or directions, descriptions of ongoing research projects, design, implementation and applications of software tools and systems.

After the workshop is held we shall consider publication of full papers in a special issue of the *Annals of Mathematics and Artificial Intelligence*.

Authors will be notified of acceptance or rejection of submitted abstracts by 20 February 1993.

Please direct contributions to the following address:

PDK '93, c/o W. Zadrozny, IBM Research, T. J. Watson Research Center, 30 Saw Mill River Road (route 9A), Hawthorne, NY 10532, USA.

Conference Chair

Wlodek Zadrozny, IBM Yorktown Heights, wlodz@watson.ibm.com.

Program Co-chairs:

Hector Geffner, Universidad Simón Bolívar, Caracas, and IBM Yorktown Heights, hgeffner@conicit.ve. Jean-Louis Lassez, IBM Yorktown Heights, jll@watson.ibm.com.

Downloaded from https://academic.oup.com/comjnl/article/35/6/600/352763 by guest on 10 April 2024