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:

CASE Support for Methods Integration: Implementation of a Translation from a Structured to a Formal Notation

S. Hooker, M.A. Lockyer and P.C. Fencott

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

CASE Support for Methods Integration : Implementation of a translation from a structured to a formal notation

S. Hooker
M. A. Lockyer
P. C. Fencott

School of Computing and Mathematics Centre for Modelling and Simulation, University of Teesside
Middlesbrough, Cleveland, ENGLAND.

Abstract

Work carried out at The University of Teesside has resulted in an integrated method between the Ward Mellor (WM) Structured Analysis Real-Time (SA/RT) notation and Value Passing Synchronous Calculus of Communicating Systems (SCCS-VP), an extension to SCCS to cater for values. This is achieved through a formally specified Semantic Function (SF) which defines a mapping from WM models to their re-expression in SCCS-VP.

The work presented here takes the Z specification of the Semantic Function and implements it in the functional programming language, ML. This paper looks in turn at the steps found necessary to develop a complete CASE supported integration of WM models to SCCS-VP program translation, and their simulation on the Edinburgh Concurrency Workbench (CWB).

1 Introduction

Methods Integration is the combination of two (or quite possibly more) software development methods, usually from the different paradigms of Structured, Object Oriented and Formal. The efforts at the University of Teesside have resulted in such an Integrated Method [3] using Ward-Mellor SA/RT from the structured paradigm and Value Passing Synchronous Calculus of Communicating Systems from the formal.

Numerous integrated methods have been proposed e.g. [12, 13, 15, 14, 16] which consist of formalising the semantics of the less formally defined notations common to structured analysis. The work carried out at the University of Teesside has resulted in an extensive integration, between Ward-Mellor SA/RT and SCCS-VP, with provision for the entire Ward-Mellor *Essential Model*. This model is defined abstractly [7, 6], in Z [17], from which its concrete representation in ML [20] is derived.

WM models expressed in the syntax familiar to structured analysis are translated into their equivalent concrete representation, ready for input to the implementation of the Semantic Function. The output from the SF, an SCCS-VP program, is similarly defined abstractly in Z and concretely in ML for use in the translation.

Ward and Mellor is a widely used SA/RT method and is supported on the ASCENT Version 3.0 [9] CASE tool developed at the University of Teesside. The Concurrency Workbench [11] allows for the simulation and checking of formal properties, of SCCS programs. So already in place is stand alone tool support for the two paradigms and the Semantic Function [5] definition of an integration from the structured to the formal.

Ward and Mellor models in the structured domain are referred to as simply WM models or Ward and Mellor models. Reference to the abstract definition of WM models, or a specific abstract WM model, shall be made using the acronym WMZ, for Ward and Mellor Z. This shall also apply to the concrete representation (in ML) of WMZ.

This gives precise definitions of a source object (WM), target representation (SCCS-VP) and how to arrive at the latter from the former via the application of the semantic function. This could conceivably be achieved by manually applying the re-expression rules of the translation but undoubtedly there is scope for automation.

This work aims to address the questions of automation

- To provide automated assistance to the integrated method.

- To model larger systems in order to test the method.
- To investigate the extent that the method could be automated.
- To use this as a first stage in the development of a fully automated integration, beyond that of translation, which may:
  1. Highlight areas of concern, uncovered by analysing the formal model, on the structured model.
  2. Allow for queries to be made at the structured model in a less formal manner, which are themselves translated with the WM model to there formal calculus.

1.1 Ideal Tool Situation

The ideal situation of automated support is depicted in Figure 1.1, showing the construction of a WM model on ASCENT and automated translation to and input of SCCS-VP into the CWB. Thus, giving a structured analysis based front end to the production of a formal model. The advantages of a structured method with its ease of use, of conveying ideas and communication is supported, in fact strengthened by its equivalent formal representation which can then be subjected to proof and other formal verification techniques. The observations from which are shown feeding back to the structured model. The process as a whole offering the advantages of both the structured and formal domains.

Figure 1: Ideal Integration of ASCENT, Semantic Function and CWB

1.2 Current State of Tool Automation

The current situation is less automated as a whole and involves a number of extra sub-steps, as shown in Figure 2. The important aspect is that invocation of the Semantic Function on the WMZ model results in an SCCS program which is CWB ready. Two trivial differences between the scenarios are the need to go from PC to UNIX and invoke the SF and then to invoke the CWB on the result. This is only a short term

Figure 2: Current Integration of ASCENT, Semantic Function and CWB

issue as we have recently had some success in running the CWB under Windows 95 and this will enable a more complete integration with ASCENT shortly, leading eventually to the *ideal situation*.

At this time the CWB accepts only Basic SCCS (or simply SCCS) [10], i.e. it has no value passing extensions. This requires the production of a translator from SCCS-VP (the output of the Semantic Function) to SCCS. The specification of this can be found in [4], and this combined with parts of an existing translation [1] from CCS-VP to Basic CCS [10], contributed heavily to the SCCS-VP to SCCS translation. This is a substantial and crucial difference, but it is subsumed into the ML program and not visible. A more intrusive difference at this time is the need for some manual intervention in the production of the WMZ model, which is elaborated later. Extensive developments were necessary to make ASCENT produce the WMZ and to obtain the SCCS-VP to SCCS translator.

2 Ward Mellor Models Expressed in ML

This section contains some details of how the WM model is arrived at in a form suitable for input to (the implementation of) the Semantic Function. The formal specification language, Z, has been used to specify the components of a WM model [7, 6], and this specification was used in this work to represent WM models in ML.

The data flow diagram of WM uses the established symbols (or similar) and their meaning [2], common to numerous structured analysis techniques, and are constructed by following the WM method. The DD and minispecs are textual and may be expressed by using anything from plain English to a formal specification language. They support the diagrams with the DD providing a logical foundation for consistency and correctness checks, and the minispecs defining low level detail not visible on the DFD.

2.1 DFD to Z to ML example extract

As an example of how an ML data type definition is arrived at from its Z specification consider the data flow *volume of contents*, from the terminator *Val* to the process *Mixer System* shown on Figure 3. A data flow is

an object on the DFD and thus requires representation in the Z specification of a WM model (similarly for the data stores, terminators, event flows and transformations) and therefore has a WMZ representation.

Figure 3: Example Data Flow from Silly Mixer

The Z definition for a data flow in WM [7] is given as

<i>DF_TYPE</i>	
<i>name</i> : <i>IDENT</i>	
<i>orig</i> : <i>IDENT</i>	
<i>dest</i> : <i>IDENT</i>	
<i>type</i> : <i>FLOWTYPE</i>	
[<i>IDENT</i>]	(Z)
<i>perim</i> : <i>IDENT</i>	(Z)

where the given set *IDENT* is the set of all possible names of DFD components including the above special identifier *perim*, as shown by the accompanying axiom. Every data flow is categorised as either being time discrete or time continuous by its *FLOWTYPE*.

$$FLOWTYPE ::= Time\_Continuous \mid Time\_Discrete \quad (Z)$$

The ML data types used to implement this are

```

datatype DF_TYPE = Df_Type of IDENT * IDENT * IDENT * FLOWTYPE;
datatype FLOWTYPE = time_continuous \mid time_discrete;
datatype IDENT = perim
                \mid Ident of string;

```

(ML)

Resulting in the following data object representing the data flow *volume of contents*.

```

Df_Type(Ident "volume_of_contents", Ident "VAT", Ident "MIXER_SYSTEM",
        time_continuous);

```

This gives rise to the three types *DF_TYPE*, *FLOWTYPE* and *IDENT*, for use in the implementation of the translation. The many Z Schemas used to define the DFD, DD and minispecs are treated in a similar manner to obtain their respective data types for use in the translation.

2.2 The Data Dictionary

The data dictionary in WM defines the data within the essential model. This data is represented as being stored (at rest) by data stores, and the movement of data is depicted via the use of data flows. The data dictionary consists of *elements*, which are indivisible entries, and *structures*, which are composite entries made up of collections of elements and possibly other structures. All elements and structures are typed, which determines the set of values that an element can take on and the type of a structure is gleaned from the elements and structures composing it. The data flows and data stores are then declared using existing elements and structures, with additional attributes if necessary.

Unlike the DFD which is automatically translated from its graphical syntax into the WMZ equivalent by ASCENT, syntax does not exist to define the DD in a form suitable for inclusion into ASCENT and amenable to translation into its WMZ form. At this time what does exist are the value expressions, a subset of which was shown earlier. From the structured model, ASCENT produces a 'skeleton' of the WMZ data dictionary from the DFD and some DD information. Manual intervention is required to complete this using the concrete (i.e. ML) syntax for the value expressions. An example of this is shown later in Section 3.1.

2.3 Mini Specifications

The minispecs define the behaviour of functional primitive data transforms by relating the values of their output flows to those of their input flows. Like the data dictionary this is textual support, and the ways of expressing this relationship are many.. For example Structured English, Decision Tress, and Pre-condition and Post-condition pairs [19].

The semantic function requires the expression of minispecs in the pre/post-condition format, and this is also advocated by WM. Therefore, a new technique has not been introduced but a formal syntax and semantics has been established for WM minispecs [6].

At present the process of obtaining the minispecs in WMZ form is akin to that for the data dictionary. Every minispec has a set of declarations which can be gleaned from the DFD via the input and output flows of the functional primitive in question, allowing for the automatic production of a 'skeleton' WMZ minispec. However, though WM advocate the use of pre and post-conditions, a syntax does not exist that is sufficiently 'soft' for use in a structured method and syntactically rigorous for translation into the WMZ model.

Again manual intervention is necessary to fill in the 'gaps' left by ASCENT in the minispecs. Once in the WMZ form the minispecs are translatable to SCCS-VP (and Basic SCCS). So in place is abstract syntax for the DD and minispecs, corresponding concrete syntax and a translator which accepts a complete WMZ model and produces the desired formal model.

As the value expression syntax is formal in nature, ASCENT is not going to be extended to allow DD and minispec production using this notation, which is unsuitable for use at the essential modelling stage. Future work will look at the development of syntax sufficiently user friendly for use in the essential model and amenable to translation into WMZ form.

3 Implementation of the Semantic Function

The Semantic Function defines the re-expression of WM models in terms of SCCS-VP agents and the details of how this is achieved are fully documented in [5].

The Essential Model in its WMZ form as a *Valid_Flow_Model* [7, page 21], is input to the Semantic Function, and the resulting output is an SCCS-VP program representing the formalisation of the structured Ward/Mellor model.

$$\begin{array}{|l}
 \hline
 \text{SemFun} : \text{Valid\_Flow\_Model} \longrightarrow \text{SCCS\_Prog} \\
 \hline
 \text{SemFun} = \{ M : \text{Valid\_Flow\_Model} \bullet \\
 \quad M \mapsto \text{DeclSettoSeq}(\\
 \quad \quad \text{DDConststoConsts}(M.AMS.DataD.consts) \cup \\
 \quad \quad \text{FMtoLabels}(M) \cup \\
 \quad \quad \text{FMtoAgents}(M)) \} \\
 \hline
 \end{array} \tag{Z}$$

The data elements and structures in the data dictionary are translated into SCCS-VP **const** declarations. SCCS-VP **label** declarations are obtained from event flows and data flows within the WMZ model, requiring both DFD and DD information. Finally, the SCCS-VP **agent** definitions are obtained from the DFD, minispecs and store definitions.

The DFD is a single compound data object [7, page 10], consisting of a title, sets of terminators, data flows and event flows and the context diagram process. This process is a non functional primitive consisting of a name and the set of its constituent DFD components. It is defined in SCCS-VP as the parallel composition of its subordinate non-flow components, some of which themselves may be non functional primitives and they in turn are defined as the composition of their subordinate non-flow components. This strategy continues until there are no more non-functional primitive data transforms. The result, similar to that of the DFD, is a levelled set of agent definitions.

3.1 The Semantic Function in Action

Here we present a small example of the translation of part of a WM model to SCCS-VP. Consider Figure 3, shown earlier, and assume the following data dictionary entries in the structured model:

- An element, *voc*, declared to range over integers 0 to 4.
- The data flow, *volume of contents*, is typed with respect to *voc*.

In isolation, these two DD entries are translated by ASCENT into the following form:

```
(Variable "voc", null)
(Ident "volume_of_contents", null)                                     (WMZ)
```

where *null* signifies that a value expression is required, and in the case of data flows this may be explicitly written or, as in this case, be a variable. The manually completed version of these entries is like so:

```
(Variable "voc", Enumset[Const(Con"0"), Const(Con"1"), Const(Con"2"),
                        Const(Con"3"), Const(Con"4")])
(Ident "volume_of_contents", var(Variable"voc"))                       (WMZ)
```

where *Const*, *Con*, *var*, *Variable* and *Ident* are used in the WMZ model and derived from the Z specification of the data dictionary and value expressions in the manner shown in Section 2.1. This constitutes an extract of the completed WMZ data dictionary, ready for automatic translation to SCCS-VP.

```
const voc = {0,1,2,3,4}
label volume_of_contents(voc)                                         (SCCS-VP)
```

Now, assume that the flow, *volume of contents* is to a primitive process, *F* (Figure 4), on the main data flow diagram. Where, upon request, *F* is defined to read the value on *volume of contents* and issue *empty* if this value is zero, *full* if 4 and *ok* otherwise.

Figure 4: Example Primitive Transform

The specifics of mini specification production shall be ignored here, in favour of presenting only the resulting SCCS-VP agent which defines this behaviour.

```
agent F = 1:F +
  trigger#volume_of_contents(x) if x = 0 then #empty:F
  else if x = 4 then #full:F
  else #ok:F                                                         (SCCS-VP)
```

4 Result of the Semantic Function

The result of the translation is an ML data object representing the SCCS-VP program. At this point the application of the Semantic Function has returned its result and the optional branch shown in Figure 2, to produce an SCCS-VP program string, can be employed. However, for practical purposes further work was necessary beyond the implementation of the SF to obtain Basic SCCS which is CWB ready.

4.1 SCCS-VP to Basic SCCS

In order to simulate the formal representation of the WM model it was necessary to first convert its defining SCCS-VP program into SCCS, due to the CWB accepting only SCCS at this time. This conversion is fully defined in [4] and below is the basic SCCS agent for the value passing agent, *F*, from the previous section. It shows the parameterised value passing particle, *volume_of_contents(x)*, explicitly instantiated with each of the values that the variable, *x*, may take.

```
F  $\stackrel{\text{def}}{=} 1:F +$ 
  trigger#volume_of_contents_0#empty:F +
```

```

trigger#volume_of_contents_1# $\overline{ok}$ :F+
trigger#volume_of_contents_2# $\overline{ok}$ :F+
trigger#volume_of_contents_3# $\overline{ok}$ :F+
trigger#volume_of_contents_4# $\overline{full}$ :F

```

(SCCS)

For any value passing action with a parameterised particle, the result will be as many basic SCCS actions as there are values for the parameter. If no parameterised particles exist in the action, then the same single action results. However, if more than one particle in an action is parameterised, then we get as many basic actions as the product of all the values that the particles may take. For example, assume the input from two sources, *Vats* say, each ranging over the variable, *voc*, we get a value passing action

```
vat1_volume(x)#vat2_volume(y)
```

and basic SCCS actions

```

vat1_volume_0#vat2_volume_0
vat1_volume_0#vat2_volume_1
vat1_volume_0#vat2_volume_2
vat1_volume_0#vat2_volume_3
vat1_volume_0#vat2_volume_4
:
vat1_volume_4#vat2_volume_4

```

4.2 Concurrency Workbench Ready SCCS

An SCCS-VP agent declaration takes the form of

```
agent agent_name = agent_expression
```

and its equivalent in SCCS may (or may not) be a set of declarations of the form

```
Basic_agent_name = Basic_agent_expression
```

Each SCCS agent declaration is then presented to the CWB in the following format.

```

bi
Basic_agent_name
Basic_agent_expression

```

(CWB)

The CWB command **bi** is used to bind an identifier (or agent name) to an agent expression.

4.3 Simulation of Model

The Edinburgh Concurrency Workbench allows agent behaviour to be interactively simulated. With a WM model formalised in SCCS, it is possible to simulate its behaviour. Furthermore, propositions may be formulated in a powerful modal logic to check that the model (or sub-systems of it) satisfy a specification in this logic.

5 Case Study - The Silly Mixer

The *Silly Mixer* is a non-trivial model used for developing and simulation testing on the CWB [8]. The DFD was constructed on ASCENT and along with the DD and minispec skeletons, converted automatically to its WMZ form. The DD and minispecs were then completed manually and the WMZ model input to the implementation of the Semantic Function. During the course of developing the ASCENT to CWB stages, a number of consistency and logistical problems were encountered. Some of which necessitated modifications to the DFD, which did not result in the essence of the model changing but prevented syntactic and semantic errors in the WMZ model which would otherwise have been created.

5.1 Problems Expressing the Silly Mixer in WMZ

These problems are centred around the DFD conversion, taking the ASCENT diagram and producing the equivalent in ML, but in such a way that formal syntactic and semantic meaning were preserved. For examples, WM components on the DFD can be named with more than one word, thus using spaces, and flows can also be duplicated. Firstly, the spaces are syntactically incorrect when translated to SCCS-VP and so underscores are inserted. Secondly, as defined by the given set [IDENT], in the Z specification of the semantic function, plus the fact that unique particle names are required in the formal model, all duplicate flows are uniquely named.

5.1.1 Event Splitter Flows

A number of scenarios occurred in the *Silly Mixer* which appeared solvable by unique naming. However, a further complication was that two (or more) flows were not simply duplicated, but also had the same source. This is a diverging flow and simple renaming altered the semantics of the formal model. The following Figure 5 shows two DFD extracts from the *Silly Mixer*, prior to the uncovering of this problem.

Figure 5: Diverging Event Flow Scenarios

5.1.2 Solution to Event Splitter - Scenario One

Ward and Mellor demonstrate four possible scenarios of flow convergence and divergence [19, page 43]. They do not apply these specifically to event flows, but one of the scenarios can be applied here. The WM convention and interpretation, and its proposed application to event flows are shown in Figure 6. Later work by Ward [18] does suggest the application to all flows but no semantic details are considered in depth.

Figure 6: Diverging Event Flow

The semantics of the WM diverging (data) flow can be applied to duplicate event flows with the same source, thus resulting in a diverging event flow. This means that at the instance of occurrence of a diverging event flow it is received at both destinations, and can be extended to give a diverging flow with more than two destinations. The flows are not semantically distinct, but in fact there is only the single event, but two recipients.

Uniquely naming the two *empty vat* flows in the first scenario, results in two distinct flows which no longer carry the semantic that they occur at exactly the same time, i.e. in the same transition. This behaviour is restored to the model by using *empty vat* as a condition to a control transform which, upon receipt issues the two uniquely named event flows to their correct destinations. The necessary amendments to the DFD are shown in Figure 7, along with the STD for the newly introduced control transform.

Figure 7: Solution to Duplicate Event Flow with same Environment Origin

5.1.3 Solution to Event Splitter - Scenario Two

As shown in Figure 5 the 'duplicated' flows originate from the functional primitive *Monitor Vat Levels*. An additional problem is encountered here, where the output of the flows representing *liquid A OK* cannot be syntactically modelled in WMZ. WM impose the following constraint on outputs from a functional primitive data transform -

"If there are two or more discrete outputs they must be alternatives and at most one may be produced by each operation of the transformation."

Ward and Mellor [19]

Galloway and O'Brien [6] model this via the following *PostCondition* schema, syntactically ensuring that post conditions in WMZ can contain at most one time discrete event output. Adopting the renaming convention gives two uniquely named event flows which cannot be modelled as output in the same post-condition. This scenario cannot be converted into WMZ format after unique naming. Without the WM model expressed as a semantically equivalent WMZ model the result of any translation would be incorrect.

<i>PostCondition</i>
<i>Time_Discrete_Data_Output</i> : <i>Optional_Flow</i>
<i>Time_Discrete_Event_Output</i> : <i>Optional_Flow</i>
<i>Time_Continuous_Outputs</i> : $\mathbb{F}$ <i>IDENT</i>
<i>Condition</i> : <i>BExpr</i>

The solution is essentially the same as that defined in scenario 1, making use of the fact that output actions from an STD occur in the same transition. A single event output from the functional primitive is used as input to an STD, similar to that in Figure 7, with only one state and one transition to itself. This transition has a condition representing its single input and has two output actions which model the diverging event flow.

5.1.4 Diverging Continuous Data Flow

The *Silly Mixer* did not exhibit diverging continuous data flow behaviour, but a larger model under construction at the same time did and this was used to propose a solution to the problem. The DFD extract shown in Figure 8, shows the initial situation of the external *Steam Measure* continually issuing a value to the four data transforms, illustrated by the duplicated data flow *v*. Here the diverging flow is one of data and not event, ruling out the use of an accompanying control transform and STD. The solution necessitates the use of a functional primitive data transform and the production of a minispec.

Figure 8: Diverging Continuous Data Flow

The minispec defines the behaviour to be the continuous receipt of a data flow and issue of four continuous output data flows, uniquely named and carrying the same value. The change to the DFD is shown in Figure 9, but the inclusion of the minispec is omitted as it is defined using the *Value Expression* syntax, see [8]. The behaviour defined is to receive a value on v and issue this value on the four uniquely named data flows $v1$, $v2$, $v3$ and $v4$.

Figure 9: Solution to Diverging Continuous Data Flow

Further solutions to these problems are being investigated. Improvements are needed because at present the solution does introduce the overhead of an increased state space if the control transform or functional primitive is non-permanent, i.e. it can be enabled/disabled.

5.2 Problems in obtaining the SCCS-VP Program

Once a Ward/Mellor model is in WMZ format it is input to the semantic function program. Presented here are a number of problems encountered in trying to produce the program's output i.e. an SCCS-VP program, for the *Silly Mixer*. These are not errors but practical limitations as a result of implementing non-determinism.

5.2.1 Top Level Agent Definition and the Action Permission Set

The top level agent represents the entire model and actions it may perform. It is defined as an action permission agent, incorporating the set which contains all the actions that the system may perform based on the mutual exclusion of time discrete inputs from the environment. The particles from which these actions may be composed are those that result from the translation of the flows on the context diagram. Mutual exclusion of the time discrete inputs ensures that no action is permissible which contains more than one time discrete input from the environment. The agent declaration for the whole model, given that the context diagram, CD is declared, takes the following form:

$$\text{Model} = (\text{CD}/\text{Translated\_CD\_flows}) \backslash \text{Actions}$$

Despite the ease with which the ML program produced this agent, the action permission set was too large for the CWB to accept as input. Simply omitting the set enabled the CWB to accept the formal model but lost mutual exclusion of the time discrete inputs from the environment.

The solution was an extension to the agent syntax of SCCS-VP (and thus SCCS), and to the CWB to handle this new agent. This is termed the *Mutual Exclusion Agent* and makes use of the new *Mutual Exclusion Operator*, *. The top level agent declared using this new operator looks like so

$$\begin{aligned} \text{Model} &= \text{CD}/\text{SCCS\_Particles} * \text{Time\_Discrete\_Inputs} \\ &\text{where } \text{Time\_Discrete\_Inputs} \subseteq \text{SCCS\_Particles} \end{aligned}$$

So the agent *Model* performs those actions composed of *SCCS_Particles* on the proviso that at most one of *Time_Discrete_Inputs* is a constituent particle.

5.2.2 SCCS-VP Re-expression of STDs

The SF re-expresses a state in a WM State Transition Diagram (STD) by summing all the agent expressions representing a single transition from said state. Each STD has a set of input flows (conditions), and there is an agent expression for every permutation of every subset of these flows. As the cardinality of this set increases the number of agent expressions increases exponentially. The production of agent declarations for states soon takes an inordinate amount of time.

This problem results from having to model each state as being capable of responding to any number of its inputs in any order. This non-determinism is easily expressed with the abstraction available in Z, but such abstract concepts are not so readily implemented. The current solution was to limit the extent to which a state could simultaneously respond to inputs.

Certain changes can be made to reduce the non-determinism that each state defines, thus reducing the size of the agents, without removing any observable behaviour:

- Input conditions to the STD which are time discrete from the environment are already restricted to being mutually exclusive in a transition. Therefore, an agent expression in the declaration of a state with an action which has more than one particle representing such an input is redundant.
- Of the 'other inputs' if two or more originate from the same functional primitive then at most one of these can participate in a single transition of the STD. This is due to the earlier pre-requisite that time discrete outputs from a primitive data transform are alternatives.

6 Conclusions and Future Work

The Methods Integration group at the University of Teesside, having previously specified a translation from WM SA/RT to SCCS-VP, now has a program which can take an instantiation of a WM model and produce an SCCS-VP program.

No tool exists to analyse/simulate SCCS-VP, a major desire of obtaining a formal specification of a system. The CWB will accept and simulate Basic SCCS and so it was necessary to write the appropriate conversion. A specific WM model can be formalised as an SCCS program and simulated on the CWB.

The whole process has helped manually verify the initial documents and the theory on which they are based, due to the intense scrutiny they received and debate that this provoked. Practical problems were discovered which can be termed limitations of the implementation as they require the implementation to dispense with some of the non-determinism so easily expressed in the abstract syntax of Z. It was found that re-expression of non-trivial STDs and their simulation on the CWB posed a practical limitation. This is not an error in the Semantic Function or its implementation, but a result of trying to implement the non-determinism.

Other issues, not covered in the original works have subsequently been resolved, a necessity when faced with a real problem and model. These are concerned with diverging data and event flows with a number of real scenarios encountered in the case study and their solutions presented and explained. Taking the form of guidelines/heuristics gleanable from the examples these can be used to obtain a WM model in a form which is behaviourally equivalent and acceptable to the translation, these same guidelines/heuristics are also amenable to CASE resolution.

A tool(set) has been successfully developed to support the integrated method between the Ward-Mellor SA/RT notation and SCCS-VP. This work required careful scrutiny of the WM semantics with respect to intended behaviour and how this was re-expressed formally. The Semantic Function did already exist but this work found other subtleties of ambiguous and incomplete WM behaviour which needed to be rectified to be formally represented. Ongoing work now involves removal of the sub steps towards full integration on a common platform. Work is also being undertaken to investigate how safety properties can be identified on a WM model and formulated into CWB propositions, the results of which in turn being traced back to the WM model. An area yet to be addressed (and automated) is the translation of event stores whose representation on a DFD is merely shorthand for behaviour expressed using existing DFD components.

7 Acknowledgements

Firstly, and mostly to Andy Galloway and other members of the *WM-SCCS* group at the University of Teesside. The former being the main author of the documents referenced earlier which define the WMZ, the Semantic Function and the SCCS-VP to Basic SCCS translation which form the basis for the work presented here. Also, for being available for interrogation on the finer points of all the above works.

To Glen Bruns for making freely available a CCS-VP to Basic CCS translator which went a large way to solving the SCCS-VP to SCCS problem.

References

- [1] G. Bruns. A Language and Translator for Value-passing CCS. Technical report, University of Edinburgh, January 1993. Preliminary Version.
- [2] T. DeMarco. *Structured Analysis and System Specification*. Prentice-Hall, Hemel Hempstead, 1978.
- [3] P. C. Fencott, A. J. Galloway, M. A. Lockyer, S. J. O'Brien, and S. Pearson. Formalising the Semantics of Ward-Mellor SA/RT Essential Models Using a Process Algebra. In *Proceedings of Formal Methods Europe 1994 (FME '94)*. Springer Verlag, To be published 1994.
- [4] A. J. Galloway. A Bruns-Style Definition for Value Passing SCCS. Technical Report TEES-SCM-2-94, University of Teesside, School of Computing and Mathematics, Southfield Road, Middlesbrough, Cleveland. TS1 3BY., September 1994. e-mail: A.Galloway@tees.ac.uk.
- [5] A. J. Galloway. Formalising the Semantics of Ward Mellor SA/RT Essential Models using Value Passing SCCS. Draft Copy TEES-SCM-5-94, University of Teesside, School of Computing and Mathematics, Southfield Road, Middlesbrough, Cleveland. TS1 3BY., October 1994. e-mail: A.Galloway@tees.ac.uk.
- [6] A. J. Galloway and S. J. O'Brien. A Formal Abstract Syntax for Pre/Post Condition Mini-specification Descriptions in Ward-Mellor SA/RT Essential Models. Technical Report TEES-SCM-3-94, University of Teesside, School of Computing and Mathematics, Southfield Road, Middlesbrough, Cleveland. TS1 3BY., August 1994. e-mail: wmccs@tees.ac.uk.
- [7] A. J. Galloway and S. J. O'Brien. A Formal Abstract Syntax for Ward-Mellor SA/RT Essential Models. Technical Report TEES-SCM-4-94, University of Teesside, School of Computing and Mathematics, Southfield Road, Middlesbrough, Cleveland. TS1 3BY., October 1994. e-mail: wmccs@tees.ac.uk.
- [8] S. Hooker. CASE Support for Methods Integration: Implementation of a translation from a structured to a formal notation in ML. Technical Report No. TBA, University of Teesside, School of Computing and Mathematics, Southfield Road, Middlesbrough, Cleveland. TS1 3BY., August 1995. e-mail: S.Hooker@tees.ac.uk.
- [9] M. A. Lockyer and G. Griffiths. ALSETT - Towards an Educational IPSE. *Software Engineering Journal*, June 1989.

- [10] R. Milner. *Communication and Concurrency*. Prentice-Hall, Hemel Hempstead, 1989.
- [11] F. Moller. The Edinburgh Concurrency Workbench (Version 6.1). Technical report, University of Edinburgh, 1992.
- [12] F. Polack. Integrating formal notations and systems analysis: Using entity relationship diagrams. *Software Engineering Journal*, 7(6):363–371, September 1992.
- [13] G. P. Randell. Translating data flow diagrams into Z (and vice versa). Technical Report 90019, Defence Research Agency, Malvern, 1990.
- [14] G. P. Randell. Improving the translation from data flow diagrams into z by incorporating the data dictionary. Technical Report 92004, Defence Research Agency, Malvern, 1992.
- [15] G. P. Randell. Translating data flow diagrams in z (and vice versa). In *Proceedings of the 5th annual Z user meeting*. Springer Verlag, December 1990.
- [16] L. T. Semmens and P. Allen. Formalising yourdon. In A. Bryant P. Allen and L. Semmens, editors, *Proceedings of the Methods Integration Workshop*,, Leeds, September 1992. Springer-Verlag.
- [17] J. M. Spivey. *The Z Notation : a reference manual*. Prentice Hall, Hemel Hempstead, 1989.
- [18] P. T. Ward. The transformation schema: An extension of the data w diagram to represent control and timing. *IEEE Transactions on Software Engineering*, SE-12(2):198–210, February 1986.
- [19] P. T. Ward and S. J. Mellor. *Structured Development for Real-Time Systems*, volume 1 : *Introduction & Tools of Computing Series*. Yourdon Press, 1985.
- [20] A. Wikstrom. *Functional Programming Using Standard ML*. Prentice Hall, 1987.