ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**D. J. Duke , University of York, UK and A.S. Evans, University of Bradford,
UK (Eds)**

2nd BCS-FACS Northern Formal Methods Workshop

Proceedings of the 2nd BCS-FACS Northern Formal Methods
Workshop, Ilkley, 14-15 July 1997

Generating Formal Specifications from CASE Repositories

B. Ryan

Springer

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

Published in collaboration with the
British Computer Society

Generating Formal Specifications from CASE Repositories

Benjamin Ryan

School of Computing, Leeds Metropolitan University

Beckett Park Campus, Leeds LS6 3QS

e-mail: b.ryan@lmu.ac.uk

August 25, 1997

Abstract

A method for formally specifying the generation of formal specifications from a CASE tools' repository is described. The limitations of the approach are identified and summarised. An alternative approach is outlined that addresses an important limitation, that of the original approach being specific to one software development method. The use of a neutral representation to generalise the approach to other software development methods is outlined. A mechanism is outlined that uses the original approach and the neutral representation to enable an incremental approach to the adoption of formal specification techniques. The incremental adoption is supported by the default generation of a formal specification that allows the inclusion of hand written formal specification fragments in a controlled way. The work builds on the established area of method integration by thoroughly investigating one approach to integration and suggesting ways that this could be improved. The techniques presented go some way to providing a framework in which the use of formal specifications techniques can be adopted in an incremental manner.

1 Introduction

Formal specification techniques have developed considerably over the past two decades from being the products of pure research to practical techniques that are used by the software development community to specify and design dependable and correct software systems. However, their penetration has been most successful in the area of safety critical and mission critical systems where software failure is unacceptable. Many of these uses have been forced upon developers by standardisation authorities who mandate the use of formally based techniques [3].

The use of formally based techniques has had less of an impact outside this area though this situation is changing slowly [4]. There have been many reasons proposed (and refuted) [12] as to why formally based techniques have enjoyed less success outside these limited areas.

An approach taken by the research community to address this problem is to integrate formally based techniques with other non-formal¹ methods. These approaches, called Method Integration, have concentrated on integrating the complementary aspects of the formal and non-formal to produce an improved method.

The benefits of combining non-formal methods and formal based techniques have been widely reported [12, 30, 23]. Within the literature there are numerous references to the integration disparate software development techniques to provide an integrated method. Semmens et al. give an overview in [30]. Since this overview was written other work on integrated methods has been published detailing attempts to integrate different types of development techniques and different ways of integrating them [18, 4]. Several authors have proposed different classifications for the approaches taken to Method Integration [9, 28, 19, 10, 15].

One approach to method integration, a generative approach, has been proposed by several authors [29, 8, 7, 14, 15]. This paper reports on an implementation of the generative approach. The implementation has been undertaken to determine the resources required to implement such an approach and to analyse the approach for weaknesses. The

¹the term non-formal is used to describe both structured and object oriented development methods which do not use a mathematical notation as the primary description mechanism

implementation has been targeted at an implementation of the Information Engineering (IE) Methodology [20, 21], the IPSYS Object IE 3.1 CASE tool.

An improved generative approach is proposed that address some of the negative aspects that have been identified. This improved approach generalises the generative approach by removing the bias towards one particular CASE implementation. The generalisation is gained by using a neutral representation for meta-model descriptions used by the generative approach. Using facilities of the neutral representation the improved approach also addresses a criticism of generative approaches that use of the formal notation directly by the practitioner is hindered.

The rest of this paper is organised as follows. Section 2 describes a generative approach to method integration detailing the four stages. Section 3 presents an analysis of the generative approach and details the resources required, the quality of the formal specification generated and a number of negative aspects of the approach. Section 4 outlines an improved generative approach that addresses certain criticisms of the original approach. Sections 5 and 6 summarise the conclusions and suggest areas for future work.

1.1 The Automated Teller Machine (ATM): An Example

In this paper an example will be used to give a context to the definitions given. This example is a simple Automated Teller Machine (ATM). The ATM system comprises of a number of accounts that are held by customers. There may be more than one customer per account and a customer may hold many accounts. The accounts are divided into two types; Current and Deposit. Current accounts have an overdraft limit and a weekly withdrawal limit, Deposit accounts have an interest rate. The interest is paid on the balance of the account at the end of each month. There are a number of ATMs in the system that may be used by customers who have been issued with cards and pin numbers for these cards. Transactions that customers may make at an ATM are withdraw cash, pay in cash, query a balance, change the PIN of a card and request a statement of previous transactions. The full example is given in [25]. The Entity Relationship Diagram for the ATM example is given in Figure 1.

2 A Generative Approach

The generative approach consists of four stages.

1. A description of the non-formal method's meta-model components, properties and their inter-relationship.
2. A definition of the rules for a well formed meta-model instance that ensures that the generation produces a valid formal specification.
3. A description of the abstract syntax of the target formal language that will be the receiving structures used by the mapping functions.
4. A definition of the functions which form the mapping between meta-model components and receiving structures.

A full treatment of these stages is given in [26].

2.1 Meta-model Descriptions

The initial stage of the formalism is to specify the components of the IE meta-model that are used in the mapping from the non-formal notations to the formal notations. This gives a description of the inputs to the mapping functions that will define the formal semantics of the data, process and behaviour models used in the IE method.

The specification is split into the static and dynamic components of the meta-model. The static components are the artefacts produced from the High Level Data Modelling and the Data Analysis phases of the IE method. The dynamic aspects are the artefacts produced by the High Level Functional Modelling and the Process Analysis phases.

For reasons of space only the meta-model descriptions for the static components Data Types, Attributes and Entities are given. Full descriptions of the other meta-model components are given in [26].

Figure 1: Entity Relationship Diagram for ATM System

2.2 Static Property Models

Within the IE meta-model there are components used to specify the data structures of interest in the system. These components specify the static properties of the system and serve to classify and name objects of interest. They use standard techniques such as Entity Relationship Modelling, Data Flows and, Data Stores.

2.2.1 Data Types

The type of an attribute is its *domain* and is recorded as a name. A domain may be a *PLAIN* domain where its name represents a type or an *ENUMERATED* domain where its name represents a type with a fixed set of values specified by the literals.

$$\text{DOMAINTYPE} ::= \text{PLAIN} \mid \text{ENUMERATED}$$

<i>Domain</i>
<i>name</i> : <i>NAME</i>
<i>type</i> : <i>DOMAINTYPE</i>
<i>literals</i> : $\mathbb{F}$ <i>NAME</i>
<i>name</i> $\notin$ <i>literals</i>
<i>type</i> = <i>PLAIN</i> $\Rightarrow$ <i>literals</i> = $\emptyset$
<i>type</i> = <i>ENUMERATED</i> $\Rightarrow$ <i>literals</i> $\neq \emptyset$

2.2.2 Attributes

Attributes are used to describe entities. An attribute definition consists of a unique name and a domain type that has been previously specified.

<i>Attribute</i>
<i>name</i> : <i>NAME</i>
<i>domain</i> : <i>Domain</i>
<i>name</i> $\neq$ <i>domain.name</i>

Instances of attributes which occur within entities have additional properties to plain attributes. To model these additional properties it is necessary to introduce definitions of the properties and their allowable values. These definitions will be reused in the definition of relationships.

An attribute may have a cardinality of one or many when occurring within an entity, this is to allow for *Repeating Groups* which will be removed in the Normalisation process.

CARDINALITY ::= *ONE* | *MANY*

An attribute may be optional or mandatory. If it is mandatory it must have a defined value in all instances of the entity type. Where an attribute is optional it may not be used as an identifying attribute.

OPTIONALITY ::= *OPTIONAL* | *MANDATORY*

An attribute can be an identifying attribute and it serves, possibly with other attributes to uniquely identify an entity instance.

IDENTIFYING ::= *YES* | *NO*

<i>AttrInst</i>
<i>Attribute</i>
<i>card</i> : <i>CARDINALITY</i>
<i>optionality</i> : <i>OPTIONALITY</i>
<i>identifying</i> : <i>IDENTIFYING</i>
<i>optionality</i> = <i>OPTIONAL</i> $\Rightarrow$ <i>identifying</i> = <i>NO</i>
<i>card</i> = <i>MANY</i> $\Rightarrow$ <i>identifying</i> = <i>NO</i>

2.2.3 Entities

Entities are any object that the business wishes to record information about. They have a unique name and may have attributes. The attributes are defined as a finite set of attribute instances and this set is partitioned into two subsets. There are those attributes which identify an entity and those that do not. For an attribute to be an identifying attribute it must not be optional, it must have cardinality one and have been defined as an identifying attribute.

Where an entity owns a relationship that is defined as *identifying* the identifying attributes of the entity that is the destination of the relationship will be used, in addition to the source entity's own identifying attributes, to fully identify the source entity. This may result in a progression through a hierarchy of entities and identifying relationships to fully define the set of identifying attributes for a particular entity.

An entity may also be classed as associative or as a subtype of an existing entity. Where it is an associative entity it is used to resolve many-to-many relations and does not require any attributes in addition to the identifying attributes of those entities that it associates.

ENTITYCLASS ::= *PLAIN* | *ASSOCIATIVE* | *SUBTYPE*

<i>Entity</i>
<i>name</i> : <i>NAME</i> <i>type</i> : <i>ENTITYCLASS</i> <i>supertype</i> : <i>NAME</i> <i>attributes</i> : $\mathbb{F}$ <i>AttrInst</i> <i>identifiers</i> : $\mathbb{F}$ <i>AttrInst</i> <i>nonIdentifiers</i> : $\mathbb{F}$ <i>AttrInst</i>
$\forall a1, a2 : \text{attributes} \mid a1 \neq a2 \bullet a1.name \neq a2.name$ $\langle \text{identifiers}, \text{nonIdentifiers} \rangle \text{ partition } \text{attributes}$ $name \neq supertype$ $type \neq ASSOCIATIVE \Rightarrow attributes \neq \emptyset$

2.2.4 Repository Definition

Assuming similar definitions for the other meta-model components a *Repository* can now be defined.

An Entity Model View (EMV) is the necessary information that an activity has available to it to perform its function. The EMV is made up of three components:

1. an Import View, *Import*, detailing inputs to an activity,
2. an Export View, *Export*, detailing the outputs produced by an activity and
3. an Access View, *Access*, detailing the components of the Data Dictionary affected by the activity.

Within each set of meta-model components there is a restriction that they be uniquely named.

Repository

$domains : \mathbb{F} \text{ Domain}$
 $attributes : \mathbb{F} \text{ Attribute}$
 $entities : \mathbb{F} \text{ Entity}$
 $relationships : \mathbb{F} \text{ Relationship}$
 $subjectAreas : \mathbb{F} \text{ SubjectArea}$
 $entityRelationshipDiagrams : \mathbb{F} \text{ Erd}$
 $dataFlows : \mathbb{F} \text{ DataFlow}$
 $dataStores : \mathbb{F} \text{ DataStore}$
 $externalAgents : \mathbb{F} \text{ ExternalAgent}$
 $entityModelViews : \mathbb{F} \text{ EntityModelView}$
 $events : \mathbb{F} \text{ Event}$

$\forall d1, d2 : domains \mid d1 \neq d2 \bullet d1.name \neq d2.name$
 $\forall a : attributes \bullet a.domain \in domains$
 $\forall e1, e2 : entities \mid e1 \neq e2 \bullet e1.name \neq e2.name$
 $\forall rel1, rel2 : relationships \mid rel1 \neq rel2 \bullet rel1.name \neq rel2.name$
 $\forall s1, s2 : subjectAreas \mid s1 \neq s2 \bullet s1.name \neq s2.name$
 $\forall erd1, erd2 : entityRelationshipDiagrams \mid erd1 \neq erd2 \bullet erd1.name \neq erd2.name$
 $\forall df1, df2 : dataFlows \mid df1 \neq df2 \bullet df1.name \neq df2.name$
 $\forall ds1, ds2 : dataStores \mid ds1 \neq ds2 \bullet ds1.name \neq ds2.name$
 $\forall ex1, ex2 : externalAgents \mid ex1 \neq ex2 \bullet ex1.name \neq ex2.name$
 $\forall emv1, emv2 : EntityModelView \mid emv1 \neq emv2 \bullet emv1.name \neq emv2.name$
 $\forall ev1, ev2 : events \mid ev1 \neq ev2 \bullet ev1.name \neq ev2.name$
 $\forall ent : entities; sup : Entity \mid ent.supertype = sup.name \bullet$
 $\#ent.attributes > \#sup.attributes$

$rep : Repository$

2.3 Target Language Specification

In a generative approach it is necessary to specify what elements of the specification language, in this case Z, are to be used. These definitions are based upon those used by Semmens and Allen in their work in formalising Yourdon [30] with extensions to include the differences in IE.

This description is necessary to allow the definition of mathematical functions which map meta-model components onto an abstract syntax of a formal language. When this definition is given in a formal manner the syntax and type checking facilities provided by the Z language can be used to ensure the function definition is syntactically correct and that each use of the functions conforms to its typing constraints. This rigorous specification and checking provides a sound basis for the later implementation of the mapping functions in an executable language.

The target language specification appears in the Appendix.

2.4 Mapping Functions

These functions map components from the formal description of the meta-model to the receiving structures specified for the target language. They are specified as partial functions which have their domain specified by set comprehension to ensure that they are well-formed. This is necessary as the functions use meta-model components in their definitions to which the function may not be applied. As an example the function *GPlainEntSch* below restricts the domain to the entities found in the data dictionary which have class PLAIN and have at least one attribute.

All functions have the basic form of $funcName : metaModelComponent \rightarrow targetLanguageComponent$. All functions operate on components of a repository.

| $rep : Repository$

It is not possible to detail all the functions for the IE meta-model for reasons of space. The full definitions are given in [26]. For the purposes of explanation the functions relating to Domains, Attributes and Entities are given.

It should be noted that functions used within the definitions are of two types; those ending with the suffix **ASF** (Abstract Syntax Function) are the injective functions from the abstract syntax definition of the target formal language. Those beginning with a capital **G** (Generation) are mapping functions for meta-model components.

As an aid to comprehension the result of each function is shown below the function. These results are taken from case study produced to evaluate the process of generating a formal specification.

2.4.1 Domains

For each domain within the dictionary a *Given Set* is declared if the domain is not specified as an enumerated type. If it is enumerated a *Free Type* definition will be used.

$GPlainDomain : Domain \rightarrow TYPE$
$dom\ GPlainDomain = \{ dom : rep.domains \mid dom.literals = \emptyset \}$
$GPlainDomain = (\lambda d : Domain \bullet GivenASF(d.name))$

Accounts in the ATM system have a unique identifying number. The structure of the number is not important at this point.

[ACCOUNTNO]

$GEnumDomain : Domain \rightarrow TYPE$
$dom\ GEnumDomain = \{ dom : rep.domains \mid dom.literals \neq \emptyset \}$
$GEnumDomain = (\lambda d : Domain \bullet FreeTypeASF(d.name, d.literals))$

Cards within the ATM system have an enumerated status variable representing the current state of the Card.

$CARDSTATUS ::=$
 $ISSUED \mid INUSE \mid PINVALIDATED \mid REVOKED$

$GDomain : Domain \rightarrow TYPE$
$GDomain = GPlainDomain \cup GEnumDomain$

2.4.2 Attributes

Attributes are defined as components of a schema for an entity. Where they are used in specifying the information that a process requires they will be declared as components of an Entity Model View Schema.

GPlainAttrSigel Attributes are declared as declarations in the signature of a schema.

$GPlainAttrSigel : AttrInst \rightarrow SIGEL$
$dom\ GPlainAttrSigel = \{ ai : AttrInst \mid ai.card = ONE \}$
$GPlainAttrSigel = (\lambda ai : AttrInst \bullet$ $BscDeclASF(\{ ai.name \}, TypeASF(GDomain(ai.domain))))$

Assuming entity Account that has an attribute accountNo.

$accountNo : ACCOUNTNO$

GSetAttrSigel where the attribute has a cardinality of many it will be declared as set valued variable.

$$\begin{array}{|l}
 \hline
 GSetAttrSigel : AttrInst \rightarrow SIGEL \\
 \hline
 \text{dom } GSetAttrSigel = \{ ai : AttrInst \mid ai.card = MANY \} \\
 GSetAttrSigel = (\lambda ai : AttrInst \bullet \\
 \quad BscDeclASF(\{ ai.name \}, TypeASF(FinsetASF(GDomain(ai.domain))))))
 \end{array}$$

correspondenceAddress : $\mathbb{F}$ ADDRESSLINE

$$\begin{array}{|l}
 \hline
 GAttrSigel : AttrInst \rightarrow SIGEL \\
 \hline
 GAttrSigel = GPlainAttrSigel \cup GSetAttrSigel
 \end{array}$$

$$\begin{array}{|l}
 \hline
 GAttrInstSigel : \mathbb{F} AttrInst \rightarrow \mathbb{F} SIGEL \\
 \hline
 GAttrInstSigel = (\lambda ais : \mathbb{F} AttrInst \bullet \{ a : ais \bullet GAttrSigel(a) \})
 \end{array}$$

2.4.3 Entities

For the plain entities a schema is declared using the entity name as the schema name. Each of the attributes is declared as a variable in the signature of the schema. Where an entity is a subtype it will include the schema of its parent in its signature.

GPlainEntSch generates a schema for a plain entity. The entity name is used as the schema name and each attribute of the entity is declared as a component of the schema.

$$\begin{array}{|l}
 \hline
 GPlainEntSch : Entity \rightarrow SCHEMA \\
 \hline
 \text{dom } GPlainEntSch = \{ e : rep.entities \mid e.type = PLAIN \wedge e.attributes \neq \emptyset \} \\
 \forall e : \text{dom } GPlainEntSch \bullet \exists s : SCHEMA \bullet GPlainEntSch(e) = s \wedge \\
 \quad s.name = SchNameASF(e.name) \wedge \\
 \quad s.sig = GAttrInstSigel(e.attributes) \wedge s.pred = TRUE
 \end{array}$$

Assuming the entity Account which has three attributes accountNo, correspondenceAddress and balance.

$$\begin{array}{|l}
 \hline
 Account \\
 \hline
 accountNo : ACCOUNTNO \\
 correspondenceAddress : \mathbb{F} ADDRESSLINE \\
 balance : MONEY
 \end{array}$$

GAssocEntSig where an entity is an associative entity it may not have any attributes. In this case the identifying attributes of the entities that it associates will be declared as components of the schema signature. If the associative entity has attributes these will be included as for a plain entity.

$GAssocEntSig : Entity \leftrightarrow SIGNATURE$
$\text{dom } GAssocEntSig = \{ e : rep.entities \mid e.type = ASSOCIATIVE \}$ $\forall e : \text{dom } GAssocEntSig \bullet \exists sig : SIGNATURE \bullet$ $sig = GAssocEntSig(e) \wedge$ $sig = \{ ai : AttrInst; s : SIGEL; r : rep.relationships \mid$ $e = r.source \wedge$ $ai \in r.dest.identifiers \wedge$ $s = GPlainAttrSigel(ai) \bullet s \}$ $\cup$ $\{ ai : AttrInst; s : SIGEL \mid$ $e.attributes \neq \emptyset \wedge$ $s = GPlainAttrSigel(ai) \bullet s \}$

Assuming the associative entity Holding that has an attribute signatory to indicate whether the Holding has authorization for the account. The Holding also has the identifying attributes of the customer and account that it associates.

signatory : *BOOL*
customerNo : *CUSTOMERNO*
accountNo : *ACCOUNTNO*

GAssocEntSch the schema for an associative entity will have the signature defined by *GAssocEntSig* and an empty predicate.

$GAssocEntSch : Entity \leftrightarrow SCHEMA$
$\text{dom } GAssocEntSch = \{ e : rep.entities \mid e.type = ASSOCIATIVE \}$ $\forall e : \text{dom } GAssocEntSch \bullet \exists s : SCHEMA \bullet GAssocEntSch(e) = s \wedge$ $s.name = SchNameASF(e.name) \wedge$ $s.sig = GAssocEntSig(e) \wedge$ $s.pred = TRUE$

Assuming associative entity Holding.

<i>Holding</i>
<i>signatory</i> : <i>BOOL</i> <i>customerNo</i> : <i>CUSTOMERNO</i> <i>accountNo</i> : <i>ACCOUNTNO</i>

GSupTypeSigEl where an entity is a subtype it inherits all of the attributes of its supertype. This is modelled by including the schema for the parent in the signature of the subtype.

$GSupTypeSigEl : Entity \leftrightarrow SIGEL$
$\text{dom } GSupTypeSigEl = \{ e : rep.entities \mid e.type = SUBTYPE \}$ $\forall e : rep.entities \bullet$ $GSupTypeSigEl(e) = InclASF(SchNameASF(e.supertype))$

GSubTypeSch generates a schema for a subtype entity. The entity name is used as the schema name and each attribute of the entity is declared as a component of the schema. This schema only introduces the subtype entity and does not define the relationship between subtypes and supertypes.

$GSubTypeSch : Entity \leftrightarrow SCHEMA$
$\text{dom } GSubTypeSch = \{ e : rep.entities \mid e.type = SUBTYPE \}$ $\forall e : \text{dom } GSubTypeSch \bullet \exists s : SCHEMA \bullet GSubTypeSch(e) = s \wedge$ $s.name = SchNameASF(e.name) \wedge$ $s.sig = GAttrrsSigel(e.attributes) \cup \{ GSupTypeSigEl(e) \} \wedge$ $s.pred = TRUE$

Assuming entity Account and the subtype Current Account.

<i>CurrentAccount</i>
<i>Account</i>
<i>overdraftLimit</i> : MONEY
<i>weeklyLimit</i> : MONEY
<i>cumulativeWithdrawls</i> : MONEY

$GEntSch : Entity \leftrightarrow SCHEMA$
$\text{dom } GEntSch = rep.entities$ $GEntSch = GPlainEntSch \cup GAssocEntSch \cup GSubTypeSch$

3 An Analysis of a Generative Approach

The process of implementing a generative approach to methods integration has been described and examples from each of stage of the process have been given. The purpose of implementing a generative approach is to provide the practitioner with a system for automatically generating formal specifications from the repository information that is defined for a non-formal method.

As with any approach to integration there are positive and negative aspects that will have an impact on the usability of the approach. These aspects can grouped into the following areas: the expenditure of effort that is required to implement a generative approach, the formal specification generated and the limitations that the approach imposes on the generation process. These aspects will have an effect on the utility of the generative approach as a mechanism for introducing the use of formal description techniques into the software development process.

3.1 Expenditure of Effort

The implementation of the a generative approach required the formal specification of the four stages. This formal specification contains 145 formal paragraphs. The meta-model description uses 40 schemata and 15 axiomatic or abbreviation definitions to specify the information content of the IPSYS OBJECT IE repository. The mapping functions consists of 91 axiomatic definitions of partial functions. The complete specification of the four stages of the approach, not including the source code for the implementation of the translation functions, consists of an 80 page document. The time taken to produce the documentation and implementation was approximately 4 months.

For a method such as IE the amount of work that needs to undertaken to implement a generative approach may at first seem prohibitive. However, the need for a more rigorous approach to methodology development has been recognised by researchers [5, 16, 15, 17]. The approaches that have been taken are to use formal techniques to specify the information content of a method and its associated semantics.

A related issue is that of customisation of established methods to suit particular projects and in-house needs. Hardy et. al. [13] found that 88 % of departments surveyed customise their method to some degree. 62 % of the respondents based this customisation on previous experience of tailoring methods. This is a concern as most methods have been developed to be internally consistent and rigorous, though normally not through the use of mathematical techniques, and there is a possibility of losing this consistency and rigour.

Researchers have understood the need for mathematical rigour in the development of methods and that most uses of methods are in a customised form. This need for rigour can be satisfied by the stages of the generative approach as the process involves a formal description of the information content of the method and a suite of rules which specify a semantically correct model in the method.

The benefits that can be accrued from taking this view are that the meta-model of the non-formal tool undergoes a comprehensive and rigorous specification of its content and meaning. Also there is good potential for highlighting omissions and contradictions that appear in the meta-model and generating a comprehensive understanding of the non-formal tool's representational capabilities and limitations.

Another important benefit that can be gained from this process is in relation to the well-formedness rules that are defined as part of the four stage approach. These rules specify valid specifications that can be the subject of transformation and can be used to uncover areas where the CASE tool implementing the method could be enhanced with extra enforcement and verification techniques.

An example of this is the use of Entity Relationship Diagrams (ERDs). In the IE tool it is possible for an entity to be a plain entity, an associative entity or an attributive entity and for an entity to be of a different type in different ERDs. This conflicts with the formal semantics as it not possible to decide which well-formedness rule should be applied to the entity. The well-formedness rule for a plain entity requires that the entity has at least one attribute in addition to those attributes which identify the entity. Conversely the well-formedness rule for associative entities does not require the entity to have any attributes as the entity is used to resolve a many-to-many relationship between two existing entities.

In defining sub/super-type hierarchies the IPSYS tool shows implementation bias by insisting that subtype entities have a relationship between themselves and their super-type. This relationship between the subtype and supertype is used as the basis for two relational tables, one which holds the attribute values for the common attributes of the sub- and supertype and, one that holds the additional attribute values of the subtype and a foreign key for the supertype table.

One other area of concern is in the use of Activity Hierarchy Diagrams (AHDs) and Data Flow Diagrams (DFDs). The IPSYS tools allows activities on AHDs to be subordinate to themselves and processes in DFDs may appear on the decomposition of their own DFD. This is not reported by any checks that the tool offers except for a generic facility which reports where an object is used on a diagram.

3.2 The Generated Formal Specification

A generative approach has been used to translate the information content of a CASE repository into a formal specification. The success of the approach will depend heavily on the fitness for purpose of the formal specification. A case study has been produced using the IPSYS IE tool to populate the repository and provide input to the translation process. The formal specification generated is given in [25].

The term fitness for purpose will have different meanings depending on the use of the formal specification i.e. requirements specification, detailed design or documentation. In this work the aim of the generative approach is to automatically produce a detailed requirements specification that captured all of the information in the repository. The specification produced is incomplete in terms of constraints that may be required additionally to the constraints necessary to conform to IE semantics and predicates which define the effect of activities and processes in the normal manner of pre- and post-conditions.

Figure 2: Structure of the generated formal specification

For the purposes of this analysis the following criteria are used:

- structural clarity,
- traceability of information,
- verbosity and complexity,
- the use of language facilities.

While these criteria may not cover all the aspects of fitness for purpose they have been highlighted as significant in theoretical and practical work on method integration [10, 24, 2].

3.2.1 Structural Clarity

The structural clarity of the generated specification will have a significant impact on the readability of the specification and the separation of the three aspects of a system: data, behaviour and process. In Figure 2 the high level structure of the system specification is shown. Each aspect of the system is represented by one or more definitions (represented as rectangular boxes). The data aspect is represented by four definitions; entity, entity hierarchy, relationship and data model. The behaviour aspect is represented an entity life cycle definition. The process aspect by an activity definition.

Each of these definitions consists of a number of schema which capture one facet of the aspect. Taking the entity hierarchy definition as an example. The definition consists of a number of Subtype schema, Restriction schema, Partitioning schema and Supertype schema. The subtype schema describes the semantics of subtyping in terms of subsets of the ranges of the entity identification function. This injective function maps the ID of an entity to its instance and in the IPSYS IE tool all entities in a hierarchy have the same ID. The restriction schema provides a mathematical function to convert a subtype to a supertype using the Z concept of binding formation. The partitioning schema describes how the supertype instance set is partitioned by the subtypes. The supertype schema describes the instances of supertypes in terms the instances sets of subtypes.

These four schema together with the mechanism of schema inclusion used in the entity type schema to inherit attributes from the supertype provide a clear way to model the four characteristics of subtype instance sets.

- coverage
- exclusivity
- orthogonality
- inheritance

The structure of the other definitions in terms of schemata also separates the characteristics of the objects under consideration.

An advantage of this separation also becomes apparent when the formal specification is extended with constraints. The separation allows the practitioner to concentrate on a single aspect of an object at one time and to record any constraints that may be relevant to that aspect while allowing constraints that apply at a higher level to be defined separately. It should be noted that there is an obligation to ensure that no contradiction is introduced through these separate constraints, though this is true of a hand written formal specification in any case.

3.2.2 *Traceability*

Much of this structure, though not all, is derived from the structure of meta-model components that the formal specification components relate to. This promotes good traceability between the non-formal and formal specifications that is considered vital when changes are required to be made to either the non-formal or formal. Where the change is made from the non-formal the situation is eased by the generation process that in effect produces a new formal specification. However, it is likely that once the incomplete formal specification has been developed with additional constraints and activity definitions there will be a need to revisit the non-formal specification to make changes. The close structural correspondence will aid this task in identifying where changes are required.

3.2.3 *Verbosity and Complexity*

One of the criticisms (see [2] page 36) made against formal specifications that closely follow the structure of the non-formal is:

If the form of the Z is kept close to the informal analysis, it tends to result in a verbose and cluttered specification that lacks the elegance that is essential for understanding

While this true in principal it does not always apply when the formal specification is used as a documentation tool that has formal requirements for formatting and structure. Also within the scope of the work reported this well defined structure and explicit style of specification is an advantage when considering practitioners who are not *experts* in writing formal specifications.

3.2.4 *Use of Language Facilities*

Hall [11] has identified the problem of following the structure of the non-formal specification too closely in that it ignores the richer and more expressive facilities of most formal languages as compared to graphical representations. The formal specification generated does not use all the language facilities to the best advantage, though schema inclusion is used widely to structure the specification. Further the specification can be improved through the use of promotion and the schema calculus when the activity definitions are fully specified. This lack of use of language facilities will be addressed in the improved generative approach of Section 4.

The generated specification could be improved either by using a different representation in the formal language, as has been done subsequently by the author, or by choosing an alternative formal representation that is better suited to the task by having first objects and constructs which capture the semantics of techniques such as entity hierarchies and subtyping. However, the main objective of implementing a generative approach was to investigate the process itself and to attempt to improve and generalise the process while removing as many negative aspects as feasible.

3.3 Summary

Although the generative approach has been shown to be a viable way of producing a formal specification from the products of a non-formal method [1, 22, 14] there are important aspects that need to be considered when an implementation is to be attempted [27] :

- the approach requires an expenditure of effort on the four stages.
- the formal specifications produced can be verbose.
- certain meta-model constructs do not formalise well.
- the conceptual shift from the non-formal notation to the formal.

There are certain negative aspects of the generative approach that should be addressed to improve the approach:

- additional structuring mechanisms and conventions are required.
- there is loss of information in the generation process.
- the meta-model description is targeted to one particular methodology.
- the practitioner may want to hand-write parts of the formal text.

The next section will describe an improvement of the generative approach that addresses these negative aspects. The improvements to the approach are summarised below:

- the practitioner can choose when to use formal language directly and so not lose the expressive power of the language.
- any part of the non-formal specification not explicitly given a formal text can have a default representation.
- the structure of the non-formal specification is maintained.
- the generation process is generalised to a wider spectrum of non-formal methods.
- different formal representations of the meta-model constructs may be defined for use in different projects and within different stages within a project.

4 Improving the Generative Approach

In the generative approach the formal description of the meta-model is tied to one particular implementation of a methodology. An improvement of the generative approach would remove this bias towards a particular meta-model and allow the approach to be generalised to a wider spectrum of methods (see [18] Page 335).

The generative approach requires the formal specification of the input to the mapping functions. Any approach to removing the bias still requires the meta-model information to be well defined. An alternative means of representing meta-model information is needed that is not tied to one particular method and is capable of representing the information requirements of a wide spectrum of methods.

The use of a representation that meets this need allows the techniques of a generative approach to be applied to more than one non-formal method without having to re-write the meta-model description for each method. The more general representation will be formally defined once and once only. This allows any vendor of a CASE tool for a non-formal method to export the semantic information from their tool's repository to the unbiased representation. The formal specification can then be generated from this representation.

4.1 The CASE Data Interchange Format

The CASE Data Interchange Format (CDIF) [6] collection of standards is an approach to providing a comprehensive framework to address the needs of CASE tool users and vendors in transferring semantic information between CASE tools. The standards define a single interface to enable the import and export of information together with the semantics and presentation of this information. There are three major sections to the CDIF standards: the underlying architecture of CDIF standards and the mechanism for extending the meta-models; the transfer syntax and format which provides the import/export interface; and the collection of standards which form the Integrated Meta-model. The CDIF Integrated Meta-model is divided into sub-sections called subject areas. Each subject area addresses either a particular aspect of the development life cycle, such as entity-relationship modelling, or it addresses some aspect of commonality used by other subject areas.

These subject areas define meta-models that are capable of representing the semantic information for a wide variety of techniques and methods. These meta-models will be used to replace the method specific meta-models used in the original generative approach.

4.2 Using the Formal Notation Directly - The Incremental Adoption of Formal Specification Techniques

An improvement would allow the use of the formal specification notation directly by the practitioner. This would allow the richer and more expressive formal notation to be used where the software specification can benefit most from rigour and precision.

In a simple case this could be where the practitioner has written the formal specification of an object rather than use the default translation. In a more complex case the practitioner may have recognised that a group of entities and relationships could be better represented directly in the formal notation using the richer structures that are available e.g. sequences or bags.

If the practitioner is to hand-write fragments of the formal specification these must be included with the generated parts in an orderly manner. This will be further discussed in Section 4.3.3.

The application of a generative approach where the practitioner can incorporate custom written fragments can be viewed as a starting point at which practitioners adopt the use of formal specification techniques. The generation of the formal specification allows the practitioner to gradually take over the process of writing formal specifications though ensures that the structure of the specification reflects the structure of the informal specification. As the practitioner gains experience in the use of the formal notation the amount of the specification that is automatically generated should reduce.

It should be noted that an incremental approach must be supported by education, training and expert advice. The automatic generation can support the adoption of formal techniques by alleviating the burden of producing a complete specification and by imposing a good structure on the formal specification but it cannot replace the need for a thorough understanding of the problem domain and specification techniques.

4.3 Managing the Use of Hand Written Fragments

In the improved generative approach the CDIF Integrated Meta-model is used to capture the semantic information from the non-formal tools repository that is then used as the input to the generation of the formal specification. This allows for the automatic production of the formal specification as an initial approach. As the practitioner adopts the direct use of the formal specification language it is necessary to ensure that the traceability from non-formal to formal is maintained. The CDIF General Structuring Mechanism is used to provide a traceability mechanism and the inclusion of hand-written specification fragments.

4.3.1 The CDIF General Structuring Mechanism

Figure 3: CDIF General Structuring Mechanism

The CDIF suite of standards use a mechanism to structure the decomposition of definitions into components that may be reused and shared in other definitions. This mechanism, the General Structuring Mechanism (GSM), uses a number of CDIF meta-entities and meta-relationships to achieve this structuring. Figure 3 gives the meta-model for the objects used. The two major meta-entities are the *ComponentObject* and the *DefinitionObject*.

The *ComponentObject* uses a meta-relationship *References* to link components to their definitions. The *DefinitionObject* has two meta-attributes, *SpecificationLanguage* and *SpecificationText*, which capture the specification of the *DefinitionObject* and define the language used for the specification. The *DefinitionObject* also has a meta-relationship *Contains* that links the object to its structural decomposition of components. The *ComponentObject* has two subtypes *ReferencedElement* and *EquivalenceSet* that provide a mechanism for accessing distinct instances of components that are shared in definitions and allow for identical instances of components to be identified.

4.3.2 Use of the GSM

Figure 4: Using the General Structuring Mechanism

The GSM is used to provide a facility where formal specification fragments are linked to the non-formal object that they represent. To achieve this four additional meta-entities and three meta-relationships are defined using the CDIF extensibility mechanism, see Figure 4. The meta-entity *ComponentObject* is extended with three subtypes, *Structure*, *Definition* and *Constraint*.

The *Structure* meta-entity is used to represent structural components. The *Definition* meta-entity is used to represent definitions of structural components. The *Constraint* meta-entity is used to represent constraints on structural components and their definitions. The *Constraint* meta-entity can be linked to a *Structure* or *Definition* meta-entity using the meta-relationship *Constraints*.

The third meta-entity *SpecificationObject* is a subtype of *DefinitionObject* that is used to define the specification of the object in the formal language using the meta-attributes *SpecificationLanguage* and *SpecificationText* inherited from *DefinitionObject*. The meta-entity *SemanticInformationObject*, which is the supertype of all CDIF meta-entities that carry semantic information, is extended with two new meta-relationships *IsFullySpecifiedBy* and *IsPartiallySpecifiedBy*. This allows any object that carries semantic information to be linked to the *SpecificationObject* that fully/partially defines its specification in the formal notation. Where an object has only been partially specified in the formal notation the default translation will still be performed on those components that are defined in the CDIF transfer.

It should be noted that the extensions to the GSM are only used when the formal specification of an object is decomposed into structural components. This may result from reuse of formal specification fragments. Where an object's formal specification is not decomposed it can be represented directly in the *SpecificationObject* which is linked to it via one of the meta-relationships *IsFullySpecifiedBy* or *IsPartiallySpecifiedBy*.

4.3.3 Using the Extended GSM

Figure 5: Instance Diagram showing the structure of instances of the new meta-entities

An example from the ATM subsystem will be used to demonstrate the use of the extended GSM.

In the ATM subsystem example the accounts that a customer holds are represented by three entities; Customer, Account and Holding and two relationships; OwnsAndOwnedBy and ParticipatesInAndHeldBy. Where a customer is a signatory for an account this is recorded as an attribute of the Holding entity. We may choose to use the formal specification language directly by rewriting the definition of customer holdings. This new definition consists of two new attributes and a formal definition of a Boolean function which returns True if a Customer is a signatory for an Account and False otherwise.

Figure 5 shows the structure of the instances in the CDIF transfer. The formal specification generated from this CDIF transfer is given next.

Generated Z Specification

<p><i>Customer</i></p> <p><i>customerNo</i> : CUSTOMERNO</p> <p><i>name</i> : NAME</p> <p><i>address</i> : TEXT</p> <p><i>holdings</i> : $\mathbb{F}$ Account</p>
--

<p><i>customers</i> : $\mathbb{F}$ Customer</p>
--

<i>Account</i> <i>accountNo</i> : <i>ACCOUNTNO</i> <i>correspondenceAddress</i> : $\mathbb{F}$ <i>ADDRESS</i> <i>balance</i> : <i>MONEY</i> <i>signatories</i> : $\mathbb{F}$ <i>NAME</i>
<i>accounts</i> : $\mathbb{F}$ <i>Account</i>
<i>signatory</i> : <i>Customer</i> $\times$ <i>Account</i> $\rightarrow$ <i>Boolean</i> $\forall c : \textit{Customer}; a : \textit{Account} \mid (c, a) \in \text{dom } \textit{signatory} \bullet$ <i>signatory</i> = $(\lambda c : \textit{Customer}; a : \textit{Account} \bullet$ if <i>c.name</i> $\in$ <i>a.signatories</i> then <i>True</i> else <i>False</i>) $\text{dom } \textit{signatory} = \{ c : \textit{customers}; a : \textit{accounts} \}$

The generated formal specification includes the hand written fragments within the schemata for *Customer* and *Account*. The definition of the function *signatory* is presented as an axiomatic definition. If the specification object is named then it will be presented as a schema. Other forms of formal representation can be defined by extending the specification object with an attribute describing the type of specification object or by subtype classification.

5 Summary and Conclusions

A method for formally specifying the generation of formal specifications from a CASE tools' repository, as defined by a meta-model has been described. Although this approach has been shown to be effective there are a number of limitations that this approach imposes in terms of the effort required in implementing the approach and the restrictions imposed by the inflexibility of the approach.

An alternative approach has been outlined that addresses an important limitation, that of the original approach being method specific. This allows the approach to be generalised to other methods through the use of a neutral representation and by viewing the generation of a formal specification as a transfer between two CASE environments. The neutral representation, the CDIF Integrated Meta-model, is a published suite of standards that defines an integrated meta-model that addresses the information requirements of various aspects and stages of the software development process. The standards also define an import/export format for transferring information between CASE implementations. The CDIF standards chosen to generalise the generative approach also provides a mechanism for including custom written formal specification fragments in a controlled way.

The work addresses an important aspect of the use of formal specification techniques, it's lack of wide penetration, in the general software development community by introducing techniques that allow their incremental adoption by practitioners.

6 Future Work

The approach will need to be extended to cover the other Subject Areas that have been published or are in production. The support for different specification languages and constructs should be improved by defining new meta-entities and meta-relationships to represent these languages and constructs. The software support that has currently been developed for the improved generative approach will need to be extended to other CASE implementations and platforms.

7 Appendix

Each of the items of interest has a name.

$[NAME]$.

Decorations may be applied to schema names.

$DECORATION ::= dash \mid in \mid out$

Where names are used as schema names
there are functions defined that
translate names into schema references.

$SCHEMA_REF ::= SchNameASF \langle \langle NAME \rangle \rangle$
 $\mid SchDecorASF \langle \langle NAME \times DECORATION \rangle \rangle$
 $\mid DeltaASF \langle \langle SCHEMA_REF \rangle \rangle$
 $\mid XiASF \langle \langle SCHEMA_REF \rangle \rangle$
 $\mid HidingASF \langle \langle SCHEMA_REF \times seq NAME \rangle \rangle$

The relations needed are

$REL ::= rel \mid pfun \mid pinj$

The types needed are Given Set ,
Power Set , Cross Product ,
Relation , Free Type and Schema.

$TYPE ::= GivenASF \langle \langle NAME \rangle \rangle$
 $\mid FinsetASF \langle \langle TYPE \rangle \rangle$
 $\mid CrossASF \langle \langle seq TYPE \rangle \rangle$
 $\mid RelationASF \langle \langle TYPE \times REL \times TYPE \rangle \rangle$
 $\mid SchASF \langle \langle SCHEMA_REF \rangle \rangle$
 $\mid FreeTypeASF \langle \langle NAME \times \mathbb{F} NAME \rangle \rangle$
 $\mid ConstructedTypeASF$
 $\langle \langle NAME \times (\mathbb{F} NAME \times EXPR) \rangle \rangle$

Signature elements are either
inclusions or basic declarations.

$SIGEL ::=$
 $InclASF \langle \langle SCHEMA_REF \rangle \rangle$
 $\mid BscDeclASF \langle \langle \mathbb{F} NAME \times EXPR \rangle \rangle$

Predicate elements form the basis of predicates.

$PREDEL ::= EqASF \langle \langle \mathbb{F} EXPR \rangle \rangle$
 $\mid SubSetASF \langle \langle EXPR \times EXPR \rangle \rangle$
 $\mid InSetASF \langle \langle EXPR \times EXPR \rangle \rangle$
 $\mid NotInSetASF \langle \langle EXPR \times EXPR \rangle \rangle$
 $\mid ForallASF \langle \langle SIGNATURE \times PREDICATE \times PREDICATE \rangle \rangle$
 $\mid ExistsASF \langle \langle SIGNATURE \times PREDICATE \times PREDICATE \rangle \rangle$
 $\mid SchRefPredASF \langle \langle SCHEMA_REF \rangle \rangle$

A number of expressions are used.

$EXPR ::= DomASF \langle \langle EXPR \rangle \rangle$
 $\mid ImASF \langle \langle EXPR \rangle \rangle$
 $\mid OccASF \langle \langle NAME \rangle \rangle$
 $\mid SelASF \langle \langle NAME \times NAME \rangle \rangle$
 $\mid AplASF \langle \langle EXPR \times EXPR \rangle \rangle$
 $\mid SetExpr1ASF \langle \langle \mathbb{F} EXPR \rangle \rangle$
 $\mid SetExpr2ASF \langle \langle SCHEMATEXT \times EXPR \rangle \rangle$
 $\mid MapExprASF \langle \langle EXPR \times EXPR \rangle \rangle$
 $\mid CmpExprASF \langle \langle \mathbb{F} EXPR \rangle \rangle$
 $\mid SchRefASF \langle \langle SCHEMA_REF \rangle \rangle$
 $\mid TypeASF \langle \langle TYPE \rangle \rangle$
 $\mid LamASF \langle \langle SCHEMATEXT \times EXPR \rangle \rangle$
 $\mid MuASF \langle \langle SCHEMATEXT \times EXPR \rangle \rangle$
 $\mid ThetaASF \langle \langle NAME \rangle \rangle$
 $\mid RanASF \langle \langle EXPR \rangle \rangle$
 $\mid IntersectASF \langle \langle \mathbb{F} EXPR \rangle \rangle$
 $\mid UnionASF \langle \langle \mathbb{F} EXPR \rangle \rangle$
 $\mid EmptySet$

Predicates are defined as sets
of predicate elements.

$PREDICATE ::= ElemSetASF \langle \langle \mathbb{F} PREDEL \rangle \rangle$
 $\mid ConjoinASF \langle \langle \mathbb{F} PREDICATE \rangle \rangle$
 $\mid DisjoinASF \langle \langle \mathbb{F} PREDICATE \rangle \rangle$
 $\mid TRUE$
 $\mid FALSE$

$DECLARATION ::=$
 $DeclASF \langle \langle \mathbb{F} SIGEL \rangle \rangle$
 $SCHEMATEXT ::=$
 $SchTxtASF \langle \langle DECLARATION \times PREDICATE \rangle \rangle$

A schema has a signature that can
contain a set of signature elements.

$SIGNATURE == \mathbb{F} SIGEL$

$SCHEMA$ $name : SCHEMA_REF$ $sig : SIGNATURE$ $pred : PREDICATE$

8 References

- [1] S. Aujla, A. Bryant, and L. T. Semmens. Rigorous Review Technique: Using formal notations within conventional development methods. *IEEE Journal on Selected Areas in Communications*, Feb 1994.
- [2] R. Barden, S. Stepney, and D. Cooper. *Z in Practice*. BCS Practitioner Series. Prentice Hall, 1994.
- [3] A. Bradley. Requirements for Defence Standard 00-55. Workshops in Computing, pages 93–94. Springer-Verlag, 1993.
- [4] A. Bryant and L. T. Semmens, editors. *Methods Integration Workshop, 1996*. eWiCs 1996. Springer-Verlag, 1996.
- [5] D. D’Souza and A. Wills. Catalysis - practical rigour and refinement. Technical report, ICON Computing, 1995.
- [6] CDIF Technical Committee Electronics Industries Association. CDIF : Case Data Interchange Format. Technical Report EIA/IS-106 to EIA/IS-111, EIA/PN-3070 to EIA/PN-3074, 1994.
- [7] 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. volume 873 of *Lecture Notes in Computer Science*, pages 681–702. Formal Methods Europe, Springer-Verlag, 1994.
- [8] R. B. France. Semantically extended data flow diagrams: A formal specification tool. *IEEE Transactions on Software Engineering*, 18(4):329–346, Apr 1992.
- [9] R. B. France and M. M. Larrondo-Petrie. A two-dimensional view of integrated formal and informal specification techniques. In J. P. Bowen and M. G. Hinchley, editors, *9th International Conference of Z Users*, number 967 in *Lecture Notes in Computer Science*. Springer-Verlag, 1995.
- [10] M. D. Fraser, K. Kumar, and V. K. Vaishnavi. Informal and formal requirements specification languages: Bridging the gap. *IEEE Transactions on Software Engineering*, 17(5):454–466, May 1991.
- [11] A. Hall. Formal methods in the development of an information system for air traffic control. *IEEE Software*, 66–76, March 1996.
- [12] J. A. Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, September 1990.
- [13] C. J. Hardy, J. Barrie Thompson, and H. M. Edwards. The use, limitations and customization of structured systems development methods in the United Kingdom. *Information and Software Technology*, 37(9):467–477, 1995.
- [14] S. Hooker, M. A. Lockyer, and P. C. Fencott. Case support for methods integration: Implementation of a translation from a structured to a formal notation. In A. Bryant and L. T. Semmens, editors, *Methods Integration Workshop, 1996*, eWiCs. Springer-Verlag, 1996.
- [15] H. Hubmann. *Formal Foundations for SSADM: An Approach Integrating the Formal and Pragmatic Worlds of Requirements Engineering*. PhD thesis, Faculty of Information, Technical University of Munchen, Jun 1994.
- [16] Yourdon Inc. *Yourdon Systems Method*. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.
- [17] R. B. Jackson, D W. Embley, and S N. Woodfield. Developing formal object oriented requirements specifications: A model, tool and technique. *Information Systems*, 20(4):273–289, 1995.
- [18] K. Kronlof, editor. *Method Integration: Concepts and Case Studies*. Software Based Systems. J. Wiley & Sons., 1993.
- [19] K. Kronlof, editor. *Method Integration: Concepts and Case Studies*, chapter 12, pages 323 – 340. Software Based Systems. J. Wiley & Sons., 1993.

- [20] J. Martin. *Information Engineering : Book One Introduction*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [21] J. Martin. *Information Engineering : Book Two Planning and Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [22] F. Polack, M. Whiston, and K. C. Mander. The SAZ project: Integrating SSADM and Z. volume 670 of *Lecture Notes in Computer Science*, pages 541–557. Formal Methods Europe, Springer-Verlag, 1993.
- [23] G. P. Randell. Translating data flow diagrams into Z (and vice versa). Report no. 90019, RSRE, Ministry of Defence, Malvern, Worcestershire, UK, October 1990.
- [24] M. Rawson and P. Allen. Synthesis - an integrated, object-oriented method and tool for requirements specification in z. In A. Bryant and L. T. Semmens, editors, *Methods Integration Workshop, 1996*, eWiCs. Springer-Verlag, 1996.
- [25] B. Ryan. A generative approach to methods integration: A case study. Technical report, School of Computing, Leeds Metropolitan University, Room 7, Queenswood House, Beckett Park Campus, Leeds LS6 3QS, 1996.
- [26] B. Ryan. A generative approach to methods integration using IPSYS IE 3.1 and Z. Technical report, School of Computing, Leeds Metropolitan University, Room 7, Queenswood House, Beckett Park Campus, Leeds LS6 3QS, 1996.
- [27] B. Ryan. Implementing a generative approach to methods integration. In S. Wrycza and J. Zupančič, editors, *Fifth International Conference on Information Systems Design ISD'96*, 1996. 24–26 September, University of Gdańsk, Poland.
- [28] L. T. Semmens. *Methods Integration: Rigorous Systems Specification Using Structured Analysis and Formal Notations*. PhD thesis, School of Computing, Leeds Metropolitan University, 1996.
- [29] L. T. Semmens and P. M. Allen. Using yourdon and z: An approach to formal specification. In J. Nicholls, editor, *Proceedings of the Fifth Z User Workshop*. Springer-Verlag, 1991. 17–18 December, 1990, Oxford, UK.
- [30] L. T. Semmens, R. B. France, and T. W. G. Docker. Integrated structured analysis and formal specification techniques. *The Computer Journal*, 35(6):600–610, December 1992.