

A FORMAL STUDY OF THE NATIONAL HEALTH SERVICE'S "COMMON BASIC SPECIFICATION"

Richard A. Bradford
Center for Software Reliability
Bedson Building
University of Newcastle upon Tyne
NE1 7RU

July 1996

Acknowledgments

The author would like to acknowledge the assistance and contributions of Peter Gorm Larsen and Henrik Voss of IFAD, Peter Nicklin of the IMG and Andrew Barrow a fellow student at Newcastle University, thank you all. For the whole duration of the project the author is indebted to John Fitzgerald for his time and effort, cheers John!

Rich.

History of the CBS & work to be undertaken

In 1987 the National Health Service (NHS) Corporate Data Administration sought and was granted - by the predecessor of the NHS Management Executive - authority to produce a business model of the NHS. It would start by developing a business activity model of all NHS activities. The NHS Data Model would be validated, extended and linked to the activity aspect of the picture using knowledge drawn from IT projects which the NHS units wished to undertake on their own account. These projects have become known as “Common Basic Specification (CBS) development projects”. [CBS92]

The CBS is a generic model describing the functions the NHS undertakes and the information required to carry them out. It was formulated from a refined knowledge gained from the many previous Health Service IT products and serves as the grounding for good designs for systems that are computer based and systems that are not

The fundamental concepts used in the CBS development programme were commonality and abstraction. It is important to note that here the context of commonality is meant as not only the replication of functionality across examples of a particular type of NHS organisation but the drawing on the similarities between apparent different tasks. The reader is directed to [CBS92] where an example of this is given, taking the activities of a hip replacement and decorating a ward and breaking them down to show the analogies between them. The conclusion drawn from this exercise were “These two activities involve very different skills and resources and yet - in the general sense - involve similar business processes.”

The CBS is built around the common business model philosophy of telling you what is done and not how it is done. It is the activities specific data that brings the general NHS model to life giving context and meaning to a particular model of an NHS

business. The same notions of commonality and abstraction apply to the information requirements of both tasks.

The CBS was reviewed in 1991 and results showed that, although a small number of senior managers in the NHS were quite enthusiastic, the majority of potential customers were unconvinced. Following these disappointing results an assessment board was set up to review the usefulness of the CBS in the number of projects already using the CBS. The Assessment Board's report was published on the 15th May 1995 and concludes that "... the work to date had been valuable and the existing team should be supported..." It recommended that "the building and models of health care should continue, and the results should be used to underpin a number of IMG initiatives."

As an extension to this work there has been much discussion of the use of formal techniques within the CBS, to subject the CBS model to a rigorous formal analysis and increase the confidence that the model reflects its intention. There are a variety of safety constraints within the NHS which the model hopes to satisfy. One such example of this is managing resources. The information system has been modelled to automatically manage the availability of resources, to prevent for example patients being transferred from hospital to hospital trying to locate a scanner. A rigorous formal analysis would enable the assertion to be made that the solution overcomes these problems.

The initial exploration of the formal analysis can take one of two possible models. Either replace the CBS model and specify a new formal model or to develop a formal description alongside the present model. To date there has been no attempt made at producing an actual formal version of the specification in a formal specification language such as VDM-SL or Z.

The work to be undertaken will be to produce a formal model of the CBS in VDM-SL. This includes the information requirements as well as the business activities. Due to the size and complexity of the model the work will be an initial "pilot" attempt to produce a generic representation which can be re-used to build up the full specification. The objective being to successfully model and validate a small section of the CBS using the

information data model that has been developed and validated and to make some assertions on the feasibility of producing the full model. This includes thoughts and consideration about modelling the business activities as well as the advantages and disadvantages of producing and validating the full model.

The CBS model

The CBS model consists of three volumes :

- Volume 1 - The business activity model
- Volume 2 - The data model dictionary
- Volume 3 - The data model diagrams

As its name suggests Volume 1 [CBSV1] contains the business activities descriptions. Volume 2 [CBSV2] contains the entity type descriptions and Volume 3 [CBSV3] gives the entity relationship diagrams.

The formal model of the CBS follows this structure. The business activities are to be modelled as operation specifications, the entity types model as the system state and a set of auxiliary type definitions with invariants and the entity relationship model as a set of auxiliary functions.

The following section provides some background on the CBS data model. Subsequent sections report on the formal development of the entity relationship model, the formal development of the entity types models, an insight as to the strategy of developing the formal model of the business activities and a discussion on scaling up the complete model.

The CBS data model

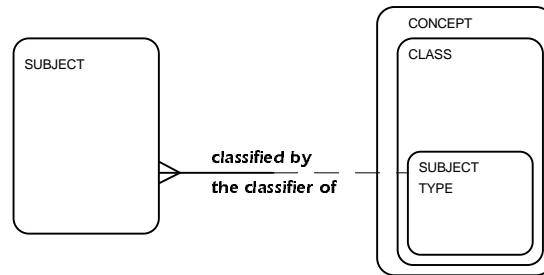
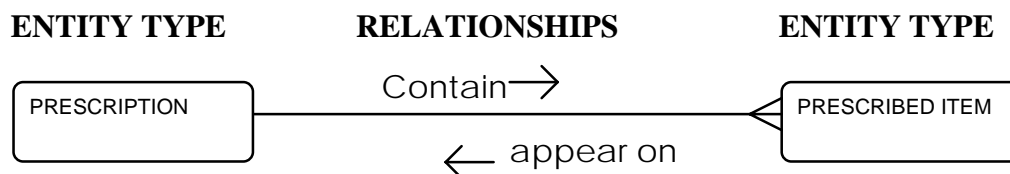


FIGURE 5 : EXAMPLE OF THE CBS E-R NOTATION

Figure 5 (from part of the diagram on page 26 of Vol. 3 of CBS) shows the notation used to represent the CBS **entity relationships (ER)**. It shows the data entity types **SUBJECT** and **CONCEPT** and the relationships between them. The syntax and semantics of this notation are given below. The formal challenge comes into splitting the two parts of the notation namely the entity types and the entity relationships and producing a generic model for each. Using generic reusable definitions of the model is of paramount importance for scaling up the model as well as being a good example of “sound software engineering.” These models can be used to build up a sample part of the, or indeed the full, CBS specification.

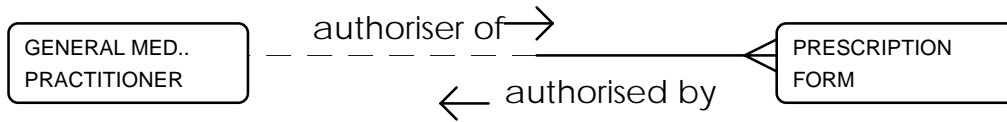
Entity relationship model

The CBS model is built around relationships between entity types. The CBS Vol. 3 gives the data model diagram convention. It is based on relationship templates and combinations of them which are given below.



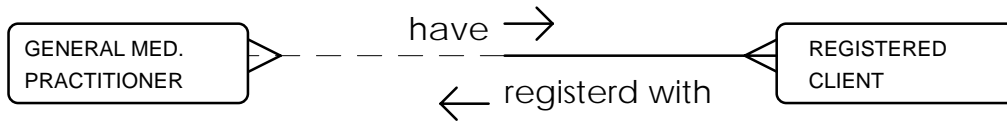
A prescription must contain one or more prescribed items

A prescribed item must appear on one prescription



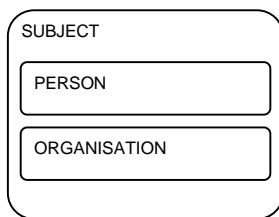
A GP may be the authoriser of one or more prescription form

A prescription form must be authorised by one GP

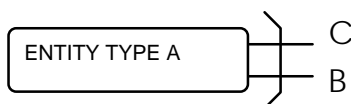


A GP may be the authoriser of one or more registered clients

A registered client must be registered with at least one GP



The entity types PERSON and ORGANISATION are sub-types of entity type SUBJECT



Relationships B and C that A has are mutually exclusive

All of the diagrams are built using the convention above in combinations. Note that in the first three conventions there are the following possible classes of relationship :

- Compulsory - Compulsory**

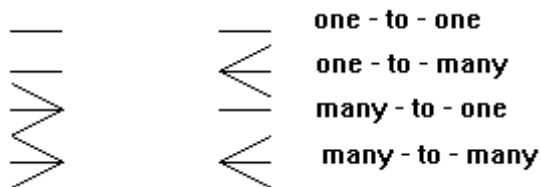
- Compulsory - Optional**

- Optional - Compulsory**

- Optional - Optional**

The meaning of the relationship is given by its direction. The Compulsory - Compulsory is interpreted as mandatory in both directions whereas Optional - Compulsory is optional in the left to right direction and compulsory in the right to left direction etc.

These relationship classes are combined with one of the “arities” :



The formal development will attempt to give a general interpretation of ERM's in VDM-SL reflecting this structure.

During the development it was decided that the sub-typing notation is more linked with the entity type model as opposed to the entity relationship model and therefore was not included in the ER model. The mutual exclusion notation will be covered in later sections.

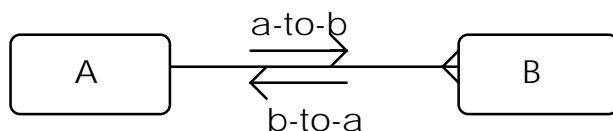
When developing the ER model a couple of different solutions were considered.

The first model was based on pairs of finite mappings specifying each direction of the relationship as a finite mapping between the two entity types tied together with a set of conditions presented in the invariant clauses.

Formal specification of entity relationships

As the following formal model was rejected in favour of the auxiliary function definitions , its syntax or type correctness were not checked using the Toolbox.

Consider the following generic 1 - M relationship :



The informal interpretation (as given in the CBS) of this model is :

“Every B must belong to exactly one A, Every A must own one or more B’s”

This ER model is represented in VDM-SL as a pair of finite mappings -

```
AB    :: a-to-b : map A to set of B
       b-to-a : map B to A
```

This must be further constrained by the following invariant clauses -

- For every B obtained from an A you must be able to get back to the A from the B

```
forall a in set dom a-to-b &
  forall b in set a-to-b(a) &
    b in set dom b-to-a and
    b-to-a(b) = a
```

- A must own one or more B’s

```
forall a in set dom a-to-b &
  card a-to-b(a) > 0
```

- Sets in the range of a-to-b are disjoint

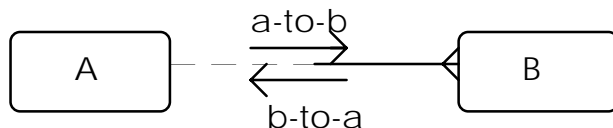
```
forall b1,b2 in set dom a-to-b &
  b1 <> b2 => b1 inter b2 = {}
```

Giving the model in full :

```
AB    :: a-to-b : map A to set of B
       b-to-a : map B to A
```

```
inv (ab) ==
  (forall a in set dom a-to-b &
    forall b in set a-to-b(a) &
      b in set dom b-to-a and
      b-to-a(b) = a) and
  (forall a in set dom a-to-b &
    card a-to-b(a) > 0) and
  (forall b1,b2 in set dom a-to-b &
    b1 <> b2 => b1 inter b2 = {})
```

Consider the following generic 1 - M relationship



The informal interpretation (as given in the CBS) of this model is :

“Every B must belong to exactly one A, Every A must own zero, one or more B’s”

This ER model is represented in VDM-SL as a pair of finite mappings -

```
AB    :: a-to-b : map A to set of B
       b-to-a : map B to A
```

These must be further constrained by invariant clauses :

- For every B obtained from an A you must be able to get back to the A from the B

```
forall a in set dom a-to-b &
  forall b in set a-to-b(a) &
    b in set dom b-to-a and
    b-to-a(b) = a
```

- Sets in the range of a-to-b are disjoint

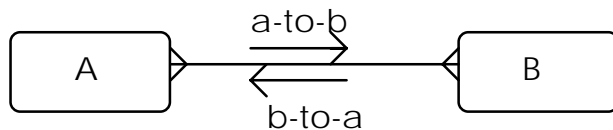
```
forall b1,b2 in set dom a-to-b &
  b1 <> b2 => b1 inter b2 = {}
```

Giving the model in full :

```
AB :: a-to-b : map A to set of B
     b-to-a : map B to A
```

```
inv (ab) ==
  (forall a in set dom a-to-b &
    forall b in set a-to-b(a) &
      b in set dom b-to-a and
      b-to-a(b) = a) and
  (forall b1,b2 in set dom a-to-b &
    b1 <> b2 => b1 inter b2 = {})
```

Consider the following generic M - M relationship



The informal interpretation (as given in CBS) of this model is :

“Every B must belong to one or more A’s, Every A must own one or more B’s”

This ER model is represented in VDM-SL as a pair of finite mappings -

```
AB :: a-to-b : map A to set of B
     b-to-a : map B to set of A
```

These must be further constrained by invariant clauses :

- For every B obtained from an A you must be able to get back to the A from the B

```
forall a in set dom a-to-b &
  forall b in set a-to-b(a) &
    b in set dom b-to-a and
    b-to-a(b) = a
```

- A must own one or more B’s

```
forall a in set dom a-to-b &
  card a-to-b(a) >0
```

- B must belong to one or more A’s

```
forall b in set dom b-to-a &
  card b-to-a(b) >0
```

Giving the model in full :

```
AB    :: a-to-b : map A to set of B
      b-to-a : map B to set of A

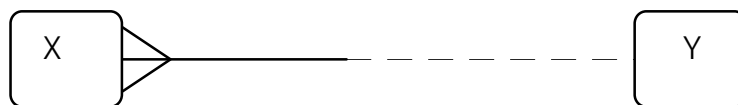
inv (ab) ==
  (forall a in set dom a-to-b &
   forall b in set a-to-b(a) &
   b in set dom b-to-a and
   b-to-a(b) = a) and
  (forall a in set dom a-to-b &
   card a-to-b(a) >0) and
  (forall b in set dom b-to-a &
   card b-to-a(b) >0)
```

If this model is followed then an exhaustive list of the combinations of the two mappings in each relation would be required.

The second, more compact, model was based on restricted binary relations representing the relation types :

- One to one compulsory
- One to one optional
- One to many compulsory
- One to many optional

Since an entity relation consists of two linked relations (one in each direction), using these four functions any combination could be achieved. The relationship would then be modelled as two sets of entities and a set of ordered pairs of entities from the Cartesian product of the entity sets. The functions would be called with the three sets to characterise the properties of the particular relation class.

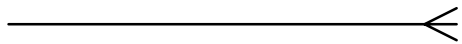


Could be represented as a single relation, one to one compulsory in the left to right direction and one to many optional in the right to left direction rather than separating into two mappings.

Formal specifications of functions

Each class of relation is represented by a function taking the two entity sets and a set of pairs of entities present in the relation. Note each of the relation classes represents one direction of the relation.

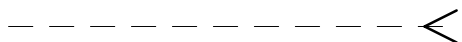
Consider the 1 - M compulsory relation



The diagram's formal semantics is given by :

```
OneManyComp : set of X * set of Y * set of (X*Y) -> bool
OneManyComp(xs,ys,rel) ==
  forall x in set xs &
    card {y | y in set ys
          & mk_(x,y) in set rel} >= 1;
```

Consider the 1 - M optional relation



The diagram's formal semantics is given by :

```
OneManyOpt : set of X * set of Y * set of (X*Y) -> bool
OneManyOpt(xs,ys,rel) == true;
```

(As the relationship allows there to be either none, one, or many pairs then nothing needs to be satisfied concerning the conditions of the relation in that direction.)

Consider the 1 - 1 compulsory relation



The diagram's formal semantics is given by :

```
OneOneComp : set of X * set of Y * set of (X*Y) -> bool
OneOneComp(xs,ys,rel) ==
  forall x in set xs &
    card {y | y in set ys &
          mk_(x,y) in set rel} = 1;
```

Consider the 1 - 1 optional relation



The diagram's formal semantics is given by :

```
OneOneOpt : set of X * set of Y * set of (X*Y) -> bool
OneOneOpt(xs,ys,rel) ==
  forall x in set xs &
    card {y | y in set ys &
          mk_(x,y) in set rel} <= 1;
```

The advantage of this approach are :

- a more compact specification due to the code required to specify another relation only requiring calling two of the functions;

- no repetition of invariant for each relation in the diagram as is required given the first model.

As the objective of the model was its inherent generic properties the second model was chosen as it better satisfied this informal requirement.

Development of the formal ER model

The definitions above use arbitrary X, Y entity types. To record this genericity formally in VDM-SL **polymorphic** definitions are used, permitting re-use across all relations.

The following formal definitions of the classes of relationship are taken from the previous function definitions and presented formally in the polymorphic form. Here the @e11, @e12 and @e13 are type variables.

```

OneManyComp[@e11,@e12] : set of @e11 * set of @e12 * set of
                        (@e11*@e12) -> bool
OneManyComp(xs,ys,rel) ==
    forall x in set xs &
        card {y | y in set ys &
              mk_(x,y) in set rel} >= 1;

OneManyOpt[@e11,@e12] : set of @e11 * set of @e12 * set of
                        (@e11*@e12) -> bool
OneManyOpt(xs,ys,rel) == true;

OneOneComp[@e11,@e12] : set of @e11 * set of @e12 * set of
                        (@e11*@e12) -> bool
OneOneComp(xs,ys,rel) ==
    forall x in set xs &
        card {y | y in set ys &
              mk_(x,y) in set rel} = 1;

OneOneOpt[@e11,@e12] : set of @e11 * set of @e12 * set of (@e11*@e12)
                        -> bool
OneOneOpt(xs,ys,rel) ==
    forall x in set xs &
        card {y | y in set ys &
              mk_(x,y) in set rel} <= 1;

```

During the validation of the polymorphic functions an interesting point was raised concerning the contents of the relation (the set of entity pairs). The function checked that the relevant conditions for a particular type of relation class held for a particular set of pairs of entities. The problem was that the function did not check for pairs which were invalid. Here “invalid” means a pair containing two entities from the same set or a

pair containing one or both entities from neither set. As the set of pairs would initially be empty and pairs would only be added through an operation specification which respects the relation's conditions then this it could be mathematically proved that this situation could never occur. As the model being specified was generic and to limit the relation so that it does not contain **junk** (where junk here is taken to be invalid pairs of entity types) it was decided to produce another function to check the validity of the set of pairs. This, as before, had to be polymorphic.

The Valid function was to take the two entity sets and the set of ordered pairs and check that each pair in the entity pair set contained an entity from the first entity set in the first position and an entity from the second entity set in the second position.

```
Valid[@e11,@e12] : set of @e11 * set of @e12 * set of (@e11*@e12)
                  -> bool
Valid(xs,ys,rel) ==
    forall mk_(x,y) in set rel &
        x in set xs and y in set ys;
```

It was at this stage that the mutual exclusion relation convention was modelled, also by a condition recorded as a polymorphic function.



This notation is informally interpreted (as given in CBS) that any entities participating in the A-C relation are not allowed to participate in the A-B relation. More formally the subset of A participating in the A-C relation and the subset of A participating in the A-B relation are disjoint. To check for this property it is necessary to extract the A entities from the set of A-C pairs set and the A entities from the A-B pairs set and then ensure that the two sets are disjoint.

Extracting the first half of a relation (or set of pairs) is analogous to the domain of a map. Therefore it was decided to specify a domain function Dom which could then be used by another function Mutex to check that the relations were mutually exclusive. Note the Dom function is not supplied in VDM-SL for relations other than finite mappings.

The functions Dom and Mutex are modelled formally as

```

Dom[@e11,@e12] : set of @e11 * set of @e12 * set of (@e11*@e12)
                -> set of (@e11)
Dom(rel) ==
  {t1 | t1 in set @e11 &
    exists t2 in set @e12 &
      mk_(t1,t2) in set rel};

Mutex[@e11,@e12,@e13] : set of @e11 * set of @e12 * set of @e13 *
  set of (@e11*@e12) * set of (@e11*@e13) -> bool
Mutex(xs,ys,zs,one,two) ==
  Dom[@e11,@e12,@e13](xs,ys,one) inter
  Dom[@e11,@e13](xs,zs,two) = {}

```

During validation of an example entity model it was noted that in order to use the polymorphic relation class functions to model the second half of the relation, the set of ordered entity tuples would have to be reversed. Therefore a polymorphic Inverse function was specified.

```

Inverse[@e11,@e12] : set of @e11 * set of @e12 * set of (@e11*@e12)
                  -> set of (@e12*@e11)
Inverse(xs,ys,rel) ==
  { mk_(y,x) | x in set xs,y in set ys &
    mk_(x,y) in set rel }

```

Having successfully validated these additional polymorphic functions (see validation Section) our first CBS objective had been successfully achieved. A generic model of the entity relationships had been specified. Now given the entity type model any CBS diagram can be formally modelled.

Entity Types model

The entity type descriptions are presented in volume 2 of the CBS :

1. The entity type name.
2. The name of the supertype of the entity type (if any).
3. A short textual definition of the entity type.
4. Following the heading "Comments", some notes and examples to aid in understanding the entity type and its purposes.
5. A list of the subtypes of the entity type, if any.
6. A list of the relationships involving the entity type. A "K" before the relationship shows that it is part of the primary key of the entity type, together with any similarly marked relationships or attribute types. An "O" before the relationship shows that the relationship is optional.

7. A list of the attribute types of the entity type, if any. Note that the relationships define what would be the attribute types required for the foreign keys, and so these are not included. A “K” preceding the attribute type name denotes the attribute type is part of the primary key of the entity type, together with any other similarly marked attribute types or relationships. An “O” before the attribute type name denotes that the attribute type is optional.

Consider the entity type SUBJECT TYPE (an example from CBS Vol. 2) :

SUBJECT TYPE

A classifier of a SUBJECT

Comments:

Examples would include ‘patient’, ‘client’, ‘medical employee’.

Subtypes of this entity type are :

ORGANISATION TYPE

INCIDENT TYPE

COMMUNICATION TYPE

Each SUBJECT TYPE

may be existing to invoke one or more SUBJECT RELATIONSHIP RULE SETS

may be existing to invoke one or more INTER RESOURCE LEVEL RULE SETS

may be existing to invoke one or more SERVICE SUBJECT RULE SETS

may be a subject type possessing the characteristics of one or more

CHARACTERISTICS FOR SUBJECT TYPES

may be the context for evaluation of one or more RESOURCE LEVEL RULES

may be the subject of one or more INTER RESOURCE LEVEL RULES

may be the classifier of one or more SUBJECTS

Attribute types of this entity type are:

	<u>Attribute Type Name</u>	<u>Domain name</u>	<u>Format</u>
K	SUBJECT TYPE IDENTIFIER	IDENTIFIER	CHAR

An important point which requires clarification from a CBS IMG representative was the sub-typing system employed in the CBS. The sub-typing is not compulsory, that is to say that a type may contain either one or none of its subtypes. Sub-typing can be thought of as a kind of inheritance although entity types can be instantiated and giving meaning through a relation as shown later in the chapter.

The entity being used for the developments of the model was the **SUBJECT** entity type. Sub-typing is used frequently within the CBS entity type model and it is for this reason that sub-typing is being included in this part of the model as opposed to the ER model. While the full SUBJECT entity type list including all of its subtypes was being generated it was observed that a number of types contained a unique key, namely an identifier for that data type. It was at this point that, with hindsight into modelling these data types, a couple of different models were to be considered.

Each model in turn would be provisionally developed, analysing its advantages and disadvantages when applied in the context of the system state.

Consider the SUBJECT entity type :

The first model made use of composite types with an identifier field where relevant.

The reader is advised to note that in order to clarify the primary key(s) of each type, signified by the K in brackets following the type definition, they have been included in the formal model but do not form part of the syntax of the model in VDM-SL and would subsequently be omitted. The “key” property would be identified in the invariant.

This model of the SUBJECT entity is presented as :

```
types

Increp :: reupdate : Date (K)
        reptime  : Time (K)
        descrip  : [seq of char]

Communication ::      incr : [Increp]
                    id   : seq of char (K)
                    text : [seq of char]
                    daterec : [Date]
                    datereq : [Date]

Incident ::      id : seq of char (K)
                dateocc : Date
                timeocc : [Time];

Organisation :: id : seq of char (K)

Person :: id : char (K)
         ethnicgroup : [seq of char]
         flagsb : [seq of char]
         dob : [Date]
         datedecnot : [Date]
         datedec : [Date]
```

```

sex : [seq of char]
countryob: [seq of char]

Subject ::      type : [Communication|Incident|Organisation|Person]
                subjectid : seq of char (K)
                description : [seq of char]
                dateintstart : Date
                dateend : [Date]
                timeintstart : [Time]

```

The optional attributes of the entity type can be identified from the above definitions by the VDM-SL “optional type” notation recorded as the attribute type enclosed in squared brackets (e.g. [seq of char]).

The sub-typing of the entity type can be identified from the above definitions by the VDM-SL “Union type” notation recorded as the sub-types separated by the vertical bar character. Due to the sub-typing convention they are also specified as the optional type (e.g. [Communication | Incident | Organisation | Person]).

When considering a subject database a condition, given in the invariant on the database, would have to be respected to allow this unique key property to hold.

The formal semantics of the subject database is :

```

Database = set of Subject
inv db == not exists p1,p2 in set db &
          p1.id = p2.id and p1 <> p2

```

Consider a “lookup function” which returns a subject given a subject identifier.

Recorded formally as :

```

lookup : Database * seq of char -> [Subject]
lookup(db,ident) == ???

```

The function would have to be implicit, i.e. specified by a **post** condition characterising the result.

An alternative model of the entity types uses finite mappings. Due to the inherent properties of maps, if the identifier was modelled to the subject details as a mapping then the uniqueness property would automatically hold.

An alternative model of this part of the SUBJECT entity type would be recorded formally as :

```

Increpdt :: date : Date
          time : Time

Increpdetails :: reptime : Time
               descrip  : [seq of char]

Increp :: map Increpdate to Increpdetails

Commid = seq of char

Commdetails :: incr : [Increp]
             text  : [seq of char]
             daterec : [Date]
             datereq : [Date]

Communication :: map Commid to Commdetails

Incidid = seq of char

Inciddetails :: dateocc : Date
              timeocc  : [Time]

Incident :: map Incidid to Inciddetails

Orgid = seq of char

Organisation = set of Orgid

Personid = seq of char

Persondetails :: ethnicgroup : [seq of char]
              flagsb       : [seq of char]
              dob          : [Date]
              datedecnot   : [Date]
              datedec      : [Date]
              sex          : [seq of char]
              countryob    : [seq of char]

Person :: map Personid to Persondetails

Subjectid = seq of char

Subjectdetails :: subid : [Commid | Incidid | Orgid | Personid]
               description : [seq of char]
               dateintstart : Date
               dateend      : [Date]
               timeintstart : [Time]

```

The sub-typing and optional type notations are modelled in the same way as described above. Since the domains of the finite mappings, usually identifiers, must be unique to respect the semantics of the VDM-SL mapping type the uniqueness property automatically holds and consequently the ‘K’s’ to identify the primary keys have been omitted from the above definitions.

Again considering a database type, which now would not need an invariant, and a lookup function presented formally as :

```

Database = map Subjectid to Subjectdetails

lookup : Database * seq of char -> [Subject]
lookup(db,ident) == if ident in set dom db

```

```
then db(s)
else nil
```

This model allows the use of an explicit lookup functions, in contrast to the set-based model.

The second model was preferred due to

- primary keys being implicitly specified;
- allowed use of explicit auxiliary functions

was chosen to represent the entity types model. The model was successfully validated using the Toolbox.

We now had achieved our objectives of developing a generic entity model (entity relationship model & entity type model) which followed the software engineering principles of re-use and could be subsequently scaled up to model any part of the CBS data model.

4.4 The CBS specification

A diagram was selected (see Figure 6) from page 23 of CBS Vol. 3 which was then modelled using the technique developed so far. Its formal representation is given in below.

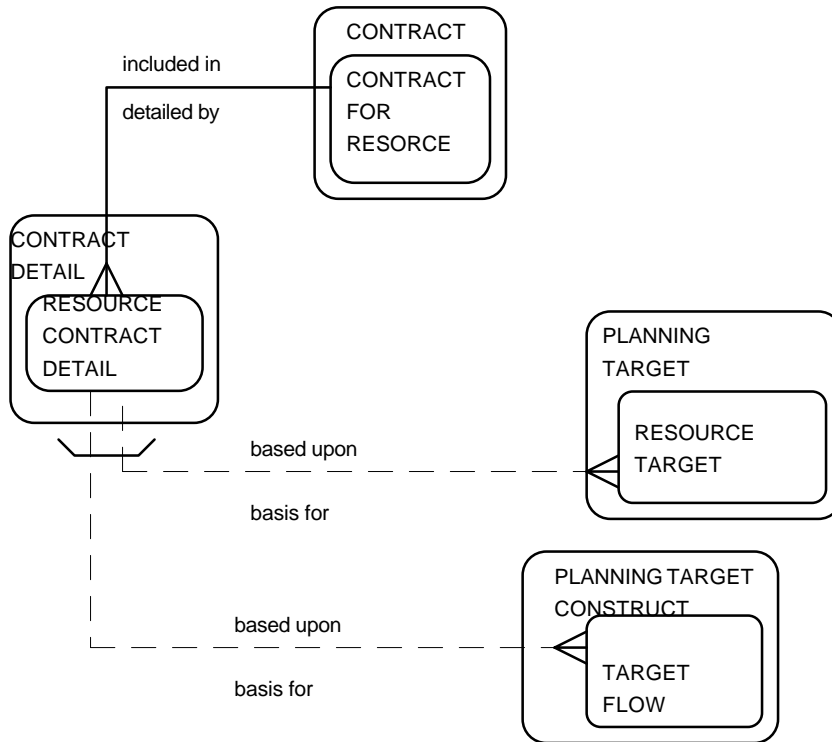


FIGURE 6 : PART OF CBS DIAGRAM

The specification was produced and successfully syntax and type checked. It is presented in full below. Note for validation purposes the entities were only represented in a simplified form composing if just of their supertype entity type.

CBS sample VDM-SL specification

```

types

Contractdetail = token;

Planningtargetconstruct = token;

Planningtarget = token;

Contract = token;

state1::Cdetail : set of Contractdetail
PTconstruct : set of Planningtargetconstruct
Ptraget : set of Planningtarget
Con : set of Contract
relcdc : set of (Contractdetail*Contract)
relcdptc : set of (Contractdetail*Planningtargetconstruct)
relcdpt : set of (Contractdetail*Planningtarget)

inv mk_state1(cds,ptcs,pts,cs,r1,r2,r3) ==

(Valid[Contractdetail,Contract](cds,cs,r1) and
OneOneComp[Contractdetail,Contract](cds,cs,r1)and
OneOneComp[Contract,Contractdetail]
(cs,cds,Inverse[Contractdetail,Contract](cds,cs,r1)))and

```

```

(Valid[Contractdetail,Planningtarget](cds,pts,r3) and
  OneManyOpt[Contractdetail,Planningtarget](cds,pts,r3)and
  OneOneOpt[Planningtarget, Contractdetail]

(pts,cds,Inverse[Contractdetail,Planningtarget](cds,pts,r3)))
and

(Valid[Contractdetail,Planningtargetconstruct](cds,ptcs,r2)
  and OneManyOpt[Contractdetail,Planningtargetconstruct]
  (cds,ptcs,r2)and
  OneOneOpt[Planningtargetconstruct, Contractdetail]

(ptcs,cds,Inverse[Contractdetail,Planningtargetconstruct]
  (cds,ptcs,r2))) and

(Mutex[Contractdetail,Planningtargetconstruct,Planningtarget]
  (cds,ptcs,pts,r2,r3))

functions

OneManyComp[@el1,@el2] : set of @el1 * set of @el2 *
  set of (@el1*@el2) -> bool
OneManyComp(xs,ys,rel) ==
  forall x in set xs &
    card {y | y in set ys & mk_(x,y) in set rel} >= 1;

OneManyOpt[@el1,@el2] : set of @el1 * set of @el2 *
  set of (@el1*@el2) -> bool
OneManyOpt(xs,ys,rel) == true;

OneOneComp[@el1,@el2] : set of @el1 * set of @el2 *
  set of (@el1*@el2) -> bool
OneOneComp(xs,ys,rel) ==
  forall x in set xs &
    card {y | y in set ys & mk_(x,y) in set rel} = 1;

OneOneOpt[@el1,@el2] : set of @el1 * set of @el2 *
  set of (@el1*@el2) -> bool
OneOneOpt(xs,ys,rel) ==
  forall x in set xs &
    card {y | y in set ys & mk_(x,y) in set rel} <= 1;

Valid[@el1,@el2] : set of @el1 * set of @el2 *
  set of (@el1*@el2) -> bool
Valid(xs,ys,rel) ==
  forall mk_(x,y) in set rel & x in set xs and y in set ys;

Dom[@el1,@el2] : set of @el1 * set of @el2 *set of (@el1*@el2)
  -> set of (@el1)
Dom(xs,ys,rel) ==
  {t1 | t1 in set xs & exists t2 in set ys &
    mk_(t1,t2) in set rel};

Mutex[@el1,@el2,@el3] : set of @el1 * set of @el2 * set of @el3 *
  set of (@el1*@el2) * set of (@el1*@el3) -> bool
Mutex(xs,ys,zs,one,two) ==
  Dom[@el1,@el2](xs,ys,one) inter Dom[@el1,@el3](xs,zs,two) = {};

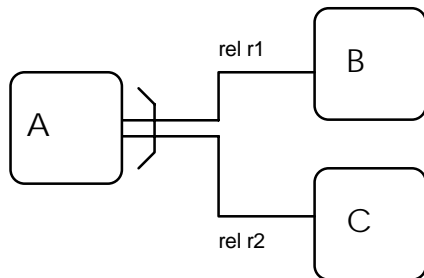
Inverse[@el1,@el2] : set of @el1 * set of @el2 * set of
  (@el1*@el2)
  -> set of (@el2*@el1)
Inverse(xs,ys,rel) ==
  { mk_(y,x)| x in set xs,y in set ys & mk_(x,y) in set rel}

```

During the validation of the formal specification of part of diagram an ambiguity in the CBS model was discovered. Where the mutual exclusion notation was used in

conjunction with pair of compulsory one to one relations, then the interpretation of the individual relationships is quite different. This ambiguity was not directly inherent in the part being modelled although during the validation this problem became problem became apparent.

Consider the following example :



The two interpretation of the diagram are :

1. The informal interpretation of the two entity relationships is -
“**Every A must** be related to a B **and Every A must** related to a C”
2. The informal interpretation of the relationships when taken in context with the mutual exclusion notation is -
“**Every A must** be related to a B **or** a C but not **both**”

Given these different informal interpretations, further investigation into the semantics of this notation would have to be carried out.

Through the exercise of modelling a significant part of the CBS the following points were raised in consideration of scaling up the model to produce the full CBS :

- since a generic CBS data model had been developed the scaling up of the CBS was expected to be a lengthy, although relatively simple, task;
- the auxiliary polymorphic functions interpreting the entity relationships would be better given as a separate VDM-SL module to further enhance ease of re-use.

The business activities model

As explained at the beginning of the chapter, the business activities are to be formally modelled as **operation specifications**. One of the components of an operation

specification is the **externals clause**. This contains an exhaustive list of all state components and access rights required for the operation specification to perform its function. These are specified as read only **rd** or read and write **rw**. This property is analogous to the “**Business Activity’s use of entity types**” specified by each business activity [CBSV1]. Here the **CRUD notation** is given. CRUD specifies the access rights required on the entity type by the business activities and are interpreted as :

- C - the entity type may be created by the business activity
- R - the entity type may be retrieved by the business activity
- U - the entity type may be updated by the business activity
- D - the entity type may be deleted by the business activity

The analogy is made between read access and the R of CRUD and read write access and the full range of access types of CRUD whereby an entity is modified by deleting the old one, creating a new one, and updating the new one. This important analogy suggests the appropriateness and suitability of the operation specification model being used to successfully specify the business activities.

Validation

To increase our confidence that the specification correctly models the behaviour of the system as we expect we have to formulate a validation plan. For CBS the validation strategy was to :

- Validate the entity relationships model
- Validate the entity types model
- Validate the simple CBS data model

Validation of entity relationships model

The validation of the entity relationships model involved the individual validation of each polymorphic function. As the Valid function is responsible for maintaining the consistency of the entity set pairs with regards to the entity sets, the proposed validation plan of the relation class functions ensures the elements in the entity sets and the elements in the sets of entity pairs are coherent. The functions were all successfully syntax and type checked. The types X, Y and Z have been defined as **token** within the

specification including the functions (see below). The token type giving numeric values were chosen for simplification reasons during validation.

The function **OneManyComp** representing the relation class 1 - M compulsory takes the two entity sets and the set of pairs of entities as arguments and return a Boolean result of true if the set of entity pairs respects the relation class and false if it does not.

Therefore the validation cases are :

- an empty set of entity pairs; expected result = true
- a non-empty set of entity pairs which respects the relation class;
expected result = true
- a non-empty set of entity pairs which does not respect the relation class'
expected result = false

The function **OneManyOpt** representing the relation class 1 - M optional takes the two entity sets and the set of pairs of entities as arguments. This function always returns true due to the properties of a one to many optional relation class as explained earlier in the chapter. Therefore the validation cases are :

- an empty set of entity pairs; expected result = true
- a non-empty set of entity pairs which respects the relation class;
expected result = true

The function **OneOneComp** representing the relation class 1 - 1 compulsory takes the two entity sets and the set of pairs of entities as arguments and return a Boolean result of true if the set of entity pairs respects the relation class and false if it does not.

Therefore the validation cases are :

- an empty set of entity pairs; expected result = true
- a non-empty set of entity pairs which respects the relation class;
expected result = true
- a non-empty set of entity pairs which does not respect the relation class;
expected result = false

The function **OneOneOpt** representing the relation class 1 - 1 optional takes the two entity sets and the set of pairs of entities as arguments and return a Boolean result of true if the set of entity pairs respects the relation class and false if it does not.

Therefore the validation cases are :

- an empty set of entity pairs; expected result = true
- a non-empty set of entity pairs which respects the relation class; expected result = true
- a non-empty set of entity pairs which does not respect the relation class; expected result = false

The function **Valid** checks the validity of the entity type vales in the set of entity value pairs. It takes the entity sets and the set of entity pairs and returns true if all the elements in the set of entity pairs respect their entity types and false otherwise.

Therefore the validation cases are :

- an entity pair which contains invalid entity types; expected result = false
- an entity pair which contains valid entity types; expected result = true

The function **Dom** extracts the first element out of all the first element of each tuple in the entity pair's set. It takes the entity sets and the set of entity pairs and returns the set of all the first element of each tuple in the entity pair's set.

Therefore the validation cases are :

- an empty set of entity pairs; expected result = { }
- a non-empty set of entity pairs; expected result = { 1,2 }

The function **Mutex** checks that every element of the entity set which is the domain of entity type of both relations is only present in one of the relations. It takes the three entity sets and the two relations as arguments and return true if this condition is respected and false otherwise.

Therefore the validation cases are :

- two relation which have first entity type elements in common; expected result = false
- two relation which don't have any first entity type elements in common;

expected result = true

The function **Inverse** reverses the ordered elements in each tuple in the set of entity pairs. It takes the two entity sets and the set of entity pairs as input and returns the re-ordered set of entity pairs.

Therefore the validation cases are :

- an empty set of entity pairs; expected result = { }
- a non-empty set of entity pairs; expected result = {mk_(5,1),mk_(6,2)}

The TOOLBOX function validation statements and actual results are present below. It was conclude that the polymorphic function had been validated successfully.

OneManyComp

```
vdm> print OneManyComp[X,Y]({ }, { }, { })  
true
```

```
vdm> print OneManyComp[X,Y]({1}, {2,3}, {mk_(1,2),mk_(1,3)})  
true
```

```
vdm> print OneManyComp[X,Y]({1,2}, {3,4}, {mk_(1,3),mk_(1,4)})  
false
```

OneManyOpt

```
vdm> print OneManyOpt[X,Y]({ }, { }, { })  
true
```

```
vdm> print OneManyOpt[X,Y]({1}, {2,3}, {mk_(1,2),mk_(1,3)})  
true
```

OneOneComp

```
vdm> print OneOneComp[X,Y]({ }, { }, { })  
true
```

```
vdm> print OneOneComp[X,Y]({1,2}, {3,4}, {mk_(1,3),mk_(2,4)})  
true
```

```
vdm> print OneOneComp[X,Y]({1,2}, {3,4}, {mk_(1,3),mk_(1,4)})  
false
```

OneOneOpt

```
vdm> print OneOneOpt[X,Y]({ }, { }, { })  
true
```

```
vdm> print OneOneOpt[X,Y]({1,2}, {3,4}, {mk_(1,3)})  
true
```

```
vdm> print OneOneOpt[X,Y]({1}, {2,3}, {mk_(1,2),mk_(1,3)})  
false
```

Valid

```
vdm> print Valid[X,Y]({1}, {3}, {mk_(2,4)})  
false
```

```
vdm> print Valid[X,Y]({1}, {3}, {mk_(1,3)})  
true
```

Dom

```
vdm> print Dom[X,Y]({1},{3},{})
{ }
```

```
vdm> print Dom[X,Y]({1,2,3,4},{5,6,7,8},{mk_(1,5),mk_(2,6)})
{ 1,2 }
```

Mutex

```
vdm> print Mutex[X,Y,X]({1},{2},{3},{mk_(1,2)},{mk_(1,3)})
false
```

```
vdm> print Mutex[X,Y,X]({1,4},{2},{3},{mk_(1,2)},{mk_(4,3)})
true
```

Inverse

```
vdm> print Inverse[X,Y]({},{},{})
{ }
```

```
vdm> print Inverse[X,Y]({1,2,3,4},{5,6,7,8},{mk_(1,5),mk_(2,6)})
{ mk_( 5,1 ),
  mk_( 6,2 ) }
```

CBS sample specification

```
vdm> print inv_statel(mk_statel({1},{2},{3},{4},{},{},{}))
false
```

```
vdm> print
inv_statel(mk_statel({1,2},{3},{4},{5,6},{mk_(1,5),mk_(2,6)},
{mk_(1,3),mk_(2,3)},{}))
false
```

```
vdm> print
inv_statel(mk_statel({1,2},{3},{4},{5,6},{mk_(1,5),mk_(2,6)},
{},{mk_(1,4),mk_(2,4)}))
false
```

```
vdm> print
inv_statel(mk_statel({1,2},{3},{4},{5,6},{mk_(1,5),mk_(2,6)},{},{mk_(
3,4)}))
false
```

```
vdm> print
inv_statel(mk_statel({1,2},{3},{4},{5,6},{mk_(1,5),mk_(2,6)},{mk_(1,3
)},{mk_(1,4)}))
false
```

```
vdm> print
inv_statel(mk_statel({1,2},{3},{4},{5,6},{mk_(1,5),mk_(2,6)},{mk_(2,3
)},{mk_(1,4)}))
true
```

Validation of entity types model

The validation of the entity types model involved the syntax and type checking of the specification. This was successfully completed.

Validation of sample CBS

The validation of the sample part of the CBS model firstly involved the syntax and type checking of the specification. This was completed successfully. The second part of the validation was to use the execution tool of the Toolbox. The model was validated by executing the state invariant.

The validation cases are :

- an entity pairs set breaking the Contractdetail - Contract relation class conditions; expected result = false;
- an entity pairs set breaking the Contractdetail - Planningtarget relation class conditions; expected result = false;
- an entity pairs set breaking the Contractdetail - Planningtargetcontract relation class conditions; expected result = false;
- an entity pairs set containing “junk”; expected result = false;
- an entity set breaking the mutual exclusion; result = false;
- valid sets of entity type and sets of entity pairs; expected result = true.

Conclusions

A generic formal model which could, in principle, be used to produce any part of the CBS data model was specified and validated. Although the model did utilise sound engineering techniques such as code reuse and polymorphism it could have been further improved by encapsulating the entity relationship functions in a module.

Due to the ambiguity presented in the CBS data model confirmation from the IMG must be sought to clarify precisely which, if necessary, changes need to be made to the formal data model.

Since a generic CBS data model had been developed the scaling up of the CBS is expected to be a lengthy, although relatively simple, task.

The VDM-SL language features of composite, union, mapping, Cartesian product and optional types were of great use in the CBS formal specification and of credit to the language. However it was clearly noted that generic domain and range operators should be provided over arbitrary Cartesian products, as opposed to solely finite mappings.

Again some kind of abstract data type or template would have been of great advantage in modelling the entity types. This would have helped further generically model the entity types. It was noted that in producing only a small sample part of a CBS entity diagram the type definitions were quite considerable in size.

Validating of the specification using the dynamic execution is again an inefficient technique for a specification of even moderate size. This especially comes to light within the CBS with the complexity of the entity types. The test coverage tool however, which was not used during the project, could have offered a solution to this problem.

If the entity relationships were further modelled in a separate module then this could be re-used since entity relationship approaches are widely used in software engineering.

The proposed approach to modelling business activities is to use operation specifications to represent business activities. This is particularly appropriate due to the externals clause component which will allow the CRUD attribute access system to be implicitly captured in the operation specification without the need for explicitly specifying this requirement.

During the development of the formal model of CBS, meetings and correspondence with IMG greatly increased the understanding of the information requirements of the system. Subsequently the formal model captured and better reflected these requirements thus with the hope of increasing the IMG's confidence in the formal specification.

REFERENCES

- [ISO95] D. J. Andrews, H. Brunn, B. S. Hansen, P. G. Larsen, N. Plat et al.,
Information Technology - Programming Languages, their environments
and system software interfaces - Vienna Development Method -
Specification Language Part 1 : Base language,
International Standards Organisation
Draft International Standard: 13817-1
1995
- [TBOX95] The VDM-SL Tool Group
The Institute of Applied Computing Science
User Manual for the IFAD VDM-SL Toolbox
October 95
- [IFAD94] The VDM-SL Tool Group
The Institute of Applied Computing Science
The IFAD VDM-SL Language(IFAD-VDM-1)
December 1994
- [CBS92] Information Management Group
The Common Basic Specification in 1992
Version 4.3
November 1992
- [CBSV1] Information Management Group
Common Basic Specification Generic Model
System Dictionary
Volume 1 - Business Activities
October 1992
- [CBSV2] Information Management Group
Common Basic Specification Generic Model
System Dictionary
Volume 2 - Data Dictionary
October 1992
- [CBSV3] Information Management Group
Common Basic Specification Generic Model
System Dictionary
Volume 3 - The Business and Data Model Diagrams, 2nd Edition
March 1994

