Translating Structured Database Schemas into Abstract Machines

Brian Matthews

Rutherford Appleton Laboratory,
Didcot, OXON, OX11 0QX, U.K.

Elvira Locuratolo*

Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche,
Via S. Maria, 46 - 56126 Pisa, Italy

October 29, 1998

Abstract

ASSO is a formal methodology for developing database applications based on B. Within ASSO, conceptual and logical descriptions of databases are linked through a formal relation to break down a database design into simpler components. We propose a systematic translation from ASSO schemas into B-machines establishing a formal relationships between them. This permits the formal semantics of ASSO to be explored using B, and also allows existing B tools to be used as a basis for the construction of ASSO tools.

1 Introduction

Formal methods such as B, Z and VDM have been developed to be applicable to a broad range of computing problems. However, there are advantages to be gained from methods specialised to particular application domains. The restricted nature of the systems to be developed may impose conditions on the methods, which can then be tailored to take advantage of the properties of that domain. Such specialised methods would also be more acceptable to developers accustomed to working within the culture of the application.

ASSO¹ is a formal methodology for developing quality database systems [7, 8, 9, 16]. ASSO combines the following properties:

- ease of specification of database applications;
- flexibility in reflecting modifications occurring in real life;
- correctness of implementations;
- efficiency in accessing and storing information.

ASSO has been developed by integrating aspects of the B Method [3] with the Partitioning Method [17], a formal method of database schema decompositions within the database area. This approach ensures that

---

* Supported by CNR-British Council agreement for the exchange of researchers.

¹ ASSO is the Italian term for the Ace in a suit of cards.

each ASSO schema is a particular Abstract Machine and that the formal semantics of the ASSO schemas can be given exploiting the pre-existing abstract machine semantics [4].

The direct use of B as a method for developing database applications has limitations. The B method lacks the abstraction mechanisms supported by the database conceptual languages, and its refinement has not been designed to obtain efficient database implementations. Within ASSO, both specifications and refinements are presented in abstract terms familiar to database developers. ASSO specifications are written using a formal notation which uses the two concepts of classification and specialisation within structured specifications. The formal semantics of the method exploit the nature of transactions, which preserve the integrity of the database, to simplify some B proof obligations. The Partitioning Method decomposes the database schema into equivalent schema and renders unnecessary further B proof obligations.

ASSO defines the novel notion of *structured database schema* which allows large database schemas to be specified in terms of smaller ones and as a consequence large consistency proofs to be decomposed. Previous work [19, 20] demonstrated that a structured model can also be usefully exploited to establish relationships between structured database schema and *Abstract Machines*, from the B Method [3]. In this paper, we generalise this relationship between ASSO and B to any conceptual schema supported by ASSO. The presentation here is given in terms of a formal translation scheme for translating from ASSO to B.

We show that a composition of B machines can be associated with the structured database schema. The relationship between structured database schema and abstract machine allows us to explore the semantics for ASSO in terms of the well-established weakest precondition semantics of B. B also provides an established proof theory for demonstrating the consistency and correctness of ASSO models. A further practical consequence of this relationship is that the well-developed tools which support the definition of abstract machines, such as the B-Toolkit, can be used as a basis for the construction of tools which support the definition of the database schemas and their consistency proofs.

This work can also be compared with [22] which also translates from formal schema descriptions into B. However, this latter approach translates all classes into a single flat specification, losing the benefits of B structuring, and also does not provide a treatment of inheritance, especially operation specialisation. This work can also be compared to various approaches to capturing object-oriented modelling in B, in for example [15, 14, 11, 12, 23].

In Section 2, the main features of ASSO are presented, with an example. In Section 3, the relationships between structured database schema and abstract machine are established, and the example is reconsidered in Section 4. Conclusions are given in Section 5.

2 Developing Databases using ASSO

ASSO is a formal database design methodology which makes it possible:

- to specify requirements with semantic data models;
- to specify static and behavioral aspects within the same formal framework;
- to obtain efficient implementations which satisfy the specifications given in the database schema.

ASSO is based on a model, called *database schema* [7, 8, 9, 17], which supports both the conceptual and the logical schema of database systems and is compatible with the B model of *abstract machine*. Each object instance of the *conceptual schema* can belong simultaneously to any class of a specialisation hierarchy, whereas, as in most object systems, each object instance of the *logical schema* belongs to one and only one class. Two phases define the ASSO methodology. The first, *conceptual design* consists of constructing a conceptual schema in which both structural and behavioural aspects of database applications are specified at a high level of abstraction. The second, called *refinement*, consists of a sequence of schema transformations. Refinement comprises the two phases of *behavioural refinement* and *data refinement*. The

behavioural refinement is a stepwise approach similar to B refinement that leaves the state unchanged while detailing the operations. First-order formulas are proved to guarantee correctness. The data refinement is an automatic process of schema decomposition, called *partitioning* that begins with a schema supported by an extended semantic model, systematically splits the domains of classes in the class hierarchy, and ends with disjoint classes supported by an efficient object model. The relationship between conceptual schema and object schema are the means to ensure both flexibility in reflecting modifications and efficiency of implementations.

The specification and proof of large databases using database schemas becomes complex. While the structural aspects of ASSO are defined by *specialisation* through the *is-a* constructor, a concept widely employed in the database area, the behavioural aspects are simply defined as state transformations on the whole state. To overcome this problem [2, 7], a notion of *behavioural specialisation* which allows both structural and behavioural aspects of database applications to be specified in a similar way, has been introduced in [4] through a new *is-a** relationship. This handles both attributes and operations in a homogeneous way whilst preserving the inherent constraints of the schema.

As a consequence of this notion, a structured model defined in terms of small database schemas, called *Structured Database Schemas*, can replace the monolithic database schema in all the phases of the ASSO methodology [4, 16]. Similarly to the database conceptual language, the operations of a structured database schema can be specified in order to preserve the *inherent constraints*. This allows the consistency proof of the conceptual schema to be decomposed into smaller proofs; the steps of behavioural refinement to be decomposed into parts and the data refinement to be carried out as a decomposition process of structured database schema.

ASSO uses a mathematical notation for logic, set theory and substitutions largely borrowed from B. Throughout this paper we assume familiarity with the notation and semantics of B.

2.1 Structured Database Schema

The *Structured Database Schema* approach supports both the conceptual and the logical schema of the ASSO methodology [16]. Within a structured database schema, classes of objects are organised into specialisation hierarchies through a relationship, denoted by *is-a**. This relationship extends the classic *is-a* relationship to behaviour [4].

Each *class* represents both static and dynamic aspects of a set of database objects. The static aspects encapsulate the set of *objects* and the set of *attributes* associated with those objects. The dynamic aspects encapsulate a set of state transformations, or *operations* on each class. Operations are defined recursively by applying pre-conditioned, partial and non-deterministic constructors to basic operations which insert objects, remove objects, change attributes or leave the class unchanged.

Classes can be defined through *is-a** relationships with other classes. In order to introduce the *is-a** relationship, we give the notions of *partial class* and *partial operation*. Informally speaking, a *partial class* is a class of a specialisation hierarchy without the *is-a** links. The top-level of a class is both a class and a partial class.

The ASSO notation uses the following two syntactic forms to specify the classes:

```
class variable of given-set with (attr-list; init; oper-list)
class variable is-a* variable with (attr-list; init; oper-list)
```

where *init* and *oper-list* denote the initialization and a list of operations on the specified class. Classes are constructed on top of *given sets*, which give unspecified non-empty collections of elements, as in [3], representing the domain of possible instance identifiers. Another syntactic form extending the second above is used to specify classes of multiple inheritance; this case is not covered in this paper and is thus omitted here.

Each partial class satisfies *inherent class constraints*, giving the typing for each class. For each class X , of given set S , with attributes $a_1:t_1, \dots, a_n:t_n$, these constraints are defined as:

1. $X \subseteq S$
2. $a_i \in X \rightarrow t_i$

Associated with each partial class are a set of four *base operations*, defined below for a general partial class X .

- **ADD** $X(x, v_1, \dots, v_n)$: Inserts a object x with attribute values $v_1, \dots, v_n$ into class X .
- **REM** $X(x)$: Removes a object x from class X .
- **SKIP** X : Does nothing in class X .
- **CHANGE** $X a_i(x, v_i)$: Updates the attribute a_i of object x in class X to v_i , if v_i is of type t_i .

These base operations preserve the inherent constraints for each partial class. All other operations on the class are defined in terms of these base operations by applying pre-conditioned, partial and non-deterministic constructors similar to generalised substitution within B. It is shown in [4] that this construction preserves the inherent constraints.

If the same partial class X stands in the relation $X \text{ is-}a^* Y$, and partial class Y has attributes $b_1:s_1 \dots b_m:s_m$, then the following properties hold:

- Class X inherits both the attributes and the operations from class Y .
- Objects of X are a subset of Y objects.
- Operations on X which insert objects are enriched with corresponding operations on the partial Y .
- Class X can have additional attributes and operations.

Thus the *is- a^** relationship imposes the following additional inherent constraints on the subclass:

3. $X \subseteq Y$
4. $(X.b_i) \in (X \rightarrow s_i)$

An *inherited operation* on X is the parallel composition of an operation on Y with the same operation specialised on the partial class X . Operations on X which insert objects must be enriched in order to satisfy the properties which define the *is- a^** relationship. Thus, $X \text{ is-}a^* Y$ exports to its environment operations which are compositions of the corresponding (by name) operations on X and Y , some of which are inherited and specialised by subclasses.

Note that there is an *implicit precondition* on the composition of classes; if the exported operation f is constructed by the parallel composition of $f.X$ and $f.Y$, then if f is applied to object $y \in Y$ such that $y \in Y \setminus X$, then $f.X$ has no effect and can be represented by *skip*, and f has the effect of $f.Y$ alone.

Each partial class has a distinguished initialisation operation *init* representing the initial state of the class. Initialisation operations are composed in parallel in the top-level schema.

Additionally, explicit constraints can be placed on partial classes and/or classes. If the explicit constraint involves two or more subclasses of a common class, a class defined as the intersection of these subclasses must be specified in the schema and the constraints must be associated with this class. However, such explicit constraints are not preserved automatically, and have to be shown to be preserved by the operations.

A structured database schema is consistent if and only if each class is consistent. However, if the application constraints involve only the partial classes, then the structured database schema can be defined as a set of partial classes linked through the *is- a^** relationship and then a structured database schema is consistent if and only if each partial class is consistent.

2.2 An Example of a Structured Database Schema

The concept of Structured Database Schema can be made clearer through the example, given in Figures 1 and 2. In this example, class *employee* is in *is-a* relationship* with the class *person*. Thus the following instances of the above properties hold:

```

structured database schema  company

class  person of  PERSON with  (
  attributes  income:N
  constraints  $\forall p (p \in person \Rightarrow income(p) \geq 1000)$ 
  operations
    init.person()  $\hat{=}$  person, income := {},{}
    new.person(pers, i)  $\hat{=}$ 
      PRE  pers  $\in$  PERSON  $\wedge$  pers  $\wedge$  i  $\geq$  1000
      THEN
        ADD person(pers, i)
      END ;
    delete.person(person)  $\hat{=}$ 
      PRE  pers  $\in$  PERSON
      THEN
        REM person(pers)
      END ;
    addto_income.person(pers, inc)  $\hat{=}$ 
      PRE  pers  $\in$  person
      THEN
        CHANGE person.income(pers, income(pers)+inc)
      END
  )

```

Figure 1: A Structured Database Schema

- Class *employee* inherits attributes and operations from class *person*.
- Objects of *employee* are a subset of *person* objects.
- Operations on *employee* which insert objects are explicitly specialised, whereas the other inherited operations are implicitly specialised.
- Class *employee* can have specific attributes and/or operations.

To expand on this, an *inherited operation* on *employee* is the composition of an operation on *person* with the same operation instantiated on the *employee* class. This instantiation is an implicit specialisation for most operations, such as *delete*. Operations on *employee* such as *new.employee* which insert objects are explicitly specialised to preserve the *is-a* constraints. *Application constraints* are part of the specified classes.

The *company* structured database schema specifies a database *company* with classes *person* and *employee*, the latter specification being a subclass of the former, with four operations exported to the external environment: *delete*, *addto_income*, *addto_salary* and *new*:

- *delete* (respectively *addto_income*) is an inherited operation implicitly specialised. It is specified by the parallel composition of the *delete.person*, with its specialisation *delete.employee* (respectively with *addto_income.employee*, which is equivalent to *skip*).
- *new* is an inherited operation explicitly specialised. It is specified by the parallel composition of the *new.person* operation with its explicit specialisation *new.employee*.

```

class employee is-a* person with (
  attributes    salary:N
  constraints  $\forall e (e \in \text{employee} \Rightarrow \text{salary}(e) \geq 500)$ 
  operations
    init.employee()  $\hat{=}$  employee, salary := {},{}
    new.employee(emp,s)  $\hat{=}$ 
      PRE     $s \geq 500$ 
      THEN
        ADD employee(emp, s)
      END ;
    addto_salary.employee(emp,inc)  $\hat{=}$ 
      PRE     $emp \in \text{employee}$ 
      THEN
        CHANGE employee.salary(emp,salary(emp)+inc)
      END
  )
  constraints
     $\forall e (e \in \text{employee} \Rightarrow (\text{income}(e) \geq \text{salary}(e)))$ 
end database schema.

```

Figure 2: Continuation of the Structured Database Schema in ASSO

- *addto_salary* is an operation specific to the *employee* class, which results from the parallel composition of the *skip.person* operation with the *addto_salary.employee*.

In each case, the operation must maintain the constraint between the classes. Note the implicit semantics of all exported operations is that if the operation is applied to an object $p \in \text{person}$, such that $p \notin \text{employee}$, then the operation only applies that component which applies to *person*.

To summarise, the Structured Database Schema *company* results from two specified classes: class *person* and class *employee*, each of them being a Database Schema. The specification of the *person* class is completely explicit, whereas the specification of the *employee* class is partially explicit, partially implicit. The explicit specification of the *employee* class defines the partial *employee* class.

In our example, the *company* schema is a consistent schema if both the *person* class and the *employee* class are consistent database schemas, and the explicit constraint is maintained. However, once the *person* class has been proved consistent, to prove the consistency of the *employee* class, it suffices to prove the consistency of the partial *employee* class since each operation on the *employee* class is defined by the parallel composition of an already proven operation, with an operation of the partial class *employee*.

3 Translating Schemas into B Machines

The above introduction to ASSO gives a brief description of ASSO formal database specification. In this section we interpret the semantics of ASSO in terms of B machines. This process clarifies some of the concepts of ASSO, exposing some of the features of ASSO approach. This translation also allows the use of the tools already developed for B for ASSO, especially for proof.

3.1 B

The B-Method [3] represents one of the most comprehensive formal methods currently being promoted as appropriate for commercial use. Jean-Raymond Abrial originated B at the Programming Research Group at Oxford University in the early 1980s, and subsequently at British Petroleum Research (BP) and DIGILOG.

In B, systems are defined as *Abstract Machines*, each of which models the desired behaviour of part of the state of the system using state transitions systems. The method defines the Abstract Machine Notation (AMN) which uses a notion of *generalised substitution* to represent state transformations. The B-method also has powerful structuring mechanisms which offer data encapsulation allowing modular design and development of systems. B's underlying semantics is grounded in weakest preconditions over untyped set theory and classical logic introduced by Dijkstra [10]; the type system is correspondingly weak, and the distinction between type-checking and proof is blurred.

The B-Toolkit [5], developed by BP and subsequently by B-Core UK Ltd focuses on rigorous/formal design by supporting refinement from abstract specification through to imperative code. Tools exist for supporting static analysis (type-checking), dynamic analysis (animation), design documentation, proof of refinement and code generation.

3.2 Translating the Structured Database Schema

In this section we generate a B representation of classes declared using a structured database schema and related using the *is-a** relation.

A Structured Database Schema is translated into AMN by associating with each partial class a *base* machine declaring the class variables and the base operations, and a *class* machine which uses base operations to provide operations on the partial class. The translations are described using “generic ASSO schemas” and “generic B machines” in a representative database specification.

Base Machines A *base machine* is constructed for each class, containing the variables of the class and its attributes, the inherent constraints on the class, and an AMN representation of the basic operations.

A base machine models a partial class as a set representing class instances, with total functions for attributes. For each class, we declare a set of basic operations, similar to those used in ASSO, which skip, add, remove and change the attributes. Base machines preserve the partial classes inherent constraints, defined as invariants on the machine, and the encapsulation principle of B machines ensures that only these operations can modify the class variables. Thus inherent constraints are always preserved. The proof obligations which are generated for each machine demonstrate that these invariants hold. These are trivial and can always be automatically discharged by the B-Toolkit.

Given the generic ASSO schema for a partial class X , on given set S , with attributes:

$$a_1 : t_1 \dots a_n : t_n,$$

initialisation

$$X, a_1, \dots, a_n := I_X, I_1 \dots I_n,$$

explicit constraint

$$P(x, a_1, \dots, a_n),$$

and operations

$$op1_X, \dots, opk_X,$$

the corresponding generic base machine is given in Figure 3. Only one update operation is shown for reasons of space; one is generated for each attribute.

Thus each partial class has its own class set, attributes, invariants, initialisation and operations. The given set S generates a stateless machine SS containing only the set SS .

```

MACHINE  $x\_base$ 
SEES  $SS$ 
SETS  $A1, \dots, An$ 
CONSTANTS  $Ix, Il, \dots, In$ 
PROPERTIES  $Ix \subseteq SS \wedge Il \in Ix \rightarrow A1 \wedge \dots \wedge In \in Ix \rightarrow An$ 
VARIABLES  $xx, a1, \dots, an$ 
INVARIANT  $xx \subseteq SS \wedge a1 \in xx \rightarrow A1 \wedge \dots \wedge an \in xx \rightarrow An$ 
INITIALISATION  $xx := Ix \parallel a1 := Il \parallel \dots \parallel an := In$ 
OPERATIONS
   $SKIP\_x \hat{=} skip;$ 
   $ADD\_x(vx, vl, \dots, vn) \hat{=}$ 
    PRE  $vx \in SS \wedge vx \notin xx \wedge vl \in A1 \wedge \dots \wedge vn \in An$ 
    THEN
       $xx, a1, \dots, an :=$ 
         $xx \cup \{vx\}, a1 \triangleleft \{vx \mapsto vl\}, \dots, an \triangleleft \{vx \mapsto vn\}$ 
    END;
   $REM\_x(vx) \hat{=}$ 
    PRE  $vx \in xx$ 
    THEN
       $xx, a1, \dots, an :=$ 
         $xx - \{vx\}, \{vx\} \triangleleft a1, \dots, \{vx\} \triangleleft an$ 
    END;
   $CHANGE\_x\_a1(vx, val) \hat{=}$ 
    PRE  $vx \in xx \wedge val \in A1$ 
    THEN  $a1 := a1 \triangleleft \{vx \mapsto val\}$ 
    END
END

```

Figure 3: Prototypical Base Machine

Class Machines Having generated a machine with base variables, operations and invariants for the class, we can then generate the machine representing the explicitly declared class, including the base machine. The operations of this machine are straightforwardly translated from the corresponding ASSO operations.

The generic class machine is outlined in Figure 4. Note that the base operations of the two classes are not exported outside this machine, and thus are not required to satisfy the explicit invariants on these machines. The B-Toolkit proves these machines satisfy the explicit invariants on these classes. The base machines are embedded using B's **INCLUDES** mechanism of AMN, which provides semi-hiding. This allows us to inspect the value of the base machine's variables, but only modify the variables of the system via the base operations, which are known to preserve the inherent constraints. Thus the mechanisms of B are used to reflect the desired behaviour of ASSO.

An significant difference between the B method and ASSO is shown by the treatment of *implicitly inherited* operations. In ASSO, inherited operations which are not redefined in the subclass are inherited and specialised in the *is-a** relationship. However, in B such operations have to be redefined *explicitly* in the B machine for the subclass. Such specialisation can be systematically generated by a syntactic transformation on the operations of the superclass, renaming all instances of the identifiers of the superclass (including those within identifiers) to that of the subclass. Thus we undertake the following transformation on operations op_Y in the superclass Y :

```

 $Y \rightarrow X;$ 
 $SKIP\_Y \rightarrow SKIP\_X$ 
 $REM\_Y \rightarrow REM\_X$ 
 $ADD\_Y \rightarrow SKIP\_X$ 
 $CHANGE\_Y\_b1 \rightarrow SKIP\_X$ 
 $op\_Y \rightarrow op\_X$ 

```

Figure 4: Prototypical Class Machine

Effectively, we only specialise explicitly into B operations those ASSO operations which delete objects as part of their action. Operations which insert are already explicitly specialised as a part of the definition of well formed structured database schemas in ASSO. Operations which modify attributes of the superclass are not well-defined in the subclass considered in isolation and can be regarded as *skip*. In the generic schema in Figure 4, the operations *opYI_X* ... *opYp_X* represent the explicitly inherited operations.

Thus the translation of the structured database schema into B cannot be a composition of independent translation of partial database schemas, but has to include a global analysis of the whole hierarchy to generate specialisations of inherited operations.

The Top-Level Machine To represent the whole of the structured database schema, including the *is-a** relation, a top-level machine is generated which exports the operations which are available to the environment. In this section, for brevity, we do not give the complete generic top-level machine, but concentrate on its more interested features.

The top-level machine **INCLUDES** the class machines of the partial classes. The *is-a** relation is defined as a constraint between the two classes. There is also a redefinition of some of the operations, modifying the composite operations, and possibly adding new ones of its own.

In B, the operations implicitly composed and exported by the ASSO structured database schema have to be given explicitly. The set of operations in $X \text{ is-}a^* Y$ are formed by the parallel composition of the corresponding operations in the two partial classes. However, in B this must also preserve the implicit and explicit constraints on the schema, by the proof obligation on operations ([3], section 4.7):

$$I \wedge P \Rightarrow [S]I$$

for an operation $op = \mathbf{PRE} P \mathbf{THEN} S \mathbf{END}$ in a machine with invariant I . A precondition which guarantees this is $P = [S]I$. This precondition may seem too strong and for presentation to the user it may be simplified by taking into account the invariant in the “before” state. However, if the use of this translation is for an analysis of ASSO invisible to the user, then this form suffices.

Thus, if operation op is defined in the structured database schema as the implicit parallel composition of partial operations, then the following operation is constructed in the top-level machine:

```

 $op(x, avs, bvs) =$ 
PRE
   $[op\_X(x, avs) \parallel op\_Y(x, bvs)] \text{ inv}_{(X,Y)} \wedge C_{(X,Y)}$ 
THEN
   $op\_X(x, avs) \parallel op\_Y(x, bvs)$ 
END

```

where:

$\text{inv}_{(X,Y)}$ are the implicit constraints on $X \text{ is-} a^* Y$ and

$C_{(X,Y)}$ are the explicit constraints on $X \text{ is-} a^* Y$

However, as noted above, the construction of external operations also has the property that if the object is outside the subclass, then the operation implicitly uses *skip* as the operation on the subclass. This is equivalent to defining the following operation on the schema.

```

op(x, bvs) =
  PRE
    [ op_Y(x, bvs) ] inv_{(X,Y)} ∧ C_{(X,Y)} ∧ x ∉ X
  THEN
    op_Y(x, bvs)
  END

```

If on the other hand, an operation is defined on the subclass only, then the equivalent operation on the superclass is effectively *skip* and we can generate the operation as in the following scheme:

```

op(x, avs) =
  PRE
    [ op_X(x, avs) ] inv_{(X,Y)} ∧ C_{(X,Y)}
  THEN
    op_X(x, avs)
  END

```

Certain parallel compositions of base operations are known to be inconsistent (such as using *CHANGE* and *REM* on the same object) and can be detected syntactically in advance (see [4] for more details).

THEORY <i>SchemaBaseTheory</i> IS $e \subseteq \text{dom}(f) \Rightarrow e - a \subseteq \text{dom}(a \triangleleft f) ;$ $a \subseteq \text{dom}(f) \Rightarrow a \subseteq \text{dom}(a < f) ;$ $a \subseteq \text{dom}(f) \Rightarrow f \in a \rightarrow b$ END

Figure 5: Set of user rules for proving inherent constraints

The inherent constraints on classes are preserved by the operations through the base operations and thus it is unnecessary to prove them. However, the B-Toolkit still generates such obligations, and the autoprover does not have sufficient power to prove them automatically. Thus user intervention is required with the interpreter. As such obligations for inherent constraints are always of the same form for every database schema, a common set of user defined rules can be developed to prove that inherent constraints are preserved. The set of rules in Figure 5, together with the built-in rules of the autoprover, are sufficient to prove that the inherent constraints of a database schema, including those of a *is-a** relation, are preserved.

Clearly, obligations will also be generated for explicit constraints and these will need to be proven separately.

4 An Example Translation.

In this section we illustrate the relationship between schemas and B machines in terms of the example database schema and the corresponding structured database schema. The machines were shown to be

consistent using the provers in the B-Toolkit [5].

```

MACHINE  person_base
SEES Person
VARIABLES person , income
INVARIANT
   $person \subseteq PERSON \wedge income \in person \rightarrow \mathbb{N}$ 
INITIALISATION
   $person := \{\} \parallel income := \{\}$ 
OPERATIONS
   $SKIP\_person \hat{=} skip ;$ 
   $ADD\_person (pers , inc) \hat{=}$ 
    PRE  $pers \in PERSON \wedge inc \in \mathbb{N}$ 
    THEN
       $person := person \cup \{ pers \} \parallel$ 
       $income := income \triangleleft \{ pers \mapsto inc \}$ 
    END ;
   $REM\_person (pers) \hat{=}$ 
    PRE  $pers \in PERSON$ 
    THEN
       $person := person - \{ pers \} \parallel$ 
       $income := \{ pers \} \triangleleft income$ 
    END ;
   $CHANGE\_income (pers , inc) \hat{=}$ 
    PRE  $pers \in person \wedge inc \in \mathbb{N}$ 
    THEN
       $income := income \triangleleft \{ pers \mapsto inc \}$ 
    END
END

```

Figure 6: Base B Machine for class *person*.

The base machine for *person* is called *person_base* and given in Figure 6. A similar machine *employee_base* is generated for the *employee* class: this is omitted for brevity. Having constructed the base machines, we define machines *person* as in Figure 7, and *employee* given in Figure 8.

A naive translation of the partial class *employee* given in Figure 2 would not include *delete_employee*, as this operation is not in that partial class, but inherited through the *is-a** relationship. Thus, implicitly, there is an operation *delete.person* in the class, specialised to *employee*, such that if the operation is called on an *employee* object it, and its corresponding *salary* attribute, is deleted.

In modelling such hierarchies in B, it has been proposed that the **INCLUDES** structuring mechanism is used to represent subclassing (see for example [14]). However, there is a fundamental difference between the notions of *is-a** relationship and the subclass relationship [4] as it is defined in ASSO. In this translation, if we **INCLUDE** machine *person* in machine *employee*, we inherit a *delete* operation which removes the *employee* object from the inherited *income* attribute but not from the desired *salary* attribute. This is because the semantics of B require that the invariants of the machine are explicitly re-established by operations rather than set implicitly. Thus, the **INCLUDES** mechanism does not adequately reflect the semantics of the *is-a** relation.

Consequently, we have to derive an explicit *delete_employee* operation to include in the *employee* machine. This can simply be given by a systematic textual replacement of *person* by *employee* in the delete operation, resulting in the *delete_employee* operation in Figure 8. By using the base operation *REM_employee* the attribute *salary* is modified accordingly.

The operation *addto_income* is also inherited by the machine *employee* and is explicitly specialised. However, in this case the only base operation used is *CHANGE_income*. As there is no *income* attribute in the class *employee*, there is no change of the state of *employee*, and so the specialised operation is effectively

```

MACHINE  person

SEES Person

INCLUDES person_base

INVARIANT
 $\forall pp . (pp \in person \Rightarrow income(pp) \geq 1000)$ 

OPERATIONS

  new_person (pers, i)  $\hat{=}$ 
    PRE  pers  $\in PERSON - person \wedge i \geq 1000$ 
    THEN
      ADD_person(pers, i)
    END ;

  delete_person (pers)  $\hat{=}$ 
    PRE  pers  $\in person$ 
    THEN
      REM_person (pers)
    END ;

  addto_income_person(pers, inc)  $\hat{=}$ 
    PRE  pers  $\in person \wedge inc \in \mathbb{N}$ 
    THEN
      CHANGE_income (pers, income (pers) + inc)
    END

END

```

Figure 7: The B machine for the *person* class

skip. We therefore omit it from this presentation.

The machine *company*, given in Figure 9, defines the external behaviour of *company*. It represents the operation *is-a** between the *person* and the *employee* classes. This machine **INCLUDES** both the *person* and *employee* machines, and thus imports the operations, (but not the base operations since **INCLUDES** is not transitive), of these machines. It has the inherent constraints of the *is-a** relationship, and models the operations as specified in the database schema.

Note that in this *company* machine, the preconditions of operations are not in the fully expanded form as described above. Here the preconditions have been simplified by expanding them out and determining which conjuncts are implied by the invariant on the machine. Note also that here we have given alternative *new* operations but have conflated the two *delete* operations by placing a guard on part of the substitution of the operation. This is an alternative approach to that described above, which captures the desired behaviour of the ASSO machine; however, it does not scale easily as in a more complex hierarchy cumbersome compositions of **IF** statements result.

5 Conclusions

In this paper we have described a formal approach to the transformation of ASSO structured database specifications into B Abstract Machines and gave an example to show the approach in action. The use of the semi-hiding principle in B leads to a hierarchical specification of the behaviour of the database, where data elements can only be modified via the base operations, known to preserve the inherent constraints. In a ASSO toolkit based on B, base machines should be generated and kept as read-only machines so that the developer cannot interfere with them.

By providing a translation from ASSO to B we have expressed the semantics of ASSO within a well-established formalism. This allows us to clarify its semantics, exposing implicit consequences of the ASSO semantics. For example the generation of external operations from partial class machines provides an

```

MACHINE   employee

SEES Person

INCLUDES employee_base

INVARIANT

 $\forall ee . ( ee \in employee \Rightarrow salary ( ee ) \geq 500 )$ 

OPERATIONS

 $new\_employee ( pers , sal ) \hat{=}$ 
  PRE    $pers \in PERSON \wedge sal \in \mathbb{N} \wedge$ 
          $sal \geq 500$ 
  THEN
    ADD_employee ( pers , sal )
  END ;

 $delete\_employee ( pers ) \hat{=}$ 
  PRE    $pers \in employee$ 
  THEN
    REM_employee ( pers )
  END ;

 $addto\_salary\_employee( emp , inc ) \hat{=}$ 
  PRE    $emp \in employee \wedge inc \in \mathbb{N}$ 
  THEN
    CHANGE_salary ( emp ,  $salary ( emp ) + inc$  )
  END

END

```

Figure 8: The B machine for the *employee* class

explicit representation of the abstract definition of such operations using the behavioural specialisation relationship *is-a**.

The *is-a** relationship is a novel feature of ASSO, differing from other approaches to interpretations of inheritance in modelling object-oriented systems in B in such work as [15, 14, 11, 12, 23, 21]. These approaches consider inheritance as simple subset inclusion of the object identifier sets, and a simple inheritance of (non-specialised) operations, with in some cases the use of B structuring mechanisms (**USES**, **INCLUDES**, **EXTENDS**) to give semi-hiding. However, they do not consider the inheritance of operations in more detail. The *is-a** inheritance mechanism is more subtle, especially in the use of operation specialisation, and requires a more global analysis of the system. It allows database specifications to be constructed from partial classes in a compositional manner, and with compositional consistency proofs. However, we have demonstrated that the **INCLUDES** structuring construct in B does not adequately capture operation specialisation of the kind required by ASSO. This may have implications beyond the domain of databases.

Class machines and operations can then be constructed from these base machines, and composed into the schema machine, reflecting the *is-a** hierarchy of the database schema, and a systematic check of the validity of the composition is possible.

This translation also allows tool support to be provided for ASSO using the existing B Tools. When the designer working in ASSO wishes to check the consistency of her specification, she invokes this automatic translation, with attendant supporting theories, and can then use the B proof obligation generators and provers to prove consistency.

Future developments of this work include assessing how this approach scales to larger more complex class hierarchies, especially with complex constraints between classes, and with multiple inheritance. Further investigations are also needed on developing real database systems using ASSO, especially using formal refinement. Work in progress is investigating the relationship between refinement in ASSO and B, partic-

```

MACHINE   company
SEES Person
INCLUDES person , employee
INVARIANT
    employee  $\subseteq$  person  $\wedge$  employee  $\triangleleft$  income  $\in$  employee  $\rightarrow \mathbb{N} \wedge$ 
     $\forall ee . ( ee \in employee \Rightarrow income ( ee ) \geq salary ( ee ) )$ 
OPERATIONS
    new_person ( pers , inc )  $\hat{=}$ 
        PRE   pers  $\in$  person  $\wedge$  inc  $\in \mathbb{N} \wedge$  inc  $\geq 1000$ 
        THEN
            new_person ( pers , inc )
        END ;
    new_employee ( pers , inc , sal )  $\hat{=}$ 
        PRE   pers  $\in$  person  $\wedge$  inc  $\in \mathbb{N} \wedge$  inc  $\geq 1000 \wedge$  sal  $\in \mathbb{N} \wedge$  sal  $\geq 500$ 
        THEN
            new_person ( pers , inc ) || new_employee ( pers , sal )
        END ;
    delete ( pers )  $\hat{=}$ 
        PRE   pers  $\in$  person
        THEN
            delete_person ( pers ) || IF pers  $\in$  employee
                                   THEN delete_employee ( pers )
                                   ELSE skip END
        END ;
    addto_income ( pers , inc )  $\hat{=}$ 
        PRE   pers  $\in$  person
        THEN
            addto_income_person ( pers )
        END ;
    addto_salary ( emp , inc )  $\hat{=}$ 
        PRE   emp  $\in$  person  $\wedge$  income(emp)  $\geq$  salary(emp) + inc
        THEN
            addto_salary_employee ( emp , inc )
        END
END

```

Figure 9: The Structured Database Schema *company* as a B machine

ularly the partitioning method, a further novel feature which the ASSO method brings from the database domain.

Acknowledgements

We would like to thank the support of the British Council and the Consiglio Nazionale delle Ricerche for supporting this work. In addition, we would like to thank our colleagues at Istituto di Elaborazione dell'Informazione, and at the Rutherford Appleton Laboratory, for their help and advice.

References

- [1] Abiteboul S, Hull R, Vianu V. Object databases. Chapter 21, Foundations of Databases, Addison-Wesley, 1995.
- [2] Abiteboul S, Hull R, Vianu V. Database dynamic aspects. Chapter 22, Foundations of Databases, Addison-Wesley, 1995.

- [3] Abrial J-R. The B-Book: Assigning Programs to Meaning. Cambridge University Press, 1996.
- [4] Andolina R, Locuratolo E. ASSO: behavioral specialisation modelling. In: Jaakkola H (ed) Information Modelling and Knowledge Bases VIII, IOS Press, 1997.
- [5] B-Core (UK) Ltd. The B-Toolkit. Welcome page URL <<http://www.b-core.com>>
- [6] B-Core (UK) Ltd. B-technology, technical overview. 1996.
- [7] Castelli D, Locuratolo E. A formal notation for database conceptual schema specifications. In: Jaakkola H (ed) Information Modelling and Knowledge Bases IV, IOS Press, 1994.
- [8] Castelli D, Locuratolo E. ASSO - A formal database design methodology. In: Jaakkola H (ed) Information Modelling and Knowledge Bases VI, IOS Press, 1995.
- [9] Castelli D, Locuratolo E. Enhancing database system quality through formal design. In: proceedings of the 4th Software Quality Conference, Abertay Dundee University and Napier University, 1995.
- [10] Dijkstra E W. A Discipline of Programming. Prentice-Hall, 1976.
- [11] Facon P, Laleau R, Nguyen H P. Mapping object diagrams into B specifications. In: Bryant A, Semmens L T (eds) proceedings of the Methods Integration Workshop, Leeds, UK, Springer-Verlag, 1996 M196).
- [12] Facon P, Laleau R, Nguyen H P. Dérivation de spécification formelles B à partir de spécifications semi-formelles de systèmes d'information. In: Habrias H (ed) proceedings of the 1st Conference on the B Method, Nantes, France, 1996.
- [13] Hull R, King R. Semantic database modeling: survey, application, and research issues. ACM Computing Surveys, 19(3) 1987.
- [14] Lano K. The B Language and Method. A Guide to Practical Formal Development. Springer-Verlag, 1996.
- [15] Lano K, Haughton H. Improving the process of system specification and refinement in B. In: Till D (ed) proceedings of the 6th Refinement Workshop, Workshops in Computing, Springer-Verlag, 1994.
- [16] Locuratolo E. ASSO: evolution of a formal database design methodology. In: proceedings of Symposium on Software Technology, (SoST'97), Buenos Aires, 1997.
- [17] Locuratolo E, Rabitti F. Conceptual classes and system classes in object databases. Acta Informatica, 1998, 35(3):181-210.
- [18] Locuratolo E. Database reengineering as modelling. IEI-CNR Report B4-43, 1998.
- [19] Locuratolo E, Matthews B M. Structured database schema and abstract machines. IEI-CNR Report B4-23, 1996.
- [20] Locuratolo E, Matthews B M. On the relationship between ASSO and B. In: proceedings of the 8th European-Japanese Conference on Information Modelling and Knowledge Bases, Vammala, Finland, 1998.
- [21] Matthews B M, Ritchie B, Bicarregui J C. Synthesising structure from flat specifications. In: Bert D (ed) proceedings of the 2nd International B Conference, Montpellier, France p.148-161 Springer-Verlag, 1998, pp148-161 (Lecture Notes in Computer Science No. 1393).
- [22] Schewe K-D, Schmidt J W, Wetzel I. Specification and refinement in an integrated database application environment. In: Prehn S, Toetenel W J (eds) proceedings of VDM'91, 4th International Symposium of VDM Europe, Springer-Verlag, 1991 (Lecture Notes in Computer Science No. 551).
- [23] Shore R. An object-oriented approach to B. In: Habrias H (ed) proceedings of the 1st Conference on the B Method, Nantes, France, 1996.