ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Antony Bryant and Lesley Semmens (Eds)

Methods Integration

Proceedings of the Methods Integration Workshop, Leeds, 25-26 March 1996

Paper:

Mapping Object Diagrams into B Specifications

Philippe Facon, Regine Laleau and Hong Phuong Nguyen

Published in collaboration with the
British Computer Society

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

Mapping Object Diagrams into B Specifications

Philippe Facon, Régine Laleau, Hong Phuong Nguyen

CEDRIC-IIE (CNAM) Laboratory
18 allée Jean Rostand - 91025 Evry - France
Email: {facon,laleau,phuong}@iie.cnam.fr

Abstract

In this paper we study the combination of object-oriented and formal methods and more specifically the mapping of object models, that are used in business information system development methods, into the formal specification language of the B method. We retain as the starting point of our study a common core of object concepts accepted by most of the groups: object, entity, association, is-a link. The first aim is to formalize these concepts. The main idea is to organize the specification around sets of object identities. Thus associations are modelled using relations between these sets and is-a links are modelled by inclusion constraints between these sets. The second aim is to modularise the final formal specification. Our purpose is to keep as much as possible the initial object diagram structure in the formal specification. Thus each entity is mapped into one abstract machine, an association is put in a specific machine according to its own properties, inheritance is simulated by using the existing links between abstract machines.

1 Introduction

In the specification of information systems, object-oriented methods provide diagrammatic notations for system statics and dynamics. These notations are intended to be intuitive, that is close to the usual mental processes people use in the initial modelling of an application domain. They are based on semantic data concepts [12] like entity or association that are widely used but not clearly formalized. On the other hand, only formal specification allows consistency verification, proof of properties and refinement to a provably correct code. Thus it is not surprising that a number of groups have started to work on the combination of object-oriented and formal methods [10]. From a methodological point of view, a first approach is to consider that a formal method can be used at the specification phase once the requirements analysis has been completed. Generally that analysis provides object diagrams from which formal specifications can be derived.

In this paper we address a specific problem in this area, namely the mapping of object models, that are used in methods like OMT [14] or OOA [16], into the formal specification language of the B method [1] that we briefly presents in Part 2. Contrary to some other projects we will discuss, our aim is to keep as much as possible the initial object diagram structure in the formal specification. We retained as the starting point of our study a common core of object concepts accepted by most of the groups working in business information systems: object, entity, association, is-a links.

The first step of our study, described Part 3, is to formalize the object concepts. In business information systems, most constraints and operations concern sets of existing objects. Thus it is fundamental to organize the specification around these sets and not around single objects nor around sets of possible objects (types). More precisely these sets are sets of existing object identities. For each entity a state variable will denote the corresponding set, that is included in some given set of possible object identities, e.g. $car \subseteq CAR$. Then attributes are modelled by state variables denoting relations between these sets and the attributes types, e.g. $colour \in car \rightarrow COLOUR$. Associations are modelled by state variables denoting relations between these sets, e.g. $owner \in car \rightarrow person$. An is-a link is modelled by set inclusion, e.g. $employee \subseteq person$.

The second step, presented in Part 4, is to modularise the formal specification. The problem is that the structuration mechanisms of B and the object model are relatively different. Indeed, the first one aims at decomposing the specification in order to separate the proof whereas the second one aims at modelling the real world faithfully. Then the mapping is not always straightforward. An entity is naturally mapped into an abstract machine. Associations play a central role in business information systems. Reflecting that role in the formal specification means in particular not to reduce them to references from some entities to other entities. Thus an association is denoted by a relation between object identities sets and put in a specific machine, distinct from the entities machines (except in few cases detailed in Part 4). The same reasoning must be followed for the inheritance concept: each entity of the hierarchy is mapped into one machine. The inheritance links are simulated by using the existing B links between abstract machines.

Part 5 presents related works. They are divided in two groups : those using the B method and those using other specification languages such as Z or VDM.

2 The Specification Language of the B Method

In the B method, specifications are organized in Abstract Machines. Each machine encapsulates state variables, an invariant constraining the state variables and operations on the state variables. The invariant is a predicate in a simplified version of ZF-set theory, enriched by many relational operators. Operations are specified in the Generalised Substitution Language, which is a generalisation of Dijkstra's guarded command notation, enhanced with preconditioning and unbounded choice. Proof of internal consistency of an Abstract Machine requires that each machine operation on the state variables maintains the invariant. Operations and Abstract Machines may be parameterized. It is possible to declare in an Abstract Machine abstract sets by giving their name, without further precision. A Machine Ma may *use* another machine Mb : the variables of Mb can be used but not modified in Ma . A machine Ma may *include* another machine Mb : the variables of Mb can be used or modified in Ma but only by using the operations of Mb . A machine may be included in at most one machine. The aim of these mechanisms and restrictions is to achieve incremental specification with separate consistency proofs.

3 Formalization of the Object Model Concepts in B

The object model we define in this part is not new : we have retained the most widely known concepts [2] such as object, entity, association or is-a link. Other more elaborated concepts such as qualification, role or connecting entity have been taken from the OMT method. Formalizing with mathematical notations sometimes induces us to choose between different possible interpretations of the OMT concepts, specially for the connecting entity and qualification concepts. Anyway the benefit from this work is to clearly precise the concepts by raising the ambiguity of natural language definitions.

For each concept we present a definition and the graphical notation through an example then the B representation illustrated by the example, that is, some parts of a machine. In order to simplify, we consider in that first step that a diagram is mapped into only one abstract machine.

3.1 Entity, Attribute, Object Identity, Key

The entity concept is used to assemble the objects sharing common characteristics. We distinguish two kinds of characteristics : the attributes which describe the structure of the entity and the operations which determine its behaviour. An object represents an instance of an entity.

We prefer the word entity rather than class because in business information systems it is used to assemble objects of the real world whereas class is used mainly in the programming community.

An attribute is defined by a name and has a type that determines the set of its possible values. An attribute value *cannot be an object*. An attribute type can be atomic (printable : string, boolean, integer, ...) or complex by the use of the type constructor set (multivalued).

As we have already explained, we have chosen to organize the B specification around the sets of existing objects. Then the representation of the static part of an entity e is achieved in two times : representation of the existing instances and representation of the attributes.

The set of the existing instances of an entity e is modelled by a variable V_e constrained to be a subset of a deferred given set S_e . S_e denotes the set of the possible instances of e (present or future). V_e is then typed in the abstract machine invariant by : $V_e \subseteq S_e$

So an entity Person is represented by the following abstract machine :

MACHINE	MA_Person
SETS	PERSON
VARIABLES	Person
INVARIANT	Person $\subseteq$ PERSON

Each attribute a of e is modelled by a variable V_a . The attribute typing is achieved in the invariant by defining the attribute as a binary relation between V_e and T_a , the set of all the possible values of the attribute.

Depending on whether the attribute is mandatory or optional, multi or mono valued, the relation can become a total or partial function.

For example if the attribute is monovalued and optional, V_a is a partial function : $V_a \in V_e \rightarrow T_a$;

if it is multivalued and mandatory : $V_a \in V_e \leftrightarrow T_a \wedge \text{dom}(V_a) = V_e$.

Let us complete the previous entity Person :

Figure 1: An Example of an Entity

Then we have the following machine (recall that all the examples of machines we give are partial) :

MACHINE	MA_Person
SETS	PERSON
VARIABLES	Person, lastname, firstname, age, children
INVARIANT	$\text{Person} \subseteq \text{PERSON} \wedge$ $\text{lastname} \in \text{Person} \rightarrow \text{STRING} \wedge$ $\text{firstname} \in \text{Person} \rightarrow \text{seq}(\text{STRING}) \wedge$ $\text{age} \in \text{Person} \rightarrow \mathbb{N} \wedge$ $\text{children} \in \text{Person} \leftrightarrow \text{STRING}$
OPERATIONS	
	<i>Add_Person(p, l, f, a)</i>
pre	$p \in \text{PERSON} - \text{Person} \wedge l \in \text{STRING} \wedge f \in \text{seq}(\text{STRING}) \wedge a \in \mathbb{N}$
then	$\text{Person} := \text{Person} \cup \{p\} \mid \mid \text{lastname}(p) := l$ $\mid \mid \text{firstname}(p) := f \mid \mid \text{age}(p) := a$
end	

Remarks

- Object identity and key:

Each object has a unique identity, corresponding to the object identifier concept in object-oriented database systems, *independent of its value* and constant in time. In the other hand, an entity key is a subset of its attributes whose *value* determines a unique instance. An entity can have zero, one or more keys.

As shown above, the B modelling of the object identity concept is implicitly achieved by defining a deferred given set for each entity. This is a set whose only the name is known at a first level of specification. It is assumed to be finite and non-empty. It allows to define a type and implicitly includes all the possible values of this type.

On the other hand, the key concept is modelled by adding an invariant constraint expressing that instances of the entity are in injection with the key attributes : if $(a1, a2, a3)$ is the key then we have : $\forall a1 \otimes \forall a2 \otimes \forall a3 \in V_e \rightarrow T_{a1} \times T_{a2} \times T_{a3}$

Recall that, if $V_1 \in E \leftrightarrow F$ and $V_2 \in E \leftrightarrow G$, the direct product $V_1 \otimes V_2$ is the set of pairs $(x, (y, z))$ such that $(x, y) \in V_1$ and $(x, z) \in V_2$. Thus $V_1 \otimes V_2 \in E \leftrightarrow (F \times G)$.

- Object identity and given sets:

It is natural to have different given sets (*PERSON*, *CAR*, ...) for different entities, each denoting the set of all possible object identities for an entity: $\text{person} \subseteq \text{PERSON}$, $\text{car} \subseteq \text{CAR}$, etc..

But this is not the only possible solution : we may have a unique given set *OBJECT* for all possible object entities with $\text{person} \subseteq \text{OBJECT}$, $\text{car} \subseteq \text{OBJECT}$ etc.. (and some disjointness constraints).

A third solution would be to have not only *OBJECT* but also the set object of all existing objects ($\text{object} \subseteq \text{OBJECT}$), the set of all entity names and a relation *instanceof* between objects and entities.

With different variables, and even better with different given sets, it is possible to encapsulate sets of instances in different machines. With an *OBJECT* type, and even better with an object variable, it is easy to define general functions between objects, and to formalize explicitly the metamodel concepts.

We have only detailed the first approach. The other approaches will be the subject of another paper.

3.2 Association, Role, Fixed Association, Qualification

3.2.1 Association

The association concept is used to connect two entities and then allow access from one object to its linked objects. An association is characterised by its name, the cardinality constraints with the values $1,1$ or $1,n$ or $0,1$ or $0,n$ or m,n (implicitly $n \geq m > 1$). It can have attributes.

Figure 2.a: An Example of an Association

As an association instance can be represented by a couple constituted with an instance of each linked entity, it is not necessary to define a set of all the possible instances like for an entity. In B an association r between two entities e and f is then modelled by a variable V_r which is a relation between V_e and V_f . This relation, or its inverse V_r^{-1} , can become a partial function or an injection, ... depending on the cardinalities. Here are some examples where the first (resp. second) cardinality is that of the entity e (resp. f) in the association r :

$$\begin{array}{ll}
 (0,n),(0,n) & \Rightarrow V_r \in V_e \leftrightarrow V_f \\
 (1,n),(1,n) & \Rightarrow V_r \in V_e \leftrightarrow V_f \wedge \text{dom}(V_r)=V_e \wedge \text{ran}(V_r)=V_f \\
 (0,n),(1,1) & \Rightarrow V_r^{-1} \in V_f \rightarrow V_e \\
 (0,1),(0,1) & \Rightarrow V_r \in V_e \rightarrow V_f \\
 (1,1),(1,1) & \Rightarrow V_r \in V_e \rightarrow V_f
 \end{array}$$

To take into account an association with its own attributes, a variable and a typing invariant are added for each attribute by using the same rule as for entity attributes. So the diagram of the previous figure is translated by the following abstract machine (that uses implicitly a machine defining the type DATE) :

MACHINE	Diagram_fig_2.a
SETS	PERSON, CAR
VARIABLES	Person, Car, own, purchase_date
INVARIANT	$\text{Person} \subseteq \text{PERSON} \wedge \text{Car} \subseteq \text{CAR} \wedge$ $\text{own}^{-1} \in \text{Car} \rightarrow \text{Person} \wedge$ $\text{purchase\_date} \in \text{own} \rightarrow \text{DATE}$

3.2.2 Role

An association determines two roles. A role gives for each object the object or the set of objects linked to it by the association.

Figure 2.b: An Example of Roles in an Association

PERSON plays the *owner* role towards *CAR*, *CAR* plays the *ownership* role towards *PERSON*.

This concept has a direct equivalent in B thanks to the DEFINITIONS clause. So, for each role a definition is added. In the previous example the abstract machine is completed with :

MACHINE	Diagram_fig_2.a
...	
DEFINITIONS	$\text{owner}(x) \triangle \text{own}^{-1}(x) \wedge$ $\text{ownership}(x) \triangle \text{own}[\{x\}]$

3.2.3 Fixed Association

An association is fixed for an entity if, for each instance of the entity participating to the association, its links can be created only at the instance creation and are never changed for its lifetime. We have defined this concept because it is very important for the mapping of a conceptual diagram into abstract machines, as we will explain in Part 4. It is closed to that of the CODASYL model "retention is fixed" [4].

Figure 3: An Example of a Fixed Association for the Entity ORDER

place is a fixed association for *ORDER*. Indeed a link between an *ORDER* and its *CUSTOMER* never changes : an order will always be associated to the same customer. On the other hand a customer can add a new order or, in the example of Figure 2.a, a car can change its owner (and conversely), so *own* is not a fixed association for any of the linked entities.

From a static point of view, it is a classical association and its B representation is the same. From a dynamic point of view, a fixed association does not own specific operation. More precisely, an association instance is never modified and is created (resp deleted) by the operation that creates (resp deletes) an instance of the entity for which it is fixed (here *ORDER*). This is translated by the following proof obligation. If an association *r* between *e* and *f* is fixed for *e*, then for each operation OP:

$$(x \in V_e \wedge V_r[\{x\}] = q) \Rightarrow [OP] (x \in V_e \Rightarrow V_r[\{x\}] = q)$$

where [OP] Q is the predicate stating that operation OP establishes condition Q.

3.2.4 Qualification

In Paragraph 3.1 entity keys are restricted to subsets of the entity attributes. This concept can be extended by considering the OMT concept of qualification. It specifies that an entity instance can be determined by a pair formed by a subset of its attributes that is not a key, and another linked entity instance.

Figure 4: An Example of Qualification

filename is a qualification in the association *contain*: in a given directory, each file has a different name. In other words a directory together with the filename determines one file.

In B this external key is translated by the following invariant constraint. If $(e, \{q1, q2, \dots\})$ forms a qualification of *f* in *r* :

$$(V_r^{-1} \otimes V_{q1} \otimes V_{q2} \otimes \dots)^{-1} \in V_e \times T_{q1} \times T_{q2} \times \dots \rightarrow V_f$$

Depending on the cardinality of *f* in the association, the function can become an injection (0,1), a surjection (1,n) or a bijection (1,1) but is generally partial. In the bijection case, we find the "weak entity" concept of Chen [3].

If there is only one attribute in the qualification, then we can define a total function and the constraint becomes :

$$(V_r^{-1} \otimes V_q)^{-1} \in V_r ; V_q \rightarrow V_f$$

So in the previous example, we obtain: $(\text{contain}^{-1} \otimes \text{filename})^{-1} \in \text{contain} ; \text{filename} \rightarrow \text{File}$

3.3 Connecting Entity

As we have explained in the above paragraph, an association is a binary relation between two entities. However, if n-ary associations or associations with associations have to be considered, then this concept is not sufficient. For instance, suppose we have an association *ORDER_LINE* between *ORDER* and *PRODUCT* and we want to create another association between *ORDER_LINE* and *INVOICE_LINE*. Thus *ORDER_LINE* becomes an entity that must behave like an association. So we use the concept of connecting entity : it is an entity determined by some entities linked to it.

Figure 5: An Example of a Connecting Entity

Each cardinality from the connecting entity towards the connected entities is equal to (1,1) and each pair of connected instances determines at most one instance of the connecting entity. Let us remark that the associations linking a connecting entity to its determining entities are fixed for the connecting entity.

In Figure 5, the connecting entity *ORDER_LINE* is determined by the entities *ORDER* and *PRODUCT*. That is, a couple (o, p) - where o is an instance of *ORDER* and p an instance of *PRODUCT* - determines at most one instance of *ORDER_LINE*. Thus, we specify that the connecting entity is still in bijection with a binary relation between *ORDER* and *PRODUCT*.

In some cases own attributes can participate to the determination of the connecting entity as the attribute *loaning_date* in the following figure.

Figure 6: A Connecting Entity with a Determining Own Attribute

Here (r, b, l) - where r is an instance of *READER*, b is an instance of *BOOK* and l is a value of *loaning_date* - determines one instance of *LOAN*.

The B representation of a connecting entity is the same as for an entity with additional invariant constraints. Let c be such an entity, determined by the entities $e_1, e_2, \dots, e_n$, via the associations $r_1, r_2, \dots, r_n$, then we have :

- S_c : set of the possible instances
- V_c : set of the existing instances, $V_c \subseteq S_c$

- one variable V_{r_i} for each association with the determining entities: $V_{r_i} \in V_c \rightarrow V_{e_i}$ because c cardinality is equal to (1,1). Other constraints can be added according to the determining entities cardinality in the associations.
- $V_{r_1} \otimes V_{r_2} \otimes \dots \otimes V_{r_n} \in V_c \rightarrow V_{e_1} \times V_{e_2} \times \dots \times V_{e_n}$
 c instances are in injection with tuples of instances of the determining entities.

Remarks:

- the constraint on each V_{r_i} is deduced from this predicate.
- if an attribute a of type T_a participates to the determination of the connecting entity, then we have :

$$V_{r_1} \otimes \dots \otimes V_a \otimes \dots \otimes V_{r_n} \in V_c \rightarrow V_{e_1} \times \dots \times T_a \times \dots \times V_{e_n}$$

For instance, the diagram Figure 6 is translated into :

MACHINE	library
SETS	READER, BOOK, LOAN
VARIABLES	Reader, Book, Loan, R_L, L_B, loaning_date, reminder_date
INVARIANT	$\text{Reader} \subseteq \text{READER} \wedge \text{Book} \subseteq \text{BOOK} \wedge \text{Loan} \subseteq \text{LOAN} \wedge$ $\text{loaning\_date} \in \text{Loan} \rightarrow \text{DATE} \wedge$ $\text{reminder\_date} \in \text{Loan} \rightarrow \text{DATE} \wedge$ $\text{R\_L} \in \text{Loan} \rightarrow \text{Reader} \wedge \text{L\_B} \in \text{Loan} \rightarrow \text{Book} \wedge$ $\text{R\_L} \otimes \text{L\_B} \otimes \text{loaning\_date} \in \text{Loan} \rightarrow \text{Reader} \times \text{Book} \times \text{DATE}$

3.4 Is-a Hierarchy

We consider the usual concept defined in [17]. It is used to refine an entity, called the super-entity, into one or more entities, its sub-entities (the link is called a specialisation link). The super-entity contains all the instances of its sub-entities. Additional conditions can be defined such as: the super-entity cannot own instances (such entities are sometimes called abstract entities or generalisations).

In this paragraph, only the static aspect is presented. The operation inheritance is linked to the decomposition of a diagram into modules and is detailed in the next part.

Figure 7.a: Is-a Link

Figure 7.b: Is-a Link with Generalisation Constraint

These concepts are translated into B according to the following rules:

- There is only one given set for the whole inheritance hierarchy. It is associated with the highest level entity and represents the set of all the possible instances in the hierarchy. There are as many state variables representing the existing instances as entities in the inheritance hierarchy. For instance the translation of Figure 7.b is :

Given set : READER

Variables : Reader, Adult, Child

In B the typing invariant of *Reader* is $\text{Reader} \subseteq \text{READER}$, the typing invariants of the other variables follow the next rules.

- Each is-a link is modelled by an inclusion constraint : e is-a f gives the invariant constraint $V_e \subseteq V_f$. (e.g. $\text{Employee} \subseteq \text{Person}$)

- As previously, each attribute of the hierarchy is represented by a state variable that is a function from the corresponding entity to the attribute type (e.g. $\text{name} \in \text{Reader} \rightarrow \text{STRING} \wedge \text{profession} \in \text{Adult} \rightarrow \text{STRING}$).
- Additional conditions are translated by additional invariants, for instance the generalisation in Figure 7.b : $\text{Reader} = \text{Adult} \cup \text{Child}$ (of course in that case $\text{Adult} \subseteq \text{Reader}$ and $\text{Child} \subseteq \text{Reader}$ become redundant).

4 Modularising the Formal Specification

The use of modules is necessary to manage the size and complexity of a system. Indeed it allows to improve the system comprehension and to develop it in an incremental way. Furthermore it facilitates the evolutivity and the reuse of some components. In B this mechanism is achieved by the use of abstract machines. Let us recall that machines can be related by using the *includes* or the *uses* clause.

This part is devoted to the problem of modularising the formal specification obtained by applying the rules defined in the previous part. The basic and natural idea is to respect the structuration of the conceptual diagram whose aim is to model the real world faithfully. However B has its own modularisation mechanism that does not necessarily match the data structuration because its aim is mainly to separate proof. So the mapping from a conceptual diagram into a B specification is not straightforward if we want to conciliate the two points of view. In the following of the chapter we successively present the solution we have adopted for each data concept and we compare it with other approaches if it is different.

4.1 Entity

As an entity includes both static and behavioural properties of a set of objects, it seems natural to map it into one abstract machine. Thus the *given set* clause contains at least the set of possible instances (perhaps other sets necessarily to the attributes definition), the *variables* clause contains the set of existing instances and the attributes variables, the *invariant* clause contains at least the variables typing plus other constraints like integrity or connecting entities constraints. Finally the *operations* clause contains the operations that manipulate one instance or a set of instances.

4.2 Association

In information systems, associations play an essential role. Many operations are creation, suppression or modification of links between objects. As we have seen above, the most direct formalization of the association concept is the mathematical relation, that is a set of pairs. The problem now is where to put the association: in one of the linked entity machines or in a third machine?

Previous solutions have been either to create a machine for each association [11] or on the contrary to always put the association variable in an entity machine with a *use* link between the linked entity machines [10]. None of these solutions seems really satisfactory: the first one may produce too many machines, some being uninteresting (because the really interesting invariants will be in including machines). On the contrary, the second solution may include too much information in the same machine: some associations may have many operations either on the links themselves, either on the link attributes and are normal candidates for an independent machine; furthermore rather frequently some entities will have to be put in the same machine to avoid circularity in the *use* link.

In order to obtain a good modularization, we create a machine for the association if and only if it is subjected to independent operations, that is operations either on the association attributes or on links between existing objects. Obviously there are operations on attributes if there are attributes. There are operations on links between existing objects if these links are not fixed for one or the other side.

Recall that an association r is fixed for an entity e if the r links of an e -instance are created together with the e -instance and may not be changed during the e -instance lifetime.

The following rules are applied to translate an association r between e and f :

- If r has attributes or is not fixed for e or f , we will have an independent machine Mr that uses Me and Mf (and a machine including Me , Mf and Mr). This machine has only the variables linked to r (Vr and possible attributes variables), the typing invariants and operations that manage the association.
- If r is without attributes and is fixed for e but not fixed for f , it will be put in Me and Me uses Mf . Me contains in addition Vr and possible attributes variables and the typing invariants.

- If r is without attributes and fixed for e and f , there is a priori no reason to choose between Me and Mf . In this case the choice is up to the user; a consideration that may help to choose is: is the association mandatory for e and not for f ? in this case it seems better to put r in Me (because creation of e instances have to be done together with creation of r links).

In the case of a connecting entity, the associations linking it to its determining entities are fixed and thus one of the two last rules applies.

Of course to avoid circularity of *use* links, we may have also to put two entities in the same machine (because two associations) but this happens very rarely in practice.

Note that if an association r is mandatory for e and $\forall r$ has not been put in Me , the creation of a e instance alone is hidden in the machine Me : the including machine (where the invariant expressing the mandatory parameter will be proved) uses that operation to define an operation creating in parallel the instance of e and its links.

Let us take again the example of Figure 3 : the association *place* has no attributes, is fixed for *ORDER* and not for *CUSTOMER*, so it will be put in the abstract machine defined for *ORDER* :

Machine	MA_Order
USES	MA_Customer
SETS	ORDER
VARIABLES	Order, date, place
INVARIANT	$\text{Order} \subseteq \text{ORDER} \wedge \text{date} \in \text{Order} \rightarrow \text{DATE} \wedge \text{place}^{-1} \in \text{Order} \rightarrow \text{Customer}$
OPERATIONS	<i>Add_Order(date, Cu)</i> Pre $\text{Order} \subset \text{ORDER} \wedge \text{Cu} \in \text{Customer} \wedge \text{date} \in \text{DATE}$ then ANY Or WHERE $\text{Or} \in \text{ORDER} - \text{Order}$ THEN $\text{Order} := \text{Order} \cup \{\text{Or}\} \parallel \text{date}(\text{Or}) := \text{date} \parallel \text{place} := \text{place} \cup \{\text{Cu} \mapsto \text{Or}\}$ END end <i>Delete_Order(Or)</i> Pre $\text{Or} \in \text{Order}$ then $\text{Order} := \text{Order} - \{\text{Or}\} \parallel \text{date} := \{\text{Or}\} \triangleleft \text{date} \parallel \text{place} := \text{place} \triangleright \{\text{Or}\}$ end

4.3 Inheritance

In B there is no concept equivalent to the inheritance concept of information systems. Then, it must be simulated by using the *uses* and *includes* links between abstract machines. We have to solve the same problem for inheritance than for associations: if an entity e inherits from an entity f , we have seen that $\forall e \subseteq \forall f$, but where to put $\forall e$ and $\forall f$?

As for associations, some solutions have already been proposed:

- to put the variables in the same machine [10] : that solution loses the benefit of incremental specification by inheritance.
- to put them in different machines with including links (*Me includes Mf*) [11]; that solution does not work because, as one machine may be included in at most one other, a class may have only one direct subclass.

Our solution is to have for such an inheritance link two machines Me and Mf with $Me \text{ uses } Mf$. For the whole hierarchy, we will have a machine Mi including all these machines. In Me we can create e instances (but not as f instances) or specialise existing f instances as e instances or remove the e qualification from f instances. In Mi we can use these operations to really create e instances as special cases of f instances (by applying in parallel e -creation and f -creation). Let us take again the example of Figure 7.a with *MA_Person* given in Paragraph 3.1. Then we have the following B specification architecture :

MA_Person contains the characteristics common to the entities *PERSON* and *EMPLOYEE* and the instances of *PERSON*. *MA_Employee* contains the special characteristics and the own instances of *EMPLOYEE*. *MA_Staff* includes both machines, has no variables of its own and contains the inheritance hierarchy interface operations.

```

Machine    MA_Employee
USES      MA_Person
VARIABLES   Employee, salary
INVARIANT   Employee  $\subseteq$  Person  $\wedge$  salary  $\in$  Employee  $\rightarrow \mathbb{N}$ 
OPERATIONS
  Add_Employee(p, sal)
  pre   p  $\in$  PERSON - Employee  $\wedge$  sal  $\in \mathbb{N}$ 
  then  Employee := Employee  $\cup$  {p} | | salary(p) := sal
  end

```

Note that the invariant *Employee* $\subseteq$ *Person* can only be proved in the including machine *MA_Staff*, and constraints like disjoint subtypes (eg. *Employee* $\cap$ *Manager* = $\emptyset$) can only be put in this machine.

```

Machine    MA_Staff
INCLUDES  MA_Person, MA_Employee
OPERATIONS
  person  $\leftarrow$  Arrival(lastname, firstname, age)
  pre   Person  $\subset$  PERSON  $\wedge$  lastname  $\in$  STRING  $\wedge$  firstname  $\in$  seq(STRING)  $\wedge$  age  $\in \mathbb{N}$ 
  then  any p where p  $\in$  PERSON - Person
        then Add_Person(p,lastname, firstname, age) | | person := p
        end
  end
  Qualif_Employee (p, sal)
  pre   p  $\in$  Person - Employee  $\wedge$  sal  $\in \mathbb{N}$ 
  then  Add_Employee (p, sal)
  end
  emp  $\leftarrow$  Arrival_Employee(lastname, firstname, age,sal)
  pre   Person  $\subset$  PERSON  $\wedge$  lastname  $\in$  STRING  $\wedge$  firstname  $\in$  seq(STRING)  $\wedge$  age  $\in \mathbb{N} \wedge$  sal  $\in \mathbb{N}$ 
  then  any p where p  $\in$  PERSON - Person
        then Add_Person(p,lastname, firstname, age) | | Add_Employee (p, sal) | | emp := p
        end
  end
  etc...

```

5 Related Work

Other projects have studied the translation from informal to formal specifications. They all differ from our approach in some of the following aspects:

- some projects do not use object identity or not so systematically as we do. For instance, there is a kind of object identity in SAZ [13] but associations are defined between values, not between object identities. Of course this is due to the SA method which is not object-oriented but value-oriented. In our case, associations are modelled by relations between object identities, not between object values. The advantages of object identities are now well recognised. For instance the relation $owner$ with $owner \in Car \rightarrow Person$ contains no information about Car or $Person$ attributes. Thus nothing has to be propagated on $owner$ after the modification of a car colour or ID number.
- most projects use explicit typing, which is mandatory in most formal specification languages. Explicit typing is especially painful in information systems, because attributes and associations are for existing objects and that constraint generally implies typing information. This was the case in a previous work where we used VDM [8,9]. For instance, if you say that $owner$ is of type $CAR \rightarrow PERSON$ you will have to add something like $dom(owner) = Car \wedge ran(owner) \subseteq Person$. In our case, we simply write: $owner \in Car \rightarrow Person$, the type $CAR \rightarrow PERSON$ will be deduced.
- most projects describe each entity instance by a state machine including attributes variables [7] or they use the cartesian product of the attributes types [5]. The aim of [7] is to encapsulate object properties as in object oriented programming languages. The whole state is then described by maps from object identifiers to object values.

Our approach is maybe less intuitive (we have no object internal state but we use a "whole state" approach) but constraints and operations are much easier to express:

- an optional attribute is expressed by the partiality of the associated relation; null values are not necessary.
- cardinality constraints for an association are immediately expressed by properties of the corresponding relation (functionality, injectivity, surjectivity, totality).
- most other constraints involving attributes and/or associations (candidate keys, qualification) are very easily expressed by using the direct product between relations modelling these attributes and associations. Let us recall the following example : in a directory a file name is associated to a unique file, we simply write : $dir \otimes name \in file \rightsquigarrow DIR \times NAME$
- concerning operations, other approaches need to propagate on the whole state modifications that are specified on a single object; changing an attribute value by an operation of the object state machine has to be propagated to the map (object identities $\rightarrow$ object values) [7]. In our case, such a modification is directly expressed on the function modelling that attribute; no propagation has to be done.

Three projects have addressed the more specific translation from semi-formal specifications to B:

[11] proposed rules from extended entity association diagrams to B. There are several problems raised by that proposal: there is no distinction between existing and possible objects (in fact types seem to have been forgotten) ; associations are modelled by object identities sets (where we use relations) with projection functions, but the given constraints are not sufficient (with the given definition, a pair of objects may be linked by two different links of the same association); inheritance is modelled by set inclusion and abstract machine inclusion : but in B a machine may be included in at most one machine, thus a class may have at most one (direct) subclass.

The DAIDA project [15] has addressed the translation from the semi-formal language TDL to B. But in TDL there is no association concept, and the translation produces a single machine whereas our work is specially concerned by modularising the formal specification. In fact that work is more about refinement, we do not handle here.

Another interesting work has been presented in [10]. This work is also more about aspects we do not deal with here: specification of the dynamics and refinement. Concerning the static part, the formalization of classes is the same as ours but the decomposition in abstract machines, specially with respect to associations and inheritance, is very different : neither associations nor subclasses have their own machines. Furthermore, no rules are given for the translation of different kinds of constraints.

6 Conclusion

The most important benefit from using a formal language like B in information system design is that the specifications are much more precise and concise than the traditional, semi-formal, ones. Secondly, the use of formal specifications enables refinement towards provably correct code. Among formal languages, B seems to be specially well adapted to information system design, thanks to characteristics such as encapsulation, implicit typing, set and relational operators [18]. However frequent objection to the use of formal notations is that they are not easily understood by mathematical novices, are difficult to learn and are not close to the usual mental processes people use to model application domains.

This is why in this paper we have proposed to model the object concepts in B, so that the designer can first define his object diagram with a complete development environment of an information system analysis and design method, and not directly with B, he will then obtain the static component specification of the B abstract machines by deriving from his diagram. Contrary to some other projects, our aim was to keep as much as possible the initial object diagram structure in the formal specification. A careful decomposition in abstract machines allowed us to achieve this end.

One of our future tasks is to study the dynamic component. The starting point can be state/transition diagrams or management rules expressed as Event-Condition-Action rules, that are the formalisms generally used in business information system development.

Furthermore we investigate another approach to map an object diagram into B [6]. The problem is that there is generally no exact equivalent, in the formal specification language, of the metamodel concepts (like object, entity, association, inheritance,...). Thus, even in our solution, the formalization of these concepts is hidden in the formalization of the application concepts (like customer, product, order, ...). As mentioned in Paragraph 3.1, a solution could be to use given sets and variables for metamodel concepts : $object \subseteq OBJECT$, $instance\_of \in object \leftrightarrow ENTITY$, etc. ... , that enables to formalize once for all the metamodel. Then, for each application, only an immediate translation from the object diagram is needed : $ENTITY = \{customer, product, order\}$, $ASSOCIATION = \{own, place\}$. We are currently developing that two-level solution.

7 References

1. Abrial J.R. - The B Book : Assigning Programs to Meanings, To be published, 1996.
2. Brunet J. - Modelling the world with semantic objects. IFIP TC8 International Conference on the Object-Oriented Approach in Information Systems, Quebec, October 1991.
3. Chen P.P. - The Entity-Relationship Model : Toward a Unified View of Data. ACM Transactions on Database Systems, Vol 1, N° 1, March 1976, pp 9-36
4. CODASYL - Feature Analysis of Generalized Data Base Management Systems. Sys. Committee Tech. Report, ACM, New York.
5. Dick J., Loubersac J. - Integrating Structured and Formal Methods: a Visual Approach to VDM. 3rd European Software Engineering Conference, LNCS, Springer-Verlag, Milan, Italy, October 1991.
6. Facon P., Laleau R. - Des spécifications informelles aux spécifications formelles : compilation ou interprétation?. 13th French Conference INFORSID'95, Grenoble, France, May 95.
7. Hall A. - Using Z as a Specification Calculus for Object-Oriented Systems. 3rd International Conference VDM'90, Kiel, LNCS n° 428, Springer-Verlag, April 1990.
8. Laleau R., Facon P. - Modeling Semantic Data Concepts with VDM. 5th International Conference Software Engineering and Knowledge Engineering, San Francisco, June 93, pp 228-230.
9. Laleau R., Facon P. - Using VDM as a Conceptual Modelling Language. 1st Basque International Workshop on Information Technology, Biarritz, France, February 1994, pp. 157-168.
10. Lano K., Haughton H. - Improving the Process of System Specification and Refinement in B. 6th Refinement Workshop, D. TILL Editor, Springer Verlag, Workshops in Computing, 1994.

11. Nagui-Raïss N. - A Formal Software Specification Tool using the Entity-Relationship Model. 13th International Conference on the Entity-Relationship Approach, LNCS, Springer-Verlag, Manchester, UK, December 1994.
12. Peckham J. and Maryanski F. - Semantic Data Models. ACM Computing Surveys, Vol 20, N° 3, September 1988, pp. 153-189.
13. Polack F., Whiston M., Mander K. - The SAZ Project: Integrating SSADM and Z. International Symposium Formal Methods Europe, Odense, Denmark, April 1993, pp. 541-557.
14. Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. - Object-Oriented Modelling and Design. Prentice Hall : New Jersey, 1991.
15. Schewe K.D., Schmidt J.W., Wetzel I. - Specification and refinement in an integrated database application environment. 4th International Conference VDM'91, Noordwijkerhout Holland, LNCS, Springer-Verlag, October 91.
16. Shlaer S. and Mellor S. J. - Object-oriented systems analysis. Modelling the World in Data, Yourdon Press Computing Series, 1988.
17. Smith J.M. and Smith D.C.P. - Database Abstractions : Aggregation and Generalization. ACM Transactions on Database Systems, Vol 2, N° 2, June 1977, pp 105-133.
18. Tisserand P - Traduction d'un schéma Entité/Association en spécifications formelles VDM et B. DEA thesis, Evry, France, September 1995.