To appear in *Mathematical Foundations of Computer Science* 93, LNCS, Springer-Verlag

# Data Encapsulation and Modularity:
# Three Views of Inheritance

**J.F.Costa, A.Sernadas & C.Sernadas**

INESC & Dept. Matemática - IST
Apartado 10105, 1017 Lisboa Codex, PORTUGAL
fgc@inesc.pt

**Abstract**. A semantic domain based on state-machines is proposed for object-orientation in order to clarify the most important constructions: aggregation, interconnection and specialization. Three kinds of specialization are discussed: subtyping (specialization without side-effects and no non-monotonic overriding); monotonic specialization (possibly with side-effects but still with monotonic overriding only); and non-monotonic specialization (possibly with side-effects and non-monotonic overriding).

## 1.Introduction

The advantages of the object-oriented approach to software engineering are by now well understood. Many object-oriented languages, systems and methods have been proposed and some of them are now extensively used. To name just a few, there are programming languages like Smalltalk, $C^{++}$ and Eiffel, database systems like GemStone, $O_2$, IRIS and ORION, development methods like GOOD, MOOD and HOOD, as well as high-level specification languages [Goguen 91, Jungclaus *et al* 91, SernadasA *et al* 89]. Moreover, some effort has been dedicated to the theoretical foundations of this promising paradigm. The overview [Ehrich *et al* 92] points to several schools of thought in this area: *e.g.* ADT-based like [Goguen 91] and process-based like [Ehrich *et al* 91]. Higher programmer productivity through software reuse is paramount among the claimed advantages of object orientation. A rich mechanism of inheritance, including both specialization and abstraction, is essential for supporting the envisaged levels of software reuse. Another essential ingredient is a general mechanism for putting objects together, that is, for setting-up communities of interacting objects. Therefore, a semantic foundation of object-orientation should address all these issues: specialization, abstraction and interconnection. Herein, we adopt the view that an object is a state-machine, adapting the notion of reactive system proposed in [Manna and Pnueli 81]. Then, we develop the notion of object morphism in order to explain the basic relationhips between objects: *is-a* and *is-part-of*. To this end we apply the program outlined in [Costa *et al* 92] to objects as state-machines. The advantages of the new semantic domain are all related to the abstract view of the state: both the behaviour (process) and the memory structure (slot valuation) of the object appear as secondary concepts on top of the notion of state. By avoiding concrete states we arrive at a cleaner notion of object morphism that makes possible a unified semantic domain explaining all basic relationships between objects. Namely, we explain three kinds of specialization (strict, with side-effects, and with overriding), the part-of relationship between an object and each of its components and two basic kinds of mechanisms for interconnecting objects (component sharing, of which event sharing is a special case, as well as event calling). To our knowledge this is the first semantic domain for objects that incorporates these features.

In Section 2 the basic category of objects as state-machines is introduced. Section 3 addresses subtyping (strict specialization). Section 4 deals with object interconnection. Finally, in Section 5 we deal with liberal forms of specialization. We make moderate use of the theory of categories. The reader may find all the necessary category-theoretic notions in the book by Jiří Adámek *et al* [Adámek *et al* 90].

## 2.Objects

We start with an extension of the model found in [Winskel 84]. By Pfn we denote the category of sets and partial functions. Let V be a fixed class of sets.

***Definition 2.1***. A *slot alphabet* is a pair Att=$<A,\partial>$ where A is a set and $\partial$ is a total map from A into V. ◇

The idea is that, for each slot *a* in A, $\partial$ returns its codomain, i.e., an element of V. In this way $\partial(a)$ is the set of the possible values that slot *a* may have.

***Definition 2.2***. An *object* ob=$<E,Att,S,Q,B,\mu,\delta>$ is a septuple consisting of a set E, a slot alphabet Att=$<A,\partial>$, a set S, two subsets Q and B of S, a family $\mu=\{\mu_a\}_{a\in A}$ of total functions $\mu_a:S\rightarrow\mathbf{2}^{\partial(a)}$ and a family $\delta=\{\delta_e\}_{e\in E}$ of total functions $\delta_e:S\rightarrow\mathbf{2}^S$. ◇

The elements of E are called *events*, the elements of S *states*, the elements of Q *quiescent states* and the elements of B birth (*initial*) *states*. Each *valuation function* $\mu_a:S\rightarrow\mathbf{2}^{\partial(a)}$ gives the set of values that *a* may have in a given state. If $\mu_a(s)$ is empty we say that *the slot a is undefined in state s*. It may also happen that a slot is given more than one value. Finally, each function $\delta_e:S\rightarrow\mathbf{2}^S$ is called the *transition function for e*: gives the set of states reachable from a given state by the occurrence of event *e*.

Consider a *bank* account as object *acc*, with $A_{acc}$={balance}, $\partial_{acc}$(balance)=$\mathbb{Z}_\perp$[†] , $E_{acc}$= {open}$\cup${withdrawal(i),deposit(i):i$\in$integer}. The set of states is $\mathbb{Z}_\perp$. The set of quiescent states is also $\mathbb{Z}_\perp$. The set of initial states is $\{\perp\}$. We have a unique valuation function:

$$\mu_{acc,balance} = \lambda s.\{s\}$$

and transition functions are given as follows:

$$\delta_{acc,open} = \lambda s.\text{if } s=\perp \text{ then } \{1000\} \text{ else } \emptyset$$

$$\delta_{acc,deposit(i)} = \lambda s.\text{if } s=\perp \text{ then } \emptyset \text{ else } \{s+i\}$$

$$\delta_{acc,withdrawal(i)} = \lambda s.\text{if } s=\perp \text{ then } \emptyset \text{ else } \{s-i\}$$

It is now possible to establish a suitable *category of objects* by choosing an appropriate notion of *object morphism*.

---

† By $\mathbb{Z}_\perp$ we mean the set of integers plus the distinguished element $\perp$.

***Definition 2.3***. A *slot morphism* $f:<A_1,\partial_1>\to<A_2,\partial_2>$ is a total map $f_A:A_1\to A_2$ such that $\partial_1(a)=\partial_2(f_A(a))$, for every $a\in A_1$. ◇

***Proposition 2.4***. The slot alphabets and slot morphisms constitute a complete and cocomplete category Slt (called the *category of slots*). ◇

Now, we want to ensure that if we extend a given object with more slots and events then the additional events should have no effects on the old slots (*no side-effects*). Clearly, that should be the case when an object is part of a composite object: the events of the other parts should not affect within the whole the slots of the part. The same is to be expected in the case of a strict form of specialization without side-effects: the additional events of the specialization should not affect the inherited slots. Finally, in both cases, the effects of original events on the original slots should be the same in the extended object (*no overriding*). Assuming a *frame constraint* (no side-effects and no overriding) we arrive at the following definition:

***Definition 2.5***. A *frame morphism*

$$h:<E_1,<A_1,\partial_1>,S_1,Q_1,B_1,\mu_1,\delta_1>\to<E_2,<A_2,\partial_2>,S_2,Q_2,B_2,\mu_2,\delta_2>$$

is a triple $<h_E,h_A,h_S>$ where $h_E:E_2\to E_1$ is a partial function, $h_A:A_1\to A_2$ is a slot morphism and $h_S:S_2\to S_1$ is a total function such that the following conditions hold: (a) $Q_2\subseteq h_S^{-1}(Q_1)$, (b) $B_2\subseteq h_S^{-1}(B_1)$, (c) $\mu_{1,a}(h_S(s))\subseteq\mu_{2,h_A(a)}(s)$, for every $a\in A_1$, $s\in S_2$, and (d) for every $e\in E_2$, $s\in S_2$, if $h_E(e)\downarrow$ then $\delta_{2,e}(s)\subseteq h_S^{-1}(\delta_{1,h_E(e)}(h_S(s)))$ else $\delta_{2,e}(s)\subseteq h_S^{-1}(h_S(\{s\}))$. ◇

From now on, when $h_E(e)\uparrow$ we write $\delta_{h_E(e)}$ for the function $\delta_\perp:s\$@\{s\}$. We also write $\delta_{2,e}\subseteq h_S^{-1}\delta_{1,h_E(e)}h_S$ for $\delta_{2,e}(s)\subseteq h_S^{-1}(\delta_{1,h_E(e)}(h_S(s)))$ for every $e\in E_2$ and for every $s\in S_2$.

Since the morphism condition should reflect the envisaged frame constraint, the reader might expect equality instead of inclusion. However, that would be too strong. The resulting category would not be cocomplete. It is easier to work with the more relaxed notion of morphism and recapture the frame constraint by the interplay between the morphism constraint (inclusion) and the universal property of the colimits as explained below in the case of object aggregation.

***Proposition 2.6***. Objects and frame morphisms constitute a cocomplete, concrete category FrOb over $Pfn^{op}\times Slt\times Set^{op}$. ◇

***Definition 2.7***. Let $ob=<E,<A,\partial>,S,Q,B,\mu,\delta>$ be an object. *Menu* of ob is the total function $M_{ob}:S\to\mathbf{2}^E$ defined by $M_{ob}=\lambda s.\{e\in E:\delta_e(s)\neq\emptyset\}$. ◇

Clearly, $M_{ob}(s)$ is the set of enabled events at s. Starting with a given intial state we may establish a life-cycle $\lambda$ of the object by successively picking up an enabled event and determining the subsequent state using $\delta$ until a quiescent state is reached. We might call the set of all such life-cycles the process induced by the object, recovering the model used e.g. in [Costa *et al* 92]. As we shall see in Section 4, as expected, coproducts in FrOb do correspond to parallel composition of the induced processes.

## 3.Subtyping

At this point we are ready to *start* the discussion of what is the meaning of ob' *is-a* ob, that is, ob' *is a specialization of* ob. Intuitively, we would like it to be: ob' can be taken as ob if we forget a few details. For instance, a savings account is an account in this sense. The former may have a few more slots and events, more specific behaviour and valuation, but it should be possible to forget all that and recognize a simple account. Therefore, a *strict is-a* relationship should correspond to a frame object morphism in the opposite direction. In this strict sense, ob' *is-a* ob iff there is a *frame inclusion morphism*[†] from ob into ob'. We classify this relationship as strict because as we shall see it disallows both side-effects and overriding. For instance, we would say that a savings account sav_acc *is (strictly) an* acc iff there is a frame inclusion from acc into sav_acc: $E_{sav\_acc} = \{open, new\_day, start\_saving, end\_saving\} \cup \{withdrawal(i), deposit(i): i \in integer\}$, $A_{sav\_acc} = \{balance, saving?, days\}$, $S_{sav\_acc} = \mathbb{Z}_\perp \times \mathbb{Z}_\perp \times \mathbb{Z}_\perp$, $Q_{sav\_acc} = \mathbb{Z}_\perp \times \mathbb{Z}_\perp \times \mathbb{Z}_\perp$, $B_{sav\_acc} = \{<\perp,\perp,\perp>\}$. With respect to behaviour, the morphism requirement is very simple and natural: any possible menu at a given state of sav_acc should degenerate into a submenu of acc at the projected state when we forget the specific events. Note that we are free to impose further constraints on the behaviour of sav_acc even with respect to the events inherited from acc. For instance, we might impose that if the sav_acc is in the saving mode (i.e., if saving?=1) then no withdrawals are allowed (only deposits). But, in order to illustrate another aspect discussed below, we prefer instead to allow a withdrawal to happen even when acc is in the saving mode and consider that in that event the mode is changed back to non-saving. The slot days counts the number of days since the beginning of the saving period and is not changed when sav_acc is not in the saving mode. The slot saving? indicates if acc is in the saving mode (saving?=1) or not (saving?=0). Thus[††] :

$$\mu_{sav\_acc,balance} = \lambda s.\{\pi_1(s)\}$$

$$\mu_{sav\_acc,saving?} = \lambda s.\{\pi_2(s)\}$$

$$\mu_{sav\_acc,days} = \lambda s.\{\pi_3(s)\}$$

$$\delta_{sav\_acc,open} = \lambda s.\text{if } s=<\perp,\perp,\perp> \text{ then } \{<1000,0,\perp>\} \text{ else } \varnothing$$

$$\delta_{sav\_acc,deposit(i)} = \lambda s.\text{if } s=<\perp,\perp,\perp> \text{ then } \varnothing \text{ else } \{<\pi_1(s)+i,\pi_2(s),\pi_3(s)>\}$$

$$\delta_{sav\_acc,withdrawal(i)} = \lambda s.\text{if } s=<\perp,\perp,\perp> \text{ then } \varnothing \text{ else } \{<\pi_1(s)-i,0,\pi_3(s)>\}$$

$$\delta_{sav\_acc,start\_saving} = \lambda s.\text{if } s=<\perp,\perp,\perp> \text{ then } \varnothing \text{ else } \{<\pi_1(s),1,0>\}$$

$$\delta_{sav\_acc,end\_saving} = \lambda s.\text{if } s=<\perp,\perp,\perp> \text{ then } \varnothing \text{ else } \{<\pi_1(s),0,\pi_3(s)>\}$$

$$\delta_{sav\_acc,new\_day} = \lambda s.\text{if } s=<\perp,\perp,\perp> \text{ then } \varnothing \text{ else } \{<\pi_1(s),\pi_2(s),\pi_3(s)+1>\}$$

---

† By frame inclusion morphism f from ob into ob' we mean a morphism induced by an inverse of an inclusion on events, an inclusion on slots and a surjective map on states.

†† We use $\pi_i$ for the i-th projection of a product.

Note that the original events of account are allowed to change the specific slots of sav_acc — namely withdrawals may change the extra slots. On the contrary, the new events of sav_acc do not affect the values of the old slots inherited from acc (no side-effects!) because of the frame morphism constraint on functional objects. *E.g.*,

$$\pi_1(\delta_{sav\_acc,start\_saving}(s))=\delta_{acc,\pi_1(start\_saving)}(\pi_1(s))=\{\pi_1(s)\}$$

and this equalily reads as follows: the set of sucessor states of s in sav_acc are all mapped onto the state $\pi_1(s)$; thus the balance at state s ($=\pi_1(s)$) coincide with the balance at any state belonging to $\delta_{sav\_acc,start\_saving}(s)$.

Moreover, the effects of the old events on the old slots inherited from acc are the same in sav_acc (no overriding!) again because of the frame morphism constraint on functional objects. Eg, if $s\in S_{sav\_acc}$ is different from $<\perp,\perp,\perp>$, then, we have $\{<\pi_1(s)+i>\}= \delta_{acc,deposit(i)}(\pi_1(s))=\pi_1(\delta_{sav\_acc,deposit(i)}(s))=\pi_1(<\pi_1(s)+i,\pi_2(s),\pi_3(s)>)=\{<\pi_1(s)+i>\}$.

This is a strict version of the *is-a* relationship. We shall provide more liberal versions in a subsequent section of this paper.

## 4.Interaction

We just saw that frame morphisms explain only a rather strict form of specialization: subtyping. Therefore, for inheritance purposes we need to consider other more general morphisms. However, the frame morphisms are precisely what is needed to explain object aggregation! Indeed, the slots of an aggregation component should not be affected by the events of the other parts (no side-effects!). And the effects of the events of an aggregation component should be the same in the whole (no overriding!).

Therefore, we are now ready to characterize object *aggregation* (without and with interaction) using some colimits in the category FrOb, keeping in mind that wrt behaviour it should correspond to parallel composition. For instance, binary coproducts provide the disjoint aggregation of the argument objects: disjoint parallel composition of behaviours encoded within the product of transition functions, plus addition of valuation maps.

As an illustration consider the coproduct of two bank accounts (two isormorphic copies of acc): $acc_1||acc_2$. Note that we are extending the traditional notation for parallel composition of processes to object aggregation, since they are so closely related. As expected these two independent objects can be put together without affecting each other. Clearly, there must be a frame injection morphism from $acc_i$ into $acc_1||acc_2$, for i=1,2.

Now consider a client cl owning an account acc: $E_{cl}=\{open\_acc,close\_acc\}\cup \{become\_client(i),withdrawal(i),deposit(i):i\in integer\}$, $A_{cl}=\{number\}$, $S_{cl}=\mathbb{Z}_\perp$, $Q_{cl}=\mathbb{Z}_\perp$, and $B_{cl}=\{\perp\}$. We also have:

$\mu_{cl,number} = \lambda s.\{s\}$

$\delta_{cl,become\_client(i)} = \lambda s.\text{if } s=\perp \text{ then } \{i\} \text{ else } \emptyset$

$\delta_{cl,open\_acc} = \lambda s.\text{if } s=\perp \text{ then } \emptyset \text{ else } \{s\}$

$\delta_{cl,close\_acc} = \lambda s.\text{if } s=\perp \text{ then } \varnothing \text{ else } \{s\}$

$\delta_{cl,withdrawal(i)} = \lambda s.\text{if } s=\perp \text{ then } \varnothing \text{ else } \{s\}$

$\delta_{cl,deposit(i)} = \lambda s.\text{if } s=\perp \text{ then } \varnothing \text{ else } \{s\}$

We want to establish their joint behaviour in the presence of some interaction. For instance, the account is opened by the client. Assuming that the objects acc and cl are as specified before, we would like to impose the following set of interaction (in)equations:

| | | | | |
|---|---|---|---|---|
| ρ = { | cl.open_acc | *calls* | acc.open, | |
| | cl.deposit(i) | *calls* | acc.deposit(i), | |
| | cl.withdrawal(i) | *calls* | acc.withdrawal(i), | |
| | cl.close_acc | *calls* | acc.close | } |

That is, open, deposit, close and withdrawal actions are to be related in some way by the two objects. The aggregation $cl\|_\rho acc$ displays the joint behaviour of these two interacting objects. Even in such a case of interacting components, each component *is-part-of* the whole. For instance, acc *is-part-of* $cl\|_\rho acc$.

Another form of interaction can be achieved by *slot sharing*. However it should be stressed that, in order to obtain the envisaged joint behaviour, all the events affecting the slot must also be shared. That is, it is only possible to share whole objects. This seems to be essential to object orientation.

***Proposition 4.1***. The coproduct of objects $ob_1=\langle E_1,\langle A_1,\partial_1\rangle,S_1,Q_1,B_1,\mu_1,\delta_1\rangle$ and $ob_2=\langle E_2,\langle A_2,\partial_2\rangle,S_2,Q_2,B_2,\mu_2,\delta_2\rangle$ is, up to an isomorphism, a 2-sink with vertex $ob_1\|ob_2=\langle E_1\times_{Pfn}E_2,\langle A_1+_{Set}A_2,[\partial_1,\partial_2]\rangle,S_1\times_{Set}S_2,Q_1\times_{Set}Q_2,B_1\times_{Set}B_2,\mu_1+\mu_2,\delta_1\times\delta_2\rangle$, where $[\partial_1,\partial_2]$ is the universal arrow from the coproduct $A_1+_{Set}A_2$ into the set of carriers V, $(\mu_1+\mu_2)_a = \lambda s{:}S_1\times_{Set}S_2.\cup_{x\in inj1_A{}^{-1}(a)}\mu_{1,x}(\pi_1(s))\cup\cup_{x\in inj2_A{}^{-1}(a)}\mu_{2,x}(\pi_2(s))$ and $(\delta_1\times\delta_2)_e = \lambda s{:}S_1\times_{Set}S_2.\delta_{1,e}(\pi_1(s))\times_{Set}\delta_{2,e}(\pi_2(s))$. ◇

The aggregation (coproduct) of two objects induces the parallel composition of their induced processes. Indeed, the product of menus leads to the interleaving of the life-cycles as expected, *i.e.*, $M_{ob_1+_{FrOb}ob_2}(s)=M_{ob_1}(\pi_1(s))\times_{Pfn}M_{ob_2}(\pi_2(s))$. Therefore, the event set of $ob_1\|ob_2$ should correspond to the categorial product in Pfn of the event sets of $ob_1$ and $ob_2$. An element $a|b$ of $M_{ob_1}(\pi_1(s))\times_{Set}M_{ob_2}(\pi_2(s))$ stands for concurrent execution of $a$ by $ob_1$ and $b$ by $ob_2$. An element $a$ of $M_{ob_1}(\pi_1(s))$ stands for the execution of $a$ by $ob_1$ while $ob_2$ remains idle. And an element $b$ of $M_{ob_2}(\pi_2(s)))$ stands for the execution of $b$ by $ob_2$ while $ob_1$ remains idle. That is, the object $ob_1\|ob_2$ would be able, at each instant, to be involved either in an event of $ob_1$ (while $ob_2$ remains idle), or in an event of $ob_2$ ($ob_1$ remaining idle), or in both an event of $ob_1$ and one of $ob_2$.

Some concepts of category theory are useful in order to provide the required techniques for object synchronization. Here we follow the technique of fibrations used in [Winskel 89].

***Proposition 4.2***. The forgetful functor $\upsilon$:FrOb$\rightarrow$Pfn$^{op}$ that maps each object onto its set of events is a cofibration. ◇

For each morphism f:$Z_2 \rightarrow Z_1$ in Pfn, the cofibration $\upsilon$:FrOb$\rightarrow$Pfn$^{op}$ induces a canonical functor $f^{>}:\upsilon^{-1}(Z_1)\rightarrow\upsilon^{-1}(Z_2)$ from the fiber over $Z_1$ into the fiber over $Z_2$.

***Definition 4.3***. An *interaction structure* is a triple $<E_1,E_2,«>$ where $E_1$ and $E_2$ are sets, and $« \subseteq E_1 \times E_2 \cup E_2 \times E_1$ is a one to one binary relation, called the *interaction relation.* ◇

Each interaction structure $<E_1,E_2,«>$ determines a subset E of the categorial product $E_1 \times_{Pfn} E_2$ (=$E_1 +_{Set} E_1 \times_{Set} E_2 +_{Set} E_2$) in Pfn given by

$$E=\{e \in E_1 \times_{Pfn} E_2 \text{: } (\pi_1(e) \notin \text{dom}(«) \text{ and } \pi_2(e) \notin \text{dom}(«)) \text{ or } \pi_1(e)«»\pi_2(e)\}^{\dagger},$$

where dom denotes the domain of «, and an inclusion map $\subseteq_{«}:E\rightarrow E_1 \times_{Pfn} E_2$. This means that if *a«b* then the happening of *a* leads to the happening of *b* — *a* is not allowed to happen alone or even to happen with any other event different from *b*.

An interaction structure endowed with a symmetric interaction relation is said to be a *sharing structure* and the underlying relation is called a *sharing relation*. Otherwise they are said to be a *calling structure* and a *calling relation*, respectively.

***Definition 4.4***. Given two objects $ob_1$ with events in $E_1$ and $ob_2$ with events in $E_2$, and an interaction relation «, we define $ob_1 \|_{«} ob_2 = (\subseteq_{«})^{>}(ob_1+ob_2)$. ◇

# 5.Liberal Specializations

In order to allow side-effects we propose the following definition of object morphism (relaxing the frame constraint):

***Definition 5.1***. An *object morphism*

$$h:<E_1,<A_1,\partial_1>,S_1,Q_1,B_1,\mu_1,\delta_1>\rightarrow<E_2,<A_2,\partial_2>,S_2,Q_2,B_2,\mu_2,\delta_2>$$

is a triple $<h_E,h_A,h_S>$ where $h_E:E_2\rightarrow E_1$ is a partial function, $h_A:A_1\rightarrow A_2$ is a slot morphism and $h_S:S_2\rightarrow S_1$ is a total function such that the following conditions hold: (a) $Q_2 \subseteq h_S^{-1}(Q_1)$, (b) $B_2 \subseteq h_S^{-1}(B_1)$, (c) $\mu_{1,a}(h_S(s)) \subseteq \mu_{2,h_A(a)}(s)$, for every $a \in A_1$, $s \in S_2$, and (d) for every $e \in E_2$, $s \in S_2$, if $h_E(e)\downarrow$ then $\delta_{2,e}(s) \subseteq h_S^{-1}(\delta_{1,h_E(e)}(h_S(s)))$. ◇

***Proposition 5.2***. Objects and object morphisms constitute a complete and cocomplete, concrete category Ob over Pfn$^{op}\times$Slt$\times$Set$^{op}$. Moreover, FrOb is a wide subcategory of Ob. ◇

Fortunately, Ob is also finitely complete (limits within Ob are also important for overriding as we shall see).

† By $\pi_1(e)«»\pi_2(e)$ we mean $\pi_1(e)«\pi_2(e)$ or $\pi_2(e)«\pi_1(e)$.

For instance, assume that we would like to have a fee (say of 50) being automatically taken from a savings account whenever a saving period starts (side-effect of start_saving on balance). We would like to impose

$$\delta_{sav\_acc,start\_saving} = \lambda s.\text{if } s{=}{<}\bot,\bot,\bot{>} \text{ then } \emptyset \text{ else } \{{<}\pi_1(s)\text{-}50,\{1\},\{0\}{>}\}$$

The existence of an *inclusion* in Ob from acc into sav_acc tells us that if we choose a slot $a$ in $A_{acc}$ and an inherited event $e$ in $E_{sav\_acc}$ then $\delta_{acc,e}\pi_1=\pi_1\delta_{sav\_acc,e}$. However, nothing is said about what happens if we choose a new event of $E_{sav\_acc}$. Thus morphisms in Ob are particularly suitable for providing the semantics of inheritance (side-effects being possible).

With respect to behaviour, the morphism condition remains as discussed in FrOb: any possible *menu* of sav_acc should degenerate into a possible *menu* of acc when we forget the new events of sav_acc.

Morphisms of Ob are liberal enough to explain specialization with side-effects. But they completely disallow non-monotonic overriding in the sense that the effects of inherited events on inherited slots are not changed at all. In order to cope with non-monotonic overriding we need to consider even more relaxed morphisms. Since the effects in the specialization may be rather different from those in the original object, a decoupling is needed between the two objects, suggesting the use of partial morphisms. Indeed, a partial morphism from ob into ob' is a morphism from a subobject of ob into ob'. That subobject characterizes what is kept from ob in ob'. The rest may be changed (overriden).

Let M be a class of morphisms in category A. A 2-source ($X \xleftarrow{m} \bullet \xrightarrow{f} Y$) with $m \in M$ is called an M-partial morphism from X into Y.

***Definition 5.3***. Let M be the class of $Pfn^{op} \times Slt \times Set^{op}$ identity morphisms. A 2-source in Ob, $ob_1 \xleftarrow{\iota} x \xrightarrow{f} ob_2$, with $\iota \in M$ is called a *partial morphism* with source $ob_1$ and target $ob_2$. ◇

For a partial morphism $ob_1 \xleftarrow{\iota} x \xrightarrow{f} ob_2$ within Ob we just write $ob_1 \leftarrow x \xrightarrow{f} ob_2$ or even $f{:}ob_1 \nrightarrow ob_2$.

$$\begin{tikzcd}
 & & ob_{12} \times_{ob_2} ob_{23} \arrow[dl] \arrow[dr, "f"] & & \\
ob_1 & ob_{12} \arrow[l] \arrow[dr, "f"'] & pullback & ob_{23} \arrow[dl] \arrow[r, "g"] & ob_3 \\
 & & ob_2 & &
\end{tikzcd}$$

**Fig. 5.1** Composition of partial morphisms.

Let us now define *composition of partial morphisms*. But first recall that in every category $x_1 \leftarrow x_1 \xrightarrow{f} x_2$ is a pullback of $x_1 \xrightarrow{f} x_2 \leftarrow x_2$ and $x_1 \rightarrow x_1 \xleftarrow{f} x_2$ is a pushout of

$x_1 \xleftarrow{f} x_2 \to x_2$. Thus, if $ob_1 \leftarrow ob_{12} \xrightarrow{f} ob_2$ and $ob_2 \leftarrow ob_{23} \xrightarrow{g} ob_3$ are partial morphisms within Ob, then a pullback of $ob_{12} \xrightarrow{f} ob_2 \leftarrow ob_{23}$ can be obtained by providing the initial lifting of the source [see Adámek *el al* 90]

$$<E_{12},A_{12},S_{12}> \leftarrow <E_{12},A_{12},S_{12}> \xrightarrow{f} <E_{23},A_{23},S_{23}>$$

in $Pfn^{op} \times Slt \times Set^{op}$. Let $ob_{12} \times_{ob2} ob_{23}$ be the vertex of the corresponding initial source in Ob.

***Definition 5.4***. Composition of two partial morphisms $ob_1 \leftarrow ob_{12} \xrightarrow{f} ob_2$ and $ob_2 \leftarrow ob_{23} \xrightarrow{g} ob_3$ is the partial morphism $ob_1 \leftarrow ob_{12} \times_{ob2} ob_{23} \xrightarrow{g \circ f} ob_3$. ◇

***Proposition 5.5***. Objects and partial morphisms constitute a category POb over $Pfn^{op} \times Slt \times Set^{op}$.

*Proof*: Partial morphisms compose and this composition is associative. Identity morphisms are 2-sources of the kind $ob \xleftarrow{id} ob \xrightarrow{id} ob$. ◇

We already saw an example of a rather restricted form of overriding: the withdrawal event of the acc object is redefined to some extent in the sav_acc object. Indeed, in the latter an withdrawal has more effects than in the former — namely, it also affects the values of the additional slot saving?.

The semantics of specialization given above within Ob does allow for this kind of event overriding: an event when inherited may have further effects, but the effects on the inherited slots must remain the same. We could say that we have so far only *monotonic* overriding. Any stronger form of specialization will also require a more liberal overriding mechanism, allowing, for instance, a different treatment of the inherited slots by the inherited events in the specialized object. In order to provide the semantics of such a *non-monotonic overriding* mechanism we have to consider morphisms in POb.

A suitable partial morphism h:acc$\not\to$sav_acc introduces a *subobject* acc' of acc that coincides with acc with the exception of the map $\delta_{acc',e}$ corresponding to the method we may want to override. In acc' we must impose $\delta_{acc',e}=\lambda s.S_{acc'}$ for those maps: then for all slots *a* of acc, the maps $\delta_{acc,a}$ and $\delta_{sav\_acc,a}$ are no longer necessarily the same. Therefore, if sav_acc *is-an* acc, that is, if there is a partial morphism h:acc$\not\to$sav_acc, then we may have method overriding: whenever some rules of acc are to be rewritten in sav_acc, the partial morphism $acc \leftarrow acc' \xrightarrow{h} sav\_acc$ should be chosen so that $\delta_{acc',e}= \lambda s.S_{acc'}$ in acc' for each method *e* to be overridden.

It should be stressed that both source and target of a partial morphism can be deterministic. However, the auxiliary middle object fails to be deterministic! In practice, we can go on defining only deterministic objects (and imposing configurations based upon inclusions and injections of Ob) if we disregard such auxiliary objects.

## 6.Further Work

In this paper we do not distinguish between objects and aspects, contrarily to what is proposed in [Ehrich *et al* 92]. Therefore, we ignore the problem of having the same object with different aspects (*i.e.*, the same savings account x with several aspects, such as x.ACCOUNT and x.SAVINGS_ACCOUNT). According to the terminology in [Ehrich *et al* 92], we may look at this paper as a study of object template inheritance and aggregation. An immediate research concern is to bring the results presented herein to bear on a suitable framework of object identifiers, aspects and classes.

## Acknowledgements

This work was partially supported by the JNICT Project PMCT/C/TIT/178/90 (FAC3) and by the ESDI Project OBLOG. The authors are grateful to their colleagues in these two projects for many rewarding interactions.

## References

[Adámek *et al* 90]
J.Adámek, H.Herrlich and G.Strecker, *Abstract and Concrete Categories*, Wiley, 1990

[Costa *et al* 92a]
J.F.Costa, A.Sernadas and C.Sernadas, "Object Inheritance Beyond Subtyping", INESC Report, 1992, to appear in *Acta Informatica*

[Ehrich *et al* 91]
H.-D.Ehrich, J.Goguen and A.Sernadas, "A Categorial Theory of Objects as Observed Processes", in J.W. deBakker, W.P.deRoever and G.Rozenberg (eds), *Proc. of the REX90/Workshop on Foundations of Object-Oriented Languages*, LNCS 489, Springer-Verlag, 1991, 203-228

[Ehrich *et al* 92]
H.-D.Ehrich, G.Saake and A.Sernadas, "Concepts of Object-Orientation", to appear in *Proc. of the 2nd IS/KI Workshop*, Ulm, 1992

[Goguen 91]
J.Goguen, "Types as Theories", *Proc. Conf. on Topology and Category Theory in Computer Science*, Oxford University Press, 1991, 357-390

[Jungclaus *et al* 91]
R.Jungclaus, G.Saake, T.Hartmann and C.Sernadas, *Object-Oriented Specification of Information Systems: The TROLL Language*, Informatik-Berichte, Tech. Univ. Braunschweig, 1991, *to appear*

[Manna and Pnueli 81]
Z.Manna and A.Pnueli, "The Temporal Framework for Concurrent Programs", in R.Boyer and J.Moore (eds), *The Correcteness Problem in Computer Science*, Academic Press, 1981, 215-274

[SernadasA *et al* 89]
A.Sernadas, J.Fiadeiro, C.Sernadas and H.-D.-Ehrich, "Basic Building Blocks of Information Systems", in E.Falkenberg and P.Lindgreen (eds), *Information System Concepts: An In-depth Analysis*, North-Holland, 1989, 225-246

[Winskel 84]
G.Winskel, "Synchronization trees", in *Theoretical Computer Science* 34, 1984

[Winskel 89]
G.Winskel, "An Introduction to Event Structures", in J.W.deBakker, W.P.de Roever and G.Rozenberg (eds), *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, Springer-Verlag, 1989, 364-397