South African Computer Journal
Number 23
July 1999

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 23
Julie 1999

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarvereniging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

World-Wide Web: `http://www.cs.up.ac.za/sacj/`

Subscriptions

	Annual	Single copy
Southern Africa	R80.00	R40.00
Elsewhere	US$40.00	US$20.00

An additional US$15 per year is charged for airmail outside Southern Africa

to be sent to:

Computer Society of South Africa
Box 1714, Halfway House, 1685
Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

Guest Editorial
Computer Science and Information Systems: The Future?

Philip Machanick

Department of Computer Science, University of the Witwatersrand, South Africa
philip@cs.wits.ac.za

1 Introduction

As president of the South African Institute for Computer Scientists and Information Technologists (SAICSIT), I have visited a number of campuses and companies, in an attempt at arriving at a general assessment of the state of our subjects in South Africa.

An issue which I consistently pick up is that while everyone seems to think that computer-related skills are extremely important and in short supply, our academic departments are also extremely under-resourced.

At the last Southern African Computer Lecturers Association (SACLA) conference (28-29 June, Golden Gate), I had the opportunity to discuss the problems other academics see. This editorial lists some of the problems reported at SACLA, and proposes a way forward.

2 Problems

At SACLA, I led a discussion of problems seen in our academic departments.

There was wide agreement that both Computer Science (CS) and Information Systems (IS) departments were under pressure to increase student numbers (massification), and were seen as cash cows to prop up less popular subjects. It was broadly agreed that staffing was a critical issue: too few posts for the workload, salaries way out of line with industry (half or less, as compared to the US, where an academic salary may be 80% of an industry salary). Recent graduates often make more than professors which makes it hard to persuade our students to become academics (even to do higher degrees). Attracting a recent PhD with a sense of adventure is may be possible, but attracting experienced people used to earning a salary in a strong currency is hard. IS jobs are worse than CS, as the skills required are more like those in business. Support staff salaries are an even harder issue: their skills relate even more directly to job descriptions in industry.

A problem in addressing our concerns is that we are so overworked that we don't have time for "politics": academics with no students have time on their hands, but we don't. More industry support not only with directly addressing problems but with taking on university administrations would be useful, but they too have major problems and don't have free time.

3 Solutions?

Solutions are harder to identify than problems.

The SACLA session ended with a proposal that we conduct surveys of our institutions and businesses, to find out what the problems are, as a starting point for going to university administrations, government and business.

Another idea was to attempt to find common cause with business in taking on problems they have in common with academia, including the skills shortage, the insufficient capacity of our education system, and dealing with employment equity.

One of our biggest difficulties is to free up time to deal with issues such as resource allocation within our universities. The "competition" is frequently other academics with time on their hands, since they have too few students, and therefore are in a position to spend time looking after their interests.

What is needed now is some thought about how to pull ourselves out of the mess we are in. In particular, we need strategies to exploit our strengths: our high demand among students, the high demand for the skills we produce and the ubiquitous applicability of computer technology.

Given the wide use of computers, it would seem obvious that our areas should be strongly supported by a range of role players, yet the fact that so many different groups are interested in computer technology in one way or another has tended to fragment efforts to enhance our industry and academic institutions.

Clearly, from conversations I have held, some departments are in much better shape than others. Even so, some kind of collective effort is likely to achieve more results than if we allow ourselves to be pushed around as individuals. Addressing the fragmentation of efforts seems a worthy goal in itself, to reduce duplication and contradictory goals.

I appeal to anyone who has constructive ideas on how to take our subjects forward to contact me. Let us work on building ourselves up. The economy depends on us, much more than on most other academic disciplines. It's time we made that point, and made it strongly.

SAICSIT'99

South African Institute of Computer Scientists and Information Technologists

Annual Research Conference 17-19 November 1999

Prepare for the New Millennium

Is there life after y2k?

Mount Amanzi Lodge, Hartebeespoort

near Johannesburg and Pretoria

keynote speaker: Barbara Simons, ACM President

and many other local and international speakers (academic and industry)

Call for Participation

papers in a many areas of Computer Science and Information Systems are expected

Price Waterhouse Coopers prizes: Best Paper R10000 • Best Student Paper R5000

please check the conference web site for accepted papers:

http://www.cs.wits.ac.za/~philip/SAICSIT/SAICSIT-99/

To Register

go to the conference registration web page:

http://www.cs.wits.ac.za/~philip/SAICSIT/SAICSIT-99/reservation.html

or contact **SAICSIT'99 Secretary** *for details:*

Department of Computer Science, Senate House 1137
University of the Witwatersrand
Jorissen Street
Wits, 2050
South Africa

phone (011)716-3309 fax 339-3513 (international: replace 011 by 27-11)

saicsit99-info@cs.wits.ac.za

Dates and Publication Details

early booking deadline: 14 September • on-site registration starts: 17 November 1999

workshops, tutorials 17 November 1999 • paper sessions: 18-19 November 1999

papers will appear in a special issue of South African Computer Journal

sponsors

NRF PRICEWATERHOUSECOOPERS

Research Article

A Formal Model for Objectbases

P.A. Patsouris[a], M. Korostenski[b] and V. Kissimov[c]

[a]*Department of Computer Science, University of the Witwatersrand, South Africa,* panos@cs.wits.ac.za
[b]*Department of Mathematics, University of the Witwatersrand, South Africa*
[c]*IBM Consultant*

Abstract

We introduce FMOB: a formal model for Objectbases. The term "Objectbase" represents better the widely used "object-oriented database". FMOB consists of (1) objects (classes and their instances) with extended encapsulation capabilities, (2) second-order objects (respectively 2-classes and 2-instances) for supporting the structural relations specialization, association, aggregation and grouping. Every 2-object encapsulates a two-level-hierarchy (t-l-h) of objects formally defined. The inheritance notion is generalized and replaced by the link concept, which allows, in a uniform way, the proper reusability of data, methods and other object-properties across one or more 2-object of the above type. (3) FMOB also supports complex objects through an appropriate composition of 2-objects. The model is based on a universal algebra of words and appropriate extensions. Every building block of the objectbase has its corresponding algebra. The algebra of abstract words is functionally complete. The modified greedy algorithm optimizes all linear objective functions over the search (or branching) greedoid defined on the underlying digraph of a second order-object.

Keywords: *FMOB (formal model for objectbases), universal algebra of words, functionally complete algebra, two-level hierarchy of objects, 2-class, 2-object, digraphs, accessible set-systems, greedoids.*
Computing Review Categories: *H.2.2, H.2.4*

1 Introduction

The Object-Oriented Systems are continuously gaining popularity as the most promising powerful "vehicles" for the design and implementation of complex real-world applications. The reason for this is the entirely different approach in modelling the real-world entities as *objects*. An object is characterized by the notions of the unique *object identity* [9], considered distinct from the object's contents, and of *encapsulation*. The latter is a property through which the data and the operations (acting on these data) reside in the object to which they belong. From this point of view, the "internal contents" of an object are totally hidden from its outside world. Thus, the behaviour of an object is captured in *messages* to which it responds [2], while its active nature allows it to send messages to other objects. Furthermore, the most essential features of object-orientation are the notions grouped under the general term: *"reusability mechanisms" (instantiation* and *inheritance)*. Inheritance is closely related to the notion of *class hierarchy* [2]. Since the mid 80's a considerable amount of research has been done on all aspects of object-orientation at different levels (semantic modelling, systems design, programming languages, databases) by unveiling its power and its perspectives and inspiring new researchers to carry on. Our main focus lies in the field of database technology (the term, generally used, refers to all existing database types including the object-oriented databases). The main problems in shifting from the classical approach (Codd's work) are found in Malcolm Atkinson et *al* [3]. In [3] it is stated that no analogous specification, with respect to Codd's work, exists for the object-oriented databases (OODBs). The field of OODBs has three weak points:

(a) the lack of a common data model,

(b) the lack of formal foundations and

(c) strong experimental activity.

We put strong emphasis on the point (b), since the need for a solid underlying theory is the only way to lead to consensus on what is the commonly accepted object model. That theory does not facilitate the experimental work which is underway (in object-oriented programming languages and OODBs), since there is no common denominator for comparisons and uniform evaluation.

However, the experimental activities provide a huge amount of feedback and relative freeness in exploring different prototypes (and ideas developed in them) but in that sense we inevitably follow the Darwinian approach [3] by hoping that from those prototypes a fit model will emerge. Further, in a commonly accepted approach, the lifting of the confusion caused by concepts coming from other disciplines (like *encapsulation* and *inheritance* originated in biological sciences) is considered as crucial. We should therefore, provide a crisp boundary between the terms object and data as well as a proper definition of what is an *Objectbase*. As in [6], Objectbases were initially called object-oriented databases. It has been pointed out that

the juxtaposition of "object" and "data" is essentially self-contradictory and the new name appears to be clearer and more appropriate. The requirement for development of an Objectase Management System (OBMS) can be done by following one of three approaches:

(1) extending an Object-Oriented Programming Language (OOPL) by adding persistence, data sharing control and a query facility,

(2) to rework relational DBMS to support object-oriented structures and

(3) to design an OBMS totally on object-oriented principles.

For a successful OBMS, five areas are considered of special interest:

(a) unique object identifiers,

(b) support for the object model,

(c) compatible database and programming language representations,

(d) efficient system performance and

(e) representation of ordered aggregates [4],[13].

An OBMS requires the interaction of an object manager (for the definition, management and manipulation of the objectbase), an object server for the management of the traffic whenever the objectbase is implemented across a number of sites) and an object store (for the actual persistent storage of objectbase's various information units like objects, their classes etc.). Three languages are conventionally required: an *Object Definition Language* for the definition of classes, an *Object Manipulation Language* for general manipulation and management of the object base and an *Object Control Language* to support the integrity constraints. However, in most cases of object-oriented databases, there exists an OOPL (or an appropriate extension of an OOPL) that plays all three roles above.

An understanding of what objectbases actually are is still evolving. Some of the features required for an OBMS are:

(1) *Transaction properties*,

(2) *Security Authorization*,

(3) *Query capability*

(4) *Data Independence/Schema Modification*,

(5) *Distributed Database Systems*,

(6) *Object Model* and

(7) *Architectural features*.

Further analysis of these features are listed in [6]. However we provide here additional details for (6):

In order to evaluate the existing object models of both OOPLs and existing OODBs, we also have to take into account the additional features of *Versioning* (referring to all the types of objects, i.e. classes or instances as well as simple or complex objects), *Persistence* (the mechanism through which every created object is permanently stored in the objectbase) and *Query facilities* i.e. the possibility to query

(i) the internal structure of one or more objects of the same class;

(ii) objects, 2-objects and complex objects.

Table 2 shows different important features of well known object-oriented database management systems. These features characterize the quality of the OODBMs and therefore, their underlying object models (cf. [8]).

2 THE FORMAL OBJECT MODEL

Our model is based on word algebras. In 2.2 we introduce two word algebras, namely the algebras of abstract/concrete words. In an implementation abstract words correspond to object-properties which correspond to parts of the internal structure of classes (abstract objects) and concrete words correspond to object-properties which correspond to parts of the internal structure of instances of classes. The basic operations of the abstract word algebra are used to model the standard objectbase operations (see 2.5: An algebra of objects)

2.1 Some Preliminaries

We here recall several basic concepts. Let X be a non-empty set and T an arbitrary set of (basic) operations. The concept of a *word* is defined inductively (cf.[11]:

(i) all elements of X and all symbols of nullary operations are words;

(ii) if $w_1,\ldots,w_n$ are words, then for an arbitrary n-ary operation $\tau \in \Lambda$, where $n \geq 1$, the formal expression $\tau w_1 \ldots w_n$ is also a word.

If $w = \tau w_1 \ldots w_n$ then $w_1,\ldots,w_n$ are called *subwords* of the word w. The property of being a subword is transitive. In particular all elements of X and symbols of nullary operations occurring in the expression for w are subwords of w.

The set of all possible words relative to a system of operations Λ and a set X can be regarded as a universal algebra with Λ the set of basic operations. We denote this *algebra of words* by $W_\Lambda(X)$.

FEATURES REQUIRED TO BE SUPPORTED BY AN OBJECT MODEL
Objects
Identity
Encapsulation
Polymorsphism
Classes
Inheritance
Non-traditional objects*
Aggregates
Composites or Complex Objects
Integrity (Structural/Functional)
Dynamic Schema Evolution
Database operation extensibility and Object Language

*Entities that require non-standard constituent elements like data/methods.

Table 1: Essential features to be supported by an object model

Object-Oriented DBMS (name)	Object Model	Persistence	Complex Objects	Version-ing	Query Facilities	Dynamic Schema Evolution
Gem Stone	Extension of Small talk's object model	Yes	No	No	Simple	No
Itasca	Extension of Common Lisp's object model	Yes	No	No	Simple	No
Mattise	Hybrid model	Yes	Yes*	Yes	Simple	Yes
O_2	Its own object model	Yes**	No	No	No	No
Objectivity /DB	Extension of C++'s object model	No	Yes†	No	No	No
Object Store	Improvement of C++'s object model	Yes	No	No	Yes	No
Ontos	Extension of C++'s object model	Yes	No	No	Object SQL‡	No
Open ODB	Data model	–	No	No	Simple	No
Statice	Common Lisp's object model	No	No	No	No	No
UniSQL	Extended rela-tion model	No	No	No	No	No
Versant	Extension of C++'s object model	Yes	No	No	Simple	No

*:but different nature than pure complex objects
**:distinction between persistent/transient objects
†:association ‡:strong support of object queries

Table 2: Comparative evaluation of object-oriented database, management systems' features

Before considering particular word algebras we recall the following definitions for an arbitrary universal algebra $\mathbf{A} = \langle A, F \rangle$: A *composition of operations* in F is the construction of an n-ary opera-

tion f from k given n-ary operations $f_1,\ldots,f_k$ and a k-ary operation g through the defining formula $f(a_1,\ldots,a_n) := g(f_1(a_1,\ldots,a_n),\ldots,f_k(a_1,\ldots,a_n))$. A *clone* on the set A is the set of operations on A that is closed under all compositions and contains the n-ary projections p_i^n. (for all n and i satisfying $1 \leq i \leq n$). The clone of *algebraic operations* on A denoted by $Clo\mathbf{A}$, is the clone on A generated by the basic operation and projections. Those operations in the clone on A generated by the basic and constant operations on A together with the projections are called *algebraic functions*. Often one obtains the algebraic functions from the algebraic operations by substituting some variables by constants. $\mathbf{A}$ is *functionally complete* if it is finite and all operations on A are algebraic functions.

An algebra $\mathbf{A} = \langle A, F\rangle$ is a *reduct* of the algebra $\mathbf{B} = \langle B, G\rangle$ if $A = B$ and $Clo\mathbf{A} \subseteq Clo\mathbf{B}$. For the basic notions of *subalgebra* of an algebra, of *homomorphism* or *isomorphism* between two similar algebras, of *congruence* of an algebra, of *quotient algebra*, and of *direct product* of a system of similar algebras see for example [5].

2.2 Particular Word Algebras

We now fix two alphabets Γ_a and Γ_c. The elements of Γ_a are called *abstract elements* and those of Γ_c *concrete elements*. We consider the following set of basic operations $T = \{\delta, 1, \triangleright, \triangleleft, \cap, -, \Diamond, \lrcorner\}$ and the set of all words Λ_a relative to T and Γ_a. The results of applying each of the operations in T to the elements of Λ_a are given in Table 3.

Remarks

1. we define $\triangleleft(w_1,\delta) := w_1$ and $\triangleleft(\delta,w_2) := w_2$. So, $w_1\delta \equiv w_1$ and $\delta w_2 \equiv w_2$ and as a result δ is not a subword of any word.

2. In the definitions of $\cap$ and $-$ the subwords $v_1,\ldots,v_n$ are obtained by "scanning" w_1 from left to right, e.g. if

$$w_1 = \lrcorner(\Diamond xx)(\triangleleft y_1 y_2)(\triangleright z)$$

and

$$w_2 = \triangleleft(\triangleright z)(\Diamond xx)$$

then

$$\cap w_1 w_2 = \triangleleft(\Diamond xx)(\triangleright z) = \triangleleft x\delta = x$$

and

$$-w_1 w_2 = \triangleleft y_1 y_2 = y_1 y_2$$

3. The ternary operation $\lrcorner$ is *Mal'cev* [7] i.e. $\lrcorner(w_1,w_1,w_2) = w2 = \lrcorner(w_2,w_1,w_1)$ for all $w_1, w_2 \in \Lambda_a$.

We denote the *algebra of abstract words* $W_T(\Gamma_a)$ by $\blacksquare_a$.

We show that Λ_a is functionally complete (Theorem 2.2.1), using a characterization of functionality complete algebras in terms of the ternary discriminator function $t(t(x,y,z) = z$ if $x = y$ and x if $x \neq y)$:
A finite algebra $\mathbf{A} = \langle A, F\rangle$ is functionally complete if and only if the ternary discriminatory is an algebraic function of A [17].

2.2.1 Theorem

The abstract word algebra $\blacksquare_a$ is functionally complete.

Proof

The ternary discriminator $t : \Lambda_a^3 \to \Lambda_a$ is an algebraic operation of A:

$$t(w_1,w_2,w_3) = t(\overline{w}) =$$
$$\lrcorner(p_1^3(\overline{w}), \Diamond(p_1^3(\overline{w}), p_2^3(\overline{w}), p_3^3(\overline{w})))$$

Since this expression, after replacing the projections with their results, becomes

$$\lrcorner(w_1, \Diamond(w_1,w_2)w_3) =$$

$$\begin{cases} \lrcorner(w_1,w_1,w_3) = w_3 \\ \text{if } w_1 = w_2 \\ \lrcorner(w_1,\delta,w_3) = w_1 \\ \text{if } w_1 \neq w_2 \end{cases} .\blacksquare$$

2.2.2 Corollary

$\blacksquare_a$ is simple, i.e. has only two congruences. $\blacksquare$

Let Γ_c be an alphabet disjoint from Γ_a and Λ_c be the set of all words relative to $T' = \{\delta, \triangleleft\}$ (δ a nullary operation and $\triangleleft : \Lambda_c^2 \to \Lambda_c$ defined in Table 3) and Γ_c. We denote the word algebra $W_{T'}(\Gamma_c)$ by $\blacksquare_c$ and call it the *algebra of concrete words.*

We now define a classification homomorphism between $\blacksquare_c$ and a reduct of $\blacksquare_a$ as follows: let $\blacksquare_a^{\mathcal{R}} = \langle \Lambda_a, T'\rangle$ and let $\varphi : \Gamma_c \to \Gamma_a$ be a given function.

The *classification homomorphism*

$$clf_\varphi : \blacksquare_c \to \blacksquare_a^{\mathcal{R}}$$

with respect to φ is defined as the unique homomorphic extension of φ, i.e.

$$clf_\varphi(w) = \begin{cases} w & \text{if } w \in \Gamma_c \\ \triangleleft clf_\varphi(w_1) clf_\varphi(w_2) & \text{if } w = \triangleleft w_1 w_2 \\ \varphi & \text{if } w = \delta \end{cases}$$

e.g. if

$$\Gamma_c = \{1, 2, 12, a, k, z\},$$
$$\Gamma_a = \{\text{integer, char}\}$$

and $\varphi : \Gamma_c \to \Gamma_a$ the "type-of" function. Then, for example,

Operation in T	Definition
• nullary operation: $\delta : \{\emptyset\} \to \Lambda_a$	$\delta(\emptyset) := \delta$
• unary operations $1 : \Lambda_a \to \Lambda_a$ $\triangleright : \Lambda_a \to \Lambda_a$	$1(w) := w$ $\triangleright(w) = \triangleright w := \delta$
• binary operations: $\triangleleft : \Lambda_a^2 \to \Lambda_a$	$\triangleleft(w_1, w_2) = \triangleleft w_1 w_2 := w_1 w_2$ (concatenation operation)
$\cap : \Lambda_a^2 \to \Lambda_a$	$\cap(w_1, w_2) = \cap w_1 w_2 :=$ { $\triangleleft v_1(\triangleleft v_2 \ldots (\triangleleft v_{n-1} v_n)\ldots)$where $v_1, \ldots, v_n$are the common subwords of w_1, w_2 ; δ otherwise
$- : \Lambda_a^2 \to \Lambda_a$	$-(w_1, w_2) = -w_1 w_2 :=$ { $\triangleleft v_1(\triangleleft v_2 \ldots (\triangleleft v_{n-1} v_n)\ldots)$where $v_1, \ldots, v_n$ are the subwords of w_1 that are not subwords of w_2 ; δ if $w_1 \neq w_2$
$\Diamond : \Lambda_a^1 \to \Lambda_a$	$\Diamond(w_1, w_2) = \Diamond w_1 w_2 :=$ { w_1 if $w_1 = w_2$; δ otherwise
• ternary operation: $\lrcorner : \Lambda_a^3 \to \Lambda_a$	$\lrcorner(w_1, w_2, w_3) =$ { $\lrcorner w_1 w_2 w_3$ is the word with every w_2 "replaced" by w_3 in w_1 if w_2 is a subword of w_1 ; w_1 otherwise

Table 3: Definitions of operations in T

$$
\begin{aligned}
&clf_\varphi(\triangleleft(\triangleleft((12)(a))(z))) \\
&= \triangleleft((\triangleleft clf_\varphi(12) clf_\varphi(a)) clf_\varphi(z)) \\
&= \varphi(12)\varphi(a)\varphi(z) \\
&= (\text{integer})(\text{char})(\text{char}) \\
&\equiv \text{'integer' 'char' 'char'}.
\end{aligned}
$$

Since $clf_\varphi : \blacksquare_c \to \blacksquare_a^{\mathcal{R}}$ is a homomorphism, $ker(clf_\varphi) = \{(w_1, w_2) \in \blacksquare_c^2 | clf_\varphi(w_1) = clf_\varphi(w_2)\}$ is a congruence of $\blacksquare_c$, and we have

$$\blacksquare_a^{\mathcal{R}} \cong \blacksquare_c / ker(clf_\varphi)$$

if clf_φ is onto.

2.3 The m-word Algebra

Our immediate aim is to describe an algebra where the elements of the carrier are objects (see 2.4, 2.5). In order to describe the internal structure of an object we introduce the notion of an m-word:

Consider n alphabets Γ_1, ..., Γ_n (not necessarily distinct) of abstract elements and the finite set T of basic operations as defined in 2.2. Let $\blacksquare_i = W(\Gamma_i), i = 1, \ldots, n$, and let $\mathbf{U}_a = \langle U_a = \prod_{i=1}^n \Lambda_i, T \rangle$, the direct product of the $\blacksquare_i$.

So, if τ is an m-ary operation in T and $u^1 = (u_1^1, \ldots, u_n^1), \ldots, u^m = (u_1^m, \ldots, u_n^m) \in U_a$ then

$$\tau(u^1, \ldots, u^m)(j) := \tau(u_j^1, \ldots, u_j^m).$$

The elements of U_a are called *ordered multi-origin abstract words* (or simply, *abstract m-words*) and $\mathbf{U}_a$ the *algebra of abstract m-words.*

Similarly, we define a *concrete m-word algebra* with T' (as in 2.2) the set of basic operations. This algebra is denoted by $\mathbf{U}_c$. We write δ^n for the abstract/concrete m-word $(\delta, \delta, \ldots, \delta)$.

If $u = (u_1, \ldots, u_n)$ and $u' = (u_1', \ldots, u_n') \in U_a(U_c)$ then u' is a *sub-m-word* of u if and only if u_j' is a subword of u_j in $\blacksquare_j$ for each $j = 1, \ldots, n$.

Let $\Gamma_1, \ldots, \Gamma_n$ be alphabets of abstract elements and $\Gamma_1', \ldots, \Gamma_n'$ alphabets of concrete elements. Suppose, for each $i = 1, \ldots, n$, we have $\varphi_i : \Gamma_i' \to \Gamma_i$, let $\blacksquare_i^{\mathcal{R}} = W_{T'}(\Gamma_i)$, and $\Lambda_2' = W_{T'}(\Gamma_i'), i = 1, \ldots, n$. We define a *classification homomorphism*, $clf : \mathbf{U}_c \to \mathbf{U}_a^{\mathcal{R}}$ (where $\mathbf{U}_c$ is the direct product of the $\blacksquare_i'$ and $\mathbf{U}_i^{\mathcal{R}}$ the direct product of the $\blacksquare_i^{\mathcal{R}}$) as follows: for $u = (u_1 \ldots, u_n) \in U_c$, $clf(u) := (clf_{\varphi_1}(u_1), \ldots, clf_{\varphi_n}(u_n))$.

2.4 Objects

An object is a pair $\langle O,u\rangle$ ($\equiv O[u]$) where O is an element of a given set $O^{(id)}$ of identifiers, called the *object-id*, and u is am m-word, called the *internal structure* of the object. The set of objects is denoted by $O_a^{(1)}$ if $u \in U_a$ and by $O_c^{(1)}$ if $u \in U_c$. Objects in $O_a^{(1)}$ are called *classes* and those in $O_c^{(1)}$ *instances*. Any $u \in U_a(U_c)$ can be considered as an object with nil object-id $\emptyset$, i.e. $u \equiv \emptyset[u]$. These objects are called *pre-objects* and denoted by $O_a^{(0)}\left(O_c^{(0)}\right)$. Thus $U_a \equiv O_a^{(0)} \subseteq O_a^{(1)}$ and $U_c \equiv O_c^{(0)} \subseteq O_c^{(1)}$.
By identifying each $O \in O^{(id)}$ with $O[\delta^n]$ we can consider $O^{(id)} \subseteq O_a^{(1)}\left(O_c^{(1)}\right)$.

Two objects $O_1[u^1]$ and $O_2[u^2]$ are called *equivalent* if $u^1 = u^2$. We write $O_1[u^1] \approx O_2[u^2]$.

2.4.1 Object Properties

The alphabets $\Gamma_1,\ldots,\Gamma_n$ are chosen in such a way that the coordinates of an m-word $u = (u_1,u_2,\ldots,u_n)$ correspond to certain object properties. We distinguish the following object-properties:

- The first word u_1 corresponds to the first object-property which consists of data definitions, each one being of a certain data type or of variables' values depending on whether the object is a class or an instance. We call this the *data object-property*.
- The second word u_2 corresponds to the second object-property which consists of functions, called *methods*, acting upon the values of the above-mentioned data types of the instance objects. This is called the *methods object-property*. Note that a method is a string specifying its name and number of parameters (as function headings are defined in most programming languages). The actual implementation of a method is not encapsulated in the object.
- The third word u_3 corresponds to the third object-property and contains the object-id of the class from which this object under consideration has been instantiated. The class of all classes is the single meta class under the name OBMS. The meta class is denoted by OBMS[obms]. Thus for every class $O[u]$ we write u_3 = OBMS. We call this the *class object-property*.
- The fourth word corresponds to the fourth object-property which consists of the (functional) relations that can be established between the different classes (respectively instances). The alphabet of that object-property includes functional relations [1] [12] [15] [16] of any (finite) arity established among the object identities. We call this the *relations object-property*.
- The fifth word corresponds to the fifth object-property and is defined as follows [15] [16]: The *Arc Data* is a special data type defining the inter-objects connections (see in next section). The elements of the alphabet has the form #O(F1F2F3) and is encapsulated in an object (called "taker") specifying
  - (i) an object O (called "giver") participating in the same hierarchical structure with the "taker",
  - (ii) the *scope of arc*, i.e. how many data, methods and relationships are inherited by the *filtering function* F1F2F3 for data (F1), methods (F2) and relationships (F3) respectively. F1F2F3 takes values from the set AAA, AAN, ANA, ANN, NNN where "A" stands for "All" and "N" stands for "None".
  - (iii) The *multiplicity* of the arc is defined by the *optional* prefix symbol "#". It is used only in links of class-two-level-hierarchies (see 2.6.1) and defines whether an instance of the "taker" can be linked simultaneously with more than one instance of the "giver".

The arcs between two objects constitute *links*. The links can be single (in this case there is only one "taker" and one "giver") or double (both objects are "takers" and "givers"). The double links are not necessarily similar in terms of their filtering functions and their multiplicity.

In practice not all the elements of $O^{(1)}$ are "active", in the sense that, at a given moment, not all objects are in the Objectbase. Those objects in the Objectbase are called *active*. OBMS[obms] and $\emptyset[\delta^n]$ are always active.

In Figure 1, we illustrate the object's structure.

2.4.2 Encapsulation and Instantiation Functions

We define two partial functions:

(1) **The encapsulation function**, enc: $O^{(id)} \times O_a^{(0)} \to O_a^{(1)}$. For $O[\delta^n] \in O^{(id)}$ and $\emptyset[u] \in O_a^{(0)}$ we have

$$\mathrm{enc}(O[\delta^n],\emptyset[u]) = \begin{cases} O[u] & \text{if } O \text{ is not in the object-id of an active class} \\ \emptyset[\delta^n] & \text{otherwise} \end{cases}$$

(2) **The instantiation function**, ins: $O^{(id)} \times O_c^{(0)} \times O_a^{(1)} \to O_c^{(1)}$. For $O[u] \in O^{(id)}, \emptyset[\delta^n] \in O_c^{(0)}, W[w] \in O_a^{(1)}$ and a given classification homomorphism clf_φ we have

$$\mathrm{ins}(O[\delta^n],\emptyset[u],W[w]) =$$

$$\begin{cases} O[u] & \text{if } O \text{ is not the object-id of an active object, } W[w] \text{ is an active class and } clf_\varphi(u) = w, \\ \emptyset[\delta^n] & \text{otherwise} \end{cases}$$

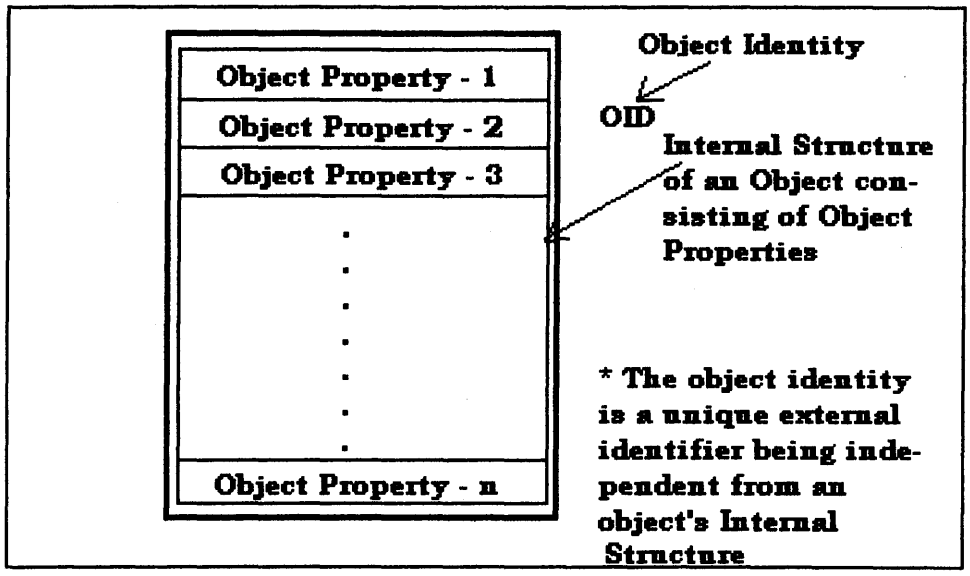

Figure 1: A schematic representation of an object

2.5 Algebras of Objects

We define using the operations T of the abstract word algebra, a set of basic operations $F^{(1)} = \{\emptyset[\delta^n], 1, Dlo, Amo, Ino, Dfo, Cmo, Upo\}$ on the set $O_a^{(1)}$ and then consider the algebra $\blacksquare_a^{(1)} = \langle O_a^{(1)}, F^{(1)} \rangle$, called the *algebra of abstract objects* or the *algebra of classes*.
A detailed description of each operation is given below:

1. Nullary operation
   $\emptyset[\delta^n] : \{\emptyset\} \to O_a^{(1)}, \emptyset[\delta^n](\emptyset) = \emptyset[\delta^n]$.

2. Unary operations

   - Identity 1

   - Delete object $Dlo : O_a^{(1)} \to O_a^{(1)}$

$$Dlo(O[u]) = \begin{cases} \emptyset[\delta^n] & \text{if } O[u] \text{ is a class with no instances} \\ O[u] & \text{otherwise} \end{cases}$$

3. Binary operations
   Add object to object $Amo : \left(O_a^{(1)}\right)^2 \to O_a^{(1)}$

$$Amo(O_1[u_1], O_2[u_2] =$$

$$\begin{cases} O_1[\lhd \delta^n u^2] & \text{if } u^1 = \delta^n \text{ and } O_2 = \emptyset \text{ (creating a new class)} \\ O_1[\lhd \delta^n u^2] & \text{if } u^1 = \delta^n,\ u^2 \neq \delta^n \text{ and } O_1 \neq \emptyset,\ O_2 \neq \emptyset \text{ (making an exact copy of the second object under the class-id } O_1) \\ O_1[\lhd u^1 u^2] & \text{if } u^1 \neq \delta^n \text{ and } O_2 = \emptyset \text{ (inserting properties in the first class's internal structure)} \\ \emptyset[\lhd u^1 u^2] & O_1 = O_2 = \emptyset \text{ (the operation is essentially the operation } \lhd \text{ of the } m\text{-word algebra)} \\ O_1[u^1] & \text{if } O_1[u^1] \approx O_2[u^2] \\ \emptyset[\delta^n] & \text{otherwise} \end{cases}$$

   - Intersection between objects
     $Ino : \left(O_a^{(1)}\right)^2 \to O_a^{(1)}$

$$Ino(O_1[u^1], O_2[u^2]) =$$

$$\begin{cases} O_1[u^1] & \text{if } O_1[u^1] = O_2[u^2] \text{ or } O_1[u^1] \approx O_2[u^2] \\ \emptyset[\cap u^1 u^2] & \text{if } O_1 = O_2 = \emptyset \\ \emptyset[\delta^n] & \text{otherwise} \end{cases}$$

   - Difference of objects
     $Dfo : \left(O_a^{(1)}\right)^2 \to O_a^{(1)}$

$$Dfo(O_1[u^1], O_2[u^2]) =$$

$$\begin{cases} \emptyset[-u^1 u^2] & \text{if } O_1 = O_2 = \emptyset \\ O_1[-u^1 u^2] & \text{if } O_1 \neq \emptyset,\ O_2 = \emptyset \\ \emptyset[\delta^n] & \textit{otherwise} \end{cases}$$

- Compare two objects
  $Cmo : \left(O_a^{(1)}\right)^2 \to O_a^{(1)}$

$$Cmo(O_1[u^1], O_2[u^2]) =$$

$$\begin{cases} O_1[u^1] & \text{if } O_1[u^1] = O_2[u^2] \\ & \text{or } O_1[u^1] \approx O_2[u^2] \\ \emptyset[\Diamond u^1 u^2] & \text{if } O_1 = O_2 = \emptyset \\ \emptyset[\delta^n] & \textit{otherwise} \end{cases}$$

4. Ternary operation
   Update object $Upo : \left(O_a^{(1)}\right)^3 \to O_a^{(1)}$

$$Upo(O_1[u^1], O_2[u^2], O_3[u^3]) =$$

$$\begin{cases} \emptyset[\lrcorner u^1 u^2 u^3] & \text{if } O_1 = O_2 = O_3 = \emptyset \\ O_1[u^1] & \text{if } O_1[u^1] = O_2[u^2] \\ & = O_3[u^3] \\ O_3[u^3] & \text{if } O_1[u^1] = O_2[u^2] \\ O_1[u^3] & \text{if } O_1[u^1] = O_2[u^2] \\ & \text{and } O_3 = \emptyset \\ O_1[\lrcorner u^1 u^2 u^3] & \text{if } O_1 \neq \emptyset, O_2 = O_3 = \emptyset \\ O_1[u^1] & \textit{otherwise} \end{cases}$$

Note that in the last case all the possible remaining combinations of classes which cannot give us any updating capability are included. It is easy to see that this operation is Mal'cev.

Two algebras $\mathbf{A} = \langle A, F_A \rangle$ and $\mathbf{B} = \langle B, F_B \rangle$ are *weakly isomorphic* if there exists an isomorphism $\varphi : A \to B$, and for each m-ary operation $f_A \in F_A$ and $\alpha_1, \ldots, \alpha_m$ in A, there is an m-ary operation $f_B \in F_B$ such that $\alpha(f_A(\alpha_1, \ldots, \alpha_m)) = f_B(\varphi(\alpha_1), \ldots, \varphi(\alpha_n))$. [5] [7].

2.5.1 Proposition

The m-word algebra $\mathbf{U}_a = \langle U_a, T \rangle$ and the algebra $\langle O_a^{(0)}, F^{(1)} \rangle$ are weakly isomorphic.

Proof

Define $\varphi : O_a^{(0)} \to U_a$ by $\varphi(\emptyset[u]) = u$. For each m-ary operation $f \in F^{(1)}$ there exists an operation of the same arity $\tau \in T$ such that $f(\emptyset[u^1], \ldots, \emptyset[u^m]) = \emptyset[\tau(u^1, \ldots, u^m]$.

Thus
$\varphi\left(f(\emptyset[u^1], \ldots, \emptyset[u^m])\right) = \tau\left(\varphi(\emptyset[u^1]), \ldots, \varphi(\emptyset[u^m])\right)$.■

2.5.2 Corollary

The m-word algebra $\mathbf{U}_a = \langle U_a, T \rangle$ is weakly embedded in the algebra $\blacksquare_a^{(1)} = \langle O_a^{(1)}, F^{(1)} \rangle$.■

The *algebra of concrete objects (algebra of instances)* is the universal algebra with carrier $O_c^{(1)}$ and set of basic operations $\{\emptyset[\delta^n], Amo\}$ where $\emptyset[\delta^n]$ is a nullary operation and $Amo : \left(O_c^{(1)}\right)^2 \to O_c^{(1)}$ is defined as above for abstract objects.

2.6 SECOND-ORDER OBJECTS (2-OBJECTS)

2.6.1 Two-level-Hierarchies (t-l-hs) of Objects

We recall from [16]: A class (respectively *instance*) *two-level hierarchy (t-l-h)* is a special type of directed acyclic graph D whose vertices belong to the set $O_a^{(1)}$ (respectively $O_c^{(1)}$) with the following properties:

(1) If D consists of a single vertex then we say that D is an *empty t-l-h.*

(2) A non-empty t-l-h D consists of at least two vertices, one is called *Representative (REP)* and the rest are called *Components (COs).* The REP-object in D is linked to every CO, i.e. *(REP, CO)* belongs to $E(D)$ which is the set of arcs of D for each CO in D.

For any non-empty t-l-h D, we write $D = [O[u]; O_1[u^1], \ldots, O_n[u^n]]$ where $O[u]$ is the *REP* and $O_1[u^1], \ldots, O_n[u^n]$ are the *COs.*

We use $D_\emptyset$ to denote $[\emptyset[\delta^n]; \emptyset[\delta^n], \ldots, \emptyset[\delta^n]]$ and if D is an empty t-l-h, we write $D = [O[u]]^{(2)}$.

If $(CO, REP) \in E(D)$ for some CO in D then $(CO, REP) \in E(D)$ for all COs in D. There is no arc between any two COs in D. The arc(s) between a *REP* and one of its CO constitute the *link.*
If $D \in L^{(2)}$, the set of t-l-hs, and $e \in E(D)$ then

(3) e cannot belong to any other D in $L^{(2)}$ and

(4) Any two objects can only be linked by a unique (node disjoint) path where by *path* we mean the concatenation of arcs leading from one of the objects to the other.

In Figure 2, we illustrate the different types of two-level-hierarchies in a generic way.

2.6.2 2-objects

A *second-order object* (2-object) is a pair $\langle H, D \rangle \equiv H[D]^{(2)}$ where H is the object identity (unique) and D is a t-l-h encapsulated in H. We call a 2-object $\langle H, D \rangle$ *empty* if D is an empty t-l-h.

The set of 2-objects is denoted by $O^{(2)}$. We write $O_a^{(2)}$ if the set of vertices of D, $V(D) \subseteq O_a^{(1)}$ and $O_c^{(2)}$ if $V(D) \subseteq O_c^{(2)}$. The elements of $O_a^{(2)}$ are called *abstract 2-objects* or *2-classes* and those of $O_c^{(2)}$ *concrete 2-objects* or *2-instances.*

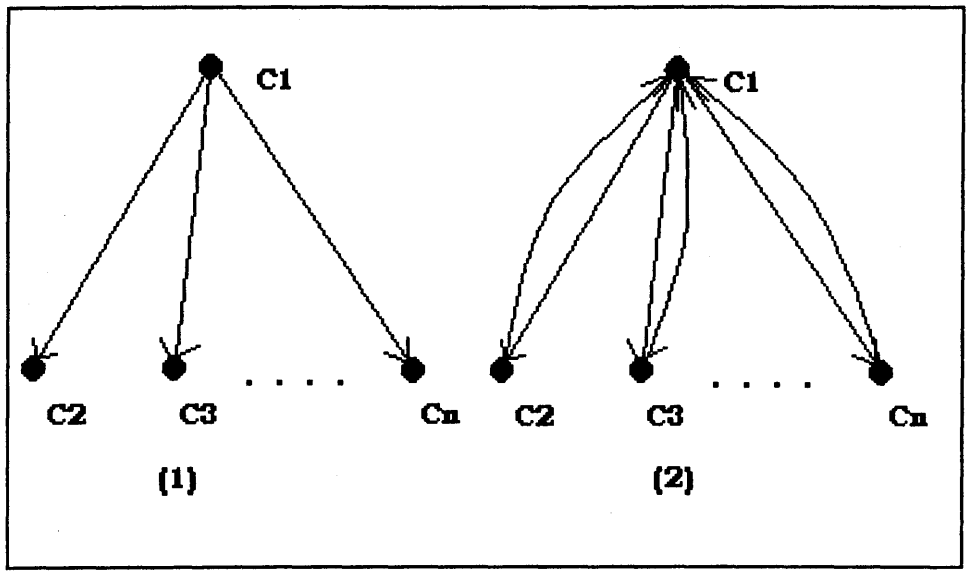

Figure 2: We distinguish two types of t-l-hs: (1) the links consist of one arc from the *REP* to ever *CO*. With this type we model specialization and grouping 2-objects. The t-l-hs of type (2) have links that consist of two arcs and are used for modelling association and aggregation 2-objects

2.6.3 Definition of Complex Objects

Complex objects are derived by composing 2-objects as follows:

(i) Any 2-object is a complex object, called an *elementary complex object.*

(ii) Let $\langle H_1, D_1\rangle$ and $\langle H_2, D_2\rangle$ be 2-objects. If one or both are empty then they are *non-composable.*

For non empty 2-objects

(a) if there exists an object $O[u]$ that is a *CO* of $\langle H_1, D_1\rangle$, say and a *REP* of $\langle H_2, D_2\rangle$, we define $\langle H_1, D_1\rangle \circ \langle H_2, D_2\rangle = \langle F, D\rangle$ a complex object where $F = (H_1, H_2)$ and D is the directed graph with $V(D) = V(D_1) \cup V(D_2)$ and $E(D) = E(D_1) \cup E(D_2)$.

(b) if there exists an object $O[u]$ that is a *CO* of $\langle H_1, D_1\rangle$ and of $\langle H_2, D_2\rangle$ we define $\langle H_1, D_1\rangle \circ \langle H_2, D_2\rangle = \langle F, D\rangle$ a complex object where $F = \{H_1, H_2\}$ and D as in (a).

2-objects $\langle H_1, D_1\rangle$ and $\langle H_2, D_2\rangle$ satisfying (a) or (b) are *composable.* Otherwise $\langle H_1, D_1\rangle$ and $\langle H_2, D_2\rangle$ are *non-composable.*

(iii) If $\langle H_1, D_1\rangle, \ldots, \langle H_k, D_k\rangle$ are non-empty 2-objects that are pairwise non-composable, but each $\langle H_i, D_i\rangle, i = 1, \ldots, k$, is composable with a 2-object $\langle H, D\rangle$ as in (ii)(a), then $\langle H, D\rangle \circ (\langle H_1, D_1\rangle, \ldots, \langle H_k, D_k\rangle) := \langle F, D_{k+1}\rangle$ is a complex object where $F = (H, \{H_1, \ldots, H_k\})$ and D_{k+1} is the directed graph with $V(D_{k+1}) = V(D) \cup V(D_1) \cup \ldots \cup V(D_k)$ and $E(D_{k+1}) = E(D) \cup E(D_1) \cup \ldots \cup E(D_k)$

(iv) If $C_1 = \langle F_1, D_1'\rangle$ and $C_2 = \langle F_2, D_2'\rangle$ are complex objects then C_1 and C_2 are composable if there exists a pair of 2-objects $\langle H_1, D_1\rangle \in C_1$ and $\langle H_2, D_2\rangle \in C_2$ that are composable and we write $C_1 \circ C_2 = \langle F, D\rangle$ for the resulting complex object where $F = (F_1, F_2)$ if $\langle H_1, D_1\rangle$ and $\langle H_2, D_2\rangle$ are composable as in (ii)(a) and $F = \{F_1, F_2\}$ if $\langle H_1, D_1\rangle$ and $\langle H_2, D_2\rangle$ are composable as in (ii)(b).

Note that, by definition of a t-l-h, if such a pair of 2-objects exists then it is necessarily unique. Hence, we can compose two complex objects in only one way. Otherwise C_1 and C_2 are non-composable.

(v) If $C_1 = \langle F_1, D_1\rangle, \ldots, C_k = \langle F_k, D_k\rangle$ are pairwise non-composable complex objects and each C_i, $i = 1, \ldots, k$, is composable with a complex object $C = \langle F, D\rangle$ then we define the composition $C \circ (C_1, \ldots, C_k)$ analogous to (iii). The object identity of the composition is $(F, \{F_1, \ldots, F_k\})$.

A collection of complex objects (classes and their instances) form an *object-base.*

2.6.4 2-Objects and Complex Objects as Greedoids

In this section we show that we can associate every 2-object and every complex object with a greedoid. Greedoids have been introduced to provide a structural framework for the greedy algorithm [10].

A set system $(E, \mathcal{F})$ (with E finite and $\mathcal{F} \subseteq 2^E$) is a *greedoid* if

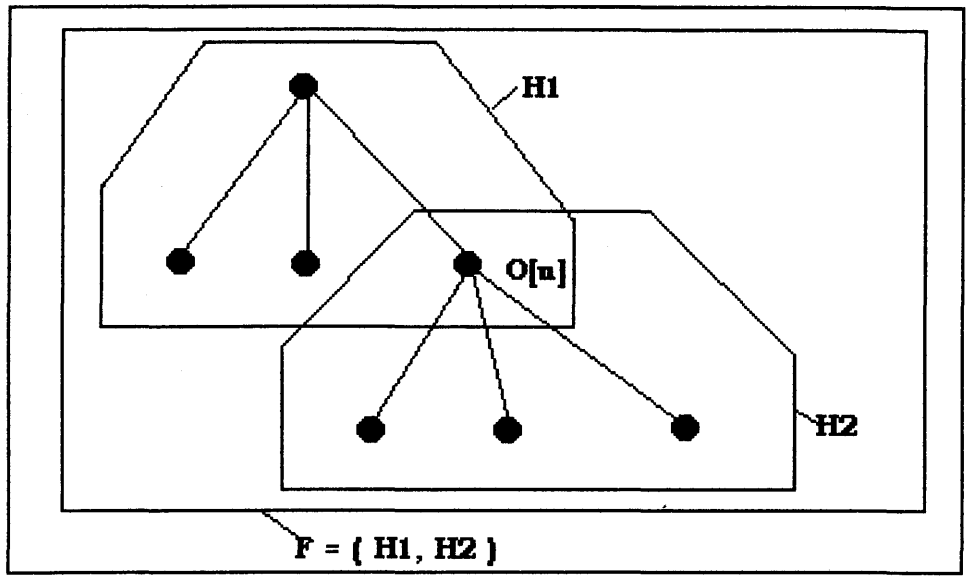

Figure 3: $\langle H_1, D_1 \rangle \circ \langle H_2, D_2 \rangle$

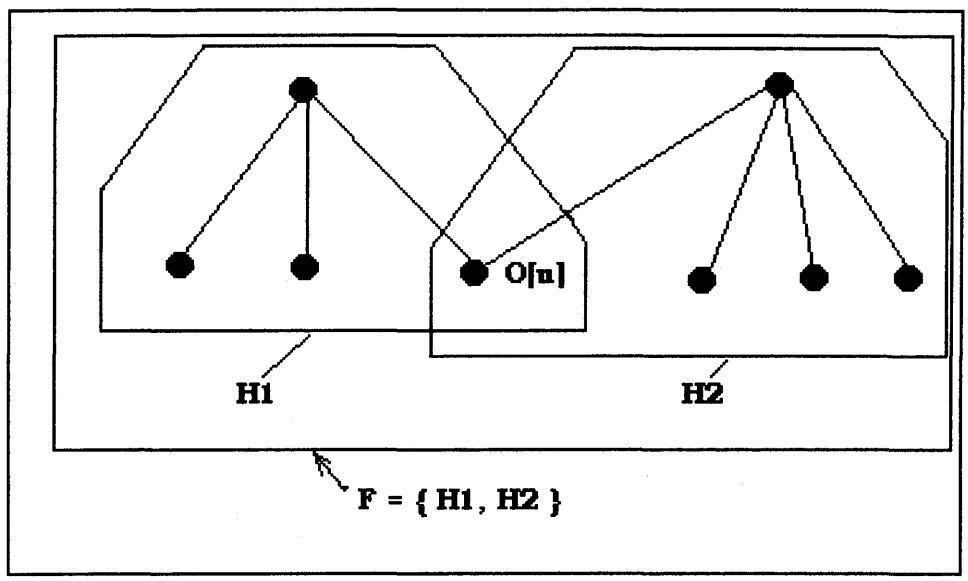

Figure 4: $\langle H_1, D_1 \rangle \circ \langle H_2, D_2 \rangle$

(i) $\emptyset \in \mathcal{F}$

(ii) for all $X \in \mathcal{F}$ there exists an $a \in X$ such that $X - \{a\} \in \mathcal{F}$

(iii) if $X, Y \in \mathcal{F}$ and $|X| = |Y| + 1$ then there exists an $a \in X - Y$ such that $Y \cup \{a\} \in \mathcal{F}$.

A set system $(E, \mathcal{F})$ satisfying (i) and (ii) is called an *accessible set system* and the elements of $\mathcal{F}$ are called *feasible sets*.

Informally, the greedy algorithm when run on a set of system, builds a solution by beginning with the empty set and successively adding the best remaining element while containing feasibility. For completeness we recall (cf.[14]) a modified version of the greedy algorithm for accessible set systems: let $(E, \mathcal{F})$ be an accessible set system and $f : 2^E \to \mathbb{R}$ is an objective function and consider the optimization problem

$$max\{f(F) : F \in \mathcal{F}\}.$$

The modified greedy algorithm goes as follows:

(1) Take $F^* = \emptyset$ as the initial feasible solution and set $F = \emptyset$;

(2) choose $x \in E \backslash F$ such that $F \cup \{x\} \in \mathcal{F}$ and $f(F \cup x) \geq f(F \cup y)$ for all $y \in E \backslash F$ with $F \cup y \in \mathcal{F}$; if no such x exists - *STOP* (F^* is the greedy solution);

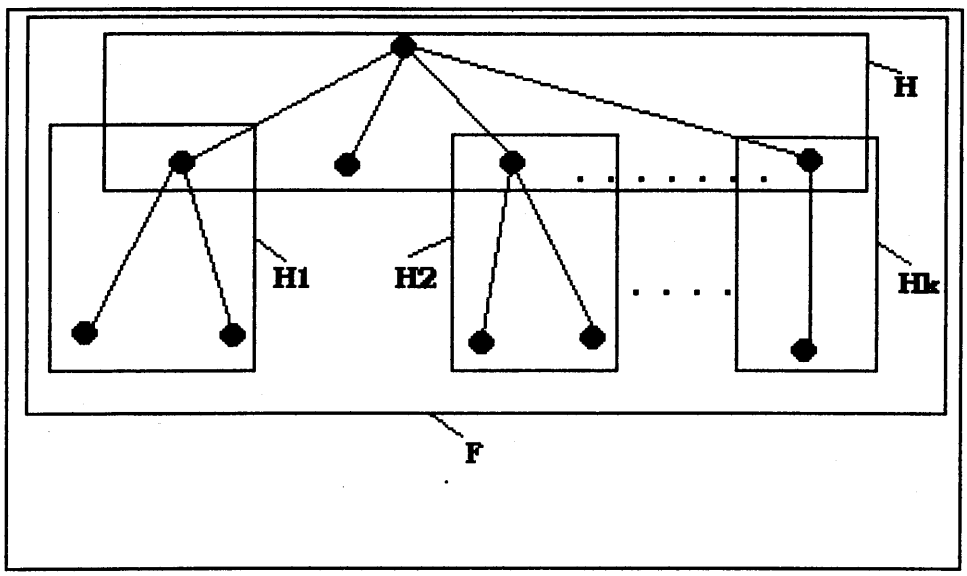

Figure 5: $\langle H,D\rangle \circ (\langle H_1,D_1\rangle,\ldots,\langle H_k,D_k\rangle)$

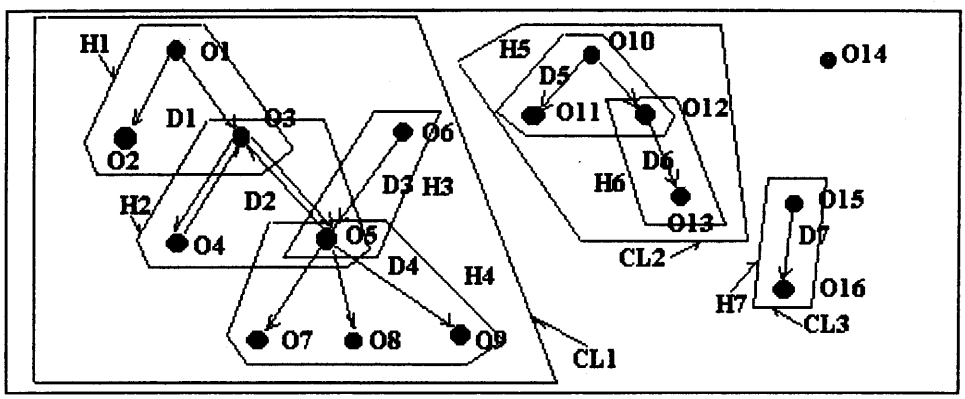

Figure 6: The classes of the above objectbase are denoted by $O1,\ldots,O16$ while the 2-classes' object identities are: $H1,\ldots,H7$, and their t-l-hs: $D1,\ldots,D7$ (the class $O14$ is considered an empty 2-class). The complex objects' identities are: $CL1 = (H1,\{H2,H3\},H4), CL2 = (H5,H6)$, and $CL3 = H7$.

(3) if $f(F\cup x) > f(F^*)$ then replace F^* by $F\cup x$;

(4) replace F by $F\cup x$ and $GOTO(2)$.

in [14] Goecke, Korte and Lovász characterize those structures for which the modified greedy algorithm computes an optimal solution for every linear objective function:

Theorem

Let $(E,\mathcal{F})$ be an accessible set system, then the following are equivalent

(i) For every linear objective function $f : 2^E \to \mathbb{R}$ the modified greedy algorithm determines an optimal solution for the problem

$$max\{f(F) : F \in \mathcal{F}\}.$$

(ii) $(E,\mathcal{F})$ is a greedoid and no $B \subseteq E$ has the following property:

$$\left.\begin{array}{l} z \in B, x,y \notin B \text{ such that } B \in \mathcal{F}, \\ B - z\cup\{x,y\} \in \mathcal{F}, \\ B - z\cup\{y\} \notin \mathcal{F} \\ \text{and } B\cup\{y\} \notin \mathcal{F} \end{array}\right\} (*).\blacksquare$$

For a 2-object (or complex object) $\langle H,D\rangle$ we consider the *search* (or *branching*) *greedoid* defined on the directed graph D: A vertex $P_0 \in V(D)$ is marked and called "root". The ground set for the greedoid is $E = E(D)$, the set of arcs of D, and $\mathcal{F}$ is the collection of trees in D rooted at P_0.

It is easy to see that property $(*)$ holds for the greedoid defined on the underlying directed graph of a 2-object $\langle H,D\rangle$: e.g.
If $P_0 = REP_{\langle H,D\rangle}$, $\mathcal{F} = 2^{E(D)}$.
If $P_0 =$ a component, $\mathcal{F} = \{\emptyset\}$

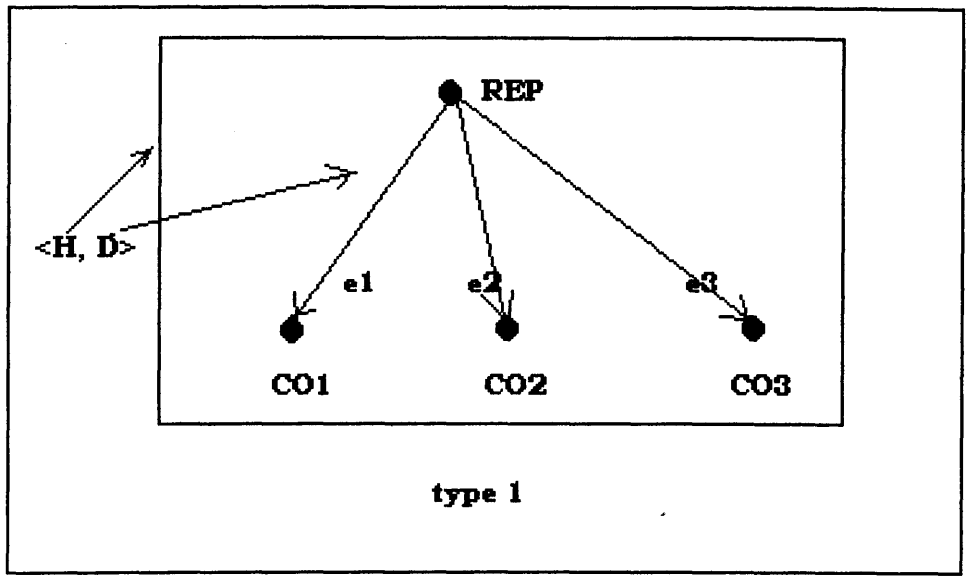

Figure 7: $\langle H,D\rangle$ where D is of type 1.

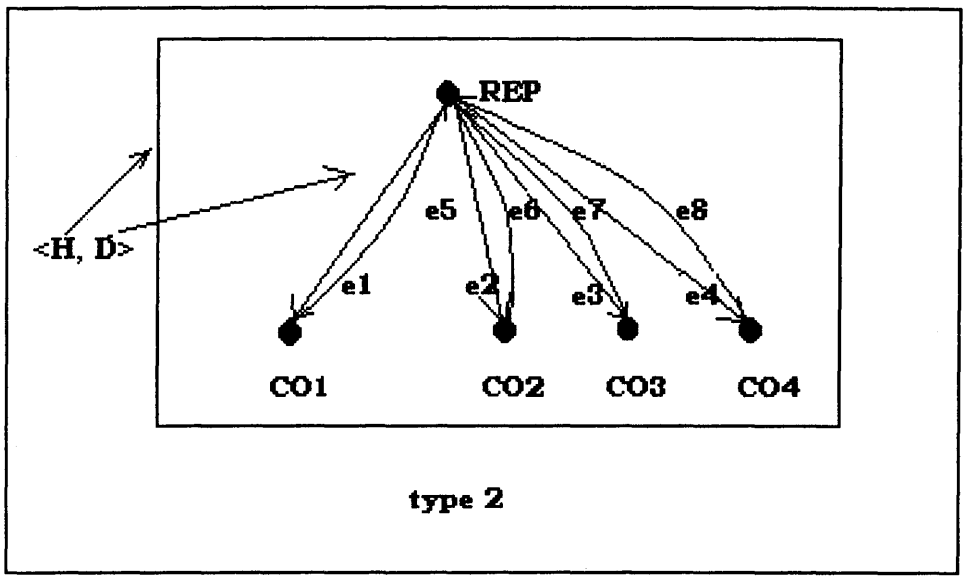

Figure 8: $\langle H,D\rangle$ where D is of type 2.

If $P_0 = REP_{\langle H,D\rangle}$, see Figure 9.
If $P_0 = CO_1$ (say), see Figure 10.

However it is not the case for complex objects, for example: let $C = \langle H,D\rangle \circ \langle H',D'\rangle = \langle F,D''\rangle$, a composition of 2-objects $\langle H,D\rangle$ and $\langle H',D'\rangle$. Suppose D is of type 1, say D (See Figure 11)

and D' is of type 2, say D' (See Figure 12).

then D'' becomes (See Figure 13).

If we choose $P_0 = REP_{\langle H,D\rangle}$, then $(*)$ is not satisfied: let $B = \{e_1,e_2\}$, $z = e_1, x = e_3, y = e_5$. Then $B \in \mathcal{F}, B - z \cup \{x,y\} = \{e_2,e_3,e_5\} \in \mathcal{F}, B - z \cup \{y\} = \{e_2,e_5\} \notin \mathcal{F}$ and $B \cup \{y\} = \{e_1,e_2,e_5\} \notin \mathcal{F}$.

In an implementation the weight function $w : E(D) \to \mathbb{R}$ can be defined in such a way that it depends on the filtering function of the arc. The induced linear objective function $f : 2^{E(D)} \to \mathbb{R}$ with $f(A) := \sum_{e \in A} w(e)$ also depends on the message's features (i.e. how many datatypes should be properly instantiated during the message resolution process).

2.7 An Algebra of 2-Objects

2.7.1 Active 2-Objects

The *active 2-objects* are defined analogously to the active objects in Section 2.4. To describe the specific types of an active 2-object, i.e. the structural relationship that specifies the detailed types of the links of its internal structure D, we use subscripts on the object-id. Let $H_i[D]^2$ be an active 2-object:

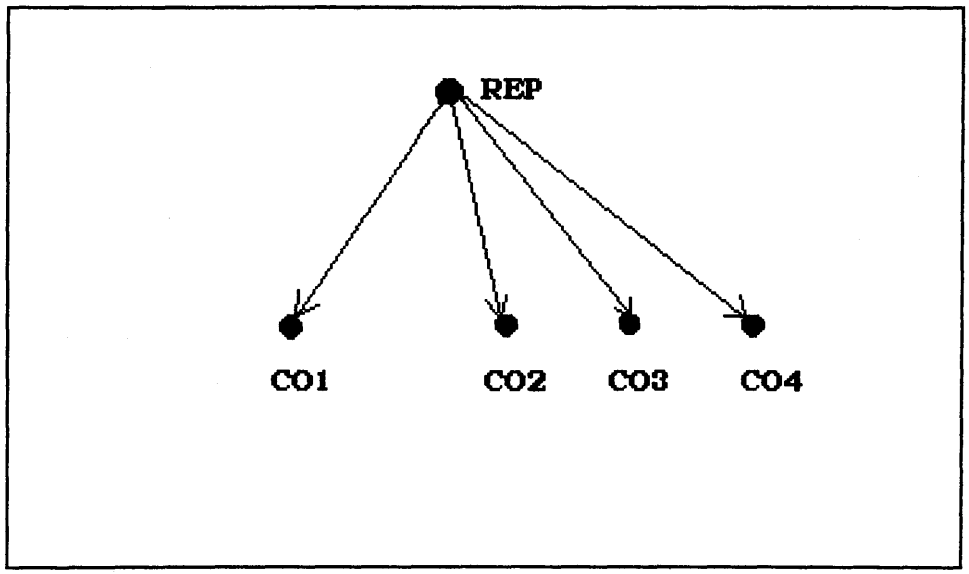

Figure 9: $\mathcal{F}$ = the set of subtrees of the above tree (root *REP*).

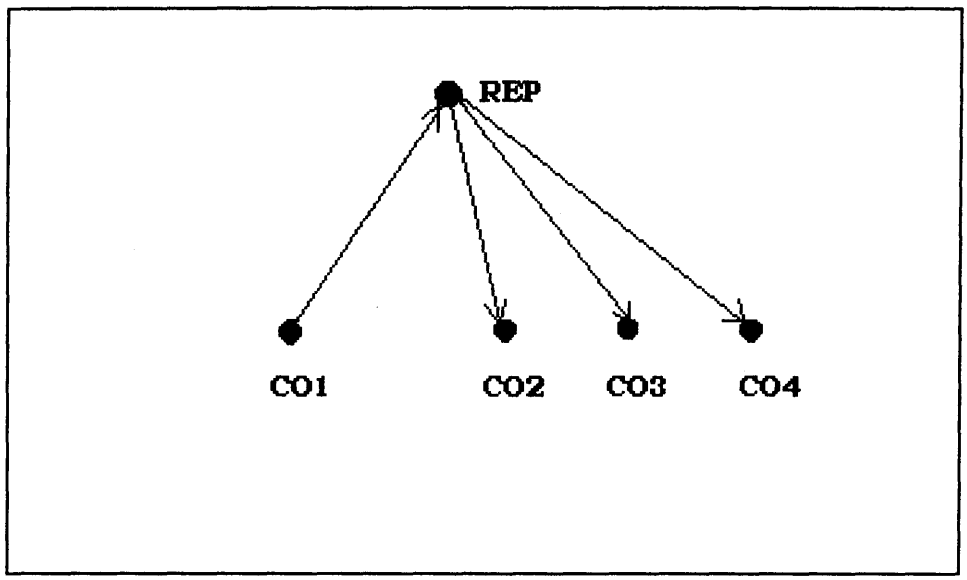

Figure 10: $\mathcal{F}$ = the set of subtrees of the above tree (root CO_1)

(1) i = *"m"* if D is of type 2 and the filtering function equals *"AAA"* for all links. These 2-objects correspond to the "member-of" structural relationship supported by the association operation *ASC* (the operations are defined and analysed below),

(2) i = *"p"* if D is of type 2 and the filtering function on each (*REP*,*CO*) in D equals *"AAA"* and on each (*CO*,*REP*) in D equals *"ANA"*. These 2-objects correspond to the "part-of" relationship supported by the aggregation operation *AGG*,

(3) i = *"g"* if D is of type 1 and the filtering function equals *"ANN"*. These 2-objects correspond to the "group-of" structural relationship supported by the grouping operation *GRP*,

(4) i = *"s"* if D is of type 1 and the filtering function equals *"AAN"*. These 2-objects correspond to the "is-a" structural relationship supported by the specialization operation *SPE* and

(5) i = *"$H_j[D']$"* if D is an instance t-l-h of type j (as defined in the above four cases) created (instantiated) from the active 2-class $H_j[D']$.

Classes (1) - (4) are used to characterize the 2-classes. In the fifth case every 2-instance is characterized through its own 2-class.

In Figure 14 we give an example of a simple object-base's model, where we explain the various 2-classes and the values that the links are taking:

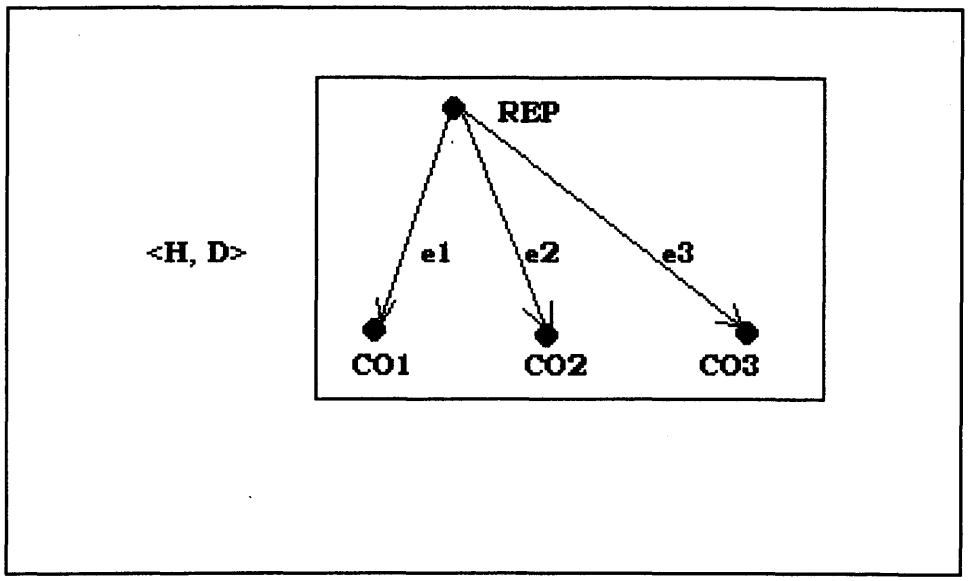

Figure 11: $\langle H, D\rangle$, D of type 1.

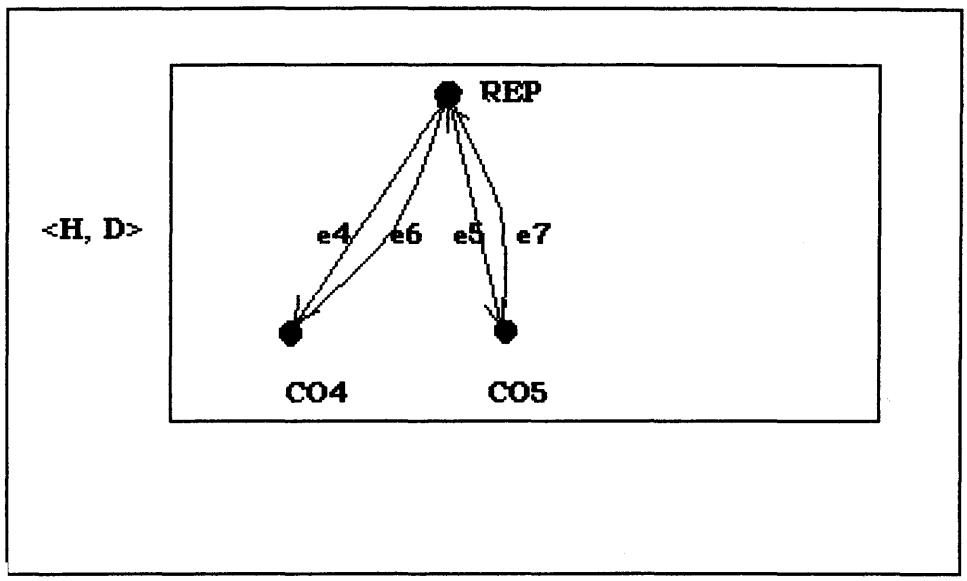

Figure 12: $\langle H', D'\rangle$, D' of type 2.

We now have an algebra of 2-objects $\Omega^{(2)} = \langle O^{(2)}, F^{(2)}\rangle$ where $F^{(2)} = \{\emptyset[D_\emptyset]^{(2)}, I, DL20, SPE, ASC, AGG, XSPE, RSPE, XASC, RASC, XAGG, RAGG, XGRP, RGRP, CR2I\}$.

Before giving a description of each operation in $F^{(2)}$ we first specify different categories of constraints:

Constraint-A: For every pair of 2-classes $H[D]^{(2)}, H'[D']^{(2)}$ we have $H \neq \emptyset, H' = \emptyset, D = D_\emptyset$ and $D' = [O[u]; O_1[u^1] \ldots O_n[u^n]]^{(2)}$.

Constraint-B: For every pair of 2-classes $H_i[D]^{(2)}, H'[D']^{(2)}$ where $i \in \{m, p, g, s\}$, we have $H_i \neq \emptyset, H' = \emptyset, D = [O[u]; O_1[u^1] \ldots O_n[u^n]]^{(2)}$ and $D' = [\emptyset[\delta^n]; O_{n+1}[u^{n+1}]]^{(2)}$.

Constraint-C: For every pair of 2-classes $H_i[D]^{(2)}, H'[D']^{(2)}$ where $i \in \{m, p, g, s\}$, we have $H_i \neq \emptyset, H' = \emptyset, D = [O[u]; O_1[u^1] \ldots O_n[u^n]]^{(2)}$ and $D' = [\emptyset[\delta^n]; O_j[u^j]]^{(2)}, 1 \leq j \leq n$.

Constraint-D: For every pair $H[D]^{(2)}, H'_{H''[D'']^{(2)}}[D']^{(2)}$ where $i \in \{m, p, g.s\}$, we have $H \neq \emptyset, H' = \emptyset, D = D_\emptyset, D' = \left[O'[u']; O'_1[u'^1] \ldots O_{n+k}[u'^{n+k}]\right]^{(2)}, D'' = \left[O''[u'']; O''_1[u''^1] \ldots O''_n[u''^n]\right]^{(2)}$, $O''[u'']$ is a class of $O'[u']$ and every $O''_j[u''^j] (1 \leq j \leq n)$ is a class of at least

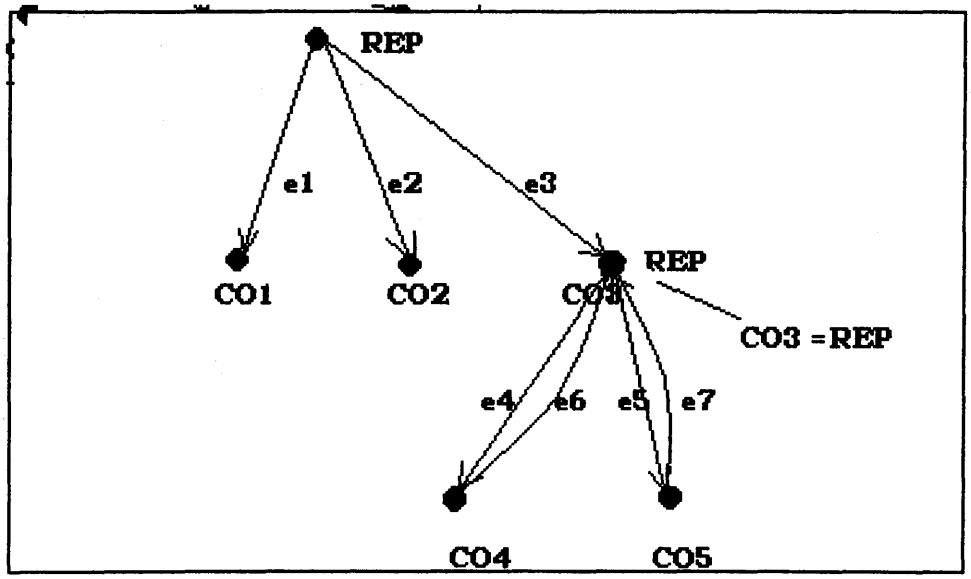

Figure 13: $\langle H,D\rangle \circ \langle H',D'\rangle = \langle (H,H'),D''\rangle$

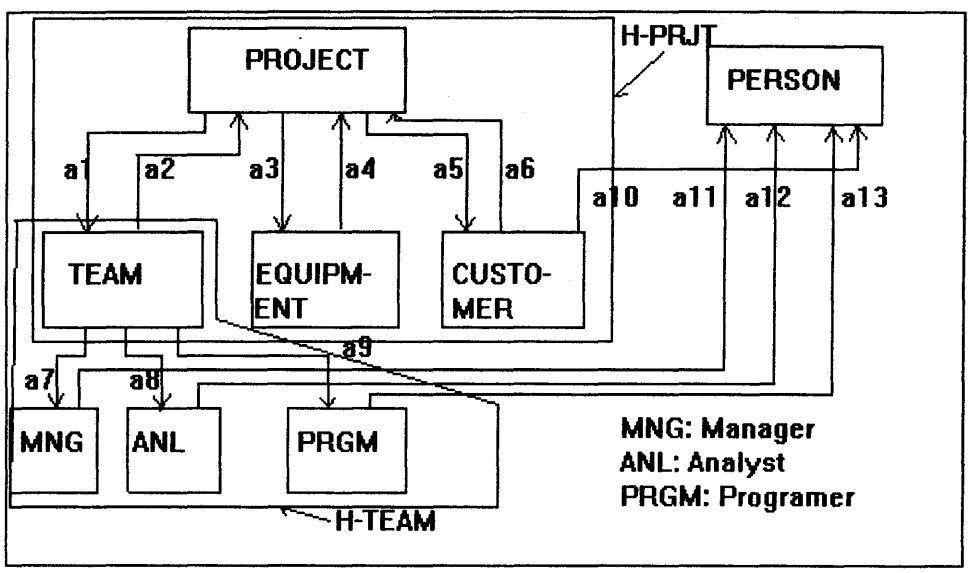

Figure 14: The objectbase's meta-model above consists of an association 2-class $H-PRJT$, where the *REP* is the class *PROJECT* and *COs* are the classes *TEAM*, *EQUIPMENT* and *CUSTOMER*. The class *TEAM* is also the *REP* of a grouping 2-class $H-TEAM$ with *COs* the classes *MNG*, *ANL* and *PRGM*. There also exist other specialization 2-classes, which we can name $H-MNG, H-ANL, H-PRGM$ and $H-CUSTOMER$, where for each one we have the classes $MNG, ANL, PRGM$ and $CUSTOMER$ to be the *REPs*, respectively, while the class *PERSON* is the unique and common *CO* of those 2-classes. The arcs $a1,\ldots,a11$ are specified b the "giver" class, the filtering function in compliance with the role of the class in the 2-class and the type of that 2-class, as well as, for the multiplicity that may be required by the designers. For example, $a3 = \#EQUIPMENT(AAA), a6 = PROJECT(AAA), a9 = \#PRGM(ANN)$ and $a11 = PERSON(AAN)$.

one of $O'_m[u'^m](1 \leq m \leq n+k)$.

1. Nullary operation

$$\emptyset[D_\emptyset]^{(2)} : \{\emptyset\} \rightarrow O^{(2)}, \emptyset[D_\emptyset]^{(2)}(\emptyset) = \emptyset[D_\emptyset]^{(2)}$$

2. Identity

$$1 : O^{(2)} \rightarrow O^{(2)}$$

3. Delete

$$DL2O : O^{(2)} \rightarrow O^{(2)},$$
$$DL2O(H_i[D^{(2)}]) =$$

$$\begin{cases} \emptyset[D_\emptyset]^{(2)} & \text{if the 2-object is a 2-class with no instances or it does not have common objects of another 2-object,} \\ H_i[D]^{(2)} & \text{otherwise} \end{cases}$$

4. Specialization

$$SPE : O^{(2)} \text{ x } O^{(2)} \rightarrow O^{(2)},$$
$$SPE(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_s[D']^{(2)} & \text{if constraint-}A\text{ holds,} \\ \emptyset[D_\emptyset]^{(2)} & \text{otherwise} \end{cases}$$

5. Association

$$ASC : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$ASC(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_m[D']^{(2)} & \text{if constraint-}A\text{ holds,} \\ \emptyset[D_\emptyset]^{(2)} & \text{otherwise} \end{cases}$$

6. Aggregation

$$AGG : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$AGG(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_p[D']^{(2)} & \text{if constraint-}A\text{ holds,} \\ \emptyset[D_\emptyset]^{(2)} & \text{otherwise} \end{cases}$$

7. Grouping

$$SPE : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$SPE(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_g[D']^{(2)} & \text{if constraint-}A\text{ holds,} \\ \emptyset[D_\emptyset]^{(2)} & \text{otherwise} \end{cases}$$

8. Expanding Specialization

$$XSPE : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$XSPE(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_s[D'']^{(2)} & \text{if } H[D]^{(2)} = H_s[D]^{(2)} \text{ and } \textit{constraint-B} \text{ holds, where } D'' = [O[u]; O_1[u^1], \ldots, O_n[u^n], O_{n+1}[u^{n+1}]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

9. Reducing Specialization

$$RSPE : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$RSPE(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_s[D'']^{(2)} & \text{if constraint-}C\text{ holds and } H[D]^{(2)} = H_s[D]^{(2)} \text{ where } D'' = [O[u]; O_1[u^1], \ldots, O_{j-1}[u^{j-1}], O_{j+1}[u^{j+1}], \ldots, O_n[u^n]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

10. Expanding Association

$$XASC : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$XASC(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_m[D'']^{(2)} & \text{if } H[D]^{(2)} = H_m[D]^{(2)} \text{ and } \textit{constraint-B} \text{ holds, where } D'' = [O[u]; O_1[u^1], \ldots, O_{j-1}[u^{j-1}], O_{j+1}[u^{j+1}], \ldots, O_n[u^n]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

11. Reducing Association

$$RASC : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$RASC(H[D]^{(2)}, H'[D']^{(2)} =$$

$$\begin{cases} H_m[D'']^{(2)} & \text{if constraint-}C\text{ holds and } H[D]^{(2)} = H_m[D]^{(2)} \text{ where } D'' = [O[u]; O_1[u^1], \ldots, O_{j-1}[u^{j-1}], O_{j+1}[u^{j+1}], \ldots, O_n[u^n]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

12. Expanding Aggregation

$$XAGG : O^{(2)} \text{x} O^{(2)} \rightarrow O^{(2)},$$
$$XAGG(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_p[D'']^{(2)} & \text{if constraint-}B\text{ holds and } H[D]^{(2)} = H_p[D]^{(2)} \text{ where } D'' = [O[u]; O_1[u^1], \ldots, O_n[u^n], O_{n+1}[u^{n+1}]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise.} \end{cases}$$

13. Reducing Aggregation

$$RAGG : O^{(2)} x O^{(2)} \rightarrow O^{(2)},$$
$$RAGG(H[D]^{(2)}, H'[D']^{(2)}) =$$
$$\begin{cases} H_p[D'']^{(2)} & \text{if constraint-}C\text{ holds and} \\ & H[D]^{(2)} = H_p[D]^{(2)} \text{ where} \\ & D'' = [O[u]; O_1[u^1], \ldots, \\ & O_{j-1}[u^{j-1}], O_{j+1}[u^{j+1}], \ldots, \\ & O_n[u^n]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

14. Expanding Grouping

$$XGRP : O^{(2)} x O^{(2)} \rightarrow O^{(2)},$$
$$XGRP(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_g[D'']^{(2)} & \text{if constraint-}B\text{ holds and} \\ & H[D]^{(2)} = H_g[D]^{(2)} \text{ where} \\ & D'' = [O[u]; O_1[u^1], \ldots, \\ & O_n[u^n], O_{n+1}[u^{n+1}]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

15. Reducing Grouping

$$RGRP : O^{(2)} x O^{(2)} \rightarrow O^{(2)},$$
$$RGRP(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_g[D'']^{(2)} & \text{if constraint-}C\text{ holds and} \\ & H[D]^{(2)} = H_g[D]^{(2)} \text{ where} \\ & D'' = [O[u]; O_1[u^1], \ldots, \\ & O_{j-1}[u^{j-1}], O_{j+1}[u^{j+1}], \ldots, \\ & O_n[u^n]]^{(2)}, \\ H[D]^{(2)} & \text{otherwise} \end{cases}$$

16. Create 2-instance

$$CR21 : O^{(2)} x O^{(2)} \rightarrow O^{(2)},$$
$$CR21(H[D]^{(2)}, H'[D']^{(2)}) =$$

$$\begin{cases} H_{H_i''[D'']^{(2)}}[D']^{(2)} & \text{if } H'[D']^{(2)} = \\ & H_{H_i''[D'']^{(2)}}[D']^{(2)} \\ & \text{and } \textit{constraint-D} \text{ holds} \\ \emptyset[D_\emptyset]^{(2)} & \text{otherwise} \end{cases}$$

Conclusion

In this work we introduced FMOB: a formal model for objectbases. FMOB is based on a universal algebra of words and appropriate extensions, while it supports objects, 2-objects and complex objects. The inheritance notion is replaced by the link concept, thus allowing to introduce additional structural relations (association, aggregation, and grouping). The algebra of the abstract words is functionally complete, as well as the modified greedy algorithm optimizes all linear objective functions over the branching greedoid associated with a 2-object.

2.7.2 Acknowledgement

The authors wish to thank the referees for helpful comments. The authors also wish to thank N. Mwanangulu-Massey for the impeccable editing of the camera-ready copy.

References

[1] F Civello. Roles of Composite Objects in Object-Oriented Analysis and Design. In *8th Annual Conference*, volume 28. OOPSLA, 1993.

[2] J Banerjee et al. Data Model Issues fro Object-Oriented Applications. *ACM, Transactions on Office Information Systems*, 5(1), January 1987.

[3] M Atkinson et al. The Object-Oriented Database System Manifesto. In *1st International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989.

[4] P Hientz et al. Object-Oriented Databases and their impact on future business applications. In *OOPSLA '91*, pages 171–183. ACM, 1991.

[5] Grätzer. *Universal Algebra.* D. Van Nostrand Company Inc., 1968.

[6] B Henderson-Sellers and J M Edwards. *The Working Object, Book Two on Object-Oriented Knowledge.* Prentice Hall Object-Oriented Series, 1994.

[7] D Hobby and M J MacKenzie. The structure of finite algebras. *American Mathematical Society*, 76, 1988.

[8] A Kemper. *Object-Oriented Database Management: Applications in Engineering and Computer Science.* Prentice Hall Intern, Editions, 1994.

[9] S N Khosafian and G P Copeland. Object Identity. *ACM SIGPLAN 21*, 11:406–416, 1986.

[10] B Korte and L Lov'asz. Mathematical structures underlying greedy algorithms. *Fundamentals of Computation Theory, Lecture Notes in Computer Science*, 117:205–209, 1981.

[11] A G Kurosh. *General Algebra.* Chelsea Publishing Company, 1963.

[12] M E S Loomis. OODBMS vs. Relational. *Journal of Object-Oriented Programming*, 3(2):79–82, 1990.

[13] J D McGregor and D A Sykes. *Object-Oriented Software Development: Engineering Software foe Reuse.* Van Nostrand Reinhold, New York, 1992.

[14] B Korte O Goecke and L Lov'asz. Examples and algorithmic properties of greedoids. In *Combinatorial optimization*, volume 1403, pages 113–161. Lecture Notes in Math, 1989.

[15] P A Patsouris. A Unified Framework for Knowledge Representation: A Formal Object-oriented Approach. In *7th International Conference on Software Engineering and Knowledge Engineering*, pages 131–135, Washington D.C., 1995. SEKE.

[16] P A Patsouris. Two-level hierarchies of Objects: A Unit of Reference for Distributed Object-Oriented Databases. In *International Conference on Parallel and Distributed Systems*, pages 466–471, Tokyo, Japan, 1996. ICPADS '96.

[17] H Werner. Eine Charakterisierung Funktional Vollstandiger Algebren. *Arch. Math*, XXI:381–185, 1970.

Received: 10/7/96, Accepted: 17/6/97

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications of Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript
Manuscripts for *review* should be prepared according to the following guidelines:

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
- the title (as brief as possible)
- the author's initials and surname
- the author's affiliation and address
- an abstract of less than 200 words
- an appropriate keyword list
- a list of relevant Computing Review Categories
- Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text according to the Harvard. References should also be according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page,

`http://www.cs.up.ac.za/sacj`

which gives a number of examples.

SACJ is produced using the LaTeX document preparation system, in particular LaTeX 2_ε. Previous versions were produced using a style file for a much older version of LaTeX, which is no longer supported. Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

Charges
Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

These charges may be waived upon request of the author and the discretion of the editor.

Proofs
Proofs of accepted papers may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Letters and Communications
Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words. Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book Reviews
Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement
Placement of advertisements at R1000.00 per full page per issue and R500.00 per half page per issue will be considered. These charges exclude specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 23, July 1999
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 23, Julie 1999
ISSN 1015-7999

Contents