17

Objects and Environments in Dynamic CIMOSA Models

I.L. Kotsiopoulos
Industry Consultant
Democratias 4, GR-12244 Aegaleo, Athens, Greece
Tel: ++30-1-5813299

Abstract

Objects play a fundamental part in Enterprise Modelling and therefore Enterprise Integration. In this paper, the general algebraic framework of objects as Observed Processes is employed as a semantic basis for the object-oriented modelling approach of CIMOSA. The framework is constructed to emphasise the dynamic aspects of the Enterprise, by defining algebraic abstraction mechanisms for modelling the changes of the Enterprise Objects in time and suitable environments for embedding them. Encouraged by the reception these ideas received when first presented by the author in the 1993 CIM Europe conference, this is a sequel to the framework there employed. Utilising the experience gained in the modelling of the ELVAL S.A. Aluminium Casting Plant in Greece and research results in the field of Object Oriented Languages Theory, the exposition concentrates on primary ideas and definitions.

Keywords

CIM architectures, Modelling architectures, Reference architectures, Enterprise models, Business Process models, Model correctness, Integrating infrastructures, Mathematical semantics, Objects, Object-oriented languages, Object dynamics, Dynamic behaviour, Environments, Modelling environments, Category theory, Category morphisms, Semantic unification

1 INTRODUCTION

Modern manufacturing enterprises are increasingly dependent upon large, complex information technology structures. To tackle the problem, Enterprise models and Integrated Design through Open Standards Interconnection have been proposed through systematic, but not universally accepted, as yet, attempts such as CIMOSA (ESPRIT Consortium AMICE, 1994),

P. Bernus et al. (eds.), *Modelling and Methodologies for Enterprise Integration*

GRAI (Williams et al, 1994), SUMM (Fulton, 1992), Purdue (Williams, 1994) and others. Most effort has come from application-oriented research groups, emphasising production and future marketing of modelling tools and services integration products, through the results of industrially-oriented CIM projects such as VOICE (EP5510, EP6682), CNMA (EP2617, EP5104), DINAS (EP6779), DCC (EP8823) and others. Attention paid to the formal foundations of any proposed architecture and to a unified underlying theory of Enterprise Integration through CIM architectures has been rather limited so far.

An architecture is essentially a specification platform for the development of models to be implemented. However, for this specification to be of use, issues such as consistency, identity of implementation (is a certain implementation an implementation of a certain architecture and if not why not) and program correctness, similar to those of a formal language, have to be resolved. It is a well known fact that, to this day, there is no universal acceptance of a formal definition of what a CIM architecture is!

Objects do play a fundamental part in modern Enterprise Integration Modelling strategy. This is due to the very idea of the conception of an object: a mapping between functional units of the real world and similarly behaving structural entities acting in a controlled man-made environment of an information processing system or model. Through some abstraction mechanism, physical objects are candidate objects of the model or "Enterprise objects". Relations between them are directly mapped into encapsulation, inheritance and subtyping relations, while behaviour, including dynamic behaviour, is described by the modelling language itself.

Although these advantages are present, they are counterbalanced by the lack of widely accepted definitions as to what constitutes an abstracted object used in a model. The confusion increases when dynamic aspects are considered, arising out of the observed behaviour (in time) and properties of the corresponding physical object. The latter are amplified when different physical processes sharing objects at different instants in time must be modelled. Questions such as whether differing observed behaviours under different processes (environments) constitute behaviours of the same object can be of critical importance to an Enterprise model.

The present paper aims to elaborate on the subject of a mathematical semantics framework for a general theory of CIM architectures. It proceeds with the definition of objects embedded in environments (chapter 2) and finishes by focusing on the characteristics of the Function View of the CIMOSA reference architecture for Enterprise modelling.

The CIMOSA architecture, developed as an ESPRIT project, is a well acknowledged method of addressing the more general issue of Enterprise Integration. Its aim is executable Enterprise models supported by an Integrating Infrastructure (IIS). Our choice has been dictated by its significance to the European standardisation efforts (CEN/CENELEC ENV 40 003) and its inherently formal specification (ESPRIT Consortium AMICE, 1994).

Of the four Views characterising a CIMOSA model, the Function View (or a Functional CIMOSA model) is, in the wording of an independent author, "pre-eminent" (Goranson, 1992). As it has been established by application experience within the ESPRIT project VOICE (EP5510, EP6682), representation of the function to be performed is of the highest priority for an enterprise model and the CIMOSA FRB syntax of its Business Description Language (ESPRIT Consortium AMICE, 1994) at Particular Design Level is the most robust and developed. Familiarity of the reader with the CIMOSA modelling constructs at the Particular Design Level of the Function View is assumed.

The formalism is built around object processing blocks, called "(Business/Domain) Processes" or "(Enterprise) Activities" accepting objects as "Function Inputs" and producing objects as "Function Outputs". Although these blocks ought to be described by the modelling language constructs provided, there is no built-in capability handling the dynamic properties of an object.

This weakness, which is the result of the empirical and heuristic method adopted by the AMICE consortium at the early stages of the CIMOSA architecture design, was firstly addressed in (Kotsiopoulos, 1993b).

In the present paper, we separate objects into two different classes and levels of abstraction, namely Enterprise objects and CIMOSA objects. Correspondingly, we embed those objects in two environments, the Enterprise environment and the CIMOSA environment. For reasons of structure, the CIMOSA environment is also hierarchically divided into Business and Activity environments. As in the previous paper, the theory of objects as Observed Processes (Ehrich, Goguen and Sernadas, 1991) is used for an input-output algebraic characterisation of object transformation and interaction.

As it will be shown in the sequel, only the notion of a categorial morphism, namely the behaviour morphism, is enough to describe all our objects, environments and time-frames for accommodating discrete or continuous dynamic behaviour. This justifies our choice of a suitable theory of objects as the basis for the provision of unified mathematical semantics for CIMOSA models. In our view, **a functional CIMOSA model is a set of categorial morphisms on CIMOSA objects, embedded in an Enterprise environment.**

We shall make these notions precise in what follows.

Finally, we should add that the object oriented description has been used by the CIMOSA designers as a means of specifying the IIS services executing a CIMOSA model. This important stage, which is also linked with the as yet non-stabilised Implementation Level of the Function View, should benefit from a formal method defining objects and operating environments. Indeed, difficulties have been encountered within VOICE in the implementation of services according to the CIMOSA specification. The experience of the ELVAL pilot, in which a SCADA package is also integrated as an external application, is that many of the CIMOSA Presentation Services are also provided by the functions of the package, but not in the modularity and decomposition expected by CIMOSA. The dilemma is whether one should develop those services once again or should consider a different service specification and integrate them.

We believe that a formal framework focused on object definitions should have much to offer in this direction. Ultimately, a consistent services specification model could be constructed, thus eliminating such questions. It would also be of considerable help towards establishing internationally accepted service standards for which our approach could serve as a basis.

2 ENTERPRISE OBJECTS

We introduce objects as mathematical constructions under a framework capable of accommodating their dynamic behaviour. For reasons explained in the previous section, the "Categorial Theory of Objects as Observed Processes" (Ehrich-Goguen-Sernadas, 1991) has been chosen as the vehicle, with interactions among objects mapped to categorial object morphisms. In what follows, we briefly review basic definitions and results of the theory

(thereafter referred to as the "standard theory") and also give additional constructions particular to our purposes.

Informally speaking, an object is a mapping device between events and values of certain types called **attribute-value pairs** or **observations**. The theory of observed processes abstracts from these two notions by defining a universal set U of **behaviour atoms** which contains everything atomic, that is occurring at some single point in time. U contains all events and all attribute-value pairs (observations) any object can engage into. A subset $A \subseteq U$ is an **alphabet** for behaviour atoms. A set $s \subseteq A$ corresponding to all elements of A at a given moment in time is called a **behaviour snapshot**, while a subset $S \ni s$ of the power set of A, symbolised as 2^A serves as a **snapshot alphabet** over A. Clearly, $s \in S \subseteq 2^A$. Each behaviour snapshot is anchored to a single point in time called its **date**.

Dynamic behaviour of an object in this setting results from the attachment of behaviour snapshots to points in time. Formally this is defined as a map $\lambda: t \to S, t \in TIME$, with *TIME* some time domain be it discrete or continuous. λ is called a **trajectory** over S, while a set of trajectories is called a **behaviour**. One can see here the generalisation of the concept of trace used in Discrete Event Calculi (Hoare, 1985), (Degano et al, 1988).

With the universal set U as a universe of urelements over some fixed category *SET*, all snapshot alphabets form a category *SNAP*, as do time domains (subcategory *TIME* of *SET*). All behaviours $BHV(SNAP, TIME)$ over *SNAP* with respect to *TIME* form a complete category.

Morphisms between them are category morphisms σ defined as $\sigma: (S_1, \Lambda_1) \to (S_2, \Lambda_2)$, such that $\mathrm{dom}(\lambda) = \mathrm{dom}\sigma(\lambda)$, where $\lambda \in \Lambda$ is a trajectory and Λ, Λ_1, Λ_2 are abstract sets of trajectories. The definition of an object given below, depicts it as a behaviour morphism.

Remark 2.1

A behaviour morphism is essentially a mapping of different characterisations of points of the time axis. The full generality of the concept allows mappings between different time axes and the information "hung" on them. It would be useful to consider a special kind of morphism which preserves the original time scale. Such a morphism $\sigma: (S_1, \Lambda_1) \to (S_2, \Lambda_2)$ would map snapshots without destroying their date and can be defined so that $\sigma(\lambda) = \lambda \circ f$, where λ is a trajectory and a snapshot morphism $f: S_1 \to S_2$ exists in some fixed *SNAP*. We term σ a **date-preserving morphism** (oblivious morphism in (Ehrich et al, 1991)). Note that a date-preserving morphism is fully defined once f is defined, and has the property that $\sigma(\lambda) \in S_2$ depends only on the λ components of the same time point and not on any previous points lying either before or after or even concurrently.

Definition 2.1

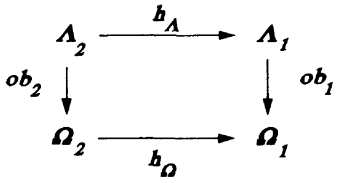
Let E, V be snapshot alphabets corresponding to sets of trajectories (behaviours) Λ, Ω. An **object** *ob* is a behaviour morphism $ob: (E, \Lambda) \to (V, \Omega)$ in the category $BHV(SNAP, TIME)$, that is complete behaviour morphisms, over the categories of behaviour snapshots *SNAP*, with respect to time domains *TIME*.

An object tells how observations in time (and/or space) depend on events in time (and/or space). We usually omit the alphabets, write $ob: \Lambda \to \Omega$, and refer to the domain Λ of the morphism as the **Process Part** and to the image Ω as the **Observation Part,** thus justifying the term Observed Process. One should pay attention, however, as to what characteristics the

TIME category should carry so that meaningful observation results are produced when the limitations of instrumentation are taken into account. We refer the reader to (Frachet and Colombardi, 1993) for an innovative treatment of the semantics of time in discrete event modelling, using the constructions of Nonstandard Analysis. According to this, the trajectories of a behaviour can only be observed at unique standard times or "dates" and at a (finite) resolution bounded below by a standard number.

Definition 2.2

Consider objects $ob_1: \Lambda_1 \to \Omega_1$ and $ob_2: \Lambda_2 \to \Omega_2$. An **object morphism** is a pair $h_\Lambda: \Lambda_2 \to \Lambda_1, h_\Omega: \Omega_2 \to \Omega_1$ of behaviour morphisms such that the following diagram commutes:



We symbolise: $ob_1 = (h_\Lambda, h_\Omega)\{ob_2\}$ for the resulting object.

Remark 2.2.

The notion of an object $ob: (E, \Lambda) \to (V, \Omega)$ as a behaviour morphism *ob* and the generality of a trajectory implies that neither the domain of the morphism (process part) consists exclusively of events nor the image (observation part) exclusively of attribute values. Both parts can contain both events and attribute values, although the case of an events-only domain and a values-only image is the most frequent.

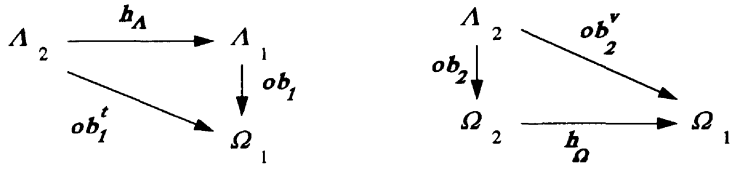
Remark 2.3.

The diagram of definition 2.2 is commutative, therefore $h: \Lambda_2 \to \Omega_1$ is an object. We call *h* the **induced object** of the morphism (h_Λ, h_Ω), a notion useful in the subsequent CIMOSA constructions.

It can be shown (Ehrich et al, 1991) that objects and object morphisms form a co-complete category. Moreover, composite objects within this category can be constructed via morphisms. Two particularly important operations on objects, namely "trigger" and "view", have also been defined at (Ehrich et al, 1991). They play a decisive part in our construction too.

Definition 2.3

Let ob_1, ob_2 be objects and h_Λ, h_Ω be behaviour morphisms such that the diagrams below are commutative. Then h_Λ is called a **triggering morphism** on ob_1 and the object $ob_1^t = h_\Lambda \circ ob_1$ is correspondingly called a **trigger** over ob_1. h_Ω is called an **observation morphism** on ob_2 and the object $ob_2^v = ob_2 \circ h_\Omega$ correspondingly called an **observation** on ob_2.



Definition 2.4

Consider the trigger and the observation morphisms h_Λ, h_Ω of the previous definition as date-preserving morphisms with the added property that they are restrictions on snapshot alphabets (that is $E_1 \subseteq E_2, V_1 \subseteq V_2$ on current notation) going with inclusions on the respective atom alphabets. Then $ob_2^v = ob_2 \circ h_\Omega$ is called an **object view** on ob_2 and the object morphism $h_\Lambda: \Lambda_2 \to \Lambda_1, h_\Omega: \Omega_2 \to \Omega_1$ a **partial observation morphism** on ob_1. ob_2 is correspondingly called a **partial observation** on ob_1.

Notice that in the diagram on the left hand side ob_1 is "triggered" via h_Λ in the sense that it "tells" the object how to "obey" the commands" in Λ_2. On the right hand side, ob_2 is "observed" or "viewed" via h_Ω. Where the conditions of definition 2.4 are met, the left diagram "disregards" events in Λ_2 which are not "in the scope" of Λ_1 and the right diagram "views" only those observations (attribute-value pairs) in the scope of Ω_1. In the case both morphisms are combined, such as that of the partial observation, the partially observing object ob_2 disregards the "uninteresting" events and observations of the object under partial observation ob_1.

Definition 2.5

Let $ob_i, i = 1,\ldots,n, n \in \mathbf{N}$ be a finite set of objects, which we consider correspond to the functions of the physical objects of an Enterprise. Suppose that for every i a partial observation morphism $(h_\Lambda, h_\Omega)_i$ has been defined such that the resulting objects $ob_i^E = (h_\Lambda, h_\Omega)_i \{ob_i\}$ are subject to a set of morphisms MO^E. We term the thus created objects ob_i^E the **Enterprise objects** with respect to the set $\{ob_i, i = 1,\ldots,n, n \in \mathbf{N}\} \cup MO^E$ constituting the **Enterprise**.

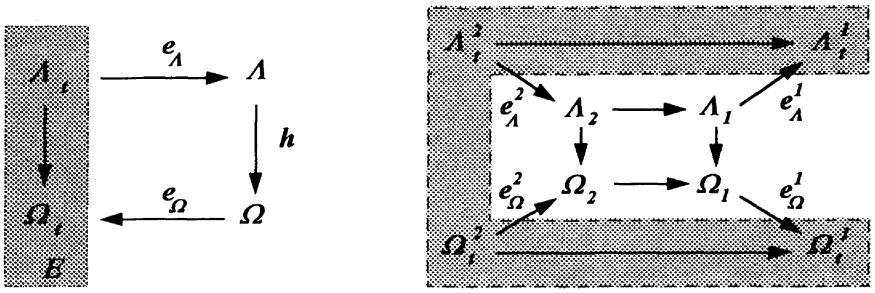
The last two definitions are added to the standard theory as an abstract construction of what we commonly term "the Enterprise and its objects", that is, loosely speaking, a set of objects and object manipulation mechanisms performing some function. Notice that the basic abstraction mechanism is the partial observation morphism, used as a means of disregarding useless detail, not in the scope of our interests.

For example, in the ELVAL pilot site for VOICE (Kotsiopoulos, 1993a), the object *metal* was of interest only as far as its temperature measurement. The partial observation morphism disregards other attributes such as "weight" and triggers observations only every two minutes instead of continuously. The Enterprise object is therefore *metal*: $\Lambda_1 \to \Omega_1$ with Ω_1 being the temperature observation. Λ_1 triggers observations continually in time at a set period of two minutes (sampling). Its behaviour alphabet is the set $B_{metal} = \{(e_i, T), i = 1,\ldots, T \in [A,B]\}$, where e corresponds to the measurement events indexed by i and T is any real number (temperature) between set limits A and B.

Finally, we diverge from the standard theory in order to define the notion of an environment to a set of behaviour or object morphisms.

Definition 2.6

Consider a behaviour morphism $h:(S,\Lambda)\rightarrow(S,\Omega)$ and the date-preserving morphisms $e_\Lambda:\Lambda_t\rightarrow\Lambda$, $e_\Omega:\Omega\rightarrow\Omega_t$ so that the left diagram commutes. Then the set $\{e_\Lambda, e_\Omega\}\cup E$, where E is a set of behaviours and behaviour morphisms is called an **environment** to the behaviour morphism h. The morphisms e_Λ and e_Ω representing the links to the environment are called **encoding** and **decoding** morphisms accordingly. Should there be other behaviour morphisms with corresponding domains and encoding and decoding morphisms, a common environment for all of them can be set by suitably enlarging the original.



By similar constructions, an environment to an object can be set which induces corresponding "environment objects", of the type $e_\Lambda \circ h \circ e_\Omega$. Finally, for the object morphism $h_\Lambda:\Lambda_2\rightarrow\Lambda_1, h_\Omega:\Omega_2\rightarrow\Omega_1$ the set of date preserving morphisms $e_\Lambda^2:\Lambda_t^2\rightarrow\Lambda_2$, $e_\Omega^2:\Omega_t^2\rightarrow\Omega_2$, (encoding) $e_\Lambda^1:\Lambda_1\rightarrow\Lambda_t^1$, $e_\Omega^1:\Omega_1\rightarrow\Omega_t^1$ (decoding), such that the right diagram is commutative, in union with the set E, is defined as an **environment** to the object morphism or, correspondingly, to a set of similarly endowed object morphisms.

The idea of an environment to an object or to an object morphism differs from the idea usually given in the object oriented literature. There, the environment is an embedding of an object into a larger object. The larger object is a container of all the events and the observations of the smaller object, while in our construction, this is not so. What we term environment here is a much more natural concept for the hierarchical control structures CIMOSA employs. It depicts the environment both as a supplier and a recipient of selected subsets of behaviour, without reference to the internal workings of the embedded objects or morphisms (events and/or observations), a notion close to the common concept of a physical environment. In (Ehrich et al, 1991) this is closer to the definition of an implementation of an object over another. In (Kotsiopoulos, 1993b) this corresponds to the Petri Net identification of the CIMOSA constructions and to environments for Petri Net building blocks found in (Baumgarten, 1988). It is readily seen that using the last definition we can construct an Enterprise environment over Enterprise objects or morphisms.

In what follows we impose additional structure over Enterprise objects and their morphisms, compliant with the constructs of CIMOSA functional models.

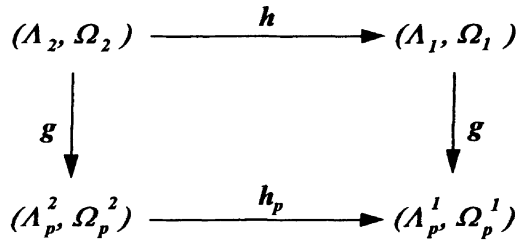
3 MODELS AND CIMOSA OBJECTS

Before focusing on CIMOSA constructions, a few thoughts on general model construction. We consider that a certain Enterprise has been selected and that Enterprise objects and

morphisms between them have been identified and abstracted from real objects and their interactions. A more structured expression of the Enterprise machinery is desired and therefore a model is employed. But what is a model?

Definition 3.1

Let $h:(\Lambda_2,\Omega_2)\rightarrow(\Lambda_1,\Omega_1)$ be a morphism between Enterprise objects and $g:(\Lambda_2,\Omega_2)\rightarrow(\Lambda_p^2,\Omega_p^2)$ be a partial observation morphism on (Λ_2,Ω_2). Then h_p is a (correct) **model** of the Enterprise if for every morphism h between Enterprise objects the following diagram is commutative.



The images of the morphism g are correspondingly called **model objects** and a class (type) of those together with a corresponding class of morphisms is called a **modelling architecture**.

Note that a modelling architecture does not ensure model correctness; indeed, this is the very reason for which different modelling architectures exist: to ensure model correctness. A supplementary concept, namely that of model completeness can be found in (Bernus, Nemes and Morris, 1995), in a model theoretic setting.

We shall now concentrate on the CIMOSA modelling architecture by constructing the class of CIMOSA objects and morphisms so that they are in agreement with the AMICE FRB (ESPRIT Consortium AMICE, 1994). The method is of importance in its own right as it may be used for the construction of other models besides CIMOSA. We apply a bottom-up approach by defining the smallest unit in a functional CIMOSA model, the Functional Operation, through a special object morphism.

Definition 3.2

Consider an object $ob_e:e\rightarrow\Omega$, such that e is a singleton in the event space and Ω is a singleton in the space of observations consisting of n-tuple vectors with typed components. Then ob_e is called an **elementary object** if its observation part is closed with respect to pauses in *TIME*, meaning that any trajectory $\lambda:t\rightarrow S, t\in TIME$ can be obtained from any other by inserting pauses.

An elementary object is uniquely identified by the name of a single event and a set of attribute-value pairs. The requirement that its behaviour is closed with respect to pauses in the category of *TIME* means that the attachment of the single event on the time axis is independent of the particular point of attachment.

The next definition introduces the corresponding notion of an elementary morphism on elementary objects.

Definition 3.3

Let $ob_1{:}e_1 \rightarrow V$, where V is the observation space consisting of all n-tuples with specified attribute types, be an elementary object. Consider an object morphism (h_Ω, h_Λ) on ob_1, such that $h_\Lambda{:}e_1 \rightarrow e_2$, $h_\Omega{:}V \rightarrow V$, where e_1, e_2 are behaviour atoms consisting of single events with dates d_1, d_2 and $d_1 < d_2$. Suppose that these behaviour atoms have trajectories $\lambda_1{:}TIME \rightarrow E\|_{TIME=d_1}$ and $\lambda_2{:}TIME \rightarrow E\|_{TIME=d_2}$ where $E = \{e_1, \vec{v}_1; e_2, \vec{v}_2 \,|\vec{v}_1, \vec{v}_2 \in V\}$ is the snapshot alphabet. We call this morphism an **elementary morphism** on ob_1, in V.

As observations are triggered by single events and since the domain of the elementary morphism consists of elementary objects, its image is also an elementary object. The requirement for the occurrence dates satisfying $d_1 < d_2$ serves the causality of the execution of the morphism by a machine. An elementary morphism is uniquely defined by the pair of tuples $(\vec{v}_1, \vec{v}_2)$, of which the first constitutes the input and the second the output tuple. Equivalently, it can be identified by (h_e, h_V) and the structure (type) of the input and output vectors. Moreover, it can be given a mnemonic name similar to that of a callable function of a programming language. Indeed, this is the way CIMOSA FRB (ESPRIT Consortium AMICE, 1994) presents a "Functional Operation".

Definition 3.4

Consider a partial observation object morphism $h_e{:}e \rightarrow \Omega$, $h_V{:}\vec{v} \rightarrow \Omega$, such that e, $\vec{v}$ are respectively singletons of events and observation n-tuple vectors with typed components. Let ob_E be an Enterprise object. The object $(h_e, h_v) \circ ob_E$, which is the result of the partial observation, is an elementary object because its behaviour is closed with respect to pauses in *TIME*. An elementary morphism on $(h_e, h_v) \circ ob_E$ is called an **elementary operation** on ob_E. Suppose now that all elementary operations on the objects of the Enterprise can be indexed by a finite number of indices with values belonging to a parameter space. If the classes formed in this way are finite per particular Enterprise and if each instantiation can be produced from the class by a parameterised algorithm (program) then any such class is called a (CIMOSA) (Specified) **Functional Operation**.

Informally speaking, the definition ensures that not every elementary morphism is a Functional Operation (FO), but that it is bound to events and observations of an Enterprise object and that it is specified by arguments containing typed parameters. As the morphism has only one event per domain/codomain the FO can be mapped to a 3rd Generation Language (3GL) function call with arguments resulting from the behaviour of Enterprise objects, parameters calculated through a machine executable algorithm and parameters possibly given through a suitable environment characterising the class structure. CIMOSA defines the FO as the smallest unit (grain) of a functional model, mapping its execution to that of a function call by an Information Processing Machine.

The complication of the identification of the Functional Operation with a class of elementary operations arises out of the role it plays in a CIMOSA model. Being the smallest grain of the model it is the sole communicator with any object modelled as "Function Input" or "Function Output". It receives and supplies events and attribute values (behaviour atoms), while it accepts events and values (parameters) from the activity environment, themselves possibly being results of other Functional Operations. Images of Functional Operations are also returned to the activity environment.

According to our structures, the image of a Functional Operation is an elementary object. This can be identified by its image $\vec{v} \in V$, a single vector of attribute values, resulting from the "execution" of a function call and released to a suitable environment, to be defined below.

The sequence of execution of these calls (mapped to FOs) are embedded within a structure of control statements pseudocode, similar to those of a typical programming language, which also allow parallel execution and variable manipulation. Examples of such control statements are GOTO, IF THEN ELSE, SPAWN, COBEGIN, COEND, etc, defining the control structures of the CIMOSA Activity Behaviour (ESPRIT Consortium AMICE, 1994).

It is implicit within the theory of algorithms (Church's thesis) that an algorithmic procedure satisfies the definition of an object as an observed process by its identification to a Turing Machine and consequently that of a set of object morphisms on the elementary objects constituting the domains and the codomains of the Functional Operations.

Example 3.1

We quote some simplified pseudocode from the CIMOSA Enterprise model of the alloying process of the ELVAL S.A. aluminium casting plant (VOICE II consortium) (Kotsiopoulos, 1993a). The separation between the pseudocode controls and the Functional operations is evident from the mnemonic names.

```
ACTIVITY BEHAVIOUR: "EA-Alloying":
PRECONDITIONS:
  Not-empty(Chemical_Composition)
BEGIN
  TAKE_METAL_SAMPLE;
  KEY_IN(Foundry_No, Sample_No, Alloy);
  ANALYSE_SAMPLE[Chemical_Composition(Foundry_No, Sample_No, Alloy), PresentComposition];
  DO WHILE AddedAlloyingMaterialWeight <> 0
   *ADD(AddedAlloyingMaterialWeight);
   *TAKE_METAL_SAMPLE;
   ANALYSE_SAMPLE[Chemical_Composition(Foundry_No, Sample_No, Alloy),
                                                          PresentComposition];
   UPDATE_DATABASES(Chemical_Composition);
  ENDDO;
  Ending_Status := 'Finished';
END
```

The embedding of the Functional Operations in an environment, described within the Activity Behaviour, is evident.

Definition 3.5

Let F be a finite set of (CIMOSA) Functional Operations, accepting elementary objects ob_E^j, $j = 1,\ldots,m$, m an integer as domains and producing also elementary objects as images. Consider an environment for all elements of F, called the (CIMOSA) **activity environment** such that the encoding morphism is a partial observation with identity in its process part and the decoding morphism is an inclusion. Endow this environment with all the control structure of the CIMOSA Activity Behaviour. As mentioned before, this is equivalent to object and behaviour morphisms and consequently to a composite (environment induced) object morphism (h_A, h_Ω) on elementary object compositions constituting on partial observations of Enterprise objects. The morphism under consideration is then called a (CIMOSA) **Enterprise**

Activity and is supplemented by a unique control code assigning a value to the variable Ending_Status when the last observation of the image object of the morphism is triggered.

Each Enterprise Activity (h_Λ, h_Ω), as an object morphism, has an underlying event alphabet map h_Λ consisting of the control structures of the CIMOSA Activity Behaviour pseudocode as domain and all the events of the elementary objects involved through F as codomain. Parameterisation values associated with control codes are handled by the underlying observations map h_Ω, which also manages the attribute values of the elementary objects.

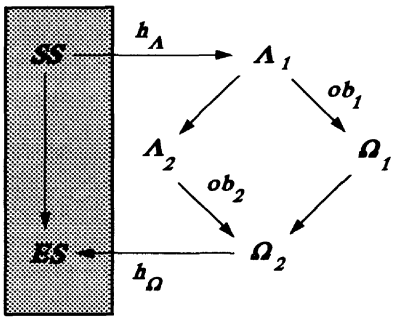
It is the responsibility of the Enterprise Activity and the constituent controls and Functional Operations to ensure that suitable objects will be produced as images of the morphism. This involves decomposition of input objects into structured elementary objects processable through elementary morphisms or Functional Operations. We name as a **CIMOSA object** any object which can belong to the domain or the codomain of an Enterprise Activity. It is also the responsibility of the modeller to ensure that a CIMOSA object corresponds to a partial observation of an Enterprise object, if the model is to have any applicability. This requirement becomes more stringent when fast dynamics are involved. The Enterprise Activity must apply Functional Operations at a fast rate so as to maintain the CIMOSA objects at "realistic levels" of partial observations and ensure model correctness (see definition 3.1 and examples of (Kotsiopoulos, 1993b) section 4).

Let us mention here that, due to the causality of the Functional Operation, an observation triggered first and another triggered last always exists in an Enterprise Activity, therefore a starting event and an ending status value is always provided. This is also a requirement for model correctness. We assume that the event triggering the first observation is supplied to the Activity Environment by another environment built over Enterprise Activities.

It is obvious that, as compositions of object morphisms are object morphisms, compositions of Enterprise Activities are also object morphisms. CIMOSA uses this fact in the construction of these composite morphisms, as follows.

Remark 3.1

Let $EA{:}ob_1 \to ob_2{:}(\Lambda_1 \to \Omega_1; \Lambda_2 \to \Omega_2)$ be an Enterprise Activity. Apply one triggering behaviour morphism h_Λ on Λ_1 and one observation morphism h_Ω on Λ_2, respectively, such that the former supplies the first event in ob_1 and the latter views the last observation in ob_2, i.e. the value of the Ending Status.



The morphism $SS \to ES$ is a composite behaviour morphism mapping the first to the last event of the induced object $Ea{:}\Lambda_1 \to \Omega_2$ (Remark 2.3). We can identify this composite

morphism as a partial observation morphism on *EA*, or as a partial observation on the induced object of *EA*. Let *SS* be the set of all starting events for a set of *EA* morphisms and *ES* be the set of all attribute-value pairs for all Ending Statuses for the same set. CIMOSA uses these two behaviour alphabets to embed Enterprise Activities into a hierarchically higher structure on which we elaborate in the next paragraphs.

Basic control structures, which we shall call *Operators*, are defined in the FRB I and II as a set of Condition-Action rules for the concurrent execution of EAs constituting an informal Discrete Event Calculus. The term Starting Status is not referred to explicitly in the FRB but is implied by the syntax of the Operators. No communication between Discrete Event Processes is catered for apart from Ending Status preconditioning and no data variables exist.

To be fair to the CIMOSA designers, these restrictions simplify the structure and allow for a hierarchical separation into compound discrete event processes. In an abstract sense, modules of subordinate processes can be readily formed provided their Starting/Ending Statuses are used only within the module itself. The modules interface with the rest of the model through Starting/Ending Statuses of their own, defined as a subset of the corresponding subordinate statuses. Such modules are called **Business Processes (BP)** and form the intermediate level of a hierarchical structure headed by a similar module exclusively called the **Domain Process (DP)**. The DP is the event interface of a whole compound Discrete Event Process of hierarchical structure. The Ending and the Starting Statuses of its subordinate Business Processes (and therefore Enterprise Activities) and a set of external events belonging to Enterprise objects are part of the **alphabet** of the DP.

We are in a position now to identify the key elements of the (CIMOSA) **Process Calculus**. This can be mapped to a Discrete Event Calculus (Procedural Rules, according to the FRB) by considering the Ending status assignment (discrete values) as a discrete event at this level:

- The Operators: *Pause, Forced, Go/NoGo, Condition, Spawn, Rendez-Vous* (FRB II, Item B12-2310).
- The alphabet *ES1, ES2,...,ESn, S1, S2,...,Sn, SS1, SS2,...,SSn, RE1, RE2,...,REn, FE1, FE2,...,FEn,* corresponding to Ending Status, Starting Status, Suspension, Resumption, and Forced Ending of Enterprise Activities.
- The special assignment Operator *Assign_Ending_Status* to subordinate processes. It is not given explicitly in the FRB but can be inferred from the property of suspending execution of an EA and the reassignment needs for Subordinate processes to be started again (see (Kotsiopoulos, 1993b) for comments on this).

With all structure in place questions can be raised on the capabilities of this rather undemanding calculus. Its specifications point to a language able to generate *Traces* of events. Its Ending Status carrying processes are Sequential in the sense of Hoare (Degano et al, 1988), (Hoare, 1985).

The alphabet of the Domain Process contains only external events, status events and possible exception-handling events. As only events are present at this level, each Domain Process is indeed a composite object morphism and its input and output objects are CIMOSA objects. For these objects, the structural requirements concerning the relation of the model to reality (model correctness) are similar to those of the Enterprise Activity morphism related objects.

4 CONCLUSIONS: ENTERPRISE AND CIMOSA ENVIRONMENTS

Two hierarchically related CIMOSA environments have been built over Enterprise objects, namely the activity environment (definition 3.5) and the **process environment** (remark 3.1). The former has been described in part 3, while the latter is given as the set of (a) encoding-decoding morphisms of type h_Λ, h_Ω in remark 3.1, (b) the behaviour alphabet which includes *SS* and *ES* and (c) the behaviour morphisms which arise out of the Process Calculus of part 3.
The process environment itself can be hierarchically divided into Business and Domain process environments, by using the same mechanism. Each process environment induces objects of type $SS \rightarrow ES$ which can be embedded in another environment with the same type of behaviour alphabet and morphisms. At the highest level of this decomposition, the Domain process environment includes external events in its alphabet, the only level at which this is allowed by CIMOSA.

The properties of the two main CIMOSA environments have been well explored in (Kotsiopoulos, 1993b) with their identification with suitable High level Petri Nets. We shall not refer to those here, but we shall point out some differences regarding our present constructions. First, on nomenclature, in (Kotsiopoulos, 1993b) the Activity environment is termed as the "Enterprise Activity environment" and the Process environment as the "Discrete Event Control (DEC) environment". Second, on properties, the Activity environment was considered as encompassing all ideal observer observations from the Enterprise environment. This had two implications. First, there was no Enterprise environment, as such, and second, the Activity environment, encompassing all, had to be considered as a generally nonretentive environment of observations. This, in practice, meant that it was not an IT programming environment, where values of variables remain constant unless operated on, but could be converted to one by the application of fast calls to its Functional operations ((Kotsiopoulos, 1993b), section 4). Here we have resorted to the Activity environment being an IT environment with all the familiar retentive properties. The role of a non-retentive environment is passed on to the Enterprise environment and consistency between the two lies in the fidelity of the partial observations constructed by the CIMOSA objects, i.e. on model correctness as in definition 3.1.

Remark 4.1

All elementary objects are "observable" through the Activity environment. By construction, the elementary objects, their events and their attribute values are released to the Activity environment, in line with the CIMOSA definition of a Functional Operation as a grain of the model which is either fully executed or not at all. This poses difficulties in the modelling of Enterprise operations such as monitoring, where fast updating takes place. In the CIMOSA model, there is no such thing as a never ending Functional Operation doing monitoring, if the results are to be used by the model itself. Instead, either each sampling operation is modelled as a separate FO, or the launch is a separate FO and the results of the monitoring are communicated to the model by external events from nonCIMOSA domains.

5 REFERENCES

Baumgarten, B. (1988) "On Internal and External Characterisations of PT-Net building block behaviour", *Advances in Petri Nets 1988*, Springer-Verlag.

Bernus, P., Nemes, L., Morris, R. (1995) "Possibilities and limitaions of reusing enterprise models", this conference.

Degano, P., Gorrieri, R., Marchetti, S.(1988) "An Exercise in concurrency: A CSP Process as a Condition/Event System", *Advances in Petri Nets 1988*, Springer-Verlag.

Ehrich, H. D., Goguen, J. A., Sernadas, A. (1991) "A Categorial Theory of Objects as Observed Processes", in *Foundations of Object-Oriented Languages, REX School/Workshop Proceedings, May/June 1990*, Springer-Verlag.

ESPRIT Consortium AMICE (1994), *CIM-OSA, Technical Baseline (Formal Reference Base)*, Releases 1991, 1994.

Frachet, J.-P. and Colombardi, G. (1993) "Elements of the semantics of time in GRAFCET and dynamic systems using Nonstandard Analysis", *Automatique-Productique-Informatique Industrielle*, Vol 27, No 1, 1993.

Fulton, J.A. (1992) "Enterprise Unification Using the Semantic Unification Meta-Model", in *Enterprise Integration Modelling: Proceedings of the 1st International Conference*, edited by C.J. Petrie, Jr, MIT Press, 1992.

Goranson, H.T. (1992) The CIMOSA Approach as an Enterprise Integration Strategy, in *Enterprise Intergration Modeling: Proceedings of the First International Conference*, Charles J. Petrie, Jr. (Ed), MIT Press, Cambridge.

Hoare, C.A.R. (1985) *Communicating Sequential Processes*, Prentice-Hall, 1985.

Kotsiopoulos, I.L. (1993a) "Modelling, the ELVAL S.A. case. The Function View at Particular Design Level.", *ESPRIT project 5510, VOICE, Final Deliverable*, January 1993.

Kotsiopoulos, I.L.(1993b) "Theoretical aspects of CIMOSA modelling", CIM Europe Conference, Amsterdam 1993, published in: *Realising CIM's industrial potential*, Kooij C., MacConaill P.A., Bastos J. (Eds), IOS PRESS.

Williams, T.J. et al, "Architectures for integrating manufacturing activities and enterprises", *Computers in Industry* 24, 1994.

Williams, T.J. (1994), "The Purdue Enterprise Reference Architecture", ibid.

6 BIOGRAPHY

Dr. I. L. Kotsiopoulos received the Dipl-Ing degree in Electrical Engineering from the National Technical University of Athens, Greece (1982) and the M.Sc. and Ph.D. degrees in Control Theory from the University of Manchester (1982) and the Imperial College of the University of London (1989), UK, respectively. In the past, he has held positions both in industry and in academia. At present he is the head of the industrial applications department of ITCC S.A., an informatics applications consultancy house. His current interests include Control Theory and Formal Methods for Enterprise Models and CIM architectures.