Mathematical Semantics for CIM Architectures

Iannis L. Kotsiopoulos

ITCC S.A., Informatics Applications Consultants

Sapfous 143,

GR-17675 Kallithea, Athens

GREECE

Abstract: The paper elaborates on the need for mathematical semantics as a means of an underlying formalism unifying the constructs of CIM architectures. It relies on mathematical characterisations of objects as category morphisms, in order to describe their dynamic behaviour and their interactions with suitable environments. To clarify the role of an enterprise model the idea of model correctness is also introduced. Application of the framework is made on the constructs of the Function View of the CIMOSA modelling architecture, chosen as an example.

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

Dr. Iannis 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 conducted researches in the theory of Optimal Control and Enterprise Modelling and held positions in both industry (Control Engineering) and academia (visiting professor at the Technical University of Crete). 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.

1. Introduction

The amount of data processed in today's manufacturing enterprises, and the complicated Information-Technology-executed scenarios on which they operate, have frequently resulted in a multitude of interconnected "software blackboxes". The result of such complexity is that quite often application maintenance, let alone enhancement, have become virtually impossible to perform.

Solutions to a problem making its presence felt within the last twenty years, initially concentrated on programming techniques, such as structured programming (early computerisation age). Today, they have evolved to integrated designs, in which Information Technology has become part of a general enterprise-wide strategy. Enterprise models and integrating infrastructure design through open standards interconnection have been instrumental in this approach, constituting what is loosely termed "a CIM architecture". These architectures are based on the abstraction enterprise models provide coupled with model enactment or execution platforms. They have been convincingly put forward to a wider audience as partial specifications accompanied by modelling applications, usually on paper, approximately three years ago. Attempts were projects and/or initiatives such as CIMOSA [9], GRAI [14], ODP [22], Purdue [22], [23], etc. Although the initial lack of an integrating infrastructure, open enough to support full model execution, has not been solved as yet in a universally accepted manner, specifications of main architectures today are nearly complete and a number of modelling tools (a result of European and national level projects [1], [10], [17]) has

emerged to support activities of the life cycle of a manufacturing enterprise. Besides, integrating infrastructures have appeared, either at specification level, such as OMG-CORBA [21], or at implementation level such as CIMOSA-VOICE [10] and CIM-BIOSYS [7].

Despite all this activity, issues related to modelling language consistency and architectures amalgamation, harmonisation and standardisation are still very much under development. Of note are initiatives, such as the IFIP/IFAC Task Force and CEN, the former on a Generic Enterprise Reference Architecture and Methodology to classify architectures [3] and the latter on the mapping between the constructs of QCIM and CIMOSA [5].

It is the opinion of the author that deep theoretical issues related to the structure and the semantics of CIM architectures lie at the heart of successful standardisation issues: a standards body has to know precisely what it is to be standardised, namely have its boundary uniquely defined, and, at the same time, have at its disposal criteria enabling it to assess the identity of various implementations of the standard: is a claimed implementation of or compliance with the standard a faithful one or not?

Up to the moment such issues are resolved, no method or associated architecture can hope to be of wide acceptance, as no available unambiguous standard for the construction of tools and infrastructure services support by vendors can be followed. It is time that part of the effort invested by the research community on prototypes and future marketing of industrial implementations of various architectures, is directed back to formal foundations and to a theory of IT architectures for CIM.

It is natural that such efforts are based on the deployment of mathematical semantics as a unifying means of expression and "inter-concept" communication. An architecture is essentially a specification for implementation. However, for this specification to be of use, issues such as consistency, identity of implementation, formal notions of equivalence of seemingly different architectures and program correctness, similar to those of a formal language, have to be resolved.

Results from such investigations will help to answer issues such as the differences or similarities of various proposed architectures, their suitability for certain tasks and communication between different architectural islands. Most of all, it is hoped to define criteria as to whether some proposed CIM structure is indeed a CIM architecture: *to this day, there is no formal and universally accepted definition of what a CIM architecture is!*

Topics of interest to a systematic development of such a theory could include:

- cognitive frameworks for abstraction of enterprise activity
- enterprise (domain) and language ontologies
- enterprise, modelling and operating environments, in which actions of models are embedded
- enterprise modelling languages as formal multi-representation languages realising meta-modelling primitives
- formal algebraic structures for modelling languages and dynamic enterprise objects
- formal specification of integrating infrastructure services, service objects and their dynamics
- model enactment (execution) platforms specification via formally defined object structures

- common frameworks for existing CIM architectures as realisations of meta-models
- equivalence of models and infrastructure services with respect to operating environments
- standardisation based on model and service equivalence

Given the young age of the subject, it is natural that we start from what is closest to a theoretical treatment: modelling. Enterprise models are a fundamental part of a CIM architecture. In fact, loosely speaking, such an architecture is perceived as a specification for execution or "enactment" [1] of an enterprise model. We deal with algebraic frameworks capable of accommodating dynamic properties of an enterprise model, viewed as formal interactions between objects embedded in suitable environments. Our approach is complementary to that of semantic-unification oriented attempts, such as SUMM [13], which emphasise the logical foundations of modelling languages semantics.

Once our basic framework is set, we focus on the mathematical characterisation of the Function View of the CIMOSA reference architecture for Enterprise modelling. CIMOSA has been chosen as an example for a number of reasons. Due to the inherently formal specification it employs [9], it has been widely examined within both the research community [19], [10], [17] and the standardisation bodies. Regarding the latter, documents such as [6], recently distributed for ballot (general guidelines for enterprise modelling), and [4], [16] (constructs to be used for enterprise modelling coupled with requirements for execution and integration services), although non-committal to CIMOSA, make strong reference to it. Finally, CIMOSA belongs to one of the four architectures which the IFIP/IFAC Task Force have found suitable [24]. The others are the GRAI-GIM Methodology, the Purdue Enterprise Reference Architecture, and, recently, TOVE (TOronto Virtual Enterprise) [11], [12], an architecture aiming at creating reusable ontologies and semantic implementations across a variety of enterprises.

2. Enterprise Objects

Objects used in enterprise models are perceived as mappings between functional units of the real world and similarly behaving structural entities acting in a controlled, man-made environment of an information processing system. This mapping can be either straightforward, at the low levels of the granularity of the model, or quite complex when new objects are created with the aid of the modelling language. These represent compositions, decompositions or abstractions of the real world objects for the purposes of effective model-aided visualisation or management. Although the role of a model is therefore critical, there are no widely accepted definitions as to what constitutes an abstracted or synthesised object used in a model. The confusion increases when dynamic behaviour is considered. Different physical processes may share objects at different instants in time; dialogue with a model may have to take place for model-driven operation; decisions affecting the enterprise (and therefore modelled) may have to be made, dependent on the continuously varying dynamic behaviour of a quantity.

We introduce objects as constructions of mathematical systems capable of accommodating their dynamic behaviour. The "Categorial Theory of Objects as Observed Processes" [8] has been chosen as the vehicle, with interactions among objects mapped to categorial object morphisms.

In this way, the full generality of category theory can ensure that no "useful real objects" can be disqualified from such a formalism and the same holds true for "useful interactions" between them and between "parent environments". We briefly describe the formalism in what follows.

Informally speaking, an object is seen as 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 is called 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 [15].

With the universal set U as a universe of urelements over some fixed category SET, all snapshot alphabets form a category $SNAP$ and so 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 $\Lambda, \Lambda_1, \Lambda_2$ are abstract sets of trajectories. The definition of an object given below depicts it as a behaviour morphism.

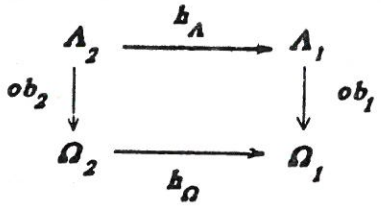
Remark 2.1. A behaviour morphism is essentially a mapping of different characterisations of points of the time axis. The full generality of the concept allows mappings between different time axes and the information "hung" on them. It would be useful to consider a special kind of morphism which preserves 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 [8]). 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 $BHV(SNAP, TIME)$, that is complete behaviour morphisms, over the categories of behaviour snapshots $SNAP$, with respect to time domains $TIME$. We usually write $ob: \Lambda \to \Omega$ when the corresponding alphabets are uniquely defined.□

An object tells how observations in time (and/or space) depend on events in time (and/or space).

We 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.

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 of behaviour morphisms $h_\Lambda: \Lambda_2 \to \Lambda_1, h_\Omega: \Omega_2 \to \Omega_1$ so that the next diagram commutes:



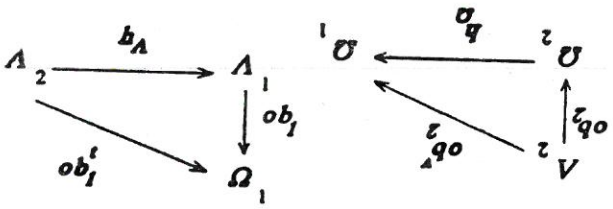
We symbolise $ob_1 = (h_\Lambda, h_\Omega)\{ob_2\}$ for the resulting object. □

Remark 2.2. The notion of an object $ob: \Lambda \to \Omega$ 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 to be of use in Section 4. □

It can be shown [8] that objects and object morphisms form a co-complete category. Composite objects within this category can also be constructed via morphisms. Two particularly important operations on objects, namely "trigger" and "view" have also been defined in [8]. They also play a decisive part in our construction.

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 $ob_1^t = h_\Lambda \circ ob_1$ 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 on the left hand side diagram, 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, then ob_2 disregards both the events

and the observations of ob_1 which are not in its scope.

Definition 2.5. Let $ob_i, i = 1,...,n, n \in \mathbf{N}$ be a finite set of objects, which we consider as corresponding to the functions of the physical objects of an Enterprise. Suppose that for every i a partial observation morphism $(h_A, h_\Omega)_i$ has been defined such that the resulting objects $ob_i^E = (h_A, h_\Omega)_i \{ob_i\}$ are subject to a set of morphisms MO^E. We term the so created objects ob_i^E the ***Enterprise objects*** with respect to the set MO^E constituting the ***Enterprise***. □

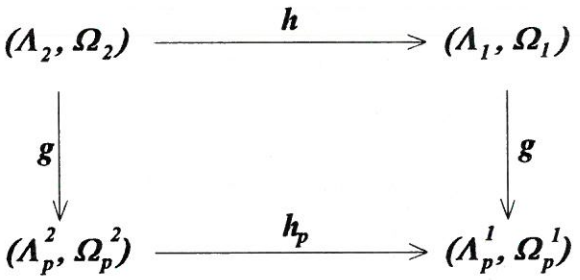
The last two definitions are added to the standard theory as an abstract construction of what we commonly term "the Enterprise and its objects", that is, loosely speaking, a set of objects and object manipulation mechanisms performing some function. Notice that the basic abstraction mechanism is the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

For example, the object *metal* is of interest to a melting furnace model only as far as its temperature measurement is concerned. The partial observation morphism disregards other attributes such as "weight" and triggers observations only every two minutes instead of continuously doing. 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 $B_{metal} = \{(e_i, T), i = 1,..., T \in [A, B]\}$, where e corresponds to the measurement events indexed by i and T is any real number (temperature) between set limits A and B.

3. Models and Environments

Let us think that a certain enterprise has been selected and that enterprise objects and morphisms between them have correctly been identified and abstracted from real objects and their interactions. To examine issues of integration and monitor overall performance, a more structured expression of the enterprise machinery is desired and therefore a "model" is employed. But what is a model and is there a way of separating good models from bad? We proceed with a definition of an object morphism belonging to a "correct" model of an enterprise, with respect to the required abstraction.

Definition 3.1. Let $h: (\Lambda_2, \Omega_2) \to (\Lambda_1, \Omega_1)$ be a morphism between Enterprise objects and $g: (\Lambda_2, \Omega_2) \to (\Lambda_p^2, \Omega_p^2)$ be a partial observation morphism on (Λ_2, Ω_2). Then h_p belongs to a (correct) ***model*** of the Enterprise if for every morphism h between Enterprise objects the following diagram is commutative.

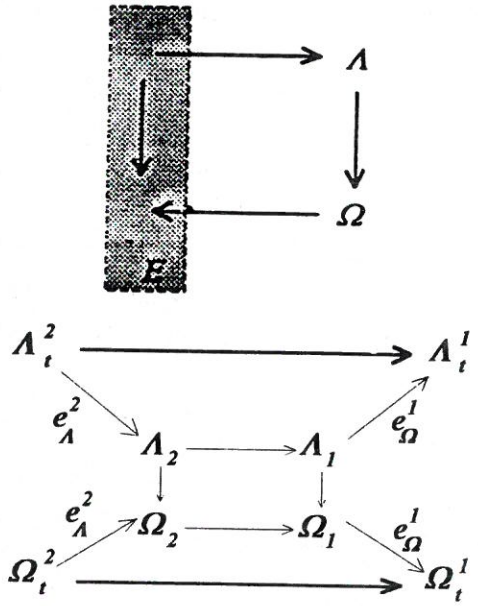


The images of the morphism g are correspondingly called ***model objects*** and a class (type) of those together with a corresponding class of morphisms is called a ***modelling architecture***.□

Note that a modelling architecture does not ensure model correctness; indeed, this is the very reason why different modelling architectures exist: to ensure model correctness, i.e. that each model object morphism, such as h_p, satisfies the previous definition.

Objects are usually seen as operating within a larger set or a larger object, frequently called an "environment". Here we also define a similar concept, which however is different from the traditional one encountered in the object oriented literature. On the other hand, it is adapted to the requirements of modelling architectures as it is amenable to the hierarchical structures those models use in an effort to approach the hierarchical organisation structures of an enterprise (Business Process models).

Definition 3.2. Consider a behaviour morphism $h:(S,\Lambda)\to(S,\Omega)$ and the date-preserving morphisms $e_\Lambda:\Lambda_t\to\Lambda, e_\Omega:\Omega\to\Omega_t$ 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 and 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 will 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 $h_\Lambda:\Lambda_2\to\Lambda_1, h_\Omega:\Omega_2\to\Omega_1$, the set of date-preserving morphisms $e^2_\Lambda:\Lambda^2_t\to\Lambda_2$, $e^2_\Omega:\Omega^2_t\to\Omega_2$, (encoding) $e^1_\Lambda:\Lambda_1\to\Lambda^1_t$, $e^1_\Omega:\Omega_1\to\Omega^1_t$ (decoding), such that the right diagram is commutative and, 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 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 which enterprise models, including CIMOSA, employ. It depicts the environment as both supplier and 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 [8] this is closer to the definition of an implementation of an object over another. In [20] this corresponds to the Petri Net identification of the CIMOSA constructions and to environments for Petri Net building blocks found in [2].

It is readily seen that using the last definition we can construct an Enterprise environment over Enterprise objects or morphisms. This environment is capable of accommodating the dynamic behaviour of objects and morphisms it

contains, as it can accept and supply behaviour atoms sequences.

We further impose additional structure over Enterprise objects and their morphisms, compliant with the constructs of CIMOSA models of the Function View at Particular Design level [9].

4. The CIMOSA Modelling Architecture

The CIMOSA CIM architecture provides an object-oriented structured modelling language and an open standards-based reference architecture of services and services-providing objects, called the Integrating InfraStructure (IIS). The aim is executable Enterprise models supported by the IIS.

Despite considerable advances in both modelling and IIS specification and provision of services prototypes made both within the AMICE consortium [17] and in satellite projects on CIMOSA validation, such as VOICE II [10], the CIMOSA architectural constructs largely remain without an underlying semantic uniformity, especially regarding handling of the dynamics of abstracted physical objects. This weakness, which is a result of the empirical heuristic method adopted at the early stages of its design, has firstly been addressed in [19]. There, the problem of the semantic uniformity of the CIMOSA Function View at Particular Design Level was treated with the deployment of mathematical semantics as a theoretical backbone to the modelling language constructs. We elaborate on this by deriving the basic CIMOSA functional constructs from the algebraic structures introduced in Sections 2 and 3. Familiarity of the reader with the CIMOSA modelling constructs at the Particular Design Level of the Function View is assumed.

To specify a modelling architecture according to Definition 3.2, one needs to specify the relevant object class. We shall realise this by posing additional structures to enterprise objects, thus abstracting them to CIMOSA objects. The same will be done with their corresponding morphisms and environments. In our view, ***a functional CIMOSA model is a set of category morphisms on CIMOSA objects, embedded in a CIMOSA environment***. CIMOSA objects are constructed as images of particular forms of object morphisms acting on Enterprise objects, where the responsibility of correctness lies with the modeller. The characteristics of these morphisms are chosen so as to agree with [9]. The method is of importance in its own right as it may be used for the construction of other models beside 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. We shall make these notions precise in what follows.

Finally, we should add that the object-oriented description has also 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 to the not-yet-stabilised Implementation Level of the Function View, should benefit from a formal method defining service objects and their operating environments. Indeed, difficulties have been encountered within EP6682 (VOICE II) in the implementation of services according to the CIMOSA specification. The experience of the ELVAL pilot, in which a well-known SCADA package is also integrated as an external application, is that most 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 again or should consider a different service specification and integrate them. We believe that the answer lies in a consistent open services specification model semantically uniform with the enterprise model and derived from the same formal framework, incorporating the issue of service equivalence. This could also be the basis for the establishment of international standards for services specification.

Definition 4.1. Consider an object $ob_e : e \to \Omega$, 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 $\lambda : t \to S, t \in TIME$ can be obtained from any other by inserting pauses.

The mechanism introduced by the previous definition is an abstraction mechanism mapping objects onto named entities with attributes. Indeed, 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.

Definition 4.2. Let $ob_1 : e_1 \to V$, where V is the observation space consisting of all n-tuples with specified attribute types, be an elementary CIMOSA object. Consider an object morphism (h_Ω, h_V) on ob_1, such that $h_V : e_1 \to e_2$, $h_\Omega : V \to V$, where e_1, e_2 are singletons of events with dates (points of occurrence in time) d_1, d_2, and $d_1 < d_2$. Suppose that the behaviour atoms have trajectories

$\lambda_1 : TIME \to E\|_{TIME=d_1}$, $\lambda_2 : TIME \to E\|_{TIME=d_2}$,

where $E = \{e_1, \vec{v}_1 : e_2, \vec{v}_2 \mid \vec{v}_1, \vec{v}_2 \in V\}$ is the snapshot alphabet. We call this 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 $d_1 < d_2$ has only been put to stress 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 [9] presents a "Functional Operation". □

Definition 4.3. Consider a partial observation object morphism $h_e : e \to \Omega$, $h_V : \vec{v} \to \Omega$, 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, but that it is bound to events and observations of an Enterprise object and that it is specified by arguments containing typed parameters. CIMOSA defines the FO as the smallest unit (grain) of a functional model, mapping its execution by an Information Processing System to that of a programming language function call. According to our structures, the image of a Functional Operation is an elementary object. We can identify this object 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 is embedded in a structure of control statements pseudocode, which also allows 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 [9].

It is implicit within the theory of formal languages (Church's thesis) that an algorithmic procedure satisfies the definition of an object as an observed process, the event part being the function calls executed in time and the observation part being the returned parameters. For our CIMOSA objects, the statements are defined by the Activity Behaviour language control structures and the Functional Operations.

We quote a simplified example of such a pseudocode described program from the alloying process of the ELVAL S.A. aluminium casting plant CIMOSA functional model (VOICE II consortium) [10]. The separation between the pseudocode controls and the Functional Operations is clear from the mnemonic names.

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

Loosely speaking, it seems that the Functional Operations in a CIMOSA model are included in an environment, which supplies the "launch events", in order to start their execution, and accepts the "stop events", in order to proceed on the next execution. We formalise this in the next definition.

Definition 4.3. Let F be a finite set of (CIMOSA) Functional Operations, accepting elementary objects ob_E^j, $j = 1,...,m$, m an integer, as domains, and also producing 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 of 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. □

Being an object morphism, each Enterprise Activity (h_A, h_Ω) has an underlying event alphabet map h_A 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 range. Parameterisation values associated with control codes are handled by the underlying observations map h_Ω, which also takes care of 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 are 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 co-domain 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 is involved. The Enterprise Activity has to apply Functional Operations at 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 [19], Section 4).

It is obvious that as compositions of object morphisms are object morphisms, compositions of Enterprise Activities are also object morphisms. CIMOSA uses this fact and supplies a Business environment hosting these composite morphisms.

Remark 4.1. Let $EA: ob_1 \rightarrow ob_2$: $(h_\Lambda; h_\Omega)$: $(\Lambda_1 \rightarrow \Omega_1; \Lambda_2 \rightarrow \Omega_2)$ be an Enterprise Activity. Construct an environment to such morphisms by a 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 (CIMOSA) Ending Status. The morphism $SS \rightarrow ES$, where SS is the set of all starting events for a set of EA morphisms and ES is the set of all attribute-value pairs for all Ending Statuses for the same set, is a composite behaviour morphism mapping the first event of the induced object $Ea: \Lambda_1 \rightarrow \Omega_2$ (Remark 2.3) to its last attribute-value pair, the (CIMOSA) Ending Status. We can identify this composite morphism as a partial observation morphism on EA, or as a partial observation on the induced objects of EA. Therefore it is an elementary object with a behaviour alphabet containing only the events of type SS, and the values of type ES, standing for the starting and the ending status respectively.

CIMOSA uses these for embedding Enterprise Activities into a hierarchically higher structure called Business Process and this, in turn, into

another structure called Domain Process. These structures are basically identical from the point of view of their event calculi, and correspond to our definition of an environment. We collectively term them the *(CIMOSA)* ***Process Environment.*** □

The starting and ending status events *SS* and *ES* (the values of *ES* are finite per Enterprise Activity and their appearance can be mapped to events), along with basic control structures, defined in [9] as a set of Condition-Action rules for the concurrent execution of EAs, constitute a simple Discrete Event Calculus (Procedural Rules in [9]), whose operators and alphabet, in the acceptance of Hoare [15], have been identified in [19]. No other communication between Discrete Event Processes is catered for apart from Ending Status preconditioning and no data variables are manipulated at this level. In an abstract sense, a hierarchy of modules of subordinate processes can be readily set, interfacing with the rest of the model through Starting/Ending Statuses of their own. Such modules are called ***Business Processes (BP)*** and ***Domain Processes (DP)***. The alphabets of the domain processes can also contain external events, in contrast with those of the Business Processes whose alphabet contains only events of type *SS* and *ES* and possible exception-handling events.

The specification of such a simple calculus for BPs and DPs point to an imperative language able to generate traces of events. Its Ending Status carrying processes are sequential in the sense of Hoare [15]. As only events are present at this level, each BP or DP is indeed a composite object morphism and its input and output objects are CIMOSA objects, for which the structural requirements concerning the relation of the model to reality are similar to those for the Enterprise Activity morphism related objects.

Remark 4.2. The properties of the two main CIMOSA environments have been explored in [19] with their identification by suitable Petri Nets. We shall not refer them here, but we shall point out some differences regarding our present constructions. First, on nomenclature, in [19] 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 in [19] 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 non-retentive environment of observations. This actually meant that it was not an IT programming environment, where values of variables remained constant unless operated on, but could be converted to one by the application of fast calls to its Functional operations ([19], Section 4). Here we have resorted to the Activity environment as 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 the consistency of the two lies in the fidelity of the partial observations constructed by the CIMOSA objects and the provision of model correctness according to Definition 3.1. □

Remark 4.3. All elementary objects are "observable" through the Activity environment. According to the construction of the elementary objects, their events and attribute values are released to the Activity environment. The construction is in accordance to the CIMOSA

definition of a Functional Operation as a grain of the model which is either fully executed (i.e. its elementary objects released to the environment) 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 launch of a Functional Operation doing monitoring, if the results of the monitoring are 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. □

Finally, each of the two CIMOSA environments defined in this section is endowed with a set of morphisms on all subordinate objects. This set is termed the ***boundary*** of the relevant environment. In contrast to the original CIMOSA definition, where the boundary is defined on a Domain Process as all the objects it exchanges with the outside world, our notion is both clearly defined and in line with the general idea of what a boundary is in mathematics.

5. Conclusions and Further Work

We have treated CIM architectures from the point of view of mathematical semantics unifying their constructs and concepts. We selected a theory of objects based on events, observations and time behaviour of algebraic objects, forming a complete category. Input/output descriptions of object transformations were mapped to category morphisms and so did interactions of objects with suitable environments. Model correctness was introduced as a property of morphisms between the enterprise objects and the objects of a model. The latter belong to a class and, along with their morphisms, constitute the modelling architecture.

The general theory was narrowed down in Section 4 in order to specify the Function View at Particular Design Level of the CIMOSA modelling architecture. The basis for the definition of the class of CIMOSA objects was the elementary operation on an elementary object. Activity environments were then constructed and, in turn, embedded into Process environments. The construction involves building an environment around the induced objects of an Enterprise Activity (Remark 4.1). This resulted in the elementary objects *Ea* exchanging the starting and the ending status events with a Discrete Event calculus object constituting the Process environment for all elementary objects *Ea*.

Finally, it should be noted that the CIMOSA objects, as partial observations of Enterprise objects, are also contained in an Enterprise environment and so are the elementary objects and all the induced objects of Enterprise Activities. In this way a class hierarchy of environments is set containing the Enterprise, Process and Activity environments in strict inclusion order, so that each acts as an environment for the previous.

There is a considerable amount of work still to be done in order to establish the full properties of a modelling architecture such as CIMOSA within the categorial setting we used. Of all the issues which lie pending, we would single out two: a set of criteria for model correctness with respect to a given architecture and enterprise, and the inclusion of a state characterisation of objects as category morphisms, with state transition maps to describe dynamic behaviour. The former issue would be of considerable interest, as, to this day, no answer is a priori available on whether a certain modelling architecture is sufficient to provide a model for a certain enterprise.

REFERENCES

1. AGUIAR, M. W. C., WESTON, R. H. and COUTTS, I. C., **Manufacturing Systems Design And Implementation Based on Formal Modelling Methods and Tools,** Proceedings of the 27th International Symposium on Automotive Technology and Automation (ISATA), Germany, 1994.

2. BAUMGARTEN, B., **On Internal and External Characterisations of PT-Net Building Block Behaviour**, Advances in Petri Nets, SPRINGER-VERLAG, 1988.

3. BERNUS, P. and NEMES, L., **A Framework To Define A Generic Enterprise Reference Architecture and Methodology**, Proceedings of ICARV'94, Singapore, 1994.

4. CEN/TC 310/WG1, **CIMOSA Document R0493/0,** Denmark, 1993.

5. CEN/TC 310/WG1, **Contributions from QCIM and CIMOSA - Comparison CIMOSA - IEM Modelling Constructs. Document no. 41**, Denmark, 1994.

6. CEN/TC 310/WG1 - ENV 40003, **Framework for Enterprise Modelling**, 1991.

7. CLEMENTS, P., COUTTS, I. and WESTON, R.H. **A Life-Cycle Support Environment Comprising Open Systems Manufacturing Modelling Methods and the CIM-BIOSYS Infrastructure Tools**, Symposium on Manufacturing Automation Programming Language Environments (MAPLE), Ottawa, Canada, 1993.

8. EHRICH, H.D., GOGUEN, J.A., and SERNADAS, A., **A Categorial Theory of Objects As Observed Processes**, Foundations of Object-Oriented Languages, REX School/Workshop Proceedings, May/June 1990.

9. ESPRIT Consortium AMICE, **CIMOSA, The Formal Reference Base**, Releases 1991, 1994.

10. ESPRIT, VOICE II. **Validating OSA in Industrial CIM Environments by Integration and Implementation**, Final deliverables, EP 6682, February 1995.

11. FOX, M.S., **The TOVE Project: Towards A Common-Sense Model of the Enterprise**, Enterprise, in C.J.Petrie Jr. (Ed.) Integration Modelling, Proceedings of the first International Conference, MIT PRESS, USA, 1992.

12. FOX, M.S. and GRUNINGER, M., **Ontologies for Enterprise Integration**, Proceedings of the second Conference on Cooperative Information Systems, Canada, 1994.

13. FULTON, J.A., **Enterprise Unification Using the Semantic Unification Meta-Model**, in C.J.Petrie Jr. (Ed.) Enterprise Integration Modelling, Proceedings of the first International Conference, MIT PRESS, USA, 1992.

14. GRAI Laboratory, **Study of the Conceptual Information Model for An Advanced Factory Management System Defined for Two Work Centres: Inspection and Tooling**. Final Report (R-84-FM-01). France, 1984.

15. HOARE, C.A.R., **Communicating Sequential Processes**, PRENTICE-HALL, 1985.

16. ISO/TC184 SC5 WG1, **Framework for Enterprise Modeling**, 1994.

17. K. Kosanke (Ed.) **Proceedings of the Workshop on CIMOSA**, CIMOSA Club, Germany, 1994.

18. KOTSIOPOULOS, I.L., **Modelling, the ELVAL S.A. Case. The Function View at Particular Design Level**, ESPRIT Project 5510, VOICE, Final Deliverable, January 1993.

19. KOTSIOPOULOS, I.L., **Theoretical Aspects of CIMOSA Modelling**, CIM Europe Conference, Amsterdam 1993, published in C. Kooij C., P.A. MacConaill and J. Bastos (Eds) Realising CIM's Industrial Potential, IOS PRESS, 1993.

20. C.J. Petrie Jr (Ed.) **Enterprise Integration Modelling,** Proceedings of the first International Conference, MIT PRESS, USA, 1992.

21. F. Waskiewicz (Ed.) **OMG Manufacturing, SIG Minutes**, Nshua/NH, USA, October 1994.

22. WILLIAMS, T.J. *et al*, **Architectures for Integrating Manufacturing Activities and Enterprises**, COMPUTERS IN INDUSTRY 24, 1994.

23. WILLIAMS, T.J., **The Purdue Enterprise Reference Architecture**, ibid.

24. T.J. Williams (Ed.), **Architectures for Integrating Manufacturing Activities and Enterprises**, IFAC/IFIP Task Force on Architectures for Integrating Manufacturing Activities and Enterprises, USA, 1993.