Object-process Analysis: Maintaining the Balance Between System Structure and Behaviour

DOV DORI, *Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa 32000, Israel.*
E-mail: dori@ie.technion.ac.il

Abstract

The object-process analysis (OPA) methodology combines ideas from object-oriented analysis (OOA) and data flow diagrams (DFD) to model both the structural and procedural aspects of a system in one coherent frame of reference. This is contrary to conventional object-oriented approaches, that use different tools to describe the structure and the behaviour of the system. The underlying observation of the OPA paradigm is that every thing in the universe of interest is either an object or a process, and that a process is not necessarily a method of a single object class. This opens the door for the possibility of modelling systems so that both their structural and procedural relations are represented within the same frame of reference, without suppressing each other. The two major differences between OPA and OOA are OPA's detachment of processes from objects and its recursive scaling capability. The OPA methodology has proven to model faithfully complex systems, such as computer integrated manufacturing, documentation and inspection and an intelligent computer-assisted instruction shell. This work lays down the foundations of OPA. It provides concise definitions of the basic building blocks of the method: objects, processes, classes, features, and the structural and procedural relations among them. The object-process diagram (OPD) is presented as an effective visualization tool that incorporates elements from both DFD and OOA. Due to synergy, both the information content and expressive power of OPD are greater than those of DFD and OOA diagrams combined.

Keywords: Information systems analysis, software engineering, object-oriented analysis, ontology, system analysis paradigms, structure–behaviour integration.

1 Introduction

System analysis is the discipline that attempts at modelling a portion of reality as seen from a particular viewpoint in order to understand existing systems and/or improve systems and design new ones. In this regard, it is closely related to the discipline of ontology—a branch of philosophy concerned with articulating the nature and structure of the world [1, 2, 16].

Systems (defined precisely below) exist in virtually any conceivable area of science and technology, including physics, social sciences, life sciences, health sciences and computer science. All systems have in common the feature that they consist of a collection of structured things that interact with each other in a meaningful way.

Object-process analysis (OPA) [8] is an ontology based on the distinction between objects and processes and the recognition that a model of reality must accommodate both and show how objects interact with each other through processes. While this may seem like a setback to the early function/data approaches, the clear-cut distinction between objects on one hand and processes on the other hand makes it possible to incorporate both the structural and dynamic aspects of the system under consideration within one frame of reference, thereby avoiding the multiplicity of models, each with its own diagramming technique and set of symbols.

A primary motivation for the development of the object-process paradigm has been to restore the balance between objects and processes, or between the structural and procedural aspects of a system. Another motivation has been to provide a scaling capability as a means to control the visibility of complex systems' details and enable a top-down representation.

Object-process analysis results in a set of object-process diagrams (OPDs), that unify the system structure and behaviour. OPDs are amenable to recursive up- and down-scaling to show different levels of detail and refinement. Yet, at any level, they have the same notation and semantics.

OPA lends itself to modelling any kind of system, regardless of whether it is mostly static or dynamic in nature and independently of the domain area under investigation. We have used OPA to analyse systems as diverse as trauma team training [7], intelligent computer-assisted instruction shell [9], R&D productivity measurement [13], understanding engineering drawings [6], and incorporating machine vision into computer integrated manufacturing [8].

1.1 Procedural- versus object-oriented approaches to systems analysis

A considerable number of paradigms of system analysis have been suggested. Rooted in the information systems domain, early approaches have advocated the information-flow idea, whose graphic representation is the data-flow diagram (DFD) [5]. The DFD approach emphasizes the processes that occur within a system and the flow of data among these processes. What is currently referred to as 'objects' are the 'external entities' and 'data stores' of DFD. DFD is implicitly solution-driven: through the use of terms from the solution space, such as data flows and data stores, rather than terms of the problem domain, it tries to push towards a design, before all the intricacies of the system being analysed have been fully considered and understood.

As systems became more complex, the DFD approach was gradually turned down in favour of the currently accepted object paradigm. As an interim approach, entity-relationship diagrams (ERD) [4], that model almost exclusively the static-structural aspects of a system, became popular as a means of data modelling and has been used extensively for relational database design. ERD is built from entities, recognized as objects in OOA, and the relationships among them.

Shifting the emphasis from processes to objects has been a step in the right direction, because objects, the basic building blocks of OOA, are as fundamental as processes. To a certain extent, objects and processes can be viewed as the dual or complement of each other. The argument for these statements is that a process requires at least one object in order for it to take place, and has an effect on at least one object (possibly the same one). On the other hand, creation of an object is a process, without which the object would not exist. Similarly, an object disappears following a process of destruction. Processes can be viewed as the 'glue' that holds a collection of objects together through interaction, such that they can be considered a system.

The dynamic, procedural aspect of a system is modelled in OOA through services (methods), that are attached to objects, and a mechanism of message passing among objects. Object-based analysis is a tool that usually yields a more complete analysis than the one obtained from process-based methods. However, a side effect of the switch from processes to objects has been the suppression of the procedural aspects of the system under consideration, since a process in OOA exists solely as a service of some object. This has been a major stumbling block in OOA, because in most systems the occurrence of many processes involves the participation of more than one object.

The conventional wisdom has been that there is an inherent dichotomy between object-oriented and process-oriented (data-flow) approaches, and one has to make a choice and give up one in

order to be able to work with the other. Although some OOA techniques do use DFD as part of their method, this is done as a separate analysis activity. For example, Shlaer and Mellor [15] use what they call 'Action DFD' (ADFD) as a last step in an object-oriented analysis process, but the DFD is not truly integrated into the OOA. Coad and Yourdon [3] use 'service charts' which are flow charts that provide for loops. Rumbaugh *et al.* [14] (p.88) also state that 'The functional model shows how values are computed, without regard for ... object structure. Data flow diagrams are useful for showing functional dependencies'.

2 The object-process analysis methodology

Contrary to the belief that DFD and OOA cannot be merged, at the center of the object-process analysis (OPA) paradigm is the fusion of structural and procedural aspects of a system within one frame of reference, using the same graphical tool—the object-process diagram (OPD). OPA stems from the distinction between objects and processes as two types of things, or items, or entities, existing in the world, or universe of interest. It is based on the recognition that a model must accommodate both objects and processes and show how they relate to and interact with each other, such that both the structural and the procedural aspects of the system are adequately represented. To this end, OPA combines the advantages of object-oriented and process-oriented approaches.

The initial motivation for the development of this paradigm has been the attempt to model transformations among product representations [8]. A meta-description of the methodology has been provided in that work along with an example of an analysis of a complex computer-integrated manufacturing, documentation and inspection system.

3 DFD, OOA, and OPA: an illustrated comparison

This section provides an example that demonstrates the analysis of the waterfall model of an information system lifecycle, using three different methodologies:
(1) a process-oriented (data flow diagram—DFD) analysis approach,
(2) an object-oriented analysis (OOA) approach, and
(3) an object-process analysis (OPA) approach.

The waterfall model advocates a top-down, carefully planned, linear development and implementation of an information system. Customer requirements are used as a basis for analysis, which is then used for design and, subsequently, for implementation—coding, testing, delivery and maintenance.

This 'benchmarking', in which the three methodologies are applied to the same case, provides a good perspective of the strengths and weaknesses of each one of them.

3.1 The process-oriented, DFD approach

Figure 1 is a DFD of the waterfall model. The requirements document is the 'external entity', which is the basis for the analysis. The results of the analysis process are stored in the 'analysis document' data store, and are read as input for the design process. This process, in turn, yields a 'design document', which is used for implementation. Finally, the implementation produces a program and its documentation.

As is evident from its name, the DFD is data- and process-oriented, and does not concern itself with the objects that make the processes happen, except for 'external entities'.

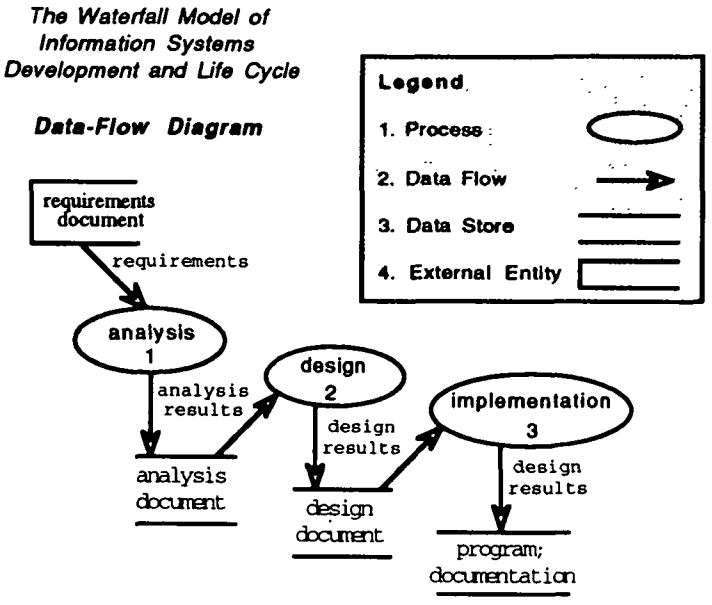

FIG. 1. A DFD of the 'waterfall model of information system development'

3.2 The object-oriented analysis approach

Figure 3.2 is a scheme of an object-oriented analysis of the waterfall model. Here, the emphasis is on the objects that perform the processes—the analysis, design and implementation teams, and the respective objects (products, or deliverables) of their endeavours—analysis document, design document, and program with its documentation.

The structural aspect of the system is well represented: the project team is an aggregation of the design, analysis, and implementation teams, and a product deliverable is a generalization of the resulting analysis document, design document, and program. However, the procedural aspect of the system is suppressed. It is revealed only through the services recorded at the bottom compartment of the relevant objects and the message connections that emanate form them. Explicit processes do not exist, and so it is not possible to infer cause and result of any process. There is no direct way, for example, to attach the object 'requirements' to any other object in the system.

3.3 The object-process analysis approach

Figure 3is an OPD of the waterfall model. Not only do processes from DFD and objects from OOA co-exist in harmony in the same diagram, but they even complement each other and enhance the overall understanding of the system. Cause and result are explicitly expressed through the procedural relations that exist among objects and processes. Thus, the object *Requirements* is needed for the process *Analysis*, which is done by the *Analysis Team*. The agent link, (see legend of Figure 33.3) is a specialization of a procedural (object-process) link. It explicates the agent—the object that is responsible for the execution of the process. The process *Analysis* yields the object *Analysis Document*, which is used for the process *Design*, done by the agent *Design*

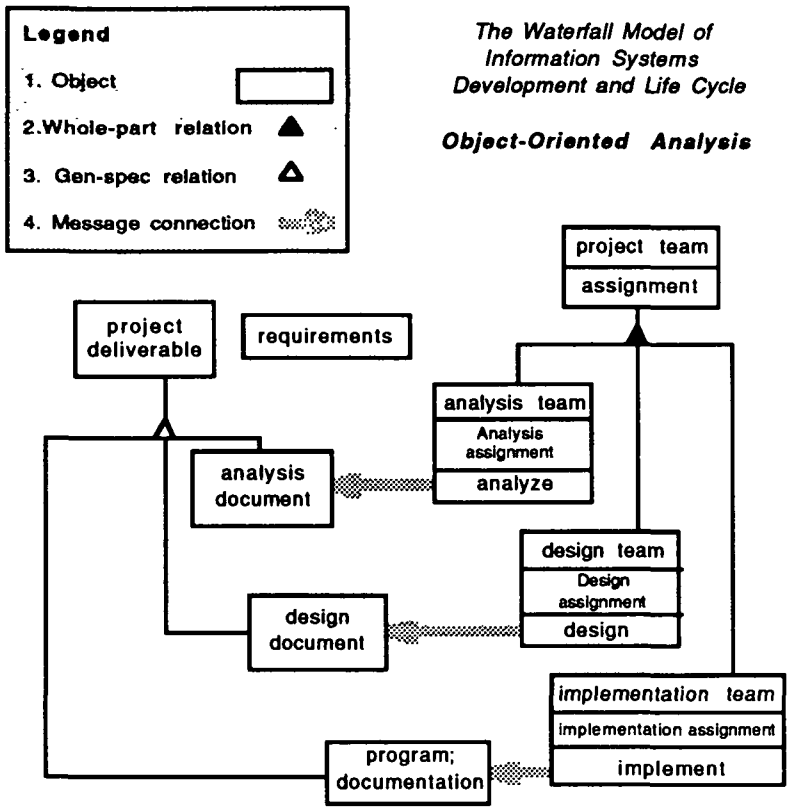

FIG. 2. An object-oriented analysis of the 'waterfall model of information system development'

Team, etc.

The structural relations in OPA include the basic structural relations generalization and aggregation, which exist also in the OOA model. These relations are so prevalent that they are denoted by special symbols. OPA provides also for general structural relations whose names are written explicitly along the line connecting two things of the same persistence. Such are the bi-directional structural link 'instructs/consults' between *Design Team* and *Implementation Team* and the unidirectional structural link 'reflects' from *Analysis Document* to *Requirements* shown in Figure 3. A unidirectional structural link exists when the reverse direction is just the passive voice of the original direction, as 'is reflected by' from *Requirements* to *Analysis Document* . These generalized structural links enhance the expressive power of OPA. A similar type of link has been proposed by Rumbaugh *et al.* [14] in their 'object model', but the direction of the link in their scheme is not denoted and must be inferred from the context.

3.4 OPA versus OOA analytical techniques: an illustrated comparison

The two major differences between OPA and OOA are OPA's detachment of processes from objects and its recursive scaling capability.

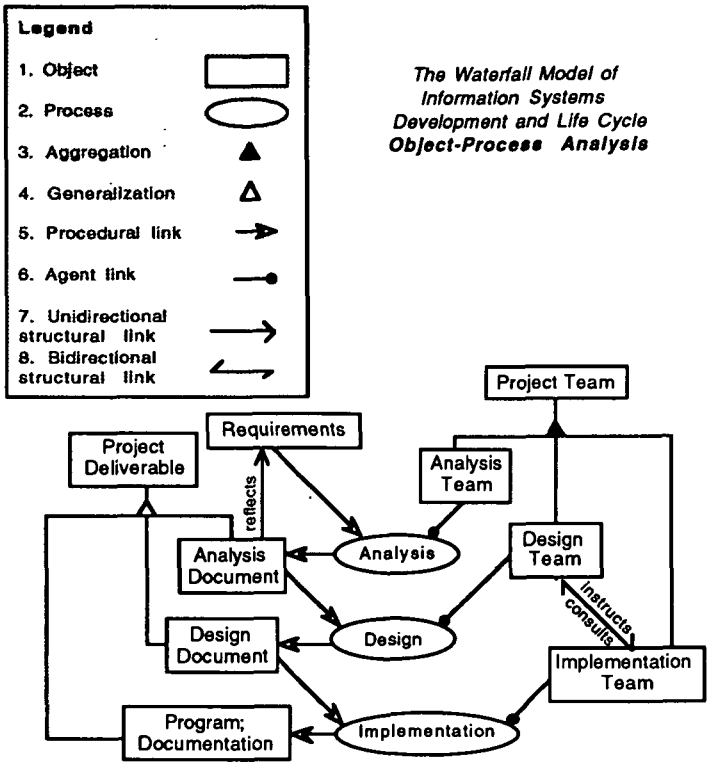

FIG. 3. An object-process diagram (OPD) of the 'waterfall model of information system development'

The detachment of processes from objects is a direct consequence of the observation that in many cases, non-trivial processes cannot be uniquely associated with a single object, or class of objects. OOA heralds encapsulation—the packaging of procedures ('services' or 'methods')—and enforces the attachment of each procedure to one particular object class. In complex systems, though, it is often the case, that a process comes into being only in the presence of, and activation by more than one object. In such cases, the choice of which object to attach the service to, is bound to be arbitrary, as it is impossible to pinpoint one particular object that is solely 'responsible' for the process. Consider, for example, the service *Manufacture*. What object class should it be attached to? *Raw material*? *Craftsman*? *Machine*? *Product*? *Model*? None of the above can be assigned this service alone. Manufacturing is a process that, in a certain universe of interest, entails a coordinated sequence of operations. In this sequence, raw material, craftsman, machine, and the model of the product to be manufactured, are all objects that need to be present and participate in the manufacturing process. This process, in turn, results in a new object—the product.

The mechanism of message passing among objects is the way OOA methodologies provide for interaction among objects. As the example below shows, this mechanism frequently yields awkward modelling. In OPA, processes are not confined to be services or methods of any particular

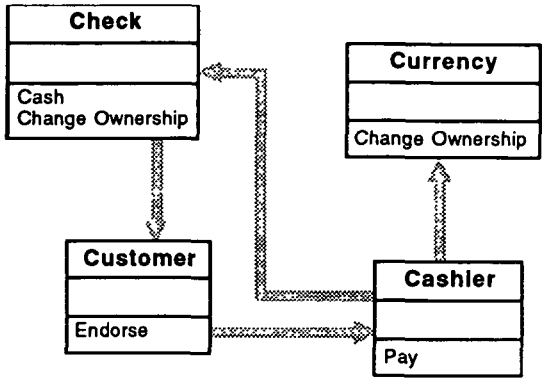

FIG. 4. Cash-a-check system: an OOA modelling

object. Moreover, the effect of an OPA process is to change the state (defined precisely below) of at least one object. This makes OOA's mechanism of message passing among objects unnecessary. Avoiding message passing prevents the difficulty this mechanism potentially introduces. This, in turn, enhances the expressive power of OPA.

The second major difference between OOA and OPA is the recursive scaling possibility of OPA. Scaling in OOA is quite limited and is not built as an integral part of the OOA philosophy. Groups of objects are termed 'subjects' [3], or 'domains' [15], but these are external concepts that are introduced for the purpose of enabling easier control over complex systems with a large number of objects.

In OPA, there are various scaling options: folding, unfolding, explosion, and implosion [8]. These options, which are defined in the sequel, arise naturally by the definition of any thing—be it object or process—as being composed of lower-level things that interact among themselves. This enables scaling the analysis both up and down.

Scaling down helps viewing the 'big picture' of the system. This is achieved by recursively grouping things into higher-level things, until a 'universe-of-interest diagram', showing the system and its interacting environment, is obtained.

Scaling up entails a recursive decomposition of things into lower-level things. The recursion halting condition is met when all the things become simple, i.e. no thing in the universe of interest needs to be further decomposed in order for the system to be fully understood.

The strong scaling mechanism of OPA is a second source of expressive power of the method, as it is used as a means of controlling the visibility and level of detail of any thing with respect to any other thing in the system.

To show the analytical techniques used with OPA contrasted with those used with OOA, we consider the Cash-a-check system. This simple system involves modelling the routine scenario of a customer cashing a check from a bank cashier. Analysis is done first using OOA then OPA. To provide an overview of major OPA characteristics, liberal use is made of terms that are defined formally later in this work.

In OOA, the objects in the system are Customer, Cashier, Check and Currency. All the processes must be modelled as services that the objects are capable of performing. Hence we must assign the services 'endorse', 'cash', and 'pay' to specified objects. We model 'endorse' as a service of Customer, 'cash' as a service of Check, and 'pay' as a service of Cashier. The service

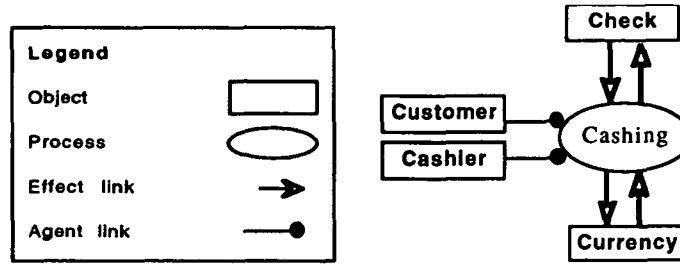

FIG. 5. Cash-a-check system: an OPA modelling, top level

'cash' of Check sends a message that triggers the service 'endorse' of Customer, which, in turn, triggers the service 'pay' of Cashier. The latter causes Currency to be transferred from Cashier to Customer and Check—from Customer to Cashier. To model this, we introduce the additional service Change Ownership for both Check and Currency. This OOA modelling of the system is expressed in Fig. 4, using the notation of Coad and Yourdon [3].

In OPA, the same four objects exist. Taking advantage of OPA's scaling capability, the system is analysed at two levels. At the top level, a single process, Cashing, exists. As the object-process diagram (OPD) in Fig. 5 shows, the four objects must be present in order for this process to occur. As in OOA and DFD, objects and processes are denoted by rectangles and ellipses, respectively. Both Check and Currency have procedural links—arrows to and from Cashing. This denotes the fact that these two objects undergo a change of state as a result of the process. The two other objects—Customer and Cashier—do not change their state. Rather, they are agents—intelligent objects that enable the Cashing process. This is expressed graphically by the agent link—a line from the agent to the process ending with a black circle.

To see more details of the Cashing process, this process undergoes explosion, and the result is shown in the OPD of Fig. 6. Explosion is a type of up-scaling that follows the idea of explosion of a process in DFD. When exploded, the thing (object or process) that is scaled up disappears, and its constituents, with their procedural links to other things, are shown instead.

Thus, in the scaled up OPD of Fig. 6, the process Cashing is modelled as consisting of two lower-level processes: Endorsement and Payment. The large gray ellipse in the background is a reminder of the higher-level process. To explicitly show the effect of each one of the two constituent processes, affected objects appear with their attributes whose values change. Both Check and Currency feature the attribute 'Owner', with values 'Customer' and 'Cashier'. In addition, Check is characterized by the attribute 'Endorsement Status', with values 'not endorsed' and 'endorsed'.

In an OPD, an earlier process is drawn above a later one. Hence, Cashing starts with Endorsement, which requires Customer as an agent and Check as the affected object. The effect of Endorsement on Check is to change the value of its Endorsement Status attribute from 'not endorsed' to 'endorsed'. Endorsement is followed by Payment, the effect of which is to change the value of Check's Owner from 'customer' to 'cashier' and the value of Currency's Owner from 'cashier' to 'customer'.

Comparing the techniques used to analyse the Cash-a-check system in OOA versus OPA, one major difference is the order of reference to the system's structural and procedural aspects. OOA basically requires a linear order, in which the objects in the system must first be identified. The

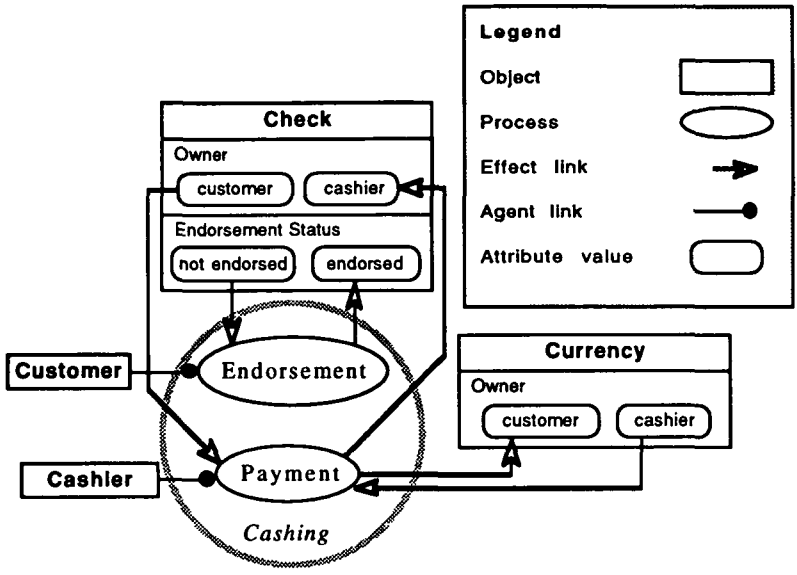

FIG. 6. Cash-a-check system: a scaled-up object-process diagram

procedural aspect of the system is then modelled through services of objects and messages that are passed among them. Contrasted with this scheme, OPA enables concurrent analysis of the structural and the procedural aspects of the system. The result is a model in which structure and behaviour are integrated. In addition, OPA's built-in recursive scaling capability provides for bi-directional transitions among descriptions with various detail levels.

3.5 Experience with OPA

Our experience with the OPA methodology has been very favourable. As noted, the first OPA-based work [8] deals with transformations among product representations. Since then we have employed this methodology in a variety of areas, including the design of an intelligent computer-assisted instruction shell, groupware-based team training shell design [7], design and implementation of a CD-ROM database for OCR (optical character recognition) performance evaluation, development of an intelligent computer-assisted instruction module [9], and automated engineering drawing understanding.

The method lends itself easily to modelling any kind of system, regardless of whether it is mostly static or dynamic in nature. The granularity of the details is controlled by the various scaling options. We found no resistance to the natural train of thoughts we were trying to model using this method, and the results have been very enlightening in their reasoning and explanatory power.

OPA is taught currently in an undergraduate software engineering course at the Technion, and is most welcome by the students. As a semestrial project, teams of two to three students each, were asked to analyse a system of a large company that provides world-wide warehouse and transportation services. The students were given a free choice of selecting an analysis methodology from among DFD, OOA and OPA. They were told that the grade would not be

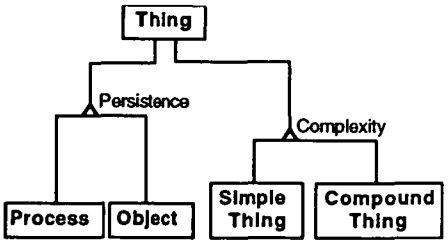

FIG. 7. Complexity and persistence: the two specialization dimensions of thing

influenced by this choice. Out of 21 team projects submitted, 13 (62%) were done using OPA, 6 (27%) were done with OOA, and 2 (11%)—with DFD. Considering the fact that the students were familiar with both DFD and OOA from a previous course, these results potentially testify to the appeal of the system. Students confirmed this positive attitude by a number of encouraging verbal comments.

4 Basic definitions

This section provides basic definitions to the terms used throughout the work. We define objects, processes, features, classes and systems, and make the distinction between structural and procedural relations.

4.1 Things are objects and processes

A *thing* is the elementary unit that makes up the universe.

The term 'thing' follows the terminology of the Bunge–Wand–Weber representational model [16], which is based on Bunge's work [1], [2].

The *universe of interest*, or *world*, is the realm that contains all the things that are relevant to the system under consideration.

As expressed in Fig. 7, things are classified on the basis of two *specialization dimensions* (classification criteria) [10]: persistence and complexity. Each of these two specialization dimensions is recorded in Fig. 7 next to a blank triangle, which is an accepted symbol [12, 14] for generalization-specialization.

In the persistence dimension, things are classified into objects and processes according to their persistence—life-time duration—in the universe of interest.

An *object* is a persistent, unconditional thing in the universe of interest. After being created, an object exists unconditionally, until it is deliberately destructed.

A *process* is a transient thing in the universe of interest, whose existence depends on the existence of at least one object.

From the design and implementation viewpoint, an object can be regarded as a variable with a specified data type, while a process is a function or a procedure operating on variables, which are the objects.

In the complexity dimension, a thing may be either simple or compound.

A *simple* (also referred to as *primitive* [16] or *atomic*) thing is a thing that, in the context of the universe of interest, needs not be decomposed into smaller constituents, nor described in further

detail, as its constituents are not required for analysis even at the most detailed level.

A *compound* (also referred to as *composite* [16]) thing consists of, and/or is characterized by one or more other things. In the context of modelling the universe of interest, for the analysis to be complete and comprehensive, a compound thing requires further specification of its constituents.

From the design and implementation viewpoint, a simple object can be regarded as a variable with a simple data type (integer or floating point number, string, etc.). A simple process can be viewed as a trivial, readily implementable function or procedure, operating on a set of simple things, such as averaging a set of numbers, concatenating strings, etc.

The persistence and complexity specialization dimensions are orthogonal to each other. Their Cartesian product gives rise to four types of things: simple objects, simple processes, compound objects, and compound processes. This is a small demonstration of *multiple inheritance* and the combinatorial explosion associated with it. In conventional multiple inheritance, an object that inherits features (attributes and services) from a number of objects, must acquire all the features from all the inheriting objects. In practice, however, it is frequently desirable to enable the specialized object to inherit features selectively from a subset of the inheriting objects. The combinatorial explosion problem arises when this mechanism is introduced. A complex web is formed when trying to depict all possible inheritance combinations in a diagram. The graphic representation that solves the display problem of the combinatorial explosion is a selective multiple inheritance symbol. The minimal and maximal number of inheriting classes, denoted within the symbol, provide for a succinct description of a large number of possibilities. A detailed discussion of the problem and the proposed solution appears in [10, 11].

4.2 System: a compound object

Having distinguished between objects and processes and between simple and compound things, we apply their definitions to characterize the terms system, environment and the interaction between them.

Interaction is a process of exchanging matter and/or energy and/or information among objects in the universe of interest.

A *system* is a compound object of particular interest within the universe of interest. As a compound object, a system is a collection of two or more objects that interact through one or more processes in a meaningful, potentially predictable way.

Environment is the collection of all the objects in the universe of interest which are not part of the system, but interact with it.

Viewing both a system and its environment as objects in the universe of interest, these two objects are related to each other through the process of interaction.

An *information system* is a system that models another system.

By this definition, an information system can model another information system. Indeed, any financial or accounting information system, for example, models the financial status and transactions of monetary objects, which are themselves information on their owners' wealth.

5 OPD: the object-process diagram

A system can be described either verbally or graphically. Verbally, it is defined by a set of one or more statements. Graphically, it is depicted by a corresponding, equivalent object process diagram.

An *object-process diagram* (OPD) is the graphic representation of objects and processes in the

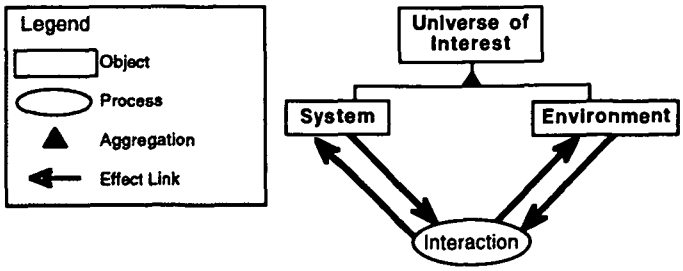

FIG. 8. The generic universe of interest OPD

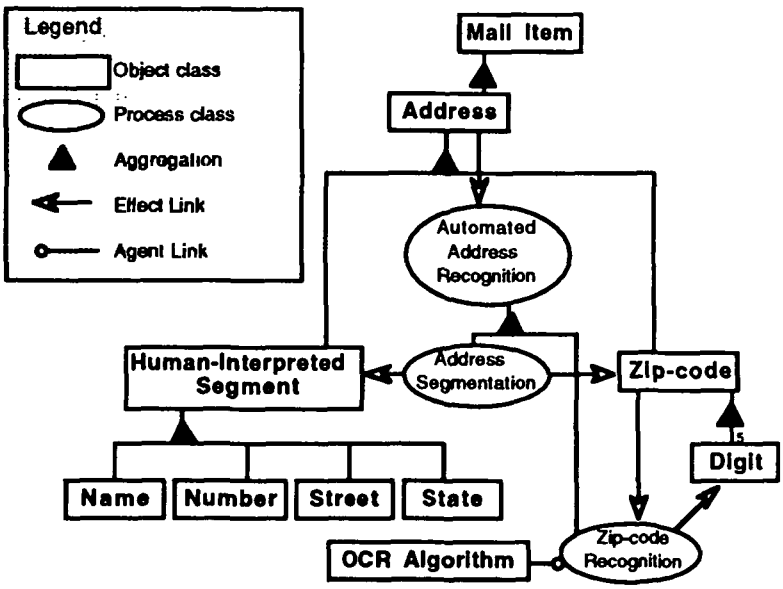

FIG. 9. An OPD of the Automated Mail Sorting system

universe of interest and relationships among them.

Figure 7 is an example of an OPD. A single class is the simplest, *trivial* OPD. Figure 8 is a *universe of interest OPD*. This is a generic, top-level OPD, showing the system and its environment as constituent objects of the universe of interest, between which the process of interaction occurs. This diagram is analogous to DFD's 'content', or 'zero-level' diagram. For any particular universe of interest, this generic diagram is instantiated into the actual system and its environment.

The symbols used in the OPD adhere to accepted notations in information systems analysis. Objects and processes are denoted by rectangles and ellipses ('bubbles'), borrowed from OOA and DFD, respectively. Following [12] and [14], the black triangle stands for structural aggregation relation, and the arrows—for procedural, object-process links. More symbols of the OPD's alphabet are introduced in the sequel and are included in the diagrams' legends, as necessary.

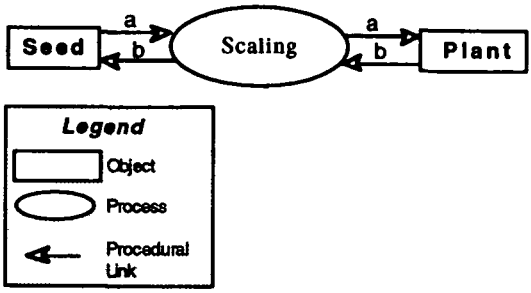

FIG. 10. Scaling converts a seed to a plant and vice versa

EXAMPLE 5.1
A Semi-Automated Mail Sorting system is a compound thing. It consists of the object *Mail Item* and the process *Automated Address Recognition*. Figure 9 is an OPD describing the system. *Address* is a compound object. It consists of (or 'has') the two compound objects *Human-Interpreted Segment* and *Zip-code*. The former consists of the objects *Name*, *Number*, *Street*, and *State*, each of which is read by a human, and therefore needs no further refinement. Hence they are considered as simple objects. *Zip-code* is read by a machine, which has to correctly interpret each of its digits. Therefore, *Zip-code* is a compound object: it consists of five *Digits*.

Automated Address Recognition is a compound process, enabled by *Address*. It consists of *Address Segmentation*, which segments *Address* into the two *Address* parts, and *Zip-code Recognition*, which recognizes the *Digits*.

6 Recursive scaling of objects and processes

An important feature of objects and processes in OPA is that they are recursively scalable. Scalability provides for complexity management of systems through controlling the visibility and level of detail of things in the system. This section defines scaling-related concepts and demonstrates their use.

Scaling is a process of changing the level of detail of a thing (object or process). *Scaling up* is a process of increasing the level of detail of a thing. *Scaling down* is a process of decreasing the level of detail of a thing. *Seed* is the thing that is about to be scaled up. *Plant* is the set of things (objects and/or processes) that result in from up-scaling a seed.

Usually, we are interested in scaling one or more things within an OPD. Since an OPD is itself an object, it can be both a seed and a plant. Fig. 10 is the OPD showing the effect of Scaling.

A *procedural path* is a series of one or more consecutive procedural relations. Graphically, it can be traced in an OPD as a chain of procedural relations, denoted by arrows. A procedural path is referred to shortly as path or, as in [3], 'thread of execution'. Each one of the two letters 'a' and 'b' above the arrows in Figure 10 is a *path label*, denoting a different path. Path names are required for reference or whenever there is a potential ambiguity as to how paths should be sequenced. One has to follow a chain of procedural links with the same label until it is no longer possible to proceed before switching labels. Thus, when a Seed undergoes Scaling, we follow thread 'a' and the result is a Plant. When a Plant undergoes Scaling, we follow thread 'b' and the result is a Seed. Omitting the path labels would yield the undesirable paths Seed-Scaling-Seed and Plant-Scaling-Plant.

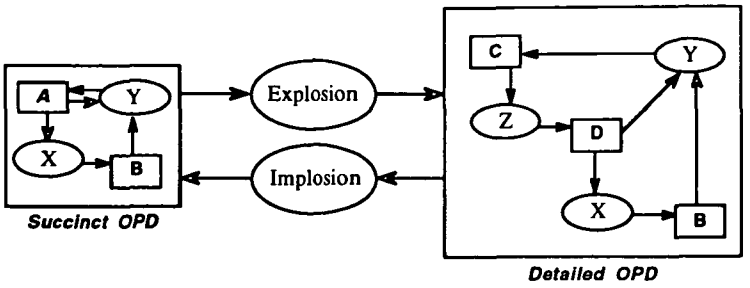

FIG. 11. Explosion of a seed—object A—in the succinct OPD to a plant—C, D and Z—in the detailed OPD and implosion in the reverse direction

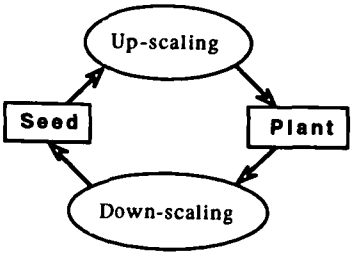

FIG. 12. The process Scaling of Fig. 10 exploded into Up-scaling and Down-scaling

6.1 Explosion and implosion

Explosion is an up-scaling process, in which the direct constituents of the seed are shown, but the seed itself is not preserved. *Implosion* is the inverse of explosion. It is a down-scaling of a set of things which are the constituents of the seed. As a result of an implosion, the seed shows up instead of its constituents.

Figure 11demonstrates explosion of a succinct OPD to a detailed OPD and implosion in the reverse direction. The succinct OPD shows the procedural relations among the objects A and B and the processes X and Y. A—the name of the seed object in the succinct OPD—is italicized, denoting the fact that A is a compound object. Exploding A results in the plant, in which A is replaced by its simple constituent objects C and D and the (simple) process Z.

In the detailed OPD, object A's constituents with their internal procedural relations are exposed. All three arrows from and to A in the succinct OPD still exist in the plant, but now they go from and to constituents of A.

In Fig. 12, the process Scaling of Fig. 10 is exploded, showing Up-scaling and Down-scaling as two specializations of Scaling. Note that the path names have been omitted, as they are not needed in this OPD.

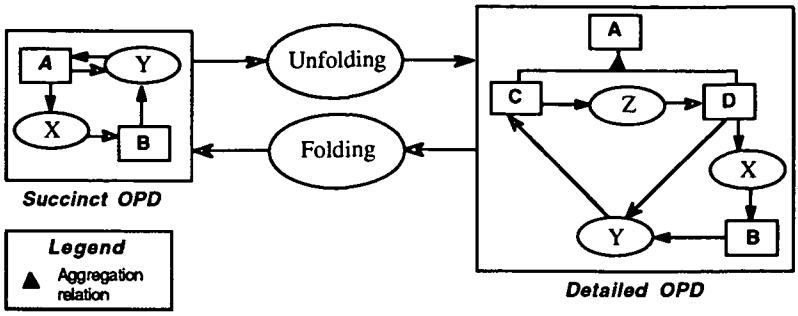

FIG. 13. Unfolding a succinct OPD to a detailed OPD and folding it back

6.2 *Unfolding and folding*

Unfolding is an up-scaling, in which the seed is preserved in the plant as the root of its constituents, showing explicitly the structural relations between them. *Folding* is the reverse of unfolding.

Figure 13 demonstrates unfolding A in a succinct OPD to a plant in the detailed OPD and folding in the reverse direction. By the definition of unfolding, A remains as the root of the aggregation structure. Using the black triangle as the aggregation symbol, the whole (object A) is connected to the tip of the triangle, and the parts (C and D) to its base. The procedural relations among C, D and Z are also drawn.

7 Classes and instances

By definition, a simple thing has no constituents. The constituents of a compound thing are of two types: parts and features.

A *part* of a (compound) thing t is a thing p that comprises t.

A temporary definition of feature, which is refined in the sequel, is as follows.

A *feature* of a (compound) thing t is a thing f that characterizes t.

The definition of feature is refined below, based on the definitions of class and instance.

EXAMPLE 7.1

In a universe of interest of aviation, *Airplane* is a compound thing. Some of its features are *Propulsion, Purpose, Landing-capability, Maximal Speed, Current Velocity, Maximal Weight, Current Weight, Home Base*, and *Current Location.*

Some of Airplane's parts are *Body, Wing, Engine, Tail* and *Undercarriage.*

A *name* is a symbol that is attached to every thing and conveys the semantics of what the thing is. By convention, names are capitalized, as in Example 7.1.

The semantics of each thing, which is referred to by its name, is expressed by its set of constituents (features and parts). Hence, all the things with the same name, have the same set of constituents and belong to the same class.

A *class* is a collection of all the things in the universe of interest that have the same set of constituents (features and parts).

OPA's definition of class is a two-way generalization of the usual OOA definition of class. First, OOA states that a class is a collection of all the objects having the same set of features (attributes and services). It does not refer to constituents (features and parts). Second, OOA refers to classes

of objects only, while OPA recognizes also classes of processes, as defined below.

Since a thing is either an object or a process, a class of things is either an *object class* or a *process class*. Like things, classes of things are categorized on the basis of the two specialization dimensions persistence and complexity. In the persistence dimension, there are object classes and process classes. In the complexity dimension, a class may be compound or simple: a *simple class* is a class of simple things, while a *compound class* is a class of compound things.

7.1 Instances: values and occurrences

An *instance* of a class is a particular thing that is a member of that class.

An instance is classified as either an object instance (value) or a process instance (occurrence), depending on whether it belongs to an object class or a process class, respectively, as follows:

An *object instance*, or *value*, is an instance of an object class.

A *process instance*, or *occurrence*, is an instance of a process class.

The terms value and occurrence reflect the distinction between an instance of an object and an instance of a process. 'Value' is persistent. A change of value amounts to replacing one object instance by another. The use of the term value for an object instance is particularly suitable when the object class is an attribute. 'Occurrence', on the other hand, reflects the volatility and time dependency of processes. An occurrence is a happening of a process that has a definite start, duration, and end along the time axis. Thus, it can never be repeated with the same start and end, even if it is performed on the same object instance, or value. Just as an object class is a template for all the object instances (values) in that object class, a process class is a scenario description, or pattern of behaviour, which all the occurrences (process instances) in that process class follow.

In OOA, 'object' is an intermediate concept between class and instance. It frequently refers to a typical representative of the class to which it belongs, but it is sometimes also used to refer either to the entire class or to a particular instance of that class. The meaning is usually understood from the context. Since things are referred to by their names, the name itself (e.g. Airplane), may mean a typical object with that name, or the class of airplanes.

OPA follows this implicit convention: When no ambiguity arises, the term 'object instance' can be abbreviated as either *object* or *instance*. Likewise, an object class can be abbreviated as either *object* or *class*. Process class and process instance, however, should not be abbreviated, to enable their distinction from (object) class and (object) instance.

7.2 Time granularity and identifiers

The *time granularity* of a universe of interest is the smallest period of time during which a meaningful process occurs in that universe.

The time granularity may differ significantly from one universe of interest to another. Consider two extreme examples. A universe of interest of geology has a time granularity of thousands or millions of years. During this period, a significant change in the state of relevant objects, such as a shift of continents with respect to each other, occurs. A sub-particle universe of interest, on the other hand, has a typical time granularity of nano- or pico-seconds, needed for an electron to jump from one atomic energy level to another.

The life-time duration of a process is normally in the order of magnitude of the time granularity of the corresponding universe of interest. The life-time duration of an object, however, is normally at least one order of magnitude longer than the time granularity of the corresponding universe of interest.

An *identifier* is a symbol that can potentially be attached to an (object or process) instance for the purpose of identification.

An *object identifier* is a pair consisting of the object class name followed by an object instance name—a label (which may be a serial number) that is unique for each object instance belonging to the object class. An object identifier enables unambiguous identification of the object in the system.

Duration of a process class is the amount of time that elapses from the start to the end of a typical occurrence in that class.

The duration is usually specified in the system's time granularity units. If all the occurrences have the same duration, it is a precise number. Otherwise, it is a random variable, with known or unknown distribution. If the distribution is known, any number of its parameters may be known or unknown.

A *time stamp* of an occurrence is an indication of the occurrence start time.

A *process identifier* is a unique label attached to each occurrence, that consists of the following elements: (1) the process class name; (2) the occurrence time stamp; and (3) a serial number, which is needed only if the the system under consideration allows multiple occurrences to start at exactly the same time.

The first element indicates what process class the occurrence belongs to. The second element differentiates among multiple occurrences of the same process class that start at different times. The third element differentiates among multiple occurrences that start at exactly the same time (either by the same object instance or different object instances). As noted, this element is omitted if the system under consideration does not allow such situations.

8 Features, states, attributes and services

The dynamic aspect of systems is manifested as a sequence of one or more changes in the state of the objects involved in processes. This section lays down the basic definitions for handling system dynamics. Prior to defining classes, a feature of a thing t has been defined in Section 7 temporarily as a thing f that characterizes t. The definition of class was based on that definition of feature. Having defined classes, the definition of feature is refined as follows.

A *feature* of a (compound) class c is a class f that characterizes c.

A *state* of a thing at a given point in time is the set (or vector) of feature instances the thing has at that point in time.

A feature can be an attribute or a service. An *attribute* is a feature which is an object class. A *service* is a feature which is a process class. A service, also referred to in OOA terminology as *method*, is considered in OOA as a mode of behaviour that a class is capable of, or responsible for exhibiting [3]. Since behaviour, like process, can be regarded as a change of state along time, the two definitions are in accord.

Features may characterize process classes just as they characterize object classes. For instance, a process might have the attributes *Duration*, *Place*, *Precondition*, *Trigger*, *Number of Resulting Objects*, etc.

Based on the above definitions, and referring to Fig. 8, *Interaction* may be considered as a service of *Universe of Interest*.

Figure 14 is an elaboration of the OPD in Fig. 7. It describes the relationships among classes and features. A compound class is displayed as comprising features, each of which is itself a class. As a class, it, too, has the two specialization dimensions Persistence and Complexity. To keep the OPD simple, only the Persistence dimension of Feature is shown. In this dimension,

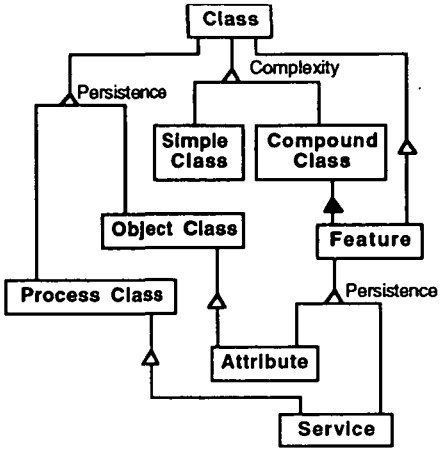

FIG. 14. An OPD of the relationships among classes and features

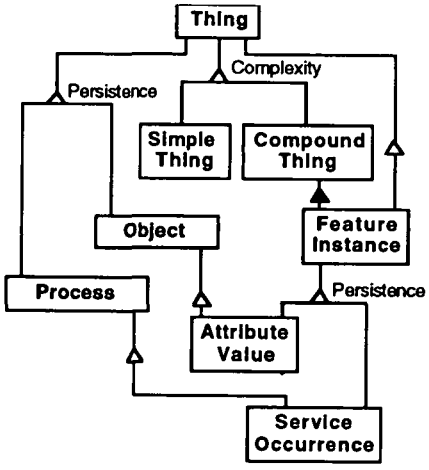

FIG. 15. An OPD describing the relationships among things and feature instances

the specialization of Feature includes Attribute and Service. At the same time, these are also specializations of Object Class and Process Class, respectively.

Just as feature specializes into attribute and service, feature instance specializes into attribute value and service occurrence.

An *attribute value* is an instance of the attribute.

A value has been defined as an instance of an object class. In particular, since attribute is an object class, value is an instance of attribute. For example, Height is an attribute of Person, and 176 cm is a value of Height for the object instance of Person whose identifier is 'Adam Adams'.

A *service occurrence* is an occurrence of the service.

Figure 15 is an OPD whose structure is exactly the same as the one in Fig. 14, but here *Thing*

appears instead of *Class* as the top-level object. As a result, Object Class and Process Class are replaced by Object and Process. Feature is replaced by Feature Instance, and Attribute and Service—by Value and Occurrence, respectively.

8.1 The roles of objects in a process: the pre-process and post-process object sets

Objects are things that both enable processes and result from them. From the point of view of a given process, each object is either needed for the process to occur and/or is affected by it. Focusing on a particular process, two object-sets are defined as follows.

A *pre-process object-set* of a process is a collection of one or more objects that are necessary for the process to occur.

A *post-process object-set* of a process is a collection of one or more objects that result from the occurrence of that process or are changed by it.

As noted, the occurrence of a process depends on the concurrent existence and participation of all the objects in the pre-process object-set, each in a certain state (defined below). As a result of the occurrence, at least one object is either created or undergoes a change of state.

The objects in the pre-process object-set are classified according to their role in the process into two types: enabling objects and affected objects.

An *affected object* of a process is an object that undergoes a change of state as a result of the occurrence of that process.

A change of state of an object includes a change in the value of its *existence* attribute from negative to positive (a process of construction) or from positive to negative (a process of destruction).

An *enabling object* of a process is an object that must be present in order for the process to occur, but, in the context of the relevant universe of interest, it is not affected as a result of this occurrence.

An *agent* is an intelligent enabling object that performs the process.

An agent is usually an individual human or a team, but it may also be a robot or any other object with a minimal amount of artificial intelligence.

An *instrument* is an enabling object that is used to perform the process.

An instrument may be information, tool, machine, algorithm, model, mold, data stored in memory that is not altered as a result of the process to which it serves as instrument, etc.

Since agents and instruments do not undergo any state transition (change of state), they are not mandatory in the analysis. However, they are frequently included for the sake of a complete description and clear understanding of the system.

In order for a process to be meaningful in the universe of interest, it must affect at least one object. Hence, any pre-process object-set must contain at least one affected object. Additionally, it may contain one or more enabling objects. If the affected object is destroyed and no other object results, then the post-process object set is empty.

The classification of an object into affected or enabling with respect to a process depends on the role it plays in the universe of interest, as demonstrated in the following example.

EXAMPLE 8.1

In a product lifecycle universe of interest, described in Fig. 16, the *Manufacturing* process has *Raw Material* as its affected object. *Raw Material* actually disappears in the process. *Model, Craftsman* and *Machine* are Manufacturing's enabling objects; none of them undergoes a state change as a result of the process. *Model*, which is an instrument, is not affected at all by the

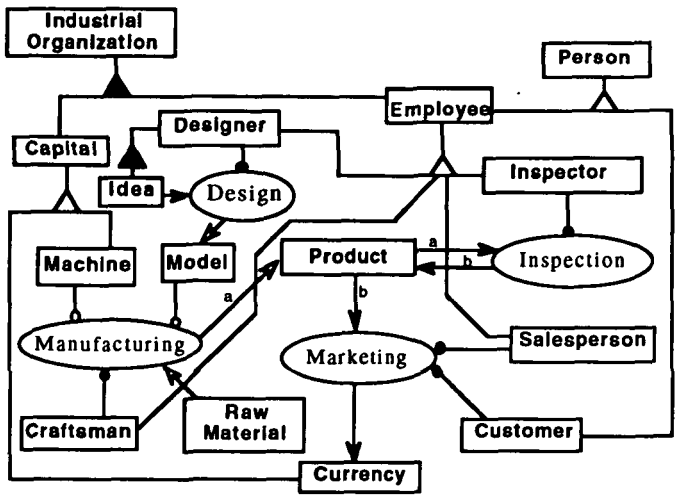

FIG. 16. OPD of an industrial organization and its product lifecycle

process. *Craftsman* is an agent and *Machine*—another instrument. *Craftsman* may be more tired and *Machine* more worn out as a result of the Manufacturing process. However, since machine wear-out and craftsman fatigue are not relevant attributes in the particular universe of interest of this example, *Craftsman* and *Machine* are not considered as being affected by the process. On the other hand, in a machine maintenance universe of interest, the interesting changes occur in the machine rather than in the raw material. Therefore, in this context, the same *Manufacturing* process would have *Raw Material* as its enabling object, while *Machine* would be the affected object. Finally, in a manufacturing plant universe of interest, where the product and the machine are of interest, *Product* and *Machine* are both affected.

A *resulting object* is an object that is constructed as a result of the process. A *consumed object* is an affected object that is destructed as a result of the process.

The effect of a process on a consumed object is to change the value of its existence attribute from 'positive' to 'negative'. The effect of a process on a resulting object is exactly the reverse. Note that by definition, a resulting object is not in the pre-process object-set. Similarly, a consumed object is not in the post-process object-set.

The pre-process and post-process object-sets of a process may or may not be disjoint. If they are disjoint, i.e. their intersection is empty, then all the affected objects in the pre-process set are consumed. Otherwise, the intersection of the two object sets contains the affected objects, which are also the resulting ones. Each such object 'survived' the process but its state changed.

EXAMPLE 8.2

With respect to *Inspection* in Fig. 16, *Product* is in both the pre-process and post-process object-sets. Graphically, this is shown by the two arrows (procedural links) between these two things. The letters 'a' and 'b' along the arrows denote procedural paths, which were defined in Section 6. The effect of Inspection on Product (not shown in the figure) is to change the value of its Status attribute from 'manufactured' to 'inspected'.

Figure 17 is an OPD describing the engineering design process and the resulting model. It

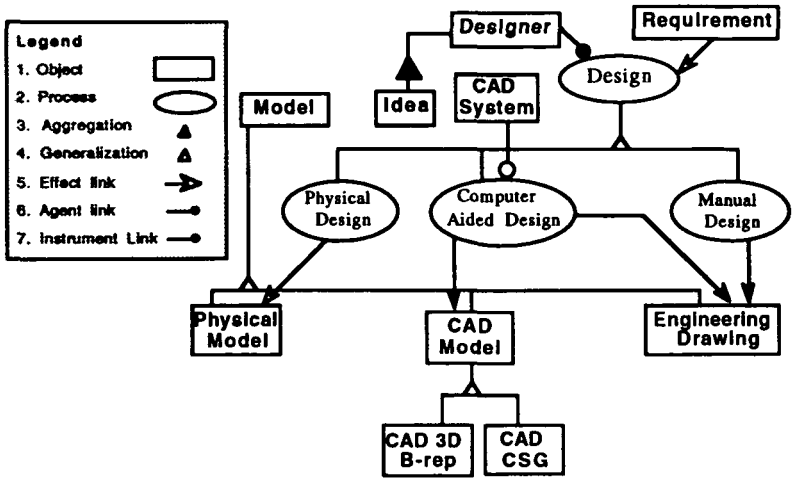

FIG. 17. An OPD of the engineering design process and the resulting model

has been obtained by up-scaling the object classes *Design* and *Model*, shown in Fig. 16. *Design* specializes into *Physical Design*, *Computer-Aided Design (CAD)* and *Manual Design*. These three types of design result in, respectively, *Physical Model*, *CAD Model* and *Engineering Drawing*, all of which are specializations of *Model*. *CAD Model* specializes into *Constructive Solid Geometry—CSG* and *Boundary Representation—B-rep*.

8.2 *Links and their specializations*

A *link* is a connection between classes. When two or more classes are present in the OPD, they are somehow linked. A non-trivial OPD consists of two or more classes and one or more links.

Links specialize into structural links and procedural links, according to their role in the OPD.

A *structural link* represents a static, long-term relation that persists in the system, like the fundamental relations aggregation and generalization. As noted, the symbol for the aggregation relation ('has-a') is a solid (black) triangle and the one for generalization ('is-a'), is a blank (white) triangle.

A *procedural link* shows the dynamic behaviour of the system by linking an object to a process or a process to an object.

Note that the classification of objects and their corresponding links into enabling, affected, and resulting is solely from the point of view of a given process; a resulting object may well be the enabling object of another process.

In the OPD of Fig. 8.2(A), Object, with respect to a particular process class, specializes into Affected Object, Enabling Object, and Resulting Object. Consumed Object is a specialization of Affected Object: It is destructed as a result of the process. Agent and Instrument are specializations of Enabling Object.

The OPD of Fig. 8.2(B) shows that a Procedural Link specializes into Effect Link (denoted by an arrowhead) and Enabling Link. The enabling link is directed from an enabling object to the process which the object enables. It specializes into Agent Link and Instrument Link, that end

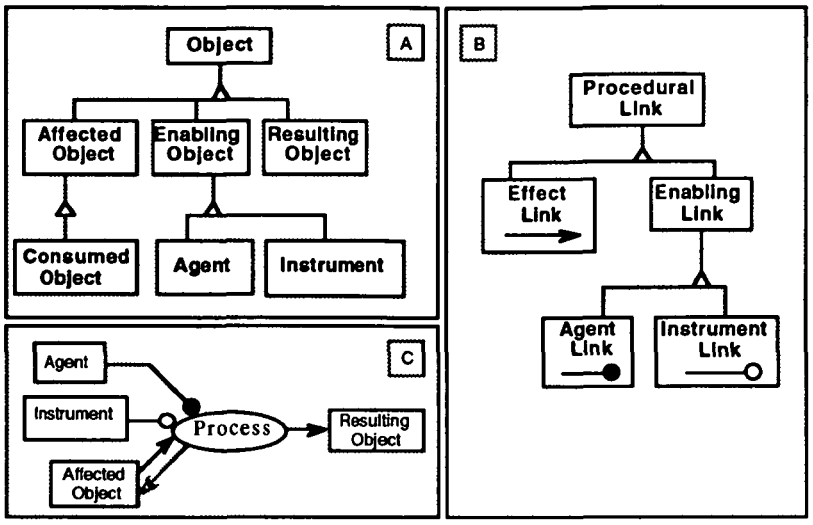

FIG. 18. Specialization of object classes (A) and procedural links (B) with respect to a process, and their usage (C)

with black and blank circles, respectively.

Figure 8.2(C) exemplifies the use of the procedural links. The effect link from Process back to Affected Object is gray rather than the usual black to denote the fact that it is not mandatory. It may be shown explicitly to stress the fact that the process has an effect on that object, but even if it is not drawn, it is implicitly there. This convention helps decrease the amount of graphic entities in an OPD.

9 Summary

A new paradigm in systems analysis, the object-process analysis (OPA) methodology, has been introduced, defined and illustrated. OPA is a fusion of two accepted approaches in information systems analysis: the object-oriented approach and the data-flow approach. These two approaches have been considered to be mutually exclusive. Nevertheless, as illustrated by a number of examples, not only does OPA manage to incorporate useful ideas from both approaches, but it also gracefully merges static-structural aspects borrowed from OOA with dynamic-procedural elements from the process-oriented DFD method. An object-process diagram (OPD) is a concise graphic representation of a system that clearly shows how object classes relate to each other, what process classes operate on them, and how one object class is transformed into another.

OPA is not restricted to the analysis of information systems. It suits any conceivable domain and has already been successfully implemented in the analysis of a variety of unrelated research and development areas in science, technology and education. Due to its logic and fitness to modelling real-life systems, it it my view that OPA is likely to become an accepted method for systems analysis and design.

Acknowledgements

The author wishes to thank the two anonymous referees for their valuable comments, which helped clarify subtle points and triggered new ideas. This research was supported by Technion V. P. R. Fund.

References

[1] M. Bunge. *Treatise on Basic Phylosophy: Volume 3: Ontology I: The Furniture of the World.* Reidel, Boston, 1977.

[2] M. Bunge. *Treatise on Basic Phylosophy: Volume 4: Ontology II: A World of Systems.* Reidel, Boston, 1979.

[3] P. Coad and E. Yourdon. *Object Oriented Analysis* (2nd edn). Prentice Hall, Englewood Cliffs, NJ, 1991.

[4] P. P. Chen. The entity relationship model—toward a unifying view of data. *ACM Transactions on Data Base Systems*, 1, 9–36, 1976.

[5] T. De Marco. *Structured Analysis and System Specification.* Yourdon Press, New York, 1978.

[6] D. Dori. Arc Segmentation in the Machine Drawing Understanding System Environment. *Proceedings of the Document Analysis Systems Workshop*, Kaiserslautern, Germany, October 1994, in press.

[7] D. Dori, M. Alon and Y. J. Dori. Team training shell: a groupware, multimedia-supported application generator. *Proc. ED-MEDIA 94—World Conference on Educational Multimedia and Hypermedia*, Vancouver, Canada, June 1994, 166–171.

[8] D. Dori, I. Phillips and R. M. Haralick. Incorporating documentation and inspection into computer integrated manufacturing: an object-process approach. In *Applications of Object Oriented Technology in Manufacturing.* Chapman-Hall, London, 1994, in press.

[9] Y. J. Dori and D. Dori. Object-process analysis of intelligent computer assisted instruction shell: the polymer courseware—a case in point. In *Multimedia and Hypermedia Educational Annual*, Proc. ED-Media conference, Vancouver, Canada, June 1994, 172–177.

[10] D. Dori and E. Tatcher. Selective multiple inheritance: coping with combinatorial explosion. *IEEE Software*, 77–85, May 1994.

[11] D. Dori and E. Tatcher. Embryonic classes: enabling selective multiple inheritance. *Journal of Object Oriented Programming*, 47–51, June 1994.

[12] D. W. Embley, B. D. Kurtz and S. N. Woodfield. *Object Oriented Systems Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[13] D. Meyersdorf and D. Dori. The R & D universe and its feedback cycles: an object–process analysis. Technical Report ISE-93-03, Israel Institute of Technology, Haifa, 1993.

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. *Object Oriented Modeling and Design.* Prentice Hall International Editions, Englewood Cliffs, NJ, 1991.

[15] S. Shlaer and S. J. Mellor. *Object Lifecycles Modeling the World in States.* Yourdon Press PTR Prentice Hall, Englewood Cliffs, NJ, 1992.

[16] Y. Wand and R. Weber. On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*, 3, 217–237, 1993.

Received 1 June 1993