INT. J. COMPUTER INTEGRATED MANUFACTURING, 1996, VOL. 9, NO. 5, 339–353

Object–process analysis of computer integrated manufacturing documentation and inspection functions

DOV DORI

Abstract. Manufacturing a product requires a variety of representations. These representations may be thought of as a continuum whose extremes are constructive solid geometry (CSG) on the analytical side and the physical product itself on the concrete side. In between these two extremes is a spectrum of representations, ranging from 3D- and 2D-boundary representations (B-rep), through wireframe to engineering drawings and images. Machine vision plays an increasing role in computer integrated manufacturing (CIM). We augment the notion of CIM to computer integrated manufacturing, inspection and documentation, and develop an object–process approach to explore the transformations among product representations and the way vision-based processes mesh in this complex environment. To analyse the CIM system with inspection and documentation augmentations, we use a new paprdigm in systems development—the object–process methodology, which fuses useful ideas from state-of-the-art object-oriented analysis with data-flow approaches to yield a modelling methodology that specifies explicitly both objects and processes. The paper explains the principles of the object–process methodology and applies it to describe the objects of the CIM–documentation–inspection system and the processes that change their states.

1. Introduction

Manufacturing of products has traditionally begun with a design specified by an engineering drawing. A product life-cycle has followed the path from the analytical to the concrete end, starting with analysis and design, transforming it to process planning, and ending with the actual manufacturing, inspection, marketing and maintenance. Modern approaches, such as reverse engineering, imply that other paths of transformations are also conceivable and have their own merits. Parallel advents in machine and robot vision and in image and engineering drawing understanding have opened options for new transformation paths.

As computer capability has increased, the engineering drawing specification has started to be done by CAD/CAM software. With the graphics capability of the CAD/CAM software, visualization of product, editing of design, and replotting of paper format engineering drawings, all became easier. However, not all product designs necessarily begin by producing an engineering drawing. Some products begin by the construction of a prototype model, without any drawings. Other products begin by modifying an existing product whose engineering drawing may only be in a paper format. In these cases, the design of the new product most economically begins with a physical prototype or a paper engineering drawing. To start the design process here requires a mechanism to scan a paper engineering drawing and convert it to CAD/CAM formation (Dori 1995a), or appropriately image a product prototype and convert the image data set to CAD/CAM format. Both these possibilities are not technically simple, each having a variety of intermediate steps, some of which have more than one variation.

In this paper, we discuss the full structure of the required transformations and the role of machine vision in enabling both representations and transformations. More specifically, we aim at establishing a comprehensive view of the various representations of a product, their extent of equivalence, the feasible transformations among them, and the processes that carry out these transformations. In particular, we emphasize non-traditional, vision-based representations that enrich the designer's modelling armamentarium

Authors: Dov Dori, Technion, Israel Institute of Technology, Haifa 32000, Israel.

0951-192X/96 $12·00

and provide for closing currently open loops of representation transformations. It is within this framework that we introduce the terms *documentation* and *inspection*.

In its broad sense, documentation within CIM is an activity that encompasses the entire life-cycle of the product, from market needs and customer specification through the various design stages, to manufacturing, usage and maintenance instructions. Since our focus is product representation, we limit the notion of documentation to capturing the geometry of the model of the product to be manufactured. This may be done by a CAD system, a set of engineering drawings, or a physical model.

In the same spirit, inspection in CIM has generally a broad sense. It is the basis for many quality maintenance and quality assurance activities, carried out throughout the product's design and manufacturing processes. Inspection in this work relates to vision-based activities that monitor and check the adherence of the manufacturing process to the geometry defined by the model.

2. The object–process methodology: merging process- and object-oriented approaches

Harel (1987) has proposed to classify systems into two types: reactive systems and transformational systems. Reactive systems, which include embedded, concurrent and real-time systems, are event-driven and are expected to continuously react to a flow of stimuli. Transformational systems, such as many types of data processing systems, require, in essence, the specification (which may be highly complex) of the system's input/output relations (Harel 1992). The system we are dealing with has reactive and transformational characteristics.

Several system analysis methodologies have been in use in the last two decades. Perhaps the two most notable are the process-oriented, data flow diagram (DFD) approach and the more recent object-oriented analysis (OOA) approach. This section first summarizes briefly the principles underlying each methodology. We then explain the object–process methodology (OPM), which combines elements and ideas from these two seemingly mutually exclusive approaches, and show how documentation and inspection within CIM systems can be clearly, coherently and succinctly modelled using this methodology.

2.1. *The data flow diagram approach*

A customary classification of systems development methodologies is into two categories (Jacobson *et al.* 1992): function/data and object-oriented. Function/data methodologies, which feature distinction between functions and the data these functions manipulate, were inspired by traditional third-generation programming languages, which, in turn, were influenced by the Von Neumann's hardware architecture. Function/data methodologies can be represented by methods like the data flow diagram (DFD) method (De Marco 1978), which emphasizes processes as the major theme of the analysis. In a nutshell, the DFD approach is process- and data-oriented, emphasizing the processes that occur within a system (depicted as bubbles) and the data (depicted as directed arrows) that flow as output from one process to become the input of the next process. The internal details of each process can be exposed by 'explosion', i.e. recursively displaying lower-level DFDs of a particular process (bubble) while preserving the incoming and outgoing data-flows of the process. Two less central elements in addition to 'process' and 'data' in this model are 'data-store', to store data for future use, and 'external entity', to interact with the processes as agents that are considered part of the system's environment.

Figure 1 is a DFD of a typical product life-cycle within an industrial organization. Since it includes data as well as actual objects, it is also referred to as 'action DFD' (ADFD) (Shlaer and Mellor 1988).

Processes and flows (of data and objects) are denoted by labelled nodes (bubbles) and directed arcs, respectively. The whole diagram can be viewed as a description of the process called 'Production'. In essence, the DFD can be viewed as a *processing dependency graph*, that shows what process, or set of parallel processes, is a pre-condition to the initiation of another process. On a higher level, the whole DFD is described in a single 'Production' bubble in what is known as a 'content diagram' or 'zero-level diagram'.

Implicit in the DFD approach is the assumption that

> We all know who the players in the game (i.e., the components of the system under consideration, or objects) are, and we also understand the structural relationships (in particular the three basic ones—whole-part, generalization-specialization, and characterization) among them. Let's get right to the point of analyzing how these components interact with each other dynamically in terms of the data they need, the way it is processed and the data that is produced as a result of this processing.

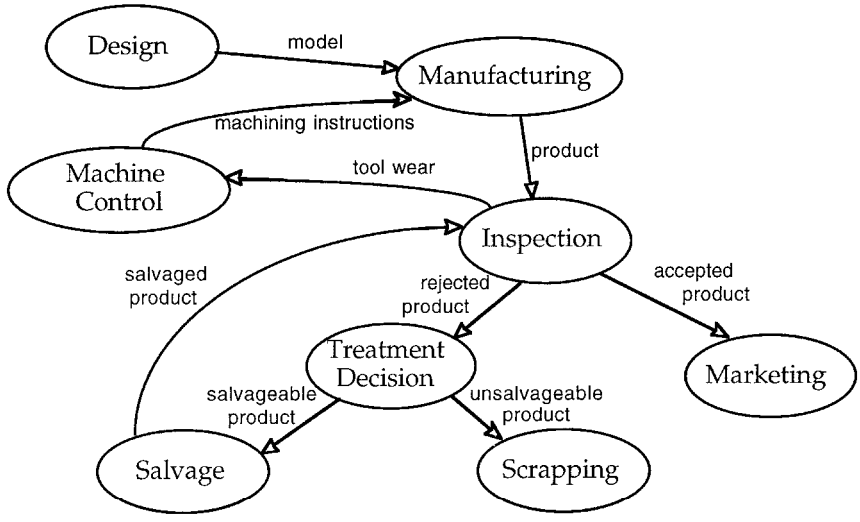

Figure 1. DFD of a typical production life-cycle within an industrial organization.

Since its introduction, the DFD method has gained considerable popularity as a means for describing system functionality and it has even been adopted as a part of several object-oriented analysis methodologies (Rumbaugh *et al.* 1991, Shlaer and Mellor 1985). Other function/data approaches are requirement-driven design (RDD, Alford 1985) and structured *analysis/structured design* (*SA/SD*), or Structured Analysis and Design Technique (SADT), which uses DFD as its main tool.

The DFD method has been very popular in systems analysis. This was fine as long as the systems analysed were of small to medium size. In recent years, however, the object-oriented (OO) approach has won more popularity and is currently an accepted method for information systems development (analysis, design and implementation) as well as a basis for new OO database architectures. The major reason for this trend is that during the last decade, as systems that needed analysis and design gradually increased in size and complexity, it has become more and more apparent that the above implicit assumption is not valid any more. It is no longer the case that a system's constituents are *a priori* recognized, nor are the structural relationships among them easily understood. This has caused a shift of emphasis in systems analysis from data and processes to objects and their relations with respect to each other, and was a primary cause of the trend to abandon the data flow approach in favour of the object paradigm.

2.2. *The object-oriented approach*

Contrasted with the function/data approach, ERDs and the currently accepted object-oriented paradigm (Booch 1994, Coad and Yourdon 1991, Embley *et al.* 1992, Rumbaugh *et al.* 1991, Shlaer and Mellor 1988), put objects at the centre of the analysis. Object-oriented analysis (OOA) approaches are based on the premise that every thing can be represented as an object. The object paradigm combines ('encapsulates') behaviour and data and regards them as integrated objects (Jacobson *et al.* 1992).

The object paradigm is more fundamental than the data flow approach in that rather than considering directly concepts that are solution- and implementation-oriented, taken from computer professionals' jargon, such as 'data', 'data stores', and 'processes', it addresses first the problem of *understanding* the system in terms of the problem domain and within the framework of that domain. In doing so, it takes advantage of human cognitive conventions, in particular aggregation–particularization (whole–part) and generalization–specialization, which, in turn, induce encapsulation and inheritance, respectively.

The basic premise of the object-oriented methodology is that every thing can be considered an object. This includes tangible and abstract objects, activities, operations, and events. All of these get the same reference and same treatment, because they are all 'objects'. Objects have attributes and interact with each other by invoking services (also referred to as methods)—specific behaviour that they can exhibit. The collection of all objects with the same set of features (attributes and services) is a class.

Figure 2 is an enhanced object-oriented analysis scheme of a typical manufacturing environment. It is based on a combination of notations suggested by

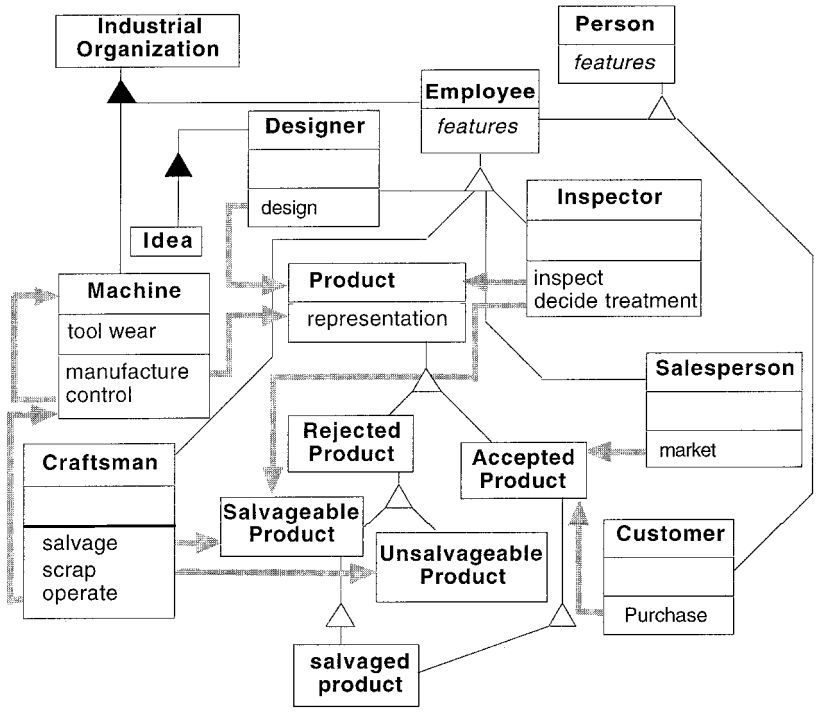

Figure 2. An enhanced object-oriented diagram of a typical industrial organization.

Embley *et al.* (1992) and Caod and Yourdon (1991) with additional features. Each class is denoted by a box which may have one, two, or three compartments, containing the name, attributes and services of the class, respectively. Thus, for example, the class Machine in Figure 2 has the attribute 'tool wear' and the services 'manufacture' and 'control'.

Aggregation and generalization relationships among classes are represented by solid (black) and blank (white) triangles, respectively, whose tips are directed towards the source (aggregate or general) class. For example, in Figure 2, 'Employee' specializes into one of the four classes 'Designer', 'Craftsman', 'Inspector', and 'Salesperson', and is itself a specialization of 'Person', which, in turn, also specializes into 'Customer'.

Message connections show the action of services on classes. Marked by a directed grey link that leaves a specific service and arrives at a class, a message connection shows the pair of sending and receiving classes. Product, for example, receives three services: 'design', 'manufacture', and 'inspect', sent from Designer, Machine, and Inspector, respectively.

Along its life cycle, an object may be in different *states*. One or more of an object's attributes may be a state attribute, whose value designates the specific state the object is currently in. The transition among states occurs after a trigger has been activated, and may be accompanied by an action. Product, for example, may have an attribute called *status*, with possible values 'design', 'prototype', 'production', etc. Transitions that include triggers and actions specify the legal moves from one state to the other. The trigger for the transition from a prototype status to a production status may be 'design requirements met', and the action, 'prepare for mass production'.

Perhaps the crux of the difference between OOA and DFD lies in the exchange of roles of processes and objects. Whereas DFD heralds processes and data as the central basic building blocks, served by data stores and external entities, OOA recognizes the object as the basic building block in the analysis. The 'external entities', which were considered almost a nuisance in DFD, now get top priority as the things that must be considered first and foremost, while processes, which were the first things to be considered in DFD, now termed 'services' or 'methods', are pushed down towards the bottom of the list of things to be considered, both mentally and graphically. DFD is *procedural*, while OOA is *declarative*: the emphasis in DFD is *how*, while in OOA it is *what*. The trend to move from procedural to declarative methodologies is analogous to the parallel development which has occurred in the domain of programming languages. Declarative languages, including OO programming languages (OOPLs) have been developed with the intention to replace third-generation, procedural programming languages. The underlying assumption

has been that very high-level (fourth-generation) languages should be capable of figuring out on their own how to carry out the tasks required by the declarations in the program.

Dealing primarily with objects as the basic building blocks of the universe is a move in the right direction, but a complete description of a system must strike a balance between the structural and procedural aspects of the system. It cannot afford to stress one aspect while neglecting the other. Unfortunately, however, this is exactly what happened to object-oriented systems analysis: emphasizing the structural aspects of a system through its comprising objects is too often done as the expense of suppressing the dynamic aspects of the system.

Just as it turned out that pure declarative languages (such as Prolog) are generally not the most adequate solution for constructing complex systems, and that some balance is needed between the structural (static, declarative) and procedural (dynamic) aspects, so did we discover that the object-centred methodology has gone too far in stressing the importance of objects in systems analysis, and that processes should regain their explicit visibility in the system analysis.

2.3. *Fusing DFD and OOA: the object–process methodology*

The object–process methodology (OPM), introduced in Dori (1995b) has been designed with the aim of describing complex systems using a single model which maintains the balance between system structure and behaviour and enables bidirectional seamless scaling. Being a general methodology, OPM is useful in many areas. Indeed, it has recently been employed in a variety of domains, including technical documentation automation (Dori 1994), design of a temporal database with data dependencies for CIM (Dori *et al.* 1995), analysis and design of a hypertext-based studyware (Dori and Dori 1995), and analysis and representation of the image understanding environment (Dori 1995d). A comparative evaluation of the effectiveness of bridging the analysis–design and structure–behaviour 'Grand Canyons' with object paradigms in general and with OPM in particular appears in Dori and Goodman (1996).

When we set out to tackle the product representations problem, OPM was not yet known. Being aware of the fact that we are faced with a complex, multi-faceted problem, we tried to extract our knowledge of the various aspects of the problem using semantic nets (Winston 1992), which are about equivalent to 'concept maps' (Wandersee 1990) in science education. This approach to knowledge representation within artificial intelligence advocates the representation of concepts and the relations among them as a directed graph whose nodes and arcs are concepts and relators (prepositions and/or verbs), respectively. Viewed from an OOA perspective, concepts and relators can be thought of as classes and their services, respectively.

Soon, though, we found that semantic nets do not provide enough expressive power to reflect the intricate relationships among the entities involved in the problem. We then decided to try to adopt the OOSA methodologies of Embley *et al.* (1992) and of Coad and Yourdon (1991) for our purpose. We tried to describe the various aspects of the system using these methods. Even though we managed to formulate a fair amount of our knowledge of the subject, we still felt that from the behavioural point of view, the picture these methods are able to convey is far from being satisfactory. While the structural–structural relationships (aggregation and generalization) among the various objects in the system were presented clearly, the dynamics were poorly represented. This is so because, as argued, OOA is basically declarative, a feature that underscores the structural, static aspects, but at the same time suppresses the procedural, dynamic aspects of the system under consideration.

The inferiority of services in OOA is a consequence of their inherent attachment to some class, which prohibits them from existing as independent things, or entities. This binding of each service to a particular class is frequently unnatural, since events are often enabled by the concurrent presence and/or action of more than one object class. For example, looking in Figure 2, we see that the services of the Salesperson and Customer classes are *market* and *purchase*, respectively, both applied to the class Accepted Product. However, there is no explicit class 'Purchasing' or 'Marketing', which is the actual operation, or process, that captures the nature of this activity. We could have described the relationships 'Salesperson markets Accepted Product to Customer and receives Currency from Customer' and/or 'Customer purchases Accepted Product from Salesperson and accepts Currency from Customer' with a diamond connected to each one of the four classes involved and these sentences written next to it, as suggested in Embley *et al.* (1992). However, we found this notation to be cumbersome and unsatisfactory, as we do not really know how to automate the processing of such relatively complex sentences in natural language.

Examining thoroughly the things that play role in our system, we observed that they can be clearly divided into two distinct groups: things that exist — the objects — and things that act on them — the processes. Since we

are dealing with transformations (of product representations), it is quite common for an object of one class to lose its identity and be transformed into an object of another class. For example, an object of the class 'Raw Material' in the presence of the agent 'Craftsman' and the instruments 'Machine' and 'Model', undergoes the process 'Manufacturing' and becomes 'Product'. Other processes, such as 'Inspection', are not so drastic as to cause a change of identity of the object Product. Rather, their effect is to transform it from one of its states to another state. Suppose the object class Product has the following states: manufactured, accepted, rejected, salvageable, unsalvageable, salvaged, marketed. Inspection is the process that transforms Product from the state 'manufactured' to one of the mutually exclusive states 'accepted' or 'rejected', while Marketing is the process that transforms a Product from the state 'accepted' to the state 'marketed'. Admittedly, in some borderline cases, the decision as to whether a process destroys an object and generates a new object or whether an object merely changes its state may be arbitrary and subjective.

We wanted to have all the benefits of capturing the structural relationships that OOA provides without suppressing these important dynamic aspects of the system. This has led us to search for a representation in which the structural and the dynamic aspects of the system will co-exist in harmony without the 'shadow effect' of highlighting one at the cost of suppressing the other.

2.4. *Basic object–process methodology concepts*

The basic observation underlying the object–process methodology is that the world (universe of interest) consists of interacting *things* (or entities). Any thing is classified as either an *object* or a *process*.

An object is born as a result of a process called *construction* or *generation*. After being constructed, the object exists unconditionally, until it is deliberately destructed by another process, called *destruction*.

A *process* is a transient thing in the world that changes at least one object.

During the life-cycle of the object, it undergoes changes that are caused by processes. Construction and destruction are among the changes a process makes to an object. To avoid circular definitions, the definition of object does not include process. Nevertheless, there is a reciprocal link between objects and processes.

A process has a definite start, lifetime duration, and hence an end. The duration of a process is generally a random variable, which is of the same order of magnitude as the time granularity (Dori 1995b) of the corresponding world. In contrast, the lifetime duration of an object is normally at least an order of magnitude longer than that of a typical process.

A change of an object is a change of the object's *state*, which is the set of attribute values describing the object at a particular point in time. Each object has by default the attribute *existence*, whose values include *existent* and *non-existent*. The effect of the construction process is to change the value of the existence attribute from non-existence to existent, while the effect of destruction is the opposite.

The customary OOA definition of class is that a class is a collection of objects having the same set of attributes and services. In OPM, a class is defined as a

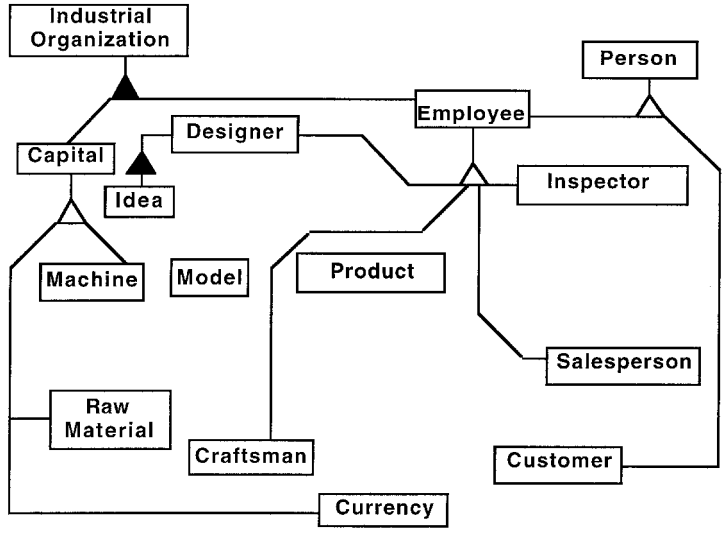

Figure 3. A partial OPD showing the main object classes in a typical industrial organization.

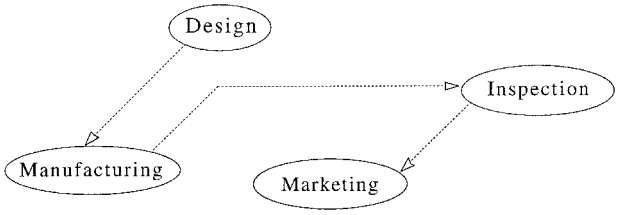

Figure 4. A partial OPD showing the main processes in a typical industrial organization.

collection of *things* (entities) having the same set of features (attributes and services). Since a thing may be an object or a process, there are two class types: *Object class* and *process class*.

Examples of object classes are 'Machine', 'Product' and 'Inspector', each of which may exist unconditionally, independently of any other thing. Figure 3 is part of an object–process diagram (OPD), which shows only the main object classes in a typical classical industrial organization: Employee, Capital, Product, etc., without showing the processes through which these objects interact.

Figure 4 is a complimentary partial OPD, whose purpose is to demonstrate how processes in a system can be described separately, much like DFDs. This picture is only one of the two major aspects of the system, the other being the objects and their structural relations. It shows the main process classes in the same typical classical industrial organization: Design, Manufacturing, Inspection and Marketing. A directed dashed arc, the *process link*, designates the order of occurrence of these processes. The term 'classical' here pertains to the fact that inspection is done at the end of the manufacturing process, while modern CIM approaches, as well as quality assurance standards, advocate on-line, real-time monitoring to ensure built-in quality.

From a viewpoint of a particular process class, object classes may be classified into two sub-types: enabling object classes and resulting object classes. An *enabling object* of a process is an object that enables the occurrence of that process. A *resulting object* of a process is an object that results from (or is the output of) the occurrence of that process.

Note that an object class is classified as either enabling or resulting only with respect to a particular process class. For example, Product, which is a resulting object of Manufacturing, is an enabling object of Inspection.

Enabling classes are further classified into three types according to their function in the process: *affected* (or input) classes, *agent* classes, and *instrument* classes. For example, the agent class of the process Manufacturing is Craftsman, its instrument class is

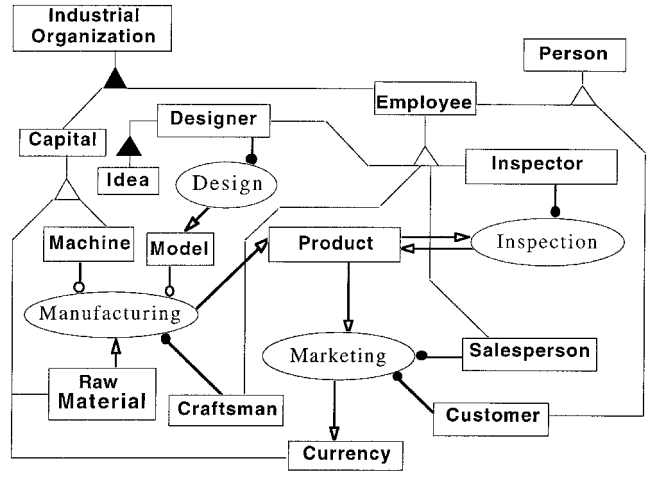

Figure 5. A object–process diagram (OPD) of a typical industrial organization.

Machine and its affected class is Raw Material. The affected class is the only one whose state changes. The resulting (output) class is either the affected (input) class (whose state has been changed), or a newly constructed object class.

When we merge the two partial OPDs of Figure 3 and Figure 4, we get, in Figure 5, a complete, top view OPD of an industrial organization.

Combining objects and processes within one diagram, shown in Figure 5, results in a comprehensive view that shows very clearly both the structural and procedural aspects of the system and the relations between them. Merging object classes with process classes within the OPD enables us to specify for each process the object classes that enable it and the object classes that result from it. These *procedural links* are denoted in the OPD as solid directed arcs. For an affected object coming into or going out of a process such a solid arc is called an *effect link* and it ends with an arrowhead. For an incoming agent class, the procedural link is called an *agent link* and it ends with a solid (black) circle, and for an incoming instrument class, the procedural link is called an *instrument link* and it ends with a blank (white) circle.

2.5. *A meta-description of the object–process methodology*

In this section we describe the object–process approach using object–process diagrams embedded with examples.

Figure 6 is an OPD showing the structural elements that take part in an OPD with an example showing the symbol used for each element. An OPD consists of

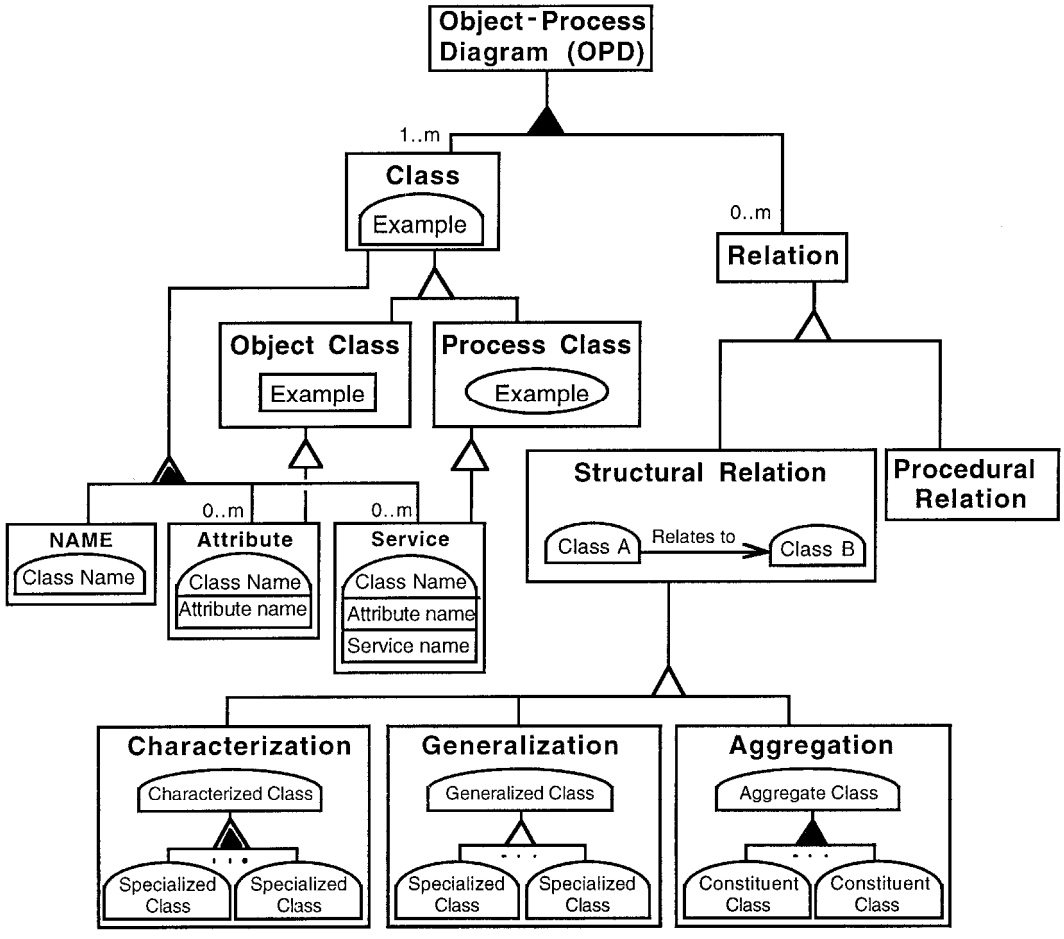

Figure 6. OPD–structural elements.

classes and relations. A class, denoted as half ellipse half rectangle, is a generalization of Process Class (ellipse) and Object Class (rectangle).

As Figure 6 shows, Attribute and Service (generically termed features), characterize the class to which they are connected via the characterization relation, symbolized by black-on-white triangle. Attribute is a specialization of Object Class, while service is a specialization of Process Class. Each value of an attribute is an instance of the corresponding Object Class. As an example, consider the Object Class Drilling Machine, which has the Attribute (characterizing object class) Control-Mode, which, in turn, has three values (instances): 'manual' 'NC' and 'CNC', and a second Attribute, Maximum-Hole-Depth, with the value '50 mm'. Both Control-Mode and Maximum-Hole-Depth are Object Classes. Drilling Machine also has the Services Drilling and Boring, both of which are Process Classes. The Process Class Drilling may have the Attributes Maximum-Drilling-Speed and Maximum-Drilling-Diameter, both of which are Object Classes, and the Services Change-Tool and Exit-Drilled-Hole, both of which are Process Classes.

As shown in Figure 6, in addition to its solid triangle symbol, Aggregation also has one *participation constraint* for each constituent class. The default is 1. Thus, since Class has one name, no participation constraint is written next to the object name. Class has zero or more Attributes, and zero or more Services. The name of the Class is written in the upper compartment, which may be the only one, if the OPD is at a high-level, where just the name is important. The attributes and services of the class are written in the second and third compartments below the name compartment. If the services do not exist or are not important for a particular view the third compartment may be omitted. Thus, a class symbol may contain one, two, or three compartments, as needed. Alternatively, the features may be recorded using the characterization symbol.

Relation is a generalization of Structural Relation and Procedural Relation. Structural Relation in general is denoted by a labelled line directed with an

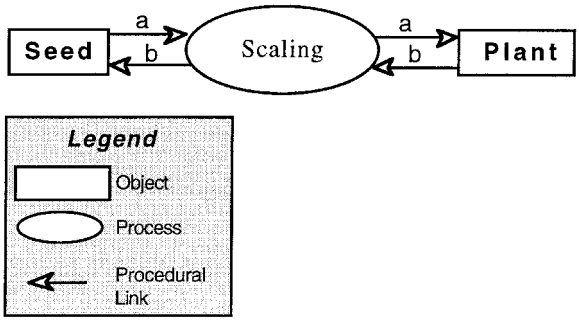

Figure 7. Scaling converts a seed to a plant and vice versa.

open arrowhead. For example, the Structural Relation between the classes CAD System and CAD Drawing in Figure 12 is 'stores'.

Characterization, Generalization and Aggregation are three specializations of Structural Relation, which, due to their frequent usage, are granted special symbols, as shown in the three small example OPDs inside the Characterization, Generalization and Aggregation boxes. These relations are denoted by an empty and a filled triangle, respectively. The tip of each triangle type is directed towards the higher-level class (characterized, generalized and aggregate) and its base, towards the lower-level (characterizing, specialized and part) class(es).

2.6. *Scaling*

Scaling is the mechanism through which OPM manages the complexity of real life systems. The IDEF family (Jorysz and Vernadat 1990a,b) and SADT (Structured Analysis and Design Technique (Yourden 1989)) also have scaling mechanisms. However, there are three main differences between the scaling of OPM and the other two. First, OPM treats objects and processes and their scaling equally, and enables scaling of objects and/or processes concurrently. In contrast SADT, and DFD in particular, for example, enable scaling of processes only.

Second, both IDEF and SADT scaling mechanisms are pure top-down, i.e. at each diagram only a single level can be represented, while in OPM it is possible to focus on a particular subset of things (objects and/or processes) and elaborate on the details of each one by scaling up each thing to any desired number of levels. Thus, it provides for selective refinement of classes in the OPD to any desired level of granularity, while the general picture is maintained by keeping other classes at their non-detailed representation. This is instrumental for generating different views of the system for different types of people or groups which are interested in or are supposed to develop and be responsible for different aspects of the system.

Third, scaling in IDEF and SADT means exposing the parts of the whole, i.e. they refer to specifying the details of the aggregate. While OPM does so too, it is but one of three scaling options, the other two being specialization scaling and characterization scaling. Specialization scaling means that instead of specifying the parts of the whole, we specify the specializations of the generalized thing, while in characterization scaling, the features (attributes and services) are detailed. Any concurrent mixture of the three scaling types is possible within a single diagram. The unfolding scaling option, described below, is most suitable for the latter two scaling types.

Seed is the thing (object or process) that is scaled up.

Plant is the set of things (objects and/or processes) that result in from up-scaling a seed.

Figure 7 is the OPD showing the effect of scaling. When a Seed undergoes scaling the result is a Plant. When a Plant undergoes Scaling the result is 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. Scaling is characterized by two attributes: Direction and Seed Preservation. Direction has the two values Up and Down. Up-scaling (expansion) means increasing the level of detail and down-scaling (collapse), decreasing the level of detail. Seed Preservation pertains to the graphic result of scaling up the seed. It has three values: Negative, Background and Root, giving rise to three up-scaling types: Explosion, Blow-up and Unfolding. The three corresponding down-scaling types are Implosion. Shrinking and Folding.

Explosion is an up-scaling 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.

Explosion and implosion are the pair of operations DFDs employ to carry out scaling. Figure 8 demonstrates 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.

Blow-up is an explosion in which the seed is preserved in the background of the plant.

Graphically, the preservation of the seed in the

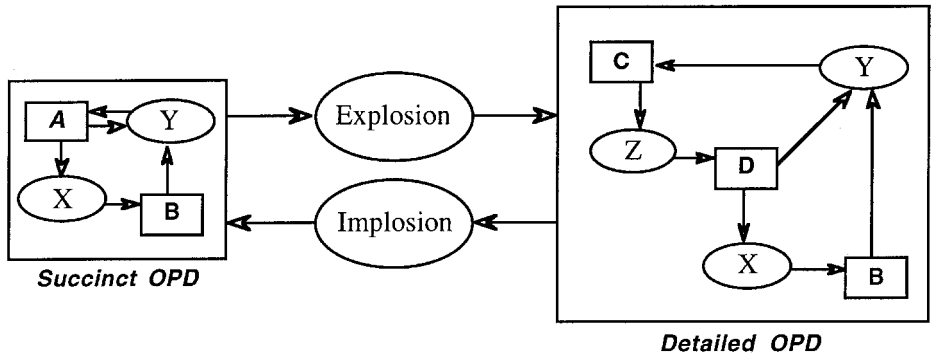

Figure 8. Explosion of a seed—object A—in the succinct OPD to a plant—C, D, and Z—in the detailed OPD and implosion in the reverse direction.

plant's background is denoted by a thick, grey *blow-up frame*, surrounding the seed's constituents. If the seed is an object, the blow-up frame is rectangular, as in Figure 9, and if the seed is a process, it is elliptic.

Shrinking is the inverse of blow-up. Like implosion, shrinking is a down-scaling of a set of things surrounded by a blow-up frame, which are the constituents of the seed. As a result of a shrinking process, the seed shows up instead of its constituents and the blow-up frame reverts to the original thin, solid frame.

Figure 9 demonstrates blow-up of the seed A in the succinct OPD to a plant in the detailed OPD. As a result of the blow-up, object A, which is the seed, remains in the plant as the blow-up frame in the background. The constituents of A—C, D, and Z—are drawn with their respective procedural relations. Like implosion with respect to explosion, shrinking is the reverse of blow-up.

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 10 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. Z is a process which is internal to A, as it processes C and results in D. As such, it is referred to as a service.

3. Machine vision and computer integrated manufacturing

In recent years, computer integrated manufacturing has evolved from a remote target to a tangible system, in which manufacturing is governed to a great extent by a system of coordinated computers that keeps track of the various stages in the manufacturing process and is capable of reacting to a host of occurrences throughout the process. Concurrently, machine vision has also matured as an important discipline within computer science, involving a host of fields ranging from pixel-level image processing to artificial intelligence-based semantic constructs.

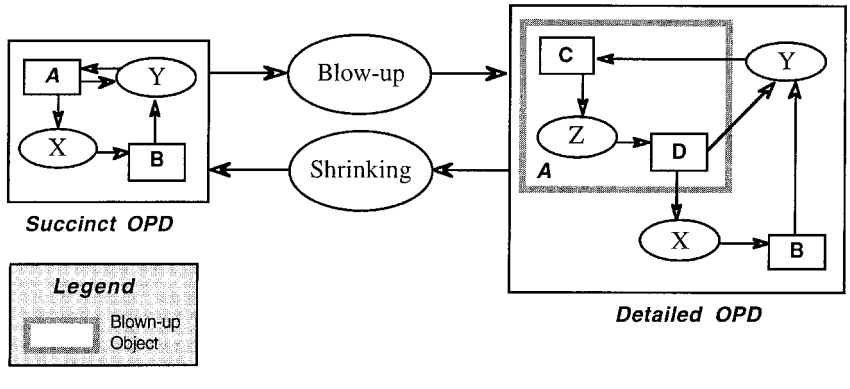

Figure 9. Object A is the succinct OPD is blown up to become a plant in the detailed OPD and the plant is shrunk back.

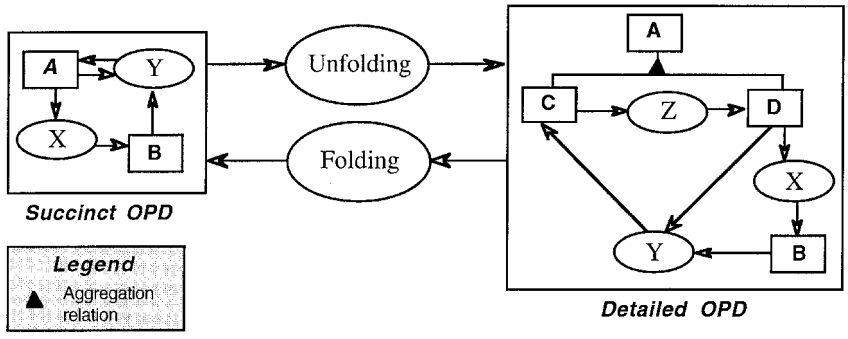

Figure 10. Unfolding a succinct OPD to a detailed OPD and folding it back.

The original motivation for the development of the object–process model was to establish an integrated approach of handling the host of product representations, their identical and different information content, the degree of uncertainty that each one of them conveys, the possible pathways among the representations, and the processing that is needed for each of these transformations to take place.

There are two elements in the automated manufacturing process that are not yet fully integrated into CIM systems: inspection and documentation. It is not accidental that these two elements involve machine vision, because, as argued, these two areas have matured in parallel.

Incorporating machine vision technology into the CIM system greatly enhances its capabilities in a variety of ways. Object–process analysis (OPA) of the CIM system has revealed two major loops of interleaved object and process classes: the design–modelling–documentation loop and the manufacturing–inspection–control loop. In the rest of this section we apply OPA to study these two loops.

3.1. *The design–modelling–documentation loop*

Figure 11 unfolds the process class Design and the object class Model that appear in Figure 5. Requirements are affected class and Designer (who uses his Ideas) is the agent in the Design process. Design may be manual Design, Computer Aided Design, in which case a CAD system is the instrument, or Physical Design. The resulting Model of these three design options is Engineering drawing, CAD CSG (Constructive Solid Geometry) or B-rep (Boundary representation), and Physical Model.

Figure 12 shows the documentation module of the system. Unfolding Engineering Drawing exposes three drawing specializations: CAD Drawing, Paper Drawing, and Raster Drawing (Dori 1994). CAD Drawing, resulting from CAD and stored by CAD System, can undergo plotting, which is enabled by the instrument class Plotter and results Paper Drawing. Paper Drawing may also be the result of Manual Design.

Electronic Archive consists of a capture device—usually Scanner, and secondary storage, such as a

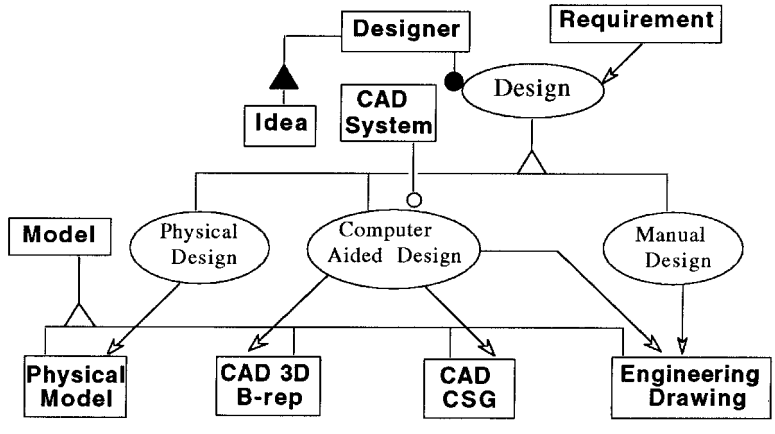

Figure 11. Unfolding the process Design and the object Model from Figure 5.

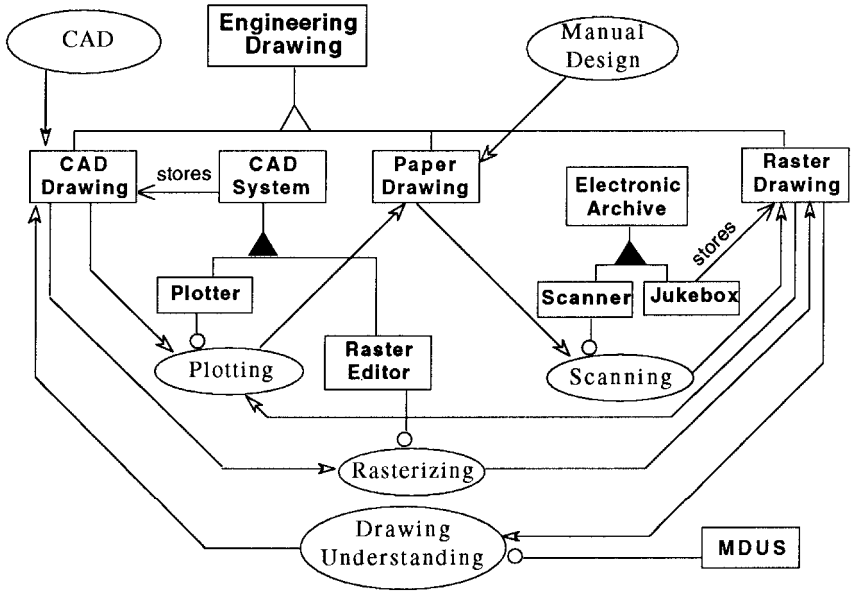

Figure 12. The documentation module.

magnetic or optic disc Jukebox. It handles Raster Drawings, which are digital images of drawings. Paper Drawings (originating from either Manual Design or from CAD Plotting) undergo Scanning (digitization) by a Scanner and are stored in the Jukebox. A direct conversion of CAD Drawing to Raster Drawing is enabled by Raster Editor, which does Rasterizing.

All the components of the documentation module discussed so far are operational technologically and algorithmically. They enable one-way transformations in the sequence CAD Drawing–Paper Drawing–Raster Drawing. Closing this open loop requires a sub-system that is capable of converting Raster Drawings into CAD Drawings. The Machine Drawing Understanding System (MDUS), which is an active subject of research (Dori 1989, 1992, 1995a, Dori *et al.* 1993) is aimed at converting mechanical engineering drawings into CAD drawings.

3.2. *The manufacturing–inspection–control loops*

As can be seen in Figure 13, Inspection is applied to Product, which is compared to Model, and classifies Product into Accepted Product and Rejected Product. The dashed line marks a logical exclusive OR relation between the two Inspection outcomes. A Rejected Product is reworked and the Resulting Product closes the loop. Another loop in which inspection is involved is that yielding Tool-War Estimate, which is fed back into the Machine Control process to compensate for deviation from the specification of the model during the Manufacturing process.

In Figure 14, Inspection is blown up, exposing Sensing and Acceptance Decision as the two lower-level processes that comprise Inspection. Sensing is applied by Sensing Equipment to Product to obtain Measurements and Tool Wear Estimates. Measurements are used for Acceptance Decision and Tool Wear Estimates for Machine Control.

In Figure 15, Figure 14 is further scaled up. Sensing is classified into Proximity Sensing and Remote Sensing. The traditional Proximity Sensing uses Gauges to obtain Proximity measurements of the product under inspection. Remote Sensing requires the instrument Remote Sensing Equipment to produce

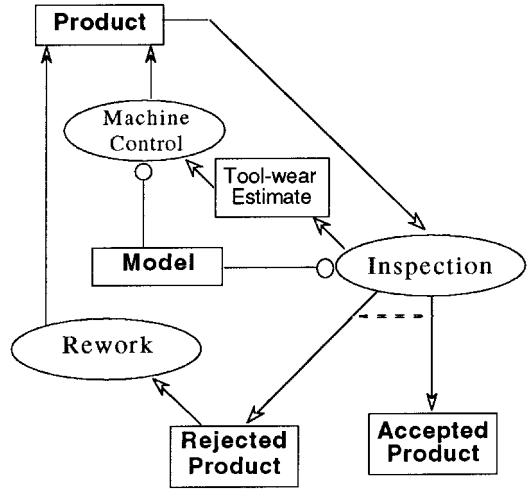

Figure 13. The two manufacturing–inspection–control loops.

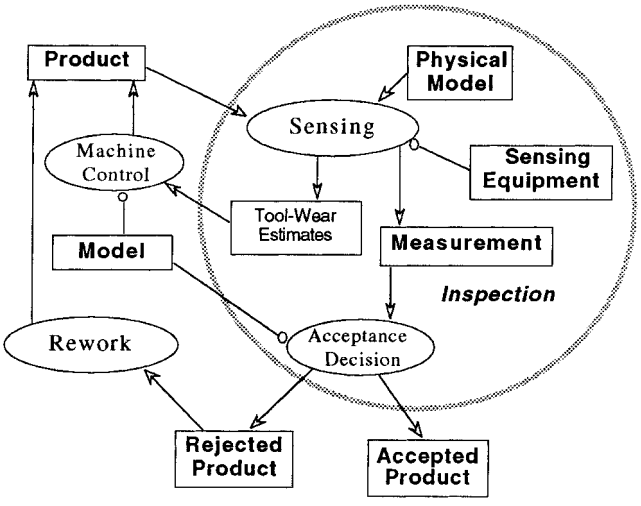

Figure 14. The process Inspection of Figure 13 blown up.

Image, which, in turn, is the input to the Machine Vision process class. Machine Vision results in Sensed Measurements, which, along with Proximity Measurements, are input to Tool Wear Estimation. This Estimation establishes Tool Wear Estimate, which, along with Model and Craftsman, in input to Machine Control.

By inspection of the manufactured products with machine and robot vision, where information from both visual and/or range images are fused to obtain the 3-dimensional structure of the product, we allow for the reconstruction of the analytical model with a certain degree of uncertainty. This, in turn, enables a non-traditional reverse engineering fabrication technique. Monitoring the manufacturing process with proximity and remote sensing techniques enables on-line feedback to the manufacturing instruction controller to compensate for tool wear.

Understanding engineering drawings closes the currently open model–drawing loop. This loop consists of the constructive solid geometry model, its respective 3D Boundary representation (B-rep), the set of 2D B-reps obtained by its orthographic projection, the CAD engineering drawing obtained by dimensioning and tolerancing placement, the paper engineering drawing obtained by plotting, and the electronic drawing obtained by scanning the paper drawing or by rasterizing the CAD drawing.

4. Discussion and summary

We have introduced a new paradigm in systems analysis—object-process methodology (OPM)—which

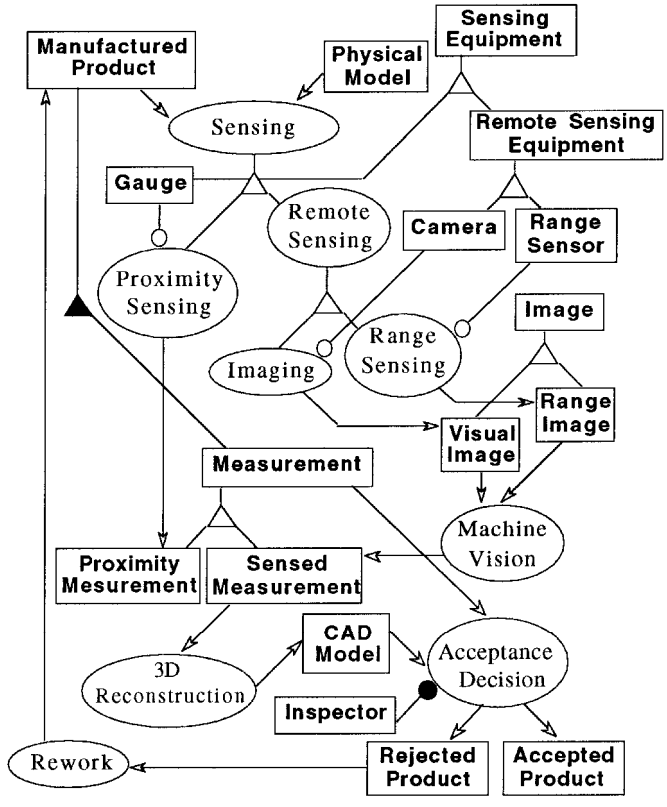

Figure 15. Figure 14 scaled up.

is a fusion of two accepted approaches in information systems analysis: the object-oriented approach and the data-flow approach. Although these two approaches have been considered to be mutually exclusive, OPM not only manages to incorporate useful ideas from both approaches, but also gracefully fuses static–structural aspects borrowed from OOA with dynamic–procedural elements from the DFD method. An object–process diagram (OPD) clearly shows how object classes relate to each other, what process classes operate on them, and how one object class is transformed into another. The principle of alternating object and process classes in the flow of activity is very powerful in providing insight into the analysed system.

OPM has been implemented in this work in the analysis of Computer Integrated Manufacturing, Documentation and Inspection. We have seen that incorporating Documentation and Inspection elements as an integral part of a CIM system greatly enhances the system's integrity and capabilities.

In the documentation domain, understanding engineering drawings is the missing link needed to close the currently open loop from paper and electronic drawings back to CAD drawings. Using OPD, we have shown that, much like a compiler, such understanding can be done in three analysis phases: lexical, syntactic and semantic. The resulting set of projections can be used as input for a fleshing-out projections algorithm to obtain the CAD drawing representing the CAD model. The Machine Drawing Understanding System is currently capable of performing the lexical phase and some of the syntactic phase.

In the inspection domain, machine vision is already applied in many industries to enhance or replace proximity sensing in the measurement of parts for inspection purposes. Less developed is the application of machine vision to real-time control over the manufacturing process and still less developed is its use for reverse engineering. Here there are several unsolved theoretical problems.

From the way we managed to apply OPM to analyse such a complex system as Computer Integrated Manufacture Documentation and Inspection (CIMDI), OPA seems to be a very effective methodology of systems analysis. Several of the ideals pertaining to the CIMDI system are a direct consequence of its development, while a number of OPM rules were inspired by real-life problems posed by CIMDI. Thinking concurrently about the two problems has been a great challenge, which proved fruitful for both OPM methodology and the CIMDI system. We envision that OPM will become an accepted method for systems analysis and that CIMDI will become a reality in the near future.

Acknowledgements

The author wishes to thank the two anonymous referees for their helpful comments. This work was supported by the Technion VPR Fund.

References

ALFORD, M., SREM at the age of eight: the distributed computing design system, *IEEE Computer*, **18**, 36–46.

BOOCH, G., *Object-Oriented Analysis and Design*, 2nd edn (Benjamin Cummings, Redwood City, CA).

COAD, P., and YOURDON, E., *Object Oriented Analysis*, 2nd edn (Prentice Hall, Englewood Cliffs, NJ).

DE MARCO, T., *Structured Analysis and System Specification* (Yourdon Press, New York).

DORI, D., 1989, A syntactic/geometric approach to recognition of dimensions in engineering machine drawings. *Computer Vision, Graphics, and Image Processing*, **47**, 271–291.

DORI, D., Dimensioning analysis: a step towards automatic understanding of engineering drawings. *Communications of the ACM*, 92–103.

DORI, D., 1994, Automated understanding of engineering drawings: an object-oriented analysis. *Journal of Object Oriented Programming*, Sept 1994, 35–43.

DORI, D., 1995a, Representing pattern recognition-embedded systems through object-process diagrams: the case of the machine drawing understanding system. *Pattern Recognition Letters*, **16**, 377–384.

DORI, D., 1995b, Object-process analysis: maintaining the balance between system structure and behaviour. *Journal of Logic and Computation*, **5**, 227–249.

DORI, D., 1995c, Arc segmentation in the machine drawing understanding environment. *IEEE Transactions on Pattern Analysis and Machine Intelligence, T-PAMI* (to appear).

DORI, D., 1995d, Analysis and representation of the image understanding environment using the object-process methodology. *Journal of Object Oriented Programming* **17**, **11**, 1057–1068.

DORI, D., and DORI, Y. J., Object-process analysis of a hypertext organic chemistry module. *Journal of Computers in Mathematics and Science Teaching* (to appear).

DORI, D., and GOODMAN, M., 1996, On bridging the analysis-design and structure-behavior grand canyons with object paradigms. *Report on Object Analysis and Design*, **2**, **5**, 25–35.

DORI, D., GAL, A., and ETZION, O., 1995, A temporal database with data dependencies: a key to computer integrated manufacturing. *International Journal of Computer Integrated Manufacturing*, **9**, 89–104.

DORI, D., LIANG, Y., DOWELL, J., and CHAI, I., Sparse pixel recognition of primitives in engineering drawings. *Machine Vision and Applications*, **6**, 69–82.

EMBLEY, D. W., KURTZ, B. D., and WOODFIELD, S. N., *Object-Oriented Systems Analysis* (Prentice Hall, Englewood Cliffs, NJ).

HAREL, D., Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, **8**, 231–274.

HAREL, D., Biting the silver bullet: toward a brighter future for system development. *Computer*, 8–20.

JACOBSON, I., CHRISTERSEN, M., JOHNSON, P., and OVERGAARD, G., *Object Oriented Software Engineering, A User Case Driven Approach* (Eddison Wesley, Reading, MA).

JORYSZ, H. R., and VERNADAT, F. B., 1990a, Cim-osa Part 1: Total enterprise modeling and function view. *Journal of CIM*, **3/4**, 144–156.

JORYSZ, H. R., and VERNADAT, F. B., 1990b, Cim-osa Part 2: Information view. *Journal of CIM*, **3/4**, 157–167.

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EFFY, F., and LORENSEN, W., *Object-Oriented Modelling and Design* (Prentice Hall, Englewood Cliffs, NJ).

SHLAER, S., and MELLOR, S. J., *Object-Oriented System Analysis* (Prentice Hall, Englewood Cliffs, NJ).

WANDERSEE, J. H., Concept mapping and the cartography of cognition. *Journal of Research in Science Teaching*, **27**, 923–936.

WINSTON, P. H., *Artifical Intelligence*, 3rd edn (Addison-Wesley, Reading, MA).

YOURDON, E., *Modern Structured Analysis* (Prentice Hall, Englewood Cliffs, NJ).