INT. J. COMPUTER INTEGRATED MANUFACTURING, 1996, VOL. 9, NO. 2, 89–104

A temporal database with data dependencies: a key to computer integrated manufacturing

DOV DORI, AVIGDOR GAL and OPHER ETZION

Abstract. A temporal database scheme with data dependencies is proposed which supports complex tasks, such as version control and configuration management in computer integrated manufacturing (CIM). This architecture adds functionality to the CIM database, thereby reducing the programming effort required outside the database. The database correctness is maintained even after introducing engineering changes. A case study shows that the proposed scheme also supports automated decision making for shop-floor control.

1. Introduction and motivation

Complex systems are becoming increasingly demanding with respect to the performance of their supporting databases. Computer integrated manufacturing (CIM) systems, which encompasses the entire life-cycle of design, manufacturing and support of products in the required quality, quantity and schedule, should support simultaneous views of objects that vary over time, and should apply data dependencies that affect past or future views of objects. However, current CIM systems usually employ traditional databases, in which much of the system's required functionality is carried out by procedural, third generation languages. For complex applications, procedural programming tends to be tedious, time consuming and difficult to verify (Abiteboul 1988). The gap between the complexity of CIM systems on one hand, and the limited abilities of traditional database models on the other hand, requires a massive translation process from real life to a computerized application when a set of conventional programming tools is used. This translation is often inaccurate and time consuming.

We introduce an architecture for the support of data dependencies and temporal simultaneous views of a CIM database. Advanced database architectures, such as the one introduced in this paper, are aimed at providing high-level abstractions. These abstractions add functionality, reduce the programming effort required outside the database and decrease the number of translation levels between requirements phrased in natural language and the corresponding executable code. The combination of data dependencies and temporal support is a novel approach in the database research area, and to the best of our knowledge it has not been applied to CIM. This approach is useful in particular in version control and configuration management, where past, and possibly future, viewpoints are as equally important as current information. To demonstrate the utilization and the effect of data dependencies and temporal information, we incorporate them as building blocks in a detailed CIM case study and demonstrate their ability to maintain version information and automatic decision support operations based on cost/benefit and other strategic considerations.

1.1. *The role of computers and databases in CIM*

The CIM paradigm asserts that the various portions of the manufacturing process should obtain cooperation among the distinct software products and support high-level automation for each of the manufacturing-related activities (Ranky 1986). Computers play a central role in CIM systems. A network of computers should support the organization by integrating various computer-based activities, including design (information regarding the geometry and materials of the product), manufacturing (the bill-of-materials for each product and the sequence of processes through which that part is made), inventory management (inventory levels of the various raw materials and

Authors: Dov Dori, Avigdor Gal and Opher Etzion, Technion-Israel Institute of Technology, Faculty of Industrial Engineering and Management, Haifa, 32000, Israel.

0951-192X/96 $12·00

parts), marketing (maintaining customer details, including their orders, financial balance and marketing means), and human resources.

The lack of a common language for the different objects in CIM is one of the obstacles on the way to achieving a comprehensive CIM system. Recent work (Etzion *et al.* 1995) used data inter-dependencies to construct a coordinator as a tool for satisfying the cooperation requirement. A *coordinator* is a database that uses a global data dictionary of the data and its organization in different machines to allow transparent access to each data item on any machine. Both the data dictionary and the coordinator are parts of a *coordinating knowledge base*, which contains the knowledge regarding the various underlying machine architectures and the location and access method of each data item within each machine. Database operations are transparently translated by the coordinator to the 'language' of the appropriate machine.

In this paper we use a temporal database and data dependencies to extend the support of CIM applications discussed in (Etzion *et al.* 1995). The database supports the following three features, which are demonstrated in the CIM domain.

(1) Simultaneous values. To trace and audit knowledge that was available in the past, we need to maintain historical knowledge along with the current knowledge. For example, although a product may have new versions, a company should also provide users of older versions with both technical support and spare parts. Hence, simultaneous values of a number of versions need to be maintained. This approach can advance the production of multi-version products, analogous to the advances that have been achieved in areas such as production management (Monden 1983). The use of simultaneous values would facilitate the adoption of a product to the needs of specific customers, rather than using the conventional version method. This approach would yield more flexible production plans, resulting in a major competitive advantage.

(2) Data dependencies. Values of data items in the CIM database may depend on values of other data items. For example, if a product has several representations, altering one of them should automatically trigger the modification of the others, such that consistency is maintained across the different representations. This process is supported by *data dependencies*—meta data elements that are triggered automatically, with no need for user interference.

(3) Retroactive and proactive updates. A retoractive (proactive) update operation is an operation that refers to past (future) time points. Frequently, knowledge that refers to the past is obtained at a later stage. To keep the database intact, data dependencies and knowledge should have the capability of being applied in a retroactive manner. For example, in a CIM database, updating the inventory level of items in a remote warehouse, to which there is no direct, on-line link, must be made retroactively to make the data valid with respect to some time interval which started, and possibly even ended, in the past. A CIM system needs to model knowledge about predicted future events, as well as about past events. This is a natural extension of simultaneous views of various time points in the past, in which we also allow proactive updates and reference to future time points.

The CIM paradigm has been materialized partially by using different technological building blocks that were somehow put together. These building blocks meet the needs of a CIM system at the level of loosely connected 'islands of automation'. Consequently, such integration has no unifying methodology that can 'challenge the old-fashioned ways of management thinking and practice in many manufacturing organizations, and thereafter assist in constructing flexible structures and formulating competitive strategies' (Wang *et al.* 1993). For example, the massive changes that are frequently associated with an engineering change resulted in a common policy that requires the clustering of several minor changes into a single 'version change'. A manufacturer who can reduce the complexity of version management, and offer products that fits customers' special needs, would gain considerable competitive advantage. We show how temporal databases can serve this purpose.

1.2. *Related work*

The lack of a unified approach that spans across the technological building blocks of a CIM system makes their management and integration into an existing organization a highly complex task (Bennett 1987, Hayes and Jaikumar 1988). Some examples of attempts to model CIM systems follow.

SofTech's Structured Analysis and Design Technique (SADT), later extended to the IDEF0, model tasks of the organization as functions transforming input into output via a control mechanism (Meta Software

1990). This system represents inputs, outputs and controllers at a level of graphical labels, with no underlying information structures or database functionalities. The GRAI approach (Doumeingts 1987) comprises tools for identifying decisions and the associated control horizons and frequencies. It examines physical and managerial activities and the information required to model particular decisions.

Each of these systems consider only a single main aspect of the organization requirements (decision making in GRAI, and information structure in IDEF0). Other aspects are either omitted or dealt with partially. Consequently, a variety of tools are needed to satisfy the entire spectrum of organizational requirements. Attempts to develop multi-faceted systems have not managed to cover all the functionalities required for CIM systems. For example, the IDEM system (Wang *et al.* 1993), which is a case-driven tool, pays little attention to data, which is the main constituent of database systems. This is a source of problems, such as incorrect presentation of data items and overly complex representation of certain activities.

CIM-OSA (Jorysz ad Vernadat 1990a,b) is an open-systems architecture which defines an integrated methodology to support all phases of a CIM system lifecycle, from requirements specification, through system design, implementation, and operation to maintenance. CIM-OSA introduces an information modelling framework for CIM, which represents graphically the information content and the structures related to a system, but lacks the use of time-varying properties and data dependencies. Since enterprises are expected to last for long periods of time, support of the temporal aspect is vital in any such model.

Developing a complete product description in a natural format is one of the goals of the PDES/STEP effort (International Organization for Standardization 1991). The standard is subdivided into separate International Standards called Parts, which include geometric and topological representation, materials, tolerances, etc. STEP uses the EXPRESS data definition language as a tool for providing object-oriented, integrated views of product data. While support of the time dimension is not explicitly specified, we argue that for the sake of completeness, a database model must have temporal-supporting facilities. We further argue that due to its genericity and scope, the model proposed in this work may well be suited to fit within the database implementation method, which is an integral part of STEP. The PDES/STEP is still evolving, and the support of simultaneous values and data dependencies can be considered as additional features in future versions of the standard.

The active database research area comprises languages and tools, designed to provide the database with the ability to detect events and activate operations in response. The following points have motivated the establishment of active databases (Chakravarthy 1989):

- support of event-driven operations, especially in time-constrained applications, such as process control, network management and battle control;
- maintenance of derived data and views; and
- consistency enforcement, i.e. enforcement of data interchange and configuration management.

A prominent paradigm in active databases is Event-Condition-Action (ECA), proposed in the HiPaC project (Chakravarthy 1989). According to the ECA paradigm, the basic primitive of an active database is a rule, which consists of the following three components:

- an event: a database operation, an external signal or a recursive composition of other events;
- a condition: a query issued on a database; and
- an action: a user defined program.

The logic of the ECA paradigm is straightforward: whenever an event occurs, evaluate all rules that refer to this event. For each relevant rule, evaluate the condition. If the condition is satisfied, execute the action.

Research in the area of active databases has yielded the following observations regarding the ECA paradigm.

(1) The dependencies among a derived data element and its derivers, as well as the integrity constraints, possess inherent semantic relationships that can neither be captured by ECA rules nor by ECA execution models. The lack of semantic abilities prevents any reasoning capabilities about an application behaviour, eliminates optimization capabilities and does not support rule interaction definitions.
(2) The information an ECA model contains regarding the entire update process is partial. While the *event* and *condition* parts are explicitly represented within the model, the logic of the *action* part is hidden within a user defined program. Since the model does not 'know' about the logic of the action part, it is impossible to reason about the transitive closure of an event, that is, to foresee all the consequences of an update. This hampers the analysis of the application behaviour and its optimization (Bolzer 1992, Hudson and King 1986, Segev and Zaho 1991).
(3) The semantics of the ECA model does not impose an inherently consistent execution

model. Therefore, contradicting rules may coexist, and the relationships and interactions among rules and mutual effect of rules on each other may not always be predictable.

A heterogeneous, active database architecture for engineering data management was proposed by Urban *et al.* (1994). The active element is based on the ECA approach, and thus suffers from its aforementioned shortcomings. Although the model keeps track of design history, it does not present a full temporal model, as we propose in this paper.

A database that supports *data dependencies* is a special case of an active database. Such a database supports rules and can control the flow of data in response to rules' activation. The concept of data dependencies was defined by Hudson and King (1986) and within the PARDES project (Etzion 1993) and its extension TAPUZ (Etzion 1994).

Temporal semantics was dealt with in works on temporal databases, including Clifford and Crocker (1987), Gadia (1988), Navathe and Ahmed (1989), Pissinou *et al.* (1994), Snodgrass (1987), Snodgrass *et al.* (1994) and Su and Chen (1991). Temporal databases provide temporal support at the system level, relieving users from the need to explicitly model the temporal effect. Novice users can use time defaults, instead of specifying the temporal elements that are associated with every data item. At the same time, professional users can model complex temporal relationships using primitive system elements.

Simultaneous values imply the existence of a non-unique value for a single data item, even at the same time point. The issues related to simultaneous values have not been fully investigated in previous works. The ability to update simultaneous values results in new features of both update and retrieval operations in temporal databases. Most of the research in this area has focused on the structural semantics, using limited update protocols. For example, in Snodgrass (1987), the valid time of a new value for a data item is derived by subtracting all existing valid times for that data item from the user defined valid time. Other models, including Ariav (1986), Shoshani and Kawagoe (1986) and Wiederhold *et al.* (1991), also enforce a single value for each time point. Consequently, a mechanism to handle simultaneous values is either disabled or applied in an unnatural manner.

Several works, including Abbod *et al.* (1987), Edara and Gadia (1993), Klopprogge and Lockman (1983), Sarda (1993), Su and Chen (1991) and Wau (1991), have investigated aspects of temporal active databases. In Klopprogge and Lockman (1983), a limited active framework was introduced, in which past states that are not explicitly stored, can be inferred from the current state, using ground rules. Integrity constraints that vary with time are discussed in Abbod *et al.* (1987). The OSAM*/T, presented in Su and Chen (1991) is an object-based temporal knowledge representation model that combines update rules with temporal characteristics, extending the object-oriented model OSAM* (Su *et al.* 1989). Rules in OSAM*/T are used to capture temporal semantics beyond the valid start and end times of a value. Corrections result in overwrites or deletions, resulting in possible loss of information. The temporal update uses a restricted update protocol. The information modelling and analysis methodology defined for CIM-OSA is largely based on a predecessor of the OSAM*/T, called the SAM* model (Su 1986). It is possible to extend the CIM-OSA's information modelling to include rules and some temporal elements from OSAM*/T, but the notions of simultaneous values and data dependencies would still not be supported. The planning database model, presented in Wuu (1991), is based on the Extended Entity-Relationship (EER) model, in which the database components change states as a result of external events. However, the active part of the model is based on imperative programming and lacks proper mechanisms to support database consistency.

Other works, including Chakravarthy (1989), Gehani *et al.* (1992) and Dey *et al.* (1992), add temporal events to active databases, but do not handle temporal conditions or temporal actions. These works have a common drawback which stems from the fact that the functionality provided by active databases extended with temporal events is a subset of the functionality of temporal active databases. Unlike these extensions, the relationships and interactions between the temporal and the active dimensions in temporal active databases are explicitly modelled.

1.3. *Temporal database with data dependencies — an overview*

The architecture presented in this work is aimed at supporting CIM systems and other systems of analogous complexity. This support is achieved by embedding data dependencies and temporal information in a database. We assume an append-only database, in which changes can be introduced to data and metadata (including data dependencies). Each one of these changes can be either retroactive or proactive. Data dependencies are presented by using *invariants* (Etzion 1993). Invariants are consistency assertions that the database should maintain at all times. For example, the invariant *Total-Price:*= sum (*Quantity* * *Price-Per-Unit*)

states that a modification of an order quantity or a part quantity requires re-computation of the total order price. The temporal capabilities are made possible through time-supporting data structures, associated with data items.

Our work demonstrates these extended capabilities by modelling a CIM application in a temporal database with data dependencies. We show the role of both temporal structure and data dependencies in maintaining the correctness of the database even when engineering changes are applied to a product in the database. We also exemplify the benefits of using the architecture for supporting automated decision making.

The rest of the paper is organized as follows. In the following three sections we introduce the database features in three layers. The basic data architecture is introduced in Section 2, the temporal information representation on top of the basic structure is presented in Section 3. The data dependencies representation is described in Section 4, using the notations defined in the previous two sections. Section 5 exemplifies the use of the database abstractions for managing engineering changes.

2. The basic database architecture

In this section we present the basic data structures of the database. Sections 2.1 and 2.2 provide the details of the case study. The translation of the data involved in the case study to a database is given in Section 2.3.

2.1. *Flanger—a detailed CIM case study*

The manufacturer Flanger operates in the metal industry. One of its products is the 'double-disc-flange', drawn in Figure 1. It comprises two six-hole-discs (Figure 2), six bolts (Figure 3) and six nuts (Figure 4).

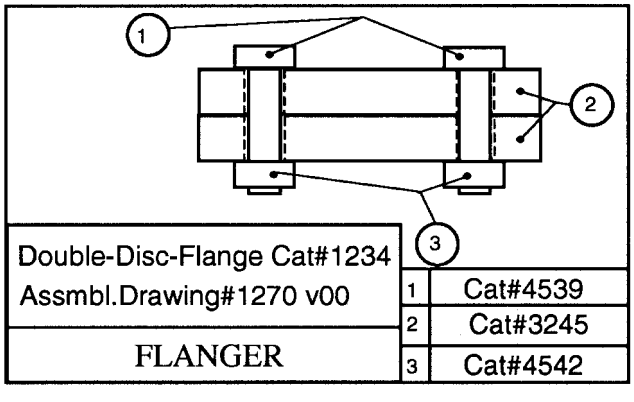

Figure 1. Double-disc-flange.

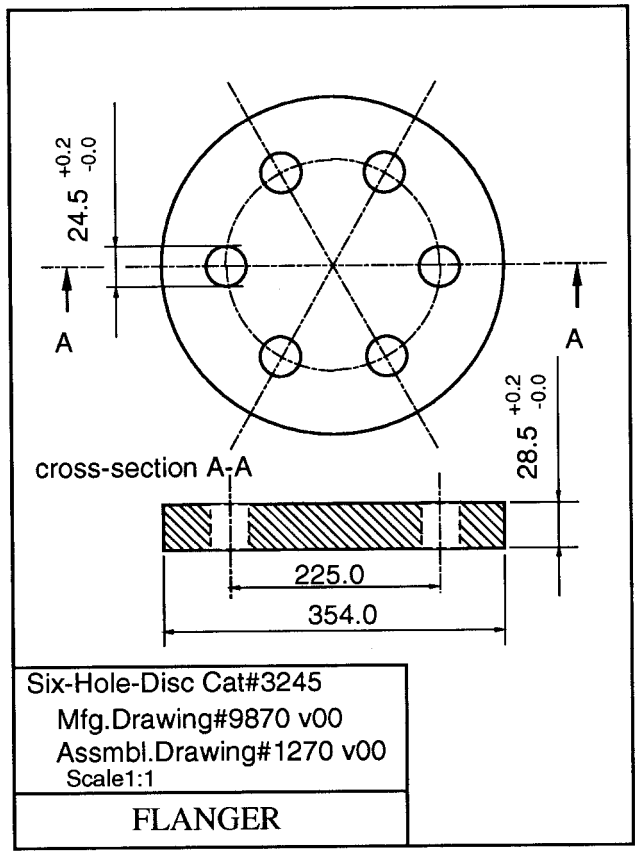

Figure 2. Six-hole-disc.

2.1.1. *The bill-of-material.* Each product has a bill-of-material (BOM), or product tree—information about the parts comprising the product, organized in a tree structure. Each node in the tree is a part assembled from parts that are its children in the tree. The root of the tree is a complete product that is shipped out of the factory, while the leaves are raw materials or parts, bought by the factory. Each arc in the tree is annotated by two numbers. The first is the assembly ratio, i.e. the

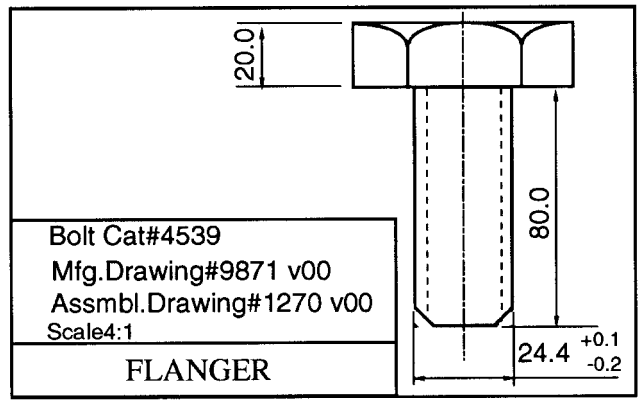

Figure 3. Bolt.

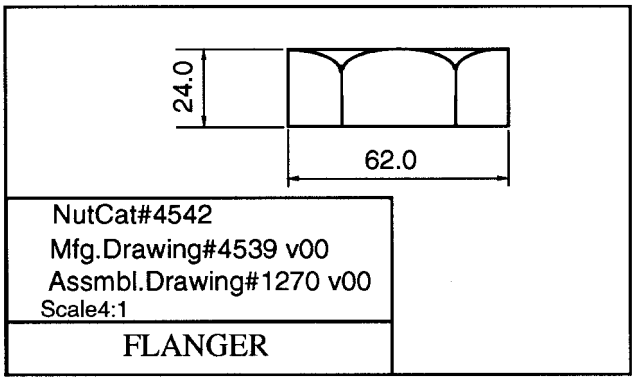

Figure 4. Nut.

(integer) number of parts of the immediate successor that is required in the final assembly of a single product. The second number is the consumption ratio, i.e. the running average number of parts actually required for assembly of a single product due to fallout in the manufacturing process. The consumption ratio is a derived information, updated and refined over time. A PERT network (Hillier and Liberman 1990) for manufacturing the product can be automatically extracted from the knowledge about the nodes in the product tree. Figure 5 presents the BOM of the double-disc-flange.

2.2. *The engineering database*

Each product in the CAD system used by Flanger is defined geometrically using three complementary methods: constructive solid geometry (CSG), boundary representation (B-rep) and wireframe.

In the *CSG model*, the geometry of each part of the product is determined by a set of binary Boolean operations (union, intersection and difference) applied to 3-D primitives (cylinder, cone, sphere, etc.). In the *B-rep model*, the geometry of each part is determined by a set of faces (boundaries) of the object, and in the *wireframe model*, the product is described as a set of vertices and edges in 3-D. Wireframe is the model

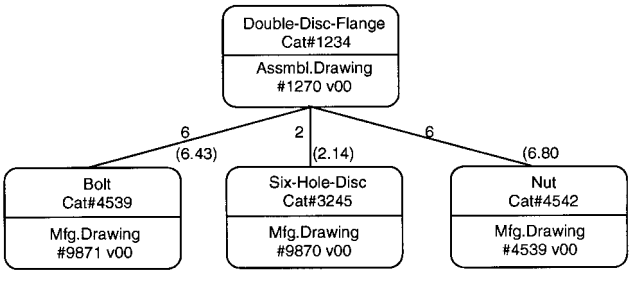

Figure 5. Bill-of-material of the double-disc-flange.

that also outputs drawings of the product and each one of its parts. All three types of product representations must be coherent: changing the product definition in one of them must be reflected in the others. Two possible ways to achieve coherence is to use *feature extraction* (Lee and Fu 1987, Joshi and Chang 1987) or *feature definition language* (Laakko and Mantyla 1991).

An *engineering drawing* is a 2-D description of a product or of part(s) of a product. An engineering drawing is either a manufacturing drawing or an assembly drawing.

A *manufacturing drawing* is an engineering drawing that describes the precise geometry and tolerances of one or more parts that belong to a product by annotating its projections using some dimensioning and tolerancing standard, such as ISO or ANSI.

An *assembly drawing* is a drawing that describes the topology and assembly order of two or more parts of a product. Figure 1 is an example of an assembly drawing, which describes how the parts, shown in Figures 2–4, are to be assembled.

2.2.1. *The manufacturing process.* A *manufacturing process* is attached to each node in the product tree. The manufacturing process describes how the part associated with the corresponding node is to be made. The process is a series of *technological operations*, which convert the raw material into a part ready for assembly. Each operation has two parameters:

Technology. The machine, set of tools, craftsman qualifications and the applicable NC code fragments needed to manufacture a given part. Alternative technologies that enhance the process versatility are optional.

Completion time. The time required for completion of the operation. The completion time includes two components: setup time and production time of one unit, each of which may be a random variable with some given distribution and parameters.

2.2.2. *Customers and inventory information.* The details of all the company's customers are represented in the database. For each customer, credit, balance, payments and a list of orders is kept, along with the customers basic information (name, address, etc.).

Each part in the company's inventory is recorded along with the following details: catalogue number, standard name and minimal, maximal and current inventory levels in each of the warehouses of the company. In addition, each bought part has details regarding suppliers that provide the part, supplier catalogue number and supplier lead time. If a stored

part gets close to its minimal level, an automatic order is issued, based on well-defined considerations. The ordered quantity is derived by taking into account the maximal level of that part, the current level in the inventory, the part lead time, and the rate of consumption.

2.3. *The database representation*

The database schema of 'Flanger' is depicted in Figure 6. The database in our model is a collection of *objects*. For example, *Double-Disk-Flange* is an object in Flanger's database. A complex object can be viewed

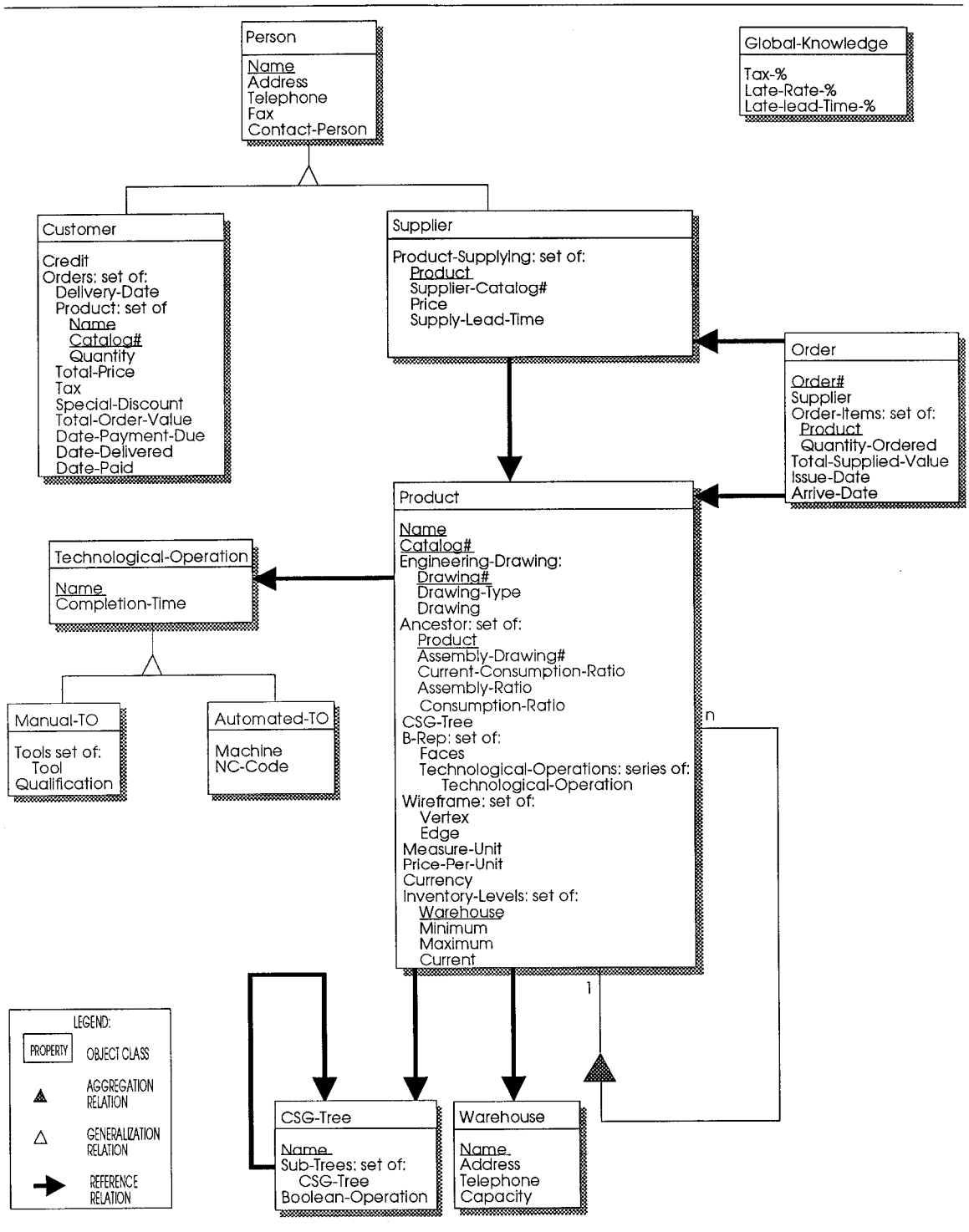

Figure 6. Schema of 'Flanger' database.

as an object that embeds other objects. (The approach taken here is an object-oriented database approach (Bancilhon 1988). A relational database approach would require a *normalized* database, where each relation embodies only simple properties.) Similar objects are instances of the same *class*. For example, the class *product* has Double-Disk-Flange and many other products as its instances.

An object class definition includes the specification of *properties* that are applicable to its instances. For example, *Name* and *Price-Per-Unit* are properties of *Product*, hence each instance of *Product*, including Double-Disk-Flange, has these properties. Each property can be either a *simple property* (e.g. *Name*) or a *complex property*. A complex property is a tuple or a set that consists of other properties. An object has a set of associated *variables*, each of which is an instance of the corresponding object class property.

Classes in the database scheme are interconnected by structural relations: generalizations (e.g. *Company/Person* is a generalization of *Customer* and *Supplier*) and aggregations (e.g. the class *Product* may be an aggregation of several constituent objects or parts of *Product*). All specialized classes, such as *Customer*, inherit all properties and relations from their generalized class (which, in this case, is *Company/Person*). The aggregation relations among parts of *Product* of the various levels constitute the bill of material. For example, *Six-Hole-Disc*, *Bolt* and *Nut* are instances of the *Product* class. These products are part of *Double-Disc-Flange*, which is also a product. The aggregation relation is an abstraction of this feature.

The underlined property in the classes *Person*, *Product*, *CSG-Tree* and *Technological-Operation* in Figure 6 is the object identifier of a class and its specializations. An *object identifier* is a predefined subset of the object's variables which uniquely identifies it. For example, the object identifier of *Product* is the set of properties *Name*

Class= Product
Name= Double-Disk-Flange

Catalog#= 1234
Engineering-Drawing:
Drawing#= 1270v00
Drawing-Type= Assembly
Price-Per-Unit= 70.00
Currency= $US
Ancestor:
null

Inventory-Levels:
Minimal= 100
Maximal= 1000
Current= 250

Class= Product
Name= Bolt

Catalog#= 4539
Engineering-Drawing:
Drawing#= 9871v00
Drawing-Type= Manufacturing
Price-Per-Unit= 4.00
Currency= $US
Ancestor:
Product=Double-Disk-Flange
Current-Consumption-Ratio=7
Assembly-Ratio=6
Consumption-Ratio=6.43
Inventory-Levels:
Minimal= 600
Maximal= 6000
Current= 1700

Class= Product
Name= Six-Hole-Disk

Catalog#= 3245
Engineering-Drawing:
Drawing#= 9870v00
Drawing-Type= Manufacturing
Price-Per-Unit= 43.00
Currency= $US
Ancestor:
Product=Double-Disk-Flange
Current-Consumption-Ratio=2
Assembly-Ratio=2
Consumption-Ratio=2.14
Inventory-Levels:
Minimal= 200
Maximal= 2000
Current= 250

Class= Product
Name= Nut

Catalog#= 4542
Engineering-Drawing:
Drawing#= 4539v00
Drawing-Type= Manufacturing
Price-Per-Unit= 1.75
Currency= $US
Ancestor:
Product=Double-Disk-Flange
Current-Consumption-Ratio=7
Assembly-Ratio=6
Consumption-Ratio=6.80
Inventory-Levels:
Minimal= 600
Maximal= 6000
Current= 1000

Figure 7. Portion of the Flanger database at time t_0.

and *Catalog#* . Since *Global-Knowledge* is a singleton class, it does not require an object identifier.

In the non-temporal version, the value of a variable can be an atom, a set, a sequence, a tuple (i.e. a non-homogeneous sequence) or a reference to another object. A reference is a relation from a property of a class to another class. For example, using the reference relation, the *Supplier* property of *Order* is related to the class *Supplier*. The reference relation is subject to the dependencies of *referential integrity*. Referential integrity implies, for example, that order O can be issued to supplier S if and only if supplier S exists in the database.

In general, each instance of *Product* may participate in more than one BOM. Therefore, each product or part may have a set of ancestors, one for each BOM in which the product participates. For each ancestor, the product has its *Assembly-Quantity*—the integral number of parts required in the final assembly, the *Current-Consumption-Quantity*—the integral number of parts actually required in the last time the part was assembled in *Product*, and the *Consumption-Quantity*—a statistically calculated quantity regarding the running average number of parts actually required, taking into account loss due to fallout in the manufacturing process of *Product* during a defined moving time window.

A variable of type *Ancestor* is a set of tuples which is considered as a part of another object. Its syntactic structure is similar to that of an object, but its identifier is a concatenation of the object identifier and its own identifier. For example, as Figure 6 shows, *Ancestor* is represented as a set of objects within *Product*. An object of type *Ancestor* is identified using the properties *Name* and *Catalog#* (of *Product* class), and *Product* (of *Ancestor*). If a variable is a set of tuples, as in the *Ancestor* case, each tuple in the set is an object with a unique identity. A class description represents the structure of its instances. Several instances of the object class *Product* are described in Figure 7.

3. Representation of temporal information: variable states

In this section we extend the basic data architecture to include temporal support. While in conventional databases the handling of temporal data is left to the user's discretion, we present a model in which temporal relationships are handled by the system. For example, *Current-Consumption-Ratio* is a property that changes over time. In conventional databases, the user can define this property as a set of pairs $\langle Value, Time \rangle$. The user should process these pieces of information to handle the temporal properties. In a temporal model, time is handled by the system. The temporal model allows a novice user to use the system without defining even a single time point, taking advantage of system defined defaults, while an expert can intervene in the system decisions and affect the temporal properties of data items. We present and demonstrate the main principles of temporal representation, which is further discussed in Etzion *et al.* (1993).

3.1. *Time perspectives*

As argued in Snodgrass and Ahn (1986), more than one type of time is needed to be associated with a variable's value to record its behavior over time. A basic set of time type consists of the following three time definitions.

Transaction time (t_x)—the time when a data item becomes current in the database. This time type may be implemented by using a transaction commit time, i.e. the time point at which a transaction is (successfully) terminated.

Decision time (t_d)—the time at which a data item's value was decided in the database's domain of discourse (Etzion *et al.* 1993). This time point denotes the time at which an event occurred, or the time at which a decision was made. For example, if a new *Price-Per-Unit* was set for a product, t_d would be the time point of the decision regarding the new price. From the database point of view, t_d reflects the time point where an occurrence in the modelled reality entails a decision to initiate a database update transaction. If a value was decided at t_1, and committed by the database at t_2, then $t_d = t_1$ and $t_x = t_2$. The decision time represents the correct order of events in the real world, which is not always identical with the order of transaction times.

Valid time (t_v)—the set of time points at which the data item is considered to be true in the modelled reality. The valid time is expressed using a *temporal element* Gadia 1988), which is a time-point or an interval $[t_s, t_e]$, where t_s and t_e are the interval start and end times, respectively, or a collection of intervals and/or time-points.

The relationships among the various time types are expressed by two constraints:

(1) $t_s \leqslant t_e$ (intervals cannot be negative); and
(2) $t_x \geqslant t_d$ (decisions cannot be speculated).

The transaction time (t_x) is set by the system, while the decision time (t_d) and the valid time (t_v) are defined by the user. Yet it is possible to set defaults that would exempt the user from the need to specify any time values. A reasonable set of defaults would be $t_d = now()$, and $t_v = [now(), \infty)$, where $now()$ is a function that returns the current time of the system clock. The term ∞ stands for the fact that the value of the upper bound of the time interval is believed to hold forever (Plexousakis 1990), yet it can be modified. An interval of the type $[t_1, t_2]$ contains the time point t_1 but not t_2. The default settings can change from one application to another.

The user has the option to insert the valid time and decision time into the database. However, the manipulation of the temporal knowledge is handled only by the system. For example, the system determines the valid values of a property at a given time point. This mechanism is more adequate for modelling complex applications, as the user is not required to manipulate the time notion.

3.2. *Variable states*

The temporal database architecture extends basic database concepts by incorporating the time perspectives defined above. Each value of a variable is associated with a *temporal extension*—a triplet (t_x, t_d, t_v) of values of the three time types. A value along with its temporal extension is called a *state-element*. (Other attributes of information regarding the data-item source, validity, accessibility, etc., can be added to a state-element. These extensions are discussed in Gal *et al.* (1994).) We assume that there is no *a priori* decision as to whether a property is time-invariant or not. Thus, each variable is associated, by default, with a temporal extension. For reasons of space conservation, though, the user may be able to suppress the temporal extension for selected data items.

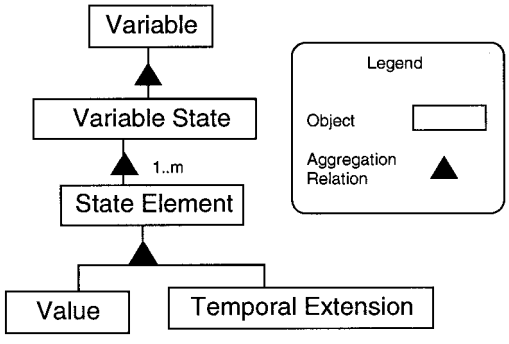

Figure 8. The structure of a variable in the temporal database architecture.

A *variable state* of a variable α is a sequence of state-elements representing the history of the variable's values. Figure 8 shows the structure of a variable in our temporal database architecture.

To exemplify the use of state-elements, we show below a variable state with three state-elements of the variable *Current-Consumption-Ratio* of *Bolt*, as assembled for *Double-Disk-Flange*.

$$(s1) = \{\ 6,\ \{t_x = Sep.1\,1995,\ 8{:}02am,\ t_d = Sep.1\,1995,\ 8{:}00am,\ t_v = [Sep.1\,1995\ 8{:}00am, \infty)\}\}$$
$$(s2) = \{\ 7,\ \{t_x = Sep.1\,1995,\ 9{:}00am,\ t_d = Sep.1\,1995,\ 8{:}30am,\ t_v = [Sep.1\,1995\ 8{:}00am, \infty)\}\}$$
$$(s3) = \{\ \underbrace{6}_{value},\ \underbrace{\{t_x = Sep.1\,1995,\ 9{:}05am,\ t_d = Sep.1\,1995,\ 9{:}00am,\ t_v = [Sep.1\,1995\ 9{:}00am, \infty)\}}_{temporal\ extension}\}$$

Each state-element is constructed to reflect a change of the variable state. For example, (s1) indicates the assignment of the value '6' to the variable *Current-Consumption-Ratio* (s2) 'corrects' the value to '7' and (s3) indicates a new assignment of '6' to the same variable. The use of t_d in this example suggest that the original value, '6', was decided upon at 8:00am, while the revised value was given at 8:30am. The decision time cannot be replaced with the transaction time in this case, since its accuracy is subject to the load on the DBMS.

In temporal databases, change of decisions about the value and validity time of a property may cause a situation where two values of the same variable have overlapping valid times. For example (s2) and (s3) have an overlapping valid interval ([Sep. 1 1995 8:30am, Nov. 5 1995 9:00am]). The question 'which one of these state-elements is valid in December 1995?' does not have a trivial answer. In some cases, the value that was inserted later corrects an earlier value. In other cases, both values are possibly correct, each with respect to a different time point. For example, (s2) is valid in December 1995 from a view point of Sep. 1 1995 from 8:30am to 9:00am, and (s3) is valid in December 1995 from a viewpoint later than Sep. 1 1995, 9:00am.

This mechanism, which represents changes in the real world, is required in a temporal database that supports past viewpoints, i.e. the retrieval of the state of the database as it was observed from some time point in the past. For example, we can issue a query of the form: what was the value of *Current-Consumption-Ratio* of *Bolt* as assembled for *Double-Disk-Flange* on Sep. 1 1995 at 8:15am, as seen from a viewpoint $t <$ Sep. 1 1995 at 8:30am. Many temporal models (e.g. Rose and Segev 1991) classify this type of real world changes as *corrections*, thereby blurring the distinction between changes in data referencing the real world model and corrections of erroneous values that were inserted into the database as a result of typographical or other errors.

As illustrated above, state-elements of the same variable state may have intersecting valid times. To select the desired value, we device a *preference relation* among state-elements. Preference relations may be

defined using various criteria, including the source of the information, a certainty value associated with the information, any one of the time types, etc. We denote the preference relation by the $\prec$ symbol, such that $s_i \prec s_j$ means that s_j is preferred over s_i. We define our preference relation on the basis of the following axiom of the time monotonicity of knowledge.

The time monotonicity of knowledge axiom

Given two state-elements s_i and s_j of the same variable α, such that $t_v(s_i) \cap t_v(s_j) = \tau$ and $\tau \neq \emptyset$, for each $t \in \tau$, $s_i \prec s_j$ if and only if $t_d(s_j) > t_d(s_i)$.

The axiom represents the belief that knowledge improves monotonically with time, since a later decision is based on additional accumulated knowledge. The decision making regarding the validity of a value at a given time point is made automatically by the system, using predefined preference relations. For example, using the preference relation based on this axiom, the valid value of *Current-Consumption-Ratio* of *Bolt* in *Double-Disk-Flange* in Sep. 1995, as observed from Jan. 1996 is '7' (using (s2)) and not '6' (the value of (s1)).

4. Data dependencies

Data dependencies are dependencies among variables in the database. For example, the value of *Consumption-Ratio* is dependent upon the value of *Current-Consumption-Ratio*. Data dependencies are enforced using a data-driven mechanism that activates calculations as a response to specific data modifications. For example, modification of *Current-Consumption-Ratio* results in re-calculation of *Consumption-Ratio*. The definitions of the activities that should be performed when a variable is modified are embedded in the database using two language constructs: derivations and constraints. Derivations and constraints define the semantic requirements of the database. The database is *consistent* if and only if all its derivations and constraints are satisfied.

A *derivation* is a language construct which defines the relationship that must hold between a *derived variable* and its *derivers*, where a *deriver* is a variable in the database whose value participates in the derivation.

A *derived update* is a database update operation of a derived variable that may be activated whenever one or more of the derivers associated with a derivation changes. The derivers appear on the right-hand side of the derivation. Several examples follows.

(d1) Total-Price := sum (Quantity * Price-Per-Unit
(d2) Tax := Total-Price * Tax% / 100
(d3) Consumption-Ratio := temporal-avg (Current-Consumption-Ratio)
(d4) Total-Order-Value := Total-Price + Tax − Special-Discount when Date-Paid ⩽ Date-Payment-Due
(Total-Price + Tax − Special-Discount) *
(Date-Paid − Date-Payment-Due) * Late-Rate% otherwise

The derivation (d1) calculates the total price of an order as the sum of consumed product's quantities multiplied by the product price per unit. An update to *Quantity* and/or *Price-Per-Unit* triggers (d1). Although *Quantity* and *Price-Per-Unit* are properties of different classes, there is no need for explicit matching definition of the relevant object instances, as this matching is determined by the system. The derivation (d2) calculates the tax to be added to the order's total price, using the *Tax%* from the *Global-Knowledge* class.

The derivation (d3) uses cumulative historical information to calculate the statistic *Consumption-Ratio*. The function **temporal-avg** uses all preferred state-elements (based on the preference relation defined for the application) to derive the average ratio. For example, the state-elements of *Consumption-Ratio* derived from *Current-Consumption-Ratio* using (d3) are presented below. The value, t_d and t_v of the derived state-elements are derived from the state-elements of *Current-Consumption-Ratio*, as shown in Section 3.2.

$(s'1) = \{ \ 6.0, \ \{t_s = Sep.11995, 8{:}02am, \ t_d = Sep.11995, 8{:}00am, \ t_v = [Sep.11995\,8{:}00am, \infty)\}\}$
$(s'2) = \{ \ 7.0, \ \{t_s = Sep.11995, 9{:}00am, \ t_d = Sep.11995, 8{:}30am, \ t_v = [Sep.11995\,8{:}00am, \infty)\}\}$
$(s'3) = \{ \ \underbrace{6.0}_{value}, \ \underbrace{\{t_s = Sep.11995, 9{:}05am, \ t_d = Sep.11995, 9{:}00am, \ t_v = [Sep.11995\,9{:}00am, \infty)\}}_{temporal\ extension}\}$

The derivation (d4) calculates the total order value for a clients' orders. The calculation depends on whether or not the payment for the order is made on time. If not, the value increases by adding the late rate percentage multiplied by the number of days past due.

A *constraint* is a language construct that defines anassertion which restricts a consistent state of the database. A *constraint enforcement* is a data-driven activity, i.e. when one of the participating variables in a constraint is modified, the constraint validity is evaluated, and if the constraint is violated, an *exception handler* is invoked (Etzion 1991). Two constraint examples follow:

(1) (c1) Inventory-Levels. Minimal ⩽ Inventory-Levels. Maximal

The constraint (c1) restricts the upper/lower bounds on the different inventory levels.

(2) (c2) Inventory -Levels. Current ⩽ Inventory-Levels. Maximal
An update that either increases *Inventory-Levels. Current* or decreases *Inventory-levels. Maximal* activates the constraint (c2) to restore the database consistency.

In the first constraint, we can assume an exception handler of type *abort*, where the transaction that caused the constraint violation should not be committed. In the second constraint we can assume an exception handler of type *repair transaction*, which return the 'leftovers' to the supplier.

Figure 9 presents an object-process diagram (OPD) (Dori 1995) of the temporal database architecture with data dependencies. An OPD represents objects and processes by rectangles and ellipses, respectively, with solid and blank triangles denoting aggregation and generalization, respectively. The semi-solid triangle denotes an instantiation relationship.

5. Managing engineering changes through database abstractions

An *engineering change* is a change in one or more of the parts of which a product is assembled or a change in the way it is manufactured. Engineering changes are introduced either to improve the product functionality or to save production costs. These changes may affect the properties of the product. For example, a change in one of the parts of which a product is assembled causes a change in both its assembly drawing and a reconstruction of its bill-of-material. Temporal knowledge and data dependencies may be useful tools for managing engineering changes.

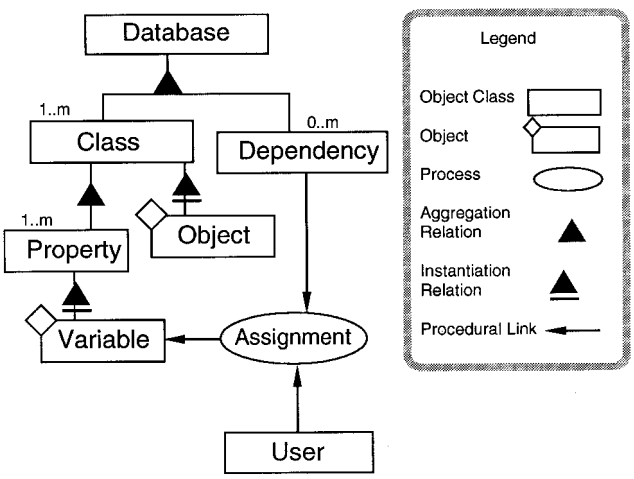

Figure 9. Object-process diagram of the temporal active data architecture.

The use of a temporal database provides for better control of product development over time. The task of product configuration management is highly complex. Engineering changes are therefore clustered into periodic changes called *versions*. However, given the current marked environment, a manufacturer can lose a competitive edge by postponing the application of an engineering change to the next version. A change in managerial strategy supported by an intelligent temporal database, would encourage the application of an engineering change as soon as it is technically possible to do it. The engineering change, along with its validity time, is recorded in the database and the manipulation of the data is done automatically by the system. Each change is time-stamped, so that it can later be traced by the system to recreate the *valid version* that was available at a given time point.

An engineering change in a typical real-life manufacturing scenario involves a host of activities, including signoffs and approvals by the authorized designers and logistics personnel, possible changes to relevant NC programs, manufacturing schedules, etc. Each such activity should have (and in practice has a partial) temporal extension. Our claim is that using a reliable, well developed temporal database with data dependencies, it is possible to implement the strategy of applying an engineering change as soon as it becomes feasible, rather than wait for a major new version. The implementation entails that the engineering change be defined and the engineering change process be embedded as a set of constraints within the database. A top-level rule, for example, might look like: 'When all the activities for applying the engineering change have been completed, send the engineering change order to the manufacturing department.' This rule is recursively broken into simpler rules. At the bottom level these rules assume the form of temporal database constraints, and when all the relevant constraints are satisfied, the engineering change is launched.

In this section we introduce an example of an engineering change, made to the Flange, its effect on the CIM database and the use of data dependencies in engineering change management and in automatic decision support. We assume that all the preparatory activities for the engineering change described above have been completed, and concentrate on the addition of several derivations, which enable the computation of the number of parts to be produced so as to minimize dead stock.

5.1. *An engineering change scenario*

As a result of value analysis, the double-disc-flange of Figure 1 is to undergo an engineering change on December 12, 1995, 9:00am. The modified product would still carry the same functionality, yet its components would be changed as follows.

(1) The new assembly process uses three sets of standard 3″ bolts and nuts instead of the original thinner six fabricated sets. The price of the thinner bolt and nut is \$4.00 and \$1.75, respectively, while a standard 3″ bolt and nut cost \$2.00 and \$0.75, respectively. The saving on bolts and nuts is therefore $6\times(4.00+1.75)-3\times(2.00+0.75)=\$26.25$ per unit product.

(2) The new *Double-Disc-Flange* has just three larger holes with looser tolerances. The bigger hole diameter and looser tolerances enable a change in the manufacturing process so that instead of drilling the six holes using a CNC machine, the three holes are done by the plasma cutting machine. The cost of the Three-Hole-Disc is \$34.00, compared with \$43.00 for the drilled Six-Hole-Disc. The saving on discs is $2\times(43.00-34.00)=\$18.00$. Hence, the total saving per one *Double-Disc-Flange* is \$44.25, about 44% saving.
On December 12, 1995, 9:00am, there are 250 units of *Six-Hole-Disk* in the plant's inventory. The decision of introducing the engineering change, made on November 10, 1995, 8:00am, resulted in the following future changes in the database:

(3) The *Assembly-Drawing#* accepts a new value (1270v01) as of December 12, 1995, 9:00am.

(4) The *Assembly-Drawing* itself accepts a new value (which is the graphics shown in Figure 10) as of the same time point.

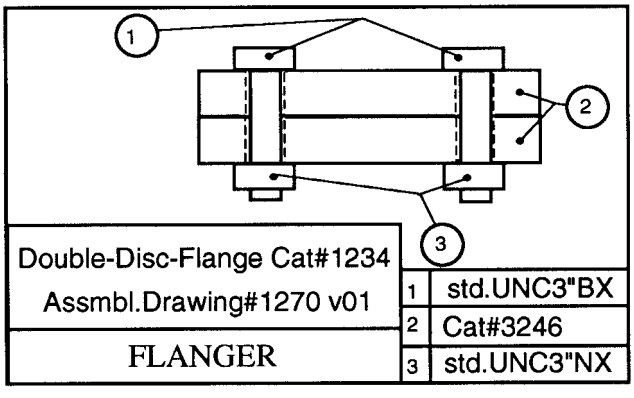

Figure 10. Double-Disc-Flange as of December 12th 1991, 8:00am.

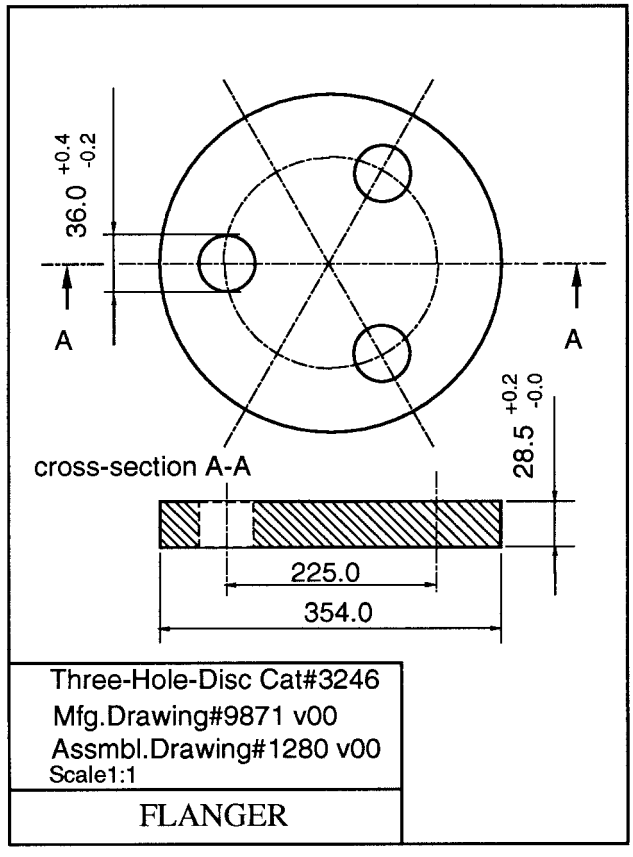

Figure 11. Three-Hole-Disc.

(5) A new part is defined in the database on November 10, 1995, 9:00am. Its name is *Three-Hole-Disc*. Its manufacturing drawing is shown in Figure 11, and its ancestor is *Double-Disc-Flange*. Its *Assembly-Ratio* is set to 2, while its *Current-Consumption-Ratio* and its *Consumption-Ratio* have no initial values. The validity of this part (which means its availability for use) is as of December 12, 1995, 9:00am.

5.2. *Data dependencies for automated decisions making*

An engineering change involves a change in the production schedule. Assembling the part according to the new assembly drawing can start on December 12, 1995, 9:00am, yet the production of the old version cannot be stopped immediately. The current inventory of each part of the older version, the customers orders for the product and the number of old version units in the field that need spare parts should all be taken into consideration when making the scheduling decision. For example, since the bolts and nuts in the old version of the assembly drawing were not standard, we would not use them in other products, and they will be

considered as dead stock. Suppose that in order to eliminate dead stock as much as possible, we wish to use all of the parts left in the inventory. To achieve our goal, we would assemble old versions of Double-Disc-Flanger, with minimum leftovers of Six-Hole-Discs, Bolts and Nuts. All that is required for that decision is a small change in the database schema and three new data dependencies.

In the database scheme of Figure 6, each instance of *Product* has a property named *Engineering-Drawing*, which is a tuple of *Drawing#* , *Drawing-Type* and *Drawing*. We add to it the property *Residual-Production*. This property defines the number of units left to be produced or purchased. Usually, this number is not limited, so its value would be ∞, which is a symbol for 'unlimited'. However, when the user wishes to limit the stock, as in the case of an engineering change, this number is replaced by the number the user would like to produce or purchase from that time on.

Limiting the production of a certain product affects all its predecessors. To infer this effect automatically, we introduce the following new derivation:

(d5) Engineering-Drawing.Residual-Production := sum(Ancestor.Engineering-Drawing.Residual-Production*Consumption-Ratio) − Inventory-Levels.Current

Derivation (d5) states that for each product (part), the residual production amount is the difference between the requirements, coming from its ancestors, and its current level. A single ancestor with the value of ∞ is enough to give the same value to the predecessors. However, if the amount of all ancestors is limited, the amount of the predecessor is limited as well.

In case of an engineering change, we allow a direct update to the *Residual-Production* property. A violation, in this case, that is, a direct change of the instance value, affects its ancestors. For example, consider the values given in Figure 7. There are 250 units of Double-Disc-Flange of the old version in the inventory, 1250 Bolts, 250 Six-Hole-Discs and 1000 Nuts. Assuming all assembly parts of Double-Disc-Flange are used solely for that purpose and using dependency (d5), a modification of *Residual-Production* of *Double-Disc-Flange* to 250 (500 including parts in the inventory), would automatically modify *Residual-Production* of Bolts to be 358 (1250 current inventory, subtracted from 250 residual production of *Double-Disc-Flange* multiplied by a consumption ratio of 6.43), of Six-Hole-Disc to be 285 and of Nuts to be 700. On the other hand, a modification of *Residual-Production* of Double-Disc-Flange to 150 (400 including parts in the inventory), would automatically modify *Residual-Production* of Bolts to be −285. A negative number means leftovers of bolts. To prevent it, we add a new data-driven constraint to the system:

(c3) Engineering-Drawing.Residual-Production $\geqslant$ 0

In the case of violation, the database activates an automatic 'consistency restoration' mechanism, which modifies the database in order to maintain consistency. To satisfy the constraint, by reverse calculation of the dependency (d5), the value of *Residual-Production* of Bolts would be 0, due to constraint (c3); that is, there are 1250 bolts in the inventory. Since *Double-Disc-Flange* requires 6.43 Bolts per item, an extra production of $1250/6.43 \approx 194$ units is required, instead of 150. Using (d5) again, the value of *Residual-Production of Six-Hole-Disc* would be 165 and that of *Nuts* would be 319.

Finally, to prevent production of parts in quantities greater than required by the *Residual-Production* property, we add another data-driven constraint:

(c4) Engineering-Drawing.Residual-Production $\leqslant$ old (Engineering-Drawing.Residual-Production).

The reserved word *old* obtains the value that existed prior to the transaction start. (c4) is a monotonicity constraint. It states that the value of the *Residual-Production* property can only decrease. Since *Residual-Production* is affected by the number of parts in the inventory, no new parts can be added to the inventory. Any exception would result in a rejection of the transaction.

This detailed case study clearly demonstrates how a temporal database with data dependencies can support automatic decision making for shop-floor control. The proposed temporal support provides for convenient management of simultaneous versions by assigning the appropriate valid time to each version. The information of the validity of a version derives the engineering information that is relevant to that version. For example, the version 1270v01 is valid as of December 12, 1995, 9:00am. As a result we can conclude that the *Three-Hole-Disc* is the valid disc that is used for the product, rather than the *Six-Hole-Disc*.

6. Conclusions

We have introduced a database architecture suitable for CIM that incorporates data dependencies and temporal information into an integrated system. The architecture was demonstrated by a case study that represents the type of challenges posed by complex systems, and how they can be met.

Currently, applications are developed using conventional models that force the user to use self-defined procedures to achieve the desired functionalities. Such modelling is tedious, time consuming and difficult to verify. Furthermore, the functionality

of the application is frequently comprised due to conceptual or technical limitations. The task of modelling systems can be made easier using the proposed database architecture instead of conventional DBMSs. The high level abstractions and the dependency system provide the basis for the following characteristics:

(1) Representation of complex real-life systems in a natural way.
(2) Reduction of programming efforts required outside the database.
(3) Support of automated decision making based on a set of predefined constraints and derivations.

The extra space and overhead requirements of the proposed architecture are not negligible. However, the case study demonstrates that the potential benefits that can be gained by using it outweigh this cost. The ever improving storage and retrieval technologies make this assertion more valid as time progresses.

Our temporal active database architecture is a step towards establishing a comprehensive tool for handling CIM and other applications of similar complexity. The number of applications requiring such high-level data models increases along with the growing demand for combining decision support systems within databases. The engineering change case study demonstrates the benefits of our approach for a manufacturer in current highly competitive markets. The use of data dependencies may help optimize decisions regarding quantities of parts to be produced, while automatic support of temporal knowledge makes it possible to keep track of a valid version even when many engineering changes are simultaneously involved. We believe that using this approach, a manufacturer can gain a substantial competitive advantage.

A prototype of the data dependencies model was built using MS-Access 1.0 for Windows, in the MS-Windows 3.1 environment. A full prototype, including the temporal support is currently under development on the basis of MS-Access 2.0 for Windows, in the MS-Windows 3.11 environment.

Ongoing research planned in this domain includes the following subjects:

(1) Dealing with various implementation issues, such as storage management, possible cases of incremental updates, flexible transaction protocol to allow asynchronous subtransactions, and query optimization.
(2) Extending the architecture to cope with additional dimensions, such as the belief dimension for uncertain data using some ordinal scale, the accessibility dimension, i.e. for which user the data is accessible, and the space dimension, e.g. manufacture floor planning and control.
(3) Extending the architecture to a distributed environment, with several local schemata, some of which are used as database servers while others play the role of clients. This extension can make use of the new generation of client–server DBMSs.
(4) Exploring the interaction between the product description and the manufacturing activities, as well as typing engineering changes in the product to the corresponding change in the extracted features as reflected over time. Some of the subjects related to CIM have been demonstrated in this work, but there are many other CIM aspects, like feature extraction, and manufacturing activities, that are left to be further investigated.

Acknowledgement

The theory of temporal data dependencies discussed in this paper was developed in collaboration with Professor Arie Segev of University of California, Berkeley.

References

ABBOD, T., BROWN, K., and NOBLE, H., 1987, Providing time-related constraints for conventional database systems. In *Proceedings of the 13th International Conference on VLDB*, Brighton, pp. 167–175.

ABITEBOUL, S., 1988, Update, the new frontier. In *Lecture Notes on Computer Science, 326* (Springer Verlag, Berlin), pp. 1–18.

ARIAV, G., 1986, A temporally oriented data model. *ACM Transactions on Database Systems*, **11,** 499–527.

BANCILHON, F., 1988, Object-oriented database systems. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 152–162.

BENNETT, R. E., 1987, Justifying the acquisition of automated equipment. *Management Accounting (US)*, XX, 39–46.

BOTZER, D., 1992, Active database optimization. Master thesis. Technion-Israel Institute of Technology.

CHAKRAVARTHY, U. S., 1989, Rule management and evaluation: An active dbms perspective. *SIGMOD RECORD*, **18,** 20–28.

CLIFFORD, J., and CROCKER, A., 1987, The historical relational data model (hrdm) and algebra based on lifespans. In *Proceedings of the International Conference on Data Engineering*, pp. 528–537.

DEY, D., BARRON, T. M., and STOREY, V. C., 1992, Events in representation of temporal data. In *Proceedings of the Workshop on Information Technologies & Systems (WITS)*, pp. 241–253.

DORI, D., 1995, Object-process analysis: Maintaining the balance between system structure and behavior. *Journal of Logic and Computation*, **5,** 1–23.

DOUMEINGTS, G., 1987, Grai integrated method: A methodology to implement cim in the enterprise. In A. W. Scheer, ed., *Proceedings of Workshop on Implementing CIM*, Saarbrucken, pp. 39–46.

EDARA, M. L., and GADIA, S. K., 1993, Updates and incremental recomputation of active relational expressions in temporal databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Database*, Arlington, TX.

ETZION, O., 1991, Active handling of incomplete or exceptional information in database systems. In *Proceedings of the Workshop on Information Technologies & Systems (WITS)*, pp. 46–60.

ETZION, O., 1993, PARDES—a data-driven oriented database model. *SIGMOD RECORD*, **22,** 7–14.

ETZION, O., 1994, Tapuz: an information repository approach to active database applications. Technical Report ISE-TR-94-2, Technion-Israel Institute of Technology.

ETZION, O., DORI, D., and NOF, S. Y., 1995, Active coordination of CIM multi database system. *International Journal of Computer Integrated Manufacturing*, **8,** 116–126.

ETZION, O., GAL, A., and SEGEV, A., 1993, Temporal active databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Database*, June.

GADIA, S. K., 1988, The role of temporal elements in temporal databases. *Data Engineering Bulletin*, **7,** 197–203.

GAL, A., ETZION, O., and SEGEV, A., 1994, Representation of highly-complex knowledge in a database. *Journal of Intelligent Information Systems*, **3,** 185–203.

GEHANI, N., JAGADISH, H. V., and SHMUELI, O., 1992, Composite event specification in active databases. In *International Conference on Very Large Databases*, Vancouver, Canada.

HAYES, R. H., and JAIKUMAR, R., 1988, Manufacturing's crisis: New technologies, obsolete organizations. *Harvard Business Review*, September, 77–85.

HILLIER, F., and LIBERMAN, J., 1990, *Introduction to Operating Research*, 5th edn, (McGraw-Hill, New York), pp. 364–382.

HUDSON, S., and KING, R., 1986, Cactis: A database system for specification functionality defined data. In *Proceedings of the IEEE OOBDS Workshop*, pp. 26–37.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1991, Sub Committee 4: Product data representation and exchange—part 1: Overview and fundamental principles. ISO CD 10303–1.

JORYSZ, H. R., and VERNADAT, F. B., 1990a, Clim-osa part 1: Total enterprise modelling and function view. *Journal of Computer Integrated Manufacturing*, **3,** 144–156.

JORYSZ, H. R., and VERNADAT, F. B., 1990b, Clim-osa part 2: Information view. *Journal of Computer Integrated Manufacturing*, **3,** 157–167.

JOSHI, S., and CHANG, T. C., 1987, Graph-based heuristics for recognition of machined features from a 3-D solid model. *Computer-Aided Design*, **20,** 58–66.

KLOPPROGGE, M. R., and LOCKMANN, P. C., 1983, Modeling information preserving databases; consequences of the concept of time. In *Proceedings of the International Conference of VLDB*, Florence, Italy.

LAAKKO, T., and MANTYLA, M., 1991, A new feature recognition algorithm. In *Proceedings of Computer Applications in Production and Engineering (CAPE'91)* (Elsevier, Amsterdam), pp. 369–376.

LEE, Y. C., and FU, K. S., 1987, Machine understanding of CSG: Extraction and unification of manufacturing features. *IEEE Computer Graphics and Applications*, **7,** 20–32.

MONDEN, Y., 1983, *Toyota Production System: Practical Approach to Production Management* (American Institute of Industrial Engineering, Atlanta, Georgia).

NAVATHE, S. B., and AHMED, R., 1989, A temporal relational model and a query language. *Information Sciences*, **49,** 147–175.

PISSINOU, N., *et al.*, 1994, Towards an infrastructure for temporal databases. Technical Report TR 94-01, The University of Arizona.

PLEXOUSAKIS, D., 1990, An ontology and a possible-worlds semantics for telos. Technical Report KRR-TR-90-7, University of Toronto.

RANKY, P., 1986, *Computer-Integrated Manufacturing* (Prentice-Hall, Englewood Cliff, NJ).

ROSE, E., and SEGEV, A., 1991, Toodm-a temporal, object-oriented data model with temporal constraints. In *Proceedings of the International Conference on the Entity-Relationship Approach*, San Mateo, California, pp. 205–229.

SARDA, N. L., 1993, HSQL: Historical query language. In *Temporal Databases*, (Benjamin/Commings, Redwood City, CA.), Chapter 5, pp. 110–140.

SEGEV, A., and ZAHO, J. L., 1991, Data management for large rule systems. In *Proceedings of VLDB 91.*

SHOSHANI, A., and KAWAGOE, K., 1986, Temporal data management. In *Proceedings of the International Conference of VLDB*, pp. 79–88.

SNODGRASS, R., 1987, The temporal query language TQUEL. *ACM Transactions on Database Systems*, **12,** 247–298.

SNODGRASS, R., and AHN, I., 1986, Temporal databases. *IEEE Computer*, **19,** 35–42.

SNODGRASS, R., et al., 1994, TSQL2 language specification. *ACM SIGMOD RECORD*, **23,** 65–86.

META SOFTWARE, 1990, *Design/idef user's manual.*

SU, S. Y. W., 1986, Modelling integrated manufacturing data with sam*. *IEEE Computer*, **19,** 34–49.

SU, S. Y. W., and CHEN, H. M., 1991, A temporal knowledge representation model OSAM*/T and its query language OQL/T. In *Proceedings of the International Conference on VLDB.*

SU, S. Y. W., LAM, H., and KRISHNAMURTHY, V., 1989, An object-oriented semantic association model (OSAM*). In S. T. Kumera et al. eds, *AI: Manufacturing Theory and Practice* (Norcross, GA), Chapter 17.

URBAN, S. D., et al., 1994, A heterogeneous, active database architecture for engineering data management. *International Journal of Computer Integrated Manufacturing*, **7,** 276–293, 1994.

WANG, W., POPPLEWELL, K., and BELL, R., 1993, An integrated multi-view system description approach to approximate factory modelling. *Journal of Computer Integrated Manufacturing*, **6,** 165–174.

WIEDERHOLD, G., JAJODIA, S., and LITWIN, W., 1991, Dealing with granularity of time in temporal databases. In R. Anderson et al., eds, *Lecture Notes in Computer Science 498* (Springer-Verlag, Berlin), pp. 124–140.

WUU, G. T. J., 1991, SERQL: An ER query language supporting temporal data retrieval. In *Proceedings of the 10th International Pheonix Conference on Computers and Communications.*