

SWEDISH INSTITUTE FOR SYSTEMS DEVELOPMENT

---

SISU REPORT 1992 : 01

ISSN 1103-1700  
ISRN SISU-REP--01--SE

**THE ENTITY RELATIONSHIP TIME MODEL  
AND THE CONCEPTUAL RULE LANGUAGE**

**C. THEODOULIDIS  
P. LOUCOPOULOS  
B. WANGLER**



SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING

# **The Entity Relationship Time Model and the Conceptual Rule Language**

---

Abstract.....	1
1. Introduction .....	2
2. The Entity-Relationship-Time Model.....	7
2.1 Basic Concepts and Externals.....	7
2.2 Time Semantics.....	12
2.3 Complex Object Semantics.....	15
3. The Conceptual Rule Language.....	19
3.1 Constraint Rules.....	19
3.2 Derivation Rules.....	30
4. Conclusions.....	35
References.....	36

The Swedish Institute for Systems Development Report Series serves two main purposes.

As a considerable part of the our work is done in international collaboration it is important to make the results available to our member organizations in Sweden.

Publishing in English also serves the purpose of opening the results of the Institute to our colleagues abroad.

This Report on "The Entity Relationship Time Model and the Conceptual Rule Language" by C. Theodoulidis, P. Loucopoulos, B. Wangler is a result of the ESPRIT-project TEMPORA. It is published with the kind consent of the authors whom we thank warmly for this permission.

Kista september 1992

## The Entity Relationship Time Model and the Conceptual Rule Language

C. Theodoulidis<sup>1</sup>, P. Loucopoulos<sup>1</sup>, B. Wangler<sup>2</sup>

<sup>1</sup> Department of Computation  
UMIST  
P.O. Box 88  
Manchester, M60 1QD  
U.K.

<sup>2</sup> Swedish Institute for Systems  
Development (SISU)  
Box 1250  
S-164 28 KISTA  
SWEDEN

### Abstract

Recent years have witnessed an increased demand for information systems which cover a wide spectrum of application domains. This, inevitably, has had the effect of demanding conceptual models of enhanced functionality and expressive power than is currently possible in practice. This paper introduces the TEMPORA modelling paradigm for developing information system applications from a unified perspective which deals with definitional, intentional and constrain knowledge. The paper discusses in detail two of the components of the TEMPORA conceptual model namely, the Entity-Relationship-Time (ERT) model and the Conceptual Rule Language (CRL). The ERT model deals with the structural aspects including *time* and *complex objects* modelling whereas the CRL language deals with constraints and derivations on the ERT model.

## 1. Introduction

In recent years there has been an increasing demand for enhanced functionality of information systems that deal with applications beyond the traditional data processing type. It has also been argued that the development of the next generation of information systems will require the adoption of approaches which provide a closer alignment between business policy and system operation [van Assche et al, 1988; Loucopoulos, 1989]. A number of issues arising from these requirements are addressed within the TEMPORA<sup>1</sup> project [Loucopoulos et al, 1990] which provides the framework for the work reported in this paper. The need for enhanced system functionality is addressed through the use of a conceptual modelling formalism which caters for: the modelling of *business rules*, the modelling of *time* and the modelling of *complex objects*. This formalism is supported at the database level by an extension of the relational model with temporal semantics and an execution mechanism that provides *active* database functionality.

The TEMPORA paradigm advocates that development of an information system should be viewed as the task of developing or augmenting the policy knowledge base of an organisation, which is used throughout the software development process. Within TEMPORA, this knowledge base is concerned with the definition of the principal facts and operations within the organisation together with the rules which guide these operations and ensure the integrity of these facts.

The structural properties of an application are expressed in terms of the ERT model. The process component deals with the definition of operations. A process is the smallest independent unit of business activity of meaning, initiated by a specific trigger and which, when

---

<sup>1</sup> The TEMPORA project has been partly funded by the Commission of the European Communities under the ESPRIT R&D programme. The TEMPORA project is a collaborative project between: BIM, Belgium; Hitec, Greece; Imperial College, UK; LPA, UK; SINTEF, Norway; SISU, Sweden; University of Liege, Belgium and UMIST, UK. SISU is sponsored by the National Swedish Board for Technical Development (STU), ERICSSON and Swedish Telecomm.

complete, leaves the business in a consistent state. The analysis of processes in terms of the ERT model results in a set of primitive actions suffered by an entity such as 'salary increase'. Control of the behaviour of a system is modelled in terms of rules. Two general classes of rules are recognised: *constraint rules* which are concerned with the integrity of the database, *derivation rules* which are concerned with the derivation of new information and *action rules* which are concerned with the control of transactions.

This paper is concerned with a detailed description of the ERT model and the constraint rules and derivation rules of the CRL language. Details of earlier work in the process and rule models are reported in [van Assche et al, 1988; Loucopoulos, 1989; Theodoulidis et al, 1990; Loucopoulos et al, 1991, McBrien et al, 1991].

The ERT model uses as its basic structure the Entity-Relationship approach in order to preserve the well known advantages of this approach namely, graphical display, increased readability and wide acceptance by practitioners. The basic mechanism differs from the original Entity Relationship model [Chen, 1976] in that it regards any association between objects in the unified form of a relationship. Thus, the conceptually unnecessary distinction between attributeships and relationships [Kent, 1979; Nijssen, 1988] is avoided. On this basic mechanism the ERT model is extended in its *semantics* and *graphical notation* in two directions: the modelling of time; and the modelling of complex objects.

The need for modelling time explicitly is that, for many applications when an information item becomes outdated, it need not be forgotten. The lack of temporal support raises serious problems in many cases. For example, conventional DBMS cannot support historical queries about past status, let alone trend analysis which is essential for applications such as Decision Support Systems (DSS). The need to handle time more comprehensively surfaced in the early 70's in the area of medical information systems where a patient's history is particularly important [Wiederhold et al, 1975]. Since these early days there has been a large amount of research in the nature of time in computer-based information systems and the handling of the temporal aspects of data [Ariav & Clifford, 1984; Ahn & Snodgrass, 1988; Ben-Zvi, 1982; Lum et al, 1984; Dadam et al, 1984]. Research interest in the time modelling area has increased dramatically over the past decade as shown by published bibliographies [McKenzie, 1986] and comparative studies [Theodoulidis & Loucopoulos, 1991].

Without temporal support, many applications have been forced to manage temporal information in an ad-hoc manner. Recent research work attempts to overcome this problem from a number of different perspectives. The ERAE model [Dubois et al, 1986; Hagelstein, 1988] extends the semantics of the entity-relationship model with a distinguished type *Event* as one of its basic constructs. The ERAE approach considers only the requirements specification area without providing any integration to subsequent activities towards the implementation of a database system. Furthermore, the language is confined to simple objects and avoids the issue of modelling objects with component properties. The CML language uses an object-centered viewpoint and includes time as a primitive notion [Loki, 1986; Jarke, 1989]. This language has rich time semantics and the modelling of complex objects is implicitly defined in the object-centred framework adopted by the language. However, because it deals with two time dimensions and also caters for the modelling of relative temporal information, its efficient use in large database applications is debatable. Furthermore, it lacks a notation which is conducive to concept elicitation from and validation by end users.

The modelling of complex objects [Adiba, 1987] arises from the need to deal with applications which require the management of objects of arbitrary complexity. For example, in CAD/CAM or CASE applications one needs to be able to deal with objects that consist of a number of components and to reason for them whilst being able to deal with their components. Traditional data models fail to deal with this requirement; the structural constraints for example, of the relational model [Codd, 1970] force a developer to decompose the representation of a complex object into a set of relations. Extensions to the relational model include new types of attributes [Haskin, 1982] and the relaxation of the first normal form constraint [Abiteboul et al, 1989]. In both cases modelling of complex objects is carried out from a machine rather than a user-oriented perspective. In semantic modelling Dayal [1987] proposes an extension of DAPLEX [Shipman, 1981] as the means for dealing with complex objects. Whilst this approach is based on work which has the formally attractive feature of representing everything as an object, from a conceptual modelling perspective it is important to permit views which more naturally reflect a user's conception of the application domain.

The role of the CRL is twofold. Firstly, it is concerned with constraints placed upon the elements of ERT and with the derivation of new information based on existing information. Secondly, it is concerned

with the eligibility for the firing of operations and constraints placed on their order of execution. In this paper, only the first type of CRL expressions are discussed namely, the constraint rules and derivation rules. The constraint rules express restrictions on individual ERT states or state transitions. These rules are further classified to *static constraint rules* and *transition constraint rules* based on whether they refer to one state only or whether they refer to more than one states. The derivation rules are also classified to *transition derivation rules* and *static derivation rules* depending on whether the derived ERT components are time-varying or not.

The CRL formalism possesses a number of features in order to be used as part of the TEMPORA conceptual modelling formalism. These features are identified as follows:

- orthogonality and minimality of concepts i.e., controlled redundancy.
- ease of construction
- ease of communication
- well defined interface with level models

The CRL should have orthogonal concepts in two dimensions. Firstly, its own concepts should not have overlapping semantics in order to make clear distinction what it is used where and when. Secondly, its concepts must not express exactly the same things with the other two components of the requirements specification formalism in order to avoid redundancies and specificational overhead. Furthermore, its concepts and constructs must be sufficient to express all the intended knowledge. Limited redundancy is permitted only when it increases its understandability but not to the level that introduces ambiguity. In addition, CRL must be easily used and understood by everyone involved in the requirements specification process with the minimum of training and expertise. Finally, its interface with the other components of the conceptual modelling formalism must be well defined in order to obtain a strong synergetic effect between the different components of a requirements specification. Thus, the result will be an effectively integrated modelling formalism with the expressive power and the necessary structuring mechanisms.

Section 2 of the paper describes the basic formalism of the ERT model in terms of the basic concepts, the external graphical notation, the semantics of complex objects and the semantics of time. Section 3 introduces the CRL language in terms of the classification schema for



its expressions and exemplifies each kind. Section 4 concludes the paper by presenting the current status of the work and discussing future research directions.

## 2. The Entity-Relationship-Time Model

### 2.1 Basic Concepts and Externals

The orientation of the ERT model is the Entity-Relationship formalism which makes a clear distinction between objects and relationships. On this basis, the ERT model offers a number of features which are considered to be necessary for the modelling of complex database applications. Specifically, it accommodates the explicit modelling of time, taxonomic hierarchies and complex objects. The different aspects of data abstraction that are dealt with in the ERT model are: classification, generalisation, aggregation and grouping.

The most primitive concept in ERT is that of a *class* which is defined as a collection of individual objects that have common properties i.e., that are of the same type. In an ERT schema only classes of objects are specified. In addition, every relationship is viewed as a named set of two (entity or value, role) pairs where each role expresses the way that a specific entity or value is involved in a relationship. These two named roles are called *relationship involvements* and for completeness reasons, they are always required in an ERT schema. By using relationship involvements we have the possibility to express each relationship with two sentences which are syntactically different but semantically equivalent.

Time is introduced in ERT as a distinguished class called time period class. More specifically, each time varying simple entity class or complex entity class and each time varying relationship class is timestamped with a time period class. That is, a time period is assigned to every time varying piece of information that exists in an ERT schema.

The term time varying refers to pieces of information that the modeller wants to keep track of their evolution i.e. to keep their history and consequently, to be able to reason about them. For example, for each simple entity class or complex entity class, a time period might be associated which represents the period of time during which an entity is modelled. This is referred to as the *existence period* of an entity. The same argument applies also to relationships i.e., each time varying relationship might be associated with a time period

which represents the period during which the relationship is valid. This is referred to as the *validity period* of a relationship.

Besides the objects for which history is kept, another type of object, called *event* is also supported. These are objects that prevail for only one time unit and thus, the notion of history does not apply to them. Alternatively, one can say that these objects become history as soon as they occur. Events are denoted by defining the duration of their timestamp to be one time unit.

As a consequence of the adopted timestamping semantics, only the *event time* is modelled in ERT; i.e. the time that a particular piece of information models reality. At a first glance this might seem to be restrictive in the sense that the captured information is not semantically as rich. However, this assumption is considered to be necessary in order to keep the proposed approach manageable and to permit computational attractive algorithms for reasoning about time.

For each timestamp its granularity should be defined. The granularities however, should be carefully chosen as they affect the relative ordering of events stored in the database. For example, if DATE is the granularity of the SHIPMENT object, then two shipments received on the same date will be treated to have occurred at the same time even if their exact time of arrival differs.

Another distinguished class that is introduced in ERT is that of a complex object. The distinction between simple and complex objects is that simple objects are irreducible in the sense they cannot be decomposed into other objects and thus, they are capable of independent existence whereas a complex object is composed of two or more objects and thus, its existence might depend on the existence of its component objects. The relationship between a complex object and its component objects is modelled through the use of the IS\_PART\_OF relationship.

The ERT model accommodates explicitly generalisation/ specialisation hierarchies. This is done through a distinguished ISA relationship which has the usual set theoretic semantics. Furthermore, for each relationship involvement, a user supplied constraint rule must be defined which restricts the number of times an entity or value can participate in this involvement. This constraint is called *cardinality constraint* and it is applied to the instances of this relationship involvement by restricting its population.

Each of the simple entity classes and user defined relationship classes

The Entity Relationship Time Model and the Conceptual Rule Language

in an ERT schema can be specified as *derived*. This implies that its instances are not stored by default but they can be obtained dynamically i.e. when needed, using the *derivation rules*. For each such derivable component, there is exactly one corresponding derivation rule which defines the members of this entity class or the instances of this relationship class at any time. In addition, if the derivable component is not timestamped then the corresponding derivation rule instantiates this component at all times whereas if this component is time varying then the corresponding derivation rule obtains instances of this class together with its existence period or validity period.

In figure 1 an example ERT schema is given.

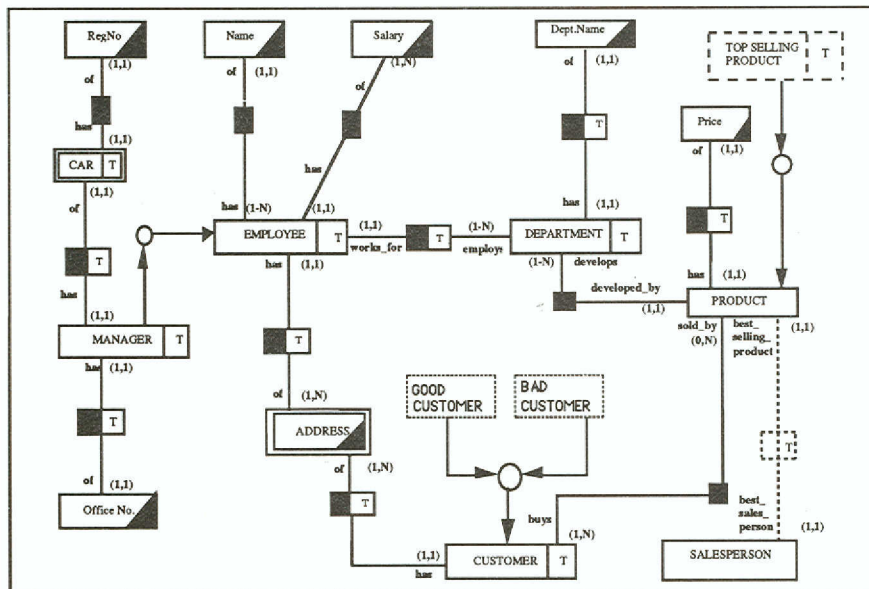


FIGURE 1: AN EXAMPLE ERT SCHEMA

As shown in this schema, entity classes e.g., EMPLOYEE are represented using rectangles with the addition of a 'time box' when they are time varying. Derived entity classes e.g., TOP\_SELLING\_PRODUCT, are represented as dashed rectangles. Value classes e.g., Name are also represented with rectangles but with a small black triangle at the bottom right corner to distinguish them from entity classes. Complex entity classes e.g., CAR and complex value classes e.g., ADDRESS are

represented using double rectangles. Relationship classes are represented using a small filled rectangle (e.g. the relationship class between MANAGER and CAR), whereas derived relationship classes are represented using a non filled dashed rectangle (e.g. the relationship between PRODUCT and SALESPERSON). In addition, relationship involvements and cardinality constraints are specified for each relationship class. The interpretation of these is for example “a MANAGER has one CAR and a CAR belongs to only one MANAGER” (the diagram shows minimum and maximum relationship involvements).

The notation of the ISA hierarchies e.g., MANAGER ISA EMPLOYEE is also given in figure 1. A distinction is made between different variations of ISA hierarchies. These are based on two constraints that are usually included in any ISA semantics namely, the *partial/total* ISA constraint and the *disjoint/overlapping* ISA constraint. It is assumed of course, that these constraints are applicable to hierarchies under the same specialisation criterion. These constraints are defined as follows:

- The *partial ISA constraint* states that there are members in the parent or generalised entity class that do not belong in any of its entity subclasses. On the other hand, the *total ISA constraint* states that there no members in the parent or generalised entity class that do not belong in any of its entity subclasses.
- The *overlapping ISA constraint* states that the subclasses of a given parent class under the same specialisation criterion are allowed to have common entities whereas the *disjoint ISA constraint* states that the subclasses of a given parent class under the same specialisation criterion are not allowed to have common entities.

The first of these constraints refers to the relationship between the parent class or generalised class and the child class(es) or specialised class(es). The second constraint refers to the relationship between child classes. Based on the above constraints the four kinds of ISA relationships are supported namely, Partial Disjoint ISA, Total Disjoint ISA, Partial Overlapping ISA and Total Overlapping ISA.

In summary, the components of ERT are defined as follows:

**Entity** is anything, concrete or abstract, uniquely identifiable and being of interest during a certain time period.

**Entity Class** is the collection of all the entities to which a specific definition and common properties apply at a specific time period.

**Relationship** is any permanent or temporary association between two entities or between an entity and a value.

**Relationship Class** is the collection of all the relationships to which a specific definition applies at a specific time period.

**Value** is a lexical object perceived individually, which is only of interest when it is associated with an entity. That is, values cannot exist in their own.

**Value Class** is the proposition establishing a domain of values.

**Time Period** is a pair of time points expressed at the same abstraction level.

**Time Period Class** is a collection of time periods.

**Complex Object** is a complex value or a complex entity. A complex entity is an abstraction (aggregation or grouping) of entities, relationships and values (complex or simple). A complex value is an abstraction (aggregation or grouping) of values (complex or simple).

**Complex Object Class** is a collection of complex objects. That is, it can be a complex entity or a complex value class.

In addition, the following axioms apply to the concept of a relationship class.

- 1 An entity can only participate in a relationship if this entity is already in the population of the entity class specified in the relationship. Furthermore, the validity period of the relationship should be a subperiod of the intersection of the existence periods of the two involved entities.
- 2 Each entity in a subclass population has also a reference in the population of its superclasses. In addition, the existence period of the specialised entity should be a subperiod of the existence period of the generalised entity.
- 3 If an entity belongs to a population of an entity class, it cannot also belong to the population of a value class at any time and vice-versa. Furthermore, any two entity classes which are not themselves subclasses of a third entity class and all have no common subclasses, must be disjoint at any

time point. Note that this definition does not prevent entities from moving between entity classes during their lifetime.

The graphical notation for the ERT is summarised in figure 2.

## 2.2 Time Semantics

In the approach described in this paper, time is introduced as a distinguished entity class. For example, each entity class can be timestamped in order to indicate that its history is of importance to the particular application. The same argument applies also to user defined relationship classes and IS\_PART\_OF relationships.

The 'time periods' approach was chosen as the most primitive temporal notion because it satisfies the following requirements [Villain, 1982; Ladkin, 1987]:

- Period representation allows for imprecision and uncertainty of information. For example, modelling that the activity of eating precedes the activity of drinking coffee can be easily represented with the temporal relation before between the two validity periods [Allen, 1983]. If one tries, however, to model this requirement by using the line of dates then a number of problems will arise since the exact start and ending times of the two activities are not known.
- Period representation allows one to vary the grain of reasoning. For example, one can at the same time reason about turtle movements in days and main memory access times in nanoseconds.

The formal framework employed as the temporal reasoning mechanism is that of *Interval Calculus* proposed in [Allen, 1983] and which was later refined in [Loki, 1986] but with the addition of a formal calendar system in order to provide for the modelling and reasoning about the usual calendar periods. In figure 3, the semantics of the adopted time model is defined using ERT notation.








ERT Graphical Notation	Explanation
	Entity class A and derived entity class A (dashed)
	Time stamped entity class B and time stamped derived entity class B (dashed). T is a symbolic time period.
	Complex entity class C and complex value class D.
	Simple value class E and derived value class F. May have relationships to nodes of type A, B, C and D
	Relationship (binary) that may connect nodes of type A, B, C or D. a and b are relationship names (b is inverse of a). m1 and m2 indicate mapping in the format (x:y) where x,y are non-negative integers, or N. Non-filled box indicates derived relationship.
	Time stamped binary relationships. T is a symbolic time period
	ISA relationships Filled box -> total, non-filled -> partial. Several arrows pointing to round box indicate disjoint subsets.

FIGURE 2: ERT EXTERNALS

The modelling of information using time periods takes place as follows. First, each time varying object (entity or relationship) of ERT is assigned an instance of the built-in class *SymbolPeriod*. Instances of this class are system-generated unique identifiers of time periods e.g. SP1, SP2, etc. Members of this class can relate to each other by one of the thirteen temporal relations between periods [Allen, 1983]. Instances of the *SymbolPeriod* class may be displayed in an ERT schema. If they are not included then a simple T in the time box indicates that the corresponding object is timevarying.

The two subclasses of the Time Period class are disjoint as indicated in figure 3. This is because symbol periods are used to model relative time information while calendar periods model absolute time information. Thus, both views are accommodated.

In figure 3, the symbol  $\tau$  represents a temporal relationship and the



symbol  $\tau$  its inverse. In addition, time periods start and end on a *tick* and also have a duration expressed in ticks. A tick is defined as smallest unit of time that is permitted to be referenced and it is usually the time unit *second*. The *CalendarPeriod* class has as instances all the conventional Gregorian calendar periods e.g., 10/3/1989, 21/6/1963, etc. Members of this class are also related to each other and to members of the Symbol Period class with one or more of the time period comparison predicates. The temporal relationships between calendar periods follow a formal calendar system [Theodoulidis, 1990] which is based on the work reported by Clifford and Rao in [Clifford & Rao, 1988] with the addition of the calendar unit 'week'. This was considered to be necessary since there is often reference to this unit depending on the particular application domain.

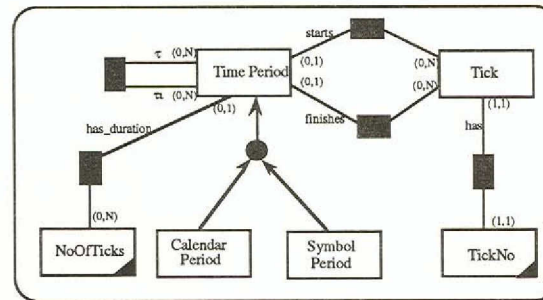


FIGURE 3: THE TIME METAMODEL

Any desirable calendar time unit can be defined as a combination of already defined calendar units. Thus, expressions like `END_OF_MONTH` and `NEXT_FORTNIGHT` are easily defined. The usual operators are provided including set operators and comparators. These are distinguished to those that are applied to elements of the same domain and those that are applied to elements of different domains [Clifford & Rao, 1988]. Additional operators like the time period comparison predicates are also provided together with functions that transform elements of one domain onto another. Note however, that the reasoning always takes place at the lower level of the ones involved.

Other notions of time such as *duration* and *periodic time* are also represented directly in the proposed formalism in addition to the above specified ones. As a consequence, the expressive power of the proposed formalism is increased and so does its usability. These

notions of time are expressed in the Conceptual Rule Language (CRL) as constraints upon the structural components and also as constraints on the behaviour of procedures.

The definition of the duration class is shown in figure 4. Members of this class are simple durations expressed in any abstraction level. Each duration consists of an amount of calendar time units expressed using real numbers and it is uniquely identified by the combination of its amount and abstraction level. For example, the duration '1,5 year' is a valid duration according to this definition.

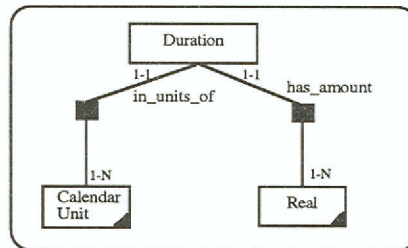


FIGURE 4: METAMODEL OF THE DURATION CLASS

The periodic time class is defined in figure 5. As seen in this figure, a periodic time has a base which is a calendar period, a duration and also it can be restricted by a two calendar periods which restrict the set of values for this periodic time. For example, the expression 'first week of each month during next year' is a valid definition of a periodic time according to the above definition. In this case the calendar period corresponds to "1-7 days", the duration corresponds to "1 month" and the restricting calendar period is the next year corresponding to [1/1/1991, 31/12/1991]. A periodic time is uniquely identified by the combination of its base and its duration.

### 2.3 Complex Object Semantics

Complex objects can be viewed from at least two different perspectives [Batini, 1988]. The first one is the representational perspective and focuses on how entities in the real world should be represented in the conceptual schema. This entails that objects may consist of several other objects arranged in some structure. Events in the real world are then mapped to operations on the corresponding objects. In contrast, if complex objects are not allowed, like e.g., in the relational model, then

information about the object is distributed and operations on the object are transformed to a series of associated operations. The second perspective is the methodological perspective which means that the complex object concept is regarded as a means of stepwise refinement for the schema and for hiding away details of the description. This in turn, implies that complex objects are merely treated as abbreviations that may be expanded when needed.

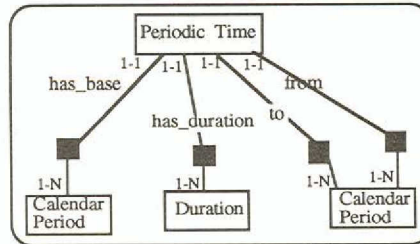


FIGURE 5: METAMODEL OF THE PERIODIC TIME CLASS

The basic motivation for the inclusion of the complex entity/value class in the external formalism, is to abstract away detail, which in a particular situation is not of interest. In addition, no distinction is made between *aggregation* and *grouping*, but rather a general composition mechanism is considered which also involves relationships/attributeships.

Graphically, composition is shown by surrounding the components with a rectangle representing the composite object class. The notation of a complex object in ERT is shown in figure 2. The complex value class ADDRESS and the complex entity class CAR of figure 1 may be viewed at a more detailed level as shown in figure 6 and figure 7 respectively.

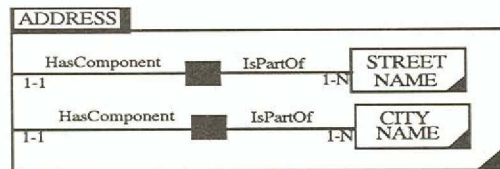


FIGURE 6: THE COMPLEX VALUE CLASS ADDRESS IN MORE DETAIL

The components of a complex object comprise one or more hierarchically arranged substructures. Each directly subordinate component entity class is part\_of-related to the complex entity class

border so that the relationship between the composite object and its components will be completely defined. Whether the *HasComponent* involvement is one of aggregation or grouping, it can be shown by means of the normal cardinality constraints. That is, if its cardinality is (0,1) or (1,1), the component is aggregate whereas if its cardinality is (0,N) or (1,N), the component is a set.

Most conceptual modelling formalisms which include complex objects [Kim et al, 1987; Lorie & Plouffe, 1983; Rabitti et al, 1988], model only *physical part hierarchies* i.e, hierarchies in which an object cannot be part of more than one object at the same time. In the ERT model, this notion is extended in order to be able to model also *logical part hierarchies* where the same component can be part of more than one complex object.

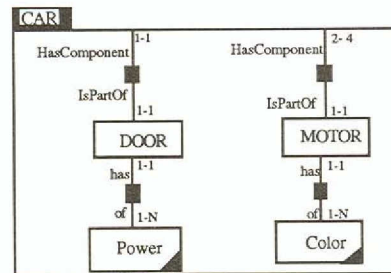


FIGURE 7: THE COMPLEX ENTITY CLASS CAR IN MORE DETAIL

To achieve this, four different kinds of IS\_PART\_OF relationships are defined according to two constraints, namely the *dependency* and *exclusiveness* constraints. The dependency constraint states that when a complex object ceases to exist, all its components also cease to exist (dependent composite reference) and the exclusiveness constraint states that a component object can be part of at most one complex object (exclusive composite reference). That is, the following kinds of IS\_PART\_OF variations [Kim et al, 1989] are accommodated:

- dependent exclusive composite reference
- independent exclusive composite reference
- dependent shared composite reference
- independent shared composite reference

Note that no specific notation is introduced for these constraints. Their interpretation comes from the cardinality constraints of the IS\_PART\_OF relationship. That is, assume that the cardinality of the

IS\_PART\_OF relationship is  $(\alpha, \beta)$ . Then,  $\alpha=0$  implies non dependency,  $\alpha \neq 0$  implies dependency,  $\beta=1$  implies exclusivity while  $\beta \neq 1$  implies shareness.

The following rules concerning complex objects should be observed:

- 1 Complex values may only have other values as their components. In addition, the corresponding IS\_PART\_OF relationship will always have dependency semantics unless it takes part in another relationship.
- 2 Complex entities may have both entities and values as their components. Every component entity must be IS\_PART\_OF-related to the complex entity.
- 3 Components, whether entities or values, may in turn be complex, thereby yielding a composition/decomposition hierarchy.

Timestamping in a time varying IS\_PART\_OF relationship is translated to the following constraints. The dependency constraint in a time varying IS\_PART\_OF relationship means that:

*The existence periods of the complex object and the component object should finish at the same time with the validity period of the IS\_PART\_OF relationship.*

Also, the exclusiveness constraint is translated to:

*If an object A is part of the complex objects B and C, then the period during which A is part of B should have an empty intersection with the period during which A is part of C.*

### 3. The Conceptual Rule Language

The CRL language expresses constraints on the ERT components in addition to those specified in the ERT formalism itself. Also, it expresses the derivation of new information based on existing information. This is necessary in order to define formulas for the derived ERT components which they can be used to obtain the values of these components when needed.

The proposed rule classification schema which is in accordance with the above identified pieces of information that need to be modelled, is shown in figure 8. As shown in this figure, the following different types of rules are distinguished:

- **Constraint rules** which are concerned with the integrity of the ERT components. They are further subdivided to *static constraint rules* which are expressions that must hold in every valid state of an ERT database and *transition constraint rules* which are expressions that define valid state transitions in an ERT database. An example of a static constraint rule might be 'The number of employees working in a department must be less than 100 at all times'. An example of a transition constraint rule might be 'The salary of an employee must never decrease'.
- **Derivation rules** which are expressions that define the derived components of the ERT model in terms of other ERT components including derived components. There must be exactly one derivation rule for each such component. As the constraint rules, derivations rules are also subdivided to *static derivation rules* and *transition derivation rules* depending on whether the derived ERT component is timestamped or not. An example of a static derivation rule might be 'A supplier is the cheapest supplier for a particular product if his offer for this product has the minimum price'. An example of a transition derivation rule might be 'A customer is the best customer of this month if the total amount of his orders placed this month is the maximum'.

### 3.1 Constraint Rules

As specified previously, constraint rules express restrictions on the ERT components by constraining individual ERT states and state transitions where a state is defined as the extension of the database at any tick. More specifically, static constraint rules apply to every state of the database and thus, they are time independent. In fact, they represent definitions of conditions which must hold between different classes (entity, value or relationship classes) in any individual state.

The purpose of a static constraint rule is to restrict each valid state of one or more items of data and it can be said to hold (or not hold) simply on the basis of the extension of the database with respect to a single state. These rules are also called *extensional constraints* [Clifford & Warren, 1983], *functional dependency rules* and *multivalued dependency rules* and their mapping to first order logic formulas is well defined [Nicolas, 1978; Nicolas & Gallaire, 1978].

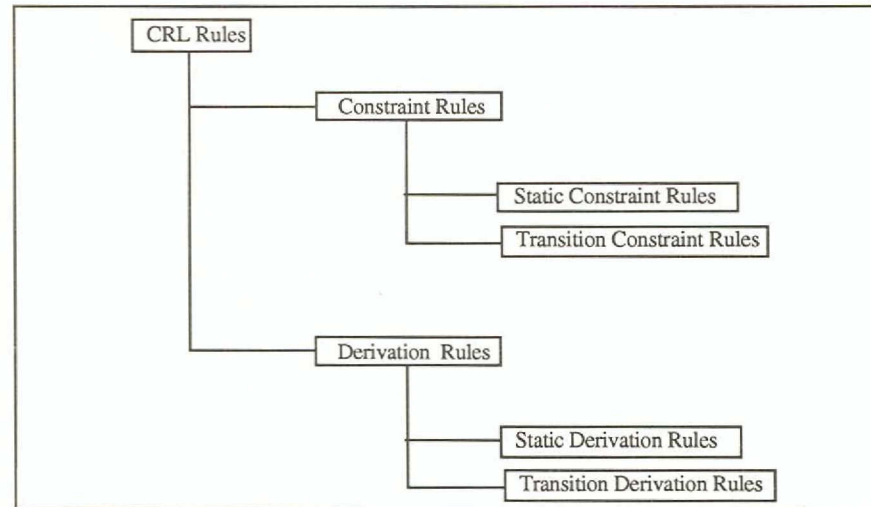


FIGURE 8: CLASSIFICATION OF CRL RULES

On the other side, transition constraint rules place restrictions on two or more states of the database by specifying valid state progressions. This type of rules is possible to express directly in the context of the proposed formalism because of the explicit modelling of the evolution of data. Each transition constraint rule is said to hold (or not hold) only by examining at least two states of the database. These rules are also

called *intensional constraints* [Clifford, 1983], *dynamic constraints* and *constraints upon update operations* [Smith & Smith, 1977; Nicolas & Yazdanian, 1978; Casanova & Bernstein, 1979].

### 3.1.1 Static Constraint Rules

#### 3.1.1.1 Classification of Static Constraint Rules

As stated previously, static constraint rules are used to describe each permissible state of the ERT and thus, they are logical restrictions on the data. A rule in this category always refers to a property of an entity class or a value class or a relationship class and it can be either true or false. The description of this property is done in terms of imperative or conditional statements which are called *state conditions* and refer to classes or instances of classes. In fact, static constraint rules preserve the integrity of the database by restricting the operations that can be applied in any individual state and must be true at any time period.

As mentioned previously, static constraint rules refer to properties of entity classes, value classes and relationship classes. This reference mode motivated the proposed classification schema for the static constraint rules seen in figure 9. According to this schema, the following types of static constraint rules are distinguished:

- **On entity classes.** These static constraint rules are concerned with restrictions on the number of instances of a particular entity class or the way that the instances of a specific entity class may be identified.
- **On value classes.** These static constraint rules are concerned with restrictions imposed on the domain of values of a particular value class.
- **On relationship classes.** These static constraint rules are concerned with restrictions on the number of instances of a single entity class that are involved in different relationships and also, with restrictions between sets of instances of the same entity class that are involved in one or more relationships.

Note that in the set of static constraint rules applied to an ERT schema, belong also constraints on ISA hierarchies and IS\_PART\_OF relationship classes. However, these constraints are shown as part of the graphical notation of the ERT model. The decision to do so reflects the opinion of the author as to which is the best balance point between expressing



everything in ERT notation and expressing everything in CRL notation.

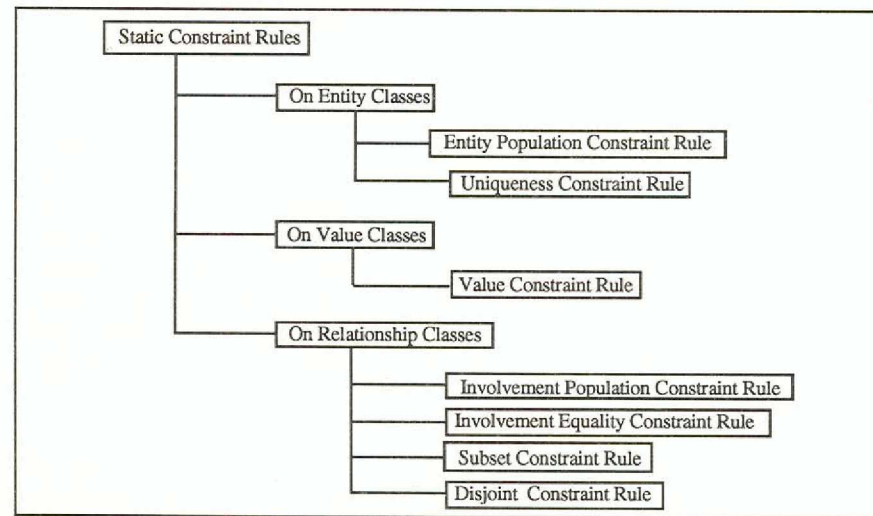


FIGURE 9: CLASSIFICATION OF STATIC CONSTRAINT RULES

The classification shown in figure 9, contains a further partitioning of the static constraint rules depending on whether they are applied on entity classes, value classes and relationship classes. These will be introduced explicitly in the following sections where each type of static constraint rules is introduced and exemplified.

### 3.1.1.2 Static Constraint Rules on Entity Classes

The static constraints rules on entity classes are CRL expressions which restrict the number of instances of a particular entity class or the way that the instances of a specific entity class may be identified. As seen in figure 9, two types of static constraint rules on entity classes are identified namely, the *entity population constraint rule* and the *uniqueness constraint rule*.

#### *Entity population constraint rule*

The entity population constraint rule is intended to denote the number of entity instances within a given entity class. Because entity population is always expressed as a non negative integer number, this constraint might state the minimum and maximum number of entity instances. The values between the **minimum and maximum integer** numbers are regarded as contiguous. Exact number of instances is

indicated by having the same minimum and maximum number.

An example of an entity population constraint rule is the following :

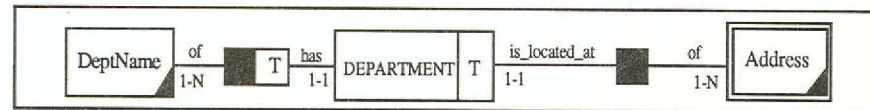
$$\text{Number\_Of}(\text{Department}) < 20 \text{ and } \geq 2$$

This constraint expresses a restriction on the number of instances of the entity class DEPARTMENT to be in the range [2,20).

**Uniqueness constraint rule**

The uniqueness constraint rule expresses more complex uniqueness constraints on a particular entity class than simple cardinality constraints by involving more than one relationships in which this entity class participates. In particular, this type of constraint states that a combination of involvements from different entity classes and value classes uniquely identifies the instances of a specific entity class.

Assume for example, the following ERT schema:



In this schema, the following uniqueness constraint rule might be defined:

**DEPARTMENT is identified by the DeptName that has**

**and the Address that is\_located\_at.**

This constraint states that each existing DEPARTMENT is uniquely identified by the combination of its name and its address.

**3.1.1.3 Static Constraint Rules on Value Classes**

The static constraints rules on value classes are CRL expressions which restrict in any possible way the instances of a particular value class in its involvement(s) with an entity class. No further classification schema is defined for these constraints because they are dependent on the particular domain of instances of the value class. Thus, only one generic type of these is defined which is called *value constraint rule*.

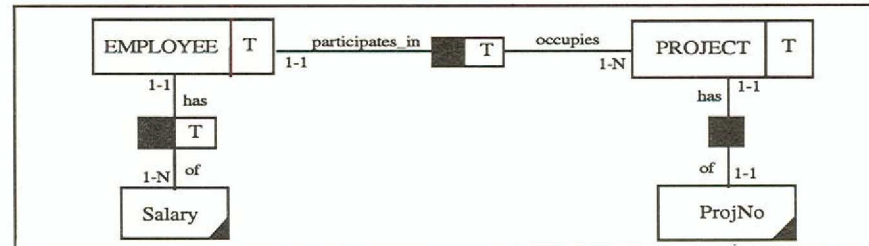
**Value constraint rule**

A value constraint rule restricts the values that can be drawn from a domain. It is applied to the involvement(s) of a value class with an entity class(es) because this combination provides the unique

identification of a domain of values.

Besides the value domain identification, a value constraint rule must have a comparison part which includes comparison operator(s) and corresponding value expression(s) drawn from the same domain.

Consider the following ERT schema:



An example value constraint rule for this schema might state that each employee who participates in the project with project number E2469 must have salary in the range (£9,000, £15,000). This rule can be formulated in CRL as follows:

```
(Salary that_is of EMPLOYEE who participates_in PROJECT
                                which has ProjNo=E2469)
    > 9000 and < 15000.
```

As seen from the above example, the value constraint rule is applied to the 'Salary of EMPLOYEE' involvement which is further restricted with the involvements 'EMPLOYEE participates\_in PROJECT' and 'PROJECT has ProjNo=E2469'. This whole expression provides the unique identification of the domain of values for the salary of an employee. In the remainder of the rule, two comparison expressions on this domain are given namely '>9000' and '<15000' assuming of course, that the domain of salary is non negative integers.

#### 3.1.1.4 Static Constraint Rules on Relationship Classes

The static constraint rules on relationship classes restrict the number of instances of a single entity class that are involved in different relationships and also, they express restricting conditions between sets of instances of the same entity class that are involved in one or more relationships.

As seen in figure 9, four types of static constraint rules on relationship classes are identified namely, the *involvement population constraint*

rule, the *involvement equality constraint rule*, the *subset constraint rule* and the *disjoint constraint rule*.

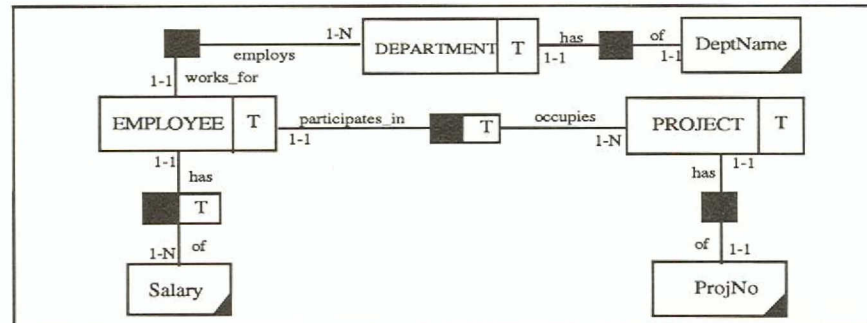
***Involvement population constraint rule***

The involvement population constraint rule restricts the minimum and maximum number of times that an instance of an entity class can participate in one or more involvements. In its simplest form, this type of rule restricts the number of times that an instance of an entity class can relate with the same instance of one another entity class or value class. For example, the rule

**Number\_of(EMPLOYEE who participates\_in the same PROJECT) < 15**

restricts the number of employees for the same project to be always less than fifteen. In this form, the involvement population constraint rule is a specialisation of the cardinality constraints placed on the relationship class between the entity classes employee and project.

However, these rules can also express more complex population constraints involving set expressions which restrict the number of times that an entity class can participate in more than one involvements. For example, consider the following ERT schema:



A valid involvement population constraint rule for this schema could be the following:

**Number\_of(EMPLOYEE who has Salary > 11000  
and who participates\_in PROJECT  
that has ProjNo=E2469  
and who works\_for DEPARTMENT  
that has  
DeptName='Computation')  
> 2 and < Number\_of(EMPLOYEE who works\_for DEPARTMENT**

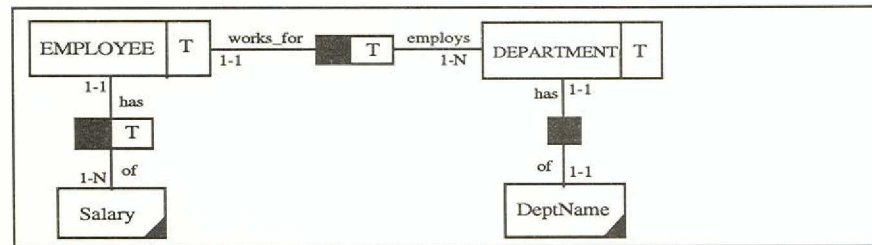
that has

DeptName='Computation').

As seen in the above constraint, more complex expressions can also be used to identify the set of instances of the EMPLOYEE entity class and, to express the maximum number of the permitted population of this set.

***Involvement equality constraint rule***

The involvement equality constraint rule specifies that the set of instances of an entity class participating in one or more involvements is equal to the set of instances of the same entity class participating in a number of different involvements. For example, consider the following schema:



A valid involvement equality constraint rule for this schema might be:

EMPLOYEE **who** works\_for DEPARTMENT

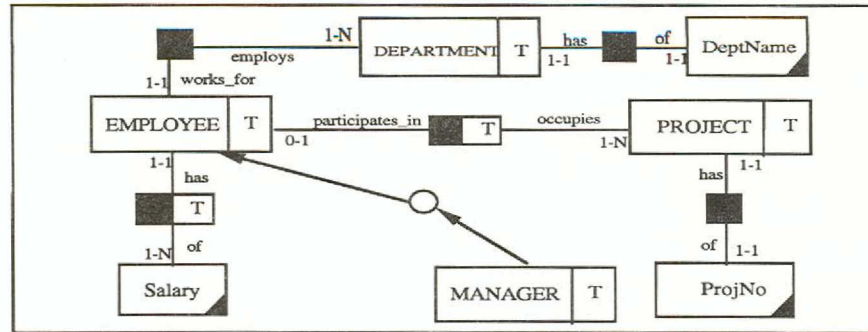
**is equal to**

EMPLOYEE **who** has Salary

This rule states that all the employees who work for a department must also get paid.

***Subset constraint rule***

The subset constraint rule specifies that the set of instances of an entity class participating in one or more involvements is a subset of the set of instances of the same entity class participating in a number of other involvements with the same or different entity (or value) class. Consider for example, the following ERT schema:



A valid subset constraint rule for this schema might be:

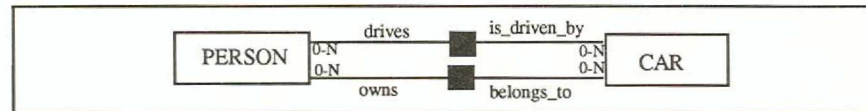
**EMPLOYEE who has Salary > (Salary that\_is of MANAGER who works\_for DEPARTMENT.T) - 3000 and who participates\_in PROJECT**

**is subset of**

**EMPLOYEE who works\_for DEPARTMENT.T**

This rule expresses the constraint that the set of employees who have salary greater than the salary of their manager minus £3,000, and who participate in some project, is subset of the set of employees who work for the same department as that of their manager.

A specific class of these rules refer to involvements with the same entity (or value) class. For example, consider the following ERT schema:



A valid subset constraint rule for this schema might be:

**PERSON who owns CAR.T**

**is subset of**

**PERSON who drives CAR.T**

This rule specifies that the set of persons who own a specific car is subset of the persons who drive the same car. Note the use of

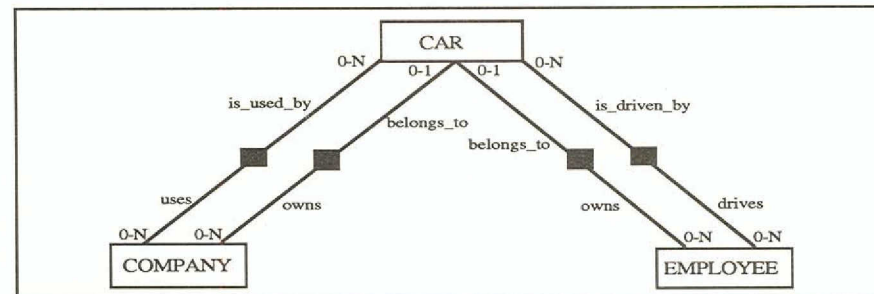
instantiation for the class CAR which provides the link between the two expressions by indicating that both must refer to the same instance.

**Disjoint constraint rule**

The disjoint constraint rule specifies that a set of instances of an entity class are disjoint of another set of instances of the same entity class. The set of instances in both the expressions is of course, defined through a number of involvements.

In fact, this constraint specifies that the set of instances of an entity class participating in a number of involvements with certain instances of other entity (or value) classes, is disjoint from the set of instances of the same entity class participating in the same number of involvements with the same instances of the above classes.

For example, consider the following ERT schema:



According to this schema, the following rule is a valid disjoint constraint :

**CAR which is\_driven\_by EMPLOYEE.A**  
**and which belongs\_to COMPANY.B**  
**is disjoint from**  
**CAR which belongs\_to EMPLOYEE.A**  
**and which is\_used\_by COMPANY.B**

This rule states that the set of cars which belong to a company and which are driven by an employee are differently from the cars that belong to the same employee and used by the same company.

This completes the introduction of the static constraint rules. From the above, it can be concluded that this set of rules expresses the

possible constraints on the structural components of the proposed specification formalism in a well defined and classified framework.

### 3.1.2 Transition Constraint Rules

Transition constraint rules place restrictions on two or more states of the database by specifying valid state progressions. It is possible to express directly in the context of the CRL formalism this type of rules because of the explicit modelling of the evolution of data. Each transition constraint rule is said to hold (or not hold) only by examining at least two states of the database.

No detailed classification schema for these is proposed as the one proposed for the static constraint rules. The reason for this is that it is difficult to obtain a useful classification of expressions that refer to the evolution of data. However, transition constraint rules can also be seen as applicable to entity classes, relationship classes and value classes and this can be used to define acquisition heuristics for these.

An example of a transition constraint rule, is the classic mythical rule 'salaries of employees never decrease'. This can be formulated in CRL as follows:

EMPLOYEE **that** has Salary.S1 **at** T1  
                  **and that** has Salary.S2 **at** T2  
**and** T1 starts\_after T2  
**and** S1 > S2.

Another example of a transition constraint rule is the following:

PRODUCT **that** is reordered with Quantity.Q1 **at** T1  
                  **and that** is reordered with Quantity.Q2 **at** T2  
**and** T1 after T2  
**and**  $Q1 \leq 0.2 * Q2 + Q2$ .

The above rule states that when ordering products, the reordered quantity should not exceed by more than 20% the previous reordered quantity for the same product. The keyword *at* is used to denote the validity period of a relationship whereas the keyword *exists\_at* is used to denote the existence period of an entity. In addition, a rich set of time period comparison predicates is provided in order to help the modelling of relative time expressions in a natural way.



### 3.2 Derivation Rules

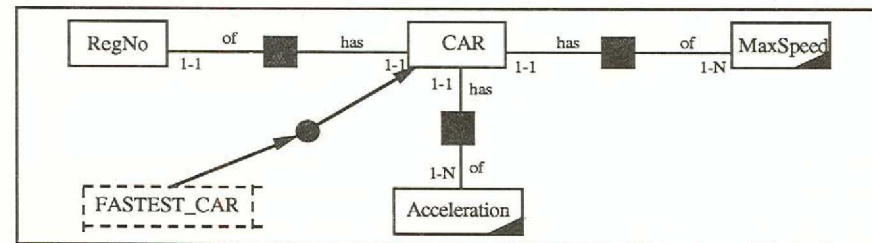
As defined earlier, the derivation rules are expressions that define the derived components of the ERT model in terms of other ERT components including derived components. These rules are further subdivided to *static derivation rules* and *transition derivation rules* depending on whether the derived ERT component is timestamped or not.

Derivation rules are introduced as a means of capturing structural domain knowledge that need not be stored and that its value can be derived dynamically using existing or other derived information. Note that recursive definitions of derived ERT components is not supported.

#### 3.2.1 Static Derivation Rules

The static derivation rules are defined as formulas that derive instances of entity classes or relationship classes which are not included in an ERT schema in terms of other ERT components. The purpose of this kind of rules is to specify a way by which one can obtain the value of an ERT component when needed, instead of having it explicitly stored. Moreover, these rules are valid in every state of the database i.e., at all times.

For example, consider the following ERT schema:



According to this schema, the derived entity class FASTEST\_CAR could be defined as follows:

### FASTEST\_CAR

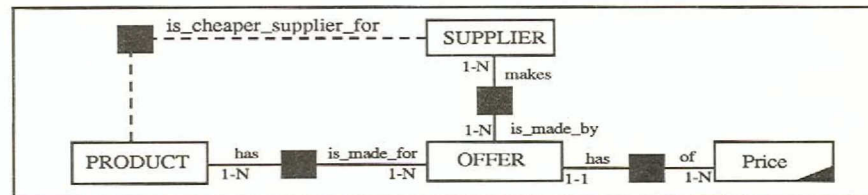
**is defined as**

CAR.C such that CAR.C that has MaxSpeed  $\geq$  **maximum**(MaxSpeed that\_is of CAR)  
**and** CAR.C that has Acceleration  $\leq$  **minimum**(Acceleration that\_is of CAR).

In the above example, the instances of the derived entity class FASTEST\_CAR are defined from instances of the entity class CAR as the cars that have the maximum possible speed and the minimum acceleration.

Note that in the above ERT schema, the derived entity class FASTEST\_CAR is connected to the entity class CAR using an ISA link. Even if this link is the same as it is the one between non derived entity classes, its semantic interpretation is different. That is, in the case of non derived entity classes this link is not explicitly stored but it is maintained through the use of the derivation formula.

Besides entity classes, relationship classes can also be derived. For example, in the following ERT schema the derived relationship class *is\_cheapest\_supplier\_for* is defined between the entity classes SUPPLIER and PRODUCT.



The derivation formula for this relationship class could be defined as follows:

SUPPLIER.S is\_cheapest\_supplier\_for PRODUCT.P

**is derived as**

SUPPLIER.S **who** makes OFFER.C **that** is\_made\_for PRODUCT.P  
**and** Price **that\_is** of OFFER.C = **minimum**(Price **that\_is** of OFFER  
**which** is\_made\_for PRODUCT.P).

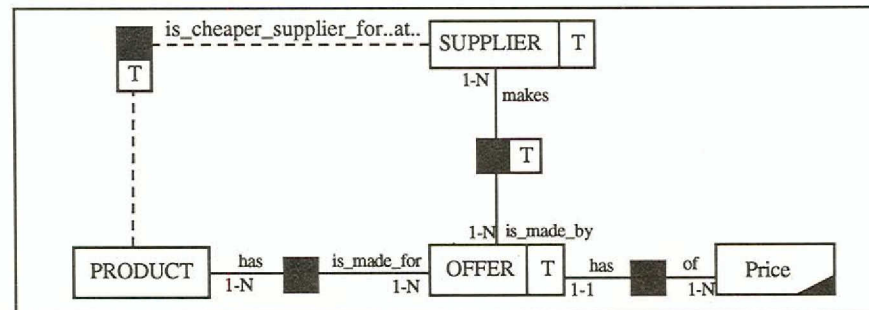
This means that for a specific product the above derivation formula will give as result a set of suppliers whom offer for this product has the minimum price compared to the prices of the offers made in total.

Note that for this relationship class no cardinality constraints are defined. This is not necessary because these are derivable through the corresponding derivation formula.

### 3.2.2 Transition Derivation Rules

The transition derivation rules define how the instances of a time varying derived ERT component can be obtained by utilising historical information stored in the database. Since the derived object (entity or relationship) class is timestamped, this formula must use the corresponding time period as a constraining period.

Assume for example, the following ERT schema:



A valid transition derivation rule for the  
*is\_cheapest\_supplier\_for..at..*  
 relationship class could be the following:

SUPPLIER.S *is\_cheapest\_supplier\_for* PRODUCT.P at T

**is derived as**

SUPPLIER.S **that** makes OFFER.C at T1 **that** *is\_made\_for*  
 PRODUCT.P

**and** T1 during T

**and** Price.**that\_is** of OFFER.C=**minimum**(Price **that\_is** of OFFER

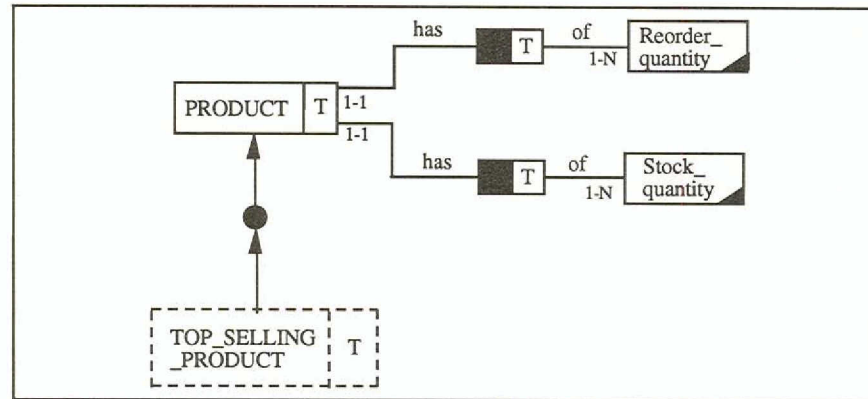
**which** *is\_made\_by* SUPPLIER at T2

**and which** *is\_made\_for* PRODUCT.P

**and** T2 during T),

This rule states that the cheapest supplier for a product P at a time

period T is the one whose offer has the smallest price during the same time period. Transition derivation rules can also be defined for derived entity classes. For example, consider the following ERT schema:



The derivation rule for the TOP\_SELLING\_PRODUCT entity class can be defined as follows:

PRODUCT.P isa top\_selling\_product at last\_month

**is derived as**

PRODUCT.P **that** has Stock\_quantity=

**maximum**(Stock\_quantity.X1 - Stock\_quantity.X1 **such that**

Stock\_quantity.X1=Stock\_quantity.S1 of PRODUCT.P at first\_day\_of\_last\_month +

**sum**(Reorder\_quantity.S2 of PRODUCT at T **and** T during last\_month)

**and** Stock\_quantity.X2 = Stock\_quantity.S2 of PRODUCT.P at last\_day\_of\_last\_month).

This rule states that the top selling product for last month can be derived as the one which sold best during last month. In order to compute the quantity sold, the stock quantity at the last day of the last month is subtracted from the sum of the stock quantity at the first day of the last month and the total reordered quantity of this product for the last month.

Exactly one transition derivation rule can be specified for each time varying derived ERT component and in addition, as it can be seen from the above examples, the transition derivation rules refer always to the timestamp of the corresponding entity class.

The derived relationship class example uses the time stamp as a variable which needs to be instantiated in order to obtain the cheapest supplier of the last month or the last year. On the contrary, the derived entity class example obtains the top selling product for the last month only. Thus, if one wants to find the top selling product of the last year then a different derivation rule can be defined or a new derived entity class should be specified for this piece of information. This is consequence of the assumption that only one derivation rule should be defined for each timevarying derived ERT component.

Consequently, irrespective of the approach followed when defining a derivation rule, the constraining time period should always refer to the timestamp periods of the involved entity classes and relationship classes.

## 4. Conclusions

The main aim of any conceptual modelling approach is to capture knowledge about the universe of discourse and represent it in such a way so as to enable a system developer to reason about this knowledge, communicate this understanding to end users for validation and specify the allowable structures of and transitions on the information base.

This paper discussed the ERT model and the CRL language, developed in the context of the TEMPORA conceptual framework, and argued that these models provide the necessary and desirable properties for the development of database systems which, as well as dealing with traditional applications, require to deal with applications which require the explicit modelling of time and complex objects.

The CRL formalism is introduced as a means to capture application domain knowledge that crosses between the structural part and the behavioral part of the specification. Thus, the role of the CRL formalism is concerned with constraints placed upon the elements of ERT and with the derivation of new information based on existing information.

Within the CRL formalism different rule types are distinguished in order to achieve orthogonality of concepts, well defined interface with level models, more assistance in the rule elicitation process and a unified requirements specification formalism. Moreover, a textual layout with natural language semantics was chosen for the CRL language in order increase its understandability and usability.

In the context of TEMPORA, the process of information systems development is viewed as a sequence of model-building activities which require appropriate mechanisms for 'knowledge elicitation', 'knowledge representation' and 'knowledge validation' about the modelled application domain and their mapping onto corresponding design and implementation structures. Current work relating to the issues discussed in this paper is concerned with the development of CASE tools to support the design process, developing mapping tools between the conceptual and executable levels and testing the feasibility of the paradigm on large scale industrial applications.

## References

- [Abiteboul et al, 1989] Abiteboul, S., Fischer, P.C., Schek, H.-J. (eds) *Nested Relations and Complex Objects in Databases*, Lecture Notes in Computer Science #361, Springer-Verlag, 1989.
- [Adiba, 1987] Adiba, M.E. *Modelling Complex Objects for Multimedia Databases*, in Entity-Relationship Approach: Ten Years of Experience in Information Modelling, S. Spaccapietra (ed), North-Holland, 1987.
- [Ahn & Snodgrass, 1988] Ahn I., Snodgrass R., *Partitioned Storage for Temporal Databases* *Information Systems*, 13(4), 1988.
- [Ariav & Clifford, 1984] Ariav G., Clifford J., *A System Architecture for Temporally Oriented Data Management*, Proceedings of the 5th International Conference on Information Systems, Tucson Arizona, Nov.1984.
- [Allen, 1983] Allen J.F. *Maintaining Knowledge about Temporal Intervals* *CACM*, 26(11) Nov.1983.
- [Batini, 1988] Batini, C. and Di Battiste G. *A Methodology for Conceptual Documentation and Maintenance*, *Information Systems*, 13(3), pp.297-318, April 1988.
- [Ben-Zvi, 1982] Ben-Zvi J., *The Time Relational Model*, PhD Dissertation, Univ. of California, L.A., 1982.
- [Casanova, 1984] Casanova M.A., Amaral de Sa J.E. *Mapping Uninterpreted Schemes into Entity-Relationship Diagrams: two Applications to Conceptual Schema Design*, *IBM Journal of Research and Development* 28(1) pp. 82-94, 1984.
- [Chen, 1976] Chen P.P.-C. *The Entity-Relationship Model-Toward a Unified View of Data* *ACM TODS* vol.1 no.1, pp.9-36, March 1976.
- [Clifford & Rao, 1988] Clifford J., Rao A., *A Simple, General Structure for Temporal Domains*, Proceedings of the IFIP 8/WG8.1 Working Conference on Temporal Aspects in Information Systems, Sophia Antipolis, France, May 1987.
- [Codd, 1970] Codd, E.F. *A Relational Model of Data for Large Shared Data Banks*, *CACM*, 13(6), June 1970.
- [Dadam et al, 1984] Dadam P., Lum V., Werner H.D., *Integration of time versions into a relational database system*, *Proc. VLDB*, Singapore, 1984.
- [Dayal, 1987] Dayal, U. *Simplifying Complex Objects: The PROBE Approach to Modelling and Querying them*, Proc. GI Conference Datenbanksysteme in Büro, Technik und Wissenschaft, Darmstadt, april 1987.

- [De Troyer, 1987] De Troyer O., Meersman R. *Transforming Conceptual Schema Semantics to Relational Data Applications*, Information Modelling and Database Management, Ed. Kangassallo H., Springer Verlag, 1987.
- [Dubois et al, 1986] Dubois E., Hagelstein J., Lahou E. et al *The ERAE Model : A Case Study* in Information Systems Design Methodologies: Improving the Practice, T.W Olle, H.G. Sol and A.A. Verrijn-Stuart (eds), North-Holland, 1986.
- [Hagelstein, 1988] Hagelstein, J. *Declarative Approach to Information Systems Requirements Modelling*, Knowledge-Based Systems, 1(4), Sept 1988.
- [Haskin, 1982] Haskin, R.L., Lorie, R.A. *On Extending the Functions of a Relational Database System*, Proc. ACM SIGMOD Conference, Orlando, 1982.
- [Jarke, 1989] Jarke, M. *The DAIDA Demonstrator: Development Assistance for Database Applications*, ESPRIT Conference Proceedings, 1989.
- [Kent, 1979] Kent W. *Limitations of Record-Based Information Models*, TODS, 1979.
- [Khoshafian, 1986] Khoshafian S. N., Copeland G. P. *Object Identity*, Proceedings of first International Conference on OOPSLA, Portland, Oregon, October 1986.
- [Kim et al, 1987] Kim W., Banerjee J., Chou H.T., Garza J.F., Woelk D. *Composite Object Support in Object-Oriented Database Systems*, in Proc. 2nd Int. Conf. on Object-Oriented Programming Systems, Languages and Applications, Orlando, Florida, Oct. 1987.
- [Kim et al, 1989] Kim W., Bertino E., Garza J.F. *Composite Objects Revisited*, SIGMOD RECORD 18(2), June 1989.
- [Ladkin, 1987] Ladkin, P. *Logical Time Pieces*, AI Expert, Aug, 1987, pp.58-67.
- [Loki, 1986] ESPRIT P107- LOKI, *A Logic Oriented Approach to Knowledge and Databases Supporting Natural Language User Interfaces* Institute of Computer Science, Research Center of Crete, Greece, March 1986.
- [Lorie & Plouffe, 1983] Lorie R., Plouffe W. *Complex Objects and Their Use in Design Transactions*, in Proc. Databases for Engineering Applications, Database Week 1983 (ACM), San Jose, Calif., May 1983.
- [Loucopoulos, 1989] Loucopoulos, P. *The RUBRIC Project-Integrating E-R, Object and Rule-based Paradigms*, Workshop session on Design Paradigms, European Conference on Object Oriented Programming (ECOOP), 10-13 July 1989, Nottingham, U.K.



- [Loucopoulos et al, 1990] Loucopoulos, P., McBrien, P., Persson, U., Schumaker, F., Vasey, P, TEMPORA - *Integrating Database Technology, Rule Based Systems and Temporal Reasoning for Effective Software*, In Proceedings of ESPRIT '90 Conference, Brussels, November 1990.
- [Loucopoulos et al, 1991] P. Loucopoulos , B. Wangler , P. McBrien , F. Schumacker , B. Theodoulidis and V. Kopanas, *Integrating Database Technology, Rule Based Systems and Temporal Reasoning for Effective Information Systems: The TEMPORA Paradigm*, Information Systems Journal, Vol.1, No 1, April 1991.
- [Lum et al, 1984] Lum V., Dadam P., Erbe R., Guenauer J., Pistor P., *Design of an integrated DBMS to support advanced applications*, Proc. Conf. Foundation Data Organization, Kyoto, Japan, 1985.
- [McBrien et al, 1991] McBrien P., Niezette M., Pantazis D., Seltveit A-H., Sundin, U., Tziallas G. and Theodoulidis, C. *A Rule Language to Capture and Model Business Policy Specifications*, 3rd Nordic Conference on Advanced Information Systems Engineering (CAiSE'91), Trondheim, Norway, 1991.
- [McKenzie, 1986] McKenzie E., *Bibliography : Temporal Databases*, ACM SIGMOD, Vol.15, No.4, December 1986.
- [Nijssen, 1988] Nijssen G.M., Duke D.J., Twine S.M. *The Entity-Relationship Data Model Considered Harmful*, 6th Symposium on Empirical Foundations of Information and Software Sciences, Atlanta, Georgia (USA), October 1988.
- [Rabitti et al, 1988] Rabitti F., Woelk D., Kin W. *A Model of Authorization for Object-Oriented and Semantic Databases*, in Proc. Int. Conf. on Extending Database Technology, Venice, Italy, March 1988.
- [Shipman, 1981] Shipman, D. *The Functional Data Model and The Data Language DAPLEX*, ACM TODS 6(1), March 1981.
- [Shoval, 1987] Shoval P., Even-Chaime M. *ADDS: A Systems for Automatic Database Schema Design Based on the Binary-Relationship Model*, Data and Knowledge Engineering 2(2), North Holland, 1987.
- [Theodoulidis et al, 1990] Theodoulidis, C., Wangler, B. and Loucopoulos, P. *Requirements Specification in TEMPORA*, 2nd Nordic Conference on Advanced Information Systems Engineering (CAiSE90), Kista, Sweden, 1990.
- [Theodoulidis, 1990] Theodoulidis, C. *A Declarative Specification Language for Temporal Database Applications*, PhD Thesis, UMIST, 1990.
- [Theodoulidis & Loucopoulos, 1991] Theodoulidis, C. and Loucopoulos, P. *The Time Dimesion in Conceptual Modelling*, Information Systems, 16(3), 1991.
- [Tsang, 1987] Tsang E.P.K *The Consistent Labelling Problem in Temporal Reasoning*, AAAI-87, Seattle, Washington, 1987.

- [Ullman, 1988] Ullman J.D. *Principles of Database and Knowledge Base Systems*, Pitman, 1988
- [Van Assche et al, 1988] Van Assche, F., Layzell, P.J., Loucopoulos, P., Speltincx, G., *Information Systems Development: A Rule-Based Approach*, *Journal of Knowledge Based Systems*, September, 1988, pp. 227-234.
- [Villain, 1982] Villain M.B. *A System for Reasoning about Time* Proceedings of AAAI-82, Pittsburgh, Pa., Aug.1982.
- [Villain, 1986] Villain M.B., Kautz H. *Constraint Propagation Algorithms for Temporal Reasoning* Proc. of AAAI-86, 1986.
- [Wiederhold et al, 1975] Wiederhold G., Fries J.F., Weyl S., *Structured Organization of Clinical Databases*, in Proceedings of the NCC, AFIPS Press, Montvale, New Jersey, 1975.