LOGEION
Filosofia da Informação

ARTIGO

DOI: https://doi.org/10.21728/logeion.2022v8n2.p130-150

Submetido em: 13/10/2021 **Aceito em:** 11/03/2022 **Publicado em:** 18/03/2022

TAXONOMY OF THINGS FOR THE DATAISM

Ulrich Schiel[1]
Universidade Federal de Campina Grande (UFCG)
uschiel@gmail.com

Abstract

There is an unmistakable actual trend to computerize more and more everyday activities in human society. AI tools, decision making and machine learning are some of the used technologies. In this paper we analyses how conventional modelling of database applications must evolve to meet these requirements. Adequate structural things and happenings are considered and, for each concept, adequate rules must be determined to ensure the integrity of the whole system. Especially the requirements of complex active and reactive systems, such as autonomy, space and time aspects as well as inaccurate information, must be taken into account.

Keywords: Conceptual modeling. Taxonomy. Ontology. Dataism.

taxonomia de coisas para o dataismo

Resumo

Existe uma tendência evidente de computadorizar cada vez mais atividades do dia-a-dia. Ferramentas de IA, de tomada de decisão e aprendizado de máquina são algumas destas tecnologias. Neste artigo analisamos como a modelagem de aplicações de bancos de dados deve evoluir para atender a estes requisitos. Coisas estruturais e ocorrêncais adequadas são consideradas e, para cada conceito, têm que ser definidas regras que garantem a integridade do sistema como um todo. Caracteristicas específicas de sistemas complexos ativos e/ou reativos, questões de autonomia, aspectos espaciais e temporais, assim como informações imprecisas, devem ser levados em consideração.

Palavras-chave: Modelagem conceitual. Taxonomia. Ontolgia. Dataismo.

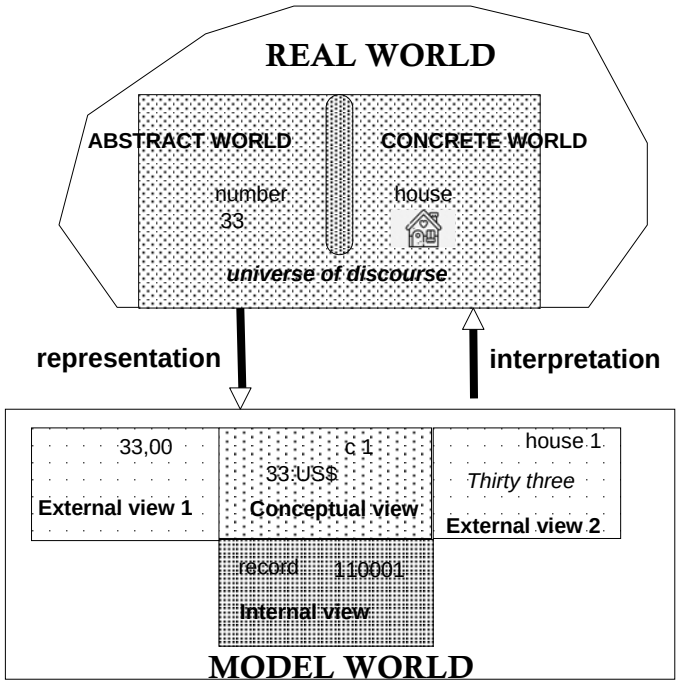
[1] Professor Titular do Departamento de Sistemas e Computação da Universidade Federal da Paraíba, na área de sistemas de informação e banco de dados. Bacharel em matemática pela Universidade Mackenzie/SP, mestre em informática pela PUC-Rio e Dr. rer. nat. pela Universidade de Stuttgart/Alemanha. Atividades de pesquisa em bancos de dados temporais, bancos de dados textuais e metodologias de projeto de sistemas de informação.

1 INTRODUCTION

When we create a Computerized Information System (CIS), we create a model of a small piece of the real world, called **Universe of Discourse (UoD)**, and represent things and happenings as data and processes in this model. On demand, the data are interpreted back for the user. Things of the UoD may be concrete, such as persons, products, a rain or a murder, and abstract, such as number, a price, a conviction or a congress. Therefore, we have the schema presented in Fig. 1.

The Model World encompasses several distinct views. The conceptual view is an integrated view of the whole Universe of Discourse. For distinct categories of users a specific external view should be adequate. The processing environment of all data is the internal view. The main interest of our taxonomy is the conceptual view, also known as **Universe of Discourse Description (UDD)**. This denomination is not completely adequate, as many activities in the real world are carried out in the model world.

Figure. 1. The Three-World view



With the tendency to pass more and more tasks to the computer, the model world must have the capacity to properly understand things and happenings and behave properly. The evolution of the Model World to realize more and more activities of the Real World is known as Dataism (HARARI, 2016). It is the tendency to leave to the computer the detection of

problems and to solve them. This requires the creation of increasingly sophisticated models, well suited to work independently.

The creation of models for the representation and storage of data remains to 3000 BC, where the Sumers created the written language. Through the existence of a written language, many things or facts of the world are retained in an adequate media for later use. Using a computer, data are not only retained but also processed and transformed, generating new data. Therefore a data processing system includes a static part (the data) and a dynamic (the processes).

In this paper we investigate what kind of things and happenings a computerized system can require in order to fulfill its objectives, and how they are represented and processed in the model world. Wieringa (2003) distinguish between two categories of systems: Transformational Systems and Reactive Systems. A transformational system has a well defined unique objective which generates an adequate output for a given input. On the other hand, a reactive system is a complex aggregate of multiple interdependent functionalities, attending various kinds of users. The realization of an external or internal request depends on the actual state of the system and eventually the interaction with other users or modules of the system.

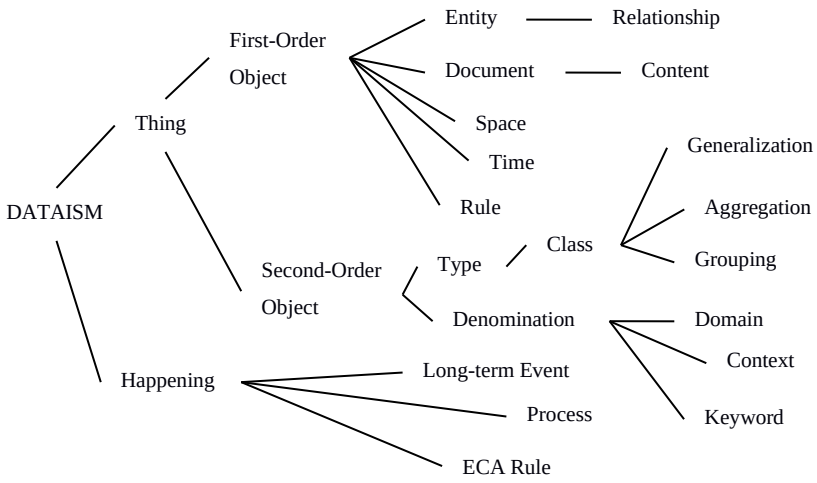
Hirschheim, Klein and Lyytinen (2008) distinguish between Fact-Based Systems and Rule-Based Systems. The former focuses on structures as entities, facts and conceptual schema, whereas the latter on deduction rules, as for legal or health applications.

The trend of sophisticated reactive systems is to integrate facts and rule and to become less open, performing most activities due to some internal event state, without external intervention.

Things and happenings of the real world may be of two natures: concrete or objective and abstract or subjective. Objective things are those of the physical world, such as a person, a city, a book, a rain, a sound. Subjective entities are immaterial. They can be associated to a single person, such as pain and expectation, or be a common sense, also called Transsubjective (HARARI, 2016), such as a company, a family, a number, a congress, a tennis game, the rules of tennis playing. Definitely, the construction of a conceptual model is determined by what we know about the Universe of Discourse. If our knowledge is incomplete or imprecise this partial knowledge must be kept as it is, in order to avoid the loss of this information.

In Figure 2 we show the main concepts of the taxonomy to be presented in this paper.

Figure 2 - Taxonomy of Dataism



The distinction between the concrete or abstract nature of things becomes irrelevant when they are represented in the system. Both are converted to a linguistic representation stored as data in a database.

The next chapter details the structure of static things and how they are interrelated. The subsequent chapter examines the dynamic happenings.

2 THINGS

As *things* we mean the static elements of the world, which can be described and are distinguishable from one another. This includes concrete and abstract things, called **first-order objects**, and descriptive concepts such as domains, classes and types, called **second-order objects**. Things may be clearly identifiable in space and time, as a person or a cloud, may be abstract and universal, as a number or a company, or vague in some sense, as an emotion, the age of a person or yesterday's weather.

Things may be related together and these relationships are themselves things. We distinguish relationships from rules, these being universal statements and happenings which can be true and cause a consequence.

The data of the UDD, representing a thing of the Real World, may be stored explicitly in the system by an identifier or a description, or are derived from other data by the application of a function or a derivation rule.

2.1 ENTITIES AND RELATIONSHIPS

An entity is a thing that is identifyeable and distiginguishable from other entities. It may be concrete, as a physical object, such as a car, a person, a road, a cloud or the solar system. Usually such entities have a specific position in space. Some of them are mobile and some have a limited existence in time.

Other kinds of entities are abstract things, in the sense that they exist by human imagination. These are numbers, colors and addresses, but also corporations, families and universities.

Since all things of the real world are represented as data in the model world, their meaning must be fixed adequately. '15' can be a number, an age, a weight or other unity. Some entities are universal and self-identifying as a number, a name or a color. We call these **weak entities**. Besides numbers and names, typical weak entities are qualities (as *heavy, far away, hard*) and situations (as *happy, in love, hot, cloudy*). Note that whereas an adjective as *be happy* is subjective to a person, the noun *happiness* is an universal concept of an emotional status.

Other entities are represented by some distinguishing attributes, a description or an explicitly created identifier. These are the **strong entities**. All physical objects as well as well-defined abstract entities, such as organizations and societies, are strong entities. A strong entity must not ever be well identified in the system. For instance, the *third man* of a crime may be unknown, but I'm sure he was at the crime scene.

Entities may be static, as the earth, a river or a company, or dynamic and move in space, or appear and disappear in time. This quality can depend on the context, as a *company* may be considered static if its lifetime is not of interest. An entity as a whole may be timeless, but have a very time dependent dynamic components or state evolutions. This state is given by the values of their attributes and the history of its components. A *company* is an example of such an entity with dynamic components.

Between entities one or more named **relationships** may hold. We denote this as *rel(e1, e2)* where *rel* is a name given to the relationship. If *e2* is a weak entity, the relationship is also called an **attribute** of *e1*. For instance, *married(e1, e2)* is a relationship but *has-name(e1, e2)* determines an attribute of *e1*. A relationship *rel(e1, e2)* can also be considered a predicate, and we denote *rel(e1) = {e2 / rel(e1, e2)}*.

The classification of a thing to be modeled as an entity or a relationship depends on the context. For example the relationship *married(e1, e2)* can also be considered an entity called

couple with two components *e1* and *e2*. This choice is recommended if the *couple* has its own attributes and relationships.

If an entity *e* has an identifier, this is denoted by *id(e)*. For a dynamic entity *e* we denote its location and its temporal validity as *sp(e)* and *tm(e)*. A mobile entity is a triple (*e, s, t)* we have the functions *sp(e,t)=s* and *tm(e,s)=t*.

Several kinds of indeterminacy can occur with entities, mainly with weak entities. We can have a person *e* with an attribute *age(e)*. This age may be precise, as 60, may be fuzzy, as *old* imprecise as *60 or 61* or indeterminate, as *unknown*. In all cases we can know that another entity e_1 has the same age of *e*, i.e. $age(e_1) = age(e)$, or we know only that $age(e_1) < age(e)$. Also relationships can be indeterminate, such as $married(e_1,?)$ if we know that e1 is married but we don't know with who.

2.2 DENOMINATIONS, TYPES AND CLASSES.

In order to refer to an entity in the UDD we use linguistic ressources such that a language and other ressources, such as mathematical or musical notations.

Some concepts are not first-order entities, but are used to refer to them, since they play a significant role in their qualification. These are **denominations** as domains (as *Mathematics*), keywords (as *Taxonomy*) and contexts (as *University*).

A denomination can be related to a documents, a text or to some entities in order to support a classification of these things. If *d* is such a thing we define the relations *about(d, dom)* for domains, *has-key(d, k)* for keywords and *has-context(d, c)* for contexts. With these denominations we can model the case where an entity has distinct behavior depending of their context. For instance a person *p*, may have the rules:

IF has-context(p) = Family THEN behaves(p) = authoritarian

IF has-context(p) = University THEN behaves(p) = calm

Since textual references to entities may be ambiguous, the use of denominations gives support in the disambiguation of the references (SCHIEL 2021).

Entities are individuals with some characteristics, called its **type**. The set of all entities of a specific type is called a **class**. In order to set what means 'a specific type' all instances of a class must satisfy some characteristic rules of the type.

Types and classes are second-order objects. They are written in uppercase. Examples are *PERSON, FAMILY, INTEGER* and *AGE*.

An entity may remain an instance of a class only temporarily. For instance, a person may be a *STUDENT* for some time and, after this, become an *EMPLOYEE*. He also can be a *STUDENT* and an *EMPLOYEE* at the same time.

The relation between an entity 'e' and its class 'C' is *instance-of(e, C)*. Note that an entity may be instance of several classes, e.g. *instance-of(e, STUDENT), instance-of(e, EMPLOYEE)* and *instance-of(e, PERSON).*

The indeterminacy of some information may hinder a precise classification of entities. Let be the rule defining a class

$$e \in ELDER\text{-}PERSON \Leftrightarrow \text{age(e)} > 65$$

But the age of *e* is only known as *age(e) > 60*. For this situation we need a notion of inaccurate pertinence: $e \approx\in ELDER\text{-}PERSON$.

2.3 DOCUMENTS AND CONTENTS

Some things are not relevant by itself, but more by the things they encompass. We call these **containers**. Even though trains and houses can also be considered containers, we limit ourselves here to information containers, which content is some piece of information. The other containers are classified as entities related to their content by the relation *has-part*.

Information containers are called **documents** and may contain readable information as text, hiperlinks, figures and images, or audible/visible data, as music a speech or a video. This does not rules out that the container itself may also matter. It may have an author, a title an owner and localization.

A document may contain from a single text, as a poem, to a complete multimedia encyclopedia, a video or an audio. The content of the document may be printed on paper, recorded on audio or video media, or stored electronically (**e-documents**).

Each piece of information contained in a document is called a **content**. The content may be a single word referencing an external entity, an image, or a longer piece of text as a statement, a phrase, a paragraph or a poem. For e-documents a computerized system has automatic access to its content.

Note that a content is not an entity by itself, but a sign referencing to one or more things. Furthermore, a content can be analyzed by its meaning, its author, its context, and audio or video contents can be executed. If a content refers to a program, the statement *run(c)* executes the program.

The relations between a document *doc* and a content *c* is *contains(doc, c)*. A content *c* that references to an external entity *e* is given by *refers-to(c,e)* and we state the rule

IF contains(doc, c) ∧ refers-to(c, e) THEN refers-to(doc, e)

If a content is a keyword, we defines this as

IF contains(doc, k) ∧ keyword(k) THEN has-key(doc, k)

The finding of the contents of a document and the corresponding referred entities can be realized using adequate Information Retrieval tools. A term extracted from a document may be ambiguous and refer to several distinct entities. In order to integrate the term with the whole system a word sense disambiguation algorithm must be provided to find candidate classes for the extracted term. In order to determine the correct *refers-to(doc,e)* we must find a class *C* with *instance-of(e,C)*. For that, the denominations associated to the terms of the document may be useful (SCHIEL 2021). Just like that the document content becomes useful for the system.

Typical references in e-documents are hyperlinks, which refers to other contents. In this case, if *h* is such a hyperlink, *refers-to(h)* is a function that returns the content referred by *h*.

E-documents are important for web encyclopedias such as Wikipedia[2] or WolframAlpha[3], thematic directories as dmoz[4] and interactive e-commerce environments. Internal uses of documents may be risk prevention manuals for product manufacturing or distribution. These manuals will be consulted by system internal routines and, if necessary, appropriate actions will be taken. Documents of this kind are called **Active Documents** and will be discussed in section 3.3.

2.4 TIME AND SPACE

Most entities are not universal things but have their existence limited in time or are located in some place. We speak about temporal entities, spatial entities or spatio-temporal entities. In order to specify these positions we use specific temporal and spatial elements. The reference to time and space in this section apply also to relationships between entities as (*friend ('John', 'Mary')*) or attributes as sensations and qualities *state ('John', happy)*. The content of a document may refer to an entity in a specific spatio-temporal environment. Such spatio-temporal entities are denotated as <e, sp, t>. For instance, we can have a document *doc* and the relation *refers-to (doc, <'Napoleon', sp(Russia), 1812)*>). Given a class *C* if some of its

[2] https://www.wikipedia.org/
[3] https://www.wolframalpha.com/
[4] https://dmoz-odp.org/

instances are spatio-temporal this special attributes can be used considering a subclass of *C*, called *C-ST*, with two additional attributes, *sp*(e) and *tm(e)*.

Space elements: the basic space element is a **point** in a space. If the spatial entity is positioned on the Earth, a geographic coordinate system is used to determine its position and more extensive elements. The most common is the spherical coordinate system, given by latitude, longitude and elevation. For more abstract spaces, mathematical coordinate systems can be used, based on lines, planes or rooms.

Since a spatial entity in general is of an irregular shape, its associated space element is a minimal surrounding geometric figure, as a rectangle or a cube. Given an entity *e* we denote *sp(e)=se* its corresponding space element. The topological relations between geometric areas are transposed to their corresponding entities. If for two entities *e1* and *e2* we have *sp(e1)* $\subseteq$ *sp(e2)* then we state *in(e1, e2)*. The same applies to other relations as *overlaps(e1,e2)* and *touches(e1,e2)*.

The space of an entity may be known only imprecisely. In order to support this modeling, the following expressions must be considered:

in(sp(e),sp(e2)): the space of *e* is define as a subset of *sp(e2)*. (e.g. *in(<'ER2021'>,<'Canada'>)*;

sp(e) = se1 OR se2: the space of *e* is *se1* or *se2*;

sp(e) = NOT(se): the space of *e* is outside of *se*.

Also combined expressions of these must be considered, as *sp(e) = NOT(se1) OR se2*.

Instead of determining the correct spatial coordinates for a spatial element it is common to use other spaces. For example *in(sp('Moscow'), sp('Russia'*)).

Time elements: we consider the time as a linear ordered axis of points. The precise structure of this axis, if it is continuous, discrete, enumerable or limited, depends on the application. The most common convention is to use an enumerable set. In all cases we can distinguish two main concepts:

- **Instants**: each single element of the time axis is called an **instant**. The denotation of an instant may be given by a calendar-clock unit. This unit, called time **granularity**, which can vary from a second or less to a century, is determined by the needs of the application. In some cases, several granularities are adopted. The starting point of the time can be a fixed instant in history or, alternatively, we can consider the present as the zero point. In this case, negative times are the past and positive the future.

- **Intervals**; since the time axis is ordered, each contiguous sequence of instants is called an **interval**. An interval *I* is denoted by its endpoints <*i-*, *i+*>. An interval with a single instant {t} is also considered an interval. The interval may be closed (including the endpoints), open or semi-open.

Between two time intervals *I* and *J* we define the relations:

$$before(I,J) \Leftrightarrow I+ < J-$$

$$during(I, J) \Leftrightarrow I- > J- \wedge I+ < J+$$

From this two relations other, as *meets(), overlaps(), starts()* and *between()* can be derived (SCHIEL, 1985).

The lifetime of an entity *e*, denoted as *tm(e)= te*, is given by one or more disjoint intervals. We call each disjoint set of time intervals a **time element**. The components of a time element *te* are ordered, and we define the limits of a time interval *te* as

te- : the starting point of the lowest interval in *te*

te+ : the ending point of the highest interval in *te*

With this, we can redefine the relations between intervals for time elements:

$$before(te1, te2) \Leftrightarrow te1+ < te2-$$

$$during(te1, te2) \Leftrightarrow \forall I \in te1\ \exists J \in te2\ (during(I,J))$$

These relations between time elements can be extended to time entities, considering *before(e1, e2) ⇔ before(tm(e1), tm(e2))* and *during(e1, e2) ⇔ during(tm(e1), tm(e2))*. For instance, if 'John' and 'Mary' are two persons, we may state that *before (fell-in-love('John', 'Mary'), got-married(John', 'Mary'))* meaning the relation between the temporal events *fell-in-love* and *got-married*.

As for space elements, the time *tm(e)* of an entity can also be imprecise. Therefore, given time elements *te, te1* and *te2* the expressions as *in(tm(e), tm(e2)), te1 OR te2* and *NOT(te)* are needed, with similar semantics as for space elements. For instance, if we want specify that the murder occurred last year in September or October, we write *during(tm(<'murder'>), (September/2020 OR October/2020))*.

The value of the function *tm(e)* may have two distinct meanings. It can be the real time while *e* was true (*valid time*) or the time while the system become aware about the fact (*transaction time*). When this distinction matters, we use *vtm()* and *ttm()* for the two functions. Note that, in this case, during the time *vtm(e) – ttm(e)* the system was inconsistent with reality.

Even though we consider a linear time axis, for some applications this linearity must be weakened due to imprecise data, and some kind of branching time will allow the modeling of this weakness. For example, if we know that *fell-in-love('John', 'Mary')* happens last year in July or August, we have two possible pasts. In one of them the marriage can be happened at the beginning of August, in the other case, this is not possible. Similar imprecision can happen with alternative futures. The marriage of 'John' and 'Mary' is planned to happen in two months or on January of the next year. From each of those futures specific facts can be derived. Even the present may be dubious. We can know that he is attending the conference, but are not sure if from home or on site.

A detailed analysis of a logic for indeterminate temporal information are presented by Bezerra, Schiel e Lula (2004).

2.5 RULES

As the content of a document, a rule is not an object for itself, but a holder of some information. It can be checked and executed. We can distinguish 3 types of rules:

- **Boolean rules**: when executed they check some conditions and return a confirmation (*true*) or refusal (*false*) of the conditions. These conditions are called Integrity Conditions. In some cases, also an answer *maybe* must be accepted.
- **Deduction rules**: determine new facts derived from preconditions. Example: *IF tm(got-married(John', 'Mary'))=t THEN tm(fell-in-love('John', 'Mary')<t* or *IF fever(x) > 38^0 THEN sick(x)*
- **Active rules**: an active rule checks some conditions and, if they are true, it executes some action. They will be detailed in the next chapter.

Passive rules fulfill two objectives. They determine the correct semantics of the information structures or ensure application requirements.

Application rules: an application rule is created by the designer in order to implement business rules. Examples: *credit restrictions due to customer's profile*; *a medical prescription; a judgment based on non-compliance with a law*.

Another use of application rules is the specification of the implicit knowledge of a system. It is specified by Horn clauses of the form IFA THEN B, where A must be true in order to deduce B.

An important question is when we should check a rule. A rule management system must guarantee the integrity of the system without overloading it with over-processing. Most rules

are sensitive to changes in the state of the system, as inserts, removes and updates. But, as for the customer profile in the example above, several distinct operations or situations may change this profile.

In some cases, as in the cited examples, the rule is activated on demand but, if the demand is not clear, we can attach the checking rule as pre or post-condition of operations candidate to break the rule or, finally, the rule manager runs a periodical conflict resolution procedure. Inconsistent states must be rotated backwards.

An *IF A THEN B* rule can be matched in two ways:

- Forward chaining: check if A is true and deduce B as new information
- Backward chaining: B is queried or a candidate to enter the system, then check A in order to accept or refuse B.

3 HAPPENINGS

The main objective of a system is to *do* something. In the last chapter we showed the many kinds of thing required for running a complex application. The trend to automate even more the behavior of a system means that not only each process must be carefully planned, but also the conditions under what they can and must be realized.

3.1 LONG TERM EVENTS

A long term event is the happening of a planned and time fixed schedule of activities. It has a start time and a finishing time, and during this period one or more activities occur. Typical long-term events are congresses, meetings, lectures, courses, a travel and other similar programs. As opposed to a temporal entity which has a validity period, during which the entity remains valid, a long-term event exist as a whole. Each piece of its time period is a small piece of the whole event. Usually during a long-term event during its lifetime changes occur continuously.

As composed entities, long-term events have several components such as participants, rooms where it takes place, and possibly other subevents.

3.2 PROCESSES

A process is also a long-term happening, but without pre-established endpoints. A process may be an external happening, as a natural phenomenon like a rain, or a system intern process as the compilation of a program or the execution of a transaction.

In complex systems hundreds of processes may run simultaneously. In this case, an adequate concurrency control mechanism must be adopted in order to avoid mutual interferences.

We distinguish two types of software processes:

- **Transactions**: are programs realizing a well defined task. They can be activated on demand by an external user or triggered by an internal event. In most cases a transaction control mechanism will provide the ACID conditions (atomicity, concurrency, independence and durability).
- **Intelligent agents**: are executable pieces of software which can be moved to distinct places in a distributed environment and performs specific activities due to specific local conditions. Agents can collaborate each other with a common goal. For instance, planning a trip can use an agent to determine the flight, one for hosting and another for the local programming. They all collaborate to determine a complete and consistent trip.

A process is specified formally as:

preconditions [process] post-conditions

Since the process modifies something, *preconditions* is the state which must be true in order to allow the execution of the *process*. *Post-conditions* are the modifications caused by the *process*. This means that for each clause that appears on both sides, their value in the precondition is replaced by that of the post-condition.

3.3 INSTANTANEOUS EVENTS AND ECA RULES

An instantaneous event is the happening of a specific state of the system or a specific state change. For example, *room full* or *midnight*, are such events. The first is a reached state which remains true whereas *midnight* is a single moment. An external user message is also an instantaneous event.

The interpretation must be considered with care. Since the event is instantaneous, what shall we do when it was detected some instants later?

An instantaneous event is a state and does nothing, but will be used for the triggering of some actions or processes. This mechanism is described by the so-called ECA-Rules [BERNDTSSON, MELLIN 2017]. It means "*when event E happens, check condition C and, if true, run A*". Notation:

ON event IF condition DO action

Both the event E as the condition C may be a logical expression of several components. An event can be a basic operation as an insert or update, or can be a state reached as consequence of a process. The interpretation of an ECA-Rule seems to be simple, but involves several critical cases:

- How to detect an event and how long it remains true?
- If two rules have the same event, what trigger first? And if one rule invalidates the event of the other?
- Is an ECA-Rule a transaction with atomicity? The fulfillment of a compound event may take a longer time.

Suppose the rule *ON room full IF empty DO close the door*, after closing the door the room continues full. In order to avoid the repetition of the action, we must add a condition *C:door open*. This could be generalized in the following manner. For each ECA-Rule that must occur only once, if the post-condition of the action is *postA*, add the condition *C: not(postA)*.

ECA-Rules can be used to create production rules, deduction rules and integrity constraints:

- **Production rules**: The rule triggers an action conditioned to an event under some conditions (E.g. *ON end-of-month IF cash amount>sum(salaries) DO issue payroll.*
- **Deduction rules**: The event is a boolean or a retrieval query, the condition the prerequisite to fulfill the query and the action the confirmation (e.g. Boolean query: *ON grandfather(x,y)? IF ∃z(father(z,y) ∧ father(x,z)) DO true*, Retrieval query: *ON grandfather(?,y) IF (father(z,y) ∧ father(x,z) DO grandfather(x,y)*])
- **Integrity constraints**: in this case, the event is the attempt to realize an operation, the condition is the constraint and the action aborts the operation if it is invalid. (e.g. *ON employ(e,s) IF s < minimum-salary DO abort*). If the constraint should be checked periodically, the ON condition includes the period (e.g. *ON new hour*).

ECA-Rules may be entered to the system by an explicit interface, where the three components are defined and also the priority parameters be fixed. If we have an e-document

containing instructions for exception procedures (an Active Document), the text must be interpreted and the instructions converted to ECA-Rules.

Processes are ECA-rules without event. Integrating the two we obtain the specification:

ON demand IF preconditions DO post-condition

The semantics is that the occurrence of the event checks the *preconditions* and triggers the execution of a transition that makes the *post-conditions* true, corresponding to the action A.

3.4 ACTIVE DOCUMENTS

As cited in section 2.3 a document may contain instructions about the execution procedures in the system. These procedures may be internal processes of the system or warnings for the external user. They may be triggered by a manual activation by the reader of the document or an internal process searches for the instruction and executes it. Typical uses of active documents are:

- In a evidence-based health diagnosis system, consults reliable sources and suggests adequate treatments;
- A judicial system analyses the matching of a crime with the current rules and suggests a judgment;
- A quality control detects a defect and reacts according to a contingency manual;
- Each hyperlink of a web page can be used by the reader to navigate to other point. The link can also trigger a process.

In order to use an e-document, it needs to be prepared properly. This preparation must interpret adequately the conditions of the action and the action itself. The dynamics of such documents occur in three steps:

1. Read the document and find the point of interest,
2. Interpret and check possible preconditions for the action,
3. Interpret the described action and executes it.

These steps can easily be converted into an ECA-Rule.

4 STRUCTURAL RULES BETWEEN THINGS

The correct modeling of things and happenings is decisive for building adequate systems in a Dataism environment. We present now a series of statements and rules which must be considered to improve the expressiveness and keep the system healthy.

Notation: *A, B , C* are classes, *e, e1, e2* are entities, *p(A)* is the set of all properties of *A* (attributes and relationships). For a class *A*, |*A*| is the set of all instances of *A*. For an instance *e*, instead of writing $e \in |A|$ we write $e \in A$

4.1 ENTITIES AND CLASSES

The degree of similarity between the instances of a class is due "*according to some principles or purposes*" (HJORLAND, 2017). These purposes depend on the environment in which the entities are used. Since a reactive system is multipurpose, different classes may be necessary for distinct purposes. These classes may be related by special relations called abstractions. This means that at a higher level some details of the subclasses become insignificant. The abstractions are:

Generalization/Specialization (Hyperonym): A class *A* is a generalization of a class *B* if all instances of *B* are also instances of *A*, with eventually less properties. Definition:

$$\text{is-a}(B, A) \Leftrightarrow (\forall e \in B \Rightarrow e \in A \wedge p(A) \subseteq p(B))$$

A class *A* may be specialized several times, by distinct roles *r*. This case is denote as *is-a(B, A, r)*. For instance, *PERSON* can be specialized to *STUDENT* and *EMPLOYEE* by the role *occupation*, and to *JOUNG*, *ADULT* and *ELDERLY* by the role *age*. Some specializations may be disjoint (as by *age*) or overlapping (as by *occupation*). The generic class can have instances which don't occur in any subclass, as for a retired person. These characteristics of specializations by role must be controlled by adequate application rules.

In a specialization the property $p(A) \subseteq p(B)$ is called **inheritance**, since the instances of B inherits all properties of A. The inheritance is not restricted to properties only, but also to their values. This may also be loosened in some cases. Suppose that for *is-a(B,A)* holds and the instances of *A* (and of *B*) have an attribute *weight(e)*. For the environment in which *A* is used this weight should be given in kilograms, but in the environment of *B* pounds are used. This adaptation of the unit to the environment should be admitted.

Aggregation/Decomposition (Meronym): A class *A* is an aggregation of *B1,..,Bn* if the instances of *A* are compositions of elements of *B1,..,Bn*. Notation: *part-of(Bi,A)* or $A=<B1,..,Bn>$. Definition:

$$\text{part-of}(B, A) \Leftrightarrow (\forall a \in A \Rightarrow (a=<b1,..,bn> \wedge \text{for some } i \in \{1,..,n\} \text{ bi} \in B)$$

This proposition can also be stated at instance level by *part-of(bi, a)*.

Specific cases of aggregations are the following:

- The removal of a part removes the aggregate (e.g. $COUPLE=<MAN, WIFE>$)

- The removal of an aggregate removes all parts (e.g. *MAN=<HEAD, TRUNK, LIMBS>*
- By the replacement of a part, the aggregate is another entity (e.g. *COUPLE=<MAN, WIFE>*).
- Each aggregate must have all parts (e.g. *COUPLE=<MAN, WIFE>*).

All these facts may be true or false, depending on the specific model.

As for the specialization, we can also have distinct decompositions by roles. For instance we can have *HUMAN-BODY=<HEAD, TRUNK, LIMBS>* by *parts*, or *HUMAN-BODY=<NERVOUS-SYSTEM, CIRCULATORY-SISTEM> by subsystems.*

Grouping/Dissolution: this abstraction is often confused with aggregation. The difference is that the parts or members of a group are all instances of the same class and the number of members of a group may vary. It corresponds to the mathematical concept of powerset, with the difference that a group is an entity with properties and, in general, changes of the members of a group does not affect the identity of the group. Given a group class *G* and an instance *g* we denote by $|g|$ the set of members of this group. Definition:

$$\text{member-of}(B, A) \Leftrightarrow (\forall g \in A \Rightarrow |g| \subseteq B)$$

At instance level, we determine

$$\text{member-of}(B, A) \wedge g \in A \wedge e \in g \Rightarrow e \in B \wedge \text{member-of}(e, g)$$

Examples of groupings are *member-of(STUDENT, CLASSROOM)* and *member-of(CLASSROOM, SCHOOL-BUILDING).*

A group is similar to a class, but is a first-order entity. For a grouping also special cases must be distinguished:

- The removal of a group object removes its members
- The groups can be disjoint
- The removal of all members of a group removes the group

The abstractions of generalization and aggregation can also be applied to denominations, such as domains and contexts. For instance, we have *part-of(Algebra, Mathematics), is-a(Mathematics, Exact Science)* for domains, and *is-a(Brazilian Society, National Society).*

The following structural rules hold between abstractions:

$$(\text{is-a}(A,B) \wedge \text{is-a}(B,A)) \Leftrightarrow A=B$$

$$(\text{is-a}(A,B) \wedge \text{is-a}(B,C)) \Rightarrow \text{is-a}(A,C) \quad \text{[transitivity of generalization]}$$

$$(\text{part-of}(A,B) \wedge \text{part-of}(B,C)) \Rightarrow \text{part-of}(A,C) \quad \text{[transitivity of aggregation]}$$

(part-of(A,C) ∧ is-a(B,C)) ⇒ part-of(A,B) [inheritance of aggregation]

(member-of(A,C) ∧ is-a(B,C)) ⇒ member-of(A,B) [inheritance of grouping]

If for two classes *A* and *B* we consider $A \cup B$ and $A \cap B$ also classes, the *is-a* structure defines a partial order but not a Lattice, since we could have *is-a(A,C), (is-a(A,D), (is-a(B,C), (is-a(B, D)* which, in the case of a lattice, would imply C=D.

Note that all properties of the abstractions can be converted to ECA-rules, to be held. For instance, the definiton of a generalization *is-a(B,A)*, becomes the rule:

ON insert(e, B) IF not(instance-of(e, A) THEN insert(e, A)

4.2 DOCUMENTS AND CONTENTS

Similar to the relations between classes, shown in the preceeding section, we can relate documents with their content. In addition to components as chapters, paragraphs, indexes a document may contain **metadata**, such as title and author, and other elements, such as **named entities** and **rules**. These elements may be important for the use of the document as part of the system. This gives raise to the statements:

part-of(METADATA, DOCUMENT)

part-of(ELEMENT, DOCUMENT)

is-a(NAMED-ENTITY, ELEMENT)

is-a(RULE, ELEMENT)

part-of(CONDITION, RULE)

part-of(ACTION, RULE)

is-a(IMAGE, ELEMENT)

is-a(TEXT-BLOCK, ELEMENT)

Given a pair <*c,r*> as element of a document *d*, with an active rule *r* as reaction to contingency *c*, we have the ECA-rule

ON c IF part-of(<c,r>, d) DO run(r)

If an element of a document refers to an entity in a specific spatio-temporal context, this is given by the relation *refers-to(doc, <e, se, te>)*. This context can be obtained analyzing the context of the document and the neighborhood of the place where the citation occurs (ARAUJO, SCHIEL, MARINHO 2015). If C is the class of *e* and *C-ST* the class of <*e*, *se*, *te*> we have the rules:

is-a(C-ST, C)

IF instance-of(<e, se, te>, C-ST) THEN instance-of(e, C)

Fom this we can deduce *refers-to(doc, e)*. A similar deduction can also be obtained Similar rules apply if the entity is only spatial or only temporal.

5 CONCLUSION

Original database modeling where influenced by restrictions determined by the internal data structures. The creation and evolution of semantic data models tried to reduce these restrictions and turn the conceptual model more and more similar to the universe of discourse. The tendency to build active and reactive information systems, to encompass more and more parts of the real world, needs to integrate conventional data processing with geographic data, temporal information, document processing, automatic reasoning and decision making. If our knowledge of something is partial, incomplete or missing, the system must consider the little we know about.

The system should be autonomous enough to perform most of its activities without human intervention. It can interact with other electro-mechanical systems, with communication systems and other.

The taxonomy presented combines traditional concepts from semantic data models as entities, relationships, attributes and classes with concepts of documents with their content, space, time and rules. The correct semantics of these concepts is specified by adequate structural rules.

For the dynamic part of a system we distinguish long-term events from processes and ECA-Rules.

For the classification of static things Hjørland (2017) consider an extensive collection of term which can be classified, including Concepts, Documents, Entities, Ideas and other. But the work is restricted to the classification task, and not consider relations between classes and between things and happenings.

An interesting survey of system behavior modeling was given by Roubtsova (2015). She defines a Finite State Machine as a combination of event types (E), sets of states (S) and of transitions (T), FSM = (E, S, T). This correspond to our processes, given that a transition *t* is a triple $t=(s_i, e, s_j)$, s_i, and s_j are the pre- and post-conditions (or states) of the process/event *e*. Complex processes are built from components using a process algebra, considering sequences ($a.b$), alternatives ($a+b$) and repetitions (a^*). Three kinds of transitions are considered: can-, must- and motivate-transitions. A detailed analysis of modeling tools, from UML to Petri Nets, is performed.

Huang, Huang, Wei (2020) proposed a document processing system which integrates several tasks as named entity recognition, similar text analysis, text classification, summarization, document layout analysis and image captioning. They created a document representation model and a processing framework. What they hasn't considered where Active Documents in which rules are found and actions are executed.

To be useful, our taxonomy must be refined for the special cases occurring in real applications and the predicates converted to running ECA-Rules.

REFERENCES

ARAÚJO JR, J.G. DE; SCHIEL, U.; MARINHO, L.B.: An Approach for Building Lexical_Semantic Resources Based on Heterogeneous Information Sources. In: **30th ACM/SIGAPP Symposium of Applied Computing (SAC)**, Salamanca-Spain, 2015

BERNDTSSON, M., MELLIN, J.: ECA Rules. In: Liu, L., ÖZSU, M.T. (eds) **Encyclopedia of Database Systems**. Springer, Boston, MA. 2017. Disponível em https://doi.org/10.1007/978-0-387-39940-9_504

BEZERRA, E., SCHIEL, U., LULA, B.: LITO – A Logic for Indeterminate Temporal Objects. In: **DEXA'04: Proc. of the Database and Expert Systems Applications, 15th International Workshop**. p. 940-944, 2004.

HARARI, Y.: **Homo Deus: A Brief History of Tomorrow**. 1st ed. Harwill Secker, UK, 2016.

HJØRLAND, B.: Classification. **Knowledge Organization** vol. 44, n.2, p. 97-128, 2017.

HIRSCHHEIM, R., KLEIN, H., LYYTINEN, K. **Information Systems Development and Data Modeling – Conceptual and Philosophical Foundations**, Cambridge, 2008.

HUANG, M.-J., HUANG, C.-F., WEI, C.: DOCPRO: A Framework for Building Document Processing Systems. In **9th International Conference on Advanced Information Technologies and Applications (ICAITA),** Toronto, 2020.

ROUBTSOVA, E.: Advances in Behavior Modeling. In Memon, A.M. (ed.) **Advances in Computers vol. 97**, Academic Press, p. 49-109, 2015.

SCHIEL, U.: Time and Space in Information Systems. In: Sernadas, A., Olivé, A. Bubenco, J. (eds.) **IFIP WG 8.1 Working Conference on Theoretical and Formal Aspects of Information Systems**, Sitges-Spain, 1985.

SCHIEL, U.: Texto & Contexto: por uma recuperação da informação com mais semántica, **Ciência da Informação**, IBICT, 2021 (to appear)

WIERINGA, R. **Design Methods for Reactive Systems**, Morgan Kaufmann, 2003.