Towards a Method to Conceptualize Domain Ontologies

Asunción Gómez-Pérez, Mariano Fernández
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn.
Boadilla del Monte, 28660. Madrid, Spain.
Tel: (34-1) 336-74-39, Fax: (34-1) 336-7412
Email: {Asun, mfernand}@delicias.dia.fi.upm.es

Antonio J. de Vicente
Departamento de Automática
E.U. Politécnica
Universidad de Alcalá.
Campus Universitario, Ctra. Madrid-Barcelona 33,600.
Alcalá de Henares, 28771 Madrid, Spain
Tel:(34-1) 885-48-32, Fax: (34-1) 885-48-04
Email: avicente @aut.alcala.es

ABSTRACT

This paper presents the suite of principles, designs criteria and verification process used in the knowledge conceptualization process of a consensuated domain ontology in the domain of chemicals. To achieve agreement between different development teams we propose the use of a common and shared conceptual model as starting point. To capture domain knowledge of a given domain and organize it in a shared and consensuated conceptual model, we recommend an approach that integrates the following intermediate representation techniques: Data Dictionary, Concepts Classification Trees, Tables of Instance Attributes, Table of Class Attributes, Table of Constants, Tables of Formulas, Attributes Classification Trees, and Tables of Instances. We also provide a set of guidelines to verify the knowledge gathered inside each intermediate representations and between intermediate representations.

1. INTRODUCTION.

Ontological engineering is a craft rather than a science. The cause [3] is that *there is no definition and standardization of a life cycle and methodologies and techniques that drive the development of ontologies.* Usually, each development team follows a suit of principles, design criteria, and phases in the ontology development process. The absence of articulated guidelines and methods make difficult the development of consensuated and non-consensuated ontologies inside and between teams, the extension of a given ontology by others, and its reuse in other ontologies and final applications. We claim that the source of the mentioned problems is *the absence of an explicit and totally documented conceptual model upon which the ontology is built.*

Conceptualization is important in any software engineering development. The conceptualization phase [8] includes the objects presumed or hypothesized to exist in the word and its relationships. Its goal [2] is to structure the domain knowledge in a conceptual model that describes the problem and its solution in terms of the domain vocabulary. The IDEAL Methodology [5] proposes two simultaneous activities called analysis and synthesis in the conceptualization phase. The analysis detects the factual, tactual, strategical knowledge and the meta-knowledge of the domain. The synthesis uses this knowledge to build a static and dynamic model which leads the conceptual model of the system. Both modeling activities use a set of intermediate representations (IRs) to summarize the knowledge.

The following sections describe our approach to solve some of the problems that the ontology development process have. In section Two, we propose the first draft of a method to build ontologies. Sections Three to Ten describe a set of knowledge intermediate representation techniques that allow express knowledge in a structured and organized way. Some of them are partially based on IDEAL [9] intermediate representations. Others are specifically created to conceptualize ontologies. For each one, we provide its definition and

utility, followed of how to use them, how to verify the knowledge gathered in them, and how to perform crossed verification between IRs. We have conceptualized a domain ontology focused in objects [7] in the domain of chemicals to prove the viability of the ideas we propose in this paper.

2. TOWARDS A METHOD TO BUILD ONTOLOGIES.

Ontological engineering requires the definition and standardization of a life cycle that goes from requirement definitions to maintenance of the finished product, as well as methodologies and techniques that drive their development. Our experience in the chemicals ontology development process allows us to distinguish four steps in the development of any ontology[1]:

Step #1. To capture knowledge of a given domain and to develop a requirement specification document.

Step #2. To conceptualize it in a set of IRs. The activities we have identified for a domain ontology about objects [7] are:

1. To identify concepts, their instances, attributes and their values in a *Data Dictionary*.
2. To classify groups of concepts in *Concepts Classification Trees*.
3. To describe constants in a *Table of Constants*.
4. To describe instance attributes and class attributes in *Tables of Instance Attributes* and *Tables of Class Attributes.*
5. To describe formulas used to infer numerical values of attributes by using *Tables of Formulas*.
6. To gather the inference sequence of the attributes in *Attributes Classification Trees*.
7. To describe instances in *Tables of Instances*.

Activities 1 to 7 are not secuencial in the sense of a Waterfall life-cycle model [10], but a kind of secuenciallity of any nature must be followed to assure that the first two steps lead us to a well defined model without any redundancies, omissions and inconsistencies. At this phase, the documentation generated allow: (a) to figure out if an ontology is useful and usable for a given application without inspect its source code; and (b) to compare the scope and completeness of several ontologies, their reusability, and shareability by analyzing the knowledge expressed in each IR.

Step #3. To implement the conceptual model in a formal language like: Ontolingua, description logic, etc.

Step #4. To evaluate the ontology with respect to a frame of reference during each phase and between phases of their life-cycle [4].

We think that this approach really improves the shareability and reusability of an ontology since before coding the ontology in a formal language, agreements in the conceptual model between all the development teams are required.

3. DATA DICTIONARY.

Definition and Utility. The first thing to do when the ontology builder tries to capture knowledge of a given domain is to build a Data Dictionary (DD). The DD identifies and gathers *all* the useful and potentially usable domain concepts, their meanings, attributes, instances, etc.

How to build a Data Dictionary. For each identified concept in the domain, the ontology builder should fill in the following fields of the dictionary:

- *Concept Name*;

---

[1] See the ontology at http://delicias.dia.fi.upm.es:5915/ontologias/chemicals.html

- *Synonyms* and *acronyms* of the concept name;
- *Description*, which provides the meaning of the concept name;
- *Instances*, which includes the instances of the concept;
- *Class Attributes*, or relevant properties of the concept that describe the concept itself;
- *Instance Attributes* or relevant properties that describe the instances of a concept.

Table 1 summarizes some concepts (`Alkaline, Halogens, Elements`) identified in the domain of chemicals, as well as information about their meaning, attributes, instances, etc.

Knowledge Verification inside the DD. If the DD is almost built, a few checks will allow to detect omissions of some pieces of knowledge. The aims are:

a) To guarantee the completeness of the knowledge attached to each concept. That is, the concept description is concise and all the relevant instance attributes, class attributes and instances have been identified.
b) To determine the granularity or level of detail of the concepts covered by the ontology.
c) Consistency of the instance attributes and class attributes. That is, they make sense for the concept.
d) Concept names and descriptions. To assure absence of redundancies and to keep concision.

4. CONCEPTS CLASSIFICATION TREES.

Definition and Utility. The Concepts Classification Trees organize domain concepts in taxonomies. They are used not only to know how concepts relate each other, but to modularize the domain knowledge in independent ontologies.

How to build a Concepts Classification Tree. Once the ontology builder has almost done the DD, the next step is to develop Concepts Classification Trees. Given all the concepts of the DD, a concepts classification tree usually organizes the domain concepts in a class/subclass taxonomy in which concepts are linked by mutually-disjoint subclass-of[2] relations, exhaustive-subclass-partition[3] and subclass-of[4] relations. The ontology builder should be able to classify them in mutually disjoint sets. In fact, for each disjoint set a concepts classification tree is built. Each tree leads to build an independent and modularized ontologies in the application domain. Figure 1 shows a concepts classification tree attached to the DD given in Table 1.

Knowledge Verification inside a Concepts Classification Tree. This IR is useful to graphically display the hierarchy of concepts of the DD. Its aims are to assure: (a) non repetition of concepts, to prevent redundancies in the conceptual model; (b) absence of cycles between concepts; (c) there are not isolated subtrees about related concepts.

Crossed Verification between IRs. All concepts which are displayed in the tree must be defined in the DD. If not, must be added to the DD or removed in the tree.

Before going straightforward, it is suitable that all teams involved agree with the definitions of the DD and the concepts classification trees generated. If they are, they might distribute the effort of capturing knowledge in other IRs speeding up the ontologies development process.

5. TABLES OF INSTANCE ATTRIBUTES.

Definition and Utility. A table of an instance attribute provides information about the

---

[2] It is a set of subclasses of a class C whose objects have no member which belong to differents sets.
[3] A subrelation-partition of a class C is a set of mutually-disjoint classes (a subclass partition) which covers C. Every instance of C is is an instance of exactly one of the subclasses in the partition.
[4] Class C is a subclass of parent class P if and only if every instance of C is also an instance of P.

attribute or about its values at the instance. For each instance attribute included at the field *Instance Attribute* of the DD, a table must be created.

How to build a Table of an Instance Attribute. Each table includes:

- *Instance Attribute Name*;
- *Description*, which provides the meaning of the instance attribute;
- *Value Type* refers to the class of values (natural, real, boolean, string, ...) with which the attribute could be filled in;
- *Unit of Measure* for numerical values. If the value is not numerical, we introduce the symbol "__". This symbol will be used, throughout other IRs, to express that a given field has no values. If we wonder to show that the value is unknown, we will use the symbol "unknown.";
- *Precision* of the numerical value;
- *Range of Values*, which specifies a list or set of possible values of the attribute;
- *Default Value*, if they are known;
- *Cardinality*, which specifies the number of values of the attribute;
- *Inferred from instance attributes*, *inferred from class attributes and Inferred from-constant*s which include the name of those instance attributes, class attributes and constants that allow to infer the value of this attribute;
- *Formula*, which includes crossed references to the tables of formulas that allow to calculate the numerical value of the attribute (it might be more than one formula);
- *To infer*, which contains the name of those instance attributes whose values could be inferred using the instance attribute;
- *References*, to record the source of the information (a book, expert, etc.).

Tables 2, 3 and 4 show definitions of the instance attributes: `Atomic-Volume-At-20-Celsius-Degrees`, `Atomic-Weight` and `Density-At-20-Celsius-Degrees`.

Knowledge Verification inside a Table of Instance Attributes. The ontology builder would need to use common sense knowledge to guarantee that these tables lack of mistakes/omissions. Next points must be satisfied:

a) If an instance attribute (`Density-At-20-Celsius-Degrees`) is inferred using instance attributes (`Atomic-Volume-At-20-Celsius-Degrees` and `Atomic-Weight`), its field *Inferred from Instance Attributes* is filled in with the names of the instance attributes used to infer its value. At the tables of instance attributes, the field *To infer* is filled in with the name of the instance attribute that can be inferred.
b) Default values must be fitted with the value type of their attribute.

Knowledge Verification between IRs. To guarantee consistency and completeness with the knowledge captured in previous IRs, the ontology builder should check:

a) For all the instance attributes identified in the *Instance Attribute* field of the DD, there exists a table of instance attribute, and that for each table of instance attribute, there exists an instance attribute in the DD.
b) If an instance attribute is inferred using class attributes and constants, its fields *Inferred from Class Attributes* and *Inferred from Constants* are filled in with the names of the class attributes and constants used to infer its value. At the table of class attributes and at the table of constants, the field *To infer* is filled in with the name of the instance attribute that can be inferred.
c) If an instance attribute is inferred from other instance attributes, class attributes and constants using a formula, then it must possibly to deduce the *Unit of Measure* of the inferred instance attribute using the composition of unit of measures of each instance attribute, class attribute and constant.

6. TABLES OF CLASS ATTRIBUTES.

Definition and Utility. Class attributes describe the concept itself, not its instances. So, a table of a class attributes provides information about them and about their values. For each concept included at the field, a table of class attributes must be created.

How to build a Table of Class Attributes. Specifying the following fields:

- *Class Attribute Name*;
- *Relation Attribute Name*; it is the name of the attribute which participes in the relationship.
- *Logic Relationship,* attaches two concepts across logic operators;
- *Value* of the class attribute;
- *Unit of Measure* for numerical values;
- *To infer*, which contains the name of those instance attributes whose values could be inferred using the value of the class attributes;
- *References,* to record the source of knowledge.

Table 5 shows the definitions of the class attributes attached to the concept `Halogens.`

Knowledge Verification inside a Table of a Class Attributes. To guarantee the lack of mistakes/omissions, the ontology builder would need to use common sense knowledge. In Table 5, we checked that `Minimum-Melting-Of-Halogens` is lower than `Maximum-Melting-Point-Of-Halogens.`

Knowledge Verification between IRs. The checks to be done are:

a) For all the concepts that have defined class attributes in the DD, there exists a table of class attributes, and for each table of class attributes, there exists a concept in the DD.
b) For all the class attributes specified at the field *Class Attributes* of the DD, there is an entry at a table of class attributes, and viceversa.

7. TABLE OF CONSTANTS.

Definition and Utility. Constants are used to specify information related to the domain of knowledge, they take always the same value, and they are usually used in formulas. For example, gravity acceleration is 9.8, the water melting point is 100 Celsius degrees, and so on. A few constants in the domain of chemicals appear in Table 6.

How to build a Table of Constants. For each concept classification tree, the ontology builder should identify a set of constants and describe them following the pattern:

- *Constant Name*;
- *Description*, which contains the meaning of the constant name;
- *Value* of the constant;
- *Unit of Measure* for numerical values;
- *To infer*, with the name of the instance attributes whose values could be inferred using the value of the constant;
- *References,* which refers the source of knowledge from which the constant was elucidated.

Knowledge Verification inside a Table of Constants. Checks to be done are:

a) For any constant which appears in the table of attributes or in the formulas one, must be also listed in a table of constants.

A table of constants may contain some items that won't be used by any formula or for inferring any value of an attribute. These constants are kept for maintaining the reuse of

knowledge by other applications.

Knowledge Verification between IRs. The only checks to be done are specified in section five, when constants are used to infer values of instance attributes.

8. TABLES OF FORMULAS.

Definition and Utility. In many domains, numerical values of instance attributes might be derived from numerical values of other attributes and constants by using formulas.

How to build a Table of Formula. The standard definition of formulas includes:

- *Formula Name*;
- *Inferred Attribute*, which identifies the instance attribute that is calculated with the formula;
- *Formula* includes the mathematical formula to be used to calculate the inferred attribute;
- *Description*, which includes the theoretical foundations of the formula;
- *Basic Instance Attributes* which refer to the list of instance attributes used by the formula to calculate the inferred attribute;
- *Basic Class Attributes* contain the list of class attributes used calculate the inferred attribute;
- *Constants*, which include the list of constants used by the formula to get the inferred attribute;
- *Precision* with which the number should be calculated (i.e., two decimal numbers);
- *Constrains*, if the formula is appropriate under specific values of basic instance attributes it is necessary to specify some conditions (i.e., the formula must not be used if the value of a basic instance attribute used as divisor is zero);
- *References*, which refers the source of knowledge from which the formula was elucidated.

Table 7 shows the definitions of the formula that calculates a value of the `attribute Density-At-20-Celsius-Degrees`. Note that to infer its value, only values of instance `attributes (Atomic-Volume-At-20-Celsius-Degrees` and `Atomic-Weight`) are used.

Knowledge Verification inside a Table of Formulas. We must guarantee:

a) All the attributes which appear in the right-side of a formula must appear in the *Basic Instance Attributes, Basic Class Attributes* or in the *Basic Constants* fields.
b) Given an instance attribute (`Density-At-20-Celsius-Degrees`) whose value is inferred by using a formula, the fields *Inferred From Instance Attributes, Inferred From Class Attributes,* and *Inferred From Constants,* at its table of attributes must be filled with the names of the involved instance attributes (`Atomic-Volume-At-20-Celsius-Degrees` and `Atomic-Weight`), class attributes and constants used in the formula.
c) If the definition specifies some constrains, other formulas should cover alternative situations.
d) We must re-read the descriptions of the formulas in order to assure the absence of redundancies and to keep the concision.

Knowledge Verification between IRs. To guarantee that the information gathered in the table of a formula is complete and consistent with those gathered in tables of instance attributes, in tables of class attributes, and in tables of constants, the following checking should be done:

a) For all the formulas identified in the *Formula* field of the tables of instance attributes, there exists a table of formula. In the tables of formulas, the field *Inferred Attribute* is filled in with the name of the instance attribute inferred by using the formula.

b) Analyze the use of the formula to detect if some instance attributes need default values. For example, if a formula adds values of two instance attributes, some of them could be

optional. Then, the field *Default Value* of the optional attribute should have been filled with a default value (i.e., zero).

9. ATTRIBUTES CLASSIFICATION TREES.

Definition and Utility. Once the Ontology builder has verified the completeness and consistency of tables of instance attributes, class attributes, constants, and formulas, the next step is to develop Attributes Classification Trees. They display graphically attributes and constants that are related in the inference sequence of the root attributes, as well as the sequence of formulas to be executed to infer the root attributes.

How to build an Attributes Classification Tree. Using the fields *Inferred from Instance Attributes, Inferred from Class Attributes, Inferred from Constants, To infer* and *Formula* defined at each table of instance attributes. The result will be several attributes classification trees. Figure 2 shows one of these trees and the formula that make it possible.

Knowledge Verification inside an Attributes Classification Tree. We must check:

a) If some attributes and constants used in the sequence of inference of the root attribute are missed, or they are wrong, or they are not useful to infer values of other instance attributes or even the root attribute.
b) If some formulas are missing or if they are used wrongly.
c) There are no cycles in the tree, that is, from an instance attribute we are not able to infer itself.

Knowledge Verification between IRs. To prevent inconsistencies between an attributes classification tree and tables of attributes, constants and formulas used to build it, any modification (addition/removal) in the tree forces to modify the involved tables of attributes, constants and formulas, and the DD. For example:

a) If any attribute is included into the tree and either it is not defined in the DD or doesn't exist its table of attributes, then we should include it at the DD and/or create its attached table of attribute.
b) If any attribute is removed in the tree, the builder should analyze whether or not the attribute should be deleted from the DD and/or if its table of attribute should be also removed.
c) The same checks must be done for formulas than for attributes.

10. TABLES OF INSTANCES.

Definition and Utility. If the ontology builder is sure that all the instances mentioned at the *Instance* field of the DD exist in the domain, the next step is to create a table of instance for each instance identified in the DD. An example of a table of instance is Table 8.

How to build a Table of an Instances. We propose the following fields:

- *Instance Name*;
- *Description*, which provides the meaning of the instance name;
- *Attributes*. Allowed attributes filled in at the instance are defined into the classes with which the instance is linked and into their superclasses;
- *Values* of these attributes, could be: a number, a string, a set of values, "unknown", or "__".

Knowledge Verification inside a Table of Instances. We should guarantee:

a) Absence of redundancies between instances. We must carefully read the description of the instances to guarantee this point.
b) Consistency between attributes and their values.

Knowledge Verification between IRs. The following checks should be done:

a) For all the instances identified in the *Instance* field of the DD there exists a table of instance, and that for each table of instance, there exists an *instance* in the DD.
b) Propagation of attributes using inheritance. All the attributes identified in classes and superclasses to which the instance is attached to should be filled in with concrete values. In other words, there are not attributes in the instance that are not in the path between the instance and the root concept of its concepts classification tree.
c) Maintenance of consistency between the values of the attribute at the instance and the *Value Type* field defined at its table of instance attributes. That is, the value is a legal value for the attribute.
d) Maintenance of consistency between the number of values asserted in an attribute and the *Cardinality* field defined at its table of instance attribute.

CONCLUSION.

We provide the first draft of a method to build domain ontologies. The method distinguish four steps in any ontology development process: the capture of knowledge, its conceptualization in a set of IRs, its implementation in a formal language, and its evaluation during each phase and between phases of its life cycle. The method uses a set of IRs at the conceptual phase that allow to identify concepts and their attributes and values, as well as relationships between them. These IRs allow to specify a set of ontological commitments.

For each IR, we have identified a set of checks out to be done with independence of whether or not the ontology is been developing by different teams. These checks help to detect inconsistencies, redundancies and lacks of knowledge inside an IR and between them, at the earliest stages of the ontology development process. The use of the IRs allows generate an explicit and totally documented conceptual model upon which the ontology is built.

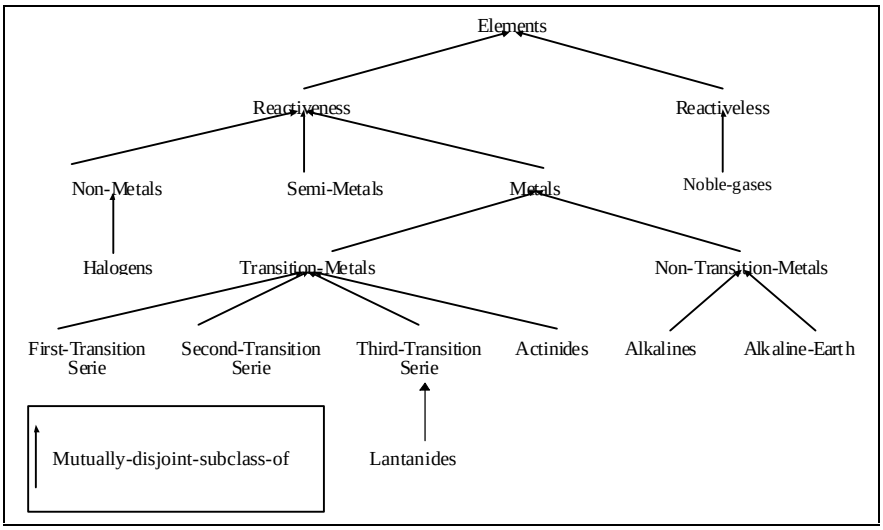

Figure 1: Concepts Classification Tree *in the domain of chemicals.*

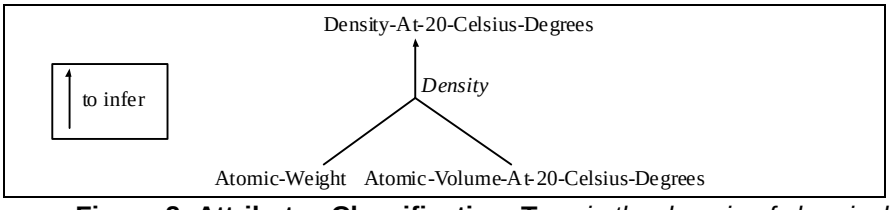

Figure 2: Attributes Classification Tree *in the domain of chemicals.*

Concept Name	Synonyms	Acronyms	Description	Instances	Class Attributes	Instance Attributes
Alkalines	Ia Group	AlK.	Their valence layer electronic-structure is ns^1. The Alkalines are soft; when they are cut, they have a white silvered colour. When they are exposed to air, they are clouded because of the oxidation, due to this, they are kept in a inert atmosphere or submerged in mineral oil. The melting-point, the boil-point and the specific-heat decrease when the atomic-number increases. They react with the hydrogen. [1].	Lithium. Sodium. Potassium. Rubidium. Cesium. Francium.	Group-Of-Alkalines Low-Electronegativity-Of-Alkalines Oxidation-States-Of-Alkalines Minimum-Of-Electronic-Affinity-Of-Alkalines Maximum-Of-Electronic-Affinity-Of-Alkalines	List-Of-Bases
Halogens	VIIa Group Halogenous	Hlg.	They have seven electrons at the valence layer, two of which are in a s orbital and the rest of them in a p one. They easily react because of they only need one electron for obtaining eight in their valence layer. [1]. Their melting-point, boil-point and density increase with the atomic-number.	Fluorine. Chlorine. Bromine. Iodine. Astatine.	Group-Of-Halogens Minimum-Of-Electronic-Affinity-Of-Halogens Maximum-Of-Electronic-Affinity-Of-Halogens Minimum-Melting-Point-Of-Halogens Maximums-Melting-Point-Of-Halogens	List-Of-Salts
Element		Elmts.	It is a substance that are made up only by atoms with the same number of protons.			Atomic-Number. Atomic-Volume-At-20-Degrees-Celsius. Atomic-Weight. Boil-Point. Crystalline-Structure. Density-At-20-Degrees-Celsius Electronegativity Electronic-Affinity Electronic-Structure Group half-life Hardness-Scale Ionisation-Energy Melting-Point Oxidation-States Period Radioactivity-Constant Resistivity-At-20-Degrees-Celsius Resistivity-Temperature-Coefficient-Per-Degree-Celsius Semidisintegration-Period Specific-Heat-At-Standard-Temperature Symbol Thermal-Conductivity-At-Standard-Temperature

Table 1: Data Dictionary *in the domain of chemicals.*

Instance Attribute Name	Atomic-Volume-At-20-Celsius-Degrees
Description	It is the volume occupied by 1 atom-gram of a element at 20 Celsius degrees. The atom-gram is the mass corresponding to atomic weight expressed in grams. [6]
Value type	real
Unit of Measure	pot(Centimeter, 3) / Atom-Gram
Precision	---------
Range of values	(0, 100)
Default value	---------
Cardinality	1
Inferred from instance attribute	unknown
Inferred from class attribute	unknown
Inferred from constants	unknown
Formula	unknown
To infer	Density-at-20-degrees-Celsius
References	[1] [6]

Table 2. Table of Instance Attribute: *Atomic-Volume-At-20-Celsius-Degrees.*

Instance Attribute Name	Atomic-Weight
Description	The atomic-weight is the relative mass that a atom of the considered element has in a scale in the which it is taken as unit the twelfth part of the carbon 12 isotope.[6].
Value type	real
Unit of Measure	AMU[5]
Precision	---------
Range of values	(1, 257)
Default value	---------
Cardinality	1
Inferred from instance attribute	unknown
Inferred from class attribute	unknown
Inferred from constants	unknown
Formula	unknown
To infer	Density-at-20-degrees-Celsius
References	[1]

Table 3. Table of Instance Attribute: *Atomic-Weight.*

Instance Attribute Name	Density-At-20-Celsius-Degrees
Description	Density of a body is the mass of a volume unit [1].
Value type	real
Unit of Measure	Gram / pot(Centimeter, 3)
Precision	---------
Range of values	(0, 25)
Default value	---------
Cardinality	1
Inferred from instance attribute	*Atomic-Weight* and *Atomic-Volume-At-20-Celsius-Degrees*
Inferred from class attributes	unknown
Inferred from constants	unknown
Formula	DENSITY = ATOMIC-WEIGHT /ATOMIC-VOLUME-AT-20-CELSIUS-DEGREES
To infer	unknown
References	a) First acquisition session with the expert in 1995 Springtime. b) [1]

Table 4. Table of Instance Attribute: *Density-At-20-Celsius-Degrees.*

Class Attribute Name	Relation-Attribute-Name	Logic Relationship	Value	Unit of Measure	To infer	References
Minimum-Of-Electronic-Affinity-Of-Halogens	Electronic-Affinity	>	2.5	Electronvolt	---------	---------
Maximum-Of-Electronic-Affinity-Of-Halogens	Electronic-Affinity	<	4	Electronvolt	---------	---------
Minimum-Melting-Point-Of-Halogens	Electronic-Affinity	≥	-219.62	Degree-Celsius	---------	---------
Maximums-Melting-Point-Of-Halogens	Electronic-Affinity	≤	302	Degree-Celsius	---------	---------

Table 5: Table of class attributes: *of the concept Halogens.*

Constant Name	Description	Value	Unit of Measure	To Infer	References
Standard-Temperature	It is the temperature at which many processes are observed and many measures are taken.	25	Celsius-Degree	---------	[Candel. 87]
Standard-Pressure	It is the pressure at which many processes are observed and many measures are taken.	1	Atmosphere	---------	[Candel, 87]

Table 6: Table of Constants *in the domain of chemicals.*

[5] It is the weight of a proton. An atom-gram of an element is the weight of 1/Avogadro's # atoms of this element.

Formula Name	Density
Inferred Attribute	Density-At-20-Celsius-Degrees
Formula	Density-At-20-Celsius-Degrees -Atomic-Weight / Atomic-Volume-At-20-Degrees-Celsius
Description	It is necessary to note that the Atomic-Weight is given in AMU's, and the Atomic-Volume in cm^3 / atom-gram. There is not problem, 1 atom-gram of a element maps with the Atomic-Weight in grams, for instance, 1 atom-gram of oxygen is 15.999 grams, therefore, when the units transformation is done, the things left like they were.
Basic Instance Attributes	Atomic-Weight Atomic-Volume-At-20-Degrees-Celsius
Basic Class Attributes	---------
Constants	---------
Precision	---------
Constrains	Atomic-Volume-At-20-Degrees-Celsius > 0
References	a) First acquisition interview in 1995 Springtime. b) Characteristic tables of elements [1]

Table 7: Table of a Formula *Density-At-20-Celsius-Degrees.*

Instance Name	Description	Attributes	Values
Chlorine	(Gr. Chloros, greenish, yellow). Discovered in 1774 by Scheele, who thought it contained oxygen; named in 1810 by Davy, who insisted it was an element. In nature it is found in the combined state only, chiefly with sodium as common salt (NaCl), carnallite $KMgCl_36(H_20)$, and sylvite (KCl). It is member of the halogens (salt-forming) group of elements; it is a greenish-yellow gas, combining directly with nearly all elements. Chlorine is widely used in making many everyday products. It is used for producting safe drinking water the world over. Even the smallest water supplies are now usually chlorinated. It is also extensively used in the production of paper products, dyestuffs, textiles, petroleum products, medicines, antiseptics, insecticides, foodstuffs, solvents, paints, plastics, and many other consumer products. Most of the chlorine produced is used in the manufacture of chlorinates compounds of sanitation, pulp blanching, disinfectants, and textile processing. Further use is in the manufacture of chlorates, chloroform, carbon tetrachloride and in the extraction of bromine. Organic chemistry demands much from chlorine, for example in the elaboration of synthetic rubber. Chlorine is a respiratory irritant. It was used as war gas in 1915. [11]	Atomic-Number. Atomic-Volume-At-20-Degrees-Celsius. Atomic-Weight. Boil-Point. Crystalline-Structure. Density-At-20-Degrees-Celsius. Electronegativity. Electronic-Affinity. Electronic-Structure. Group. half-life. Hardness-Scale. Ionisation-Energy. Melting-Point. Oxidation-States. Period. Radioactivity-Constant. Resistivity-At-20-Degrees-Celsius. Resistivity-Temperature-Coefficient-per-Degree-Celsius. Semidisintegration-Period. Specific-Heat-At-Standard. Symbol. Thermal-Conductivity-At-Standard-Temperature. List-Of-Salts	17 unknown 35.453 -34.6 --------- unknown 3.0 3.614 $3s^23p^5$ VIIa --------- --------- 12.967 -100.98 [1, 3, 5, 7] 3 0 unknown unknown --------- unknown Cl unknown ["common-salt, "carnallite", "sylvite"]

Table 8: Table of Instances: *Chlorine.*

REFERENCES.

[1] Babor et al. “Química General moderna”. De MArtín. 1979.

[2] Blum B. Y. “Software Engineering a Holistic View”. Oxford University Press. 1992

[3] Gómez-Pérez A. "From knowledge Based Systems to Knowledge Sharing Technology: Evaluation and Assessment". KSL-94-73. Knowledge Systems Laboratory. Stanford University. CA. 1994.

[4] Gómez-Pérez A., Juristo N., Pazos J. "Evaluation and Assessment of the Knowledge Sharing Technology". Towards Very Large Knowledge Bases. JOS Press. 1995. Pg.: 289-296.

[5] Gómez-Pérez A., Juristo N., Pazos J. "Ingeniería del conocimiento: construcción de sistemas expertos" CEURA. 1996 (*for coming*)

[6] Hidalgo P.J. “Curso Breve de Química, Electroquímica y Corrosión”. De. ICAI. 1984.

[7] Mizoguchi R., Vanwelkenhuysen J., Ikeda M. "Task Ontology for Reuse of Problem Solving Knowledge". Towards Very Large Knowledge Bases. JOS Press. 1995. Pg.: 46-59.

[8] Nilsson J., Genesereth M. “Logical foundations of the artificial intelligence”. Morgan-Kauffmann Publishers Inc. 1986.

[9] Pazos J. Conceptualización. Master en Ingeniería del Conocimiento. Facultad de Informática de Madrid. Universidad Politécnica de Madrid. SPAIN. 1995.

[10] Royce W.W. "Managing the Development of Large Software Systems: Concepts and Techniques". Proceedings of WESCON. AUGUST. 1970.

[11] William H.B. “Handbook of Chemestry and Physics”. 65th edition. CRC-PRESS, INC. 1984-85