An Information Platform Developed from First Principles

John E. Grisinger
Vara Data Systems

March 24, 2008

Copyright 2008 by Vara Data Systems
All rights reserved

Abstract. Conventional information platforms store information in a manner that separates the data elements from its organization, e.g., schema and code specify the organization of the data elements but store it separately from the data elements. That separation is contrary to any notion of how humans comprehend the world. The first principles of information, expressed as notions, provide the basis for a method of storing information where data elements and their organization are integrated. The generality of this method appears to enable most, if not all, forms of information to be integrated and stored within one platform. These forms include: traditional database information; mathematical relationships and functions; class and object hierarchies; value domains; temporal variability; uncertainty and random variables; and complex sequences. Sign and count are shown to be independent attributes that arise from the application of these principles. Zero and count are also shown to arise independently. Examples demonstrate that there is only one way to specify a given unit of information using this method. The storage method suggests a foundational basis for understanding information itself.

1. INTRODUCTION

Information technology is founded on the idea that any form of information, ranging from text to databases, can be stored as *data elements* that are organized by a *data storage structure* stored externally to the data elements. Each of today's information management platforms implements that paradigm using various mechanisms for specifying data storage structures (i.e., schema, file formats, data types, documentation and software, as well as primitive data storage structures such as trees and lists). The utility of externally-specified data storage structures is storage efficiency and speed. However, an externally-specified data storage structure limits the type of information that can be stored in a platform. While useful, those data storage structures do not arise from fundamental notions about how we create and use information to comprehend the world. Rather they are primarily analogs of how we organize data elements for the purpose of presenting and communicating information among ourselves, e.g., delimited strings and tabular arrays.

I contend that the problems we face in managing information arise from (1) our use of data storage structures that are analogs of how we present and communicate information, in contrast to how we comprehend the world, and (2) storing data elements and their organization separately.
Here I discuss and demonstrate an extremely general storage method developed from the first principles of information. Those principles are drawn from observations of how humans create and use information to comprehend the world. Like the human mind, this method integrates data

elements and their organization. Significantly, the generality of this method appears to enable most, if not all, forms of information to be stored in the same platform.

2. APPROACH

Logic and mathematics are either a form of information or *describe* the characteristics of various types of information. However, neither logic nor mathematics *prescribe* the characteristics of information, nor do they provide a complete or consistent description of the fundamental characteristics of information as a whole. In addition, some logical and mathematical notions are beyond comprehension (e.g., infinity) or arise from a notational convention (e.g., ordered pair). Logic and mathematics do, however, provide: methods; an expression notation; and typing of information and its components.

The approach used here identifies and then applies the first principles of information using the methods of logic and mathematics as follows:

- Express our understanding of the world in the form of axioms, i.e., independent statements accepted as true without proof;
- Reduce the characteristics observed in all forms of information to those that are the most fundamental and express them by independent self-evident statements each referred to as a <u>notion</u>;
- Develop conventions for both a graphical notation and a character expression notation that convey the subject ideas;
- Identify and distinguish among the components of information by usefully-named types and organize them in taxonomies; and
- Use logical methods (e.g., reductio ad absurdum) to recognize erroneous information.

In order to avoid inconsistent and ambiguous terminology, each term having a variety of meanings or that is possibly ambiguous is defined and underlined when first used. Subsequent use of that term is limited to its defined meaning. Italicized terms in an axiom or notion are implicitly defined therein.

For the sake of brevity, this paper focuses primarily on information typically stored in databases. In addition, only those axioms and notions of interest here are included, some of which are combinations of more resolved axioms and notions.

The discussion introduces each axiom and notion, defines each term, describes each notation convention and presents examples. The examples are intended to demonstrate that this method can store information that is readily stored in a conventional database as well as more complex information that is difficult to store in a database. The following section discusses the elements of the storage method; subsequent sections discuss their implementation, develop various element subtypes, and discuss information structures and patterns using those element subtypes.

3. STORAGE METHOD

This section introduces the axioms and notions about how we comprehend the world and discusses the development of the storage method from them.

3.1 Basic Axioms

Axiom: There is a world that is external to each human that is made of *external things* each having *external properties*. A human comprehends a portion of those external things and external properties from his/her experience with them by storing and processing *mental constructs* of those experiences, as well as anticipated (future) experiences and expectations of experiences.

Axiom: A mental construct is stored in an *information site* that is either (1) a brain or (2) an *information device* created by humans; each information site is independent of the other information sites.

The term *external properties* as used above refers to those that a human can detect by the presence of the external thing alone. Examples of external properties (other than those stored in an information site) are color and mass; name and composing parts are not external properties.

<u>Information</u> is defined as what is stored in an information site. An information device is either a <u>passive information device</u> or an <u>active information device</u>, the latter being a site where information can be altered by a process. Examples of the former are a book and a name tag; examples of the latter are a disc drive and a blackboard. An active information device plus the means of accessing and altering it (e.g., an operating system and applications) is called a <u>platform</u>. A relational database plus its management system is a platform. A brain is a platform in this sense.

Axiom: Within an information site, we can create *internal things* each valuated by *internal properties* as a means of manage our affairs.

An internal thing and an internal property have no counterparts in the external world. The term *internal properties* also refers to those properties of an internal thing alone. Imagining an external thing or property does not make it internal; it's virtual as discussed later.

Axiom: A thing or property may be *composed* of one or more other respective things or properties. There are things and properties that are not composed of other things or other properties.

A thing that composes is called a <u>composing thing</u>; a thing that is composed is called a <u>composed thing</u>. The corresponding terms property are <u>composing property</u> and <u>composed property</u>. The second part of this axiom asserts that the world is ultimately atomic and is not infinitely divisible.

A thing is understood by its: properties, its composing things and its composed things. Each of these is called a <u>thing characteristic</u>. For example the thing characteristics of a human arm are its mass and color, composing a human body and being composed of a hand and fingers. There are two <u>property characteristics</u>: composing properties and composed properties.

Axiom: Within an information site we store *concepts of things* each identifying both (1) the thing characteristics that are expected to apply to each member of a group of things and (2) the thing characteristics that are not. We also store *concepts of properties* that similarly identify property characteristics. A *concept* may be used to classify things and properties.

Here, a thing or property is said to have/not have a characteristic; a characteristic is said to be applicable/non-applicable to a thing or property; a concept is said to identify applicable/non-applicable characteristics; and a characteristic is said to be identified (as applicable/non-applicable) by a concept.

Like an internal thing or an internal property, a concept has no counterpart in the external world, i.e., it's found only in an information site. (*Concept* and *type* are equivalent notions. However, a concept is stored in a site but a type is not.) The following are examples of things and properties, both internal and external, identified by a term that is a name of its classifying concept.

external		internal	
external thing	external property	internal thing	internal property
human body	height	person	person name
rock	hardness	account	Social Security No. (SSN)
Earth	diameter	role	role name
molecule	mass	identification	Vehicle Identification No. (VIN)

Axiom: Each property of a thing applies to the thing as a whole.

For example, "eye color" is not a property of a human body as a whole; it is a property of an iris.

Axiom: Each (external or internal) thing is distinguishable from all others by its *space-time existence*; each (external or internal) property is distinguishable because the thing it valuates is distinguishable.

The earlier axioms contained the phrases: (1) a thing "valuated by" properties, (2) concepts "used to classify" both things and properties, and (3) thing and properties "composed of" other things and properties. These ideas are each embodied in the more general term association.

3.2 Elements

The above axioms described our understanding of the world and the existence of information we use to comprehend it. This subsection introduces notions describing the organization of information in terms of the atomic components common to all forms of information and introduces additional axioms.

Notion: Information is composed of indivisible (i.e., atomic) elements.

As demonstrated in later sections, information stored in a platform can imply elements that are not stored. This is the basis for the element subtypes listed in the following taxonomy.

element
- stored element – an element that is stored
- implied element – an element that is not stored but can be determined from those that are stored

Stored and implied elements are described here as "specifying" information. Thus, the unmodified term "element" means specified element.

Notion: An element is either (1) an *item* that stands for a thing, property or concept, or (2) an *association* that directionally associates two items.

Item and association are subtypes of element. (An item is comparable to a *data element*; an association is the means of organizing items and is often implied by a *data storage structure*.) The term "stands for" refers to either (1) an item *representing* an external thing or an external property, or (2) an item *being* a concept, an internal thing or an internal property. An item is said "to have" an association and "to be associated with" another item. An association is said "to associate" items. We can surmise that an item not having an association cannot specify information because it has no context. Thus an item must have one or more associations.

Axiom: A thing, property or concept either temporally precedes or succeeds a thing, property or concept with which it is associated.

Precedence implies independence; succession implies dependence. The precedence/succession of two items is specified by the directionality of their association. The two ends of an association are distinguished by an end-id that is either "1" or "0", respectively called the 1-end or the 0-end. The sequence is arbitrarily chosen to be from the 1-end to the 0-end. With respect to a given association, the items on those two ends are respectively called the 1-end item and the 0-end item. Since a 1-end item precedes a 0-end item, the two items in the context of an association are also respectively called the preceding item and the succeeding item.

3.3 Information Diagrams

An information diagram, or simply diagram, is a graphic means of depicting associated items. Four diagram subtypes are developed and used here to depict associated items and display each element's subtype. All diagram types depict an item by a circle and an association by a line with its directionality indicated by its end-ids. Figure 1 displays the first two diagram types: an illustrative diagram, or simply illustration, and a model diagram. An illustration displays each (stored or implied) element; a model diagram displays each element subtype once to represent *any* element of that subtype.

> The illustration depicts two stored items and one stored association. A stored element is depicted by a thick solid line; an implied element is depicted by a dashed line (as shown in Figures 8, 21 and 23).
>
> The model diagram displays the two element subtypes introduced above with its name displayed inside or near its depiction. Item is depicted by a thin solid line; association is depicted by a dotted line to indicate that an item having a 0-end or a 1-end association (but not both) is optional. The cardinality of an association subtype (suggested by the crowfeet on each association end) depends on the item subtype as discussed in §3.4.

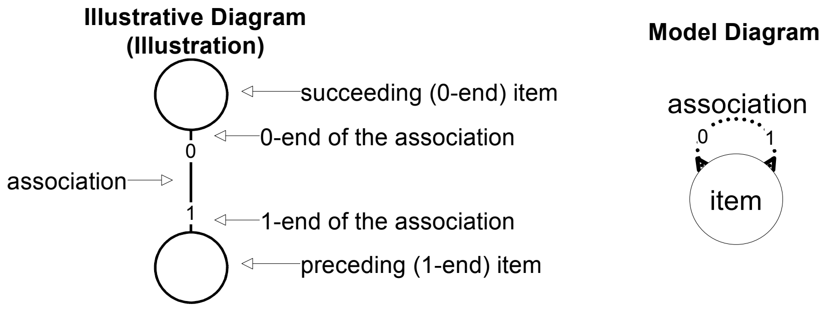

Figure 1. Illustrative and Model Diagrams with Element Display Convention

Here and elsewhere, parentheses surround a redundant portion of a type name that has been included for clarity, e.g., "preceding (1-end) item" and "(stored or implied) element." In addition, in a compound type name, one of the subtype names may be used in place of its supertype, e.g., "preceding object" where "object" is a subtype of "item."

3.4 Items and Association Subtypes

What an item stands for provides a basis for subtyping items as shown in the following taxonomy. An underlined term in this and subsequent taxonomies without a stated definition indicates that it is defined by its subtypes (e.g., instance is a supertype of object and value).

- item
  - instance
    - object – an item that stands for a thing
    - value – an item that stands for a property
  - concept
    - class – an item that stands for the concept of a thing
    - metric – an item that stands for the concept of a property

The associations among these item subtypes consistent with the above notions are as follows:

- An object is *classified by* a class and may *be valuated by* one or more values;
- A class may *classify* one or more objects and may *be valuated by* one or more metrics;
- A value is *classified by* a metric and may *valuate* one or more objects;
- A metric may *classify* one or more values and may *valuate* one or more classes;
- An item having a given item subtype (1) may be *composed* of one or more other items with the same item subtype and (2) may *compose* one or more other items with the same item subtype.

Figure 2 displays two expanded forms of the model diagram shown earlier in Figure 1 incorporating the item subtypes defined above. Also incorporated is the cardinality/optionality of associated items as stated in the above notion by using crowfoots and dashed vs. solid lines consistent with conventional data modeling notation. Each element depicted is *any* (stored or implied) element of the displayed element subtype. These diagrams also serve to define and name the depicted association subtypes (e.g., a classification association is an association with a concept on its 0-end and an instance on its 1-end). The left model diagram depicts the element subtypes based on both the classification and valuation; these element subtypes are called the basic element subtypes in order to identity and distinguish them from additional elements subtypes introduced later. The right model diagram depicts the element subtypes based on classification alone.

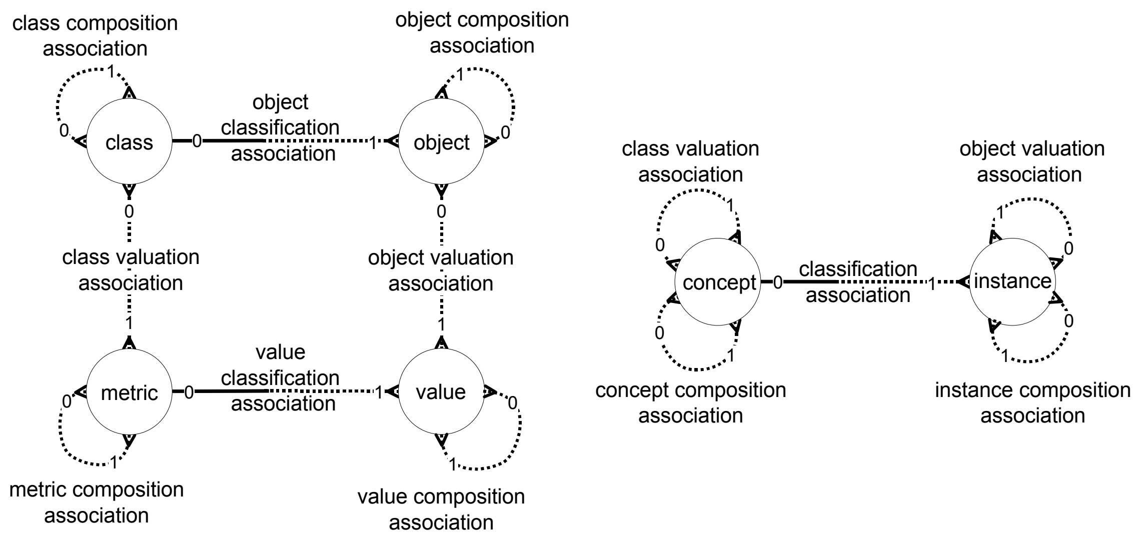
class composition association
object composition association
class
object classification association
object
class valuation association
object valuation association
metric
value classification association
value
metric composition association
value composition association
class valuation association
object valuation association
concept
classification association
instance
concept composition association
instance composition association
0
1

Figure 2. Two Expanded Model Diagrams

The basic association subtypes named and defined in Figure 2 along with their supertypes are identified in the following taxonomy of association subtypes.

- association
  - non-composition association (an association having different item subtypes on each end)
    - classification association (an association that classifies)
      - object classification association – an association that classifies an object by a class
      - value classification association – an association that classifies a value by a metric
    - valuation association (an association that valuates)
      - object valuation association – an association that valuates an object by a value
      - class valuation association – an association that valuates a class by a metric
  - composition association (an association having the same item subtypes on each end)
    - instance composition association
      - object composition association – an association that specifies composition of objects
      - value composition association – an association that specifies composition of values
    - concept composition association
      - class composition association – an association that specifies composition of classes
      - metric composition association – an association that is specifies composition of metrics

Extending the above association naming convention to items, the preceding (1-end) item of a given composition association is called a composing item and the item on the other end (the succeeding item) is called a composed item. The term "composition" is used broadly to encompass all of the ways that two items with the same basic item subtype (e.g., two objects) can be associated. Some commonly used terms for the term composition (as used here) are: inclusion, aggregation and belonging; some terms for composed (succeeding) items are: group, whole, collection, product, sum, aggregate, heap and set; and some terms for composing (preceding) items are: component, part, portion, constituent and member. The different notions of composition suggested by these terms are discussed in later sections.

Notion: The multiple items composing the same item are disjoint (i.e., they do not overlap). The zero or more items composing the same item are complete (i.e., all items are specified).

4. IMPLEMENTATION

The storage method discussed above appears to be applicable to all information in a brain or an information device. However, a human mind has greater capability than an information device for certain forms of information, e.g., facial features. In addition, an information device has greater capability than a human mind to store other forms of information, e.g., numbers and long text strings, as well as bit strings that generate sounds and images (what is commonly referred to as data). The greater capability of the former is not yet fully understood; as demonstrated in the remainder of this paper, the greater capability of the latter is provided by the attributes of elements. The remainder of this paper introduces element attributes, and discusses how elements and their attributes can be implemented in a platform called the I-A platform. The item identifier and content attribute are discussed in this section; the remaining attributes are discussed later.

A brain stores associations by physical connections; today's information devices store associations by various non-physical methods. One of the methods of implementing the I-A platform in today's information devices (and the one most easily envisioned) is to store each element in a separate record with the item identifier(s) and attribute(s) applicable to the subject element subtype. However, nothing discussed here is dependent on that implementation method.

4.1 Item Identifier

Each stored item must be uniquely identified across all I-A platforms that are mutually accessible. That identifier is a bit string called an item identifier abbreviated item-id. When useful, an item-id is expressed by a bold uppercase letter (e.g., **A**) displayed at the upper left of an item circle. Sometimes it is indexed by apostrophes or a backslash to indicate a range of items (e.g., **A'**, **A''** … **A**). An attribute abbreviation or a parameter also may be indexed in this manner.

4.2 Content Attribute

The first element attribute is content, an attribute of a stored item that is a bit string or absence thereof that can be interpreted as a character string, sound, picture, etc. Table 1 describes this attribute and tabulates its taxonomy. Content is abbreviated by "c", null content is displayed as shown in the table and "c" is a parameter for non-null content.

Table 1. Content Subtypes and Their Use as an Attribute

taxonomy of content (and abbreviation)		what is stored	display convention
content (c or blank)	null content	nothing	' '
	non-null content (c)	one or more bits	(see following discussion)

The non-null content of a class, metric or value is always a character string that is a modified or unmodified noun. The non-null content of an object is interpreted according to its class, e.g., an object whose content is a jpeg file would have "jpeg file" as its class. The non-null content of a concept is the name given to that concept and is called concept name. The following is an example of non-null character string content for each item subtype along with the convention used to display it in diagrams, expressions and text.

item subtype	display convention	example
class	bold font	**Person**
metric	italicized font	*Time*
object	within single quotes	'We hold these truths…'
value	within single quotes	'kilogram'

The content of a stored item is never unknown; null content specifies that there is no content as demonstrated in later examples. Unknown content is specified by the fact that the subject item is not stored. Cardinal numbers (numeric characters specifying a count) are not stored as content. Cardinal numbers are discussed in §9.2.

In diagrams, content or a parameter for that content is displayed inside the item circle. In text, an instance is expressed by its concept name and subtype, e.g., "**Person** object" and "*Time* value." When discussing what an instance stands for, it is written in the conventional manner, e.g., "person" and "time." In a similar manner, a value may be expressed by its metric and value with a delimiting colon, e.g., "*Color*: 'blue'."

4.3 Traits

An item's content, reality (discussed later), composition with other items, and (for instances) concept determines the information it specifies. With respect to a given item, each of these is called a trait. An instance's traits specify the storable characteristics of the thing or property it stands for. (An item's item-id, and where and when that item is stored are not traits of that item because they can be changed at will without affecting the information specified). Examples of traits are an instance being composed of/by other things (e.g., having/not having legs), objects having values (e.g., a person's height) and the object's class (which may identify non-storable characteristics). The following lists the traits that an item of a given item subtype may or must have. Except for an objects' valuating values and composing objects (as noted with an asterisk), each trait is presumed to be specified. The first four listed traits are referred to as non-concept traits.

class traits	metric traits	object traits	value traits
content	content	content	content
class reality	metric reality	object reality	value reality
composed classes	composed metrics	composed objects	composed values
composing classes	composing metrics	composing objects*	composing values
valuating specific		valuating values*	valuated objects
		associated class	associated metric

4.4 Item Domains

Those disjoint items standing for things, properties or concepts form a domain called an item domain. The items in an item domain may or may not be countable. An item domain specified in an I-A platform is one of the following subtypes (shown below with an example) based on the extent to which items are stored or implied.

item domain subtype	how specified	example objects in an object domain
fully-itemized	all items are stored	people who have been a US president
partially-itemized	items are stored and implied	living people (those I know plus all others)
un-itemized	all items are implied	pennies in my piggy bank

The items in a fully-itemized item domain are usefully expressed in a linear expression that lists each item and delimits them by "*or*", where "or" is an "exclusive or," That expression is called an OR expression that has the form: [θ' *or* θ'' … *or* θ\], where θ identifies an item by its item-id, content or (for an object) a distinguishing value. For example, [**Woman** *or* **Man**] is the OR expression for the class domain for the class: **Person** where θ is each subclass's content. (A diagram of this example is presented in §6.2.4). In diagrams, an OR expression may be displayed to the immediate right or left of an item to which that expression applies.

5. FURTHER ITEM SUBTYPES

This section introduces the subtypes that can be understood without regard to an item's being composed of/by other items. Each of these item subtypes is independent of the others, and some correspond to a distinction expressed by words and lexical components commonly used in the English language. (Item subtypes that arise from composition are introduced in §6).

5.1 Representational and Non-representational Instances

As described earlier, an instance either represents a thing or value or is a thing or value. The instance subtypes based on this distinction (i.e., whether or not an instance has an counterpart in the external world) are representational instance and non-representational instance. Examples of this distinction are discussed with concepts in §7.2.

5.2 Singular and Plural Items

Notion: An instance either (1) stands for one thing or one property, or (2) more than one thing or property. A concept either (1) classifies an instance that stands for one thing or one property, or (2) classifies an instance that stands for more than one thing or property.

The item subtypes based on the distinction identified in the above notion are singular item and plural item.

A concept's content (concept name) is either null or a noun that is one of the following: singular noun, plural noun, or either singular or plural noun (e.g., fish). A concept is either singular or plural irrespective of its concept name and is specified as such (as discussed in §9.8). Some classes have no plural class because they are uncountable. One example of such a classes is **Furniture**, where the noun: furniture is an "uncountable noun" (i.e., having no plural form). Quantitative values, i.e., values having units of measure, use the same noun for the singular and plural metric. However their units of measure do have singular and plural forms, e.g., 'gram' and 'grams.' The following are examples of singular and plural concepts and units of measure for different cases.

concept subtype	example class names				example metric names	
singular concept	**Truck**	**Radius**	**Fish**	**Furniture**	*Name*	*Mass* (unit of measure is'gram')
plural concept	**Trucks**	**Radii**	**Fish**	–	*Names*	*Mass* (unit of measure is'grams')

5.3 Particular and Common Instances

A singular object stands for a thing that (1) is individually identifiable or (2) is any one among multiple things. The following notion generalizes this singular and plural instances.

Notion: The *N* things or properties that an instance stands for are either known or unknown. If unknown, the instance stands for *any N* among *N* +1 or more things or properties that are disjoint, where *N* is one or more.

An instance that stands for one or more *known* things or properties is called a <u>particular instance</u>; an instance that stands for one or more *unknown* things or properties is called a <u>common instance</u>. The "*N* +1 or more things or properties that are disjoint" (in the above notion) is an instance domain.

Incorporating the singular and plural instance subtypes developed above, what a particular or common instance stands for is as follows:

- particular (known) singular instance – one thing or property;
- common (unknown) singular instance – *any* one thing or property;
- particular (known) plural instance – two or more things or properties; and
- common (unknown) plural instance – *any* two or more things or properties.

A common instance has an instance domain of two or more instances. A particular instance has no instance domain and is commonly referred to as being unique. The traits that determine if an instance is particular instance or common instance are discussed in §6.2.4.

Particular and common instances are most easily understood for singular objects. Mapping is sometimes used to describe the correspondence between what is stored in an information device and what it stands for. A particular singular object is mapped to the thing it stands for; a common object cannot be similarly mapped. A particular singular object is often uniquely identified by a unique *Identification* value (e.g., an *SSN* value) or *Space-time* values specifying a unique position in space and time; a common object is not similarly identified.

Plural instances and values are understood as follows:

- Plural objects – A plural object is a particular plural object if all of its composing objects are particular because what it stands for is known; if one or more are common, it's a plural common object; and
- Values – A value valuating a particular object is a particular value because what it stands for is known; otherwise it is a common value.

In the English language, the presence/absence of articles and counts identifies an object as particular or common. A particular object is identified by the definite article "the"; a common object by the indefinite articles "a" or "an". A common object is also identified by the absence of an article or the presence of a count. Examples are shown below for objects classified by **Truck** and **Furniture**:

- "the truck" – a particular singular **Truck** object representing an identified truck;
- "a truck" – a common singular **Truck** object representing *any* truck;
- "the trucks" – a particular plural **Truck** object representing multiple identified truck;
- "three trucks" – a common plural **Truck** object representing *any* three trucks;

- “the furniture” – a particular singular **Furniture** object representing identified furniture; and
- “furniture” – a common singular **Furniture** object representing *any* furniture.

5.5 Specific and General Concepts

The following notion for concepts is a complement of the previous notion for instances.

Notion: The characteristics identified (as applicable/non-applicable) by a concept are either fully known or partially known.

A concept whose characteristics are fully known is called a specific concept; a concept whose characteristics are partially known is called a general concept. The traits that determine if a concept is a specific concept or general concept are discussed in §6.2.4.

Knowledge of the characteristics of an instance is the basis for classifying it. For an object, we may or may not know or care to specify all of its characteristics. When an object’s characteristics are sufficiently known and of interest, we classify it by a specific class; when there is absence or disinterest about some of its characteristics, we classify it by a general class. For example, when we classify an object by **Person**, we are specifying that it is unknown or of no interest whether that person is a man or a woman. For a value, we always know all of its characteristics so that we always classify a value by a specific metric.

5.4 Existence

Axiom: A thing is perceived to exist continuously within a single finite interval in space and time.

existence – a continuous finite space-time interval during which a thing is perceived to exist.

Axiom: A thing’s existence is composed of three orthogonal spatial existences and a temporal existence (each commonly referred to as a dimension).

Axiom: The existence of an internal thing is where and when it is stored. The existence of an external thing in a dimension is relative to two other things in that dimension.

Spatial existence of an external thing in a given dimension is specified by *Length* values; temporal existence is specified by *Time* values. A stored *Time* value is one of the following:
- One named *Time* value found in the *Time* value domain, e.g., ‘2001’ and ‘Jurassic’; or
- One or two unnamed *Time* values that are each one of the following:
    - A start *Time* value, a *Time* value at the start of a temporal existence; or
    - An end *Time* value, a *Time* value at the end of a temporal existence.

(The method of specifying a named *Time* value, a start *Time* value, and an end *Time* value in the *Time* value domain is not addressed here). The smallest observable *Time* interval is called a moment in time. The current moment in time is a *Time* value whose content is ‘Now.’

The object subtypes based on temporal existence and their *Time* values are as follows:
- historic object – an object that has a stored end *Time* value or a stored named *Time* ending earlier than ‘Now’;

- future object – an object that has a stored start *Time* value or a stored named *Time* starting later than 'Now'; and
- current object – an object that is not a historic object or a future object.

A current object has stored or implied *Time* value(s) starting earlier than 'Now' and ending later than 'Now'. Note that an object not valuated by a stored *Time* value is a current object.

Since a composed thing succeeds those that compose it, the existence of a composed thing cannot be either earlier or later than any of its composing things.

5.6 Duplicates

Two or more items that have the same traits are indistinguishable from one another – they are duplicates, each called a duplicate item. Since a class, metric and value are each presumed to have all of their traits specified, duplicate classes, metrics and values must stand for the same concept or property. The I-A platform is presumed to be implemented to store such duplicates as identical items as discussed in §6.2.2 (identity) or to delete all but one.

In contrast, an object need not have all of its traits stored. Duplicate objects that stand for different things can arise whenever we are unable or do not care to (1) classify using a specific class, (2) store all values or (3) store composing objects. Such duplicates are not, per se, invalid; however storing such duplicates is neither useful nor necessary. The I-A platform is presumed to be implemented as follows:
- If it is known that the duplicate objects stand for the same thing, then treat them as discussed above for non-object duplicates;
- If it is known that the duplicate objects stand for different things, then specify them by other means as discussed in §9.2 (count); and
- If there is no determination of which of the above applies, then specify them as possible duplicates (not discussed here).

6. STRUCTURES

Items interconnected through associations can be thought of as forming a 3-dimensional structure. Since all items in the I-A platform are interconnected, all elements are part of a single structure called an information structure, or simply structure. We can only understand a small portion of an information structure at a time. A portion of an information structure whose elements are of interest is called an information substructure, or simply substructure.

A substructure or portions thereof can be typed by the information specified. For example, a substructure that specifies hierarchy information is a hierarchy substructure. A taxonomy of information subtypes has hierarchy information as a subtype. All substructures that specify a given type of information have a unique combination of one or more of the following: basic element supertypes (e.g., instance), ranges of attributes and ranges of composed/composing items (i.e., optional items and their composition associations). Such a unique combination is called an information pattern, or simply pattern. A pattern is named for the type of information it describes, e.g., hierarchy (information) pattern. A substructure matching a pattern specifies that type of information, e.g., a substructure matching a hierarchy pattern is a hierarchy substructure that specifies hierarchy information. A pattern cannot be directly stored in a platform but in some cases may be displayed in a single diagram. Examples of displayable patterns are shown in Figures 6, 13 and 17.

The fourth diagram type is like a pattern in that it describes a range of substructures using a combination of basic element supertypes, ranges of attributes and ranges of composed/composing items; unlike a pattern it is not named. This diagram type is used for examples and is called an exemplar diagram or simply exemplar. Exemplars are shown in Figures 4, 5 and 9.

6.1 Non-composition Substructures

A non-composition substructure has each of the four basic item subtypes and the four basic non-composition association subtypes and specifies non-composition information. Figure 3 displays the same non-composition substructure using three different display conventions used here, each specifying that a woman named 'Dot' currently exists.

> The substructure on the left applies the display convention described earlier but with a non-composition association displayed by a thin line with no indication of the end-id (end-id can be determined from the item subtype on each end). In that substructure, Class **A** with content: **Woman** classifies object **B** with null content (' '); metric **C** with content: *Name* classifies value **D** with content: 'Dot'; metric **C** (*Name*) valuates class **A** (**Woman**); and value **D** ('Dot') valuates object **B**. The line depicting the association from metric to class is dashed to indicate that the association is implied (as discussed in §7.3). The item subtype is displayed to the lower left of the item circle when useful.
>
> The middle and right diagrams display the same substructure using call-outs, orientation and text formatting to display the same information. The graphically omitted elements can be readily determined from those displayed.

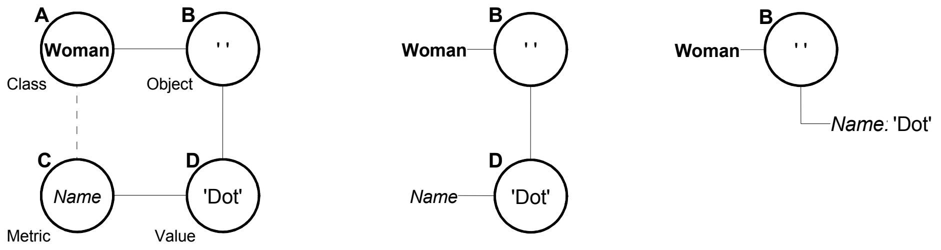

Figure 3. Alternative Displays of a Non-composition Substructure

If the information shown in Figure 3 was stored in a relational database, it would likely be in a PERSON table with Name and Gender columns. A PERSON table row would stand for the object. The Gender column would specify the subclass of the person. If a surrogate key were used, it would be equivalent to the item-id of the object, e.g., **B**. A relational database can only store information having a predetermined organization with little of that organization directly stored by associations. The concept names: "Person" and "Name," as well as "Gender," would not be stored in the PERSON table; they would be stored in a schema external to the data elements in the table. None of the non-composition associations are stored; they are implied by the database system's encoded logic.

6.2 Composition Substructures

A composition substructure has one or more composition associations. Unlike non-composition substructures, composition substructures allow many different ways for an item to compose or be composed, thereby giving rise to a variety of different patterns. The two exemplars in Figure 4 introduce additional display conventions that are applicable to substructures and patterns as well.

The left diagram introduces the convention for depicting a composition association. An element is displayed as discussed in §3.3. The ends of a composition association are distinguished by the small circle at the 0-end.

The right diagram introduces the convention for items that are omitted for brevity. In this diagram, item **A'** is composed of item **B'** and **B''**, and composes no items; item **B'** is composed of two un-displayed items and composes item **A'**; and item **B''** is uncomposed, and composes item **A'** and one un-displayed item. (The indexes on the item-ids, e.g., **B'**, **B''**, are used to simplify the display of different item-ids; they do not identify or suggest a sequence).

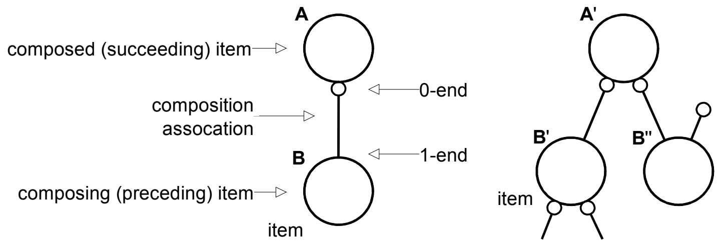

Figure 4. Conventions for Substructures, Patterns and Exemplars

Subsequent diagrams use a thin dashed line as follows: in a pattern and an exemplar, it indicates that the element is optional (as show in Figures 6 and13); in an illustration, it indicates that it is implied by stored elements (as show in Figures 8 and 23).

The following subsections discuss four of the simplest patterns and substructures that are found in more complex patterns and substructures.

6.2.1 Replication Information

The left diagram in Figure 4 (above) specifies that item **A** is composed of item **B** only. Being composed of a single item, item **A** is interpreted as a replicate of item **B**, i.e., item **A** is a subsequently-stored item that is the same as item **B**. Such a substructure is called a replication substructure with item subtypes: replicate item and replicated item. A replicated concept is a subsequently-created synonym with the same context (e.g., the same concept in the same language). Examples are displayed in Figures 7, 9, 10 and 21.

6.2.2 Identity Information

Figure 5 displays two exemplars for the same stored elements using different display conventions. The left diagram displays two items (**A** and **B**) that each compose the other through two composition associations with opposite directionality. Since two items each succeeding the other is an absurdity, they must simultaneously compose one another. These two items are interpreted as standing for the same thing, property or concept; each such items is called an identical item and is found in a substructure is called an identity substructure. Identical instances must have the same concept. Identical items are not duplicates because they have different traits, i.e., no two identical items are identical to the same items. Two identical concepts are synonyms with different context (e.g., the same concept in different languages). The two associations are conveniently discussed and displayed as if they were a single association called an identity association. The right diagram depicts an identity association by a single line. Figure 9, 16, 18-22 and 25 display identity associations and identical items.

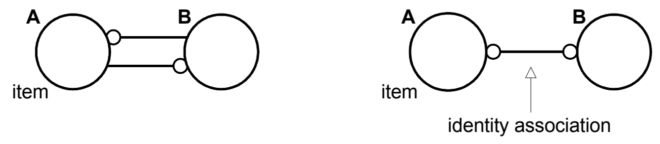

Figure 5. Exemplars with Identical Items

The distinction between identity and replication is the same as the following distinction found in mathematics where x, y, a, b and c are algebraic variables:

If y = (a+b)c and x = ac+bc, then y ≡ x, i.e., x and y are identical; and

If y = f(x) and f(x) is x, then y = x, i.e., y is dependent upon (and a replicate of) x.

6.2.3 AND Information

The two diagrams shown at the top of Figure 6 are discussed below.

The top left diagram is a pattern that describes item **A** as composed of multiple items: **B**'… **B**\. (Ellipsis indicates that there are zero or more additional items. Multiple items in a range identified by the same letter are referred to by that letter, e.g., the **B** items). Section 3.4 defined composition broadly to encompass all of the ways that two items with the same basic item subtype can be associated. Previous axioms stated that multiple items composing a given item are disjoint and complete. Given these understandings, this pattern describes information commonly expressed in English by the conjunction "and," i.e., item **A** is composed of item **B**' *and* ... item **B**\. This pattern is called the AND pattern that describes AND information ("AND" is capitalized for clarity). Item **A** is called an and-item. The expression displayed to the right of the and-item (**A**) lists the composing **B** items. This expression, called an AND expression, is similar to the OR expression but has the form: [θ' *&* … θ\], where θ identifies an item (as described earlier for the OR expression) and the italicized character "*&*" is a delimiter denoting "and." An and-instance may or may not be a plural instance as discussed later.

The right diagram is an object substructure that matches the AND pattern and specifies that a **Man** object and a **Woman** object compose the plural **Persons** object. The and-object (**A**) is a plural object that has a classification association with the plural class: **Persons**. Additional examples are presented in §8.5. Note that if objects **B**' and **B**" are particular, then the and-object (**A**) is a particular object by virtue of its composing objects being particular objects; otherwise it would be common.

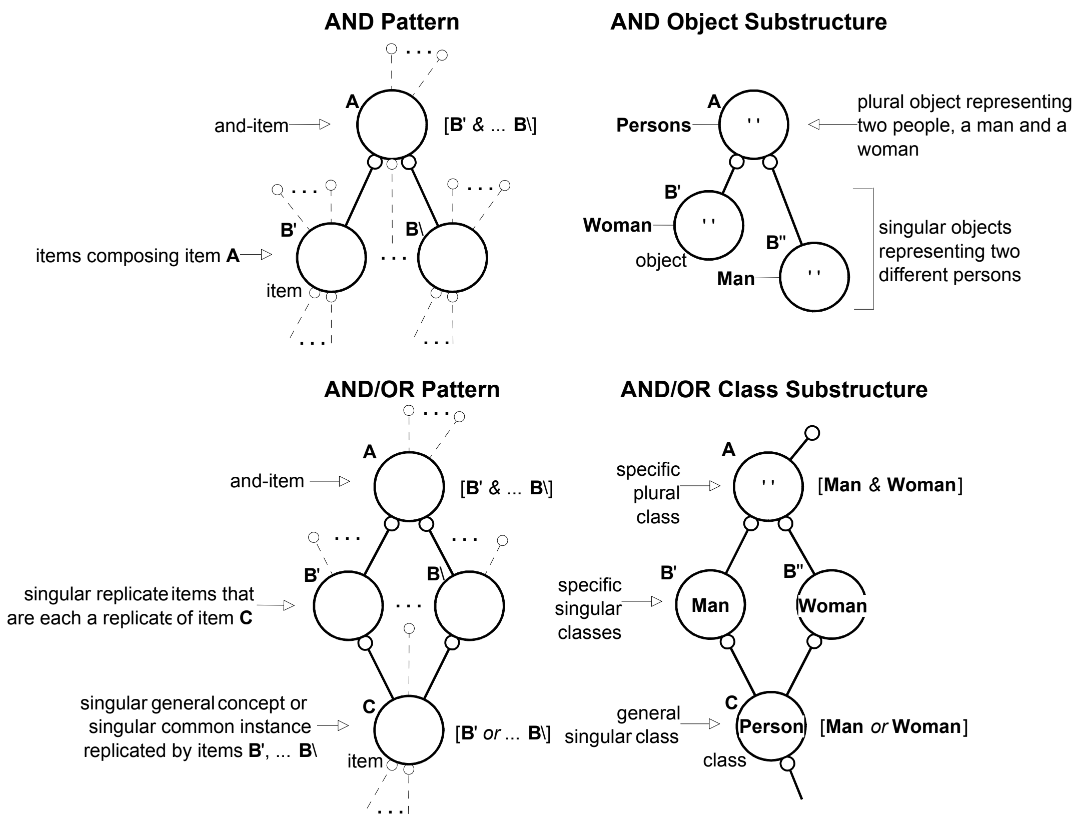
AND Pattern
AND Object Substructure
A
and-item
[B' & ... B\]
B'
B\
items composing item A
item
Persons
plural object representing two people, a man and a woman
B'
Woman
object
B"
Man
singular objects representing two different persons
AND/OR Pattern
AND/OR Class Substructure
A
and-item
[B' & ... B\]
B'
B\
singular replicate items that are each a replicate of item C
C
singular general concept or singular common instance replicated by items B', ... B\
[B' or ... B\]
item
specific plural class
[Man & Woman]
specific singular classes
Man
Woman
general singular class
Person
class
[Man or Woman]

Figure 6. Patterns and Substructures for AND Information and AND/OR Information

6.2.4 AND/OR Information

The two diagrams shown at the bottom of Figure 6 are discussed below.

> The bottom left diagram in Figure 6 is an AND pattern with an added singular item **C** that replicates each **B** item. Since the **B** items are disjoint and complete with respect to the and-item (**A**), they are also with respect to item **C**. Since the **B** items, being disjoint, cannot each be a replicate standing for the same thing, property or concept, item **C** must stand for *any* one of those multiple things, properties or concepts as discussed in §4.4. Thus item **C** is either a common instance or a general concept and the **B** items are the item domain for item **C** as indicated by its OR expression. The portion of this pattern where **C** item composes the **B** items is called an OR pattern that describes OR information. When combined with the AND pattern, it is called an AND/OR pattern that describes AND/OR information.
>
> The right diagram displays a singular AND/OR class substructure that matches the AND/OR pattern and specifies that **Man** and **Woman** are each a replicate of **Person**. The classes **Man** and **Woman** are disjoint. Thus **Person** is the class name of a general class for the subclasses **Man** and **Woman**, expressed as: [**Man** *or* **Woman**]. Additional AND/OR instance substructure examples are shown in Figures 12, 15 and 16.

Based on the above discussion, the trait that determines if a concept is a specific concept or a general concept is that only the latter composes multiple singular replicate concepts (either directly or indirectly). Similarly, the trait that determines if an instance is a particular instance or a common instance is that only the latter composes multiple (singular or plural) replicate instances (either directly or indirectly).

7. CONCEPT STRUCTURES AND SUBSTRUCTURES

This section discusses the concept structures for singular (specific and general) concepts and how concepts can be compared to further type instances and instance composition associations.

7.1 Concept Substructures

We create new singular classes and non-quantitative metrics whenever we identify new characteristic(s) and we want to distinguish among instances that have/don't have those new characteristic(s). Each new concept identifies the combination of (1) the characteristics applicable/non-applicable to the preceding concept and (2) those new characteristic(s) applicable/non-applicable to the new concept. (The characteristics of the preceding concept identified by a succeeding concept are commonly referred to as being inherited from the preceding concept). When we create those concepts, we create succeeding concepts from a preceding concept. Consistent with the conventional meaning of the prefixes "super" and "sub": (1) with respect to a preceding concept, its succeeding concepts are each called a subconcept and (2) with respect to each subconcept, its succeeding concepts are each called a superconcept.

Creating new singular concepts in this manner results in the following:
- Each concept has zero or more subconcepts and zero or one superconcept;
- Subconcepts are complete and disjoint with respect to their superconcept;
- There is one concept that is the superconcept of all concepts;
- A specific concept is a subconcept that has no subconcepts and no concept domain;
- A general concept is a superconcept that has subconcepts and a concept domain;
- A concept domain contains each of the subject superconcept's specific subconcepts; and
- The organization of the concepts is a hierarchy.

A concept that is the superconcept of all concepts is called a super-most concept. The super-most class has the class name: **Object**; the super-most metric has the metric name: *Value*. A general concept can be expressed by the OR expression where θ is the content of a subconcept (as presented in the earlier example for [**Woman** *or* **Man**] for the general class: **Person**).

Figure 7 displays different portions of a class structure for various cases to demonstrate how singular concepts are specified. Class **A** replicates one class: class **A'**; classes **B**, **C** and **D** are general classes in OR class substructures because they each replicate multiple singular classes. The content of class **B** (**Person**) is understood as the name given to the general class with the expression: [**Man** *or* **Woman**]. The content of classes **C** and **D''** are null because neither has a name in the English language. Each of the three expressions describes the subject class in terms of its composing and composed classes. Note that the sequence of the classes is the same as suggested by the notion of inheritance.

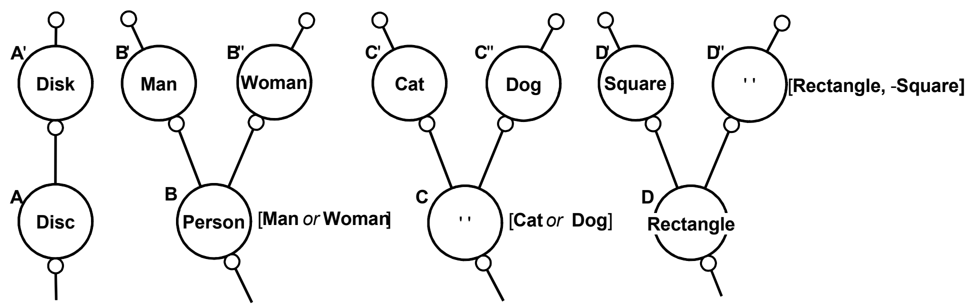

Figure 7. Example Singular Class Substructures

.

7.2 Upper Classes

As discussed earlier, the super-most (general) concepts have the content: **Object** and *Value*. Also what an object stands for either represents an external thing or is an internal thing. Thus the two subclasses of **Object** are:

- **Physical object** – subclass of **Object** that is valuated by *Mass-energy*; and
- **Non-physical object** – subclass of **Object** that is not valuated by *Mass-energy*.

For clarity, "**Physical object** object" is written "**Physical** object."

An object that stands for an arbitrary group of non-interacting things (see §8.1) is a non-representational object because it has no counterpart in the external world whether the group of things are internal things or external things. Thus a **Physical** object is either a representational **Physical** object or a non-representational **Physical** object. Respective example subclasses are listed below.

representational **Physical** object	**Automobile**
non-representational **Physical** object	**Automobile**s
non-representational **Non-physical** object	**Contract**

One additional metric of note is *Identification*, a general metric whose specific submetrics (e.g., *Name*, *VIN* and *SSN*) classify a value used to identify an object. We can observe that identification information is an internal property of (internal or external) objects. Thus *Identification* valuates the class: **Non-physical** object. Thus a **Person** object is an internal thing because it has a *Name* value; a **Human body** object in which a person resides (and having a different temporal existence) represents an external thing and does not have an *Identification* value.

7.3 Concept Specification

In a given I-A Platform, all concepts can be specified in one class structure and one metric structure. Each such structure has an and-concept composed of each singular specific concept; that and-concept is a specific concept called a global specific concept. That concept specifies that all singular specific concepts are disjoint and complete. The example in §10.2 displays a global specific class.

Figure 8 displays the same specification of concepts using two different display conventions for the upper concepts introduced above. The global specific concepts are omitted for simplicity. For the purpose of this figure only, three of the metrics are taken to be specific metrics. The diagram on the left displays classes and metrics in their respective substructures with specific metrics valuating classes through class valuation associations. The diagram on the right simplifies the display by omitting general metrics and using the display convention introduced earlier (see Figure 3).

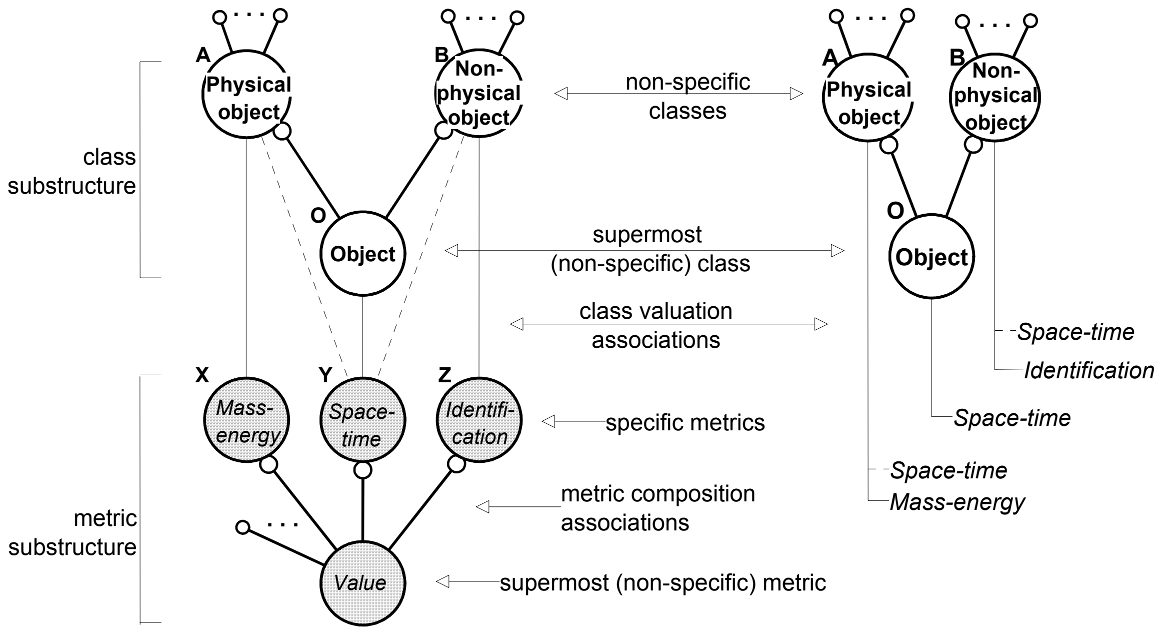

Figure 8. Alternative Displays of Singular Class and Metric Structure

Note the following:

- Only the specific metrics valuate a class;
- Each specific metric valuates one or more classes;
- Each class is valuated by one or more metrics;
- *Space-time* valuates **Object** and, therefore, valuates all objects; and
- All other specific metrics valuate subclasses of **Object**.

Since (1) a metric specifies a characteristic identified by a class and (2) characteristics are inherited by each of its (succeeding) subclasses, valuating a class by the metrics of its preceding class can be implied rather than stored. The I-A platform is presumed to be implemented to determine those implied valuation associations. Such an implied association is depicted by a dashed line as shown in Figure 8 and later figures (as well as earlier in Figure 3). As a result, each singular specific metric has one stored class valuation association with one class.

7.4 Concept Comparison

Figure 9 displays a simple singular concept exemplar with the global specific concept omitted. A few of the concept subtypes are also displayed as well as descriptions of how four pairs of concepts compare. The expression for concept **A'** describes that concept in terms of its specific subconcepts, i.e., as the concept that stands for *any* of the specific concepts in its concept domain.

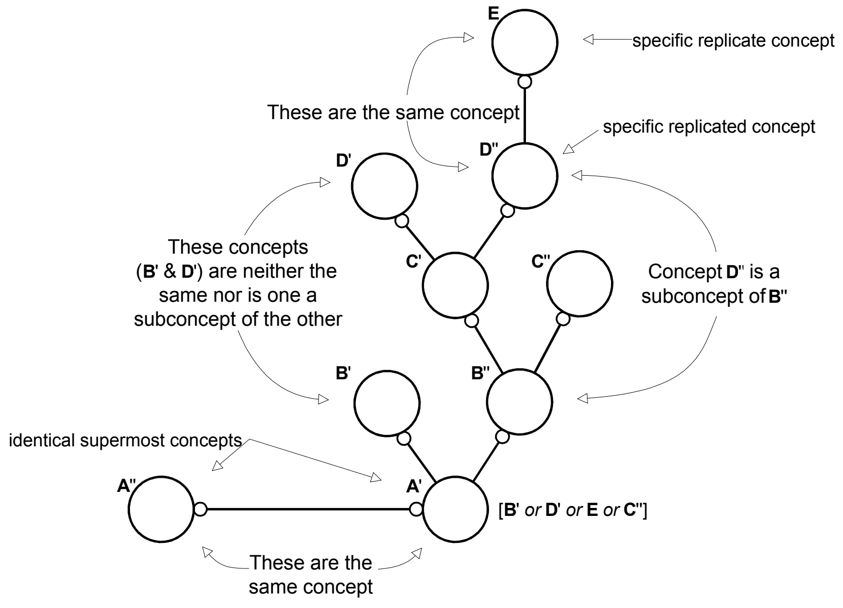
E
specific replicate concept
These are the same concept
specific replicated concept
D'
D''
These concepts (B' & D') are neither the same nor is one a subconcept of the other
C'
C''
Concept D'' is a subconcept of B''
B'
B''
identical supermost concepts
A''
A'
[B' or D' or E or C'']
These are the same concept

Figure 9. Singular Concept Exemplar with Concepts Compared

Considering only the relative composition of two concepts, there are four possible results when comparing two concepts. Those results are listed in the following taxonomy along with the item-id pairs of the concepts shown in Figure 9 that generates that result.

Taxonomy of concept comparison results	item-id pairs
different-concept	**D'** and **B'**
not-different-concept	
same-concept	**E** and **D''**, **A'** and **A''**
associated concept	
sub>super-concept	from **D''** to **B''**
super>sub-concept	from **B''** to **D''**

The result of comparing a concept to itself is same-concept. A comparison result where one or both concepts are plural concepts is the same as if all concept(s) are singular. Concept comparison provides a basis for typing elements as discussed in the next section.

8. INSTANCE SUBSTRUCTURES

Instance substructures are more complex than concept substructures and give rise to additional instance subtypes. The applicable axioms and instance subtypes are discussed below.

8.1 Subtypes Based on Concept Comparison

When the concepts of the two instances on the ends of an instance composition association are compared, the result of that comparison provides a basis for typing both that association and the instances themselves. The composition association subtypes based on concept comparison are shown in Table 2 along with its iconic abbreviation. In subsequent diagrams, an iconic abbreviation is displayed on the line depicting the instance composition association to indicate its subtype.

Table 2. Composition association Subtypes Based on Concept Comparison

taxonomy of instance composition association subtypes		concept comparison result	iconic abbreviation
definition association	instance summation association	same-concept	> <
	instance generalization association	sub>super-concept	> >
	instance specialization association	super>sub-concept	< <
production association		different-concept	< >

Note that a definition association has a concept comparison result of *not-different-concept*. The instance subtypes based on concept comparison that are of use here are limited to those based on *not-different-concept* and *different-concept*. Those subtypes are as follows:

defined instance – a composed instance having a *not-different-concept* than its composing instances, i.e., the instance on the 0-end of a definition association;
defining instance – a composing instance having a *not-different-concept* than the instance it composes, i.e., the instance on the 1-end of a definition association;
produced instance – a composed instance having a *different-concept* than its composing instances, i.e., the instance on the 0-end of a production association; and
producing instance – a composing instance having a *different-concept* than the instance it composes, i.e., the instance on the 1-end of a production association.

The terms "define" and "produce" are used to describe how a defining and producing instance respectively compose a defined and produced instance. A defined instance is a non-representational instance. The iconic abbreviation "**" is used for a definition association and absence of an iconic abbreviation is used for any composition association.

8.2 Produced and Defined And-instances

Axiom: A composed thing or property is either understood as having (1) no more than the characteristics found in its composing things or properties or (2) more than the characteristics found in its composing things or properties.

We can observe that an and-instance standing for the former composed thing or property is a defined instance and the latter is a produced instance. Examples of produced and-instances and defined and-instances are presented in the remaining subsections.

The AND expression convention is extended to recognize definition and production associations using the respective forms: [θ', θ", …θ\] and [θ' * θ" * …θ\]. See §10.1 for examples.

8.3 Interaction Among Composing Instances

Axiom: The multiple things or properties that compose a thing or property either interact (i.e., an action by either one affects the others) or do not interact.

We can observe that the instances composing a *singular* and-instance interact but those of a *plural* and-instance do not.

Interaction/non-interaction and production/definition are independent as demonstrated in the following for examples covering each case.

- A singular defined **Tree** and-object is defined by objects each standing for the same tree in different states that interact across their adjacent temporal existences. (Singular defined objects are discussed in the example in §10.4).
- A singular produced **Tree** and-object is produced by objects standing for its interacting roots, truck, limbs and leaves.
- A plural defined **Trees** and-object is defined by objects standing for different non-interacting trees (if they did interact, it would be a singular produced **Forest** object).
- A plural produced **Energy** and-value is produced by a quantitative **Force** value and a producing quantitative **Length** value that do not interact. (No examples for this case was found for objects).

8.4 Extended and Combined Instance Subtypes

The composed instance subtypes are organized in the following taxonomy. The distinctions among these subtypes discussed above are shown to the right of the type name along with examples for and-instances.

taxonomy of composed instance subtypes	distinctions among composed subtype		examples
composed instance			
replicate instance			
defined replicate instance	non-interactive and	=part	
produced replicate instance	non-interactive and	>part	
and-instance			
produced and-instance			
singular produced and-instance	interactive and	>sum of parts	dance couple
plural produced and-instance	non-interactive and	>sum of parts	4 gms *3 kms
defined and-instance			
singular defined and-instance	interactive and	=sum of parts	sum of states
plural defined and-instance	non-interactive and	=sum of parts	group of people

8.5 Composition Substructure Examples

Figure 10 displays examples of composed instance substructures with their composed instance subtypes and iconic abbreviations. The **Women** object (**C**') and **Man** object (**C**") interact to produce the singular **Couple** object (**C**). The **Women** object and the **Man** object (**B**' and **C**") do not interact; they simply define a plural **People** object (**B**). The *Energy* value (**F**) is the product of the interacting *Force* value (**F**') and the *Length* value (**F**"). The *Mass* value (**E**) is defined by (the mathematical sum of) the two non-interacting *Mass* values (**E**' and **E**").

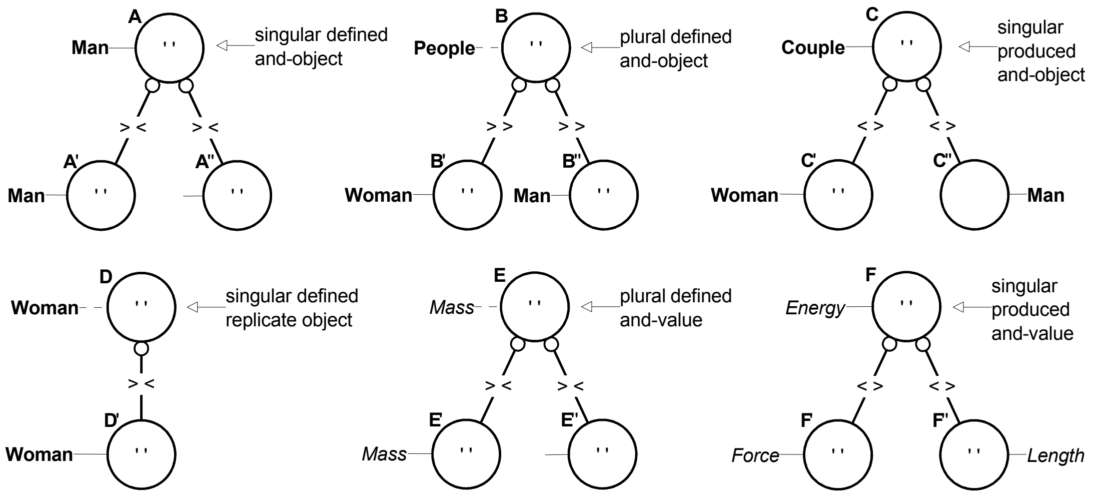

Figure 10. Substructures with Composed Instances

Note that the plural concept of the composed instances (instances **B**, **D** and **E**) can be deduced from the concept(s) of its composing instances. This allows the instance classification association for those instances to be implied rather than stored. The I-A platform is presumed to be implemented to validate those associations that are specified and determine those that are not.

9. REMAINING ATTRIBUTES AND THEIR APPLICATION

The remaining two stored attributes are discussed below with examples. An additional trait that is treated as a item attribute is also discussed.

9.1 Sign Attribute

Axiom: A thing, property or concept can stop composing another thing, property or concept if it previously composed it.

This axiom requires that we cognizes that composition and "uncomposition" are sequential. It also recognizes the absurdity of a thing, property or concept being less than nothing. If we specify composition with a composition association and then delete that association to specify its subsequent non-composition, that deletion loses information about their previous composition. We can preserve that information by instead adding a different association that specifies that the composing thing, property or concept no longer composes. Earlier, the terms: inclusion, etc., were used to describe the information specified by a composition association. Specifying the end of composition, or "uncomposition" is commonly described by the terms: subtraction, exclusion, removal, etc. An association that specifies "uncomposition" is called a –composition association; to insure clarity, the association previously called a composition association is now called a +composition association.

As demonstrated later, there are cases where we are ignorant of whether an association is either a +composition association or a –composition association. That association is specified by a third type of composition association called a +/–composition association.

The following is the taxonomy of composition association subtypes with the term "composition association" now defined by its supertypes. Also shown are commonly-used terms that suggest the composition association subtype.

taxonomy of composition association subtypes	commonly used terms			
composition association	—	—	—	—
known composition association	—	combination	—	—
+composition association	inclusion	addition	insertion	association
–composition association	exclusion	subtraction	extraction	disassociation
+/–composition association	—	—	—	—

With the exception of the case discussed in §9.6, the above notion constrains the use of a −composition association to those item pairs where there is a companion (stored or implied) +composition association.

The subtype of a composition association is stored as the composition association attribute called sign. Table 3 describes this attribute. The sign attribute is abbreviated "s", a known sign is either + or –, an unknown sign ("+/–") is a variable (also shown as an OR expression indicating that it is any known sign) and s is a parameter for a known sign.

Table 3. Sign Subtypes and Their Use as an Attribute

taxonomy of sign (and abbreviation)			display convention	information specified
sign (s)	unknown sign		"+/−" [+ *or* −]	It is unknown whether the 1-end item +composes or −composes the 0-end item.
	known sign (s)	inclusion	"+" or blank	The 1-end item +composes the 0-end item.
		exclusion	"−"	The 1-end item −composes the 0-end item.

The composition associations in previous diagrams were all +composition associations displayed with a blank sign. In subsequent diagrams, a sign is displayed inside the iconic abbreviation and in an expression to indicate the composition association subtype.

9.2 Count Attribute

A common instance and a count provide the means of specifying information in those cases where the instances are:

- Too numerous (people in a crowd or molecules in a mole);
- Not distinguishable from one another (cans of peas or units of mass); or
- Individually inconsequential (paper currency in a cash transaction).

This subsection describes how a count arises in a substructure with common instances. That description begins with an AND object substructure and then modifies it without changing the information it specifies. (Beginning with this figure, a particular instance is depicted in a diagram by light shading; a common instance by dark shading; and either common or particular by an absence of shading).

The beginning AND object substructure is displayed in Figure 10. This substructure specifies that three blue balls are included in a group. Each ball is represented by a singular **Ball** objects: **B**', **B**'' and **B**''' each valuated by *Color*: 'blue'. Together they define the plural defined and-object (**A**) that has the implied plural class: **Balls**. Each of the **B** objects is a common object because which ball it represents is unknown. Since they have the same traits they are duplicates. The duplicate **B** objects are presumed to be disjoint and complete with respect to object **A**. The AND expression for object **A** with sign incorporated is: [+**B**', +**B**'', +**B**'''].

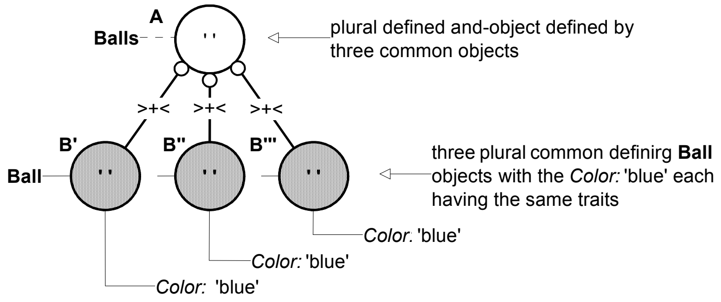
A
Balls
plural defined and-object defined by three common objects
>+<
>+<
>+<
B'
B''
B'''
Ball
three plural common definirg **Ball** objects with the *Color:* 'blue' each having the same traits
Color. 'blue'
Color: 'blue'
Color: 'blue'

Figure 11. AND Object Substructure with Defining Common Objects

The substructure in Figure 11 specifies the subject information and can be considered valid. However, storing common objects in this manner allows duplicates, can be unwieldy and is unnecessary. As discussed below and displayed in Figure 12, the same information can be specified using fewer elements and without duplicate instances.

The left substructure expands the beginning substructure by adding a singular common object **C** with *Color*: 'blue' that defines each of the three common **B** objects, and eliminating the three *Color* values that valuate those objects. The result is an AND/OR substructure where each **B** object is a replicate object replicating object **C**, and object **C** represents *any* of those three **B** objects. Object **C**'s object domain is [**B**' *or* **B**'' *or* **B**'''].

Since the **B** objects in the left substructure are each a replicate of the same common object, their use in defining object **A** in terms of object **C** is superfluous. The middle substructure simplifies the left substructure by eliminating each of the **B** objects and their associations, and replacing each of them with a single definition association directly from object **C** to object **A**.

The right substructure further simplifies the specification of this information by eliminating the three associations and replacing them with a single association that displays a count along with the sign. The count stands for how many times common object **C** defines object **A** and implies the following:

- The three replace associations with a 0-end on object **A**; and
- The three common duplicate **B** objects that define object **A** (shown in the beginning substructure and the AND/OR substructure).

Object **A** remains a plural and-object by virtue of the implied common duplicate objects.

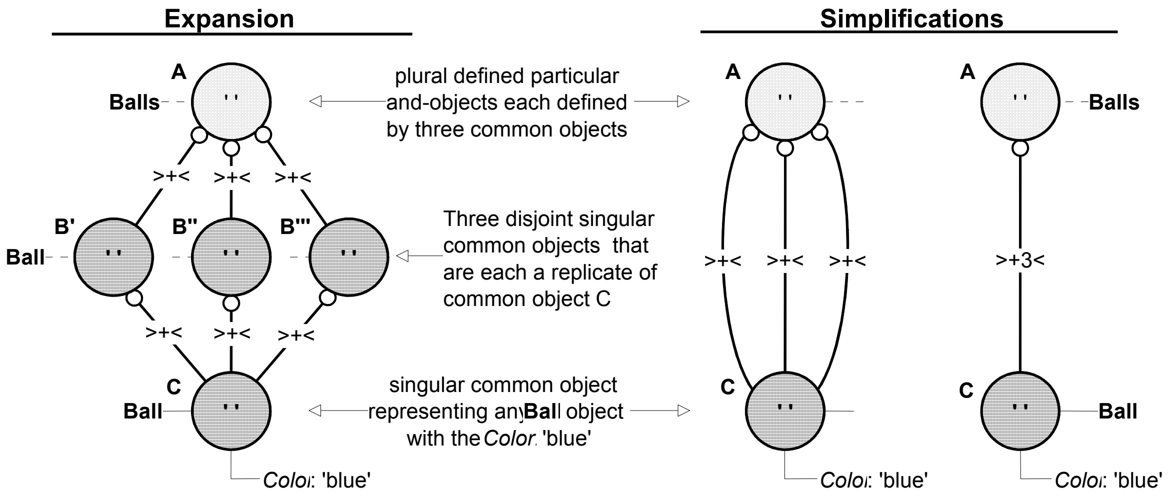
Expansion
Simplifications
A
Balls
plural defined particular and-objects each defined by three common objects
>+<
B'
B''
B'''
Ball
Three disjoint singular common objects that are each a replicate of common object C
C
Ball
singular common object representing any Ball object with the Color: 'blue'
Color: 'blue'
>+3<

Figure 12. Equivalent Object Substructures Specifying Three Duplicate Objects

The same argument applies to product AND instance substructures, as well as AND value substructures with either definition or production associations.

Incorporating count, the respective AND expressions applicable to object **A** in the two simplification substructures are [+**C**, +**C**, +**C**] and [+3**C**]. Since the substructures specify the same information, the four different **A** objects are identical to one another. An expression statement specifying that identity is as follows: [+**C**, +**C**, +**C**] ≡ [+3**C**]. The equivalent algebraic statement is the equation: **C** + **C** + **C** ≡ 3 **C**, where "+" is regarded as an operator. For value production, the expression statement is [+**C***+**C***+**C**] ≡ [**C**^+3] and the equivalent algebraic statement is: $\mathbf{C}*\mathbf{C}*\mathbf{C} = \mathbf{C}^3$, where "*" is an operator and the count is an exponent.

This simplification is implemented by storing an instance composition association attribute called <u>count</u>. Table 3 describes this attribute. The term "count," rather than number, is intended to indicate the non-applicability of sign. The count attribute is abbreviated by "n", a known count is a natural (cardinal) number, an unknown count ("null") is a variable (also shown as an OR expression indicating that it is any natural number) and <u>n</u> is a parameter.

Table 4. Count Subtypes and Their Use as an Attribute

taxonomy of count (and abbreviation)			display convention	information specified
count (n)	<u>unknown count</u>		"null" [1 *or* 2 *or* …]	The count of composition associations is unknown.
	<u>known count</u> (<u>n</u>)	one	"1" or blank	There is one composition association
		count >1 (<u>n</u>>)	not 1 and one or more numerals not beginning with 0	There are multiple composition associations.

The count attribute in previous diagrams were all displayed with a blank count standing for 1. In subsequent diagrams, a count is displayed to the right of the sign consistent with the conventional notation for an integer.

Note the following:

- Count arises entirely from the substructure simplifications discussed above consistent with the axioms and notions presented here;
- A substructure with a common instance provides the context for comprehending a count;
- Count and sign arise independently and are thus independent attributes;
- Count does not specify ordinality;
- Zero is not a count; and
- A sign can be viewed as specifying both (1) the type of operator (i.e., a composition association subtype) and (2) the sign of a non-zero integer.

Count simplifies the specification of a substructure by storing only one common instance rather than multiple duplicate instances having the same traits. The omitted instances are implied by the count attribute rather than stored. This simplification can be extended to an *information structure* as a whole by storing only one common instance having a given unique combination of traits. With this simplification, each stored instance has unique traits. The I-A platform is presumed to be implemented consistent with this simplification.

9.3 Composed Instance Patterns

Figure 13 displays patterns for instances composed of one or more other instances through a single stored composition association between the composed instance and each composing instance. The applicable sign and count attributes are also displayed. These patterns are distinguished by their instance composition association subtypes depicted by its iconic abbreviation. Each pattern has its information type and supertype name displayed above it. Each instance's identity associations are optional and are omitted for clarity. Note that one of the composition associations must have a + sign. Also note that if a count is >1, the composing instance is necessarily a common instance; otherwise it is either common or particular.

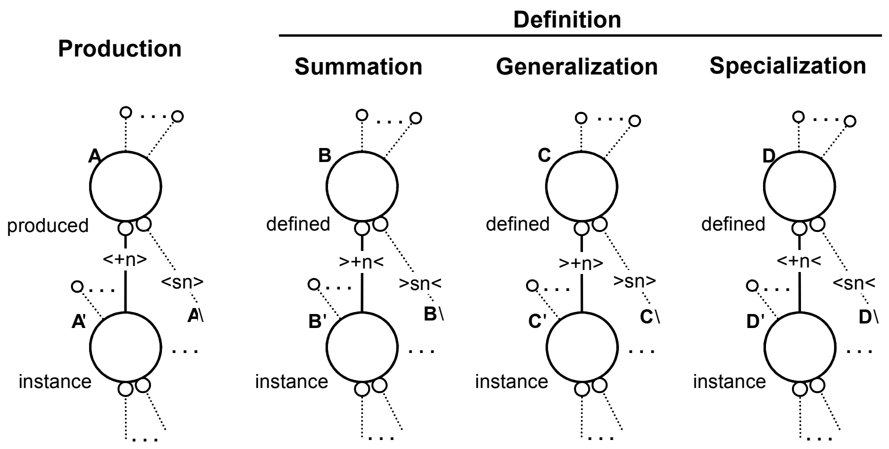

Figure 13. Composed Instance Patterns

9.4. Item Reality

Figure 14 displays example patterns for a concept composition association and an instance composition association each with the same item on each end. That association is necessarily a summation association that and is called a <u>self-defining composition association</u>.

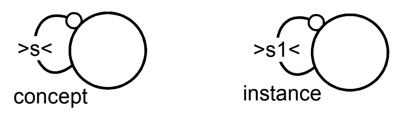

Figure 14. Self-defining Composition Association

A self-defining composition association must have a count of 1 because any other count would result in the item being not disjoint with respect to itself. Thus the information specified about an item by its self-defining composition association is due to its sign alone. That information is as follows.

> If the sign is –, then it specifies a single exclusion of the item from itself. An item that is excluded from itself once is interpreted as standing for nothing and is called a virtual item.
>
> If the sign is +, then it specifies a single inclusion of the item in itself. An item that includes itself once is the opposite of a virtual item, i.e., it is real and is called a real item.
>
> If the sign is +/–, then it specifies either of the two cases discussed above, i.e., it is unknown whether the item is a real item or a virtual item. Such an item is called an unknown reality item.

Since a –self-defining composition association must have a companion +self-defining composition association, all items have a +self-defining composition association. Since all items have a +self-defining composition association, it can be implied rather than stored. The I-A platform is presumed to be implemented consistent with this simplification. Thus the absence of any stored self-defining composition association specifies that the item is a real item. Diagrams similarly omit that association and any indication of an item's reality unless it is not a real item.

The information specified by a self-defining composition association or absence thereof is conveniently discussed, displayed and expressed as an item pseudo-attribute called reality. Table 5 describes this pseudo-attribute. The reality pseudo-attribute is abbreviated "r", unknown reality ("?") is a variable (also shown as an OR expression) and "$\underline{r}$" is a parameter.

Table 5. Reality Subtypes and their Use as a Pseudo-Attribute

taxonomy of reality (and abbreviation)			display convention	information specified
reality (r)	unknown reality		"?" [*i or* !]	It is unknown as to whether the item is real or virtual.
	known reality ($\underline{r}$)	real	"!" or blank	The item is a real item.
		virtual	"*i*"	The item is a virtual item.

In subsequent diagrams, an unknown reality item is identified by a "?" inside the item circle.

When an I-A platform process analyzes a substructure with a virtual or uncertain item, it would respectively ignore that item or treat it as discussed below for instances.

9.5 Substructures with Unknown Reality Instances

The following describes the information specified by counts of instances with unknown reality using an object substructure and it's simplification. A simplified expression as indicated by the "==>" symbol.

> The substructure on the left in Figure 15 defines object **A** in terms of object **D**. Object **D** is a common object having content 'a' and defines object **C** as a replicate in the same manner as in Figure 12 but with unknown reality, i.e., object **C** is either virtual (*i*) or real (!), expressed as +1'a' [*i or* !] (alternatively [+1'a' ?]) displayed to the right of the object. The **B** objects are each a replicate of **C** and are disjoint with respect to object **A**. Being disjoint with respect to **A**, they are independently either real (!) or virtual (*i*). Thus object **A** is defined by zero, one or two real objects. This is shown in the AND expression for **A**: [+1'a'[*i or* !], +1'a'[*i or* !]] which simplifies to: +'a'[*i or* +1 *or* +2], where *i* is the case where both **B**' and **B**" are virtual so that **A** is also virtual.
>
> The substructure on the right simplifies the previous substructure by eliminating objects **B**' and **B**" in the same manner as the simplification for counts presented earlier. The AND expression for **A** is +2'a' [*i or* !] (alternatively [+2'a' ?]) which expands to: +'a'[*i or* +1 *or* +2].

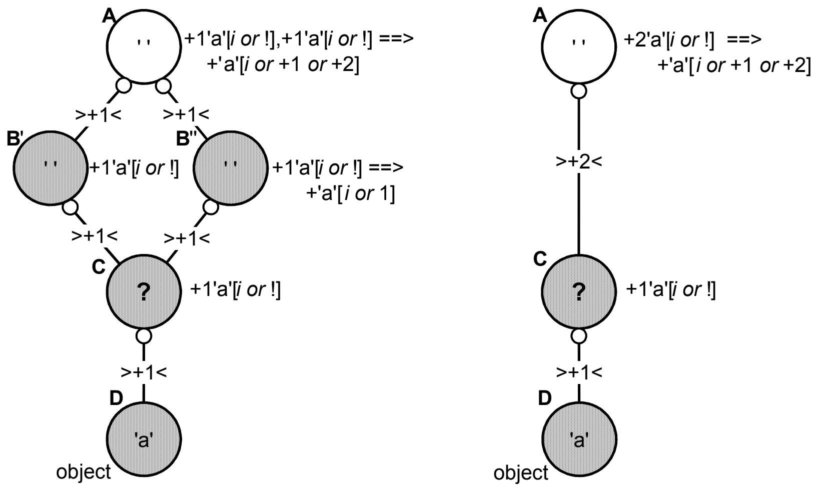

Figure 15. Equivalent Object Substructures Specifying Unknown Reality Objects

As demonstrated above, an unknown reality instance is the means of specifying uncertainty. The term "random variable" is commonly used to describe the information specified by object **A**.

9.6 Producing Value Exclusions

Earlier, §9.1 noted that the use of a −composition association is constrained to have a companion + composition association. An apparent exception to this constraint applies to a value specified as the ratio of values. In that case the numerator value(s) are specified by +production associations and the denominator value(s) are specified by −production associations, e.g., (*Mass* value) * (*Volume* value)^−1 specifying a *Density* value. However this case is not an exception to the above. First, note that the ratio value, its numerator values and its denominator values directly valuate the same object. Second, note that all non-ratio values have the same a space-time existence as the object it valuates. Third, note that all denominator values specify a portion of space-time existence. Thus, the exclusion of the denominator value(s) from the numerator value(s) simply eliminates that portion of space-time existence from the numerator value and, therefore, from the ratio value itself. (Ratios such as children per family and miles per gallon are not values of objects; they are normalized statistics of a population or a process).

For the *Density* value example, the numerator *Mass* value and the denominator *Volume* value each directly valuates the object, and each is understood as having the space-time existence of the object they each valuate. The −production association of the denominator *Volume* value excludes that portion of space-time existence from the numerator *Mass* value, leaving mass of the object but without its spatial existence. Thus a *Density* value is mass without its spatial existence, i.e., it's "volumeless mass."

The I-A platform is presumed to be implemented so that only a *Space-time* value can have a −production association.

9.7 Zero

An instance that is defined by another instance through both a −n definition associations and a companion +n definition association, where "n" is the same count, specifies that the 1-end instance no longer defines the 0-end instance. The AND expression for the defined (0-end) instance is [[+nθ,−nθ], …], where θ identifies the defining (1-end) instance and the ellipses indicate zero or more other (1-end) defining instance(s). If the 0-end instance is also (currently) defined by other 1-end instance(s), (as stated in §9.4) an I-A platform process analyzing the 0-end instance would ignore the θ instance. (A similar argument applies to production associations).

If the 0-end instance is defined *only* by the θ instance, its AND expression is [+nθ,−nθ] and is interpreted as a virtual instance since it is a defined instance that is defined by nothing (and would be so treated by an I-A platform process). Since sign, count and θ are independent, [+nθ,−nθ] can be written: [+,−]nθ. The expression portion: [+,−] expresses the fact that the two instances are not associated. That expression portion is conventionally expressed by the numeric character zero (0). Using zero in this manner, the expression for the 0-end instance simplifies to: 0nθ. This use of zero does not alter how zero applies in conventional mathematics other than eliminating the problem of division by zero.

9.8 Specifying Plural Items and All Objects

The means of specifying (1) plural items and (2) all objects having a given class are demonstrated in Figure 16. This figure displays an object definition substructure that defines other objects with the same uncomposed singular common **Person** object (**C**) representing *any* person. (The two identical **C** objects are for graphical simplicity; only one object **C** need be stored). The common **Person** object (**C**) is the only common **Person** object with the following non-class traits: real, no valuating values, null content and no composing objects. Absent *Time* values, object **C** is a current object.

> The left portion of this substructure demonstrates the specification of a plural concept. The singular **Person** object (**C**) defines plural **People** object (**C**'). The singular class: **Person** is stored and defined in a class structure; the plural class: **People** is not specified

in a class substructure, instead it is specified by its classification association with the plural object **C**'.

The right portion of the substructure demonstrates the specification of an object that stands for all things having the non-class traits of the common object. The **Person** object (**A**) is defined by multiple particular **Woman** objects, multiple particular **Man** objects and one or more common **Person** objects. The +null summation association specifies that there is an unknown count of common **Person** objects in addition to the particular **Man** and **Woman** objects. Since there are no other **Person** objects defined by object **C**, object **A** is interpreted as standing for all current persons with the traits of object **C**. Object **A** is also a particular object because what it stands for is known.

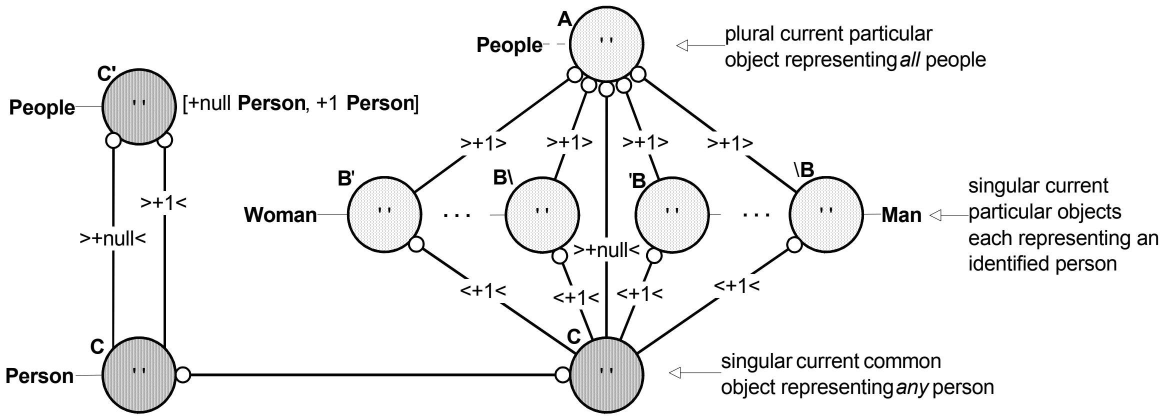
A
People
plural current particular object representing *all* people
C'
People
[+null Person, +1 Person]
>+1>
>+1>
>+1>
>+1>
>+1<
>+null<
B'
B\
'B
\B
Woman
Man
singular current particular objects each representing an identified person
>+null<
<+1<
<+1<
<+1<
<+1<
C
Person
C
singular current common object representing *any* person

Figure 16. Substructure Specifying the Plural of Person, All People and Any Person

Note that the left portion of the substructure demonstrates how a concept is specified so that it can be recognized as either a singular concept or a plural concept, whether or not the content of each is different, and whether or not one or both have non-null content. Note the following regarding the right portion of the substructure:

- It uses both sub>super-concept (> >) and super>sub-concept (< <) associations;
- Object **A** is particular even though it is defined by an unknown count of **Person** objects;
- Object **A** stands for *all* current people in terms of *any* current person;
- The **B** objects along with the +null count specifies the object domain for object **C**; and
- Each **B** object is what is conventionally referred to as an *instantiation* of **Person**.

10. EXAMPLE COMPOSITION SUBSTRUCTURES AND PATTERNS

The following are a variety of examples of composition substructures and patterns that demonstrate the application of this storage method and some additional consequences thereof. For simplicity, a sign of + or a sign/count of +1 may be omitted. These examples demonstrate the following about information specified using this storage method:

- A structure can and must specify all relevant information, i.e., no other information is stored externally to the structure other than what is needed to manage it consistent with the axioms, notions and presumptions discussed earlier;
- No simpler substructures can specify the same information;
- A more complex substructure would have redundant elements; and
- Information specified by a substructure is independent of any display convention.

10.1 Algebraic Operation Examples

Figure 17 displays patterns for basic mathematical expressions. Shown to the right of selected instances are expressions in terms of s, n and c that describe the information specified by that instance; r is omitted for simplicity. Identity associations and 1-end composition associations are omitted for clarity. The expressions use symbols as follows: "," for definition,"*" for production, "•" for sequential definition, "/" for reciprocation and "^" for exponentiation.

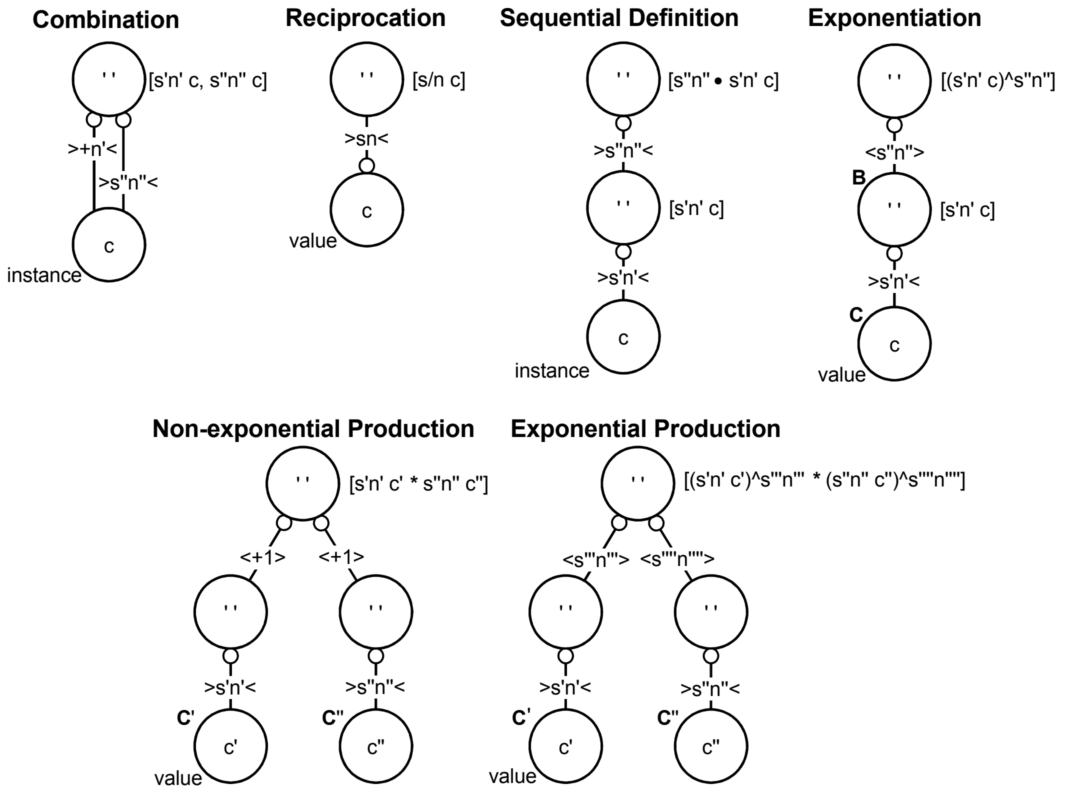

Figure 17. Patterns for Basic Algebraic Processes with Expressions

Note the following:

- Combination specifies either addition or subtraction when the signs are respectively the same or different;
- Multiplication arises in two different forms, sequential definition and production; and
- Some patterns apply to values only.

An inspection of the above expressions reveals that the information specified by an uncomposed instance and one of its composition associations is its sign, count and content, e.g., instance **C**' and its association specifies s'n' c' (r is real and is omitted). This is equivalent the information specified by a variable in an algebraic expression. The letter used to represent a variable is equivalent to an item-id, e.g., **C**'. Thus the atomic components of an algebraic variable are the attributes: s, n and c. In an I-A platform, a stored constant has all known attributes, e.g., +3 'grams'; a stored variable has unknown sign, unknown count and null content, expressed here as: +/− null ' '.

10.2 Class Substructure Example

Figure 18 displays a class substructure for subclasses of **Highway vehicle** with the global specific class. The super-most class is not displayed.

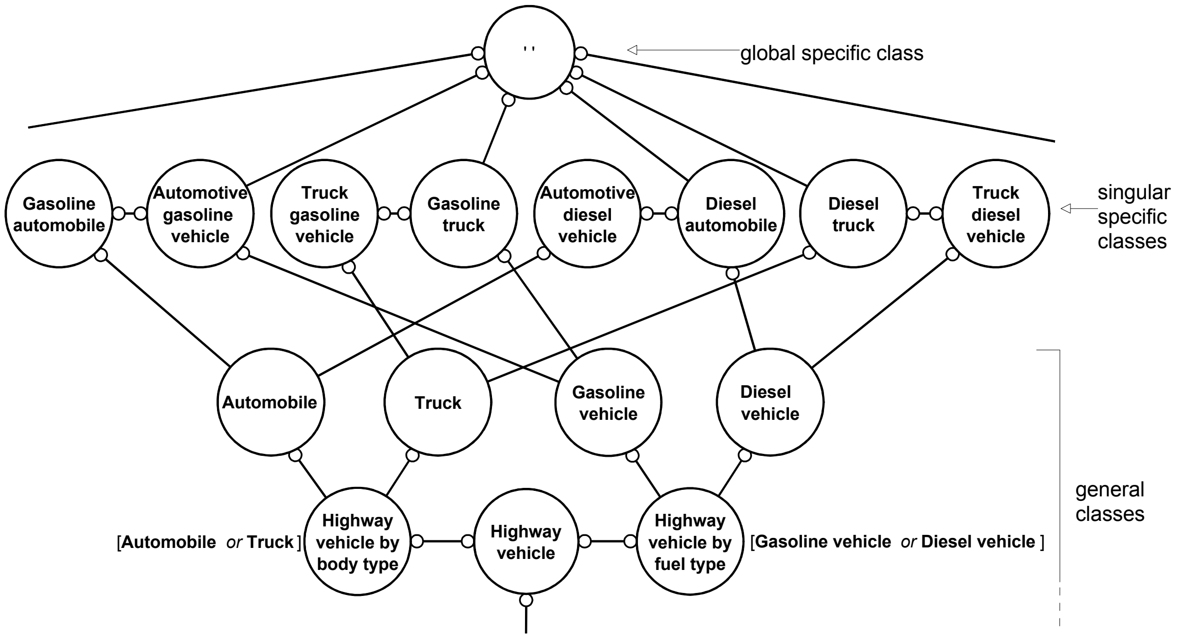

Figure 18. Portion of the Class Structure for Highway Vehicles

Note the following:

- There are two independent class hierarchy substructures mapped by identity associations;
- **Highway vehicle** has two identical classes that are synonyms: **Highway vehicle by body type** and **Highway vehicle by fuel type**, that are each the superclass of one of the independent class hierarchies;
- Each class is composed of zero or one general class;
- Classes that overlap (e.g., **Automobile** and **Gasoline vehicle**) are in different hierarchies so that all of the classes are disjoint; and
- The specific subclasses for a given class are the given class's class domain as suggested by the two OR expressions, e.g., **Automobile** and **Truck** are the class domain for **Highway vehicle by body type**.

10.3 Geographic Location Example

Figure 19 displays a substructure for a portion of the particular **Geographic location** objects for counties and congressional districts in the state of California. Since the counties and congressional districts spatially overlap (i.e., are not disjoint), the object that represents the state as a whole has two identical objects that are each at the top of an independent object AND substructure.

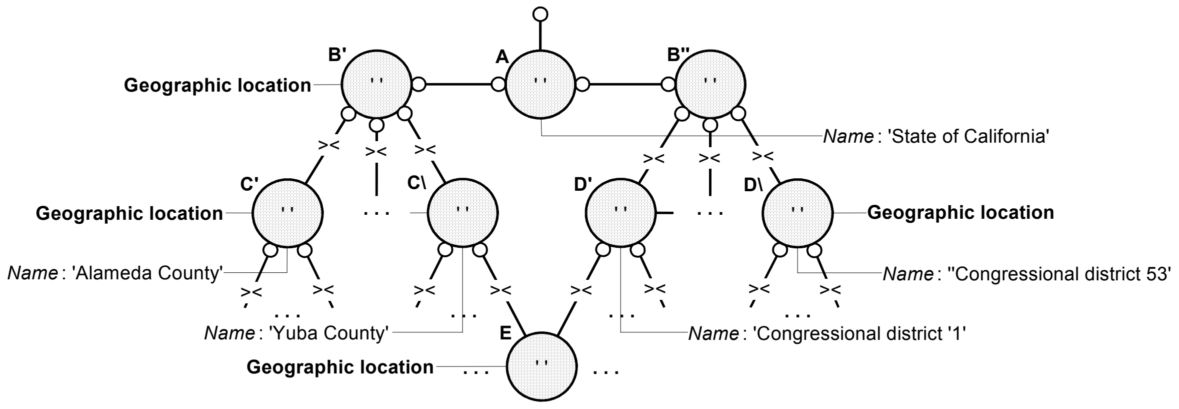

Figure 19. Object Substructure Specifying Geographic Areas

Note the following:

- Each object is a particular singular object classified by the same class;
- Objects **B'**, **A** and **B"** are identical;
- Each uncomposed object indirectly defines both objects **B'** and **B"**;
- Each defined object is a multiply-defined singular object whose defining objects are somehow adjacent;
- The objects defining the same object interact as a consequence of their adjacency;
- This substructure can be extended to specify more than two simultaneous geographic subdivisions;
- Each object except the uncomposed objects has its own *Name* value;
- Terms such as “district,” “county,” “state,” etc., are not classes, rather they are components of naming conventions; and
- Object **E** can be identified as the intersection of the two objects it defines.

10.4 Variable Object Traits Examples

A thing can have varying characteristics. The combination of characteristics applicable to a thing during a portion of its spatial existence is commonly referred as its state. An object can store only one combination of traits during a given *Time* interval. However, each different state can be specified by an object called a state object and those state objects can define an object that stands for the thing during all of its states. Each state object is disjoint and temporally adjacent to one or two other state objects.

Figure 20 displays two substructures: one that specifies different values and the other specifying different producing objects. The left substructure specifies an **Automobile** object (**A**) is defined by state objects **B'** and **B"** having adjacent temporal existences during which their *Color* values are different. The stored start and end *Time* values of object **A** are implied as applying to objects **B'** and **B"**. The right substructure is similar, but is for an object with different producing objects. The class of object **C"** has an implied classification association with an un-stored singular class that is expresses as: “**Automobile**, **-Engine**” where the double quotes indicate that its class is implied.

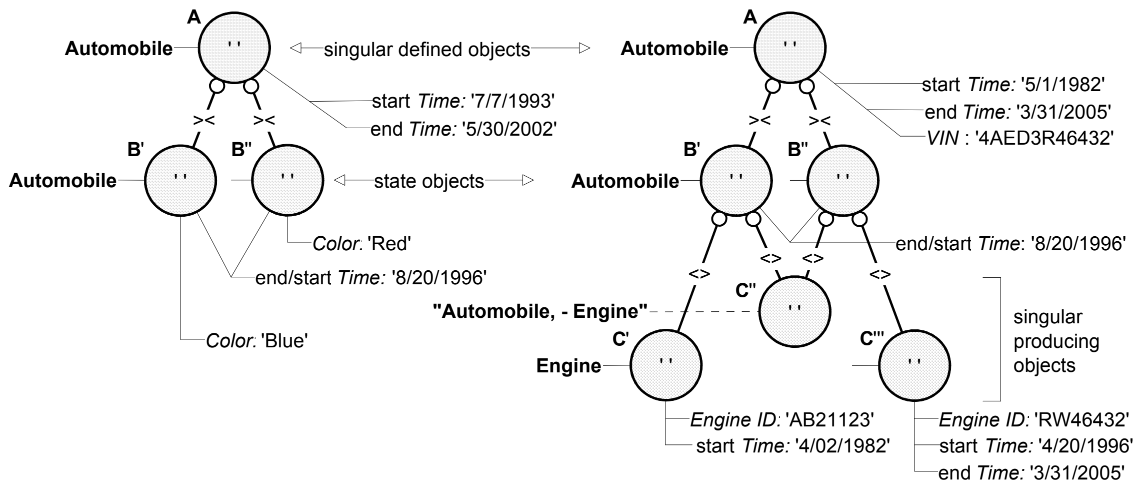

Figure 20. Object Substructures Specifying Variability in Values and Producing Objects

A substructure that specifies varying class or content would be similar to the one shown above left specifying varying value.

10.5 Object Sequence Examples

Figure 21 displays two substructures that each specify a sequence of singular particular objects **B'**, **B''**, etc., by the use of sequencing objects **A'**, **A''**, etc. The left substructure specifies a sequence of three **Character** objects forming the character string “cat”. The right substructure specifies a sequence of men and women in a queue with both partial and uncertain sequencing. Ordinal numbers are displayed below each uncomposed object to describe that object’s sequence as specified by the sequencing objects.

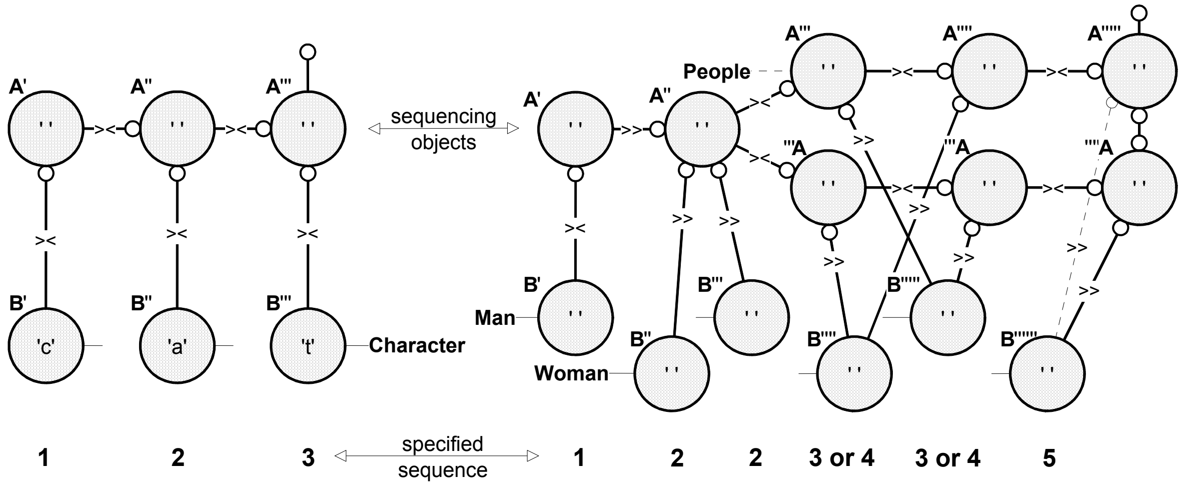

Figure 21. Object Substructures Specifying Full, Partial and Unknown Object Sequence

Note the following in each substructure:

- The sequenced objects have no sequence among themselves, sequence is specified by sequencing object that succeed the sequenced objects;
- Sequencing object **A'** is a replicate of **B'**;
- Sequence is specified without the use of ordinals; and
- There can be other simultaneous specifications of sequence for the same objects.

Also note that in the right substructure:

- The uncertain sequencing is specified by two different sequences; and
- If identical objects **A**""" and """**A** were one object, then objects **B**"", **B**""" and **B**"""" would define that object twice.

10.6 Organization Example

Figure 22 displays an object substructure specifying contracts, roles, work and assignments for the parties in an employment contract. Each object is assumed to have a different temporal existence and be uniquely identified.

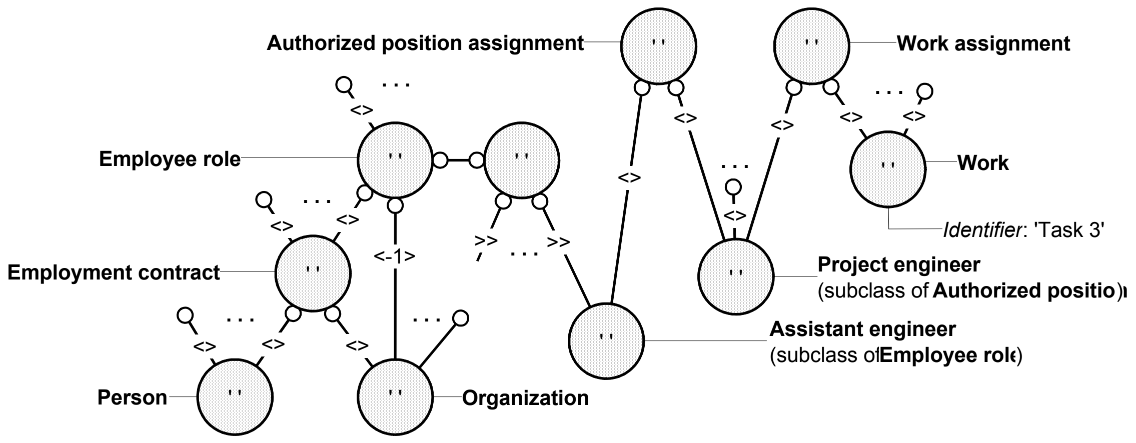

Figure 22. Object Substructure Specifying a Contract, Role, Work and Assignment

Note that the **Organization** object is *included* in the **Employee role** object (indirectly through the **Employment contract** object) and directly *excluded* from that object. This specification is consistent with our understanding of (1) a contract as specifying the roles of each party and (2) that each party's role is that part of the contract remaining after excluding all other parties. This exclusion is also necessary so that the **Employee role** object has disjoint producing objects. Also note that there are two identical **Employee role** objects: one produced and the other defined, the latter specifying all roles during that employment contract.

10.7 Value Domain Example

Figure 23 displays a portion of the value substructure that specifies the *Length* value domain for the English system. The uncomposed common value, called a metric unit value, represents any length at the threshold of observability. Non-null value content are singular or plural units of measure. Expressions are displayed for selected values in terms of the 'foot' value along with a simplified form of the expression. Also shown is the definition of the unit of measure 'rod' and its plural, as well as the singular and plural metrics and values.

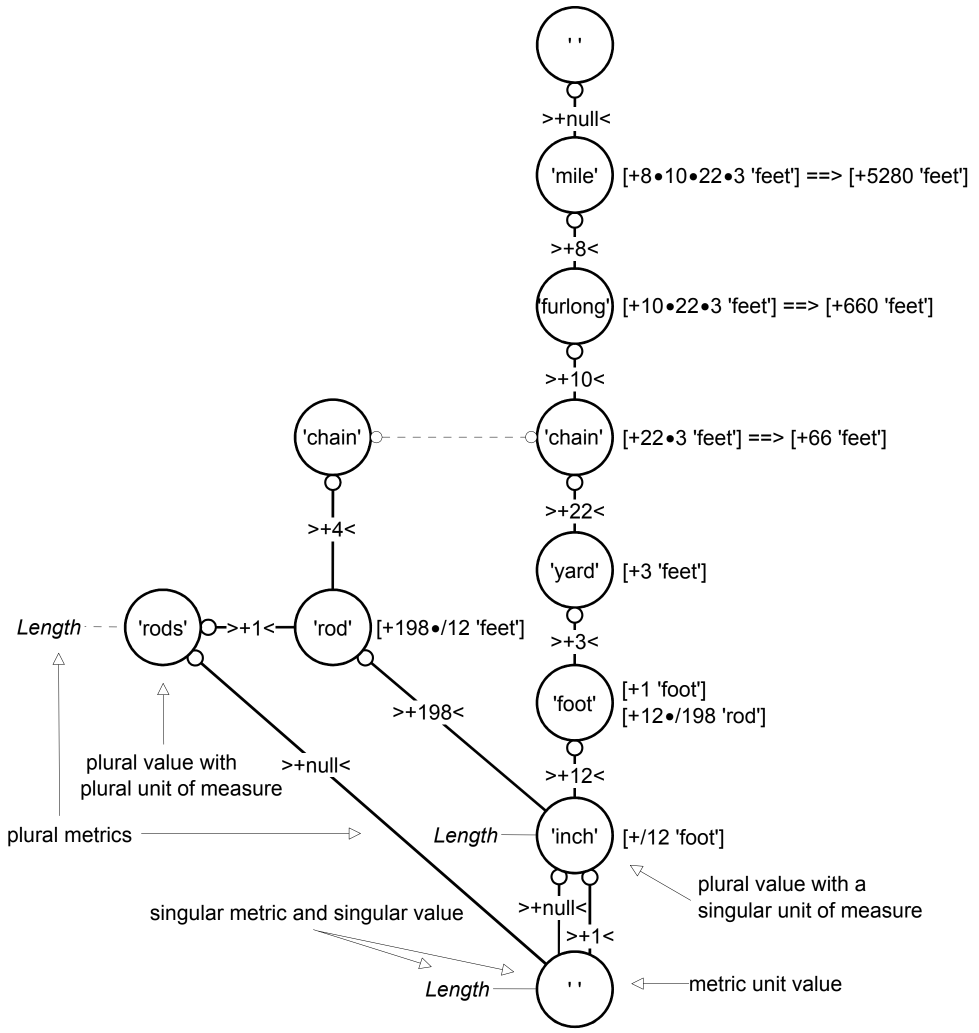

Figure 23. *Length* Value Domain for the English System Values

Note the following:

- This substructure specifies an un-itemized (and uncountable) value domain;
- Each simplified expression is a conversion factor that is generated by following the path from the 'foot' value to the subject value with the direction with respect to the definition association determining whether or not the count is a reciprocal;
- The identity association is not stored because it can be deduced by the I-A platform;
- The plural metric: *Length* is defined in this substructure and all other values have an implied value classification association with the plural metric: *Length* (some of which are displayed); and
- The metric unit value is the only singular value.

10.8 Person Uncertain Height Example

Figure 24 displays a substructure specifying the *Height* value of a particular **Person** object with an uncertainty range of 5 'feet' 7 'inches' to 5 'feet' 10 'inches'. The expressions to the right of value **A**, **C** and **D** describe the information specified by those items. This example also demonstrates how *Quantitative* values with mixed units of measure are specified.

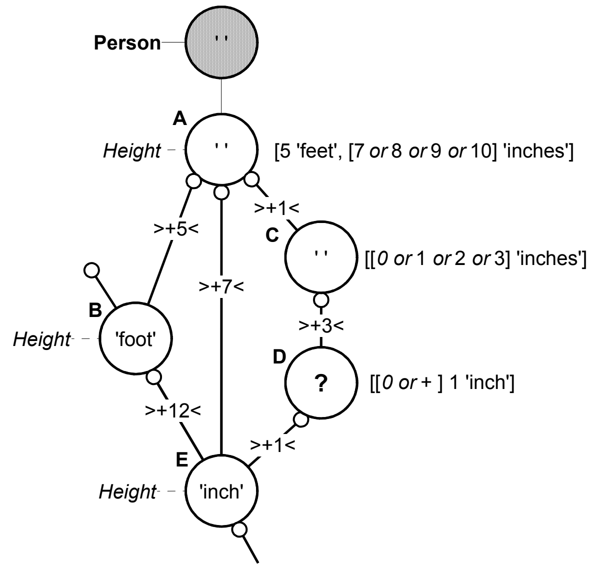

Figure 24. Value Substructure Specifying Uncertain Person Height

10.9 Algebraic Law

The algebraic law for distributivity is expressed by the following mathematical statement: $(N1\times N) + (N2 \times N) = (N1{+}N2) \times N$, where × is an operator for either form of multiplication, i.e., sequential definition or non-exponential production (see §10.1), and *N*, *N1* and *N2* are parameters for real numbers. Figure 25 displays the pattern for one form of distributivity: sequential definition for integers including zero. This pattern is described by the following expression statement: [(s'n' • sn), (s"n" • sn)] ≡ [(s'n', s"n") • sn]. Note that this expression statement is more general than the mathematical statement because (1) it separates sign and count, (2) it explicitly identifies the signs and counts as known (i.e., parameters), (3) it does not use sign as an operator, (4) it indicates that it specifies an identity not an equality or function, and (5) it is specific to one of the two forms of multiplication.

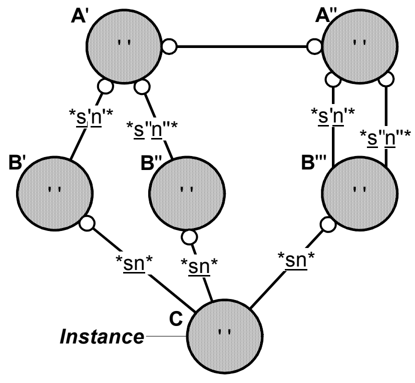

Figure 25. Sequential Definition Distributivity Pattern

11. SUMMARY AND OBSERVATIONS

Working from first principles described by axioms and notions, a very general method for managing information was developed. The axioms and notions plus the means of implementing them give rise to a method for storing information and a platform for managing it (the I-A platform). The storage method, composed of binary elements, is the simplest possible storage method. Rather than separating data elements from their organization, they are integrated so that information can be specified in a single information structure. Additionally:

- The method provides the means of specifying information not readily available using current methods, including: class and object hierarchies with overlapping items; value domains with units of measure; temporal variability; uncertainty and random variables; and complex sequences;
- Sign and count are independent attributes;
- Count arises from a structure simplification consistent with the axioms and notions;
- Zero is not a count but behaves like a count;
- Information can and must be stored in its most resolved form;
- Null content and unknown content are distinguishable; and
- Information stored in this platform would be fully normalized.

The difference between information and data is revealed to be as follows: information is what is specified by an information structure; data consists of what is stored in the sign, count and content attributes of the elements in that information structure.

The examples demonstrate the following mathematical-like consistency and rigor when information is stored in the I-A platform:

- Uniqueness – There is only one way of specifying a given unit of information; and
- Completeness – An information structure can and must completely specify the subject information.

Information stored using this method corresponds to how human use information to comprehend the world. This is evidenced by *uniqueness* and *completeness* as well as the following:

- Data and all of its organization are integrated;
- Information is stored in a single information structure;
- The method can handle a great variety of information; and
- Element subtypes correspond to commonly-used words.

The graphical notation and expression notation convey the subject information more robustly, consistently and concisely than current notations.

Parsons and Wand [2000] discussed the problem of inherent classification found in current methods used to organize information. The storage method discussed here eliminates that problem by specifying a class structure that is independent of the instance, and requiring greater rigor in the specification of that organization.

Codd's relational model [Codd, 1970] freed users from having to know the internal specification (physical organization) of the information. The storage method discussed here takes additional steps in that direction. Specifically, it eliminates:

- Having to rely on an externally-specified data storage structure (i.e., schema, file formats, data types and code);
- Needing to organize information around the ambiguous notion of an entity; and
- Separating storage from display formats.

A platform implementing this storage model would likely require more storage and possibly be slower than a platform implementing the current paradigm. However there are growing costs associated with maintaining that paradigm (e.g., incompatibility, support of multiple languages and the need to deploy multiple data structure-specific platforms). As storage costs continue to decrease and processor speed continue to increase, the relative advantage of the current paradigm may disappear if it has not already.

Although this discussion is brief, it does suggest a useful alternative to the current methods of organizing information that is closer to how we comprehend it. The method appears to explain some fundamental characteristics of information, thereby providing a foundation for the study information itself and a basis for a universal platform for storing information. It also suggests an alternative foundational basis for mathematics.

REFERENCES

Codd, E. F. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377-387. DOI=http://doi.acm.org/10.1145/362384.362685

Parsons, J. and Wand, Y. 2000. Emancipating instances from the tyranny of classes in information modeling. *ACM Trans. Database Syst*. 25, 2 (Jun. 2000), 228-268. DOI= http://doi.acm.org/10.1145/357775.357778