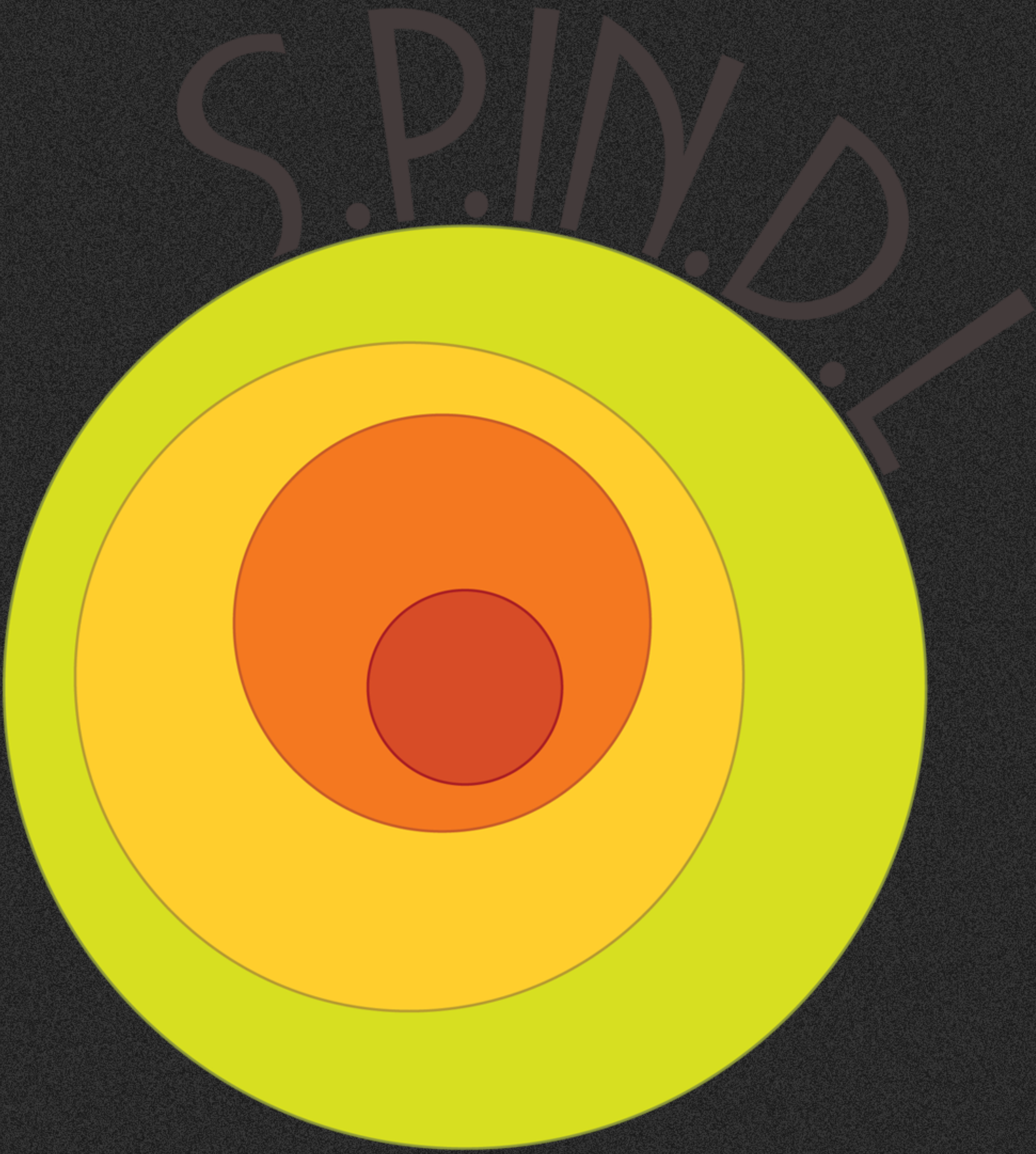
Structure vs. Semantics

the case for common reality in computing

Stefan Harsan Farr



Structure vs. Semantics

The Case for Common Reality in Computing

Author: Stefan Harsan Farr
Email: contact@semanticperspective.org

Preprint Edition 2014

Stefan Harsan Farr
www.semanticperspective.org
contact@samnticperspective.org

Abstract

Against the massive work put into what has catalyzed into the concept of Web 3.0, the promise of a single semantically charged Internet seem to continue to elude the IT community. The present work raises the flag on a trend that the author believes to be at the heart of this lack of success, namely the that of the lack of a common reality. It is being argued that the structure oriented approach that stands at the base of virtually every representation system, ultimately leads to frictions that cannot be reconciled and as a consequence prevent the formation of what would be a true revolution in the IT industry: the semantic web. The paper continues by proposing an alternative solution to knowledge representation which does away with structure relying instead of pure semantics which is borrowed unaltered from human language. The paper further argues that this paradigm shift can also be the catalyst that is needed to open the way to free flow of information and later to the formation web 3.0.

Keywords: structured data, semantic data, computer communication, b2b, concepts, natural language, knowledge representation, web 3.0

1∘The Case For Common Reality

It would probably be safe to assume that the intended audience is already familiar with the conceptual differences between data, information and knowledge, these are, after all, the pillar concepts of information technology. It is non-the-less, important with regards to the outcome of the work, to talk about them and emphasize certain aspects of them, as they stand at the base of plot the current case is built upon.

1.1 ∘ Information

If we are to look at the definition of the terms as found in the English Language Dictionary we can see that the terms are not clear cut and depend heavily on each other:

Data, “is factual information organized for analysis and processing”, *Information*, “is knowledge derived from study, observation and experience” and *Knowledge*, is defined as “the state of knowing, or possession of specific information about a certain subject”.

The definitions go full circle and they are all centered around our capability to observe, remember and adjust our future actions according to the things that we observe and remember. This is what most consider *Knowledge* to be, more informally speaking: the totality of facts that we can observe and remember coupled with our ability to manipulate consciously the course of our actions based on them. The facts that generate the decisions associated with knowledge, are chunks of information: conglomerates of facts that make sense together in a given situation enough so they can trigger that change in behavior. This aspect of making sense is essential to *Information*, without it there is no reaction, there is no knowledge, it's only *Data*: random facts, observations devoid of context that do not make sense on their own, but which can nevertheless be recorded, replayed and otherwise manipulated.

Information is always semantic, it has meaning, whereas data is not. So if we are to create a hierarchy between these terms, *Data*, would sit at the bottom and represent the blind facts drawn from the environment. When processed and put into context so that it makes sense it becomes *Information*, and when the totality of the information is put together so that decisions can be made *Knowledge* arises.

In the day to day human life we don't really operate with data, as it is a form of storing facts which is useless in most cases, not being readily accessible. In our heads, we don't store data, we store information, facts that are relevant to our own, personal existence, facts that are highly connected and very readily available making them highly valuable in the day to day decision making process. When we read a book, the letters of the book are data, but we are not concerned with those, not consciously anyways, we are driven inexorably towards extracting the information from the book: we read until it makes sense. The human mind, is not designed to work with data but the concept was born anyway (even if it was not consciously termed at that point) from necessity when some people needed to work with other people's information.

Evidently when one operates on other people's information, this (the information) will loose some or all of its meaning, so other forms of mechanisms need to be put in place to ensure the information survives a transitory meaninglessness. For example, one can copy an entire book and not understand its content, nevertheless, the information in the book is preserved, even transferred into the new book thus being multiplied. Information can survive meaninglessness, and can even be manipulated in this meaningless state, but it is essential that an individual who does understand the information encodes

(It) into this transient form, and makes it available to the data processor which does its job. Later another individual with capability to understand the information has to decode the data and make sense of it.

In today's world it is the IT professional's duty to put the information through the semantic grinder and transform it into this meaningless state in which it can be manipulated by the computer, but in a world where we are trying more and more to make sense of the vast amount of *Data* that we have come to collect, this skill that we readily learn as soon as we step into the school is becoming more of an impediment rather than an asset. It is more and more difficult to discern where information ends and data begins, and all too often the very code that is destined to hide the meaning is confused with the meaning itself.

While the field of Information Technology contains the word “information” in its name, perhaps because it is meant to bring information to people, to create information and better the decision making process its object of work is in fact data, not information. Information is very difficult to work with because it does not have clear boundaries. Data is stable, stand alone and context independent; one can always count on data to be data, but information depends on the subtlety of meaning which varies enormously from situation to situation, from interpreter to interpreter. What may represent information in a certain context it may be meaningless raw data within a different context or something in between. While data can be quantified, stored and framed between certain limits, information will always be a gradient of values that depends heavily on who observes it and how it is observed.

It is not difficult to understand that due to this intimate nature of information, people who work with it, find it difficult to draw the line between data and information. We are intelligent creatures, we possess knowledge and we operate with information on a constant basis. Whenever we look at data we will strive to make sense of it and we will always find some information that hides in there. Yet more often than not, the information that we see is just an illusion, a residual flicker of our thinking process interacting with the data that we operate with. Once that data is disconnected from us and becomes exclusively part of the cybernetic environment that entire meaning is lost. It can exist no more. Unlike us, the cybernetic system lacks the spark that is needed to transform data into information and our programming techniques are not doing a great job in improving this handicap. If we are to step into a new, semantic era, where information is stored in meaningful state outside the brains of individuals, we need to reanalyze what information is and what it represents and we need to understand its limitations and particularities. We need to especially understand how meaning (semantics) connects to data and the way they, together, create *Information*.

1.2 ◦ Subjectivity & Incompleteness

As opposed to data, information cannot exist on its own. It arises dynamically from the interaction of data with that which makes use of it. It is a disconcerting feeling for a computer scientist to realize that information, the very object of his profession is not perfect, but rather something that will be different to every single user that observes it.

To emphasize this imperfectness, let us consider the following statement:

example: 1

“The man is 35 years old.”

It is a well formed statement that conveys a very valid piece of information about the person in discussion, but to most of us, this sentence contains no real information. In the best case it is a piece of data torn out of a context that we are not aware of. In order for this to represent actual information one requires considerable knowledge in what this specific context is concerned: “Who is the man?”, “When was this stated?”, are just two of the questions that can immediately be asked by somebody who's reading this single sentence in a transcript of the underlying conversation, not having a context to place it in.

example: 2

“The man who lives on Elm Street 99999, apartment X, Aukland, New Zeeland is 35 years old.”
“The man who lives on Elm Street 99999, apartment X, Aukland, New Zeeland was 35 years old in year 2013.”

The somewhat more complete sentence in example: 2 can quickly respond to these questions, but again the statements imply a great deal of assumptions about who is going to read them: “What is an Aukland?”, “What is a New Zeeland?”, “What does year 2013 actually mean?”.

The fact of the matter is, that regardless of how much we describe the scenario, there will still be questions that are unanswered and assumptions that have to be made regarding a-priori knowledge possessed by the data consumer, regarding the context in which the information resides.

This effect is not limited to information conveyed via spoken or written language. Any object is potentially describable by an infinite or unreasonably large amount of attributes, some of which may even be inaccessible, and as such any information drawn from that subject is inherently *incomplete*.

Failure to recognize this aspects could have grave consequences with regards to information processing. If one does not recognize and accept incompleteness of information one would be tempted to analyze an object ad infinitum, trying to grasp all the details and characteristics of it. The process itself would likely generate an information overload. In the world of humans though, this is not the case. The human brain is very well equipped for these particularities. Most of the time it will only draw just as much information as we actually need to identify important aspects about the object: categorize the object, identify whether it is dangerous, useful, etc. Irrelevant facts, even if identified, are quickly forgotten making room in the memory for the next thing.

This selective observation (extraction of information) takes us directly to the other important aspect of information: subjectivity. All subjects draw information using their own particular sensors and interpreters and will filter it through their own preexisting knowledge and angle of interest. As such they will identify particular aspects of the object that are unique to them, thus giving information a highly *subjective* character.

It will become clear later on in this chapter, why these two seemingly evident characteristics are so important within the context of “knowledge representation” and how present representation systems fail to properly account for them.

1.2.1 ∘ Information Transfer (communication)

When it comes to the individual, subjectivity is not all that evident because there is no point of reference, but when we look at the case of information transfer, it becomes obvious that it is so essential, that failure to recognize it would render communication impossible. If subjectivity would not be recognized and dealt with, two subjects could never communicate because they would never be able to establish consensus about the objects of their discussion. In the world of human communication, this does not happen either, because the brain can calculate the various degrees to which certain information is particular to itself, the individual, the group of individuals that are communicating or larger, more complex circle of individuals. Common concepts that uniquely identify objects, within particular contexts, are conceived such that they are precise enough to serve their purpose yet loose enough to allow for individual perspective. For example there is a pretty wide consensus of what the color “red” means and people don't usually argue about the “redness” of an object. They may however have different perspective on certain shades like, light pink versus light violet, where some will see it pink and some will see it violet, but these cases are rare in human communication, as language is designed to grasp what's common not what's different.

1.3 ◦ Common Meaning In Human Communication

In the world of human communication, common concepts are extremely important not only because of the characteristics themselves but because they uniquely identify objects or classes of objects and determine the common reality of the communicating subjects.

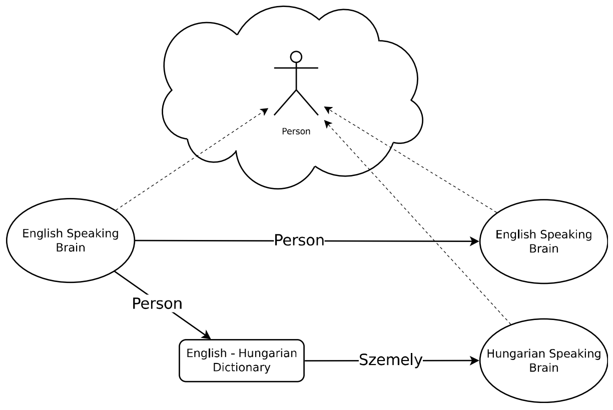


Illustration 1: Person to person communication, common reality (definitions)

When information is transferred this way, the absolute priority is determining the nature and identity of the object of the discussion first: what that object is, what it represents for both parties, what its common meaning is. Only after the common meaning is established, structural information, or details about the object, can come into discussion.

When two people speak, the information transfer is possible because both brains, the sender and the recipient relate to the same “reality”, be that the real world or some abstract world like mathematics or feelings. The recipient may be a doctor and the sender a banker, but when the “Person” word is invoked both of them associate it with an individual in the real world. They may know many different characteristics about the person particular to their profession, but when they speak, none of those particularities matter. What matters is what they have in common and to access that, they need an absolutely simple common pointer, which is the concept of “Person”. The simplicity of this is so powerful, that even if they speak different languages they can actually transmit the information using a dictionary, because regardless of how the word is pronounced or written, it references the same simple concept in a reality that is common to both parties, Illustration 1.

Sentences can be translated the same way. Rules (grammars) may be different in different languages, nevertheless they are based on the same reality, on the same objects, on the same actions, on the same context (time, person, possession, command, etc.) and as such, a translator, somebody that knows both languages, can transpose the sentences (these conglomerates of objects, contexts and rules) from one language to another such that the reality, the semantics of the message stays intact. The fact that this reality is common is a sine qua non condition why such translations are possible. Each language is in fact an encoding of the reality of people that speak it and when a translator has knowledge of both encodings and the encoded realities overlap sufficiently, he or she can transcode this portion of reality from one encoding to the other without any losses. If however, one of the languages lacks a term for a concept, this usually occurs when the reality of the people who developed the language lacks the concept itself altogether, translation becomes difficult or impossible.

The Pirahã language, for example, has no cardinal or ordinal numbers. Why this is, is still subject to academic debate: some argue that the Pirahã people cannot learn numeracy, others advocate that they can count but they choose not to. Whatever the reason, translating example: 3 into Pirahã in an exact manner, is impossible, as their reality lacks a concept behind the source language, numbers:

example: 3, Partially translatable sentence

"Twenty people went hunting and they brought back three pigs."

For the Pirahã, numbers don't exist. A translator might be able to approximate the meaning and transpose it to the other reality with some information loss. Instead of "twenty" they can use something akin to "many", instead of "three" they can use something like "few". This is a much more difficult job to do than simple translation, because direct correspondence between realities does not exist.

An interesting aspect of this translation process is that we, who's reality is compatible with that which English language models, would be tempted to say that the translation occurred with information loss, simply because the Pirahã reality lacks some fundamental aspect of the actual reality, and so, a back and forth translation will not restore the information to its original state. This view, however is judgmental and incomplete. The Pirahã reality does capture multiplicity / quantity, but it does so in different forms, which is not fully compatible with ours. The proper way to see this is that the two subjective realities are fundamentally different from this point of view and hence the translation occurs with information loss not because one reality is deficient, but because the two subjective realities contain concepts that are not fully compatible.

With such fractured reality, translations is only possible provided that some level of correspondence does exist between the two realities within the domain of the information being transferred and that the translator knows both these realities well enough to make a correlation. On the same token, the translation in the example: 4 would be utterly impossible, because there is nothing in the message that could remotely be correlated to the Pirahã reality[1]:

example: 4, Untranslatable sentence

"Three brokers sold ten thousand bonds today on the stock market and made a million dollars in profit."

In case of this sentence, there is no common reality that a translator can refer to in translating the sentence. Their world is based on different values as ours, from this particular angle, and as such there is nothing of value to them in this sentence. This particular information cannot flow from one side to the other simply because there is no context (meaning) to give birth to information on one of the sides.

1 The Pirahã are an indigenous hunter-gatherer tribe that live in the Amazon basin.

There is no reality behind the data. For fairness of treatment, I must point that for obvious reasons it is impossible to me to give an example of a sentence (information conveyed) that would transmit something that exist in Pirahã world and does not exist in ours.

It is I think reasonable to state that the reason why computers have such difficulties in human translating language may be that they do not “understand”[2] the reality in which the sentence is based. Computers are *unaware* of the meaning of the sentence and as such they must rely on translation between words and grammar and statistical matching, which highly inaccurate and sometimes can be confusing.

Computers don't have a reality and whether they can have one remains to be decided by future AI research. Until then, short of learning the correspondence in languages for every single expression that exists out there, translations will be imprecise. Even if the future computers will possess an internal cognitive process akin to consciousness that reflects in some sort of reality that they, among themselves share, it is still questionable whether they'll be able to translate our language. For that to happen, they will also have to sense our reality because only then can they create precise correspondence between real concepts and information written.

1.4 ∘ Information In Computer Programming

The essence of computer programming is the creation of software, packets of instructions, that can tell generic hardware, such as computers, to perform a specific task. There are numerous practicabilities of software but the current paper targets a specific branch of this industry, which is concerned with the information manipulation and interchange at high level. This branch deals extensively with the abstraction, collection, organization, storage and exchange of data that is collected from reality, therefore, “information in computer programming” will be analyzed from this specific angle.

It is important to re-iterate, that although we are discussing about information manipulation it is really data that is being manipulated. The state of information is lost the moment it is entered into a computational system, and it will exist in this meaning free form until an operator makes sense of it. Evidently not all information survives this transitional process, a lot of it is lost the moment the encoding happens, because the process can only capture information for which an encoding system exists. All the rest of infinitely complex reality behind each concept is lost, or presumed to exist at the destination and be identifiable from the data being transmitted.

1.4.1 ∘ Types, Static Reality

Data types or simply types, are categories of data based on limitations on the values that they may have making them easier to manipulate.

At machine code level the boundary between data and instruction becomes less evident so programming languages like Assembly had no need for data types. But even the simplest of operations are arduous to implement in such languages and with the rise of applications that were concerned with the manipulation of data, the higher level programing languages, the concept of Type appeared. Having predefined limitations on their values, brought inherently by certain characteristics imposed

2 Make sense of the information and its context with respect to the reality in which it is defined

a-priori, it became easier to implement sets of instructions (operations and functions) that can manipulate these groups of data without the need to know their values prior to the construction of these functions.

There is an important aspect to be said with regards to the type. The type itself, the category to which data belongs to can and often is information. It is probably the only kind of information that is available to the computer software and not the human operator. Because the type often determines the kinds of operations that can or must be performed on pieces of data, it represents an element that can be a precursor in the decision making process of the computer software. Even if such decision making process is nothing else but a pre-programmed set of instructions, it nevertheless is a sort of knowledge mechanism (see more at: 1.4.5 Semantics, The meaning Of things).

◊ PRIMITIVES

Primitives are the most basic of data types and have the widest acceptance of all the types. Almost every programming language makes use of them and are usually part of the core of the programming language.

Although they may be called different names (“int”, “integer”) in different programing languages they usually refer to the same thing and they are needed because they represent a necessity of the computer programming reality. Contrary to what the name suggest, an “Integer”, is not any mathematical integer but rather only one of the mathematical integers in the range [-2^{31}, +2^{31}-1], that are representable on 32 bits. As such, the Integer type is a partial wrapper of the mathematical integer numbers combined with a wrapper of a limitation in the computing reality: that information is ultimately encoded in bits (ones and zeros) and only a finite number of bits fit onto physical devices such as a block of memory or the stack. If more precision is needed, a wider range type can be used called “long integer” which can take value in the range [-2^{63}, +2^{63}-1], a 64 bit representation system, nevertheless, it will still represent the combined concept as in the case of simple simple int.

example: 5, Some common primitives

- boolean: represents a logical value (true or false, one or zero) on a bit.
- integers: represents an integer value in the range of [-2^{31}, +2^{31}-1] or [-2^{63}, +2^{63}-1],
- floating points: represent a real number in floating point representation. The range is much larger than that of integers for the same space they occupy. The precision varies with the magnitude of the number: the larger the numbers, the bigger the gap between two consecutive representable number (as real numbers have infinite precision and as such are impossible to represent with finite resources).
- character: represents an alphanumerical character on a byte, usually
- string: represents a sequence of characters, arbitrary texts (usually human readable)
- byte array: represents a raw sequence of data of any kind (usually not human readable)

These primitives (the most basic level of them) are a lot more about representation rather than the actual meaning of what they encode: concepts like integer, or real are only loosely encoded into these types. In fact, if arbitrary precision is needed for special purposes like scientific data or accounting, custom representations need to be created, like BigInteger or BigDecimal in Java programming language, because regular primitives are not suitable.

These things may not be new for the intended audience, but it is important to name them because there is a lot of confusion between the concept and representation of the concept, that stem from the fact that the representations are named suggestively to match the concept that they loosely encode and make it easier for the developer to work with.

In fact they could be just as well be named “regetni”[3] or “elbuod” or “naeloob”, because it would be just the same from the compiler / interpreter or machine code's perspective. They would only be a lot harder to work with. This is why virtually all programming languages that use mathematical operations will name the integer type suggestively as “int” or “integer” and as such, the terminology became universal.

Rarely do any programming language extend their set of primitive types beyond these representational primitive. It is the responsibility of the developer to correctly encode any custom concept onto these universal structures such that it captures all the subjective information regarding those concepts within the context of the client and the provided specification.

◊ VARIABLES

Before we get to the structured data types, it is important to mention the concept of variable, another extensively used artifact in computer science. In programming (at least from the aspect discussed by this paper), variables are containers that can be used to store values and operate on them.

example: 6, General format of a variable.

```
Type name = initial value;
```

Variables are usually defined as an identifier (a name) which is used as locator for the stored value (stack or memory), a type that serves as constraint for the values that can be assigned to the variable and it may also contain an initial value. Identifier is usually ubiquitous in all languages except for very low level languages like Assembly[4], but the other elements vary from language to language.

This seemingly simple construct is so powerful that it is used everywhere in computer programming. Everything from simple memory zones allocated for temporary storage, cells in a spreadsheet, a database or a dot on the screen can be thought of as a variable: a container in which value can be stored. This concept of placing a value in a container is essential to the computing process, it is the only way operations can be made in a serial system, but beyond that, things like type and name are really irrelevant once the software becomes machine code. Nevertheless they are extensively used and very popular too, because both, type and name have the power to carry semantics into the process of creating the computer program. We do not give it a lot of thought, but the fact that we can assign meaningful names to variable revolutionized computer programming. We'll discuss more on this subject later.

◊ ARRAYS & MATRICES, GRAPHS, TREES & MAPS

It is sometimes useful to be able to work with collections of data which can be handled in bulk according to some characteristics. Arrays, matrices (multi dimensional arrays), lists, sets, trees, maps are all such collections.

3 Integer written backwards

4 Assembly language is an instruction oriented language which is very close to machine language. Variables are not used, operations are done by inserting values directly into memory addresses or operator registries.

It is important to note that although programming languages treat them under the same umbrella, these collections are not in fact data types, but rather compound variables. Variables that have multiple slots where data can be placed, according to specific rules, some fixed some dynamic in nature. In the case of arrays and matrices, which are similar to those in mathematics, the slots are accessible by their position, and they can have uni or bi-dimensional structure respectively and occasionally even more. In the case of linked structures like linked lists, trees, or graphs, access is done according to the relation between elements. People that are still familiar with older programming languages like Standard C or Pascal, can recall that these linked data structures did not even exist, back then as part of the standard API. They had to be defined as collections of dynamically allocated memory zones and then linked with one another.

Maps are interesting because they are very similar with the concept of structure, as elements in a Map are accessible by their names, so to speak, and as such additional information exists besides the value of the variable itself, in the form of a key or a name. As opposed to structures, this information can be carried into the application, and be part of the program's execution.

◊ STRUCTURES

Concepts and values handled by these modern information manipulating applications go well beyond simple integers or reals or values of truth. To be able to easily manipulate data that are complex, structured types were created out of which developers can construct complex types that represent complex concepts from reality.

example: 7, Structure

```
Book{
        String title;
        String author;
        String publisher;
        Date datePublished;
        string ISBN;
        etc …
}
```

Suppose the concept of Book has to be wrapped in a computer program and that in the eye of the client a book would be described by a series of properties, such as title, author, etc … example: 7. As such, these properties have to be treated together for each individual book, otherwise it would be really difficult to track all these properties.

Structures contain ordered groups of data items. Unlike the elements of an array, the items within a structure can have varied data types and are accessible using similar syntax as variables. Classic programming languages treat these these properties as the definition of the type, in the current case a Book, and model them conceptually together with various paradigms.

The Relational Model, considers the type book as a relation between the typed items that construct the book and packs them together in relations (better known as tables), where each row is a group of values that together represent an individual relation, a book in our case. The final structure is in fact a matrix of values, where the rows represent individual relations (book entries) and each column represents one particular aspect of all known (stored) relations. A language was built which very efficiently handles manipulations related to the storage of the data in this form (storing, recovering, filtering, etc …) due to the reduced complexity of the architecture (both structure wise and operation wise). To make a blunt analogy, databases are memory zones, with each table being a matrix like the one mentioned above accessible via a variable, which is the table name.

To efficiently handle data like Book, programming languages don't treat them as arrays of coupled values, but rather like structures, variables that have variables inside. Instead of *book = {book[0], book[1], book[2], … }*, it becomes *book = {book.title, book.author, book.publisher, …}* which is a lot easier to work with. The approach allows developers to embed semantic elements into the construction of the program, the source code, an aid that makes the program many times easier to develop, maintain, upgrade, handed over to other developer, test, etc …

1.4.2 ◦ Operations & Functions

The vast majority of programming languages in use today are structured languages. The structuring refers to grouping functionality together into loops, blocks of codes and subroutines in contrast with the using the "go to" statement which was used to jump to different parts of a single chain of commands. This structure improves clarity and re-usability of code.

One of the most notable features of structured programming are the subroutines, also called methods, procedures, function, etc., depending on the language but they essentially consist of subsets of instructions, grouped together to perform a specific operation in a black box like fashion: whoever uses the code only needs to know what goes in and what comes out (in terms of types), in case of functional languages and additionally how the subroutine modifies the state of the application in case of imperative language (which are the vast majority of them).

There are many paradigms of computing regarding functions (declarative, function, logical, and so on) which impose restrictions on how these subroutines should act regarding the computing environment (some change state, others are not allowed to change state only have output) but the vast majority of languages that handle data modeling (procedural programming or object oriented programming) apply the concept of the subroutine as a mixed concept. They are allowed to have, or not to have, a return value and they are also are allowed to have side effects (affect application state beyond the scope of the function and its return value). This versatility, is a compromise, which is necessary in most cases due to the complications that arise from using a strict paradigm like procedural programming does. The benefits are increased efficiencies, re-usability, but there are also drawbacks to this approach.

These functions rely almost exclusively on structure imposing very little limitations which can be detrimental when it comes to inference (use of deductive reasoning) and code automation because there are no basis on which such paradigms can be implemented. The interior of the subroutine can be as long (verbose) as the developer deems fit, can perform any number of changes to the state of the application as a whole: write to disk, network, change user interface and so on. While these are all necessary things, the fact that they are allowed to be performed all in one place is a huge handicap from the perspective of semantics.

Operations on the other hand are code snippets that usually wrap a single action. Examples of operations are the mathematical plus "+" operation, subtraction, multiplication, logical operations like greater than ">", and so on.

Operations are often regarded as simple, semantically charged activity entities. Operations strongly wrap the concept what they stand for, the restrictions they impose on the operands (parameters) and for this reason, operator overloading has often been criticized that it allows developers to create operations that are confusing by using an operator that wraps a concept, for example the "+" sign, and implement code for it that perform a totally different concept, like adding an element to a set[5]. It is

5 One could state that <u>adding</u> an element to a set has similarities with the <u>addition</u> of number, for what the + sign stands for, but the similarities are brought in by the similarities in language. The two concepts are completely different.

notable how people observed this fact and even argued against operator overload for feeling the need to preserve the semantics of the activity across code and even language. This is especially interesting because functions suffer from the same problem too, (see 1.4.3 - Functional confusion), yet nobody seems to be bothered by that. The reason for this bias could be a combination of computing reality and developers' need for semantic information. Operations come from mathematics, are very strict in nature and we are taught to like that strictness because it is very reliable:

example: 8, Result of the + operation based on the input

$n + n_1 = n_2$, for $n, n_1, n_2 \in \mathbb{N}$
$n + r = r_1$, for $n \in \mathbb{N}$ and $r, r_1 \in \mathbb{R}$

The definition domains (types) of input and output values are so compact, so simple and so strict, that there is absolutely no doubt in the formulation. It is elementary, it conserves the concept and the same way mathematical proofs can be built on it, so can automation inside a software.

The problem is that operations are few because mathematics is only concerned with a limited aspect of our reality and these operations are designed to serve that strict concern. Programming on the other hand has many and various needs outside the scope of mathematics (conceptually speaking). Adding a button to a canvas object or watching for events on the network have no mathematical counterpart. To serve these many and various needs the functions (subroutines) were developed. These functions have loose definitions to be versatile, as opposed operations but as such they lost the semantic charge and developers have learned to accept that as a fact.

1.4.3 ◦ Functional Confusion

If we were to show the general function definition in computer programming to somebody whose is not a software professional it would be nothing more than an unintelligible collection of words.

example: 9, Schematic definition of a function

```
[language specific modifiers] result_type function_name(
                parameter_type_0 parameter_name_0,
                …,
                parameter_type_n parameter_name_n){
        …
        function body
        ...
}
```

As strange this may seem to the audience, the reason for this is that the generic definition form is nothing more than the definition, of the definition, of functions. The words are laid down in a very specific order, certain delimiters are used to separate words like, “modifier”, “return type”, “function name”, “parameter name”, and so on[6] that make absolutely no sense to the untrained eye. This aspect may seem unimportant, considering that such people do not get deep enough into the technical side of applications for this to matter, however this emphasizes how in the understanding process of human beings, structure means nothing without context and meaning.

If on the other hand such a person takes a look at an actual example of a function, he or she might be able to comprehend what the function is suppose to do, even if they had no connection with programming, provided they are familiar with the terms used in the function:

6 Notice the lack of underscore in the enumeration. Humans outside the technical realm, who do not understand the importance of space versus underscore from the perspective of parsing will not give any importance to it.

example: 10, Intelligible function example

```
Number divide(Number nominator, Number denominator){
        return nominator/denominator;
}
```

Even if this person has never seen an operation written like this, but has knowledge of the division operation between numbers, he or she can realize that this operation is what the layout is actually about. This is possible because the words themselves “number”, “divide”, “denominator”, “nominator” have very well defined meanings in the individual's intellect and together they hint towards the division of the nominator by the denominator.

It is not the structure of the function that leads the person to this conclusion, it is the combination of a certain set of concepts that together lead to a unique conclusion. It could be presented in any other form, as long as the person has knowledge of the division of numbers he or she would still come to the same conclusion. In fact the irony is that the function could actually do a completely different thing, with regards to the implementation, as long as the person does not see or cannot understand the implementation of the function, would still conclude that the function performs a division. This scenario however is not likely, because the fact that the function is laid out and named such is a consequence of the need of developers to embed additional knowledge in the description of the function which makes it recognizable and easy to work with.

Compilers completely do away with these human readable information, because programs do not need them to perform their pre-programmed jobs:

example: 11, Function stripped away of semantic hints

```
Number a(Number b, Number c){
        return b/c;
}
```

the definition in example: 11 is just as good from a computer's perspective, if only a programmer could keep track of functions that way.

The computer can do this, because the program, when finished, does not require meaning in the operations only series of correctly laid out operations that in the end can yield the expected result. It is the cognitive process of creating the program itself which requires additional knowledge to be embedded in these functions and the definition form of function allows that for most programming languages[7].

For simplicity, this knowledge that can be embedded in the description of the functions while building computer programs is not regulated. Neither is the order of the parameters, or their type. It is a generic scaffold that can be used to implement a great variety of operations, even multiple variations of implementations for the same operation.

The function:

7 Assembly language will not allow for such knowledge but languages like this are very rarely used and usually for small scale applications or modules.

example: 12, Division function with parameters in reversed order

```
Number divide(Number denominator, Number nominator){
        return nominator/denominator;
}
```

gives the same result as the one defined in the previous example but the input parameters need to be provided in a different order:

example: 13, Call of division function for different implementations

```
divide(10, 5) = 2, in case of the definition at example: 10
divide(5, 10) = 2, in case of the definition in example: 12
```

The program itself will not make this distinction, the programmer has to do it during the programming procedure and make sure the functions are used in the correct form. Failure to do so, result in errors in program that may be very difficult to find. For this reason, functions are usually accompanied by documentation which provide even more details about the way they behave, exceptional situations[8], parameter types, and so on.

Notice that although the order of the parameters were reversed in the declaration of the function, they appear in the same order in the body of the function:

example: 14

```
nominator/denominator
```

This is because the names themselves were chosen to represent the position of the number with the operation that the function is supposed to perform. In a division in mathematics, the concept of “nominator” represents the number that is being divided and the “denominator” the number that is being divided with. By using these names, it is easier for program developers to use the function. Had we used the conceptless representation like in example: 11, there would be no way of knowing which one is which. Within the context of a programmer's knowledge, such names, represent real information, because the names themselves are semantic charges that link whatever is being done there in the code to the actual reality that exists in the programmer's head and as a matter fact in the entire reality (mathematical reality in our case) shared by all programmers.

After compilation, functions become machine language. They are stripped of all meaning, leaving behind an optimized chain of commands, which the machine executes blindly until it reaches an exit point. This is what software is, stacks upon stacks of commands interacting with a layer of standardized interfaces and communication protocols making it very difficult to quality test an application: the machine can't do it, because it is not supposed to and it does not have the means, people can do it but they are basically testing a black box, which sometimes can be really complex.

1.4.4 ∘ Type Confusion

When applications are built, the business information scaffold is captured by the engineers, who create data structures by way of which a particular subset of the information can be stored, manipulated and if necessary communicated, Illustration 2. When such communication need arises, the engineers of the two applications agree on a common structure, which is used to encode information onto simple, highly standardized protocols, like HTTP, TCP, IP, Ethernet.

8 In the given example, division by zero is a mathematical impossibility and thus cannot yield a number as a result. This is an exceptional situation which is handled differently.

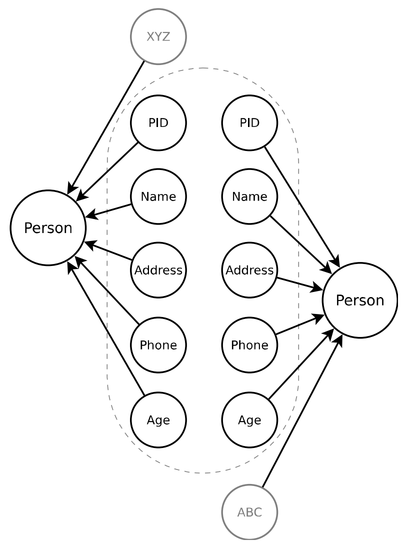


Illustration 2: Common interface, definition (source code) perspective

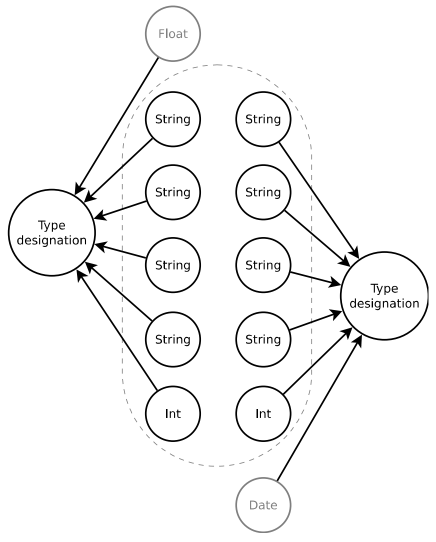


Illustration 3: Common interface, execution (compiled code) perspective

Information is transferred as an unstructured payload of bytes via these mediums and is then decoded based on this common structure or interface after which, the received data becomes available for processing at the destination. All these commonly used interfaces are grouped together in an API (application programming interface) and serve as a protocol for communication between the applications of the two parties (The Principles of a Semantically Rich Data Representation System).

But once the application is compiled and deployed the semantics of the information that are stored in the code are lost to the software the very same way they become lost in the case of functions. Names are removed and any semantics associated with them that would be capable to confer it information status is destroyed. Whatever is put in there it becomes data. As such, a human is needed at each and of the underlying applications to input and interpret the data and turn it into meaningful information again. The vast majority of data manipulation software today are not concerned with the meaning of the data that they manipulate. After the information passes the user and enters the system, it turns into a meaningless structure like in Illustration 3. This way it is imperative that an exact match exists between the structures used for information interchange, because it is the only constant that exists at that point (during application run-time) within the application. Failure to adhere to that will result in the data becoming deteriorated and the resulting information will be corrupted as it would be based on erroneous data. This strictness of the system and the lack of persistent semantics is an enormous impediment in the standardization of this layer of communication.

Needs regarding information capture and encoding are enormously various, there are as many as there are observers. Different businesses capture different characteristics of certain concepts, for example, a financial institution like a bank, would be interested in financial aspects of person, like income, employment, assets owned, a health institution like a hospital would likely be interested in things like body mass index, age, history of diseases in the family, allergies, etc, whereas an institution like an insurance company would probably be interested in a subset of both. There is no one single structure that can universally define a person, so if the three institutions are to interchange information, they need to agree on a strict API that is usable to all three. Unfortunately as the number of the potential participants in the communication raises and their focus broadens, the common interface becomes so clunky that it would be very cumbersome to use. Even if such, all encompassing interface, could be built and enforced, there still is the potential of a future business that would need yet an extra feature. This would invalidate the standard in place and generate an extraordinary effort to bring the industry up to date with the new standard.

Due to this lack of standardization, communication is limited to prearranged, pre-programmed, interfaces that are build into the software and very costly to change.

Ironically, many businesses in the world have in fact the same or very similar needs, some are even communicating using interfaces that are similar or exact matches of other, unknown businesses, yet the lack of context in the system blocks the capability of matching up these APIs, which leads to a continuous reinvention of the wheel and perpetual need for human intervention to discover, convene and integrate the communication protocols.

1.4.5 ◦ Semantics, The Meaning Of Things

Semantics, the meaning of things, what things represent, is a concept highly interlinked with the thought process, the consciousness, of humans and human existence itself. As such, it is very hard or even impossible to thoroughly define it outside this context, but it seems to have a lot to do with our ability to operate on or based on things.

It can be argued, very simplistically speaking, that the meaning of things is in fact the totality of actions that a person can autonomously perform with or based on those things, be that mechanical or mental in nature. The autonomous term is highly important, because if the person, or any agent[9] for that matter, needs “guidance” in performing an action with regards to a specific thing or cannot perform any action with that thing, it means that the object in discussion has limited or no meaning to him. These are very rear situations, because for almost anything that we pick from our reality there is something that we can do with it, not necessarily mechanically. This is completely natural because otherwise it would not be part of our reality, it would be meaningless to us the same way as atoms, quarks and other subatomic elements are irrelevant to the vast majority of humans in the everyday life and the same way as numbers are meaningless to the Pirahã.

If the meaning of objects to an agent stands in the activities that the agent can perform with these objects, the meaning of activities would stand in the objects they affect together with the activities that can be performed with these activities: equivalences, implications, triggers, etc. This blend of objects and activities is what the current paper considers as being the *Semantic Reality*.

Under this assumption, the meaning of things, cannot be perceived as a stand alone, absolute, concept, because it depends on the agent and it only makes sense in conjunction with it. One cannot ask *what the meaning of a rock is*. The question itself is nonsensical. One should rather ask *what the meaning of a rock is to them*. Depending on other factors too, like size, material, quality, one can hunt with a rock, can injure oneself or someone else with it, one can use it in a construction, polish it into a decoration, et cetera, a plethora of variations that trigger different potential actions or emotions and which have the capability to enrich the connection between the person and the rock with meaning.

But what does a rock mean to a computer? Absolutely nothing. The computer does not have any capability to perceive or perform actions on, or because of a rock, and so we cannot talk about any process of comprehending or understanding with regards to the computer and the rock. One could

9 An umbrella term covering entities that can perform activities in a voluntary or pseudo voluntary manner. The best example for this are humans, but this concept is also starting to encompass objects like robots, computers, or other complex machines that can operate on and with the surrounding environment.

argue though that a computer can be destroyed by a rock, so there is a semantical connection between the two, but this would be profoundly misleading. The semantical connection does not exist between the rock and the computer. This semantical connection is one, between the conjunction of the computer and the rock as object of the meaning, and the person that states the scenario as subject of the meaning, not between the rock and the computer.

The semantic reality of computer applications reside in the code, as a mechanism for action and data as object of meaning, on which the pre-programmed specific actions can be performed on an autonomous manner. It is important to consider this semantic reality conservatively. Performing certain actions autonomously does not mean full autonomy. Computers will not start acting on their own and possess free will, but there are many processes that can be automated, and that would provide enormous benefits to the industry in terms of cost, quality and security.

example: 15, Unit of measurement semantics

- both the concept of meter (m) and the concept of foot (ft) exist and they are universally accepted
- they are both distances or length (hence interchangeable) and that
- (m) = 0.3048 * (ft)

For example if units of measurement were to be considered universal primitive concepts, with terms like “meter” (m) and “foot” (ft) being in the collective reality and there existed an equivalence formula like in example: 15, a computer program can be written to find this correlation and provide (meter) to an operation that requires (meter) but is being provided values in (foot)s. An inference path exists between the two concepts and as such the aforementioned units and the associated activity represent a semantically rich fragment of the application's reality.

This may seem like very poor performance in human terms, but in terms of current programming techniques this could be an enormous step forward. A great deal of the code an application needs, deals with low level data validation, data integrity checks, consistency checks, as well as data access policy enforcement. Collectively, these represent a highly critical aspect of applications because they are responsible for most dangerous bugs, failures, loss of information through data hierarchy corruption, and security breaches. Consequently great resources (time, work and money) are dedicated to this aspect but even so, due to the enormous complexities associated, fast pace of changing needs and often lack of sufficient expertise, the results fall short of the expectations.

1.4.6 ◦ Object Oriented. The All In One Model

In a paper that was to appear in the Encyclopedia of Microcomputers (Chenho Kung, 1991) defines object oriented programming as:

"The object-oriented (OO) paradigm is a new approach for software development. In this paradigm, the real world is viewed as consisting of autonomous, concurrent objects interacting with each other. Each object has its own states and behavior, resembling their counterparts in the real world."

quote: 1, Chengo Kung, 1991

This twenty years old definition, which still very accurately defines the modern object oriented programming ideology, clearly emphasizes the technology's unique perspective towards software units: as self contained objects, that have characteristics, state, and means to perform actions. One of the principle paradigms of the technology, encapsulation, explains how these elements fit together creating black box like items, which have very clearly defined interfaces to show to the public but which hide inside all the inner workings, that are not essential to see when the objects are used. By doing so, clutter is reduced, making the code easy to navigate, understand and used. Other paradigms like inheritance by ways of which objects can inherit capabilities and characteristics from more abstract objects greatly facilitates the reuse of code and its portability.

But the object oriented technology is not at all concerned with meaning. These objects that are supposed to mimic the real world objects, have in fact only marginal resemblance to their real counterparts. The paradigms of OO do not enforce at all any strictness with regards to reality but rather only the principles of modeling code in a certain type of way, which greatly increases coding efficiency.

example: 16, Encapsulation of state and behavior in object oriented programming

```
Book{
        String title;
        String author;
        Date chekOutDate;
        int durationDays;
        String borrower;
        // -----
        void chekOut(String borrower, int durationDays);
        void checkIn();
        String getTitle();
        …
}

Library{
        Book[] books;
        // -----
        void checkOut(Book book, String borrower, int durationDays);
}
```

The class Book in example: 16 encapsulates functionalities and attributes of a library book and it makes a lot of sense to endow it with the capability to check itself out from the library in the name of a borrower and even to store within, the details of the borrowing. Of course, in the real world, books don't check themselves out of the library, such a thing is not within the capabilities of a real book and it would be severely counterproductive to store the details of the borrowing in the book itself, as they would be unavailable unless one actually had the book, which is not the case when one lends something. As such, it would make a lot more sense to endow the library with the capability to check out books but if we think about it for a second, this is not real either. Both approaches are just ways to abstract activities and group them so that they would be intuitive to use, portable and easy to reuse. It is really at the subjective opinion of a developer to decide which method is optimal for the given situation.

This is not to say that Object Oriented Programming and the object oriented model are not good, but just as well, it is a fallacy to consider that this efficient model of programming is suitable for every kind of information manipulation. Yet as pure object oriented programing languages are becoming more and more frequent, the boundary between objects that “do stuff” and objects that “describe stuff” has almost completely disappeared.

This blurring of the two aspects stems in the natural desire of developers to benefit from the various efficiencies object oriented programming offers for certain situation. Objects that “do stuff”, active object so to speak, perform their job by ways of methods, code encapsulated into their construction. This code, which can add up from thousands to millions of lines of code in an application has been the major focus of attention when it came to optimizing, for it represents the vast majority of the development effort. As such, any optimization in portability or re-usability brought considerable reductions in cost. By contrast, objects that “describe stuff”[10] represent only a tiny fraction of the code. So small that we cannot even talk about optimizing this layer with regards to implementation time.

The problem with this approach is that lately portability of the types that “describe stuff” has become more and more important and the subjectivity of the developers and client needs, that are inherently embedded in these types by the OO model are in sharp contradiction with the need for their universal acceptance.

Another enormous impediment in information portability and freedom that is inherently built into the OO model is the way it models relations. The common practice in OO programing is to embed the relation, be that [1 - n], or [m - n] into the construction of the type itself. While this may bring some benefit in working with these objects from code portability it probably represents the apogee of inflexibility with regards to information portability:

example: 17, Relation representing books belonging to a library

```
Library{
        ...
        List getBooks();
        ...
}
```

If any new relation needs to be added to the type definition hierarchy, it requires changing the type itself, which in the best case requires modification of at least one type at source code level, recompilation, redeploy and restart of the application. As such, a type can never acquire universal acceptance because there will always be particular needs for particular relations, even if the type itself is universally standard. Such particularity keep types in the constant and total chaos of subjectivity.

1.4.7 ◦ Comparison With The Relational Model

The way individual complex types are structured in the relational model is very similar to the object oriented model with two the major differences.

Object oriented model allows inheritance of properties and abstraction of types, whereas in the relational model all components of the structure are redefined. As such, the area of a circle as defined in the “Circle” type has nothing to do with the area of a square as defined in the “Square” type. Consequently, there is a lot of redundant terminology that can be confusing to a developer because the human mind, will constantly try to associate the two, as they have common semantics.

10 In OO, representing only the part of the type that defines the attributes of the object and none of the methods. The methods are part of what objects do.

The second major difference that the relational model has relative to the OO model, is that it sees the relations between types as a separate concept, unrelated to the types themselves. In fact it doesn't really take any sides between types and relations as both are considered relations. This characteristics gives greater flexibility to the data model, because the addition of a relation, which can occur later on in the use of the application, does not change the structure of the types. They can be added as needed throughout the lifetime of the application. This flexibility and the simplicity of the model allowed for extremely good standardization, which makes the relational model and the relational databases extremely successful to these days.

With object oriented development becoming more and more prevalent due to its efficiency of code reuse, changing the minds of developers into thinking more are more in terms of objects, there is a rising confusion in the field of data manipulation. Storing data in one model and manipulating it in a different one, especially when the difference between the two are significant, needs a lot of overhead work. A separate layer needs to be created to transform back and forth the data structure as well as the searching criteria, which sometimes can prove very difficult. Additionally, the difference in the way the two models conceptually encode information inevitably gives rise to a lot of conceptual conflict as well, and different developers see the transition between the two in different way.

To simplify this, some advocate the creation of object-oriented databases, which would do away with the overhead of coding and the conflict of ideologies, because in the object model, both, members of complex types (structures) and the relation are reusable via inheritance. By wrapping both members and relations under the same umbrella, the object model, allow easy implementation of programming operations like deep comparison, deep copy or deep deletion, operations that can potentially create a lot of data inconsistency and programming overhead in relational systems which do not necessarily enforce these dependencies.

But the subjectivity of structures associated with the object model and the rigidity created by wrapping relations into the types have taken their toll on the standardization and portability of the model. Object databases are by far less versatile and as such there are many different implementation that found their way only into niche applications.

The legacy of both these major data models is that they originate in an era when applications were not meant to communicate with each other. The vast majority of applications resided at most on a local area network or a virtual private network serving one business or one cohesive group of consumers. This meant that the abstraction of data, the pattern created to capture information into it's meaning free form did not intersect with the property of subjectivity. It was as if individuals, observed information from their own particular points of view and little did it matter that other individuals observe the same information but from an alternate subjective angle. Information losses associated with the abstraction of information could easily be factored in from the beginning. Because of this, information, had that absolute character data has. The two, information and data, were conceptually almost interchangeable as the abstractor was the same with the consumer. Whatever wasn't encoded into the data, was in the head of that who encoded it, who in the end was also responsible with decoding it. This simplification however can only exist under the premise of the lack of subjectivity.

But not so long ago came a time when applications started to break the barrier of individualism and transfer of information (not data) started to become more and more important in the information industry. This however proved a very difficult nut to crack because the moment this need arose, the encoded information lost its absolute character and encoded information suddenly became what it really is: data and not information. But the industry continued to treat it as if nothing had happened and attempted to apply to information, everything that learned and developed for data.

1.4.8 ◦ Web Services, The XML Promise

Shortly after the XML standard has been released, a plethora of communication related technologies appeared that were based on it: XML RPC, XSD, WSDL, UDDI. There was great optimism following the release around year 2000 and the industry envisioned a world in which computer to computer communication would soon become commonplace enabled by such technologies.

But in spite of their effectiveness in structuring information, the enormous number of business supporting them and unprecedented consensus with regards to the direction of technology, these standards failed to deliver that promise. The reason why, has nothing to do with *structure* and everything to do with how *meaning* is seen in the world of computation.

◊ XML

The Extensible Markup Language (XML) appeared out of the necessity to be able to exchange richly structured documents over the web, akin to HTML, which was too rigid for this purpose, having predefined tags and as such being limited to a certain kind of data. The general structure of an XML is very similar to the HTML counterpart example: 18.

example: 18, Person structured in an xml file

```
<Person>
        <PID>pidvalue</PID>
        <name>John Doe</name>
        <address>
                <street>Elm Street</street>
                <number>1234567<number>
                <country>Nowhere</country>
        </address>
        ...
</Person>
```

As opposed to HTML, XML can encode virtually any kind of information, because the tags are not standardized. XML however, was never meant to carry any semantics. In an article, A Technical Introduction to XML (Walsh Norman, 1997) was writing:

> *"XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by stylesheets."*

quote: 2, Walsh Norman, 1997

Owing to this absolute freedom for structuring there is no way to tell what actually is encoded in the XML itself. In an XML, the tag <name> does not mean anything; it could just as easily be <eman>[11] or anything else. This meant that there was no way for two applications to exchange information this way unless they agreed to rigid structures by ways of which they encoded and decoded the information. To put an end to the confusion and to allow a more general use to the XML, the XSD standard was defined, which is in turn an XML structure meant to standardize definitions in XML format.

11 Name written backwards

◊ XSD

Published in 2001, the XML Schema Definition (SXD) is a schema language which allows for the definition of restrictions in the structure of XML documents such that there could be some general common ground in the definition of types during information interchange via the web. Being specially thought up with type definition in perspective, the XSD standard also defines the basic data akin to most programming languages: String, decimal, dateTime, time, float, double, etc. (19 total) and XML schema elements that allow for construction of structured types, example: 19, like Person or Address from example: 18.

example: 19, XSD schema example

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="Address">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="Recipient" type="xs:string" />
				<xs:element name="House" type="xs:string" />
				<xs:element name="Street" type="xs:string" />
				<xs:element name="Town" type="xs:string" />
				<xs:element name="County" type="xs:string" minOccurs="0" />
				<xs:element name="PostCode" type="xs:string" />
				<xs:element name="Country" minOccurs="0">
					<xs:simpleType>
						<xs:restriction base="xs:string">
							<xs:enumeration value="IN" />
							<xs:enumeration value="DE" />
							<xs:enumeration value="ES" />
							<xs:enumeration value="UK" />
							<xs:enumeration value="US" />
						</xs:restriction>
					</xs:simpleType>
				</xs:element>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

Unfortunately this additional layer of structuring still doesn't bring types closer to containing any universally evident meaning. With all the effort that was put into it, the XSD only brought the global data type definition to the point where all the programming languages already were, with the added benefit that this time there were no programming language dependencies and the definition language is universally accepted.

◊ XML-RPC, WEB SERVICES AND THE UDDI

XML – RPC stands for (XML encoded Remote Procedure Call) which is a way of invoking functions (methods) in an application from a remote location.

There are obvious complications with invoking methods remotely, the most obvious ones being that the remote machine does not have access to local pointers, stack, memory, type definitions, and so on, so an RPC mechanism is designed to encode all necessary information into a network message on the consumer, transmit it to the provider, decode it, interpret it by matching it to a local call, obtain a result if necessary, encode it and send it back to the consumer. These providers have become known in the modern networking systems as Web Services, and are very similar to the concept of the object in OO development, comprising a set of functions that can be invoked by potential consumers, but in this case, over a network.

Abstracting from the intricacies of the implementation of RPC systems it can be seen that together with WSDL (Web Service Definition Language) it is in fact a standardized definition of the concept of interface as understood by most object oriented developers (a definition of functions that must be coded into all classes declaring / inheriting that interface). just as in the case of XSD the definition of these Web Services (objects serving over the web) is independent of any specific programming language.

This independence from programming languages of both type definition and interface definition gave great hope to the IT industry with regards to automating communication between software applications (business to business or B2B). The UDDI (Universal Description Discovery and Integration) registry was supposed to contain a vast collection of such interfaces, published by businesses, and as such all businesses could potentially interconnect and exchange information, example: 20.

example: 20, Accommodation search automation

An application could consult a UDDI and automatically discover all applications that offer room for rent at "Destination XYZ", and consult whether they conform to certain criteria: star category, price range, etc.

The reality proved different, in spite of unprecedented support from all major players in the industry. By 2010 most of them were retracting support due to lack of interest and adoption and the concept of UDDI was dead.

1.4.9 ◦ Why These Standards Fail To Deliver

The simple answer is, *they did not*. They delivered exactly what they were supposed to. They defined a platform independent framework that technically allow any two software to invoke methods on each other and exchange data based on a common definition. It was the *expectations* that were *unreasonable*.

The industry expected that creating a platform independent structuring infrastructure could solve the communication problem by allowing the definition of common types. But the desire to do so was not taken into consideration. The wide acceptance of these common definitions were impossible because there is *no such thing as common definition*. There only is a common framework to create these *definitions* and these frameworks suffers from the same problem as the type system in each individual programming language: the lack of computational semantics.

Being able to define data in a platform independent manner does not magically transform data into information. It does allow the transfer of data over the Internet as opposed to having it locally but beyond that, putting it into context, interpreting it, was still the job of the operator. No steps have been made towards the preservation of meaning during the encoding process, meaning which could potentially be transferred together with the data.

All these type systems concentrate on standardizing the way information is represented, not what information is. The primitive (most basic elements) are very basic representation systems that can partially encode various computer primitive concepts like integer or boolean onto machine language. This is essential, because ultimately information does need to be encoded, but not sufficient. The system jumps from these primitives directly to structures, the elements of which are either structures themselves or primitives. There are no intermediary types, primitives that can also carry some meaning, not only representation. If a member of a structure is a "string" that structure cannot be standardized, because a string can be anything, and anything is just not standard in terms of computational potential. As such, an application cannot test on its own, or more precisely, one cannot build an application that can test in an automated manner what an input or output parameter is.

example: 20 might have seen a little misplaced, because hotel accommodation search and booking is one of the very few niches that managed to standardize its communication. This however, is not because seamless communication exists, but because of an arduous and very difficult standardization campaign. All hotels that conform to the standard must implement rigorous proprietary middlemen

APIs that can make the connection between booking, hotel and the browser. If a hotel wants or needs to conform to more than one middlemen (in order to reach more clients) the developers of the hotel's application need to implement all APIs. There is no UDDI like system and even if it were it is still the developer's mind, that needs to read the documentation, see the types, make the connection, and implement information exchange. There is no information in the applications, the information is only with the developer or hotel personnel or consumer.

This lack of common semantics, the rigidity of the data model and the subjective perspectives of many different groups of businesses resulted in the creation of many disparate type hierarchies that are essentially incompatible outside groups. If a concept / a property is added to a structure the entire structure needs to be changed leading to continuous redefinition of the types and impossibility to having a stable basis for communication. Changing these types means that types need to be backward compatible, somebody needs to maintain a record of the types and their versions and at the same time enforce their proper usage. Because such things are impractical, applications, usually sooner rather than later, need to be updated to using the latest types or they become outdated and incompatible with the rest of the applications in the market, leading to the breakdown of the communication system.

The inflexibility of this holistic model is a major impediment in the domain of communication, at information level, within software applications. The massive amount of work that needs to be continuously invested means not only extra cost in development, but also a lot of superficial implementation, countless bugs, incompatibilities and major security vulnerabilities.

1.5 ∘ Semantic Web And RDF/OWL Ontologies

As a response to the lack of meaning in these API based communication systems, which was observed quite early in the development of the Internet, a new standard started to emerge.

The Semantic Web was coined in 1999, around the same time as the B2B and is defined as:

> *"a web of data that can be processed directly and indirectly by machines."*
>
> **quote: 3**, Tim Berners-Lee

by the inventor of the World Wide Web, Tim Berners-Lee and it was / is suppose to revolutionize information search over the Internet (hence the moniker Web 3.0). To achieve this, frameworks have been created like RDF (resource description framework) and OWL (Web Ontology Language) that can be used to create *semantically charged data* that can be published over the web. These are necessary because computers cannot process natural language for their content and searches are based on keyword matching and different tricks which don't yield results as expected. Similarly to the B2B promise, the Semantic Web promises a world in which machines can automatically process information on the web and return meaningful answers, not just answers matched by character comparison.

Unfortunately, the structuring trend is so prevalent in the IT discipline that even developers of semantic web, fall into the same trap of defining terms that are only humanly intelligible. Frameworks like RDF and OWL are by nature the same structuring languages that allow definition of types and type hierarchies (ontologies in this case), as seen in the case of XSD. The difference is, that in the case of ontologies, the types are supposed to carry semantic charge because the model puts *some* emphasis on the terminology and the relation between elements.

It will be shown later in the work how this semantic charge it is in fact impossible to achieve the way the ontologies are constructed today.

The body of people, that create such ontologies, are responsible to embed these semantic information into the ontology in such way that machines can automatically make use of it. Unfortunately, the misconception seems to be present, that terms and relations alone, actually hold the semantics associated with the object. Evidently, when humans look at these terms and relations they all make sense, but they make sense in the observers intellect not necessarily within applications. In human world, terminology is important not because it is semantic in itself, but because it links to us, to our reality (objects, actions, et cetera), which in turn is semantic to us because we can operate with those objects and relations.

By analogy, given a specific ontology, the semantics will not lie in the terminology and definitions listed there; that is only semantic to humans. It will lie in what applications are able to do with that ontology in an autonomous manner. From this angle, this linked resource model is no better in terms of machine semantics than the classical model, using the APIs, it is just different: it does not operate on structures and primitive types, it operates on terms, objects, types and relations between them.

1.5.1 ∘ Ontology, The Linked Data API

Because today's information systems work predominantly with data, the process of transforming this data into information lies in the programs themselves, which together with the end-user, are putting data into context thus transforming it into information. This is adequate for many operations but in some cases it would be useful if information systems would be capable of transforming at least part of the data into information and manipulate that information into creating more concise, humanly manageable results: A web search is very good such example, where the cause for the massive amount of inconclusive responses is a result of applications treating page content as data and not as information. They can match text in the search, but they cannot put that text into context and so they cannot give answers to questions but rather just statistical matching between texts.

Ontologies are frameworks that allow information, not data, to be transposed into computing environment, in such ways that it is possible to perform an analytical information extraction process instead of a simple statistical matching. To put it in perspective, the two representations:

example: 21

1. "John is Human" - data represented as a sequence of characters
2. "Is (John, Human)" - information stored as a proposition

In the first case a computer program is capable of finding occurrences of texts like "John" and can respond whether the text can be found in the given source text or not, whereas in the second case, a computer can actually observe a relation that exists between John and Human. As such, from the standpoint of the "is" relation, the second structure is actually information, not only data.

In the world of computer science, *ontologies*, are information packages which can be used by a computing system to perform context dependent operations. In this case, the context is the ontology itself, and the information declaration is the data pool on which operations take place. To return to the upper example, a very simplistic ontology for the proposition 2. would contain the definitions of:

example: 22

1. Human – *living human being*
2. Is – *a relation denoting inclusion in a class or set*

As such, a data package that contains the word John and the relation *Is (John, Human)*, could tell a computer system that whatever "John" is (or whatever the character array [J, h, o, n] stands for), it is a Human, within the context of the Ontology. This makes the ontology the context that gives the data in the packet information character.

The Holy Grail behind the ontology is an ontology that is generic and complete enough to serve as context for any data that can be extracted from the human world. If that were true, there could be a computing system that could answer virtually any question in a pertinent way, akin to artificial intelligence, rather than just search for raw occurrences of sets of characters. Unfortunately, the human existence is so complex that all present ontologies fail either the generic or the complete criteria and in most cases both. This gives way to opinions and trends to emerge the result of which is many, many, ontologies in most cases conceptually overlapping.

1.5.2 ◦ Ontology, A Concrete Example

To highlight the resemblance of the Ontology and the API models, from the perspective of outcome and utility, let's consider an example from an existing ontology.

FOAF (Friend of a friend) is an ontology defined on RDF and OWL which aims to describe people and relations within the context of the Internet, on-line presence. It defines a hierarchy of types which can be complex or simple and are connected between them via properties and relations:

example: 23, Snipet from FOAF Ontotlogy

```
Class: foaf:Image
Class: foaf:OnlineAccount
Class: foaf:OnlineGamingAccount  (Subclass Of: Online Account)
...
Property: foaf:mbox_sha1sum
Property: foaf:msnChatID
Property: foaf:lastName
Property: foaf:account (Range: every value of this property is a Online Account)
...
```

The semantic charge, however, from a computer's point of view, is similar or identical in nature to other classical APIs, modelled in XSD or UML (unified modeling language). This is not a problem of RDF or OWL languages, but rather a problem born from the way the industry sees data modeling: abstracting a slice of reality into a custom, proprietary model.

FOAF grabs a piece of reality, that of people and the web, disjoints it conceptually from all the rest of reality and attempts to represent it in such way that applications can make sens of it without the aid of humans. Applications that are specifically built to interpret FOAF will undoubtedly be capable of interpreting FOAF and give their users relevant response within the realm of FOAF reality, but this is no different that any other application implementing any other API. Aside from that, any other application that is not strictly designed to conform to the types defined in the FOAF ontology will be incapable to interpret any result. Why this is, has to do with the reliance of the ontology on humans to interpret it. There is no consistency in naming, real continuity between types and no link to a larger reality (ontology) to which references are made via suggestive naming and explanation (documentation).

The “OnlineGamingAccount” in example: 23 is a derivative of the “OnlineAccount”, which in turn is a derivative of “Thing” in the FOAF ontology. While the text in the description of the these types make complete sense to a person, from a computer's perspective it could be anything. In order for a computer to be capable to perform any operations with a foaf:OnlineGamingAccount, in an automated fashion, the computer has to have some operations defined that take it as input and some mechanism that trigger an automated reaction. If it does not have such an operation, it should at least be capable of doing something with foaf:OnlineAccount. If neither of these are implemented (hard coded) to take these very types as input no operation can be performed because there is no other computationally intelligible information related with the type. These are self contained types, they either make sense on their own (operations exist) or they don't. They are completely disjoint from the reality (the human reality) from where they originate.

A person on the other hand is capable to derive lots of information from the simple name “OnlineGamingAccount”. Although the designation is merged into a single word distorting slightly the meaning this way, it is sufficiently similar to “on-line gaming account” so that a person can deduce that this is what the type is about. The person, can immediately draw the conclusion that it is:

- an *account*, something representing him or her or another person
- *on-line*, in the on-line world (Internet)

  the upper two, hint towards a plethora of collateral information: people log in with these account, they have a profile, information accumulates into these accounts, actions can be performed with the information and so on....
- it has to do with *gaming*, as in play, fun, and so on and so forth...

None of these distinct concepts (account, on-line or gaming) are defined in the FOAF ontology. It is either an foaf:OnlineGamingAccount, an foaf:OnlineAccount or nothing at all and as such, a computer built around the foaf ontology cannot draw any additional information from an object of this type. There is simply no reality behind these types within the ontology.

What is worse is that the ontology is organizationally controlled: The concept “foaf:OnlineGamingAccount” is the property of somebody. The term “xyz:OnlineGamingAccount” could be the property of somebody else, who might chose (not that they will, but the possibility exists) to define it in terms of carrots and potatoes rather than gaming and on-line. The example might be exaggerated, but it is meant to show that such terminology has nothing to do with reality, or *a* reality. They are proprietary terms used by particular organizations who presume that the entire world will conform to the standard and thus enable seamless communication within that particular sector. The very concept of Ontology is slightly misplaced as it implies knowledge about a slice of reality. In human world an ontology is usually a small segregated part from the global reality, which is particular to a group of people and it only emphasizing the particularities of that group of people within the context of this global reality. In IT, ontologies are defined outside the context of this general reality, which continues to exist only in the human world. This segregationist view is very akin to an API, containing a proprietary type hierarchy that can be used for information interchange but only if the ones doing the interchange are sufficiently aware of the context to recreate information from data, because at the root that is what is being interchanged. For anybody else, putting reality together from these separate ontologies will inevitably lead to confusion, double definitions, subjectivity, conflict of interest and all the other problems of the classical APIs.

Ontologies need to emerge naturally from custom needs for various groups of people and it is the more basic, general world that needs to be coined. As long as the way of creating them stays as it currently is, these ontologies cannot bring the awaited revolution because people will never be interested or be able to using it. For example, the specification describing the FOAF (Dan Brickley, Libby Miller, 2010) is so long, that no human will be enticed to read and learn it just to be able to say a few words about themselves in the new semantic web. It is many times longer than the semantic meaning (dictionary definition) of the objects that it defines, and it represents only an insignificantly tiny fraction of the knowledgeable information within human reality. If all knowledge would be standardized like this, it would become an unreadable document. If the alternative method is presumed, that an application will help the humans to create these files, then we have to assume that in a complete Web 3.0 environment humans will be using thousands of custom made applications designed particularly for each ontology that will exists out there, which is just as unreasonable.

In either case, it is not likely that the current approach will be able to yield any real results outside small scale studies or strict lab conditions.

1.5.3 ∘ Upper Ontologies

Because the ontologies are the contexts that give data information character, the representation of data must adhere to the ontology or it cannot be observed, even if the information stored within the data would otherwise be sensible. To allow for interoperability between ontologies, the concept of **upper ontology** was defined which aims to connect ontologies and therefor allowing for translation between definitions from one ontology to the other.

Suppose we have ontology **O** consisting of the terms from example: 22, and ontology **O'**, defining the same concept in a slightly different way, example: 24.

example: 24,

1. Homo Sapiens – *living human being*
2. Belongs To – *a relation denoting inclusion in a class or set*

The proposition *Is(John, Human)*, defined in the context of Ontology **O**, would not make any sense within the context of Ontology **O'**. The fact that the reader is capable of recognizing the similarity and hence the value of truth of the proposition within the context of the **O'** ontology is result of the fact that the reader possess a larger context (ontology) in which there exists an equivalence between the terms defined in the two ontologies, but a computer simply cannot.

example: 25, Concept equivalence

1. Equivalent To – *denoting equivalence rule between concepts*
2. Equivalent To(Belongs To, Is) – *concept equivalence definitions*
3. *Equivalent To(Homo Sapiens, Human) – concept equivalence definitions*

example: 25, is what an upper ontology (**UO**) stands for. With *it* in place, a computing system operating within the context **O** and **UO** should be able to make sense of a piece of data defined in **O'**.

It is important to observe the power behind the concept of an upper ontology, in allowing the creation of equivalence between data definitions. If there wasn't an upper ontology UO, in order to state the same information under both O and O', we would have two distinct data sets, which would yield two different information sets under two different ontologies. There would be no way to recognize that there is an equivalence between the two information. They would be segregated forever each in its own context, but with the upper ontology in place, not only can the computer recognize information from a different (foreign) ontology, it can also tell that the information is equivalent.

1.5.4 ◦ The Ontology Maze

Unfortunately the complication with ontologies does not end with the appearance of the upper ontologies. The human knowledge system seems to be too complicated for any *one* upper ontology, or the opinions on how to build them part, so instead of one, many upper ontology systems exist, each of them with its own particular knowledge representation and knowledge base. Let us see the methodology of some of the most renowned ones briefly.

◊ CYC

The Cyc Ontology is a proprietary ontology developed by Cycorp Inc. The Cyc ontology is claimed to be “universal” meaning that it can accommodate any concept regardless of context through a three layer ontology system “upper”, “middle” and “lower”. The upper layer contains the more generic, broad and highly structural concepts, such as temporality, dimensionality, relationship types, etc. the middle layer, which contains widely used concepts, generalities and the lower layer contains leaf level knowledge, in other work specifics. Each level ties into the definitions from the higher level through rules(An Introduction to the Syntax and Content of Cyci).

Each element is considered to be a Cyc constant and is defined as:

#$ConstantName
comment text in natural language, ...
isa: a list of representative sets whose member #$ConstantName is
genls: a list of representative sets whose subset #$ConstantName is, if any*

* genls, will only be specified for concepts that are collections denoted by their connection through an **isa** chain to the concept #$Collection.

The **Knowledge**, the facts, are represented in Cyc via the predicative logic concept, throught Cyc **#$Relations**, like Predicates and Functions.

Predicates are logical relations that can be thought of as functions that return true or false, and their definition contains the additional description of the parameters they accept. For instance in the case of a binary relation:

```
#$mother: <Animal><FemaleAnimal>
```

which denotes that the predicate #$mother takes two parameters the first of which must be a member of the #$Animal collection and the second that of #$FemaleAnimal collection. Just as in the case of constants explanation is given in natural language to avoid confusion.

As an important side note is that the natural language explanation is not part of the Knowledge Base as a computation tool. It does not aid the computer program working with Cyc, but rather the developer of the computer programs or that of the ontology.

Additionally, Cyc also defines functions, which are constructs that receive parameters, like in the case of predicates and return results other than true or false:

```
#$ FemaleFn: <OrganismClassificationType>
        comment text in natural language, …
isa: #$CollectionDenotingFunction
arg1Genl: #$Animal
resultIsa: #$ExistingObjectType
arg1Genl: #$FemaleAnimal
```

The Cyc ontology is self referential similarly to mathematics. All constants and terminology are on higher level definitions which in turn are rooted in the upper layer axiomatic definitions. It defines over 5,000,000 assertions between approximately a 500,000 constants (concepts) out of which approximately 26,000 predicate constructs(ResearchCyc).

◊ U M B E L

The Upper Mapping and Binding Exchange Layer (UMBEL), is a subset of the OpenCyc provided in an RDF (Resource Definition Framework) Ontology and according to the creator it is designed to facilitate content interoperability on the Internet.

UMBEL vocabulary is structured around three root classes, and 38 root properties. In addition it employs external vocabularies such as RDF and SKOS. UMBEL definitions follow the pattern (Upper Mapping and Binding Exchange Layer (UMBEL) Specification):

```
Class name      -     umbel:RefConcept
Description     -     Distinct subsets of broadly understood concepts ...
in-domain-of    -     umbel:isRelatedTo, skos:prefLabel, skos:altLabel, skos:hiddenLabel,
skos:definition
in-range-of     -     umbel:isAbout, umbel:correspondsTo
Sub-class-of    -     skos:Concept
```

The base classes consist of umbel:RefConcept, umbel:SuperType and umbel:Qualifier, and the properties: umbel:correspondsTo, umbel:isAbout, umbel:isRelatedTo, umbel:relatesToXXX (31 variants like: relatesToSubstance, relatesToEarth, relatesToHeavens, …, relatesToFinanceEconomy, …, etc.), umbel:isLike, umbel:hasMapping, umbel:hasCharacteristic, umbel:isCharacteristicOf, each accompanied by definition and specific annotations.

The **Reference Concepts**, are drawn from the OpenCyc librarary which consists of approximately 28,000 Concepts and are divided into 33 mostly disjointed **Super Types**.

UMBEL's hierarchy is similar to Cyc, but because it emphasize interconnection, it consists only of taxonomy: definition of the classes and relationship between them. UMBEL does not contain relationship between instances (assertions) as it is not a knowledge base but rather a reference ontology meant to provide support for interconnecting disparate ontologies.

The in-domain-of and in-range-of sections list the properties, defined in some external ontologies, that can be used to describe that subject concept and a continuous effort is being made to tie existing ontologies into UMBEL.

◊ B F O

Basic Formal Ontology is a different ontology maintained by IFOMIS (The Institute for Formal Ontology and Medical Information Science) and it consists of a series of sub ontologies divided into 2 broad categories: continuant (SNAP) or snapshot ontologies encompassing a snapshot of ontologies describing concepts that endure through time (time invariant), and occurent (SPAN) encompassing ontologies describing processes that have a time dimensionality(SPATIAL COGNITION AND COMPUTATION).

As part of the ontology, BFO employs Logic Programming, namely predicative logic, to define connections between the concepts it defines. These connections (predicates) are also divided into the two major categories: there exist SNAP predicates and SPAN predicates. As a major criteria is that relations never span across the both domains: they are either SNAP or SPAN. For example the part(…) relation exist for both SNAP and SPAN but the parameters that they take will only be from one category: a leg, is part of a person and childhood is part of life. BFO does not allow the leg to be part of the life.

As a second baseline differentiation, BFO divides concepts into two categories: Universals and Particulars. **Universals** stand for the broad categories such as Types, Species, Classes, and **Particulars** stand for individual representatives of these. If Homo Sapiens is a Universal, than an individual human being would be a particular (Ontology for the Twenty First Century: An Introduction with Recommendations):

- Continuant
    - Independent continuant
        - Site
        - Object
        - Fiat part of object
        - Boundary of object
        - Aggregate of Object
    - Dependent continuant
        - Quality
        - Realizable entity
            - Function
            - Role
            - disposition
        - Spatial region
            - 3D (volume)
            - 2D (surface)
            - 1D (line)
            - 0D (point)
- Occurrent
    - Temporal region,
    - Spatio-temporal region
    - Processual entity
    - Process
        - Process Aggregate
        - Fiat part of Process
        - Processual context
        - Boundary of a process

BFO defines a total of 36 classes connected vertically via the **is_a** relation. Additionally to these classes BFO also defines a series of primitive relations between these classes:

- Constituent: Entity $\times$ Ontology
- Part: Entity $\times$ Entity
- InhersIn: SnapDependent $\times$ Substantial
- TemporalLocation: SpanEntity $\times$ TimeRegion
- Exists-At: SnapEntity $\times$ TimeInstance
- TemporalIndex: Ontology $\times$ TimeRegion
- SpatialLocation: SnapEntity $\times$ SpaceRegion
- TemporalLocation: SpanEntity $\times$ SpacetimeRegion
- ParticipatesIn: Substantial $\times$ Processual

1.5.5 ◦ Some Comparative Aspects

A few comparative aspects have been picked which may be key in the easy of adoption (understanding, development, extending) from both computational aspect and human perspective.

	Cyc	**UMBEL**	**BFO**
Directionality	Vertical Hierarchy, multiple ancestor permitted. Horizontal connections via functions & predicates	Strict vertical divergence via type hierarchy. Horizontal connections via properties	Strictly diverging vertical Hierarchy via Is_a relation. Horizontal connections between certain predicates. (Domain constraints may apply)
Grammar	Predicate Logic, Functional Programming	Predicate logic (relations)	Predicate logic (relations)
Constraints	Very few. 3 layers, lower layers tie coherently into upper layers	Taxonomy only	Conceptual differentiation between time dependent and time independent concepts, universal and particulars, dependent and independent.
Base Vocabulary	500K concepts, 26K relations	3 root classes & 38 properties	36 Classes
Definitions	5M assertions	28000 concepts grouped in 33 mostly disjoint supertypes	36 Classes
Form of Definitions	Agglutinated Natural Language words (in CamelCase)	Agglutinated Natural Language words (in CamelCase)	Agglutinated Natural Language words (in CamelCase)
Language	CycL, OWL	RDF, OWL	OWL

The widest adoption is enjoyed by BFO(Basic Formal Ontology Users)(UMBEL Projects), which it can be seen, is the simplest most restrictive of all. Although grammatically the restrictions represent limitations, from the human perspective they are easy to understand and potentially boost the capacity of developers to build quality ontologies based upon it. It is flexibility versus adoptability: while 15K relations give one the flexibility to connect concepts in an extremely complex manner it also pose an enormous obstacle in finding the appropriate relations for reuse, avoiding double definition for relations, and preserving consistency across the domain.

Specifically, *terminology invention* (see: 2.1.3 simplicity & Familiarity), which is a characteristic of all ontologies (both upper and lower) poses a great barrier in their adoption. The sheer volume of the *invented term,* which range from thousands to hundreds of thousands, makes them impossible to remember and in some cases even impossible to found should a person even have the intention of finding them.

1.5.6 ◦ The Demotion Of Upper Ontologies

The use of upper ontology systems in binding disparate lower ontologies together is a commendable effort that has the potential to improve interoperability in the domain of knowledge representation and information manipulation. But in spite of the major efforts that have been put into their adoption remains strictly in the boundaries of isolated projects. On the other hand, if the power of the semantic web lies in the adoption of the concepts than perhaps a simpler system, which may not allow full representation of human reality, could do more for the semantic web than the complex ones that claim to be able to do so.

The moment dissensions entered into the world of upper ontologies and multiple variations of them appeared, marks the moment of downgrading such ontologies to the rank of ontologies. Similarly as Cyc has different layers so are these ontologies to a certain degree above a subset of other ontologies, but relative to other so called upper ontologies they are in fact parallel and therefore have no structuring power over them. Application designed for an upper ontology might be able to handle sub ontologies, but they will not be able to explore ontologies on the same level.

As such it is really inappropriate of talking about an upper ontology in the same sense as reality around us is relative to our understanding. Our reality is one, we can always count on it to be one, we can measure against it because it is absolute[12]. It's existence does not depend on who observes it, only the information collected from it changes. A true upper ontology should not depend on predefined hierarchies, but rather only on universally acceptable truths. An upper ontology should be unique and universally objective.

1.6 ◦ Fractured Realities

The current mainstream approach in the creation of either APIs or Ontologies is for an organization to take charge and create a set of types, terms, connections, anything involved in the specific definition and then push it out to the community to use. As part of the contract, the organization gets to put the collection of type, terms, etc. under its own brand and be the sole governor of the package (like foaf:OnlineGamingAccount). But given the nature of information, it is both presumptuous and naïve to believe that the industry will have the will or even the interest to simply concede to adopting such API or Ontology, and for numerous good reasons:

Businesses might simply not like the fact that somebody else dictates the direction of an API / Ontology because it is a major risk from a business standpoint. What if extra features are needed in the future and the governing organization is not willing or is unable to include them. What if they will start charging for it. What if the descriptions become inaccessible due to network outages at the governing organization, etc.

What if in the multitude of APIs / Ontologies a business simply does not find what it needs. In an ontology based web 3.0 we have to assume that thousands of such ontologies will exist. Even if they are all registered in a common library, even if we assume perfect harmony between such ontologies, absolute discipline in their usage and total backward compatibility, it would still be a titanic effort to find various concepts from various ontologies, putting them together, obeying the relations between them and their limitations, perhaps even extending some with custom particularities and then keeping

12 Changing slowly enough that the change is not uncomfortable to adapt across across the lifespan of human

track of them during the lifetime of the application. On the other hand is a lot less effort to just simply redefine whatever one needs and add it back to the common repository, a process that contributes to making the ontology landscape ever more fractured.

Suppose several businesses do agree to use parts from such foreign API / Ontology, for example foaf:Person, but they need to define custom ones due to business needs. Presumably they will implement custom versions for their Person, abc:Person, bcd:Person, etc:Person, and specify foaf:Person as ancestor class. In this situation the model is generating an inevitable fracturing of the computing reality with regards to the concept of the Person.

In spite of the fact that abc:Person is in fact conceptually exactly the same as bcd:Person, only with different sets of custom properties, the model does not regard them as such. They are part of two different ontologies, and their only connection is that they both inherit foaf:Person, which conceptually is in fact also exactly the same. It is difficult to imagine this for somebody used to work with inheritance, but conceptually, the abc:Person and bcd:Person with respect to foaf:Person are not derivative classes, the way Square and Circle are with respect to Geometric_Shape. But rather they are different variations of the same class. It is the same concept from reality that is being modeled but it is modeled so in an environment that spawns three separate concepts linked together with the relation of inheritance. The single concept Person from human reality is transformed into computing reality into three separate concepts fracturing the human reality behind the Person into three disparate or partially disparate realities in the computing realm. Applications that are built on the abc ontology cannot assume any other connection with the bcd:Person other than the fact that they have common ancestry in foaf. In ontology language we can assert that abc:Person is same as[13] bcd:Person, and vice versa, but then the owner of abc needs to continuously monitor the world of ontologies for any future ontology that may contain an equivalent version of the Person.

It is much more reasonable to conclude that there will be little discipline in using ontologies and a lot of subjectivity when it comes to defining them. As a result, instead of a harmonious landscape of interconnecting ontologies we are likely to observe ontologies that define weird terminology, that redefine terms without referencing others, missing intermediate pieces and duplicate terminologies all laid out in an impossibly complicated infrastructure of terms, types and connections which has nothing to do with the homogeneous reality that the model tries to represent.

The tendency can already be observed with today's handful of ontologies, which are defined by professional organizations that are not only specialized in the field of semantic web, they are actually creating it. For example both Friend of Friend and Dublin Core ontology define the term Image (foaf:Image and dcmitype:Image) and the two are not the same. An application built on Dublin Core will not be able to interpret[14] an image defined in Friend of Friend in spite of the fact that they are in fact conceptually identical, a phenomenon that is extremely detrimental to the entire concept of semantic web.

1.6.1 ◦ The Semantics Of Web 1.0, 2.0 & 3.0

The industry speaks about web 3.0 as the semantic web, where applications working on ontologies will revolutionize information access based on the fact that ontologies are capable to capture a lot more, computationally speaking, than simple document based data. Semantics however is a relative concept

13 For example “owl:sameAs” property that can be used to assert that two resource are in fact the about the same individual.

14 Unless of course the application implements both ontologies, but that is detrimental in a different way.

and only makes sense in conjunction with an agent, to which whatever is captured in the data is meaningful. The content of HTML documents and that of image of the current web are fully semantic to people. People can interpret and make sense of what is in the documents but this is not only because they can interpret the language, it is also because whatever is in the language, links to a complex array of facts and activities in people's reality. Ontologies may be capable to capture more complex aspects of human information while being interpretable by machine, but this, in itself, does not mean that they will contain any semantics from the standpoint of computers and the world wide web.

Web 1.0 and 2.0 are not regarded as semantic by the industry and HTML hardly qualifies as ontology in the eye of any participant. This is because we regard web semantism from human perspective: how can the web provide *us* knowledgeable information, by doing the analysis and the mining on its own.

But if we are to look at this from computer perspective and not the human one, it is in fact HTML that is the true ontology and not those defined by ontology definition frameworks. HTML is a language that defines a remarkably simple ontology: that of documents (which may be texts, images or binary) and the links between them. Because the semantics lie not in the representation, but in what the system can autonomously do with what is represented, this utterly simple landscape of images, texts, links between images and texts, gave birth to an entire web of data, with trillions of interconnections. This is because computers *can* actually *do* something with these concepts. Browsers, crawlers, search engines, web services all rely on the simple common concept of interconnected documents and the simple things that they do with these documents gave birth to the world wide web. The search result may not be very semantical to the end user, nevertheless this, the web 1.0 and 2.0 are truly semantic to computers.

Had there been many variations of images (such as the case with FOAF and Doublin Core), many variations of text documents and many variations of connections between them, the world wide web would never have been born. It is important to observe, that *different variations* do not refer to different representations: JPEG, PNG etc, but rather to something conceptually entirely different. This is so ridiculous that it is even hard to comprehend that different concepts named Image might exist, but from the computer's perspective this is exactly what it is: foaf:Image and dcmitype:Image are apples and wales, so to speak. The concept “Image” defined in different ontologies / APIs represent different realities to computers, that are incompatible in nature.

This is exactly what the many disparate ontologies of today represent: a web of proprietary representations, that are useless on a global scale because they are isolated and each requires specialized programs to do something with them. They will never be able to create a global network the way HTML did. The more they are, the worst it is from the prospect of communication because they create an ever more fractured reality *to computers*, and even if they are more equipped than HTML in modeling complex relations, computers will not be able to *do* anything with it outside the realm of the ontology that they implement and an isolated topology. In the best case this will result in many different semantic webs, rather than a single world wide web 3.0.

2 ◦ S.P.IN.D.L. (Patent Pending)

There has been extensive work in knowledge representation in computer science and the discipline exists for quite a while. Earliest works in KR have been done as early as 1959, much earlier than the birth of the Internet itself. Since then, enormous efforts have been put into compiling large ontologies (such as the millions of assertions in Cyc), to create upper ontologies, and knowledge representations systems, but clearly something must be missing, because in all this time, none of the systems managed to perform outside a niche environment akin to lab conditions and it does not look as if the industry is ready to adopt any of them as unique standard, which is the only way a web 3.0 can develop. On the contrary, it appears that more and more standards appear, as if the industry is still searching for that perfect ontology.

The present work is such an attempt. Having observed the inadequacies enumerated in Chapter 1, will try to solve the knowledge representation conundrum by eliminating them. S.P.IN.D.L., stands for Semantic-Perspective Information Definition Language, and it is an knowledge representation formalism that takes a very different approach on how standardization is done, by taking inspiration in how the human communication, and implicitly knowledge representation, *evolved* to accommodate the particularities of their reality.

2.1 ◦ Paradigms Of A Web 3.0

SPInDL is a knowledge representation language, not an ontology, in fact, SPInDL does not have ontologies, at least not in the sense of what ontologies are in today's KR systems.

2.1.1 ◦ Account For The Properties Of Information

Ontologies like FOAF define types / classes (example: 26), having properties and inheritance between between classes, very much like the structures of the classical programming languages, an approach that has been shown to be incompatible with the subjectivity and incompleteness property of information.

example: 26 FOAF Class

Class:	foaf:Person
Subclass Of:	foaf:Agent, foaf:Spatial Thing
Properties Include:	plan, surname, geekcode, pastProject, lastName, family_name, publications, currentProject, familyName, firstName, workInfoHomepage, myersBriggs, schoolHomepage, img, workplaceHomepage, knows

When it comes to knowledge representation in human language, a class / type does not exist as a prerequisite to store information about something. Types or classes or sets do come into existence as part of an analyses process, when a collection of objects are analyzed based on the commonalities they have, but otherwise they do not exist, because they would limit the capability to store information about the objects in discussion.

On the same token, human language does not have a limited amount of relations that can be used to describe objects, such as *properyOf*, or *subclassOf*. In reality, there may be an infinity of relations between elements of reality, attempting define a set of relations that constructs the structures of the reality will again limit the variations of knowledge that can be captured about the underlying reality.

The way these ontologies are created is the exact opposite of how knowledge is represented in human language. In human language structuring follows analyses, and it affects only the result of the analyses, it does not have any effect on the source of information. In these ontologies, structuring (the creation of classes and relations) precede the existence of the source of the information, in fact the source of the information is a materialization of the structure itself, consequently it will limit the reality (the source of information) to whatever it is captured in the structure. No additional a-posteriori information can exist about the source of the information within the realm of this ontology unless the ontology itself is updated.

By contrast, SPInDLE has no definition types, only two meta-types and two meta-relations, but the way these are used allow for an infinite number of relations, even future relations, without the introduction of new terminology or the need for modifying the language itself.

2.1.2 ◦ A Common Reality

As a direct consequence of the constructive approach of the ontologies', combined with the subjective way industry participants handle information, web 3.0 is now the scene of an extremely complex architecture of types and relations originating in different ontologies each trying to reconstruct human reality according to their own specific view. The concept namespace, which originates in the development of the XML standard (a structuring standard) is ported into the ontology world and it is being used to denote ownership of the ontology, basically it attributes ownership of the way reality is *constructed* to a certain organization who took upon itself the responsibility to define it. The problem stems from the word constructed (emphasized in the previous sentence), because there would be nothing wrong if the namespace were to define how organization xyz *perceives* reality, but this is not the case. To re-iterate the graveness of the problem that this approach creates let us consider an example from Physics.

Physics is the discipline that studies reality, different laws explain how different forces of nature behave. Some models are more complete then others and so, they are able to observe different more profound, aspects of reality. Such would be the example of Newton's law of gravity (newton:gravity) and Einstein's law of gravity[15] (einstein:gravity). The way ontologies are constructed now, gravity (a force of nature) would be a direct result of Newton's respectively Einstein's law of gravity and not the other way around, resulting in two different universes, with two different gravitational forces, one created by one of the models and one by the other. In this scenario there is no *one gravity (the force of nature)* <u>observed</u> by two different models, but two forces of nature, <u>created</u> by two different models. Some say that an upper ontology is needed which creates a new book of rules where we can specify relations between all the different models by saying something like *newton:gravity sameAs einstein:gravity,* but the concept itself is misguided. In fact newton:gravity and einstein:gravity are not the same, they never even were intended to be the same. They may refer the same concept from reality, but this is not what the “sameAs” relation states. The paradox rises from the fact that the semantic relation sameAs is being used in the wrong context. SameAs assumes that the models observe physical reality, but the model in which sameAs is used, the models in fact define physical reality. The set of facts based on ontologies in web 3.0 are the web 3 reality from the perspective of applications that operate with them, the same way as the totality of documents and their connections on the Internet is the reality for a search engine, crawler, web browser suit. It might be more difficult

15 These laws are part of a larger models, but this is not relevant at this point

to notice the similarity but foaf:Image and dcmitype:Image behave exactly in the same manner as newton:gravity and einstein:gravity and an Internet of foaf:Images would be an alien reality to a browser built for dcmitype:Images.

This upper ontology approach is not only used in a misinterpreted manner, but it is also impossible to track within a large enough pool of ontologies, because it has a backwards nature. Suppose we have *newton:gravity, einstein:gravity, abc:gravity, bcd:gravity, etc:gravity* as different interpretations of the natural law of gravity. Then, organization ABC, defining abc:gravity, must constantly monitor this upper ontology defined relations to make sure there exists a *sameAs* relation with all the laws of gravity that ever were and ever will be. One cannot rely on the transitivity of the *sameAs* relation because then one must also rely on the fact that any new organization XYZ contributing a new theory to the same law of physics will make sure to specify the *sameAs* rules accordingly. The question remains, up to how many variations of how many concepts can such a relation be tracked? Wouldn't it be more simple to standardize *gravity* firsts, as (*the law of nature*), supreme, existing outside of any definition and then relate this custom models of the law of gravity to *it, (newton:gravity describes Gravity), (einstein:gravity describes Gravity), etc.* Then all these models would implicitly have something in common, a common concept, something akin to a reality.

***

Knowledge represented on human language follows this common reality model. Everything that is within a language has a correspondent in reality of humans: objects, time, space, perceptions, emotions, activities, et cetera. Additionally the vast majority of reality is the same for all humans so interestingly enough, even though many languages formed far apart from one another, they are still largely compatible, given very few exceptional situations (see 1.3 Common Meaning In Human Communication).

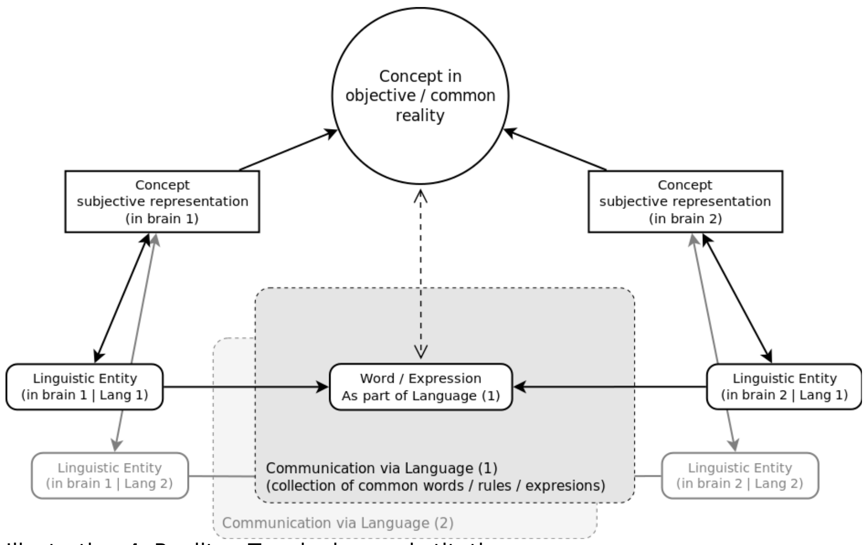

Concept in objective / common reality
Concept subjective representation (in brain 1)
Concept subjective representation (in brain 2)
Linguistic Entity (in brain 1 | Lang 1)
Word / Expression As part of Language (1)
Linguistic Entity (in brain 2 | Lang 1)
Linguistic Entity (in brain 1 | Lang 2)
Communication via Language (1) (collection of common words / rules / expresions)
Linguistic Entity (in brain 2 | Lang 2)
Communication via Language (2)

Illustration 4: Reality - Terminology substitution

Illustration 4 schematically depicting the world of human communication, emphasizes how every concept from the common reality, first passes through a subjective representation in the brain of individuals but in order for communication to be possible, ends up again as a unique objective term in another common realm, language. As opposed to computer reality, language does have the benefit of having that objective reality against which it is constantly synchronized and as such, there can be any number of languages, as long as they are synchronized against a common reality, translation between them is possible.

In the computer reality, which lacks the fundamental common reality of humans, on account that computers are unable to perceive it, we could substitute our reality with our language terminology. It would no be a complete substitution, there would be many concepts that are contextually referred to in language, but it would be none the less be a common reality: so instead of a common reality, computers would use a common terminology. Whilst in the human world the common concept goes to the word in the brain which is connected to the concept in the brain which is an exact correspondent of the actual concept that exists in the reality, Illustration 4, in the computer world, the convergence will end at the “word”.

The benefit of this approach is that there is no need to construct or impose a standard. Language already is a de facto standard present in every single culture, business, or any other group of people for that matter.

This would be a fundamental change with regards to how information is looked at, because the perspective changes from structure to concepts. The priority would not be any more how the data is structured, what properties a particular ontology captures, such as the case of *foaf:Person*, but rather what the data represents, in this case Person as defined in the language dictionary. The association carries no structural information, therefore, there really is no need for defining particular kinds of persons. As such, *foaf:Person* and *abc:Person* would not be defining the concept, *Person*, that would already be defined in the language dictionary, but rather they would refer to the common concept Person and would only elaborate on the particularities of their *view*.

From a communication perspective this would have an enormous impact. There is no need for any elaborate repository system (UDDI or upper ontology), because when communication occurs the term would automatically carry the concept with it. Bob's and Alice's applications that both handle data about the concept of *Person* under their own particular structure will be able to exchange information without ever being synchronized via an API, upper ontology or similar repositories because they refer a common reality, in which *Person* has a very specific meaning.

The very fact that the word itself carries the reference to the concept in the common reality is already some degree of semantics. Imagine that Alice is browsing through a set of data Bob has shared with her, and within it there is a list of *Persons*, which Bob's application is designed to manipulate, but Alice's computer is not. The fact that Alice's computer can rely on the fact that Bob's *Person*, is what the dictionary says it to be, and the fact that Alice is fully aware of all the words in the dictionary (even if most of them don't have any implementation associated), it can safely receive the information from Bob's computer and present it to Alice. This would be totally impossible to do in the classical way because a type in an API is not constrained to a concept of any kind, is a self contained entity. To give an example, *abc:Person* does not necessarily have to capture information about *Persons*, it could just as well be *Oranges,* if the organization behind the type would not care to give their types suggestive names.

2.1.3 ◦ Simplicity & Familiarity

The other trend that is enormously detrimental to web 3.0 is the terminology invention, briefly touched at in chapter (1.5.5). Every single API and ontology that currently exists is dominated by this trend. Types, properties and relations like the one in example: 26, define their structure with compound expressions that although have resemblance with dictionary terms they preserve only partial connection with these: *pastProject, lastName, family_name, publications, currentProject, familyName, firstName, workInfoHomepage, myersBriggs, schoolHomepage, img, workplaceHomepage, etc.* The same can be observed in BFO upper ontology *(SnapEntity, TimeInstance, TemporalIndex, Ontology, TimeRegion, SpatialLocation, TemporalLocation)*, UMBEL *(umbel:isAbout, umbel:correspondsTo)*, SKOS *(skos:prefLabel, skos:altLabel, skos:hiddenLabel, skos:definition)*, CycL *(#$relationAllExists #$biologicalMother #$ChordataPhylum #$FemaleAnimal)* and all the others.

It is important to observe that these terms are resemblant of words found in the language dictionary and so they appear to refer concepts from our reality. This resemblance however, is highly misleading because these expressions are in fact completely new terms within their own world, with definitions describing their role within the ontology definition that may or my not look or be similar to the one in the language dictionary term that looks similar. They couldn't even be any other way, because often times these are root concepts within the ontology having no reference what so ever to other concepts within the ontology that could have some explanatory nature within the ontology itself.

example: 27, FOAF Person

```
<foaf:Person>
 <foaf:name>John Doe</foaf:name>
 <foaf:workplaceHomepage rdf:resource="http://www.john-does-homepage.com/"/>
</foaf:Person>
```

workplaceHomepage in FOAF is a completely new concept that has no other explanation that the human readable one in the specification. There is not such thing in foaf as *Workplace* or *Homepage*. Additionally the use of words is not regulated either. There is nothing in any of these standards that dictates that a word which is identical to the one in the language dictionary must have exactly the same meaning as in the dictionary, which is in fact the real world meaning of that word.

The new, invented word, is a root concept within the ontology and anybody who wants or needs to use it needs to understand it first by reading the specifications. *WorkplaceHomepage* is one that is quite suggestive, even obvious, having a very good choice of words, but they are many in different ontologies that are much harder to understand: *isAbout, correspondsTo*, *#$relationAllExists, etc.* that have complex ontology level meanings (the way the intersection sign ∩ has a special role in set theory). These are really impossible to understand unless the specification is studied and understood.

Collectively these ontology definition languages and ontologies generate tens of thousands of such new expression. It is unreasonable to expect that any system could work with such an avalanche of terms especially when these are not linked to an existing reality but rather generate one of their own. A system is needed that relies extensively on the real world meaning of the words, one that is based in the common reality of humans, only than will it be possible for it to become popular within large groups of people and create the dynamics that will ultimately generate the web 3.0.

2.1.4 ◦ Openness

The human reality is huge, enormous, it has a vast amount of objects and possibly an infinite amount of possible relations between those objects. Any system that is to cope with such a reality cannot rely on predefined types and predefined relations, simply because it is impossible to account for all possibilities when these possibilities have no limit or are possibly even unknown at the time of definition (like future concepts / relations).

If we take a look at human language, we can observe that *it* evolves in tandem with human needs, human reality in a perfectly smooth, seamless manner.

2.2 ◦ The Architecture Of S.P.In.D.L

The SPInDL knowledge representation language was conceived as a result of the observation of deficiencies of current KR languages / ontologies enumerated in the previous chapters, and consequently it delivers an alternative architecture that works around these deficiencies.

2.2.1 ◦ Binding With Language

The first and probably most radical of all paradigms that SPInDL assumes, is the existence of the *Common Reality*. SPInDL does not create a reality, but rather it describes one. As a consequence, any object or fact that is being described with SPInDL it is assumed to predate the description and everything that is not described it is not assumed to be inexistent only not described. Because computers do not have the possibility to observe the common reality we humans share, the SPInDL common reality is considered to be, what has been identified in the paper as a direct binging of the human reality, the English Dictionary.

example: 28, Definition of the Person in the dictionary

Person
1. An individual human being

At this level, the level of reality, there are no namespaces allowed and no fabrication of new words is allowed. Expressions that are multi word in English language, such as “first name” are allowed but only in the very form they appear in language “first name”, not “firstName” or any other combination that could suggest something different than the human expressions. To compensate for a lack or clarity that exists virtually in every language, homonyms, words that have multiple meanings, the first meaning will be assumed that the dictionary defines.

example: 29, Meaning of Person in the dictionary that may create confusion

Person
...
2. An individual of specified character: a person of importance

The rest of the meanings will be substituted with appropriate expressions. In example: 29, such an expression could be “Public Figure”. The way these expressions are used however it is not defined. It will be at the latitude of the community (groups that use the meaning) to negotiate the appropriate expression when such moment occurs (see 2.12 Openness, Community driven model).

2.3 ◦ Cases, The Knowledge Base

Given the direct language binding, the ontology of SPInDL consists in fact of all the words and expressions defined in the English Language Dictionary having the very same meaning detailed there with the assumption that people understand and relate to these words in order to connect them with everyday concepts that they have in their reality. Additionally to these concepts that exists in the dictionary the ontology of SPInDL will contain the meta types, Concepts & Specifics, and the two meta relations, Divergences & Correlations.

The knowledge, the actual facts that are captured about the concepts that exist in this outside reality, specifics and the connections between them and other subjects, is called a case or a knowledge base.

definition: 1, Case

A case is a subjective set of facts, as captured by a particular application at a given point in time.

Implicitly every case will contain the SPInDL ontology, hence every concept from the dictionary and so this will serve as a common system of reference to all applications that are based on SPInDL. The rest of the information captured by a system is considered to be particular to that system.

2.4 ◦ Concepts & Specifics

From an existential point of view the most general differentiation that we can create with regards to the elements present in our reality is whether they are touchable actual material things or on the contrary, things that cannot be touched but nonetheless influence our existence. Based on this rule we can categorize these elements as *Concepts (immaterial, untouchable)* elements and as *Specifics (material, touchable)* or in other words concrete manifestations of the concepts.

definition: 2, Concept

Concepts are terminologies for stand alone facts, objects, phenomenons, etc, from reality in a way that humans consider them as distinct from all the other facts, objects, etc, from reality, they assign them individual identification (name) and definition.

The notation for the *concept* will be the UTF8 representation of the word in the dictionary that represent the concept in their *base form: Person, Elephant, Create, Be, Run, Time, Happiness, Future, Past, etc.*

definition: 3, Specifics

Specifics are materializations of concepts. Particular instances of a Concept.

The notation for a *specific* will be that of the concept of which the specific is a materialization of, followed by a unique reference in parenthesis: *Concept(x)*. In this case, the x in the parenthesis must be a *specific* that can uniquely identify the given specific within the case.

Not every concept can have instances. For example “one” (1) is a instance of the Number concept, A specific Person (myself for instance) is a materialization of the Person concept, a specific rock of the Rock concept and so on. These concepts can be called suggestively “Material Concepts”. On the other hand, concepts like “tall”, “sad”, “fast”, “run”, “dig”, “future”, “past” and many more have no materialization, they are “Abstract Concepts”.

SPInDL does not make a differentiation between abstract and material concepts. Within a particular case, all concepts are considered to be *abstract* until they have a materialization, at which point they become *material*. It is not however regulated whether a concept can or cannot have a materialization. Human reality is very complex and even seemingly immaterial concepts can have materializations on rare occasions. Take “time” for instance: it is a highly intangible concept which although present in our everyday life it is mostly handled as an abstract concept. Even so, we do occasional consider specific periods of time and refer to them as “the time of something”, like (“the time of Picasso”). This is not an isolated case, the infinitely complex human reality has many such example and as such it is better for any knowledge representation system to remain open. It will remain at the latitude of the program whether it will allow materialization of a concept or not.

***

A small note is warranted on the concept of specifics within a case. In everyday life when we have specifics in discussion we constantly adjust context in order to easily avoid confusion about the objects that we discuss, therefore we are tempted to talk about specifics using a characteristic (property) of it. This is especially valid when we consider universally unique identification properties, such as the case of things that are regularly stored as data (people's personal information, car information, etc). But for most objects in reality there is no such thing as an absolute identification property. An object is usually identified (in a communication) with several traits: “The pine tree that grows in front of John's house”, that together are sufficiently unique within the give context. Nevertheless, inside our brain we have a very specific, absolute reference about all objects that matter to us. When we think to ourselves, inside our heads, there is no doubt about the particular instance we refer to. We need no such description to identify which pine tree we are talking about. References within the cases should take this aspect into consideration and use internal pointers for all specifics which are separate from all the properties, facts that are recorded about the given specific.

2.5 ◦ Divergences & Correlations

The Achilles hill of every modeling language are the relations. In human reality there are countless relations that can be set up between concepts and specifics (instances of concepts) and creating an a priori list of relations to encompass them all is simply impossible. The attempt to do so resulted in the biggest and most complex part of the terminology invention in current ontology languages: “parentOf”, “memberOf”, “derivesFrom”, “superclassOf”, “propertyOf”, and so on.

example: 30

```
firstName › porpetyOf › Person
```

The result is both highly complex and limiting in terms of how many connections can be captured, resulting in complex yet insufficient, impossible to customize data structures that are locked into ontologies and depend heavily on the definitions the ontologies themselves define. Any new kind of relation that may arise in the future, needs modification of the ontology and introduction of a new definition in order to capture the relation.

Another problem is that the relations are static in nature. The relations cannot capture in their definition specifics that are part of a *Case*. *Person(x)* is part of a particular knowledge base, a *Case*, which is a manifestation of an ontology. The type Person as part of the ontology and all other types and relations must preexist any manifestation of Person, in this case Person(x). As such, Person(x) cannot be part of the definition chain of a relation.

It is important to mention that the concept of relation, just like the concept of type is in fact a result of classification, structuring, the process of creation of types. While this may be important from an analytical point of view, it has been shown int the previous chapters that from the information capturing point of view, this is detrimental. For this reason, SPInDL will avoid the use of relations altogether at the language definition level in order to allow for any possible connection between two subjects.

***

SPInDL introduces only two meta relation as part of the language (the *divergence* and the *correlation*). It is appropriate to call them meta relations because they are very low level containing no case specific semantics. Together with the dictionary and the meta types “Concepts” and “Specifics”, these two meta relations form the entire SPInDL ontology.

definition: 4, The divergence

The divergence, p(S), denoted graphically by (›) is the coupling which defines the particular angle through which the connection from a subject's perspective is connected with another subject's perspective.

The notation p(S) means that the *subject S* is seen through the perspective of *subject p*.

When we try to describe a situation, a case, in reality we analyze subjects (specifics and concepts) from within the context of other specifics or concepts. Everything in our world is a concept, something from the reality around us, a feeling, a state, a time, the self, etc. and the observed fact is a combination of a subset of these concepts.

example: 31, example of a fact

John is fast.

In example: 31 we state that a specific person is fast. When we use the word *John* we refer to a very specific Person. The context we use this word in implies that everybody present in the conversation (any consumer of information) has a special memory location where the specific person John, or more precisely the reference to him, resides. As such, John (which in this case is the commonly used reference to this particular memory location) is a *specific* from reality. The other subject used in the sentence is the concept of *being*. Being, referred to by the word *is,* is an abstract concept which has a very specific meaning that everybody in the audience understands. *Fast*, is yet another similar concept that implicitly brings with it the hidden concept (not appearing as a stand alone word in the sentence but rather just implied by the word fast), *speed*.

The object[16] *of the analyses is John and the trait observed that characterizes John is the speed that describes its being.*

example: 32, Perspective John is observed from

John ‹ being ‹ speed

example: 32 illustrates how John, the specific, is observed from the perspective of the *being concept, speed(being(John))*, which in turn is observed from the perspective of the *concept of speed* and thus creating a particular perspective from which John is viewed in the context of the stated fact.

example: 33, Correlation of John with the concept of fast

John ‹ being ‹ speed ⇢ fast

The conclusion of the analyses, the fact, states that viewed from this particular perspective John is connected to the concept of *Fast,* which is another well determined concept in people's view. The relative aspect of the fastness at this point is not necessarily relevant, because it is not captured in the case, but rather it is considered as part of the greater context in which the case takes place.

example: 34, Correlation of John with the concept of fast

John ‹ being ‹ speed ⇢ relative › fast

If we were to be more specific about the fact that there is a relative aspect to John being fast, that would mean that we are looking at the concept of *fast*, from a particular perspective, that of relativeness, rather than the absolute view as depicted in the example: 33. So in this case “John is relatively fast.”, example: 34.

16 The sentence is not analyzed based on the parts of the sentence (subject, predicate, etc.) but rather from an informational perspective.

definition: 5, The Correlation

The correlation, denoted (⇢), is the coupling which connects two particular perspectives of two subjects.

In the causal representations of the facts, the correlations will be replaced by a simple coma, since every fact contains a single correlation and the positions of the perspectives in the fact will give the directionality of the correlation.

Both, the *divergence* and the *correlation* are directed meta relations, because both have a sens of directionality, causality. The correlation in the examples above shows, that although the connection affects both, it is *John* that is connected to the concept of *Fast* and not vice versa. Similarly, the divergence meta relation emphasizes the perspective from which a subject is being analyzed from. It can refer an absolute subject or a perspective of it, generating an even more particular point of view.

definition: 6, The Perspective

The Concept or Specific that sits at the starting point of the divergence shall be termed, perspective.

The actual relation, if we can talk about it in the same way as in the case of ontologies, would be the complete path, from *John* to *Fast,* in example: 34, and contains elements from both the SPInDL ontology (language elements plus the dictionary) and the particular case in discussion. However, these complete paths resemble more facts than particular types of relations.

definition: 7, Fact

The full path from one subject to another via the divergences and perspectives, and connected by the correlation is termed as a *fact*, denoted $F(p_1, p_2)$.

Every fact will contain a single *correlation* and any number of *divergences* necessary to generate the particular perspective the fact needs to state, $F(p_n(p_{n-1}(...(p_1(p_0(S_1))))), q_m(q_{m-1}(...(q_1(q_0(S_2))))))$. As such, the fact is not part of the ontology but rather the case and the way these facts are formed they allow for the development of any number of facts between the subjects of the case and within the bigger context of the ontology.

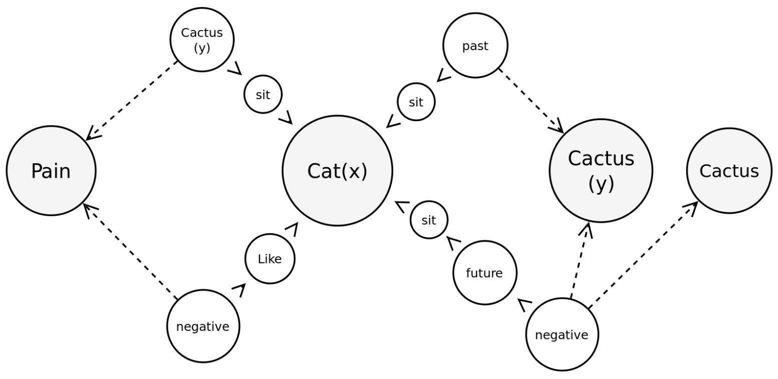


Illustration 5: Cat & Cactus Case

Illustration 5 depicts a more complex case, in which a cat set on a cactus in the past, felt pain and will not sit on that cactus or any other cactus every again.

An important aspect of the knowledge capturing process is observable here: in SPInDL, the role of perspective can be not only played by a concept, but it can also be a taken by a specific. The fact that the *Cat* is correlated with *Pain* through the perspective of *Cactus(y)*, in the classic ontology model would have to be provided by a parametrized relation, the definition of which would contain a great deal of complex assumptions. This would crowed the ontology with a lot of definitions that are difficult to track. SPInDL on the other hand solves the communication of the fact with a very simple construction.

In such cases when the perspective is itself part of a chain in another perspective, specifics would meet this criteria too, a notation artifice can be employed to avoid the awkwardness and confusion of opening and closing parenthesis:

> $p_n(p_{n-1}(\ldots(p_{1,3}(p_{1,2}(p_{1,1}(p_1)) \mid p_0(S_1)))))$, *instead of*
> $p_n(p_{n-1}(\ldots(p_{1,3}(p_{1,2}(p_{1,1}(p_1)))(p_0(S_1)))))$

In essence, the ")(" parenthesis combination is eliminated and it is replaced by a vertical bar resulting in a perspective that has a continuation but with a single imbrication chain.

***

It is important to mention that information does not have an absolute character. The same information can be conveyed in many other forms, using different concepts and it is ultimately our ability to find pattern in the use of these context that gives sense to the information captured.

2.6 ◦ Representational Concepts (Primitives)

Every system that is meant to hold information needs at the base some means by which it can represent it, within the designated medium. In human reality we use spoken words, written words, sounds recordings, pictures, movies, etc. In computer science, all data is encoded in binary or series of bytes (on a slightly upper level). If we look at it from an even higher level of abstraction these byte arrays can contain data stored in there using an algorithm. The algorithm is them used to restore the data and allow information to be extracted. The extraction of information will be performed by the entity capable to interpret the data. So from an data storing point of view, ultimately all information will end up in bytes, but from a conceptual point of view it is more beneficial to look at the data at the level where it is intelligible to the information consumer. Take for example a PNG (portable network graphics) encoded photo. While it too will end up as a sequence of bytes in a file, to a person this is really not relevant. The person will use an application to store it there and to restore it into its photograph form to be able to observe it on the screen. Any other intermediate form is irrelevant. The photo cannot be decomposed into smaller parts while maintaining information integrity.

example: 35, Primitives

> *These self contained material concepts, the specifics of which carry with them atomic representations of information uniquely discernible from all other specifics are called Primitives.*

Earlier in this document we already used a different representation system, the UTF8 (2.4 Concepts & Specifics), which is a system of encoding written words for many languages. This is necessary because concepts need to be referred to and it is possible because every concept is in fact a *Specific Concept*[17].

So *Happiness* can be represented as *Concept(happiness)*, where the word, the UTF8 character sequence *[h, a, p, p, i, n, e, s, s]*, uniquely identifies the *Happiness* concept. Since every word in the dictionary is a concept and the dictionary is in fact the predefined ontology, the simplification convention can be made to eliminate the *Concept(happiness)* notation for concepts and use it as simply *Happiness*. For other specifics that have character representations the notation is necessary, *Password(abc123)*, because in this case the specific password is not part of the ontology, it is part of a particular case in which we recognize the existence of *the specific password* identified as *abc123*.

We can see that in order to be able to represent any kind of information, the bottom most subject will always have to be a representational primitive.

example: 36, The place of primitives in every factored

Cat(ref_1) ‹ like ‹ negative ⇢ Pain
Cat(ref_1) ⇢ Name(UTF8(kitty))
Cat(ref_1) ‹ eat ‹ past ⇢ Cactus(ref_2)
Cat(ref_1) ‹ sit ‹ past ⇢ Cactus(ref_2)

Where Pain for instance is a simplification of the Concept(UTF8(Pain)), and so on.

SPInDL only enforces the existence of the following primitives:

- UTF8: Text, everything that in human language is represented as written text.
- Numeric: Real numbers, everything that humans represent as numbers
- Time point: A formatted text representing a date and a time
- Binary: everything that is primitive but cannot be captured as one of the above mentioned
- Reference: a primitive that is used as a placeholder for any specific that is not a primitive

Other primitive representations can exist, like text patterns, PNG, JPG, WAV, etc. These primitives however are not regulated, implementors are free to use any form of encoding they seem fit to create their cases: for example *Meter(123)* vs. *Meter(one hundred and twenty three)* as long the system can represent them. The only thing enforced by SPInDL with regards to new primitives is that if an international standard, or a sufficiently widespread term, does exist for the primitive (such as the case the mentioned PNG, JPG) the primitive should bear this standard name and conversely, if it does bear the standard name PNG, then it has to conform to the nominated standard.

2.7 ◦ Properties Of SPInDL Constructs

SPInDL represents information as facts: causal relations between concepts and instances, but it replaces the custom relations with correlated perspectives, dynamic constructs, concepts and specifics thus allowing for uncapped combination of correlations between subjects.

17 A specific of the *Concept* concept.

2.7.1 ◦ The Vanishing Of Detail

Perspectives are obtained by diverging a subject using a concept or a specific thus creating a more detailed perspective to this subject or some perspective of it. In this latter case, the diverged perspective becomes subject to the newly created perspective. Conceptually speaking, each perspective is in fact a particular view of the subject it diverges from, or looking at it inversely each subject is a less detailed view of all perspectives diverging from it.

We will note this relation where the details are vanishing cascadingly with the (⇒) symbol:

> $p_n(p_{n-1}(...(p_1(p_0(S)))) \Rightarrow p_{n-1}(...(p_1(p_0(S)))$*, and we interpret this: if subject S is being observed from perspective* p_n*, then subject S is also being observed from the perspective* p_{n-1}*, where* p_n *is a perspective of the subject viewed for the perspective of* p_{n-1}*.*

This property of the perspectives also extends onto the facts, a big part of the construction of which, are the perspectives. To exemplify, let us consider the case described in Illustration 5.

example: 37, Correlations from Illustration 5

1. Cat(x) ‹ sit ‹ past ⇢ Cactus(y)
2. Cat(x) ‹ sit ‹ future ‹ negative ⇢ Cactus(y)

In the case described in Illustration 5 we can see that there are two correlations between cat(x) and cactus(y), listed in example: 37. Both correlations connect cat(x) from the perspective of sitting, one from the further diverged perspective of past one from the future.

example: 38, More generic correlations from Illustration 5

1. Cat(x) ‹ [redacted] ‹ [redacted] ⇢ Cactus(y)
2. Cat(x) ‹ [redacted] ‹ [redacted] ‹ [redacted] ⇢ Cactus(y)

example: 39, Process of correlations diverging, from Illustration 5

1. Cat(x) ‹ sit ‹ [redacted] ⇢ Cactus(y)
2. Cat(x) ‹ sit ‹ [redacted] ‹ [redacted] ⇢ Cactus(y)

example: 40, Process of correlations diverging even more, from Illustration 5

1. Cat(x) ‹ sit ‹ past ⇢ Cactus(y)
2. Cat(x) ‹ sit ‹ future ‹ [redacted] ⇢ Cactus(y)

Conceptually speaking the divergence of perspectives means that we are describing a case in ever more detail by capturing these details by applying different perspectives. If we erase the perspectives, like in example: 38, all that remains to be known is that there exists a correlation between cat(x) and cactus(y). The two correlations, 1 and 2, become identical. Similarly, if we erase the divergence down from the *sit* concept, example: 39, the two correlations continue to remain identical but at this stage we know more details about the correlation, more precisely the fact, that cat(x) and cactus(y) are correlated via the perspective of the *sitting* activity. It is not known whether this happened in the past, or will happen in the future, or it could be an entirely different perspective, the fact of the matter is that details are missing from the information. Digging even deeper, example: 40, we now know that cat is connected through the perspective of sitting, both in the past and in the future. At this point the two facts are not identical any more, because the perspectives diverge sufficiently that they convey different information. As such, we can enunciate the property of facts that the existence of a particular fact implies the existence of the more generic fact from the same line of perspective:

If the fact $F(p_n(p_{n-1}(\ldots(p_1(p_0(S_1))))), q_m(q_{m-1}(\ldots(q_1(q_0(S_2))))))$ exists than the fact $F^1(p_n(p_{n-1}(\ldots(p_1(p_0(S_1))))), q_{m-1}(\ldots(q_1(q_0(S_2)))))$ also exists, and so does $F^2(p_{n-1}(\ldots(p_1(p_0(S_1)))), q_{m-1}(\ldots(q_1(q_0(S_2)))))$ and every more generic fact down to completely generic fact $F^{m,n}(S_1, S_2)$.

A very important aspect of SPInDL comes into view at this point. If we take a look back to example: 37 we can observe that the last perspective in the second fact is a *negation*, as in, *the cat(x) will never again sit on cactus(y)*. This may seem like a conflict because then the negative fact implies the existence of the fact that lacks the negative in its chain of perspectives, example: 40 second fact. This however is not the case. SPInDL does not handle logical operations. It is a description language and correlations do not carry values of truth within them, they only describes the correlations observed / known. SPInDL operates on the open world assumption, so a given correlation, does not mean that every fact that starts from a given perspective exists, but rather it states that there exist at least one fact from that given perspective, but that fact can have any number of unknown details. In the above correlation, the negation is a detail.

example: 41, Conflicting correlations

1. Cat(x) ‹ sit ‹ past ⇢ Cactus(y)
2. Cat(x) ‹ sit ‹ future ‹ negative ⇢ Cactus(y)
3. Cat(x) ‹ sit ‹ future ‹ positive ⇢ Cactus(y)

The case could very well add another fact, example: 41 third correlation, which from a logical standpoint is a contradiction of the second correlation. This would be perfectly legal, as this logical aspect is not handled. SPInDL operates just like the human language. It allows for description of facts, but the language itself does not regulate the consistency of what is being represented. Sentences can contain nonsensical information, paradoxes, contradictions, or valid information. It is the duty of an analyses tool, built upon rules like *Logic*, that can analyze and decide whether the information represented is valid or not.

2.7.2 ◦ The Specifics And The Primitives

In the context of *detail vanishing*, the first name John, *First Name(John))* or more precisely, *First Name(UTF8(John)))*, exemplifies yet another characteristics of SPInDL:

The concept of which a specific is a specific of, is in turn a particular perspective of the root subject primitive.

example: 42, Primitive viewed as specifics of different concepts

First Name(UTF8(Jasmine))
Flower(UTF8(Jasmine))

The concept *First Name*, is a particular perspective of the primitive text *UTF8(Jasmine)*. This particular primitive is a single one but it can be looked at from the perspective of both *First Name* or *Flower*, example: 42.

The same can be said for non primitive specifics, those that are represented by a reference placeholder.

example: 43, Reference viewed as specifics of different concepts

1. Person(Ref_x)
2. Individual(Ref_x)
3. Human Being(Ref_x)
4. Woman(Ref_x)
5. Female(Ref_x)
6. Mother(Ref_x)
7. Grandmother(Ref_x)
8. …
9. *Plant(Ref_x)*

A *Person* is a very complex concept. A specific person cannot be universally identified in a unique manner by one of its traits therefore a system using SPInDL KR will use an internal reference as a placeholder, and all connections to the specific will be handled through that placeholder. In SPInDL, the concept is a particular perspective, as such, the given reference can be viewed from any number of concepts. In example: 43, the *specific* that we know of and refer to by Ref_x, is successively considered as a specific of numerous concepts.

Again, it is important to mention that SPInDL does not impose validation on the correctness of the information represented. Entry 9 in example: 43 is highly unlikely to be a consistent way to reference Ref_x, but it nonetheless permitted by the language.

2.8 ◦ Basic Knowledge Operations

2.8.1 ◦ Perspective Equivalences

In human language there are many words that denote not a single concepts but rather the combination of various concepts. Additionally when sentences are formed, different terminations or forms of a word are used to express composed concepts, like possession, time, place, etc., combinations that in SPInDL are expressed by perspectives. Because these word combinations or derivations will exist in the language dictionary and hence people will be tempted to use them, we introduce the perspective equivalence operation, which will allow such altered words or concepts to take on more complex perspective but still remain compatible with the ontology:

definition: 8, Perspective equivalence

The equivalence, denoted with the symbol "=" expresses that the perspectives at the ends of the symbol are equivalent from information conveying perspective and hence are interchangeable in any construction $p_1 = p_2$, or in a more complex case $p_x = p_n(p_{n-1}(\ldots(p_1(p_0))))$.

example: 44, A case containing perspective equivalence

1. sat = sit ‹ past
2. Cat(x) ‹ sit ‹ past ⇢ Cactus(y)
3. Cat(x) ‹ sat ⇢ Cactus(y)

In example: 44 because of the definition *sat = past(sit)*, we no longer need to express the perspective in a cascading manner *Past(Sit(Cat(x)))*, point 2, rather we can use the simplified form *Sat(Cat(x))*, like in point 3. Due to the equivalence, however, the perspective chain is preserved and any analytics software can recognize the presence of concepts in the chain.

2.8.2 ◦ The Concept Definition

A special case of perspective equivalence is the concept definition. In natural language, we often observe concepts defined with the aid of other concepts.

example: 45, Concept definitions

1. Midnight = Hour(24)
2. Fortnight = Count(2 | Week)
3. Adult = Full(Growth)

Not all concepts have definitions in other concepts so the ability to be expressed by other concepts is not required by SPInDL. It is nevertheless useful because many words in the human language are specifically designed to express specific aspects of certain concepts and such expressions are likely to be used by certain groups of people. Being able to express the equivalence in such cases where this exists reduces the volume of concepts expressed and covers the variations of the expressions for concepts making knowledge representation more portable.

2.8.3 ◦ The Concept Implication

Similarly, natural language contains a lot of terms that, although are not exact equivalent of other concepts, imply the existence of certain concepts, example: 46.

example: 46, Concept implications

1. Woman → Feminine(Gender(Human))
2. Woman → Adult
3. Interstate → State(Connect(Road))

For these cases, it is useful to have an operation to specify such implications. Denoted with an arrow (→), the implication means that the concept on the left of the sign, determines the concept on the right of the sign, a relation that is not reciprocal, however. As shown in example: 46 (1 & 2), the *Woman* concept implies that we are talking about a *human being* which from a *gender* perspective is *feminine*, however the reverse does not necessarily determine the *Woman c*oncept. Other concepts exist that imply *Feminine(Gender(Human))* namely, *Gal*, *Girl*, *Lady*, etc, each of them having additional implications.

2.9 ◦ Patterns

One of the most powerful tools in our knowledge manipulation arsenal are patterns: discernible regularities within our world. Knowledge represented with SPInDL is highly natural and excellent candidate for manipulation using patterns.

To evidential the affinity for patterns let us consider again the generic form of the SPInDL fact, the atom of knowledge, so to speak:

$$F(r_n(r_{n-1}(\ldots(r_1(r_0(S_1))))),\ q_m(q_{m-1}(\ldots(q_1(q_0(S_2)))))),$$

where r & q are known perspectives, F is the fact and S are subjects.

Owing to the properties of perspectives, 2.7 Properties of SPInDL Constructs, it can be observed that any perspective, is in fact a pattern for a family of more specific perspectives, and any fact is a pattern of a family of more specific facts, whether these facts and perspectives exist or not, within the knowledge base (case):

$$F^{?}(\,?(r_n(r_{n-1}(\ldots(r_1(r_0(S_1)))))),\ ?(q_m(q_{m-1}(\ldots(q_1(q_0(S_2)))))))$$

where ? are placeholders for various perspectives of the r & q divergences.

That being said, the beginning of the perspective is not the only place to have a wild card can be placed, and depending on where it is being placed it changes the conceptual meaning of the operation that is being performed.

2.9.1 ◦ The Faceting

The simplest form, and the one that comes out directly from the properties of SPInDL, 2.7, is the *faceting*.

$$?(q_m(q_{m-1}(\ldots(q_1(q_0(S_2))))))$$

The meaning of faceting, placing the wild card in front of the divergence, or more precisely, replacing ever more perspectives in the left of a divergence and by this creating an ever more generic view of a subject, is to identify the various facets a diverged subject is looked at from. Let us consider an example:

example: 47, The faceting pattern

1. Feminine(Gender(Human(ref)))
2. ?(Gender(Human(ref)))
3. ?(Human(ref))
4. ?(ref)
5. ...
6. Young(Age(Human(ref)))

In example: 47 we consider ever more generic perspectives of the specific *Human* subject stored at *ref* all the way to where we just look at a raw reference *ref*, from no perspective at all.

If we look at this from an information extraction angle, a pattern match on *?(Human(ref))*, will yield a collection of divergences of *Human(ref)*, more specifically all the various perspectives the specific *Human(ref)* divergence is looked at from. In the case presented, *Young(Age(...)) and Feminine(Gender(...))*.

2.9.2 ◦ The Generalization

The opposite way to use the pattern is to place the wild card onto the other end of the perspective chain, or in the causal representation inside the perspective in order to obtain an ever more generic subject.

$$q_m(q_{m-1}(\ldots(q_1(q_0(?)))),\ q_m(q_{m-1}(\ldots(q_1(?)))$$

This shifts the importance placed on subject onto the perspective. As we generalize, we no longer look at the subject as a specific instance, or a concept, but we rather look at the case, the knowledge base, from the from the standpoint of the perspectives that we employ:

example: 48, The generalization pattern

7. Feminine(Gender(Human(ref)))
8. Feminine(Gender(Human(?)))
9. Feminine(Gender(?)))
10. Feminine(?)
11. …
12. Young(Age(Human(ref)))
13. Young(Age(Human(?)))
14. …
15. Young(?)
16. Feminine(Gender(Horse(ref_1)))
17. …

example: 48 is a reverse of example: 47. We can see how we gradually loose site of the *Specific Human* we started from and move upwards onto the *Human* as a concept, then we loose that too and look at the *Gender* as a perspective of *various generic things*, and so on.

If we were to extract information on the *Feminine(Gender(?))* pattern in our case, we would be obtaining a collection of all the subjects in our knowledge base that are looked at from the perspective of *Feminine(Gender(?))*, namely: *Feminine(Gender(Human(ref)))* and *Feminine(Gender(Horse(ref_1)))*.

2.9.3 ∘ The Observation Pattern

There is a notable hybrid pattern that emerges from the combination of the Generalization and Faceting patterns,

$$?(q_m(q_{m-1}(...(q_1(?)))))$$

the answer of which is a collection of divergences that are being observed on a family of subjects. In a derivative of example: 48, we can inquire with the pattern, *?(Human(?))*, which will tell us both, the series of perspectives that are observed on any subject that is *Human*, namely *Gender*, *Age*, and their divergences *Feminine(Gender)* and *Young(Age)* respectively, but also the family of subjects, in our case references, on which we observe these perspectives, namely which references are looked at from the *Human* perspective.

The observation pattern can be considered a particular case of either the faceting or generalization pattern, as it yields an intermediary result which can be further analyzed by a more restrictive pattern, from either types.

***

The enumerated patterns are all simple, perspective patterns, because they only focus on the way perspectives are used on subjects. Patterns however can also be used with facts, by replacing the two divergences in the fact with any of the previous perspective patterns, *faceting*, *generalization* or *observation*.

definition: 9, The Property

The resulting pattern is matching facts not perspectives, a construction that is termed a property.

Based on the type of the pattern used in place of each divergence, we can categorize the properties as follows:

2.9.4 ◦ Facet Coupling

As the name suggests, this pattern will be based on coupling known *Facets* and will yield families of of facts containing subjects that are coupled on the specified facets. As a result, both divergences in the fact will be replaced with generalizations so the general form becomes:

$$F^{?}(r_n(r_{n-1}(...(r_1(r_0(?))))),\ q_m(q_{m-1}(...(q_1(q_0(?))))))$$

To emphasize the behavior within the context of the Cat & Cactus scenario, the pattern $F^{?}(Past(Sit(?), Cactus(?))$, will match any fact in the case which tells about any *subject* that *sat* in the *past* on any *cactus*.

2.9.5 ◦ Subject Coupling

The exact opposite of facet coupling is where we replace both divergences with facets instead of generalizations. In this case the generic form of the property becomes:

$$F^{?}(?(r_n(r_{n-1}(...(r_1(r_0(S_1)))))),\ ?(q_m(q_{m-1}(...(q_1(q_0(S_2)))))))$$

and will yield a collection of facts containing families of divergences that have the same subjects. In a more narrative way, this pattern shows all the various perspectives used to connect two sub perspectives of two specific subjects. In the same example as presented above, $F^{?}(?(Cat(ref)), ?(ref_1))$, we will obtain all the facts that connect *ref* seen from the perspective of *Cat* and ref_1, seen plainly.

2.9.6 ◦ Causation

Facts, have a directionality, which gives us a sense of causality, even though facts are not necessarily cause & effect constructions in the literal sense of the expression. It is however useful to employ the terminology in order to emphasize the nature of a collection of facts where all the arrow point towards a single subject, or perspective of it.

$$F^{?}(r_n(r_{n-1}(...(r_1(r_0(?))))),\ ?(q_m(q_{m-1}(...(q_1(q_0(S_2)))))))$$

A property construction with a generalization on the left and a facet on the right, as shown above, will yield a collection of facts formed by a family of subjects seen from a predefined perspective that converge towards a family of perspectives of a single well defined subject. Within the context of our cat and cactus example, the pattern $F^{?}(Past(Sit(?)), ?(Cactus(x)))$, will give us all the subjects that ever sat on the specific *Cactus(x)*, seen from any perspective.

2.9.7 ◦ Effecting

Effecting is the opposite property to causation. In this pattern, the roles are switched and the generalization is on the right and the facet is on the left:

$$F^{?}(?(r_n(r_{n-1}(...(r_1(r_0(S_1)))))),\ q_m(q_{m-1}(...(q_1(q_0(?))))))$$

creating a geometrical construction where the arrows point outwards from a family of perspectives of a given subject, that are sitting at the center, to a multitude of subjects seen from a well defined perspective. This gives the impression that by using the pattern one is observing the ramifications, the effects, of a subject.

To see a concrete exemplification, let us look again at the cat & cactus example. The construction $F^E(Sit(Cat(ref)),\ ?)$, will be able to yield all the things, cactus and whatever else, our specific Cat has ever sit or will ever sit on, that we know of.

2.9.8 ◦ The Infinity Pattern

The most generic of all patterns, which is least restrictive in terms of where we put the wild card, can generate a very large number of possible combinations, hence the the tentative term, *infinity pattern*. The parameters of an infinity fact are both *observations*, leading to a general form:

$$F^i(\,?(r_n(r_{n-1}(\ldots(r_1(r_0(?)))))),\ ?(q_m(q_{m-1}(\ldots(q_1(q_0(?)))))))$$

and will yield families of divergences connected to other families of divergences. These highly complex pattern matches can be used in analyses and statistical observations to determine ways information are connected. For example $F^i(\,?(Cat(?)),\ ?(Cactus(?)))$, can tell us the multitude of ways *cats* are connected to *cacti*. This in itself may not be very useful, but in a very large knowledge base, where many relevant[18] facts exist between *cats* and *cacti* the determination can be made that: sitting on cacti is painful to cats, even if this information is not clearly stated in the knowledge base.

2.10 ◦ Advanced Knowledge Operations

Basic knowledge operations are designed predominantly at investigating, querying, the knowledge base in order to discover information. The natural next step are the operations that allow us to augment the knowledge operations that add to the information already existing in the knowledge base.

Similarly to the basic operations, the advanced operations are also based on patterns.

2.10.1 ◦ Knowledge Evolving

We have spoken earlier about the imprecise characteristics of information. It has been shown that no matter how much we explore a subject there will still be room for more information to be extracted and that it is really at the latitude of each individual observer to extract only what is needed and to ignore the rest.

Unlike the traditional API based data storage systems, SPInDL does not suffer from structure lock in. Patterns can be used to evolve the entire knowledge base into a new form, once the need arises to store more details.

The principle behind it is to take a pattern, any of the previously enumerated and perform a transformation into another pattern. To take the most generic form:

$$?(q_m(q_{m-1}(\ldots(q_0(?)))) >> ?(r_n(r_{m-1}(\ldots(r_0(?))))$$

or properties:

18 Relevant in therms of the determination, the conclusion drawn.

$$F^{pq}(\,?(p_n(p_{n-1}(\ldots(p_0(\,?))))),\ ?(q_m(q_{m-1}(\ldots(q_0(\,?)))))) \gg$$
$$F^{rz}(\,?(r_n(r_{n-1}(\ldots(r_0(\,?))))),\ ?(z_m(z_{m-1}(\ldots(z_0(\,?))))))$$

To see a concrete example, let us consider that we know that at some point in time the cat, from sitting perspective, and the cactus are connected, *F(Sit(Cat(ref)), Cactus(ref$_c$))*, but the *time perspective* of is not observed in our knowledge base. Since we are not observing time, it is really irrelevant to mention *past, present, future*, these concepts do not yet exist in our knowledge base. The need however arises to create an orderly sequence in our information in order to be able to mention that the cat will never sit on a cactus again point at which time, past, present and future, become important.

We can now perform the evolution: ?*(?(Sit(?)), ?) >> ?(?(Past(Sit(?))), ?)*, which essentially means that whatever was sitting in our knowledge base, was doing so in the past. After the transformation which augmented our knowledge base with a new dimension, we are free to add the fact that the cat will not sit on the cactus in th future, *F^f(Negative(Future(Sit(Cat(ref)))), Cactus(ref$_c$)*.

2.10.2 ∘ The Axiom

Another knowledge extending operation is what is tentatively dubbed the *axiom*, when one or more properties are known to exist in tandem, given a fixed set of wild cards.

$$F^1(X^p(p^1{}_n(p^1{}_{n-1}(\ldots(p^1{}_0(X))))),\ Y^p(q^1{}_m(q^1{}_{m-1}(\ldots(q^1{}_0(Y)))))) \ \&$$
$$F^2(X^p(p^2{}_n(p^2{}_{n-1}(\ldots(p^2{}_0(X))))),\ Y^p(q^2{}_m(q^2{}_{m-1}(\ldots(q^2{}_0(Y)))))) \ \&$$
$$\ldots$$
$$F^{rz}(X^p(r_n(r_{n-1}(\ldots(r_1(r_0(X)))))),\ Y^p(z_m(z_{m-1}(\ldots(z_0(Y))))))$$

where X^p and Y^p, X, Y are wild card perspectives and subjects respectively with the particular characteristics that are the same across an axiom.

example: 49, Axiom

1. F^1(Past(sit(X)), Y) & F^2(Y(sit(X)), Pain) & F^3(Negative(future(sit(X)), Y)
2. F^1(Past(sit(cat(ref))), Cactus) & F^2(Cactus(sit(cat(ref))), Pain) & F^3(Negative(future(sit(cat(ref))), Cactus)

example: 50 point (1) states that, *if something, denoted by X, sat on something else, denoted by Y, and X through the perspective of Y is connected to pain, and also X not sit on Y in the future any more* are connected such that they exist in synergy.

We can clearly see that if we replace *X* by *Cat,* or *Cat(ref)* if we talk in specifics, and we replace *Y* with *Cactus*, than we obtain the scenario described in Illustration 5, the cat which is not sitting on a cactus in the future the way it did on the past, as pain was a part of it.

2.10.3 ∘ The Causal Axiom

An extends form of the axiom is the causal axiom in which some of the facts are considered to be a consequence of the other facts:

$$F^1(X^p(p^1{}_n(p^1{}_{n-1}(\ldots(p^1{}_0(X))))),\ Y^p(q^1{}_m(q^1{}_{m-1}(\ldots(q^1{}_0(Y)))))) \ \&$$
$$F^2(X^p(p^2{}_n(p^2{}_{n-1}(\ldots(p^2{}_0(X))))),\ Y^p(q^2{}_m(q^2{}_{m-1}(\ldots(q^2{}_0(Y)))))) \ \&$$
$$\ldots$$
$$F^w(X^p(p^w{}_n(p^w{}_{n-1}(\ldots(p^w{}_0(X))))),\ Y^p(q^w{}_m(q^w{}_{m-1}(\ldots(q^w{}_0(Y))))))$$

=>>

$$F^{rz}(X^p(r_n(r_{n-1}(...(r_1(r_0(X)))))),\ Y^p(z_m(z_{m-1}(...(z_0(Y))))))$$

...

We can use the (=>>) sign to separate the causing facts from the caused facts.

example: 50, Axiom

3. F^1(Past(sit(X)), Y) & F^2(Y(sit(X)), Pain) => F^3(Negative(future(sit(X)), Y)
4. F^1(Past(sit(cat(ref))), Cactus) & F^2(Cactus(sit(cat(ref))), Pain) => F^3(Negative(future(sit(cat(ref))), Cactus)

Deduction is a natural result of an axiomatic construction be that causal or non causal. If we observe all but one of the facts present in the axiom that would imply the existence of the latter one, whether this is clearly stated in the case or not.

2.10.4 ∘ Behavior

A less restrictive version of the axiom is the *behavior* operation, which unlike the axiom, is not taken for granted but has the potential to evolve out of the knowledge base.

Let us consider again the axiom at example: 50. If we replace the *X* and the *Y* with questions marks *(?)*, then the elements of the construction become simple properties:

$F^?$(Past(sit(?)), ?),
$F^?$(?(sit(?)), Pain),
$F^?$(Negative(future(sit(?)), ?)

If through a pattern recognition analyses, it is observed that a considerable proportion of (X, Y) pairs have at least two of these facts present, then we might be entitled to consider this a *behavior*. A behavior means that although the existence of the resulting axiom is not a certainty, it is a likely outcome.

2.11 ∘ Types

After our knowledge system has been liberated from the constraint of types, it is time to look at this extremely useful analytical tool within the context of this new pattern based approach. Although types cannot be used efficiently into storing information, for reasons presented earlier in the manuscript, it can be very useful when manipulating concepts and specifics to consider collections of perspectives and facts pertaining to these concepts and their specifics, rather than handle individual facts on their own. Information is subjective to each observer, but when observers observe the same concept repeatedly it is likely that they will be observing from more or less the same perspective every time. This represents the domain of interest of the observer with regards to the given concept, shortly its type.

2.11.1 ∘ The Type Pattern

The principle behind the type is, repeatability. This means that one expects a specific of a concept to have a certain collection of properties (fact patetrns) associated with it whether such facts are observed, and thus be part of the knowledge base, or not observed and thus be unknown.

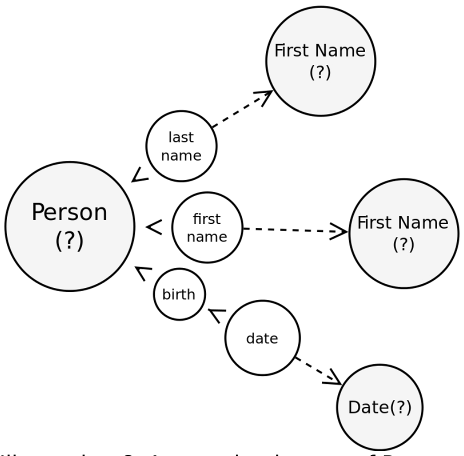


Illustration 6: A very simple type of Person

Illustration 6 depicts a subjective way of looking at *Person* in which the domain of interest are *first name*, *last name* and *birthday*. In this scenario, nothing else matters to the observer, but at the same time, the observer, will assume that these facts are characteristics of any specific *Person,* and as such any person in its universe is expected to have them.

example: 51, The facts considered in the simple type in Illustration 6

F^1(Date(Birth(Person(?))), Date(?))
F^2(First Name(Person(?)), First Name(?))
F^3(Last Name(Person(?)), Last Name(?))

These perspectives will change from observer to observer, but the principle remains: in one of the chains of perspective in every property that is part of the *Type*, there can be found the concept a type of which the *Type* is.

definition: 10, Type

A type, T_O^x, that represents the subjective view of observer O with respect to concept X, is a collection of properties having the form: O:X = {F^i(?(X(?)), ?) or F^j(?, ?(X(?))), i = 1...n, j = 1...m}

It is important to reiterate that types in SPInDL do not determine the knowledge base, they are only a tool to manipulate the knowledge base. Illustration 6 depicts *a type of Person* with minimalistic information observed, but this does not stop any specific person in the knowledge base from having any number of other facts that connect to or from it. As a matter fact, there could be many other types of Person with different sets of properties.

While types could be predefined based on initial need, they can also be a product of pattern recognition as the knowledge base grows. We could consider a type as being the collection of properties matching the form in definition: 10 for a given concept, such that at least a threshold of specifics of that concept are found to match all the patterns in the type. In such a setup, types become a constantly evolving tool that depict the particular way a certain observer is capturing reality at a particular point in time.

2.11.2 ◦ The Name Space

From the definition of the type it is noticeable that the notation for it, O:X, resembles remarkably the notation of the types in classical ontologies, where *O* stands for the name space and *X* stands for the type name. While in both cases the name space role is to attach the type a proprietary aspect, the similarities end there.

In the classical ontologies the name space's role is to ensure uniqueness in the denomination of the type, and thus avoid *name clashes*. If the name space is unique, which it must be in the world of ontology, then whatever comes after the colon the combination of the two remains unique, even if the types themselves happen to be the same: *abc:Person*, *bcd:Person,* there is no clash between the two full names. However, an exact match between the type names (whatever comes after the colon) is only a product of coincidence: *Person* in *abc* has nothing in common with *Person* in *bcd*. They could both represent the concept of *Person*, or only one of them could do so, or the case may be that none of them actually stand for the *Person* concept. As shown in more detail in 1.6 Fractured Realities, the two define two different realities.

By contrast SPInDL takes the exact opposite approach: if the local part of the type name happens to be the same, then by definition, the two types represent the same concept, they just describe different sets of properties about the concept. *abc:Person* and *bcd:Person* are two types of the same concept, *Person*, each in the subjective view of name space owner *abc* and *bcd* respectively. With this approach, SPInDL allows every subjective individual or organization to express their particular need to observe a concept without destroying the common reality in which all organizations need to operate.

2.11.3 ◦ Information Transfer

In computer operated applications, a specific, an instance of a type, such as a specific *Person*, it is considered almost exclusively as being the sum of its parts. The values associated with the instance define the instance, some of them may be unique, like personal id numbers, social security numbers, passport numbers, fingerprint data, etc, others are not unique such as first name, last name, and so on. These unique attributes of a person define the person, pinpoint the person in a database. By contrast, we humans don't rely on the uniqueness of any characteristics of a specific, our minds don't process uniqueness in the manner computers do. To us the concept of uniqueness doesn't really exist. We live in an open world where everything that is *unique* needs only be *unique enough* and it is only so until it proven otherwise. At that point another attribute is appended that makes the specific unique (if possible) and so on. We live in a very dynamic world where things do not have mathematical rigor.

Even such, when we refer to an object, a person for instance, a mother, a father, we know it in our minds very, very clearly who we refer to. In our minds there is no doubt about the uniqueness of our reference. The multitude of facts we know about that reference are associated with that reference but they don't define it. This uniqueness however cannot be communicated. It is impossible for me to transmit the reference I have in my brain with regards to my mother for example to a third person, so I do the next best thing, and I use characteristics that are sufficiently unique to both of us (me and the person I am communicating to), to establish common reference. These characteristics would be a subset of the aforementioned facts that I know about the object intersected with the facts my interlocutor knows about the object, because whatever I transmit will pinpoint a reference in his brain that I expect uniquely refers the same thing my reference does.

If we are to generalize this process of establishing common reference, we have to consider the fact that each person, or groups of persons, see objects within their own particular domain of interest. As such, they will know different facts about objects, facts that are relevant to them. Individuals or groups develop stereotypes that capture various perspectives of concept specifics, stereotypes that consist of patterns of facts known to exist with regards to concepts.

This kind of dynamics that comes into view whenever we talk about human communication is essential to human reality because it accounts for the impreciseness and subjectivity of information. If this would not be accounted for some part of the communication will suffer: either the some of the parties in the communication would not be allowed to have custom aspects, or common reference could not be established.

The SPInDL model for communication is an attempt to mimic this openness of language by relying on the established common reality, type patterns and the operations that are possible to do with these patterns, or more precisely sets of patterns.

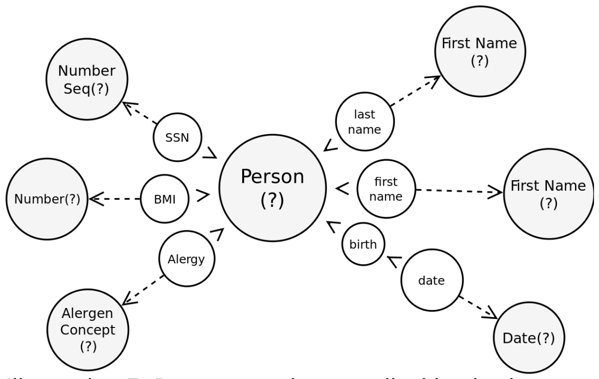


Illustration 7: Person seen by a medical institution

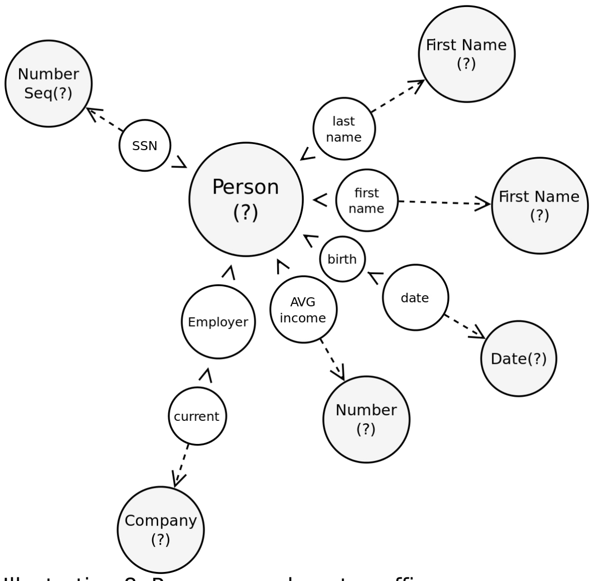


Illustration 8: Person seen by a tax office

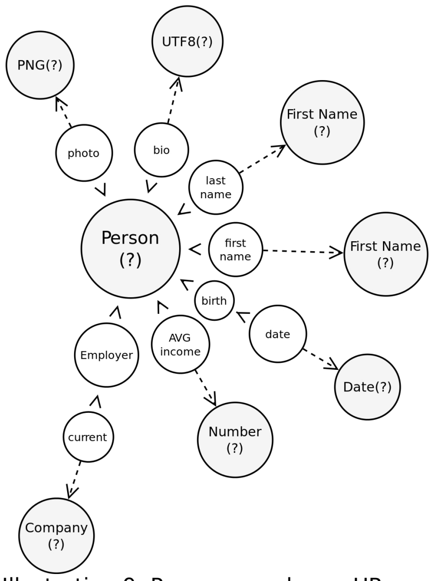


Illustration 9: Person seen by an HR company

Illustration 7, 8 and 9 depict graphically the way different institutions observe the concept of *Person*. It can be seen that true to their domain of interest with regards to the persons, some of the perspectives observed are the same, while others differ. A financial institution simply has no interest in the allergies a person might have, while a clinical institution would similarly have no interest in the CV or bio of a person. It ca be expected though, if two institutions or people for that matter want to communication about a concept, that they would also observe similar facts too, such as first name, last name, date of birth, etc. If such common perspectives do not exits, it is hard to envision that there would be any relevant information one could transmit to the other, like the case is with the stock market proposition between the Pirahã and the English Speaking Broker in chapter 1.3 (Common Meaning In Human Communication).

While the fact that these institutions would see the concept of *Person* differently, is no strange concept to either API based applications or ontologies, what they lack is common grounds to establish the fact that although their view is different, the concept in discussion is common. Because of this communication is by no means out of the box, but rather a tedious procedure the result of which is a common exchange layer, procedure that needs to be repeated with every communicating partner.

In the concept centric approach of SPInDL, there is no ambiguity about the concept in discussion, therefore any application that implements SPInDL can expect concept terminologies to point to the same thing, in our example *Persons*. Therefore when the need for communication arises the only thing that needs to be established is the set of common perspectives. The party that does not observe / handle the allergy perspective for instance would have no operations defined to work with that perspective. In the best case it could have generic operations such as storing, displaying, but the relevance of the perspective ends there. There is *no information* in that perspective to this party. The essence of the transferable information comes from the common perspectives: these perspectives are information to the receiver and are available from the provider.

example: 52, Types of Person

```
Tax:Person{
            F(First Name(Person(?)), First Name(?)),
            F(Last Name(Person(?))), Last Name(?)),
            F(date(birth(Person(?))), date(?)),
            F(SSN(Person(?))), Number Sequence(?)),
            F(AVG Income(Person(?))), Number(?)),
            F(Current(Employer(Person(?)))), Company(?)),
            F(bio(Person(?))), UTF8(?)),
}
Medical:Person{
            F(First Name(Person(?)), First Name(?)),
            F(Last Name(Person(?))), Last Name(?)),
            F(date(brith(Person(?))), date(?)),
            F(SSN(Person(?))), Number Sequence(?)),
            F(BMI(Person(?))), Number(?)),
            F(Allergy(Person(?))), Allergen(?)),
}
HR:Person{
            F(First Name(Person(?)), First Name(?)),
            F(Last Name(Person(?))), Last Name(?)),
            F(date(birth(Person(?))), date(?)),
            F(AVG Income(Person(?))), Number(?)),
            F(Current(Employer(Person(?)))), Company(?)),
            F(bio(Person(?))), UTF8(?)),
            F(Photo(Person(?))), PNG(?))
}
```

example: 52 shows the various kinds of Persons that exist in our hypothetical community each defining its own type to handle the concept of Person. In this setup, the information exchange pattern, they Common Type can be established in an ad-hoc manner, there is no need for complicated API correlation between two different organizations. All there is to it is to match the types of each party and establish their intersection to obtain the set of common pattern facts observed by each them with regards to a Concept.

example: 53, HR & Tax:Person. The common type for HR and Tax perspectives

```
HR & Tax:Person{
            F(First Name(Person(?)), First Name(?)),
            F(Last Name(Person(?))), Last Name(?)),
            F(date(birth(Person(?))), date(?)),
            F(AVG Income(Person(?))), Number(?)),
            F(Current(Employer(Person(?)))), Company(?)),
            F(Last Name(Person(?))), Last Name(?)),
            F(bio(Person(?))), UTF8(?)),
}
```

example: 54, Tax & Medical:Person. The common type for HR and Tax perspectives

```
Tax & Medical:Person{
            F(First Name(Person(?)), First Name(?)),
            F(Last Name(Person(?))), Last Name(?)),
            F(date(birth(Person(?))), date(?)),
            F(SSN(Person(?))), Number Sequence(?)),
}
```

example: 53 and example: 54 show two such common types that can be used for information exchange between an HR and a Tax institution and between a Tax and Medical institution respectively.

2.12 ◦ Openness, Community Driven Model

An interesting byproduct of this approach is, the *Blanket Type,* example: 55, representing an exhaustive set of all properties that are observed by any member of a community. If the common type is the intersection of the property sets, the blanket type would be the union of properties observed by all parties.

example: 55, Blanket Type: all perspectives captured by the community

```
HR:Person{
        F(First Name(Person(?)), First Name(?)),
        F(Last Name(Person(?))), Last Name(?)),
        F(date(birth(Person(?))), date(?)),
        F(SSN(Person(?))), Number Sequence(?)),
        F(BMI(Person(?))), Number(?)),
        F(Allergy(Person(?))), Allergen(?)),
        F(AVG Income(Person(?))), Number(?)),
        F(Current(Employer(Person(?)))), Company(?)),
        F(Last Name(Person(?))), Last Name(?)),
        F(bio(Person(?))), UTF8(?)),
        F(Photo(Person(?))), PNG(?))
}
```

Such blanket types, would be an excellent indicator of community trends and could serve as implementation reference to any newcomer into the community.

Looking at this globally and on the long term from cooperation / communication perspective, the world of types, this *Common Reality*, will become a dynamic, evolving place, where based on the frequency of occurrence certain properties can become the norm or on the contrary can become special situations. This process however, should not impact the community, because properties do not define concepts.

Due to the large amount of knowledge that needs to be modeled in order to aid various business needs the common reality would seem to grow impossibly large to maintain with time. But this picture stems from the perception of today's standards, where they are created and maintained in an institutionalized, top to bottom, manner.

In this common reality driven model, standards are not created, they emerge. By fixing the concept, which is already standardized in human reality, and creating a cooperation based common patterns, participants will determine the nature of each type as best needed. As the system is used in common, patterns will emerge for various industries and newcomers can adapt to these patterns to better aid communication. The elasticity of the system however does not enforce them to use the pattern as defined up to that point, and if they have special needs they can freely implement them, potentially benefiting the industry with new and improved types.

As a property is used more and more frequently (for ex. exceeding 85% of all implementations) the property can become standard and thus types evolve. Future implementers can make educated decisions of how to best adapt to the domain(s) they belong to in order to facilitate communication within their domain. If we presume the participants interest is to communicate, it is reasonable to assume that more often than not, they will make compromises to adapt. But the freedom of not having to fully comply with the norm yet still be able to communicate is essential to maintain participants implication.

3 ∘ Conclusion

On a global scale, with thousands or millions of participants in the community the elimination of the need for developing a common API between every two parties represents not only a massive reduction in complexity, effort and cost but a complete paradigm shift of how communication happens. The complexity of establishing common ground between structure based systems raises exponentially with the addition of new participants in the discussion by either requiring the implementation of exponentially more common APIs or by enforcing an ever more rigid common API that grows increasingly incompatible with community needs and change. There is no compromise, either approaches crash after just a iterations, and the result is an Internet of impossible to reconcile standards. The approach is simply contrary to evolution, contrary to human nature.

By contrast, natural language evolves it is not reinvented, it is not pre-created by a higher forum and not maintained by any organization. It is a grass roots effect. The crowd builds it and with every new participant language becomes richer, more complete, without becoming more difficult to master. Language can just as easily communicate the complex information of our days as it could the works of Shakespeare, the teaching of ancient Greeks and probably those of cave men. And it simply comes natural to expect it to continue conveying whatever complexities our future reality will hold.

The Semantic Perspective Information Definition Language is an attempt to replicate this elasticity and versatility in absence of which there will be no global scale semantic web. Whether SPInDL will prove to live up to the enormous power of natural language remains to be determined, not by one organization, but by the crowd. In any case it is a goal that we must strive for and change is a paradigm shift that we must accept.

Nothing in nature is meant to survive if it does not embrace evolution. Nature is Evolution. Our world evolves, we evolve, everything that is us and around us changes with time. If we are to cope with this change, we either create and adopt new tools all the time, or we create one tool that can change together with us. The choice I think, is obvious.

4 ∘ References

The Principles of a Semantically Rich Data Representation System: Stefan Harsan Farr, The Principles of a Semantically Rich Data Representation System, 2013

Chenho Kung, 1991: Chenho Kung, The Object-Oriented Paradigm, 1991

Walsh Norman, 1997: Walsh Norman, A Guide to XML, 1997

Dan Brickley, Libby Miller, 2010: Dan Brickley, Libby Miller, FOAF Vocabulary Specification 0.98, 2010, http://xmlns.com/foaf/spec

An Introduction to the Syntax and Content of Cyci: Cynthia Matuszek, John Cabral, Michael Witbrock, John DeOliveira, An Introduction to the Syntax and Content of Cyc,

ResearchCyc: Cycorp Inc., ResearchCyc, Development Platform, 2013, http://www.cyc.com/platform/researchcyc

Upper Mapping and Binding Exchange Layer (UMBEL) Specification: Michael Bergman, Frédérick Giasson, Upper Mapping and Binding Exchange Layer (UMBEL) Specification, 2013

SPATIAL COGNITION AND COMPUTATION: Pierre Grenon, Barry Smith, SPATIAL COGNITION AND COMPUTATION, 2004

Ontology for the Twenty First Century: An Introduction with Recommendations: Andrew D. Spear, Ontology for the Twenty First Century: An Introduction with Recommendations, 2006

Basic Formal Ontology Users: INFOMIS, Basic Formal Ontology Users, 2013, http://www.ifomis.org/bfo/users

UMBEL Projects: , UMBEL Projects, Visited - 2013, http://umbel.org/community/projects